

UC Berkeley

Research Reports

Title

Demonstration of Automated Heavy-Duty Vehicles

Permalink

<https://escholarship.org/uc/item/6kc7m4jr>

Authors

Shladover, Steve E.

Lu, Xiao-Yun

Song, Bongsob

et al.

Publication Date

2006-06-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Demonstration of Automated Heavy-Duty Vehicles

**Steven E. Shladover, Xiao-Yun Lu, Bongsob Song,
Susan Dickey, Christopher Nowakowski, Adam Howell,
Fanping Bu, David Marco, Han-Shue Tan, David Nelson**

**California PATH Research Report
UCB-ITS-PRR-2005-23**

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

Final Report for Task Order 4228 and 4229

June 2005

ISSN 1055-1425

Demonstration of Automated Heavy-Duty Vehicles

Final Report

Task Orders 4228 and 4229

**Steven E. Shladover, Xiao-Yun Lu, Bongsob Song, Susan Dickey,
Christopher Nowakowski, Adam Howell, Fanping Bu, David Marco,
Han-Shue Tan, David Nelson**

June 2005

Acknowledgments

This work was performed by the California PATH Program at the University of California at Berkeley in cooperation with the State of California Business, Transportation and Housing Agency, Department of Transportation. The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California.

This report describes the results of a large project that benefited from the work of many participants. The authors are a subset of the full team that worked under the leadership of Dr. Ching-Yao Chan to make this project succeed. In addition to the authors and Dr. Chan, other members of the project team at PATH included Anouck Girard, Daniel Schickele, Benedicte Bougler, Thang Lian, Paul Kretz, Joanne Chang, Scott Johnston, Daniel Empey, Elvin Vader, Bill Stone, Jay Sullivan and Bart Duncil. Vital support in purchasing equipment and supplies was provided by Barbara Cooper.

The support of Lynn Barton and his colleagues at Caltrans District 11 was essential to the success of the on-site testing in San Diego, and we also appreciate the help of Dave Schumacher and Bill Miller of San Diego Transit with local arrangements and support for the experimental buses. Caltrans' Division of Research and Innovation provided not only financial support, but also important direct participation with the work, including the efforts of Gurprit Hansra, Pete Zaniewski, Hassan Aboukhadijeh, Bob Battersby, Jose Perez, Joel Retanan, Asfand Siddiqui and Randy Woolley, with the support and supervision of Greg Larson.

Table of Contents

1. Introduction
 - 1.1 Project Background and History
 - 1.2 Goals and Performance/Functional Requirements
 - 1.3 Benefits of Transit Bus Automation
 - 1.4 Benefits of Truck Automation
 - 1.5 Overview of Report
2. Host Vehicle Platforms
 - 2.1 Transit Buses
 - 2.2 Freightliner Century Trucks
3. Modifications to the Host Vehicle Platforms
 - 3.1 Overview Description
 - 3.2 Modifications to Buses, for Automation and Data Acquisition
 - 3.3 Modifications to Freightliner Trucks for Automation and Data Acquisition
 - 3.4 Coordination Layer Control Implementation
4. Performance of Automated Heavy Vehicle Systems
 - 4.1 Precision Docking of Buses
 - 4.2 Bus Lateral Control at Highway Speed
 - 4.3 Bus Longitudinal Control
 - 4.4 Truck Longitudinal Control
 - 4.5 Truck Lateral Control at Highway Speed

Appendices

- A. In-Vehicle PC/104 Computer Systems for Trucks and Transit Buses
- B. Example Truck Speed Tracking Test Data
- C. Example Experimental Results of Two-Truck Platoon Tests
- D. In-Vehicle Network Use on AVCSS Demo Heavy-Duty Vehicles
- E. Real-Time Token Ring (RTTR) for QNX 4 Reference Manual 1.0
- F. WTRP for QNX Reference Manual 1.0
- G. QNX4 driver for Orinoco wireless Reference Manual 1.0

Chapter 1. Introduction

1.1 Project Background and History

This project was created in order to continue progress toward a future in which vehicle automation technologies are able to improve transportation operations. In the wake of the termination of the National Automated Highway Systems Consortium (NAHSC) program in 1998, the California Department of Transportation (Caltrans) created The Phoenix Project to bring together the organizations that remained interested in this future vision. The discussions within The Phoenix Project focused on the opportunities that could be gained from earlier deployment of automation technologies on transit buses and heavy trucks, as compared to passenger cars. PATH was already doing intensive research and testing on automation of one heavy truck, which provided a starting point for further work.

Caltrans recognized the value of public demonstrations in building interest and support for additional research and development work and for educating stakeholders about the benefits of deploying the technologies. Consequently, they decided to create “Demo 2002” as a high-profile public demonstration of bus and truck automation technologies, scheduling it for the summer of 2002. This was planned to include three transit buses and three Class-8 tractor-trailer rigs, driving a variety of realistic scenarios on the I-15 HOV lanes in San Diego. In order to facilitate the implementation of the latest technologies on the test vehicles, it was recognized that it was important to start with new production vehicles, equipped to the current state of the art.

The test trucks were acquired within the planned schedule, but the 18-month lead time for delivery of new transit buses made it impossible to meet the summer 2002 demonstration milestone. Consequently, the project schedule was extended to accommodate the demonstration in the summer of 2003 and it was renamed “Demo 2003”. When the California state budget crisis deepened in late 2002, Caltrans decided that it would not be politically advisable to have a high-profile public demonstration, but that the technology development and testing work of the project should still be completed as planned. Because transit buses are politically “safer” than trucks, they agreed that the testing of the buses could be witnessed by a limited group of visitors from within the ITS and transit industries, so visitors from the ITS America Board of Directors, the TRB Committee on Vehicle-Highway Automation, and the attendees of the TRB Workshop on Automated Bus Rapid Transit were able to witness the bus tests on I-15 during the weekend of August 23-24, 2003. That was the last weekend that the I-15 HOV lanes were available for testing, because from that time forward the lanes were open to the public on weekends as well as weekdays.

1.2 Goals and Performance/Functional Requirements

The performance and functional requirements for the vehicle technologies were derived based on the goals of the project. These goals were:

- a) Showing how automation technologies could produce benefits to the operation of transit buses and trucks;
- b) Showing multiple steps of a deployment sequence from present-day manual driving to future automated driving, not just the “end state”;
- c) Helping stakeholders recognize the user-friendliness of the automation technologies, in terms of both driver and traveler comfort and convenience;
- d) Demonstrating functional capabilities that could not be shown in Demo '97, to indicate advances since then in areas such as:
 - a. Easy, natural transitions between manual and automated driving
 - b. User interface with realistic driver controls and display
 - c. New functions such as precision docking of buses
 - d. Coordination of diverse vehicles within a tightly-coupled platoon.

Based on Goal (a), it was necessary to achieve accurate lane-tracking and close coupling of vehicles within a platoon, as well as flexibility in forming and separating platoons:

- Lane-tracking accuracy with lateral errors of no more than +/-15 cm makes it possible for a full-width bus to operate within a bus lane that is only 10 ft. (3 m) wide rather than the standard width of 12 ft. (3.6 m), providing opportunities for considerable savings in construction and right-of-way costs, as well as the flexibility to fit bus lanes within space-constrained locations that would otherwise be impractical.
- Bus platoons with separations between buses of 15 m make it possible for a bus lane accommodating platoons of up to three buses such as those used in this project (two 40-ft buses and one 60-ft articulated bus) to provide a capacity of more than 70,000 seats per hour, which means that a bus lane could be fully competitive with heavy rail transit for the highest-demand urban corridors.
- Truck platoons with separations between trucks of a half truck-length or less make it possible to reduce aerodynamic drag by enough to produce significant savings in fuel consumption and pollutant emissions.
- Automatic lane changing and platoon join and separate maneuvers make it possible for platoons to be reconfigured easily to accommodate different origins and destinations for the trucks or buses in the platoons, so that their operations do not need to be constrained by waiting for other vehicles with the same destination.

Based on Goal (b), it was necessary to include partial automation as well as full automation, and to address automation for specialized near-term applications:

- Automatic steering control could be done while the vehicle is operating under conventional cruise control or following a pre-determined speed profile, or while the driver is controlling speed.
- Precision docking of a bus at low speed, for application at bus stops or terminals, shows an application of steering automation that could be implemented relatively quickly, with a minimum of infrastructure modification, and with the driver retaining responsibility for speed control and obstacle detection.

Based on Goal (c), it was necessary to develop an attractive and user-friendly driver interface, as well as ways of showing system capabilities to the vehicle passengers:

- Very smooth ride quality, in both lateral and longitudinal motions, had to be designed into all the vehicle control systems. This was a dominant consideration relative to accuracy because passengers are more likely to develop a negative or uncomfortable impression from a rough or jerky ride than from a somewhat less accurate performance (even though the latter would reduce the directly-quantifiable benefits from the lateral and longitudinal control systems).
- Smooth transitions between automatic and manual driving, so that both drivers and passengers can perceive the ease of making these transitions and not be intimidated by the new mode of automated driving.
- Driver-vehicle interface with well-designed display and simple controls for the driver to use,
- Graphical display for passengers to be able to observe aspects of vehicle performance (such as driver interface features, steering and lane changing accuracy, and progress of longitudinal maneuvers) that they might not be able to perceive directly during a ride.

Most of the new functions associated with Goal (d) have already been addressed in satisfying the other goals. However, the coordination of diverse vehicles within a tightly-coupled platoon is an additional functional capability that is not necessarily directly perceived as significant by passengers on a demonstration ride, but requires considerable technical efforts to achieve. This also represents a significant advance beyond what was shown in Demo '97.

1.3 Benefits of Transit Bus Automation

Transit bus operations are particularly amenable to automated driving for a variety of reasons, as previously described in [1]. Productivity enhancement benefits can be a strong economic incentive to use of the technologies, and since transit agencies in some cases have responsibility for both vehicle and infrastructure (guideway), the vehicle-infrastructure coordination issues can be simpler than they would be for other types of vehicle operations. Moreover, various transit agencies already have experience with operation of automated and even driverless vehicles, so the cultural issues with automation are more manageable than they are with other transportation modes.

The reasons for applying automation to transit buses were set forth in one of the handout documents provided to the visitors who witnessed the testing in August 2003, the contents of which are inserted here, with modest updates:

Automated Bus Rapid Transit (ABRT)

Automated Bus Rapid Transit (ABRT) may be viewed as the integration of several elements:

- Precursor systems such as transit information and operational improvements to enhance fleet management and provide more accurate and timely information to passengers, and collision warning systems to improve bus safety
- Bus Rapid Transit (BRT) service innovations in fare collection procedures, station design and location, and more attractive vehicle designs
- Automatic precision docking to enable buses to stop immediately adjacent to a loading platform, providing passengers with quick and easy boarding and alighting, even for those whose mobility is impaired
- Automated bus operation on segregated busways, providing rail-like ride quality while minimizing the needed right-of-way width.

How will it function and with what technologies?

Bus transit automation will consist of automation functions and complementary elements (advanced public transportation systems) and include certain design attributes.

Automation functions:

- Precision docking
- Lane keeping
- Automated speed and spacing control
- Maintenance yard operations

Complementary elements:

- Collision warning (forward, side, and rear)
- Vehicle diagnostics warnings
- Transit management center operations
 - Trip information for travelers at stops, stations and on-board buses
 - Electronic fare payment and pre-pay systems
 - Passenger counting systems
 - Traffic signal priority systems

Supporting design attributes:

- Bus design changes (low floor, wide doors)
- Bus stop/station design
- Infrastructure (bus bulbs, queue-jumpers, dedicated lane, check-in/check-out system, vehicle-roadside - communications)
- Bus components (electronic throttle, brake, and steering control)

Why demonstrate automated buses now?

The *earliest* deployments of automation technologies in road vehicles will likely be on heavy vehicles—*buses* and *trucks*—operating on their own special rights-of-way because:

- It's easier to develop and acquire rights-of-way for public purposes like transit service
- In some cases, buses already operate on separate facilities, which could, if demand warranted, be switched over to automation
- Costs of the technologies are a smaller percentage of total bus costs and buses are used much more intensively so these costs are amortized faster
- Benefits in travel-time reduction, trip reliability and safety can be translated more directly into cost savings and revenue increases than for private passenger cars
- Customized, small-lot production of vehicles makes it possible to introduce automation technologies into the bus production process faster than for automotive mass production
- Packaging of new technological elements is easier on buses than on passenger cars
- Buses already have more onboard electronic infrastructure (such as data buses and electronic engine controls) to use as a foundation for more advanced capabilities than passenger cars

- Maturing technologies can be used more safely by professionally trained bus drivers on professionally maintained buses than by the general public on passenger cars that may not be well maintained.

Who benefits and how?

- Transit properties
- Bus drivers (employees)
- Bus passengers (general public)
- Infrastructure owners and operators

The overall benefit will be an improved level of people movement, mobility, and quality of service. More specifically:

Transit Properties

- Reduced dwell time at bus stops for loading and unloading of passengers and avoiding need for maintenance-intensive wheelchair ramps by use of precision docking with low floor buses
- Precision docking at bus stops should help reduce tire scuffing against the curb, resulting in reduced wear-and-tear on the bus's tires and corresponding maintenance costs
- Rail-like line-haul service at a much lower capital cost
- Narrower rights-of-way and structures for busways are possible with the use of automated steering/lane keeping
- Potentially reduced operating costs (labor, fuel and vehicle productivity) for the automated portion of the bus trip (line-haul)
- Facilitating maintenance operations and saving yard space and labor, (i.e., reducing costs), ordinarily used to move buses through routine maintenance processes
- Greater vehicle and passenger lane-capacity, by enabling buses to operate at shorter headway than under manual control
- Reduced fuel consumption and emissions for buses that can operate in automated platoons with small enough separations that aerodynamic drag can be reduced
- Potential safety improvements should have direct benefits in terms of reduced insurance costs and less time lost from buses taken out of revenue service while being repaired for crash damage
- Increased ridership is a collateral benefit that could result from the cumulative effect of the previously stated benefits

Bus Drivers

- Automating precision docking at bus stops will make this operation easier and less stressful
- Automated line-haul operations reduce workload and stress

Bus Passengers

- Reduced travel time can result from having automated bus travel on a dedicated lane
- Improved bus stop arrival reliability
- Enhanced access to and from buses for mobility impaired passengers and reduced time for -loading and unloading of all passengers from bus stop - precision docking
- Smoother travel for passengers and increased passenger riding comfort
- Flexibility to perform in "dual mode": collection and distribution, and line-haul portion (even with intermediate stops). This may reduce need for passengers to make transfers, allowing passengers to remain on same vehicle for entire "door-to-door" service. The bus would "transfer" between a local neighborhood collector/distributor and a line-haul rail-like mode of operation.

Infrastructure Owners and Operators

- Permitting operations on narrower rights-of-way (reduced lane widths), thereby saving on land use and physical infrastructure costs
- Accommodating bus-only lanes in locations where they would otherwise not be able to fit at all

What are some of the challenges facing successful deployment?

Solution strategies are currently being sought for these and other challenges.

The transit industry may have concerns over the complexity and reliability of new technologies associated with bus transit automation. These concerns may influence the acceptability of technological changes that could potentially impact:

- Role of the driver (driver training, salaries, and work rules, additional driver responsibilities)
- Maintenance cost and complexity
- Safety
- Liability (changing risk and responsibility assignment).

Large-scale public transportation projects have the potential for influencing travel patterns and surrounding land uses. Automated bus transit operations, intended to replicate high-level transit service, (e.g., rail transit), though more flexible, may raise concerns over:

- How they fit into a region's overall transportation plans to be considered as a credible alternative for regional public transportation investment
- How their inherent flexibility over rail transit may be perceived instead as a lack of permanence and inhibit potential developers from investing heavily along such transit corridors.

Will bus transit automation cost more?

There will be *incremental* costs associated with the use of bus automation technologies: additional electronic equipment installed and maintained on buses and busways and some additional protection of the busways to prevent intrusion by pedestrians, animals, unauthorized vehicles and debris.

Additional bus equipment is likely to consist of:

- Electronically controlled brake actuator (but this may soon become standard equipment associated with anti-lock braking anyway)
- Electronically controlled steering actuator
- Lateral-position sensing system
- Forward ranging sensor system similar to -commercially available sensors for adaptive cruise control systems
- Vehicle-to-roadside and vehicle-to-vehicle data communications systems, which could build on existing and impending traffic signal priority system and DSRC technologies
- Collision warning sensor systems, which are already available for some vehicle applications and are under commercial development for others
- Onboard control computer similar in power to standard personal computers

At early stages of development, the total costs of these systems are likely to add no more than 5% to 10% to the cost of a new transit bus, and over time those Incremental costs should decrease by another factor of 4 or 5. It is important to consider these additional cost elements in the context of the costs of providing higher quality transit services by other means, such as light or heavy rail transit systems. In the context of such costs, these incremental costs are negligible.

How do we get from today's bus transit system to automation?

The specific development and deployment sequence will likely vary by location, depending on local and regional needs and constraints, so some locations will undoubtedly be able to make faster progress than - others. However, a generic sequence can be defined, -building on technologies already available or nearly available and then combining additional technologies and service elements in building-block fashion to achieve increasing levels of capability.

There are existing and currently emerging technologies commercially available today in at least one major sector of the world (Europe, North America, or Asia/ Pacific) for at least some vehicle classes or for specialized applications. They are:

- Forward collision warning
- Lane-departure warning
- Adaptive cruise control (ACC)
- Vehicle-roadside communication

Transit forward collision warning is being developed now by PATH for the San Mateo County (California) Transit District (SamTrans) under the sponsorship of the Transit Intelligent Vehicle Initiative.

Special capabilities can be added to transit buses with moderate levels of development and deployment costs:

- Vehicle-vehicle data communication for cooperative adaptive cruise control (CACC)
- Low-speed precision docking of buses at stops
- Automation of bus movements through maintenance facilities

A next level of deployment includes protected-lane opportunities, particularly useful in locations where there are strong needs for enhanced transit services and/or where the right-of-way can be made available. In these locations, the operating agency can set aside a separated, protected lane for transit use. It then becomes possible to implement automatic steering control safely, permitting use of a narrower lane and relieving the bus driver of the steering responsibility.

Integrating a combination of these elements can form the basis of an initial operational scenario for bus transit automation, such as a pure line-haul run with few intermediate stops. Passengers could be collected from their origin locations at normal local bus stops, where the bus would be driven manually (except for the assistance of precision docking at stops). At the entrance to the protected busway or bus lane, the driver would switch the bus to automated operation and it would continue to operate automatically until it reached the busway's destination end. There could be intermediate stops along the automated busway, where the bus would operate exactly the way automated metros or automated guideway transit systems do now. At the end of the automated busway, the driver would resume manual control of the bus and could take the passengers to their desired local bus stops. Through this kind of "dual mode" operation, the automated bus provides the collection and distribution flexibility of conventional buses and the line-haul efficiency and service speed of conventional rail transit, while saving passengers the inconvenience and time associated with transfers. This is the great service advantage automated buses can provide.

Over the longer term, with further advancements:

- Access to the ABRT lane could be provided to suitably equipped vanpools and then carpools
- Buses or vans could be coupled together more closely with their counterparts to form platoons, increasing capacity while reducing drag, to save fuel and reduced emissions
- Entry maneuvers could be automated with the addition of more sophisticated vehicle-roadside communication
- Higher-level management functions could be implemented to serve a network of connected ABRT lanes. These could indeed become the precursors to an automated highway network.

Functions to Observe In Testing

- Precision docking, in configurations suitable for terminal or station (in-line) or at conventional arterial bus stops (full lane change)
 - Very small gap between platform and bus entrance
 - Accessibility for mobility-impaired passengers
 - Reduced stress for driver
 - Reduced likelihood of scuffing tires at curb

- Smooth and easy transitions between normal manual driving and automation functions
 - Simple user interface
 - Easy resumption of driver control when needed
 - Smooth control transitions

- Lane-keeping assistance
 - Highly-accurate lane tracking in tightly constrained locations
 - Smooth steering response
 - Reduced driver stress and workload
 - Rail-like service quality

- Automatic vehicle-following control
 - cooperative adaptive cruise control to reduce driver workload
 - close-formation vehicle following for increased capacity in high-density locations (an electronically-coupled bus train)
 - smooth acceleration and deceleration for passenger comfort and reducing energy consumption and emissions

1.4 Benefits of Truck Automation

The benefits of truck automation are somewhat different from those for transit bus automation, based largely on the differences between the operating characteristics and normal uses of buses and trucks. The opportunities that can be gained from automating the driving of trucks were described in [2], and those were summarized in background material prepared for the planned (but subsequently cancelled) demonstration of the automated trucks:

Why Consider Dedicated Truck Lanes?

Heavy trucks and passenger cars are so different from each other that it is in some ways remarkable that they generally share the same highway lanes almost everywhere today. Although this sharing of lanes simplifies the development and operation of the roadway infrastructure, it also has some significant adverse safety and economic implications. The relevant differences between trucks and cars include:

- Factor of up to 40 in mass
- Factor of up to 4 in length
- Factor of 4 to 20 in acceleration capability
- Factor of 2 to 3 in braking capability
- Factor of up to 2 in width

If trucks and cars were separated from each other on sections of highway that have heavy volumes of truck traffic it would be possible to separately optimize the design and operations of these costly infrastructure systems. This could offer a variety of significant benefits:

- Eliminating the truck-car conflicts and incompatibilities that are responsible for the majority of crashes involving trucks;
- Enabling trucks to avoid the traffic congestion caused by large volumes of automobile traffic;

- Enabling lane widths to be optimized for the different vehicle widths, so that the trucks could retain full-width lanes, but the cars could use narrower lanes, making it possible to fit more lanes within the same highway structures and rights of way;
- Enabling pavement design and maintenance to be optimized for the different loadings imposed by trucks and cars, so that costs could be significantly reduced for the lanes used only by the light-duty vehicles;
- Reducing the driving stress of drivers of both heavy and light vehicles, who would not need to worry about their incompatibility with the other class of vehicles;
- Facilitating use of different operating speeds where appropriate, such as on steep grades;
- Facilitating different schemes for paying for highway use, based on different priorities for timely arrival (particularly for trucks engaged in “just-in-time” deliveries).

What is truck automation?

Truck automation provides the capability for a truck to be driven automatically, without the intervention of a driver. This is most likely to be used for access to freight movement hubs such as ports and intermodal rail terminals or for long-distance driving in special-purpose truck lanes, where the trucks could be separated from other traffic to simplify their driving environment and reduce hazards. The trucks could be electronically coupled to form virtual trains or platoons in order to save fuel and reduce emissions, as well as to increase lane capacity. The driver could still serve a supervisory or managerial role, while the automatic control system takes care of the tedious and stressful lower-level driving tasks. Or a driver could operate the first truck in the platoon while the others might be driverless.

Precursor technologies include drive- and steer-by-wire technologies, and also the various driver-assist and collision warning technologies currently under research with the IVI program and development by industry. But truck automation goes well beyond IVI and industry product development – in addition to the IVI safety goals, truck automation will directly and favorably impact fuel economy and other operating costs incurred in delivering goods to the consumer. Also, truck automation should contribute to reducing congestion by helping to increase highway capacity.

How will truck automation function and what technologies will it use?

Truck automation will consist of both automation functions and complementary functions and will also include certain design attributes.

Automation functions include:

- Automatic speed and spacing control
- Lane keeping
- Automatic backing to a loading dock

Complementary functions may include:

- Forward, side and rear collision warning and/or avoidance
- Driver drowsiness detection
- Vehicle condition warning
- Truck management center (operations) for the processing of information from advanced communication and advanced vehicle location systems

Supporting design attributes include:

- Infrastructure (dedicated truck lane, check-in and -out, interface to local non-automated traffic)
- Vehicle (electronic fueling and brake actuator, steering control, driver-vehicle interface)
- Wireless communication (vehicle-vehicle and vehicle-roadway)
- Fault management system

Why develop automated trucks now?

The earliest deployments of automation technologies will likely be on heavy vehicles — **buses** and **trucks**— operating on their own special rights of way because:

- Costs of the technologies are a smaller percentage of total truck costs, and trucks are used much more intensively so these costs are amortized much faster
- Benefits in travel-time reduction, trip reliability and safety can be translated more directly into cost savings than for private passenger cars
- Special truck-only lanes are already under consideration for other reasons
- Customized, small-lot production of vehicles makes it possible to introduce automation technologies into the truck production process faster than for automotive mass production
- Packaging of new technological elements is easier on trucks than on passenger cars
- Trucks already have more onboard electronic infrastructure to use as a foundation for more advanced capabilities than passenger cars
- Maturing technologies can be used more safely by professionally trained truck drivers on professionally maintained fleets than by the general public on passenger cars that may not be well maintained.

Who benefits from truck automation and how?

Commercial Vehicle Operators

- Economic benefits for commercial vehicle operators could be substantial. A heavy truck typically costs at least five times as much as a passenger car and would be driven annually at least ten times as far. Thus, the economic return from an investment in automation equipment is more attractive for trucks than for passenger cars. Moreover, it will probably be easier to install automation equipment on a truck than on a car for several reasons, e.g., less constrained space for packaging equipment, vehicles built to order in smaller quantities than passenger cars, shorter lead time from design to production.
- Perhaps most importantly, with truck platooning, the reduction in fuel consumption could be substantial – anywhere from 10 – 20 %. Hence, overall operating costs could be considerably reduced.
- Reduced and predictable travel times with automated trucks on dedicated lanes will improve the utilization of capital equipment, and improve the ability to meet performance targets for “just in time” deliveries.
- Potential safety improvements should have direct economic benefits in terms of reduced insurance costs and less time lost from productive use of trucks while being repaired for crash damage.
- Automated trucks could make possible significant changes in driving duty cycles and pay rates. Fully automated trucks could make it possible for drivers to travel long distances while resting, yet still earn payment. Some of the current problems with driver fatigue and duty hours conflicting with sleep cycles could be solved.

Truck Drivers

Automation of long-distance truck driving can improve the quality of life for truck drivers in several ways, particularly because they normally need to spend many hours a day driving, regardless of adverse weather, visibility and traffic conditions. Their benefits should include:

- Reduced stress, by avoiding direct interactions with bad behaviors of other drivers
- Smoother ride
- Ability to give attention to issues other than the immediate driving environment while driving
- Enhanced personal safety
- Reduced risk of fatigue on long trips
- Ability to operate under adverse conditions, without any increase in workload
- (In the long term) Potential changes to hours of service regulations and work rules, permitting more flexibility in scheduling driving times and possibly increasing earning potential as well.

Infrastructure Owners and Operators

Roadway owners and operators are concerned with traffic flow and congestion and public safety, as well as roadway maintenance and other operating expenses. Benefits include:

- Lane widths could be optimized for the different vehicle classes
- Collisions involving trucks and light-duty vehicles could be greatly reduced, saving lives and reducing injuries
- Separate lanes could be used for different speeds, more suitable for separate vehicle classes
- Roadway structure and pavement designs could be optimized for different vehicle classes
- Permitting operations on narrower rights-of-way (reduced lane widths and/or fewer number of lanes required), thereby saving on land use and physical infrastructure costs

General Public/Societal

By reducing aerodynamic drag when trucks are traveling in platoon formation, automation can reduce emissions and contribute to overall improvement in air quality, as well as saving energy and reducing dependence on imported petroleum.

What are some of the challenges facing successful deployment of truck automation?

The challenges discussed here are currently under investigation to find solution strategies:

- Industry concerns about the complexity, reliability and maintainability of new technologies
- Changing role of the driver (hours of service, driver training, salaries, and work rules, additional driver responsibilities)
- Safety
- Insurance
- Liability (changing assignment of risk and responsibility).

In the end, the benefits (primarily in reduced operating costs via fuel economy but with the addition of others discussed above) for each stakeholder category must be clear, calculable and attainable. If so, the decision from commercial as well as public sectors will be justifiable and easier.

What extra costs will be associated with truck automation technologies?

There will be incremental costs associated with the use of truck automation technologies, primarily in terms of additional electronic equipment installed on trucks and dedicated truck lanes (together with their continuing maintenance), and some additional protection for the dedicated lanes to prevent intrusion by unauthorized vehicles and debris.

Additional truck equipment is likely to consist of:

- Electronically-controlled brake actuator that may soon become standard equipment associated with anti-lock braking anyway
- Electronically-controlled steering actuator
- Lateral-position sensing system
- Forward ranging sensor system similar to commercially available sensors for adaptive cruise control systems
- Collision warning sensor systems which are already available for some vehicle applications and are under commercial development for others
- Vehicle-to-vehicle data communications system
- Vehicle-to-roadside data communications system which could build on existing and impending electronic toll collection technologies
- Onboard control computer that needs to be no more powerful than standard personal computers

How do we get from today's trucks to automation?

Existing precursor technologies are:

- By wire control
- Forward collision warning
- Lane-departure warning
- Adaptive cruise control (ACC)
- Vehicle-roadside communication
- Electronic tow bar

Some of these are being developed by manufacturers and suppliers, and some are undergoing testing via the USDOT IVI Program and the California PATH Program. The electronic tow bar concept is being tested by DaimlerChrysler, under a European Commission program called CHAUFFEUR.

With the above as a basis, special capabilities can be added to trucks with moderate levels of development and deployment costs, including:

- Vehicle-vehicle data communication for cooperative adaptive cruise control (CACC)
- Automated backing function or assistance

A next level of deployment includes dedicated-lane opportunities. This would be particularly useful in locations where there is high truck volume, and where congestion concerns are quite important. It could also be useful where road maintenance and safety with car-truck interactions are important. There are locales (e.g., I-15, I-710 and SR-60 in Southern California, and I-10 across the U.S.) where such dedicated lanes have been considered. It then becomes possible to implement automatic steering control safely, permitting use of a narrower lane and relieving the driver of the steering responsibility.

These technologies can form the basis of an initial operational scenario for truck automation, such as a long-distance line-haul operation.

At the entrance to the dedicated lane, the driver would switch the truck to automated operation, join behind another truck and operate automatically until he reached his destination.

Over the longer term, with further advancements:

- The entry maneuvers could be automated with the addition of more sophisticated vehicle-
roadside communication
- Higher-level management functions could be implemented to serve a network of connected
dedicated truck lanes. These could indeed become the precursors to an automated highway
network.

1.5 Overview of Report

This report provides documentation of the development of the experimental buses and trucks and their testing, to show what performance can be achieved with use of the automation technologies. Section 2 describes the host vehicle platforms that were chosen for automation, both buses and trucks. Section 3 then describes the hardware and software modifications that were made to the production vehicles in order to give them automated driving capabilities, while Section 4 reports on the performance that was achieved by the automation systems, with experimental data derived from testing of the vehicles. Section 5 provides a brief set of conclusions regarding what was learned from the testing and from the reactions of visitors who participated in some of the testing.

References

1. S.E. Shladover, “Bus Rapid Transit and Automation – Opportunities for Synergy”, *Proceedings of Seventh World Congress on Intelligent Transport Systems*, Turin, Italy, November 2000.
2. S.E. Shladover, “Opportunities in Truck Automation”, *Proceedings of Eighth World Congress on Intelligent Transport Systems*, Sydney, Australia, October 2001, Paper No. ITS00155.

Chapter 2. Host Vehicle Platforms

2.1 Transit Buses

The transit buses were chosen to represent two significantly different vehicle types in order to demonstrate that the automation technology could work effectively across a representative sample of heavy-duty buses in regular service in the U.S. In particular, two of the buses were 12 m (40 ft.) single-unit buses powered by compressed natural gas (CNG) in spark-ignition engines, since this type of propulsion system has become increasingly popular among U.S. transit agencies in order to reduce bus emissions of pollutants and noise. The third bus is an 18 m (60 ft.) articulated bus powered by a diesel engine, as used in many of the heaviest-duty transit services in the U.S. All three buses were acquired from New Flyer through the cooperation of San Diego Transit, who added them to a pre-existing order that they had already placed with New Flyer. Since that order was already in the production queue, it saved approximately one year of delay for delivery, which would have occurred if we had placed a completely new order for the demonstration buses.

It was important to show that buses with significantly different engines and performance characteristics could be closely-coupled within an automated platoon for the demonstration, in order to verify the flexibility of the longitudinal control systems that were being demonstrated. Similarly, the lateral dynamics of single-unit and articulated buses differ significantly from each other, yet the same lateral control system was demonstrated to work effectively on both types of buses.

Although provided by the same manufacturer, New Flyer, the 12 m (40 ft.) and 18 m (60 ft.) buses had different components. The 18 m (60 ft.) articulated bus is powered by a diesel engine, provided by Detroit Diesel, and both the 12 m (40 ft.) single-unit buses are powered by a CNG spark-ignition engine, provided by Cummins. Other relevant specifications are in Table 2.1.1 below:

Table 2.1.1. New Flyer Bus Specifications. Engine and Transmission Details for both 18 m and 12 m Buses.

	18 m (60 ft.) Articulated	12 m (40 ft.) Single-Unit
Engine Mfr.	Detroit Diesel ¹	Cummins
Engine Model	Series 50	CGAS+280
HP	330	280
RPM	2100	2400
Transmission Mfr.	Allison	Allison
Transmission Model	B500R	B400
Weights (Curb Weight and Gross Veh. Weight)	CW: 41,500 lbs (18,820kg) GVWR: 63,880 lbs (28,980 kg)	CW: 28,875 lb (13,100 kg) GVWR: 39,630 lb (17,980 kg)

¹ The Detroit Diesel has a max torque of 850 lb-ft at 1400 RPM.

2.1.1 Longitudinal Dynamics of Transit Buses

A mathematical model was developed of the longitudinal dynamics of the buses in order to support the development of the longitudinal controller. The model had to be complex enough to capture dynamic characteristics of the system, and simple enough to make the controller efficient for real-time application. In this section, it will be shown how the control model can be simplified with information via the J1939 bus. Moreover, the control model will be unified in the sense that the equations of motion between two transit bus models are similar, with the exception of minor differences. It will allow us to design the unified longitudinal controller later. Finally, all variables and parameters are listed in Tables 2.1.2 and 2.1.3 at the end of Section 2.1.1.

① Longitudinal equations of motion

Three-state nonlinear equations of motion are proposed to represent both a 40-foot and a 60-foot transit bus. One state is used for chassis dynamics and the others are for engine and brake dynamics respectively. Before deriving the longitudinal dynamics of the bus, the considered assumptions are summarized as follows:

- A symmetric rigid body of vehicle chassis
- No slip between the wheels and ground, i.e., $v = \omega_w \cdot h$.
- The torque converter is locked, i.e., $\omega_w = \omega_e \cdot R_g$.
- The accessory engine power is constant, i.e., $T_{acc} \cdot \omega_e = C(\text{A/C, fan, ...})$.
- The vehicle mass and road grade are known a priori.

It is noted that the constant value in the next-to-last assumption relies on the status of an air conditioner (A/C), a fan, and thermal conditions. Moreover, huge changes of the mass and road grade have large impacts on the longitudinal vehicle dynamics. Therefore, if they cannot be pre-determined, it may be necessary to estimate them dynamically, as in [1].

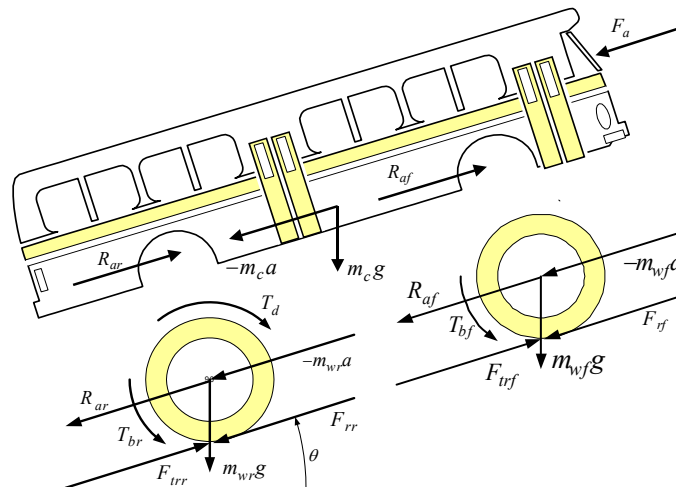


Figure 2.1.1 Free body diagram for longitudinal motion

Using the first through third assumptions above and balancing the forces in the longitudinal direction, we can derive the longitudinal equation of motion. However, these longitudinal dynamics were introduced in [2] and similarly used for the passenger vehicle [3]. Hence, the resulting single state dynamics are derived without detailed descriptions as follows:

$$\dot{v} = \frac{T_e - T_{acc} - R_g \cdot T_b}{J_{eq}} - f_1 \quad (2.1.1)$$

where J_{eq} and f_1 represent

$$J_{eq} = \frac{J_e + R_g^2 \cdot (J_w + m \cdot h^2)}{R_g \cdot h},$$

$$f_1 = \frac{R_g \cdot h}{J_{eq}} (F_a + F_r + m \cdot g \cdot \sin \theta) = \frac{R_g \cdot h}{J_{eq}} (C_a \cdot v^2 + C_r \cdot m \cdot g + m \cdot g \cdot \sin \theta)$$

It is remarked that there are three variables in (2.1.1) which may vary dramatically with respect to time. That is, we need to have the corresponding models for the production of net engine torque (T_e), brake torque (T_b), and effective gear ratio (R_g), which are discussed next.

② Engine Model

In general, the characteristics of the engine are highly nonlinear and complicated, no matter whether it is a CNG or a diesel engine [4, 5]. There are even several proprietary engine controllers embedded by the engine manufacturer for the purposes of engine protection, noise and emission reduction [6]. However, regardless of the type and complexity of engines, the measurement of the engine torque which is transferred to a transmission may be preferable in the sense that the ultimate goal of the engine model is to estimate the generated engine torque in equation (2.1.1). Despite this intuitive idea, many complex engine models have been developed in the literature to estimate the engine torque due to lack of reliability and accuracy of the torque sensor as well as cost consideration [7, 8]. However, it is interesting to remark that the in-vehicle sensor information available via the J-1939 bus has the indicated engine torque (so-called the actual engine torque in SAE J-1939 handbook [6]) and nominal friction engine torque, which are measured and/or calculated by the engine manufacturer [6]. Hence, using this information, the measured engine torque is regarded as

$$T_{e.meas} = T_{ind} - c_e \cdot T_{fric} \quad (2.1.2)$$

where c_e is a tunable parameter which will be identified later with comparison of manual driving data of transit buses.

Two engine control actuator methods are used for two different types of engines as follows: one is to use a torque control command (TCC) via the J-1939 bus for the diesel engine and the other is to use an acceleration pedal command via an analog voltage signal for the CNG engine. Although the use of TCC allows us to control the engine torque directly in the sense that the empirical engine map or data are not necessary, the Cummins CNG engine on the 40-foot New Flyer transit bus does not provide the

capability of TCC yet. Hence, the approach previously used for gasoline engine control was used for the CNG engine due to their similarities shown in [5]. That is, an empirical engine map is used to capture the characteristics of the CNG engine and its controllers quantitatively. Finally, the dynamics of the two engines are represented by the first order lag systems as follows [2]:

$$\dot{T}_e = \begin{cases} \frac{1}{\tau_{cng}} \{-T_e + T_{map}(\omega_e, u_\alpha)\} & \text{for a CNG engine} \\ \frac{1}{\tau_{diesel}} \{-T_e + T_{cmd}(t - \Delta t_e)\} & \text{for a diesel engine} \end{cases} \quad (2.1.3)$$

where T_{map} is the empirical engine torque map which indicates the net engine torque for the given engine speed and accelerator pedal position, and T_{cmd} is the torque control command via a J1939 bus.

③ Transmission model

The transmission is also a highly nonlinear and complex system like the engine above. It is generally very hard to derive a simple set of mathematical equations to represent a complete transmission model. However, if equation (2.1.1) is simple enough to capture the equation of motion for the transit bus, only two values are required from the transmission model, i.e., effective gear ratio (R_g) and transmission retarder braking torque (T_{tr}). The former comes from the engaged gear ratio (R_t) multiplied by the final gear ratio. The transmission information directly available from the J1939 bus includes selected and current gear, actual gear ratio, and an index of “shift in progress”. However, accurate gear ratio information is not available while a gear shift is occurring. The gear ratio during gear shifting can be estimated as follows: If the shift-in-progress is on,

$$\begin{cases} \dot{R}_t = \frac{1}{\tau_g} \{-R_t + R_{sel}(t - \Delta t_g)\} & \text{during shift - in - progress} \\ R_t(t) = R_{cur}(t) & \text{otherwise} \end{cases} \quad (2.1.4)$$

where R_{sel} and R_{cur} are the selected and current gear ratio, respectively.

A plot of the measured gear shift information and the model estimate for a representative experimental run is shown in Figure 2.1.2. The bottom plot shows estimation error in the term of gear ratio where the measured value is based on both input and output shaft speed in the transmission. It is noted that some peaks in the figure result from uncertainty in pure time delay (Δt_g) in equation (2.1.4), while the constant time delay is used for the estimation.

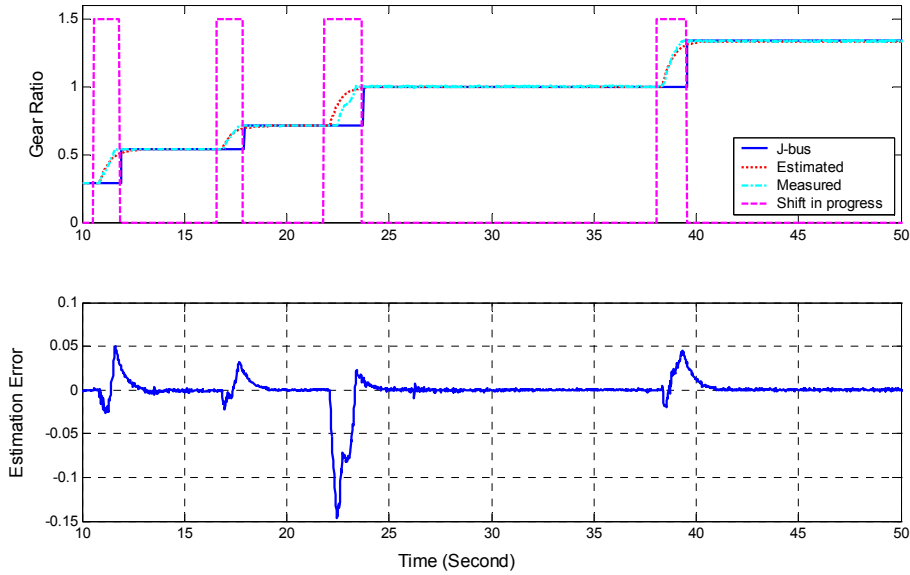


Figure 2.1.2 Time response of gear ratio from 1st to 5th gear and the corresponding gear ratio estimation error

The transmission retarder braking torque required from the transmission model is controlled and reported by the J-bus. Although the capacity of the transmission retarder braking torque is limited, its time response is usually faster than that of a pneumatic braking system. Hence, an integrated braking approach combining the transmission retarder and pneumatic braking torque is used for the braking control. Then, the overall braking torque can be decoupled as follows:

$$T_b = T_{tr} + T_{pn}$$

where T_{tr} is produced by the percentage transmission retarder torque command via J-bus [6]. Furthermore, it is noted that the transmission retarder dynamics are neglected for the control model because they are relatively faster than chassis and pneumatic brake dynamics. Next, it will be discussed in the following section how T_{pn} from the pneumatic brake system is produced and modeled.

④ Pneumatic brake model

A schematic of a pneumatic brake system for a front wheel of the transit bus is shown in Figure 2.1.3. While the pressure coming from the dual brake valve in the figure is typically controlled by a driver's foot for manual driving, an electrical brake actuator is implemented in parallel to generate additional brake pressure, P_{bv} . Sequentially, the brake pressure is the maximum of two pressure values through the mixing valve shown in Figure 2.1.3. Assuming that brake torque has a proportional relation with the brake pressure in the diaphragm chamber of a pneumatic brake system and that there is no driver command, the brake pressure dynamics to the brake chamber shown in Figure 2.1.3 can be derived as follows [9]:

$$T_{pn} = \begin{cases} K_b(P_b - P_o) & \text{if } P_b \geq P_o \\ 0 & \text{otherwise} \end{cases}$$

$$\dot{P}_b = \begin{cases} \frac{1}{\tau_{bf}} [-P_b + P_{bv} \{u_\beta(t - \Delta t_b)\}] & \text{for filling} \\ \frac{1}{\tau_{be}} [-P_b + P_{bv} \{u_\beta(t)\}] & \text{for emptying} \end{cases} \quad (2.1.5)$$

where P_{bv} is an empirical function of the brake control command u_β .

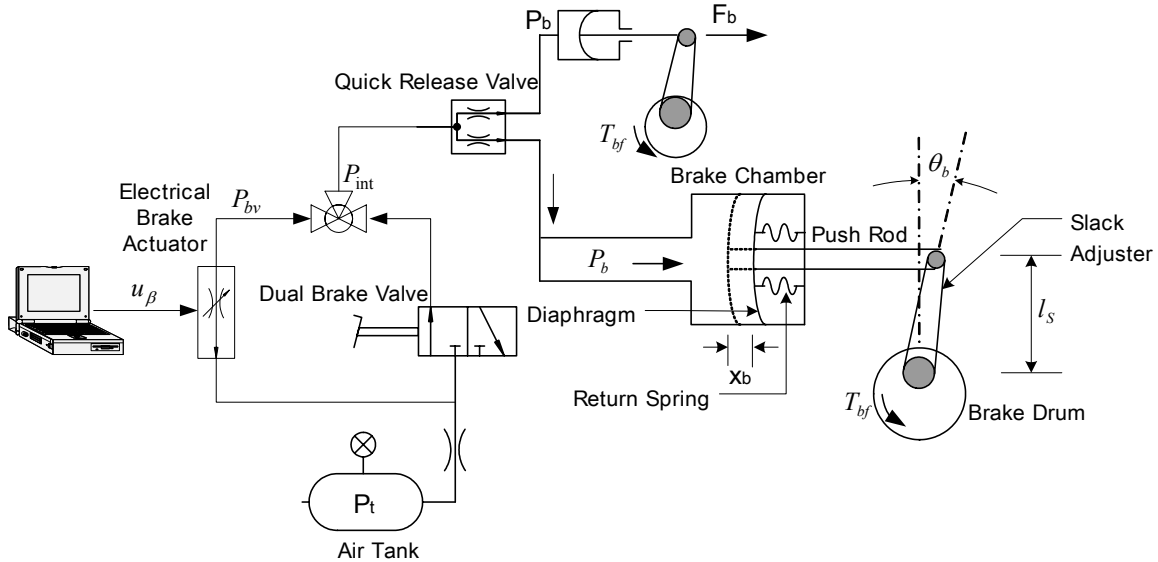


Figure 2.1.3 Schematic of a pneumatic brake system with an electrical brake actuator

The time responses for step inputs to the pneumatic brake system with an electrical brake actuator are presented in Figure 2.1.4 when the transit bus is stationary. Three brake pressure measurements from brake pressure transducers located in different positions are shown in the figure: P_{bv} is measured from an embedded sensor in the electrical brake actuator, P_{int} and P_b are measured near the mixing valve and the front wheel, respectively (also see in Figure 2.1.3). Finally, the time response of the brake pressure model shown in equation (2.1.5) is plotted together to show the accuracy of the proposed model.

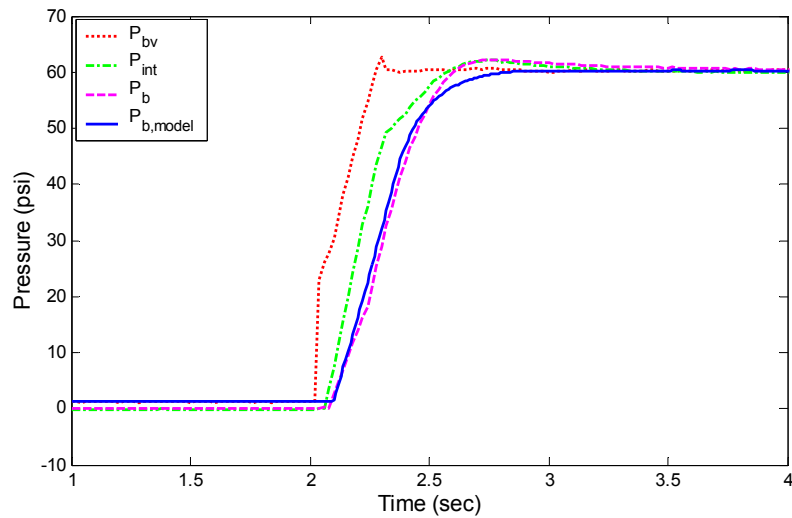


Figure 2.1.4 Time responses of pneumatic brake pressures in the air brake system and estimated brake pressure from the brake model

⑤ Model validation

Data acquisition of two types of the transit buses manually driven was performed at the Crow's Landing Test Facility. All information such as wheel-based speed, engine speed, current gear, and an accelerator pedal position were acquired via the J1939 bus. Then, their time responses are compared with those of the transit bus model proposed above. For instance, Figure 2.1.5 shows the time responses of the 40-foot transit bus in the operating velocity range from 10 to 25 (m/s), during which the gear shifts from 2nd to 5th gear and the torque converter remains locked. With the appropriate system parameters listed in Table 2.1.3, the accuracy of the model is such that the deviation in terms of velocity and engine speed is within 5% as shown in Figure 2.1.5 (a) and (b).

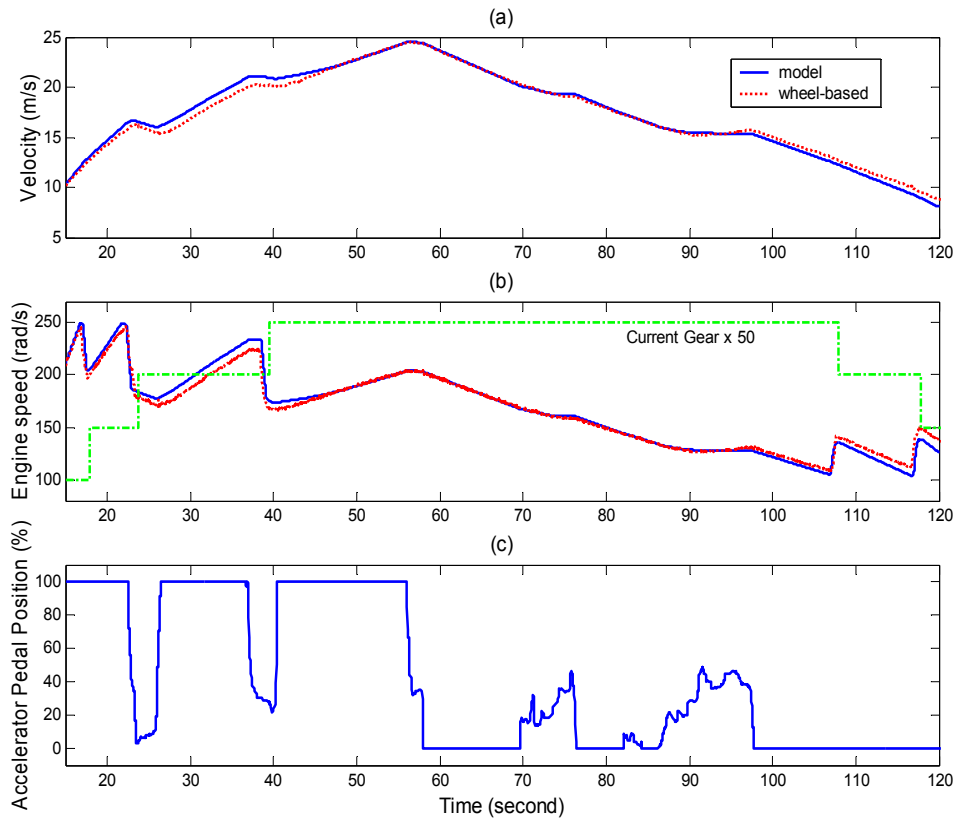


Figure 2.1.5 Time responses of velocity, engine speed, and acceleration pedal position for a 40-ft transit bus driven manually

Similarly, the experimental validation of the 60-foot bus model is shown in Figure 2.1.6. With the appropriate system parameters listed in Table 2.1.3, Figure 2.1.6 shows the time responses of the 60-foot articulated bus in the operating velocity range up to 23 (m/s). The accuracy of the model is described in terms of the velocity and engine speed deviation, and is within 5-6% as shown in Figure 2.1.6 (a) and (b). Consequently, these experimental results in Figure 2.1.5 and 2.1.6 show that the proposed control model represents longitudinal dynamics of two different types of transit buses qualitatively.

Finally, Figure 2.1.7 shows how much the change of aerodynamic coefficient and accessory engine torque affects the overall vehicle dynamics in terms of vehicle velocity and engine speed. In other words, a lane-change maneuver was conducted and an air conditioner (A/C) was switched on and off during the manual driving in order to change the aerodynamic coefficient and accessory engine torque due to A/C. Figure 2.1.7 (a) and (b) show that the velocity deviation remains within 5-8%. Although it may be said that the model accuracy is degraded due to the change of system parameters, it would rather be interpreted that the change of aerodynamic coefficient and accessory engine torque is insensitive to the proposed control model.

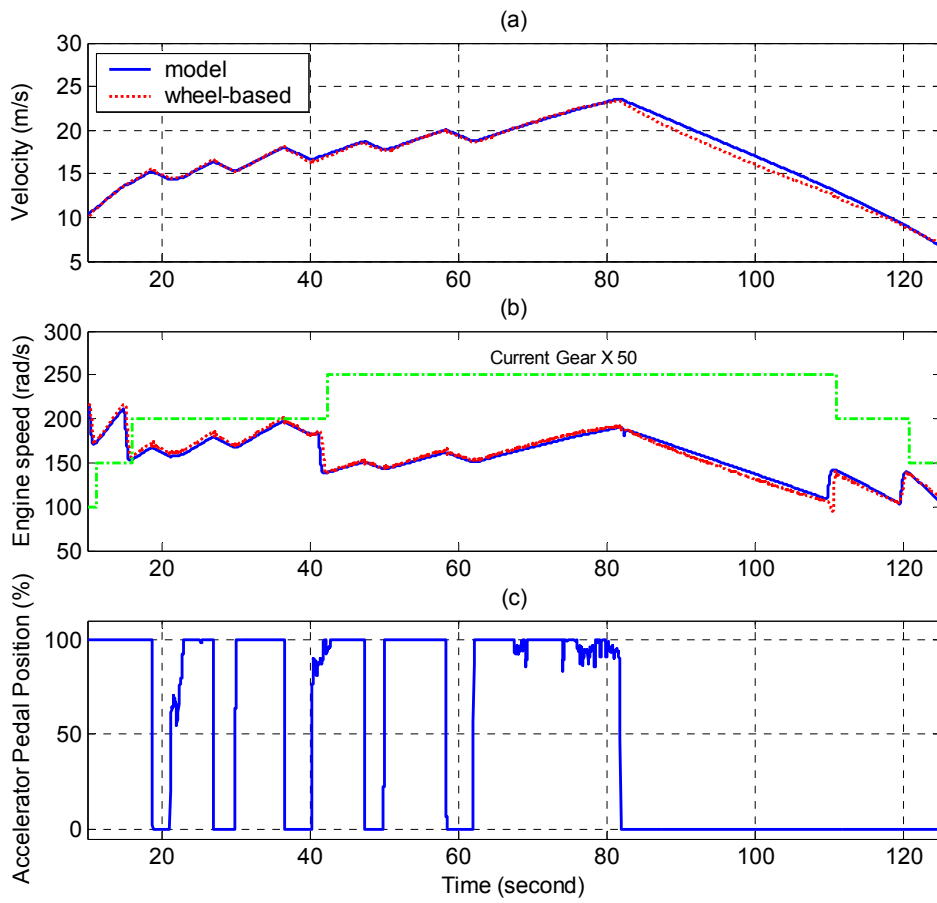


Figure 2.1.6 Time responses of velocity, engine speed, and acceleration pedal position for a 60-ft articulated bus driven manually

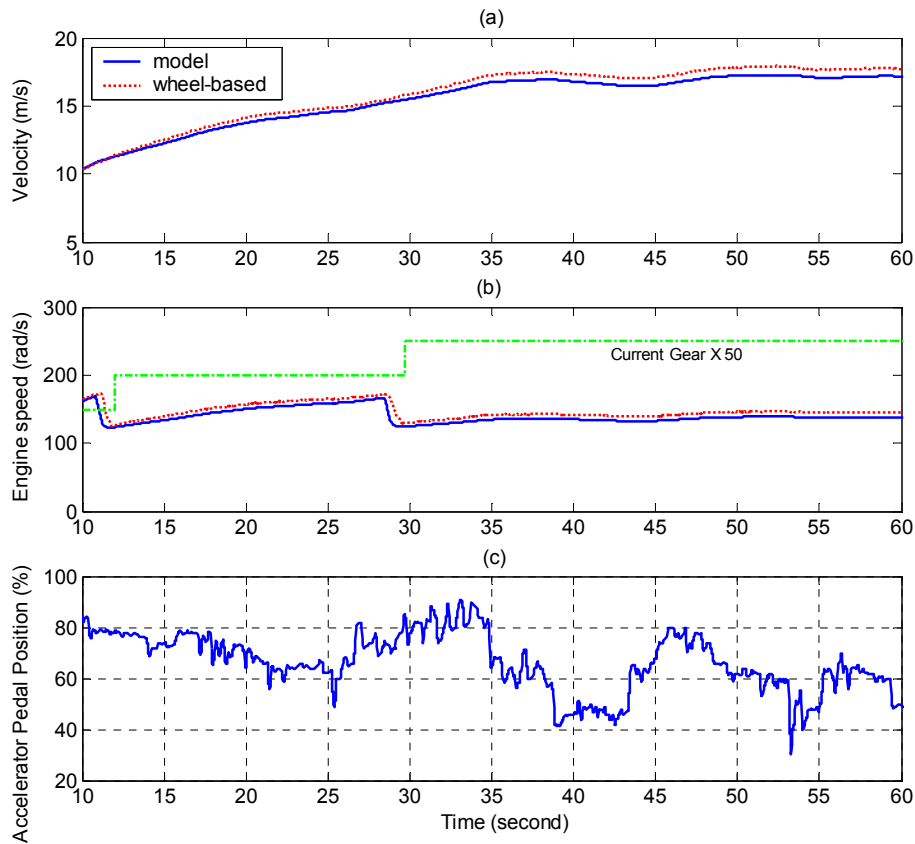


Figure 2.1.7 Time responses of a transit bus driven manually during lane-change maneuver

Table 2.1.2 Variables for Transit Bus Longitudinal Control Model

Variable	Description	Variable	Description
v	Velocity of the vehicle [m/s]	R_g	Effective gear ratio, i.e., $R_f \cdot R_f$
R_f	Gear ratio in a transmission	T_{acc}	Accessory engine torque [N·m]
T_e	Net engine torque [N·m]	T_b	Braking torque [N·m]
T_{ind}	Indicated engine torque [N·m]	T_{fric}	Frictional engine torque [N·m]
T_{tr}	Transmission retarder braking torque [N·m]	T_{pn}	Braking torque coming from a pneumatic brake system [N·m]
F_r	Rolling resistance force [N]	F_a	Aerodynamic drag force [N]
ω_e	Engine speed [rad/s]	ω_w	Wheel speed [rad/s]
u_α	Accelerator pedal position	u_β	Brake control command
θ	Road slope angle [rad]	P_b	Brake line pressure [KPa]

Table 2.1.3 Parameter Values for Transit Bus Longitudinal Control Model

Parameter	Description	Value for a 40-ft bus	Value for a 60-ft bus
h	Effective wheel radius	0.4775 [m]	0.4775 [m]
R_f	Final drive ratio	0.1887	0.1916
m	Total weight of the vehicle	13,381 [Kg]	18,757 [Kg]
J_e	Inertia of the engine	1.8818 [Kg·m ²]	1.2117 [Kg·m ²]
J_w	Overall inertia of the axle	42.4 [Kg·m ²]	63.6 [Kg·m ²]
P_o	Push-out pressure	34.48 [KPa]	34.48 [KPa]
C_r	Rolling resistance coefficient	0.01	0.0175
C_a	Aerodynamic drag coefficient	2.9436 [Kg/m]	2.4242 [Kg/m]
$\tau_{cng}, \tau_{diesel}$	Time constant for engine and throttle actuator delay	$\tau_{cng} = 0.03$	$\tau_{diesel} = 0.01$
Δt_e	Pure time delay for TCC		0.03
τ_{bf}	Time constant for brake actuator during filling process	0.13	0.13
τ_{be}	Time constant for brake actuator during emptying process	0.07	0.07
Δt_b	Pure time delay during filling process	0.07	0.07
τ_g	Time constant for gear shifting	0.35	0.35
Δt_g	Pure time delay during gear shifting	0.4	0.4
g	gravity	9.81 [m/s ²]	9.81 [m/s ²]
K_b	Brake torque coefficient	10 [N·m/KPa]	10 [N·m/KPa]
c_e	Tunable parameter for net engine torque	1	0.3
$C(A/C \text{ on})$	Accessory engine torque with A/C on	36.0 (Hp)	36.0 (Hp)
$C(A/C \text{ off})$	Accessory engine torque with A/C off	30.8 (Hp)	30.8 (Hp)

2.1.2 Lateral Dynamics of Transit Buses

Mathematical models were developed for the lateral dynamics of the buses to support the development of the lateral controller. A simple bicycle model, a 3-DOF yaw/lateral/roll model, and a bicycle model with trailer were used to represent the fundamental vehicle dynamics of the transit buses at various design stages.

2.1.2.1 Bicycle Model

Bicycle model is the most common model used for lateral control design. A linearized model is generally regarded to be sufficient for studying vehicle steering under *normal* conditions. Assuming small angles, this allows the use of the classical *bicycle model*

shown in Figure 2.1.8. The linearized state-space model, derived by balancing the force and moment equations, can be found in [24] as

$$\frac{d}{dt} \begin{bmatrix} \beta \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{c_f + c_r}{Mv} & -1 + \frac{c_r l_r - c_f l_f}{Mv^2} \\ -\frac{c_f l_f - c_r l_r}{I_\psi} & -\frac{c_r l_r^2 + c_f l_f^2}{I_\psi v} \end{bmatrix} \begin{bmatrix} \beta \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{c_f}{Mv} \\ \frac{c_f l_f}{I_\psi} \end{bmatrix} \delta_f \quad (2.1.6)$$

where β is the side slip angle between vehicle longitudinal axis and velocity vector v at CG, and $\dot{\psi}$ the vehicle yaw rate. Other variables in Eq. (2.1.6) are: δ_f the front steering angle, I_ψ the yaw moment of inertia, M the mass of the vehicle, $l=l_f+l_r$ the wheel base, and c_f and c_r the linear cornering stiffness of the front and rear tires, respectively.

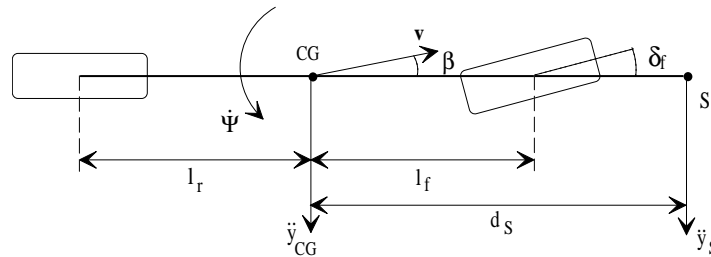


Figure 2.1.8: Bicycle model

2.1.2.2 Vehicle Model with Roll Dynamics Coupling

Many researchers use the bicycle model for vehicle steering control. However, most buses have a relatively soft suspension. The soft suspension significantly alters the frequency characteristics of the lateral vehicle dynamics from that predicted by the bicycle model. In order to study the influence of the suspension roll dynamics on the steering control design, a 3 DOF vehicle model that includes lateral, yaw and roll dynamics is used for the controller design. The schematic diagram of the 3 DOF vehicle model is shown in Figure 2.1.9. The sprung mass (m_s) interacts with the front and rear unsprung masses via the front and the rear suspensions, where K_f , D_f and K_r , D_r are the rotational spring and damper coefficients for front and rear suspension, respectively. The roll axis is defined as the line connecting the roll centers of the front and rear suspension as shown in Figure 2.1.9. It can be found that the vehicle geometric parameter that affects the coupling between lateral and roll dynamics the most is h_{m_s} , the distance between the sprung mass CG and the roll axis. Additional vehicle parameters are defined as follows: β is the side slip angle between vehicle longitudinal axis and velocity vector v at CG, and $\dot{\psi}$ the vehicle yaw rate. Other parameters are the same as the bicycle model: δ_f the front steering angle, I_ψ the yaw moment of inertia, M the mass of the vehicle, $l=l_f+l_r$ the wheel base, and c_f and c_r the linear cornering stiffness of the front and rear tires, respectively.

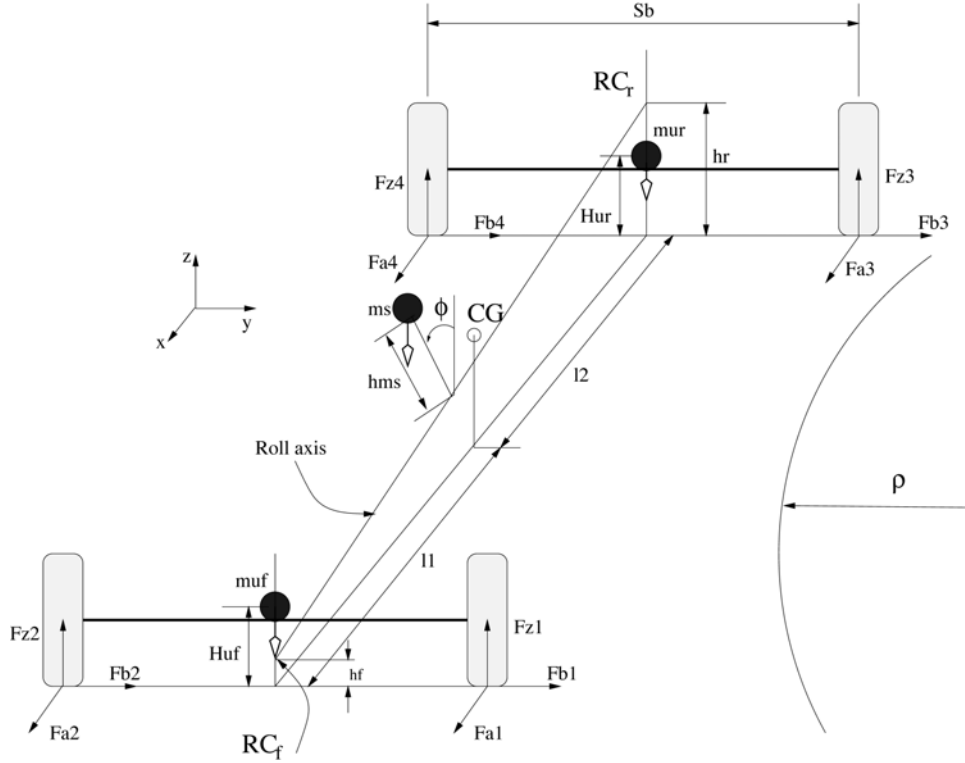


Figure 2.1.9: Schematic diagram of 3 DOF vehicle model

Under constant speed condition, the dynamic equations of motion are derived using the Newtonian method. Assuming small angles and using the linear tire model, the linear dynamics equations of the vehicle model with respect to the road reference frame are derived and shown in Eq. (2.1.7) [25]. The state-space representation takes the form of $\dot{\bar{x}} = A\bar{x} + B\delta + \bar{d}$. The state variables are: y_r, \dot{y}_r , the lateral displacement at CG w.r.t. road reference frame and its derivative; $\psi_r, \dot{\psi}_r$, the yaw angle w.r.t. the road reference frame and its derivative; as well as $\phi_r, \dot{\phi}_r$, the roll angle and its derivative. The road reference frame is attached to the road center at a point adjacent to the vehicle CG with X axis tangent to the road trajectory and moves along the road with the same speed as the vehicle. The input is the front steering angle (δ). The disturbances are: ρ , the road curvature; $\dot{\psi}_d$, the desired yaw rate from the road; F_{wy} , the disturbance force at CG along the y direction; and F_x , the front tire force along the tire orientation.

$$\frac{d}{dt} \begin{bmatrix} y_r \\ \dot{y}_r \\ \psi_r \\ \dot{\psi}_r \\ \phi_r \\ \dot{\phi}_r \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{A_1\Lambda_1}{v} & -A_1\Lambda_1 & \frac{A_2\Lambda_1}{v} & R_1 & R_2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{A_3}{v} & -A_3 & \frac{A_4}{v} & R_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{A_1\Lambda_2}{v} & -A_1\Lambda_2 & \frac{A_2\Lambda_2}{v} & R_4 & R_5 \end{bmatrix} \begin{bmatrix} y_r \\ \dot{y}_r \\ \psi_r \\ \dot{\psi}_r \\ \phi_r \\ \dot{\phi}_r \end{bmatrix} + \begin{bmatrix} 0 \\ B_1\Lambda_1 \\ 0 \\ B_2 \\ 0 \\ B_1\Lambda_2 \end{bmatrix} \delta + \begin{bmatrix} 0 \\ d_1 \\ 0 \\ d_2 \\ 0 \\ d_3 \end{bmatrix} \quad (2.1.7)$$

where the coefficients are defined as follows:

$$A_1 = -\frac{2(c_f + c_r)}{M},$$

$$A_2 = \frac{2(-l_f c_f + l_r c_r)}{M},$$

$$A_3 = \frac{2(-l_f c_f + l_r c_r)}{I_\psi},$$

$$A_4 = \frac{-2(l_f^2 c_f + l_r^2 c_r)}{I_\psi},$$

$$\Lambda_1 = \frac{MI_{x_s}}{MI_{x_s} - m_s^2 h_{m_s}^2},$$

$$\Lambda_2 = \frac{Mm_s h_{m_s}}{MI_{x_s} - m_s^2 h_{m_s}^2},$$

$$R_1 = \frac{2(\gamma_f + \gamma_r)I_{x_s} + m_s h_{m_s} (m_s g h_{m_s} - K_f - K_r)}{MI_{x_s} - m_s^2 h_{m_s}^2},$$

$$R_2 = \frac{-m_s h_{m_s} (D_f + D_r)}{MI_{x_s} - m_s^2 h_{m_s}^2},$$

$$R_3 = \frac{2(\gamma_f l_f - \gamma_r l_r)}{I_\psi},$$

$$R_4 = \frac{2m_s h_{m_s} (\gamma_f + \gamma_r) + M(m_s g h_{m_s} - K_f - K_r)}{MI_{x_s} - m_s^2 h_{m_s}^2},$$

$$R_5 = \frac{-M(D_f + D_r)}{MI_{x_s} - m_s^2 h_{m_s}^2},$$

$$B_1 = \frac{2(c_f + F_x)}{M},$$

$$B_2 = \frac{2l_f (c_f + F_x)}{I_\psi},$$

$$d_1 = \frac{\Lambda_1}{M} F_{wy} + \frac{A_2 \Lambda_1}{v} \dot{\psi}_d - v^2 \rho,$$

$$d_2 = -\frac{2(-l_f^2 c_f + l_r^2 c_r)}{I_\psi v} \dot{\psi}_d,$$

$$d_3 = \frac{\Lambda_2}{M} F_{wy} + \frac{A_2 \Lambda_2}{v} \dot{\psi}_d - 2\Lambda_2 v^2 \rho \quad (2.1.8)$$

2.1.2.3 Articulated Bus Model

The lateral dynamic model for basic tractor and semi-trailer combination was adopted for the single articulated bus [26, 27] where the articulated section of the bus is treated as the trailer and the front section of the bus is treated as the tractor. The states of the model are described relative to a “road coordinate system” which is centered at a point on the road centerline such that it is closest to the tractor center of gravity. The axes of this coordinate system are aligned along the tangent, normal and binormal to the road centerline. Under simplifying assumptions such as small relative yaw of the tractor with respect to the road, small articulation angle and small steering angle, the dynamics of the bus front section and the bus articulated section, similar to the tractor semi-trailer, in the form of $\dot{\bar{x}} = A\bar{x} + B\delta + \bar{d}$, are given by:

$$\frac{d}{dt} \begin{bmatrix} q_r \\ \dot{q}_r \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} q_r \\ \dot{q}_r \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}F \end{bmatrix} \delta + \begin{bmatrix} 0 \\ E_1 \end{bmatrix} \dot{\epsilon}_d + \begin{bmatrix} 0 \\ E_2 \end{bmatrix} \ddot{\epsilon}_d \quad (2.1.9)$$

$$y_s = [1 \quad d_s \quad 0 \quad 0 \quad 0 \quad 0] \begin{bmatrix} q_r^T \\ \dot{q}_r^T \end{bmatrix} \quad (2.1.10)$$

where $q_r = [y_r, \epsilon_r, \epsilon_f]^T$ and \dot{q}_r are the states of the system. y_r is the distance of the bus front section center of gravity from the road centerline, ϵ_r is the yaw angle between the bus articulated section longitudinal axis and the axis of the road coordinate system along the tangent to road centerline, and ϵ_f is the articulation angle (see Figure 2.1.10). A lateral displacement sensor (either a real sensor or a virtual sensor located at distance d_s meters ahead of the bus front section’s center of gravity) measures its distance, y_s , from the road centerline. The input to the model is δ , the steering angle of the front wheel. $\dot{\epsilon}_d = v\rho$ is the angular velocity of the road coordinate system where v is the longitudinal velocity of the vehicle and ρ is the road curvature. Matrices appearing in the system equation are given below.

$$M = \begin{bmatrix} m_1 + m_2 & -m_2(d_1 + d_3) & -m_2 d_3 \\ -m_2(d_1 + d_3) & I_{z1} + I_{z2} + m_2(d_1 + d_3)^2 & I_{z2} + m_2 d_3^2 + m_2 d_1 d_3 \\ -m_2 d_3 & I_{z2} + m_2 d_3^2 + m_2 d_1 d_3 & I_{z2} + m_2 d_3^2 \end{bmatrix} \quad (2.1.11)$$

$$C = \frac{2}{v} \begin{bmatrix} C_f + C_r + C_t & l_1 C_f - l_2 C_r - (l_3 + d_1) C_t & -l_3 C_t \\ l_1 C_f - l_2 C_r - (l_3 + d_1) C_t & l_1^2 C_f + l_2^2 C_r + (l_3 + d_1)^2 C_t & l_3(l_3 + d_1) C_t \\ -l_3 C_t & l_3(l_3 + d_1) C_t & l_3^2 C_t \end{bmatrix} \quad (2.1.12)$$

$$K = \begin{bmatrix} 0 & -2(C_f + C_r + C_t) & -2C_t \\ 0 & -2(l_1 C_f - l_2 C_r - (l_3 + d_1)C_t) & 2(l_3 + d_1)C_t \\ 0 & 2l_3 C_t & 2l_3 C_t \end{bmatrix} \quad (2.1.13)$$

$$F = 2C_f [1 \quad l_1 \quad 0]^T \quad (2.1.14)$$

$$E_1 = \begin{bmatrix} -(m_1 + m_2)v - \frac{2}{v}(l_1 C_f - l_2 C_r - (l_3 + d_1)C_t) \\ m_2(d_1 + d_3)v - \frac{2}{v}(l_1^2 C_f + l_2^2 C_r + (l_3 + d_1)^2 C_t) \\ m_2 d_3 v - \frac{2}{v} l_3 (l_3 + d_1) C_t \end{bmatrix} \quad (2.1.15)$$

$$E_2 = \begin{bmatrix} m_2(d_1 + d_3) \\ -(I_{z1} + I_{z2} + m_2(d_1 + d_3)^2) \\ -(I_{z2} + m_2 d_3^2 + m_2 d_1 d_3) \end{bmatrix} \quad (2.1.16)$$

m_1 and m_2 are the mass of the bus front section and the mass of bus articulated section respectively. I_{z1} and I_{z2} are the moments of inertia of the bus front section and the bus articulated section respectively. $2C_{af}$, $2C_{ar}$ and $2C_{at}$ are the combined cornering stiffness coefficients of the front tires of the bus front section, rear tires of the front section and tires of the articulated section respectively. Geometric parameters associated with the tractor semi-trailer model are illustrated in Figure 2.1.10.

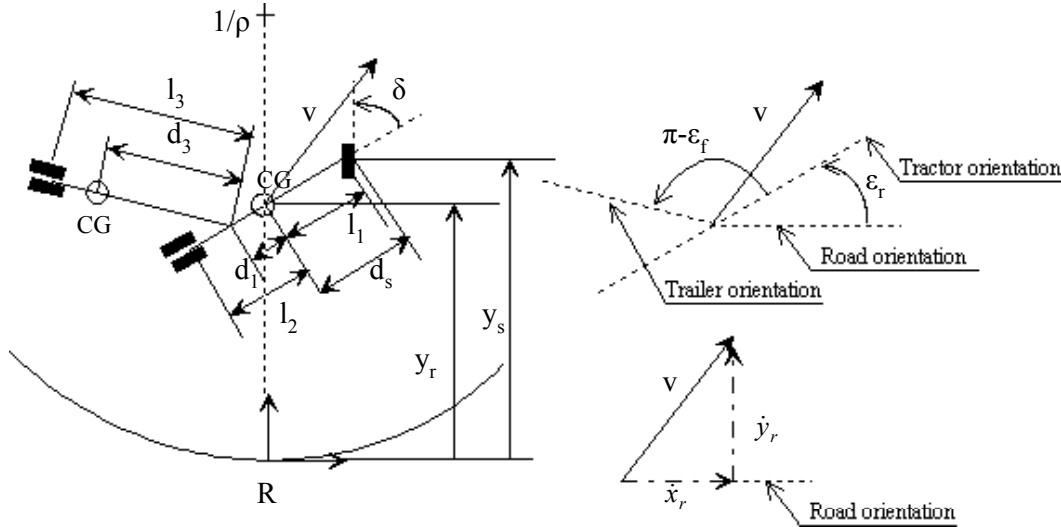


Figure 2.1.10: Geometric parameters of the tractor semi-trailer/articulate bus model

2.1.2.4 Bus Parameters

The parameters for the models for the New Flyer 40 ft CNG buses and 60 ft articulated bus are shown in Table 2.1.4, and Table 2.1.5, respectively. Due to delays in the bus instrumentations and the high noise in the inertial sensors measurements during the initial data collection phase, no validation data were available before the buses were shipped to San Diego. Therefore, the bus parameters were generated using available data with certain engineering guesses. The values in Table 2.1.4 and Table 2.1.5 with * indicate there are some uncertainties in these numbers. The controller designs were thus focused on robustness that tolerate significant parameter uncertainties.

Table 2.1.4 New Flyer 40 foot CNG Bus Parameters

Description	Values
Mass of empty bus (kg)	13,381
Bus yaw moment of inertia (kgm^2)	172,455*
Bus front tire stiffness (N/rad)	152,000*
Bus rear tire stiffness (N/rad)	310,020*
Bus wheel base (m)	7.5
Bus length (m)	12.4
Bus width (m)	2.57
Bus height (m)	3.4
Distance: front sensor bar to front axle (m)	1.1
Distance: front sensor bar to rear sensor bar (m)	5.4
Distance: CG to front axle (m)	4.79*
Steering ratio	17.1

Table 2.1.5 New Flyer 60 foot Diesel Articulated Bus Parameters

Description	Values
Mass of empty bus (kg)	18,597
Mass of front section (kg)	8,210*
Mass of articulated section (kg)	10,387*
Front section yaw moment of inertia (kgm ²)	58,344*
Articulated section yaw moment of inertia (kgm ²)	93,221*
Front section front tire stiffness (N/rad)	143,000*
Front section rear tire stiffness (N/rad)	250,452*
Articulated section tire stiffness (N/rad)	285,300*
Bus length (m)	18.5
Bus width (m)	2.57
Bus height (m)	2.8
Front section wheel base (m)	5.8
Distance: front sensor bar to front axle (m)	0.38
Distance: front sensor bar to rear sensor bar (m)	5.16
Distance: front axle to articulation joint (m)	8.1
Distance: front section CG to front axle (m)	2.28*
Distance: articulation joint to articulated section axle (m)	5.69
Distance: articulated section CG to articulation joint (m)	5.2*
Steering ratio	17.1

Note: data with * is estimated value

2.2 Freightliner Century Trucks

The acquisition of the three heavy-duty tractor-trailer trucks for the demonstration was based on an open procurement issued by Caltrans, relying on technical specifications provided by PATH. The intent of this procurement was to acquire heavy-duty trucks with the highest level of in-vehicle technology available, in order to minimize the work involved in modifying them for demonstration purposes, and with maximum-size cabs in order to accommodate the maximum number of demonstration riders. This meant, for example, that the trucks should have as many functions as possible integrated on the CAN bus, and should be equipped with Eaton-Vorad EVT-300 radar systems for collision warning and adaptive cruise control. The most favorable bid submitted in response to the Caltrans procurement was from Freightliner, for their Century-Class trucks, so these were selected and purchased.

It was deemed essential to equip the trucks with automatic transmissions in order to avoid the need to implement a mechanical robot to shift gears. The highest-capacity automatic transmission that was available (from Allison) imposed the governing power limit on the trucks, and required that they be equipped with engines that could provide only 435 Hp. Although higher-power engines were available, they could not be used with the automatic transmission.

The Freightliner Century Class vehicles had engines manufactured by Cummins, and transmission provided by Allison as mentioned previously. Specific details about the transmission and engine are in Table 2.2.1 below:

Table 2.2.1. Freightliner Century Class Specifications

	Freightliner Century Class
Engine Mfr.	Cummins
Engine Model	N14-435EI
HP	435
RPM	2100
Transmission Mfr.	Allison
Transmission Model	HD4060
Vehicle Weight	17,280 lbs

2.2.1 Longitudinal Dynamics of the Trucks

In contrast to passenger cars, longitudinal control of a heavy truck has the following characteristics:

(a) Mass dominant: Any minor change of a mass-related factor (such as road grade or acceleration demand) causes large variations in torque demand. This, in turn, makes the control system performance highly sensitive to the assumed vehicle mass, putting a premium on real-time mass estimation;

(b) Low power/mass ratio: It is easy to cause actuator saturation, which eventually leads to the loss of controllability. This ends up with the situation that there is no power available for speed and distance control. In this case, both the closed-loop stability and string stability may be destroyed;

(c) Large actuator delays: These delays come from, but are not restricted to, engine indicated torque production, the transmission torque converter, pneumatic brake and transmission retarder. These delays are the main obstacles to string stability;

(d) Disturbances during gear shifting are more prominent if the vehicle is accelerating, which is due to the large vehicle inertia and zero engine torque passed to wheels during the shift.

These factors determine naturally that short-inter vehicle distance following for heavy trucks is more difficult than that for passenger cars. To relieve this difficulty, it is necessary to analyze the main factors in vehicle dynamic modeling for control.

Vehicle modeling for longitudinal control includes the dynamics of the following components: vehicle body, engine, brake, transmission and tires. The overall dynamic system is intrinsically high dimensional and highly nonlinear. Previous research at PATH has shown that control based on a simplified linear model cannot achieve such a goal. This is understandable because, from the real-time control design point of view, the following factors degrade control performance: model mismatch, external disturbances, measurement noise and time delays.

To maximally achieve robust performance of a controller, one needs to reduce all those effects. External disturbances can be compensated for by the robustness of a properly designed controller. Measurement noise can be rejected by proper filtering and data fusion techniques, which are intended to achieve the smallest estimation error, the strongest noise rejection property and the least time delay. It is important to obtain a good model of the system to be controlled, which should be simple enough to be used for control design, but complicated enough to capture the intrinsic vehicle dynamics. This necessarily implies a nonlinear model. The model to be used here is based on the following assumptions:

1. Turbocharger dynamics are separated from the engine dynamics under the assumption that the booster pressure (manifold pressure between turbocharger and the cylinders) is measured and explicitly accounted for.

2. A static nonlinear engine mapping, which defines the functional relationship between engine speed, booster pressure, fuel rate and indicated torque is used.

3. For the pneumatic brake system, the built-in Electronic Braking System (EBS) is used as the lower level brake control actuator. For modeling this system, two delays are involved: pure time delays from actuation and release, and a first-order lag representation of pneumatic system dynamics. A variable-structure second-order brake model is established to count for the difference between activation and release delays.

4. Engine control: The built-in engine speed and torque P-I control of the engine controller is used as the lower-level (inner loop) actuator.

5. Tire slip directly reflects vehicle longitudinal dynamics and vehicle moving distance measurement. A moving average tire slip estimation proposed in [10] is adopted.

With these separations and simplifications, the second-order vehicle dynamics is globally feedback linearizable. In vehicle dynamics, one of the major components is the *total resistance*, which can be divided into three parts: aerodynamic drag force, rolling resistance and engine braking force. The drag force term depends on the drag coefficient, which can be determined by wind tunnel experiments. Rolling resistance depends on vehicle mass and weakly on speed. The engine braking force is quite different from that of passenger cars, and has not yet been reported in previous work.

Compression (“Jake”) braking [11,12] is a special feature of modern turbocharged diesel engines, which does not exist in passenger cars. For a 6 cylinder engine, one can switch on 2, 4 and 6 cylinders respectively to produce retarding torque to the vehicle. The maximum retarding power can be as high as the engine active horsepower [12]. The advantages of using the Jake brake is its faster response than pneumatic brakes, which is particularly desirable for longitudinal control. The disadvantage is that vehicle-following control needs a continuous spectrum of braking torque levels at all vehicle speeds. Obviously, the Jake brake alone does not have such a feature, so a combined braking system, including the Jake brake, pneumatic brake and transmission retarder will be used for future control design.

For reader’s convenience, all notation used in describing the truck longitudinal control will be listed at the start of the section.

2.2.1.1 Truck Hardware and Modeling

2.2.1.1.1 Notation:

M – vehicle mass

ω – engine speed

ω_{idle} – engine idle speed

ω_p – torque converter pump speed, $\omega_p = \omega$

ω_t – torque converter turbine speed

ω_{tb} – turbocharger speed

ω_{tr} – transmission output speed

ω_{dr1} – propeller-shaft speed including front part of final gear

ω_{dr2} – drive-shaft speed including front rear of final gear, final drive end

ω_{dr} – drive-line speed (considered as lump sum), final drive end

ω_w – wheel angular speed
 v – vehicle wheel speed (longitudinal) is used for all control design
 a – acceleration
 α – throttle angle
 α_f – fueling rate
 I_e – engine inertia
 I_{tr} – transmission inertia
 I_{dr1} – drive line inertia (before final gear)
 I_{dr2} – drive line inertia (after final gear)
 I_{dr} – lump sum drive line inertia ($I_{dr} = I_{dr1} + I_{dr2}$)
 I_w – wheel inertia
 P_m – intake manifold pressure or *turbocharger booster pressure*
 T_d – drive-line torque loss
 T_{ind} – engine indicated torque
 T_{net} – engine net output torque
 T_p – torque converter pump torque, $T_p = T_{net}$
 T_t – torque converter turbine torque
 T_b – service brake torque
 T_{jake} – engine brake torque
 T_{tr} – transmission output torque
 T_{dr1} – final gear input torque (or equivalently propeller shaft final end torque)
 T_{dr2} – final gear output torque
 T_w – engine torque passed onto wheel
 T_{rtd} – transmission retarder torque
 T_{fric} – engine friction torque
 T_{e_brk} – engine braking effect torque when $T_{net} < T_{net_des}$
 F_a – aerodynamic resistance force
 F_r – rolling resistance force
 F_f – friction force
 $F_{eng-brk}$ – engine braking force transmitted to wheels when throttle is released
 $F_{total} = F_r + F_{eng-brk}$
 F_ω – engine brake force when clutch is engaged and fueling is in idle
 h_r – effective tire radius
 θ – road grade, $\theta > 0$ means ascending
 r_g – transmission gear ratio
 r_d – final-drive gear ratio

R_g – gear ratio $R_g = r_g r_d, \omega_t = \frac{v R_g}{h_r}$

V_a – relative speed to the pneumatic in the longitudinal direction

ρ_{air} – air density

2.2.1.1.2 Overall System Structure and Vehicle Dynamics Modeling

For precise longitudinal control of heavy vehicles, we need more detailed models than those that are normally described in the vehicle dynamics literature [13-15]. Engine power transmitted through the powertrain is characterized by two states: engine speed and net output torque. Engine speed is closely related to wheel speed, while output torque is related to wheel acceleration. To establish a vehicle dynamics equation, one can choose wheel speed, engine speed, or even a speed of any powertrain element between them, as the fundamental state variable. Here we use wheel speed for the convenience of implementation. The power flow chart is depicted as in Fig. 2.2.1.

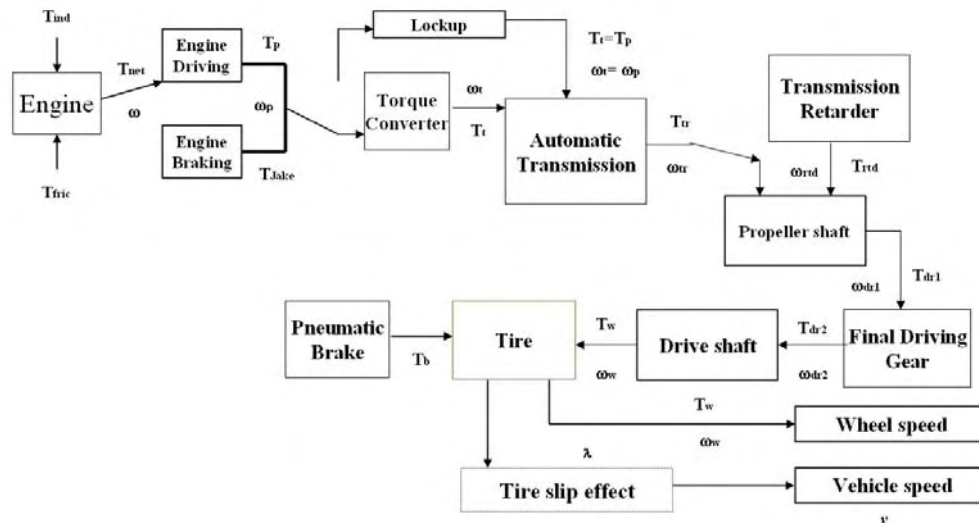


Fig. 2.2.1 Power Flow Chart

1. Engine Driving Mode

The engine net output torque is represented by:

$$\begin{aligned} T_{net} &= I_e \dot{\omega} + T_p \\ T_p &= T_t \end{aligned} \quad (2.1)$$

where it is assumed that the clutch is locked up. If the torque converter is on, T_p should be calculated from a torque converter model. A possible torque converter model is the popularly recognized static Kotwicky model [16]. Following the power flow in Fig. 1 and notice the following relationships

$$\begin{aligned} \omega_w &= \omega_{dr2} = \frac{v}{h_r} \\ \omega_{tr} &= \omega_{dr1} = \frac{v}{h_r} r_d \\ \omega &= \omega_t = \frac{v}{h_r} r_d r_g \end{aligned}$$

Longitudinal vehicle dynamics is obtained as:

$$\begin{aligned} v &= \frac{r_d r_g T_{net} - (r_d T_{rd} + T_b + F_a h_r + F_{total} h_r + M g h_r \sin \theta)}{\bar{I}} \\ \bar{I} &= r_d^2 r_g^2 \frac{I_e}{h_r} + r_d^2 \frac{I_{tr}}{h_r} + r_d^2 \frac{I_{dr1}}{h_r} + \frac{I_{dr2}}{h_r} + \frac{I_w}{h_r} + M h_r \end{aligned} \quad (2.2)$$

where

$$F_a = 0.5 C_a \rho_{air} A V_a^2$$

T_b and F_{total} will be modeled separately. The modeling of aerodynamic factors for trucks is described in [17].

2. Other Driving Modes

1. *Engine Braking Mode*: $T_{net} = -T_{jake}$ ($Jake = 0, 2, 4, 6$) is used in (2.2)

2. *Transmission Retarder Mode*: $T_{net} = 0$ in (2.2)

3. Vehicle Mass Preliminary Estimation:

Vehicle mass could be estimated from (2.2)

$$\begin{aligned} M &= r_d r_g T_{net} - \frac{1}{v h_r + g h_r \sin \theta} \left[(r_d r_g)^2 I_e \frac{v}{h_r} + r_d^2 I_{tr} \frac{v}{h_r} \right. \\ &\quad \left. + r_d^2 I_{dr1} \frac{v}{h_r} + I_{dr2} \frac{v}{h_r} + I_w \frac{v}{h_r} + T_b + F_a h_r + F_{total} h_r \right] \end{aligned}$$

but this is a singular expression if

$$\dot{v} + g \sin \theta = 0 \quad (2.3)$$

Physically, this implies that to estimate the vehicle mass, the vehicle acceleration must be non-zero, which may be generated by engine torque and/or road grade.

It is noted that this formula only provides a rough mass estimation because all the measurements contain noise. Besides, at low speed, the torque converter plays an important part, which aggravates the situation. For best performance, vehicle mass estimation is needed, and it will probably be advisable to incorporate this directly into the data available on the vehicle's internal data bus when the truck is weighed at the start of each trip.

2.2.1.1.3 Engine Modeling

Diesel and turbocharged diesel engine modeling and control have been conducted in [18, 19] and diesel engines with turbocharger and EGR (Exhaust Gas Recirculation) are considered in [20]. If booster pressure is measured and taken as input to the engine, the dynamic relationship between the engine and turbocharger can be decoupled. This simplification is sufficient for vehicle control.

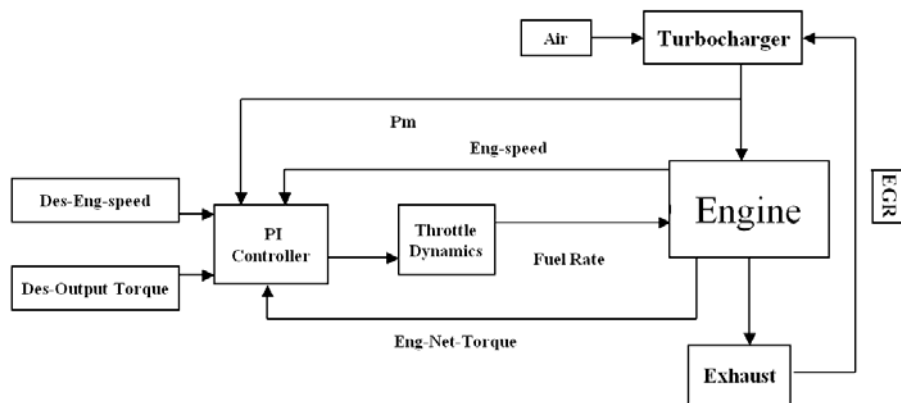


Fig. 2.2.2 Cummins N14-435 Turbocharged Diesel Engine Internal Control

Strictly speaking, engine output torque is produced by discontinuous explosion strokes. However, for longitudinal control purpose, a continuous mean-torque-value output is assumed. A static engine mapping can be used to define the relationship among engine speed, net output torque, fuel rate and booster pressure $(\omega, T_{net}, \alpha_f, P_m)$ [13,16]. The input-to-torque production delay, which depends on engine speed with other factors such as injection time, engine temperature, etc., is implicitly captured by the mapping. This mapping can be used to find (by interpolating) the desired fuel rate from the desired engine speed and torque. If engine fuel rate control is accessible, this approach implements the engine control [13, 21, 22]. However, direct access to fuel rate control is prohibited on modern trucks due to their internal engine control structure. Thus, the engine control approach had to be reconsidered.

The Cummins N14_435 turbocharged diesel engine for the Freightliner Century Truck has internal electronic control (Fig. 2.2.2, *Engine Control Module -ECM*), which takes input from the driver's pedal deflection and interprets it as either *speed control command* or *torque control command*. These commands are passed through the J-1939 data bus. The output of the controller is fuel rate (as input) to the engine. The principle of the control strategy is based on obtaining desired torque at a certain speed, while optimizing with respect to, among others, emissions, fueling time, output torque and fuel consumption. To practically use it, a simulator is necessary to interface between the vehicle longitudinal control computer and the engine ECM. Because of this structure, one cannot use engine mapping to generate fuel rate commands. Rather, the desired net output torque and desired engine speed are directly fed into the ECM. The internal controller here acts as an inverted engine map, and its performance will therefore significantly affect the longitudinal controller.

1. Engine Braking Effect

Engine braking effect comes from the mismatch between engine speed and wheel speed while gear is engaged and either the torque converter is on or the transmission clutch is locked-up. It is a retarding effect in addition to the vehicle's rolling resistance.

$$F_{total} = \begin{cases} F_r, & \text{engine driving} \\ F_{eng-brk} + F_r, & \text{throttle released} \end{cases}$$

$$F_{eng-brk} = \frac{C_{eng-brk}(\omega - \omega_{idle})}{R_g}$$

$$R_g = r_g r_d$$

This effect is taken into account whenever engine fueling is relaxed and a gear is engaged. Suppose vehicle acceleration a and road grade θ are measured/estimated. These parameters can be obtained as follows:

(1) When vehicle speed reaches a specified value v_{\max} , relax fueling to idle and keep gear engaged to leave the vehicle free rolling. From

$$F_{total} = Ma + Mg \sin \theta$$

one obtains F_{total} ;

(2) Repeating this procedure but setting gear to neutral, one obtains

$$F_r = Ma + Mg \sin \theta$$

Then

$$F_{eng-brk} = F_{total} - F_r$$

A test result using the Freightliner Century truck is shown in Fig. 2.2.3.

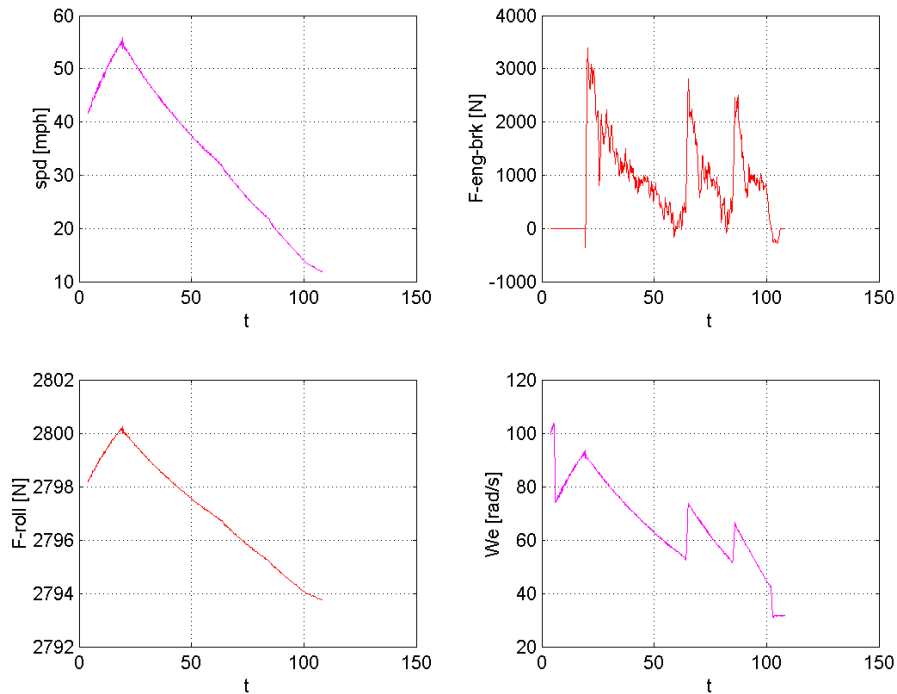


Fig. 2.2.3 Engine braking effect when throttle is released.

In the test case shown in Figure 2.2.3, the fuel pedal was released at about $t = 20$, when the speed reached 55 mph, and the truck was allowed to coast down. It shifted from fifth to second gear in the process of coasting down, and the discontinuities associated with the gear shifts are evident in the lower right plot, showing engine speed and the upper right plot, showing engine braking force. The rolling resistance force is shown in the lower left plot.

2.2.1.1.4 Braking System Modeling

2.2.1.1.4.1 Engine Compression Brake

The compression (Jake) brake system provides discrete and limited braking torque on driving wheels with fast response. Its retarding force mainly depends on engine speed. The *engine braking effect* proposed in [11] is caused by the mismatch between engine speed and wheel speed when fueling is relaxed and the drive-line is engaged. It is recognized that this effect can be considered as a special case of Jake Brake when no valve is open. From this point of view, the braking torque on driving wheels provided by the engine can be modeled in a variable structure model as:

$$T_{jk} = \begin{cases} T_{jk_0}, & 0 \text{ cylinder on} \\ T_{jk_2}, & 2 \text{ cylinder on} \\ T_{jk_4}, & 4 \text{ cylinder on} \\ T_{jk_6}, & 6 \text{ cylinder on} \end{cases}$$
$$T_{jk_0} = T_{eng-brk} = \frac{C_{eng-brk}(\omega - \omega_{idle})}{R_g}$$
$$T_{jk_i} = \frac{C_{jk_i}(\omega - \omega_{idle})}{R_g}, \omega \geq 800[rpm]$$
$$R_g = r_g r_d$$
$$i=2,4,6$$

which is a generalization from the engine braking effect in [11]. The general principle of the compression brake is illustrated in Figure 2.2.4, and data from testing a Freightliner Century truck with two, four and six cylinders of compression braking are shown in Figures 2.2.5 – 2.2.7.

With a Jacobs Engine Brake™

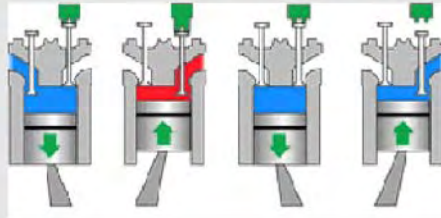


Fig.2.2.4 Engine Brake Principle

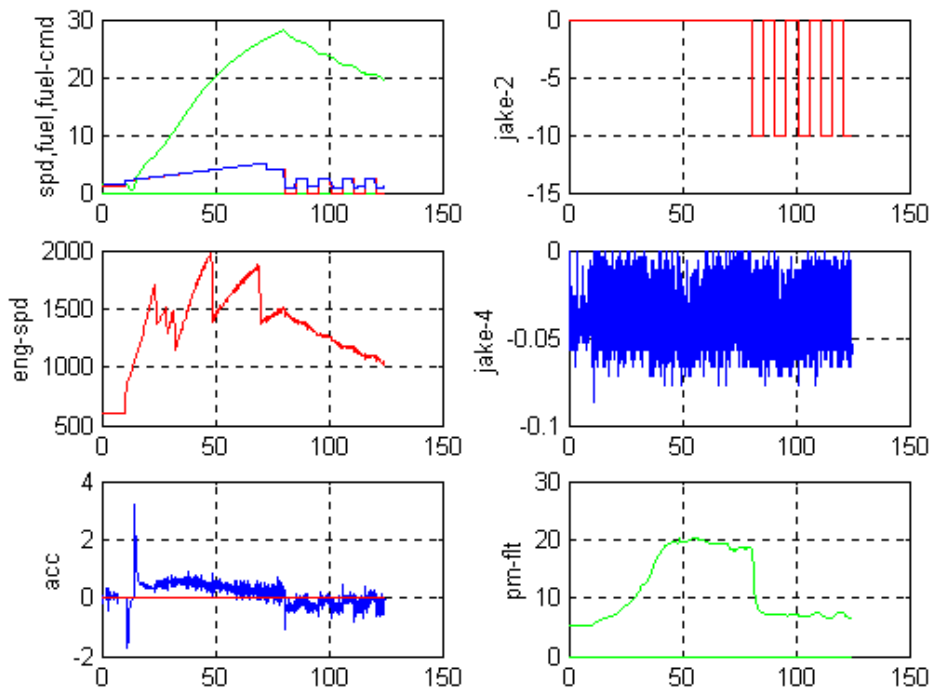


Fig. 2.2.5 Effect of 2 Cylinders of Compression Braking ON

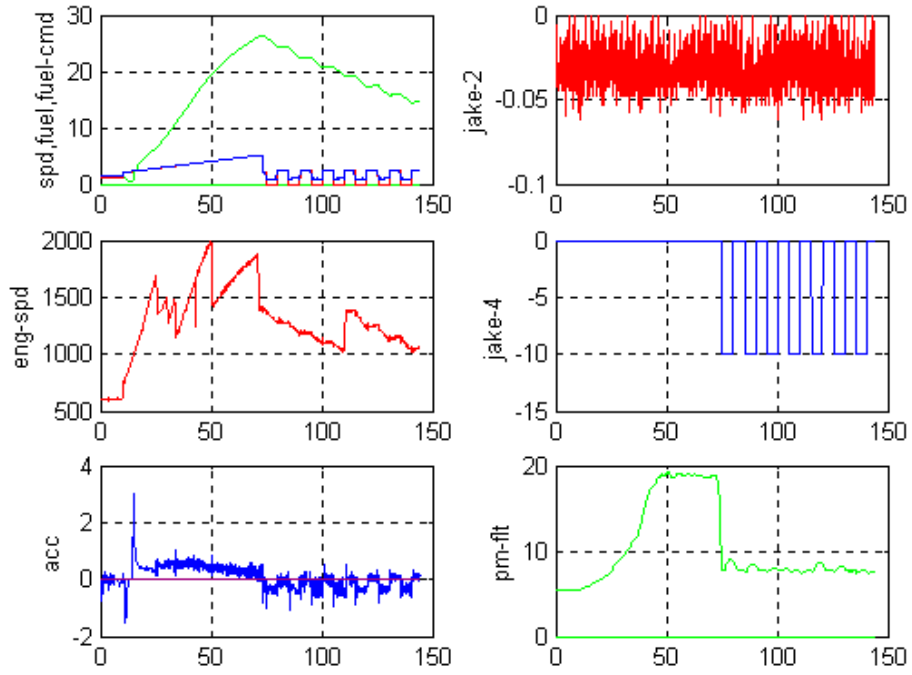


Fig.2.2.6 Effect of 4 Cylinders of Compression Braking ON

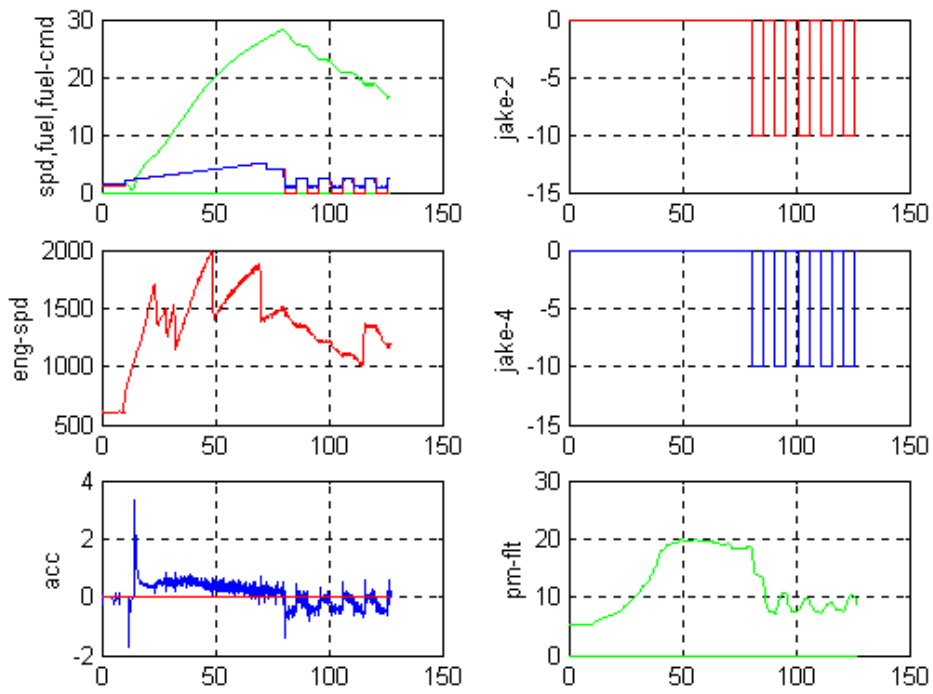


Fig. 2.2.7 Effect of 6 Cylinders of Compression Braking ON

In each of these figures, the upper left plot shows the truck speed in m/s and the commanded and actual engine fueling rates. The middle and lower left plots show engine speed in rpm and measured vehicle acceleration. The vehicle acceleration data are too noisy to use for quantitative characterization of system performance, but they serve to show the choppy ride quality associated with the application of the compression brake during the deceleration. The upper and middle right plots show the application of the two and four-cylinder increments of compression braking, with “on” corresponding to a -10 value on the plot, while the lower right plots show the engine manifold pressure.

The modeling of the effect of the compression braking for purposes of control system design is based on use of the engine speed and vehicle speed data from these tests. The slope of the truck speed curve represents the deceleration of the truck, and after the rolling resistance effects are subtracted out, the contribution of the compression brake to truck deceleration can be estimated.

2.2.1.1.4.2 Transmission Retarder

The transmission retarder can provide continuous braking torque on the driving wheels, but with slow response and limited braking torque [12,23]. A model of transmission retarder behavior is proposed as follows:

$$\dot{T}_{tr} = \begin{cases} \frac{1}{\tau_{tr}} \left[-T_{tr} + \zeta(V_{rtd}(t - \kappa_{tr}), \omega_{dr}) \right], & t < \tau_{tr0} \\ \frac{1}{\tau_{tr}} \left[-T_{tr} + \zeta(V_{rtd}(t), \omega_{dr}) \right], & t \geq \tau_{tr0} \end{cases} \quad (1.6)$$

$$\kappa_{tr} = \tau_{tr0} \left[1.0 - \frac{t}{\tau_{tr0}} \right] = \tau_{tr0} - t$$

τ_{tr0} – pure time delay parameter $\square 1.5[s]$

V_{rtd} – voltage or transmission retarder pedal deflection

$\zeta(V_{rtd}, \omega_{dr})$ – a nonlinear function

$\kappa_{tr} \approx 1.5[s]$

It is noted that the pure time delay parameter κ_{tr} is time varying and it disappears after 1.5[s]. This time period is used for filling up the liquid into the retarder chamber.

The applied torque at the wheels is calculated as:

$$T_{tr}^{(w)} = (T_{tr} - \omega_{dr1} I_{dr1}) r_d - \omega_{dr2} I_{dr2}$$

2.2.1.1.4.3 Euro-EBS: Electronic Braking System (EBS)

The Euro-EBS braking system is not used in the U.S., and had to be acquired by special arrangement with the supplier WABCO-Meritor, so it should really be considered as part of the modifications that PATH had to make on the trucks in order to make them capable of automated driving.

The features of Euro-EBS are described below. The critical part relevant to automatic longitudinal control of the trucks is the Central Module:

- Function: To control and monitor the EBS
- Input:
 1. signal from brake signal transmitter (vehicle desired retardation)
 2. Wheel speed sensor
- Output: index pressure values → front axle, rear axle and trailer control valve
- Front axle: Relay valve is used to compensate for any difference of applied and measured pressure values
- Trailer Pressure Control: similar

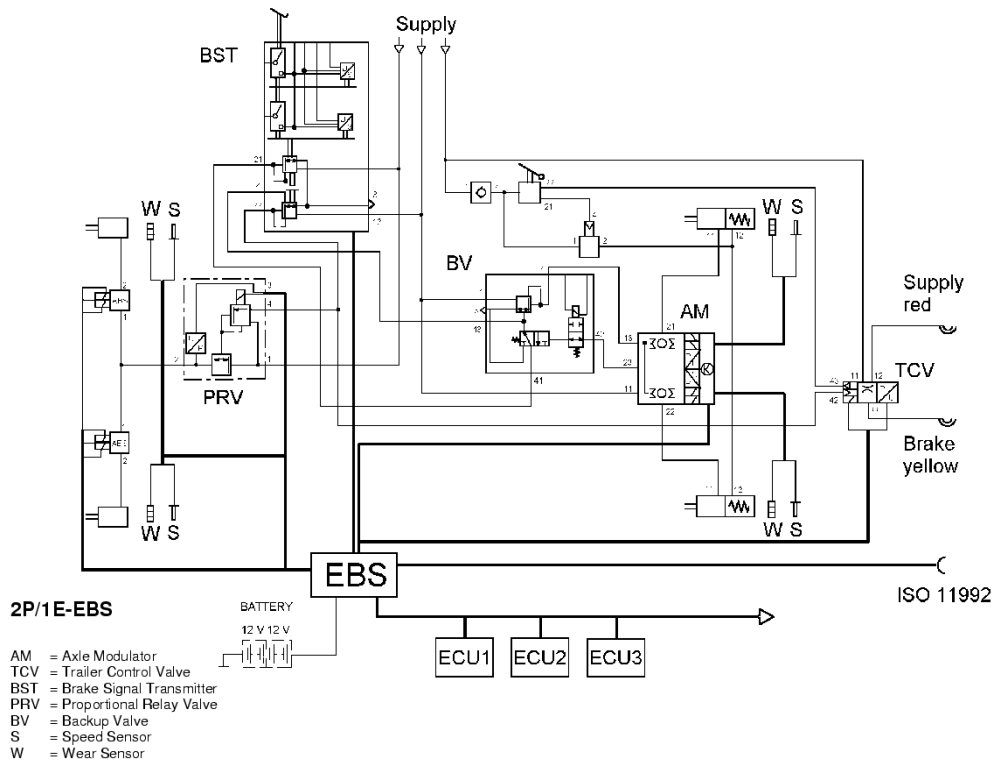


Fig. 2.2.8 EBS Control System – General Picture

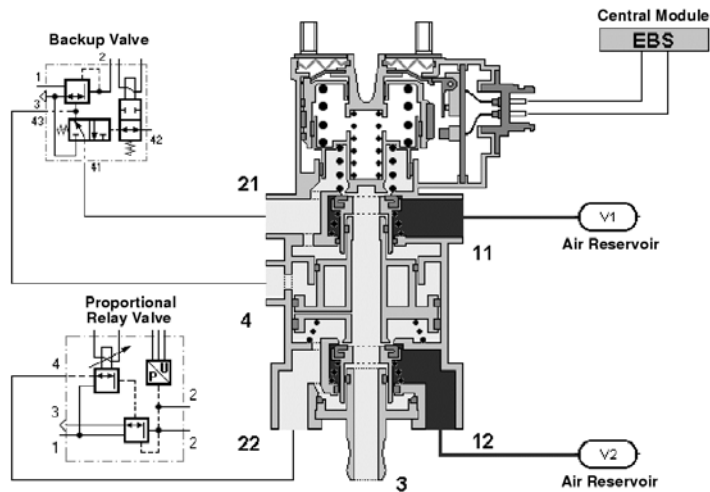


Fig. 2.2.9 Brake Signal Transmitter

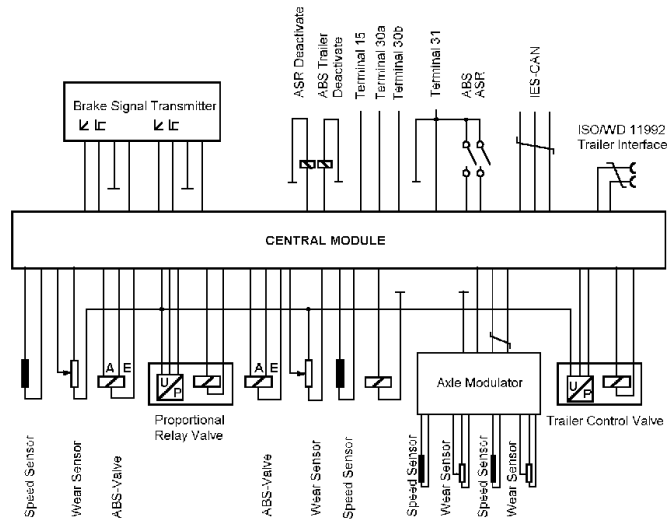


Fig. 2.2.10 Central Module

Control Implementation – Actuator: Brake (EBS)

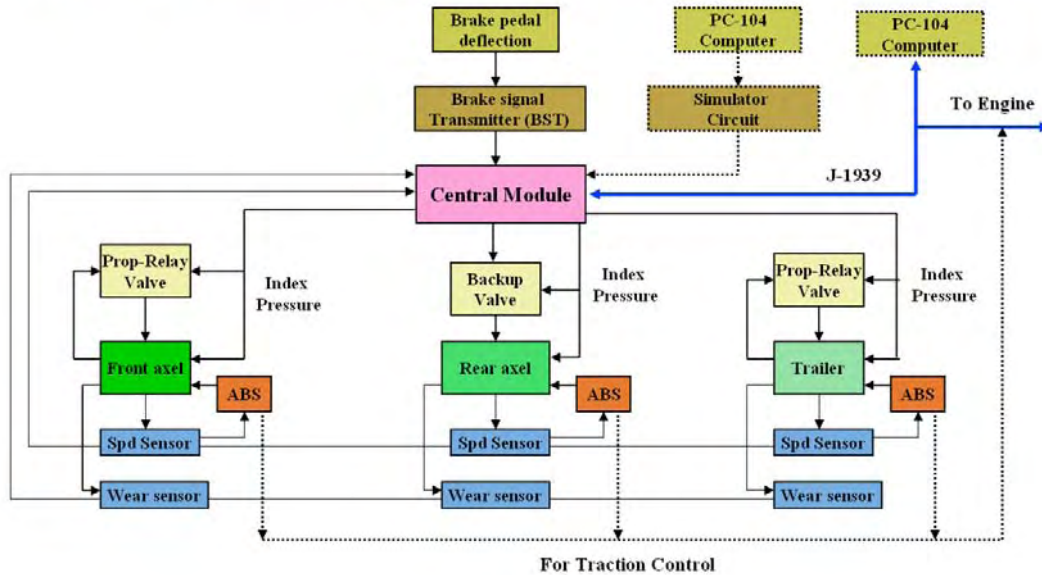


Fig. 2.2.11 Control System for EBS

Freightliner Century trucks have built-in Electronic Braking System (EBS). The main characteristics of this system are:

- Control on each individual wheel
 - Incompatibility in braking torque between tractor and trailer is compensated
 - Fault detection is available for brake components as they are managed in real-time
 - Shorter response and pressure buildup time
 - Optimized stability and traction
 - Electro-pneumatic and pure pneumatic (parallel) redundancies for all brake circuits
- Electro-pneumatic circuit is used for default and pneumatic circuit is used when the former fails.

To fully exploit the advantages of EBS, the internal control system will be used. To achieve this, a simulator circuit was developed to generate modulated pulse-width signals to replace the Brake Signal Generator in EBS to interface with a computer.

From the brake structure, the functional relationship from applied brake pressure to brake torque has the following characteristics:

$$T_b = r_d \sigma_d P_d$$

$$P_{app} = \sigma_0 V_b$$

$$P_{app} \leq P_{res}$$

P_d – brake pressure force (push-rod force) at brake drum or output pressure of diaphragm

r_d – brake drum radius

σ_d – friction coefficient between brake drum and brake pad

P_{app} – applied brake pressure

A_d – area of diaphragm

P_{res} – air pressure at reservoir

V_b – applied brake voltage (equivalent to brake pedal deflection)

σ_0 – constant coefficient determined by manufacturer

For all the parameters, only σ_d needs to be identified, while the other parameters can be measured or estimated from data.

In brake modeling, there is a pure time delay, which is observed in both activation and release, combined with a dynamic lag. Combining them both, practical brake pressure at the wheel is obtained as:

$$\dot{P}_d = \begin{cases} \frac{1}{\tau_{ba}} (P_d + A_d P_{app}(t - \tau_a)), & \text{activation} \\ \frac{1}{\tau_{br}} (P_d + A_d P_{app}(t - \tau_r)), & \text{release} \end{cases}$$

where:

τ_a – activation time delay (pure time delay)

τ_r – release time delay (pure time delay)

τ_{ba} – dynamic time constant for activation

τ_{br} – dynamic time constant for release

2.2.1.1.5 Open Loop Response of Powertrain

The following figures show the open-loop response of the powertrain of a Freightliner Century truck pulling an empty trailer. They give the qualitative and quantitative relationships among the following parameters:

Figure 2.2.12 shows the transmission input shaft speed and output shaft speed, which in turn give the dynamical relationship between input and output of the torque converter and gear shifting characteristics.

Figure 2.2.13 shows the engine speed, together with fuel rate and percentage torque generated, together with the estimated engine friction torque.

Finally, Figure 2.2.14 shows fuel rate, pedal position and wheel speed, indicating the response of vehicle speed to fuel rate.

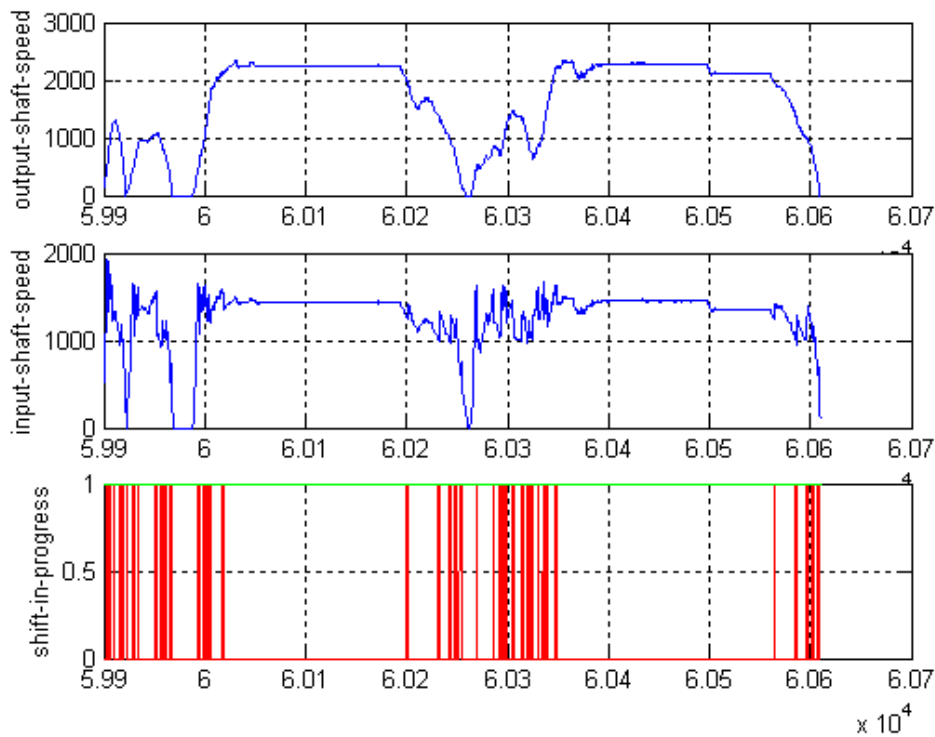


Fig. 2.2.12 Input/output shaft speed of transmission [rpm]

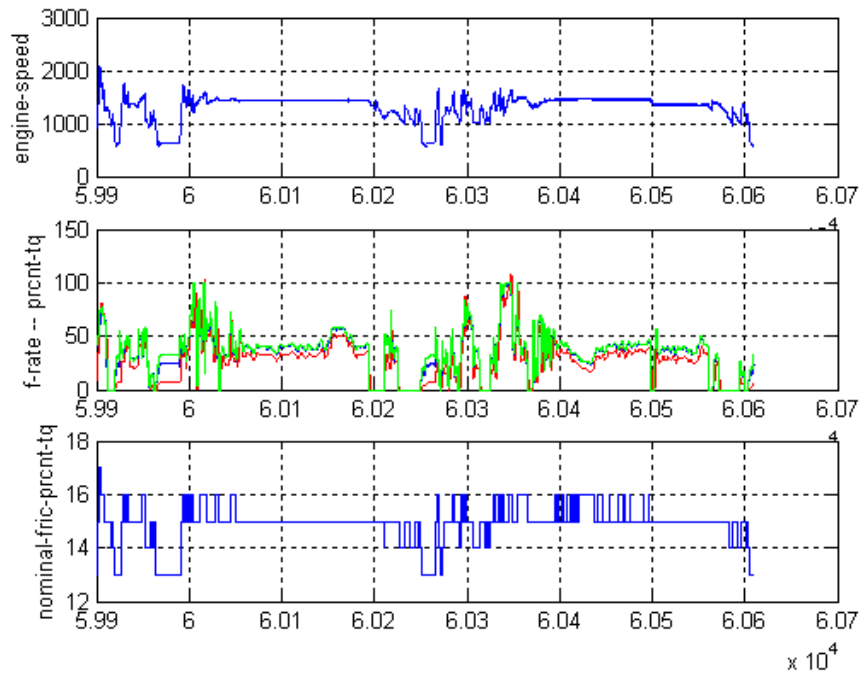


Fig. 2.2.13 Engine speed [rpm]; Scaled fuel rate and percentage torque estimated; Nominal friction percentage torque

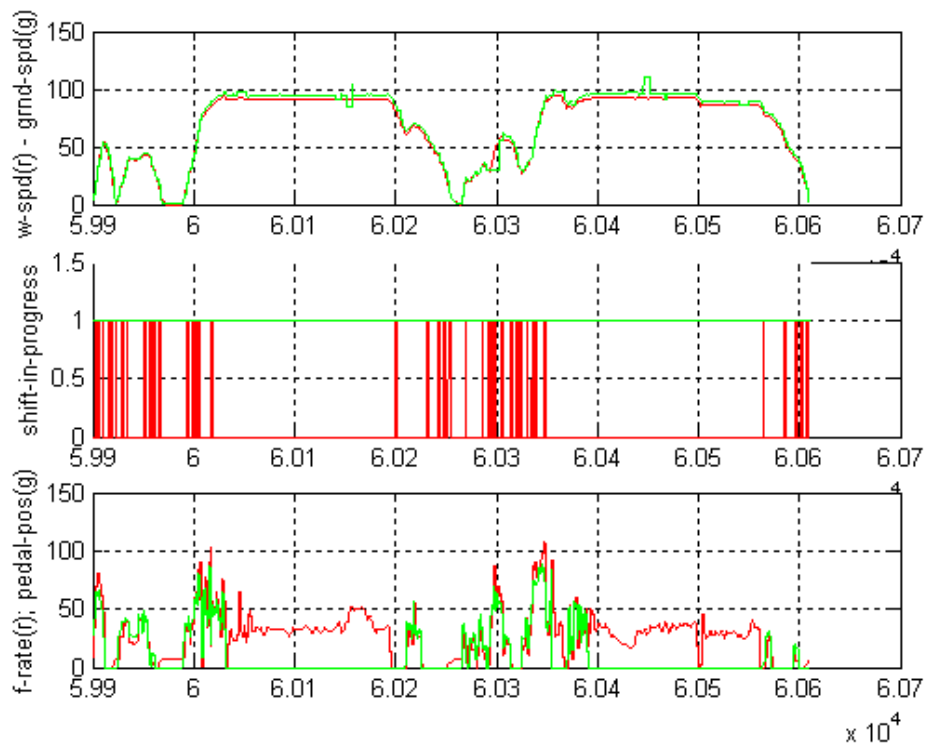


Fig. 2.2.14 Wheel speed and ground speed from GPS; fuel rate and pedal position

2.2.2 Lateral Dynamics of the Trucks

2.2.2.1 Mathematical Model

Mathematical models were developed for the lateral dynamics of the trucks to support the development of the lateral controller. The dynamical equations representing lateral dynamics of a tractor semi-trailer combination are shown in Eq. (2.2.2.1) where the state variables of the model are described relative to a “road coordinate system” centered at a point on the road centerline such that it is closest to the tractor center of gravity. The axes of this coordinate system are aligned along the tangent, normal and binormal to the road centerline. Under simplifying assumptions such as small relative yaw of the tractor with respect to the road, small articulation angle and small steering angle, the dynamics of the tractor semi-trailer, in the form of $\dot{\bar{x}} = A\bar{x} + B\delta + \bar{d}$, are given by [26, 27] as:

$$\frac{d}{dt} \begin{bmatrix} q_r \\ \dot{q}_r \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} q_r \\ \dot{q}_r \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}F \end{bmatrix} \delta + \begin{bmatrix} 0 \\ E_1 \end{bmatrix} \dot{\varepsilon}_d + \begin{bmatrix} 0 \\ E_2 \end{bmatrix} \ddot{\varepsilon}_d \quad (2.2.2.1)$$

where $q_r = [y_r, \varepsilon_r, \varepsilon_f]^T$ and \dot{q}_r are the state variables of the system. y_r is the distance of the tractor center of gravity from the road centerline, ε_r is the yaw angle between the tractor longitudinal axis and the axis of the road coordinate system along the tangent to road centerline, and ε_f is the articulation angle (see Figure 2.1.10). The input to the model is δ , the steering angle of the front wheel. $\dot{\varepsilon}_d = v\rho$ is the angular velocity of the road coordinate system where v is the longitudinal velocity of the vehicle and ρ is the road curvature. Matrices appearing in the system equation are given as those for the articulated bus in Eqs. (2.1.11) to (2.1.16). Similarly, m_1 and m_2 are the mass of the tractor and the mass of semi-trailer respectively. I_{z1} and I_{z2} are the moments of inertia of the tractor and the semi-trailer respectively. $2C_{af}$, $2C_{ar}$ and $2C_{at}$ are the combined cornering stiffness coefficients of the front tires of the tractor, rear tires of the tractor and tires of the trailer respectively. Geometric parameters associated with the tractor semi-trailer model are also illustrated in Figure 2.1.10.

The parameters for the models for a Freightliner Century tractor and a box trailer and a lowboy trailer are shown in Table 2.2.2. Those parameters were obtained by tuning the parameters to fit the experimental frequency responses as in Figure 2.2.15a – Figure 2.2.15c. Due to delays in the truck instrumentations and the high noise in the inertial sensors measurements during the initial data collection phase, the truck parameters were generated using available data with certain engineering guesses. The values in Table 2.2.2 with * indicate there are some uncertainties in these numbers. The controller designs were thus focused on robustness that tolerates significant parameter uncertainties.

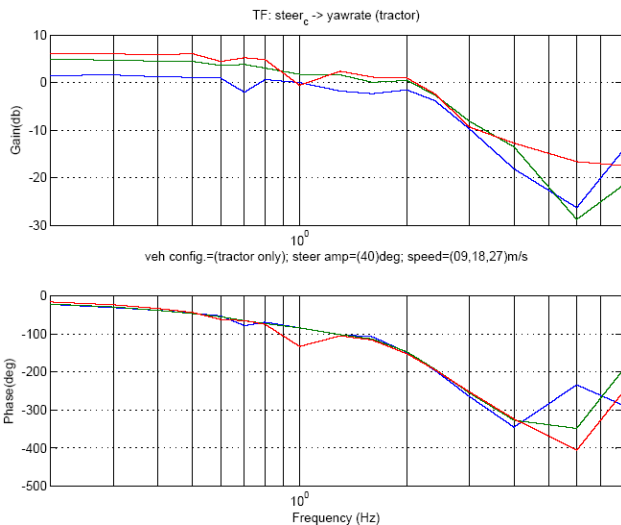


Figure 2.2.15a: Frequency Sweep: Steer command to Tractor yaw rate (Tractor only, $v = 9,18,27$ m/s)

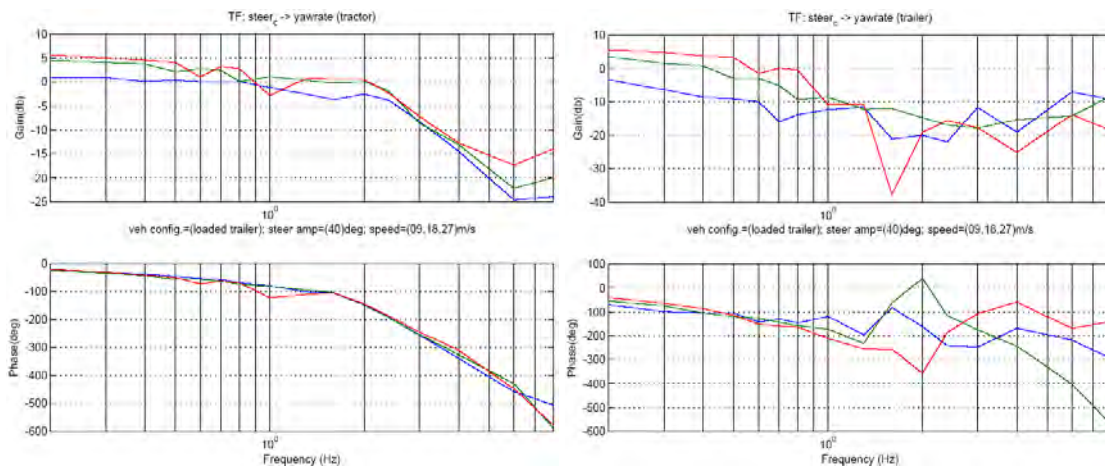


Figure 2.2.15b: Frequency Sweep: Steer command to Tractor (left) and Trailer (right) yaw rate (Tractor with fully loaded Lowboy trailer, $v = 9,18,27$ m/s)

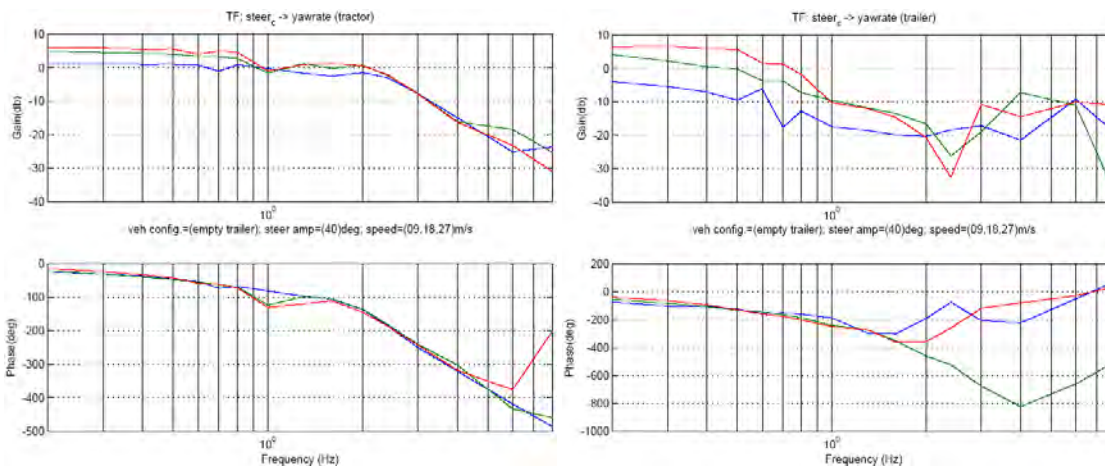


Figure 2.2.15c: Frequency Sweep: Steer command to Tractor (left) and Trailer (right) yaw rate (Tractor with empty Lowboy trailer, $v = 9,18,27$ m/s)

Table 2.2.2 Model parameters for the tractor semi-trailer

Description	Value
Mass of Tractor (kg)	8,400
Mass of empty lowboy trailer (kg)	6,465
Mass of empty box trailer (kg)	5,455
Mass of trailer load (kg)	8,100*
Tractor yaw moment of inertia (kgm ²)	46,068*
Trailer yaw moment of inertia (kgm ²)	162,400*
Tractor front tire stiffness (N/rad)	180,000*
Tractor rear tire stiffness (N/rad)	415,000*
Trailer tire cornering stiffness (N/rad)	332,000*
Distance: tractor front axle to CG (m)	1.9*
Distance: tractor rear axle to CG (m)	3.92*
Distance: trailer axle to king pin (m)	10.0
Distance: king pin to tractor CG (m)	3.30*
Distance: trailer CG to king pin (m)	3.5*
Distance: sensor to tractor CG (m)	1.8*
Tractor length (m)	7.5
Steering ratio	18

2.2.2.2 Tractor Semi-Trailer Model Sensitivity Comparisons

In order to understand the dynamics of the tractor semi-trailer lateral responses, the transfer functions from steering angle to tractor lateral acceleration at CG ($V_r(s)$), to tractor yaw rate ($V_\dot{\epsilon}(s)$), and to articulation angle ($V_f(s)$) were obtained and analyzed:

$$\begin{bmatrix} y_r(s) \\ \dot{\epsilon}_r \\ \epsilon_f \end{bmatrix} = \begin{bmatrix} V_r(s)/s^2 \\ V_\dot{\epsilon}(s) \\ V_f(s) \end{bmatrix} \delta_f(s) = \left(\begin{bmatrix} 0 & 0 & 0 & 1/s & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} (sI - A)^{-1} B \right) \delta_f(s). \quad (2.2.2.2)$$

The following is the summary of the observations from the comparisons [28]:

- The most sensitive parameters with respect to the tractor semi-trailer dynamics are the tire stiffness and the vehicle speed. The “worst” dynamic appears when low tire stiffness is combined with high speeds.
- The tractor dynamics are not very sensitive to the trailer loading conditions (as shown in Figure 2.2.16). However, the degree of the articulation coupling to the lateral dynamics increases as the CG of the trailer load moves significantly behind the trailer center; this effect is more noticeable with heavy load and at higher speed.
- Unlike tires of the passenger cars, the tire stiffness of the truck tires usually increases as the normal load increases, even beyond its rated load. Under the assumption that the truck tire stiffness is proportional to the normal load, the sensitivity of the tractor lateral dynamics with respect to other parameters is reduced as observed in Figure 2.2.16.

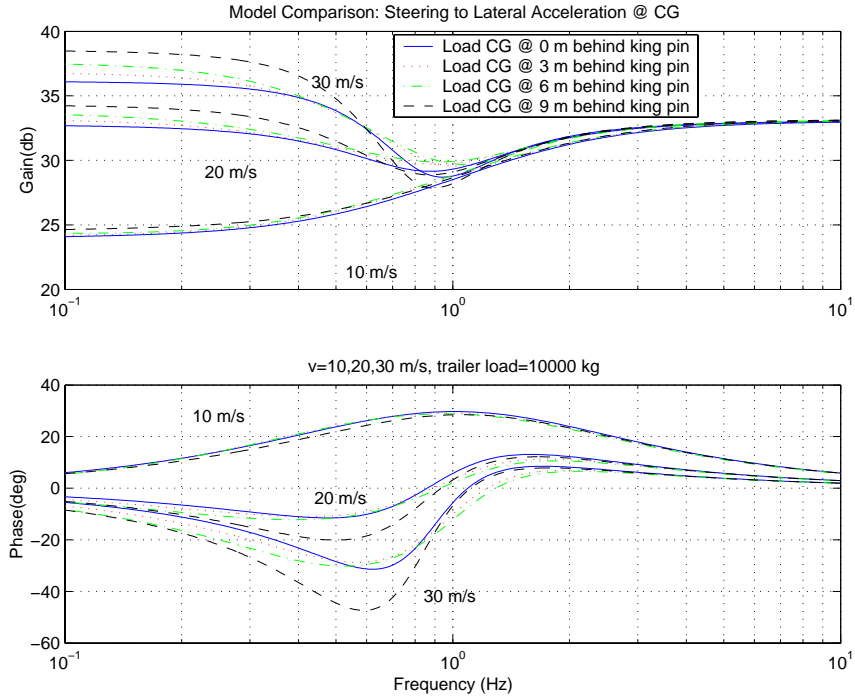


Figure 2.2.16: Tractor semi-trailer transfer function from steering angle to lateral acceleration at tractor CG with changing loading locations (tire stiffness proportional to normal load)

The tractor dynamics when the trailer is hooked up are then compared with the case where there is no trailer attached. A simple bicycle model is used to represent the tractor dynamics without trailer:

$$\frac{d}{dt} \begin{bmatrix} \dot{y}_r \\ \dot{\epsilon}_r \end{bmatrix} = \begin{bmatrix} -\frac{2C_f + 2C_r}{m_1} & -v + \frac{2C_r l_2 - 2C_f l_1}{m_1 v} \\ -\frac{2C_f l_1 - 2C_r l_2}{I_{z1} v} & -\frac{2C_r l_2^2 + 2C_f l_1^2}{I_{z1} v^2} \end{bmatrix} \begin{bmatrix} \dot{y}_r \\ \dot{\epsilon}_r \end{bmatrix} + \begin{bmatrix} \frac{2C_f}{m_1} \\ \frac{2C_f l_1}{I_{z1}} \end{bmatrix} \delta. \quad (2.2.2.3)$$

Figure 2.2.17 illustrates the general observations of the comparisons between tractor semi-trailer and tractor only under the constant tire stiffness assumption (which results in larger model variation):

- The tractor's lateral dynamics with and without trailer exhibit very similar trends with respect to all vehicle parameters as long as both loading and tire stiffness are within the specifications.
- The major differences between the dynamics of tractor with and without trailer are: (1) Higher gains at lower frequency especially under higher speeds. This is attributed to the fact that the trailer is dragging the tail end of the tractor, resulting in a smaller radius. (2) More delays around the articulation mode frequency, which is the result of the transfer of the lateral energy from tractor to trailer.

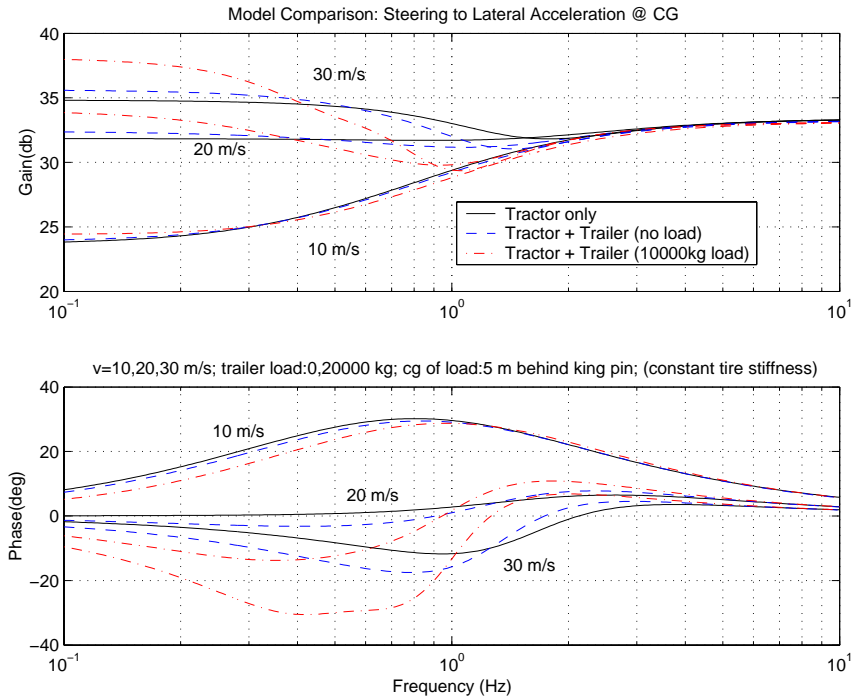


Figure 2.2.17: Tractor semi-trailer and Tractor only transfer function from steering angle to lateral acceleration at tractor CG (constant tire stiffness)

Figure 2.2.18 shows a very interesting counter-intuitive result. The lateral dynamics of tractor semi-trailer combinations change significantly with respect to speeds, trailer loading conditions, as well as tire stiffness changes. The major differences between the dynamics of tractor with and without trailer are: (1) Higher gains at lower frequency especially under higher speeds. This is attributed to the fact that the trailer is dragging the tail end of the tractor, resulting in a smaller radius. (2) Delay around the articulation mode frequency, which is the result of transferring the lateral energy from tractor to trailer. However, by practicing driving the tractor with trailer, one will realize that its lateral dynamics are very stable while driving forward except in the situation of severe braking. The only additional skill a driver needs is to properly adjust the steering so that the trailer can be positioned at the right location during turning. Very little additional dynamic compensation is necessary. Figures 2.2.18 and 2.2.19 compare the lateral acceleration dynamics at the vehicle CG and at 1-tractor length ahead of the vehicle under significant parameter variations. The comparison (especially between 0.2 to 1 Hz) suggests that “looking-ahead” dramatically desensitizes the vehicle steering dynamic variations. By looking ahead, the driver does not need to alter its steering dynamic compensation significantly to stabilize different vehicles. This observation suggests that (1) it is difficult to identify correctly the parameters of tractor and semi-trailer combinations, (2) the tractor semi-trailer dynamics changes drastically with respect many operational and loading conditions, (3) “look-ahead,” which the controller should provide, will desensitize these parameter uncertainties.

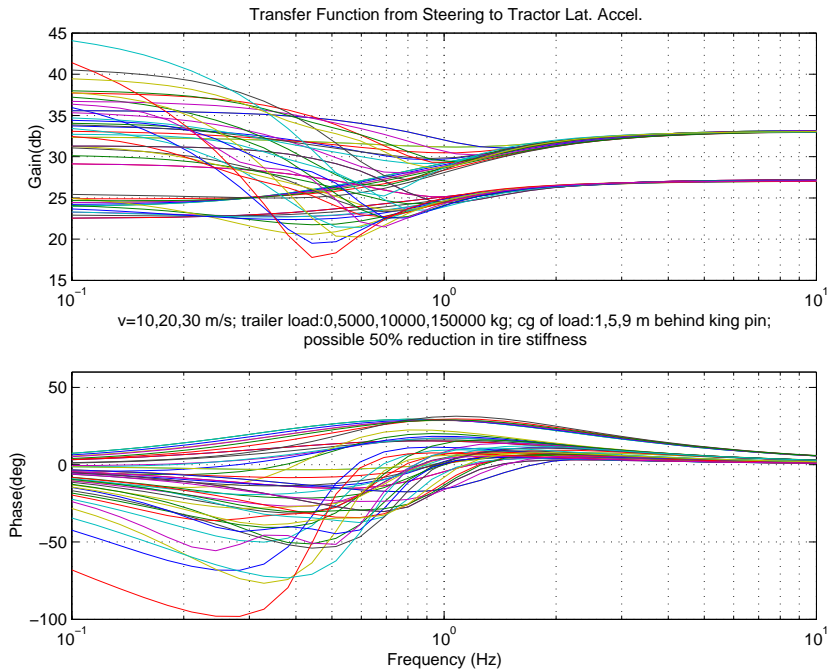


Figure 2.2.18: Tractor semi-trailer transfer function from steering angle to lateral acceleration at tractor CG: $v=10,20,30$ m/s; trailer load=0,5000,10000,15000 kg; CG of load=1,5,9 m behind king pin; possible 50% reduction in tire stiffness (assuming tire stiffness does not vary with load)

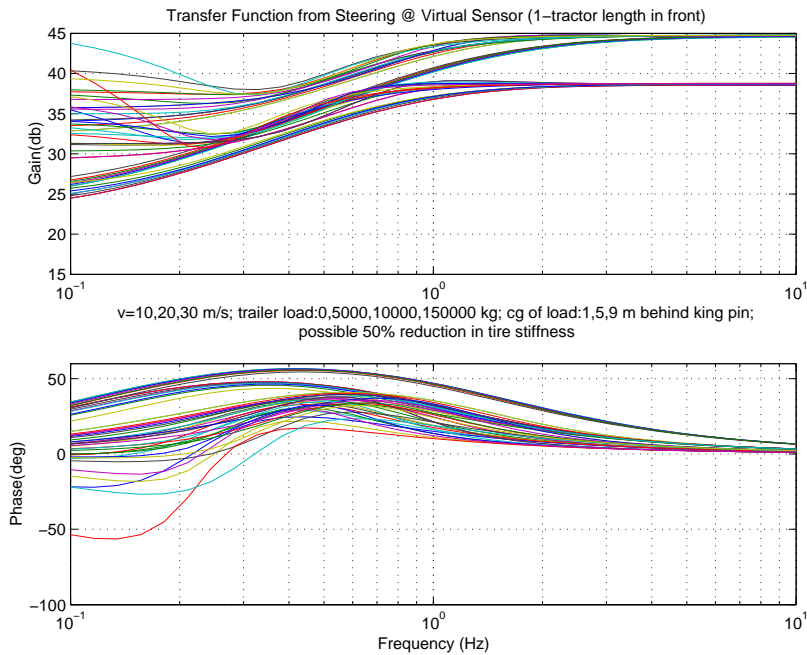


Figure 2.2.19: Tractor semi-trailer transfer function from steering angle to lateral acceleration at tractor at 1-tractor length ahead: $v=10,20,30$ m/s; trailer load=0,5000,10000,15000 kg; CG of load=1,5,9 m behind king pin; possible 50% reduction in tire stiffness (assuming tire stiffness does not vary with load)

References for Chapter 2

- [1] H. Bae, J. Ryu, and C. Gerdes, Road grade and vehicle parameter estimation for longitudinal control using GPS, *Proceedings of IEEE Intelligent Transportation Systems Conference*, 2001, pp. 166-171.
- [2] B. Song, J.K. Hedrick, and A. Howell, Fault tolerant control and classification for longitudinal vehicle control, *Journal of Dynamic Systems, Measurement, and Control*, vol. 125, September, 2003, pp 320-329.
- [3] J.C. Gerdes, Decoupled design of robust controllers for nonlinear systems: as motivated by and applied to coordinated throttle and brake control for automated highways, *Ph. D. thesis*, U.C. Berkeley, 1996
- [4] A. Gangopadhyay and P. Meckl, Modeling, validation and system identification of a natural gas engine, *Proceedings of the American Control Conference*, June, 1997, pp 294-298
- [5] R. Weeks and J.J. Moskwa, Transient air flow rate estimation in a natural gas engine using a nonlinear observer, *SAE Transactions*, no. 940759, 1994, pp 1-18.
- [6] *SAE truck and bus control and communications network standards manual* (2001 edn). SAE International, 2001
- [7] D. Cho and J.K. Hedrick, Automotive power modeling for control, *Journal of Dynamic Systems, Measurement, and Control*, vol. 111, December, 1989, pp 568-576
- [8] S. Choi and J.K. Hedrick, Vehicle longitudinal control using an adaptive observer for automated highway systems, *Proceedings of American Control Conference*, vol.5, 1995, pp 3106-3110
- [9] R. Limpert, *Brake design and safety*, SAE, Inc., Warrendale, PA, 1992
- [10] X. Y. Lu and J. K. Hedrick, 2002, Real-time estimation and compensation for tire slip in longitudinal control, Supplement to *Int. J. of Vehicle System Dynamics*, Vol. 37, p50-66. [XYL1]
- [11] Jacobs Vehicle Systems, Engine Brake Theory, <http://www.jakebrake.com/askus/faq/ebr.htm> [JAC]
- [12] Fitch, J. M., 1994, *Motor Truck Engineering Handbook*, 4th Ed., SAE, USA [FIT]
- [13] Hedrick, J. K., Nonlinear controller design for automated vehicle applications, *Proc. UKACC Int. Conf. on Contr. '98, 1-4 Sept. 1998, Swansea, U. K.*, p23-31. [HDK]
- [14] Kiencke, U. and L. Nielsen, 2000, *Automotive Control Systems, For Engine, Driveline, and Vehicle*, Springer, Berlin. [KIE]
- [15] Wong, J. Y., 1978, *Theory of Ground Vehicles*, John Wiley & Sons, New York. [WON]
- [16] Cho, D. and J. K. Hedrick, 1989, Automotive power train modeling for control, *Transaction of ASME, J. Dynamic Systems, Measurement and Control*, **Vol. 111**, p 568-576. [CHO]
- [17] Hammache, M., M. Michaelian, and F. Browand, 2001, Aerodynamic Forces on Truck Models, Including Two Trucks in Tandem, *California PATH Research Report*, UCB-ITS-PRR-2001-27. [HAM]

- [18] Kao, M. and Moskwa, J. J., 1995, Turbocharged diesel engine modeling for nonlinear engine control and state estimation, *Trans. ASME, J. of Dynamic Systems, Measurement and Control*, Vol.117, No. 1, p20-30. [KAO]
- [19] Lyshevski, E. L., A. S. S. Shinha and J. P. Seger, 1999, Nonlinear analysis and control of turbocharged diesels, *Prof. IEEE Int. Conf. on Contr. Appl.*, Kohala Coast-Island of Hawai'i, USA, August 22-27, p858-862. [LYS]
- [20] Kolmanovsky, I., P. Moraal, M. van Nieuwstadt, and A. G. Stefanpolou, 1999, Issues in modeling and control of variable geometry turbocharged engines, *Systems Modeling and Optimization*, Eds. M. P. Polis, A. L. Dontchev, P. Kall, I. Lasiecka, and A. W. Chapman, Hall/CRC Research Notes in Mathematics, p436-445. [KOL]
- [21] Rajamani, R., S. B. Choi, J. K. Hedrick and B. Law, 1998, Design and experimental implementation for a platoon of automated vehicles, *Proc. of the ASME Dynamic Systems and Control Division*. [RAJ1]
- [22] Rajamani, R., H.-S. Tan, B. Law and Zhang, 2000, Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons, *IEEE Trans. on Control Systems Technology*, **Vol. 8, No. 4**, p. 695-708. [RAJ2]
- [23] Cummins Customer Assistance Center, 1999, *Operation and Maintenance Manual - N-14 Plus series Engines*, Cummins Engine Company. [CUM]
- [24] H. Peng and M. Tomizuka, "Preview Control for Vehicle Lateral Guidance in Highway Automation," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 115, no. 4, pp. 152-166, 1991.
- [25] J. Farrell, H.-S. Tan and Y. Yang, "Carrier Phase GPS-aided INS based Vehicle Lateral Control," *ASME Journal of Dynamics Systems, Measurement, and Control*, vol.125, no.3, Sept.2003, pp.339-353.
- [26] C. Chen, M. Tomizuka, "Lateral Control of Commercial Heavy Vehicle," *Vehicle System Dynamics*, vol. 33, no. 6, 2000, pp. 391-420.
- [27] P. Hingwe, H.-S. Tan, A. K. Packard and M. Tomizuka, "Linear Parameter Varying controller for Automated Lane Guidance – Experimental Study on Tractor-Trailers," *IEEE Transactions on Control Systems Technology*, vol.10, no. 6, Nov., 2002, pp. 793-806.
- [28] H.-S. Tan, B. Bougler, and P. Hingwe, "Automatic Steering Control of Tractor Semi-Trailer Using Controller Designed for Passenger Vehicles," in *Proceedings of the 2nd IFAC Conference on Mechatronic Systems*, Berkeley, CA, USA, Dec. 2002, pp. 63-68.

Chapter 3. Modifications to the Host Vehicle Platforms

3.1 Overview Description

The base vehicles acquired from the manufacturers had to be extensively modified in order to provide them with the capabilities for automated driving. These modifications are described in this chapter. An overview of the modifications can best be visualized using Figures 3.1.1 and 3.1.2 for the buses and trucks respectively. As these figures indicate, the modifications are quite similar as soon as we rise above the level of the different actuation means for engine and braking.

Fig. 3.1.1 – Bus Component Additions

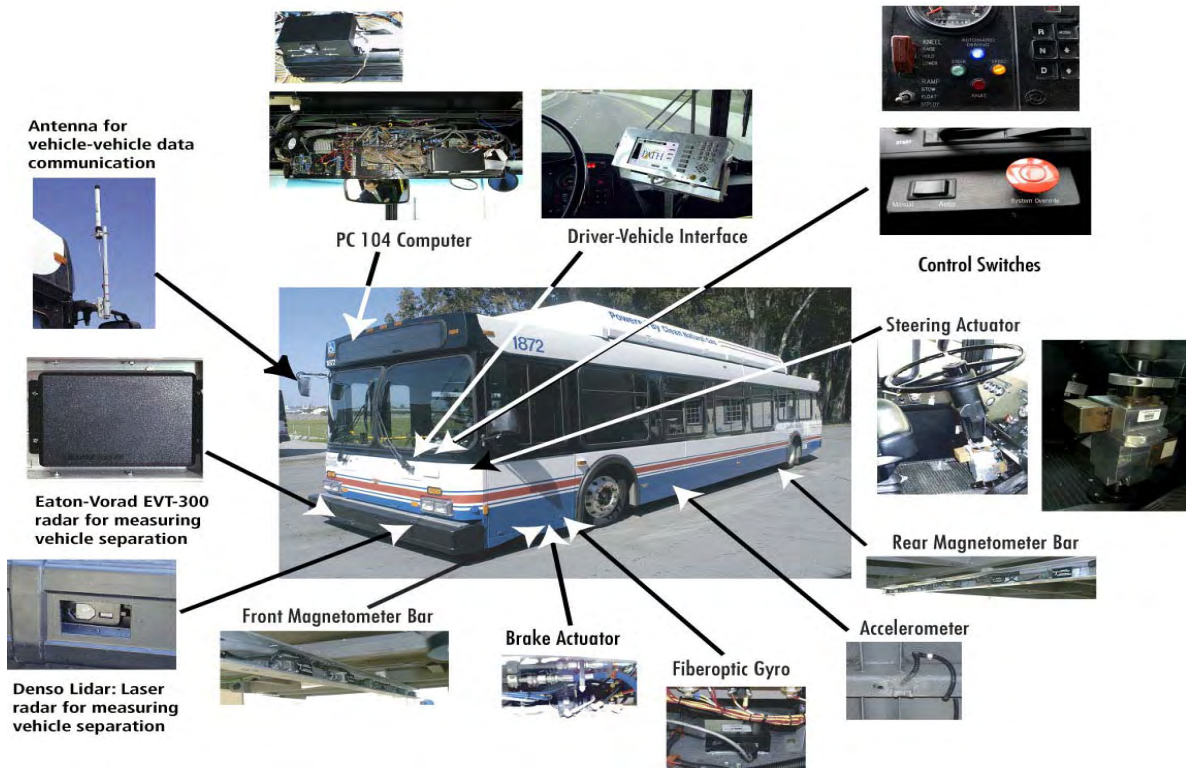




Fig. 3.1.2 – Truck Component Additions

Installing all these components required not only mechanical integration with the base vehicles, but also electrical integration, which involved significant installation of wiring and connectors. These details are specific to the individual vehicles and components involved, and are not of more general relevance.

3.2 Modifications to Buses, for Automation and Data Acquisition

3.2.1 PC-104 Control/Data acquisition and DVI Computers

3.2.1.1 Hardware Overview

PATH has adopted the PC/104 computer standard for control of heavy vehicles which includes three Freightliner trucks and three New Flyer transit buses. The PC/104 was chosen since it is rapidly becoming a standard for computers often found in factories, laboratories, and machines to provide programmable control of complex systems. PC/104 is a standard for PC-compatible modules (circuit boards) that can be stacked together to create a complete computer system. These types of systems are often found in factories, laboratories, and machinery to provide programmable control of a complex system. PC/104 systems are very similar to standard desktop PCs but with a different form factor. The name "PC/104" is derived from this likeness and the special stackable bus connector having 104 pins (Figures 3.2.1 and 3.2.2).



Figure 3.2.1. Typical PC/104 Stack.

These systems can be programmed with the same development tools used with full-size PCs which reduces the need and cost of custom development efforts. Although only about 100 cm x 100 cm, PC/104 boards are very powerful for their size. PC/104 products are designed for minimal power consumption, small foot print, modularity, expandability, and ruggedness.

The computer system configuration used on the transit buses is shown in Figure 3.2.2 and consists of four major components:

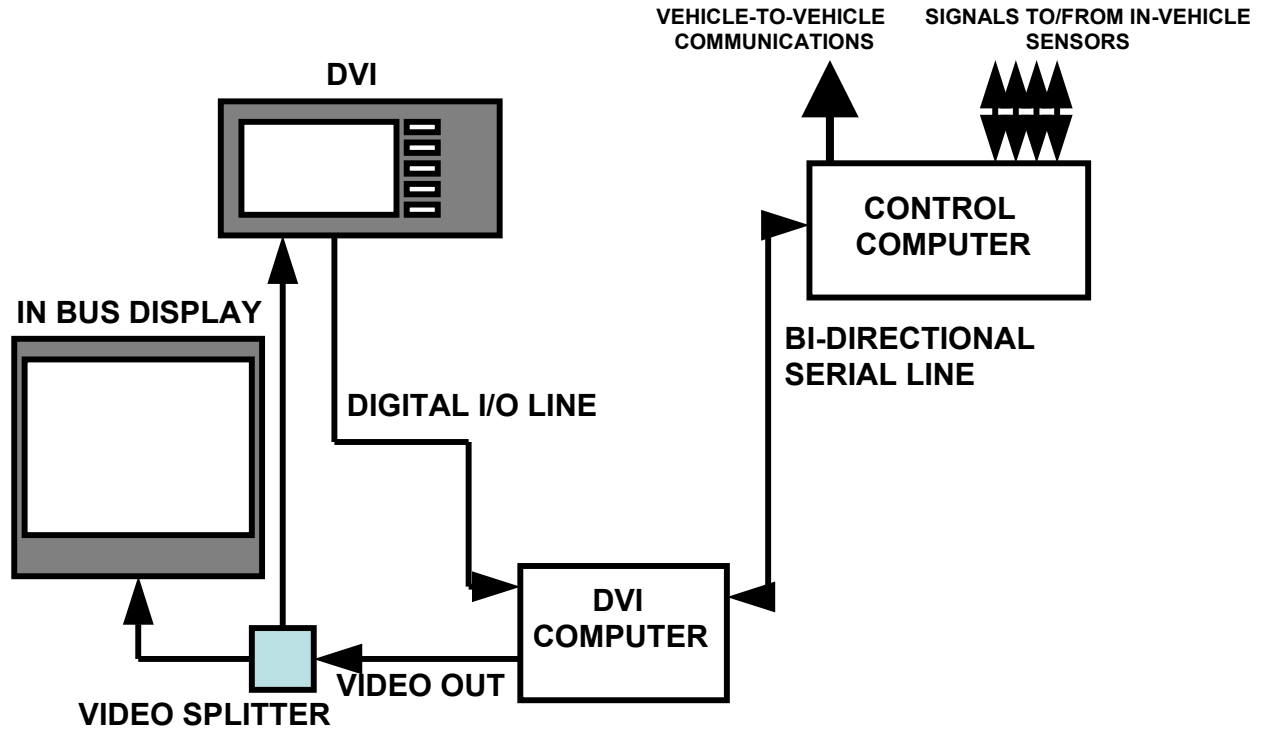


Figure 3.2.2. Computer system Configuration

1. **The Control Computer.** Used to read all sensors, issue speed and steering commands, and communicate state information to the other buses.

2-3. **The DVI (Driver Vehicle Interface) Computer/DVI Control Box.** This system is designed to generate a graphical interface of the system state to the driver. The state information is transferred to the DVI computer from the control computer using a bi-directional serial line. The DVI not only shows vehicle information, but also allows control mode/menu switching using a push button pad next to the display.

4. **In-Bus Display.** A 19" flat panel computer monitor was installed viewable to the passengers showing the display from the DVI control box. This was used since the control box display is small and viewable only to the driver.

The vehicle control computers used in the PATH heavy vehicles consist of 10 stacked boards shown in Figure 3.2.3, and the DVI computer stack is shown in Figure 3.2.4.

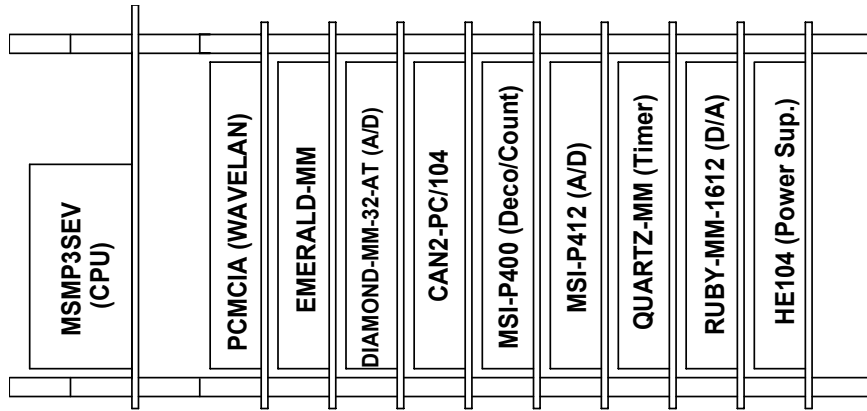


Figure 3.2.3. PC/104 Stack for the Control Computer.

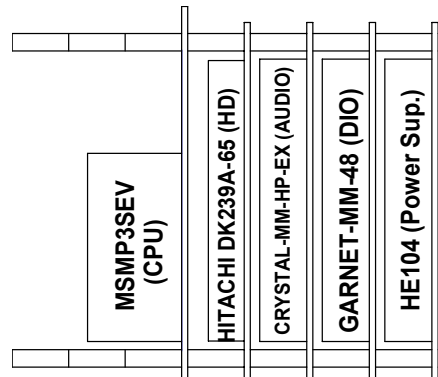


Figure 3.2.4. PC/104 Stack for the DVI Computer.

A photograph of the control computer as installed in the New Flyer transit bus is shown in Figure 3.2.5.



Figure 3.2.5. PC/104 Control Computer Installed in New Flyer Transit Bus.

The control computer is located in a compartment above the front window and behind the destination sign. The computer and associated cables/terminal blocks are mounted to a shelf that can be moved in and out of the compartment for ease of maintenance.

3.2.1.2 Individual Board Descriptions and Functionality

A brief description of each board and function is given below:

1. CPU: Intel 400 MHz P3Celeron MSM-P3SEV. The board includes VGA, 100/10Base-T LAN ethernet, E-IDE hard disk interface, 3.5" micro Floppy disk interface, and COM1, COM2 serial ports. Vendor: Advanced Digital Logic.



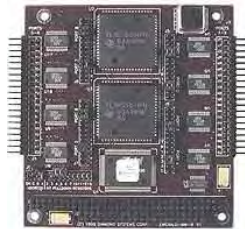
MSM-P3SEV

2. PCMCIA: 2-slot PCMCIA board MSMJ104D. Used for holding an Orinoco 802.11b vehicle-to-vehicle communications card. Vendor: Advanced Digital Logic.



MSMJ104D

3. Serial ports: 8-port RS-232 serial port board EMERALD-8232-XT. Includes 8 programmable digital I/O lines. The ports are used for reading the radar, lidar, rate gyroscope, J1587 bus, and GPS. It also communicates information to and from the DVI computer. Vendor: Diamond Systems.



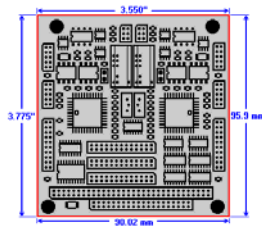
EMERALD-8232-XT

4. 32-channel 16-bit 200KHz analog to digital board DIAMOND-MM-32-AT. Includes 4 12-bit analog output channels, 24 programmable digital I/O lines, and 1 32-bit counter/timer. This board reads the analog signals from the magnetometers, accelerometer, steering potentiometer. Vendor: Diamond Systems.



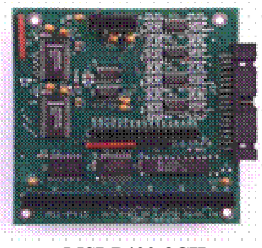
DIAMOND-MM-32-AT

5. Dual CAN field bus interface board: CAN2104-30-20. Includes 32 32-bit digital I/O lines. The J1939 data bus is read with this board. Vendor: SSV Embedded Systems.



CAN2104-30-20

6. 3-channel quadrature decoder/counter board MSI-P400-3CH. The steering wheel angular position is read by this card .Vendor: MicroComputer Systems.



MSI-P400-3CH

7. 16-channel 12-bit digital to analog output board RUBY-MM-1612-XT. Includes 24 digital I/O lines. The analog outputs are used for the steering torque, accelerator pedal, and brake valve commands, while the digital I/O lines monitor the magnetometer health states, and send control transitions. Vendor: Diamond Systems.



RUBY-MM-1612-XT

8. 50 watt DC/DC power supply board: HE104-512-V512. Vendor: Diamond Systems.



HE104-512-V512

The following are PC/104 boards are used in the DVI computer:

9. SoundBlaster Pro compatible full-duplex, high-power 16-Bit stereo audio board CRYSTAL-MM-HP-EX. Vendor: Diamond Systems



CRYSTAL-MM-HP-EX

10. 48-line high current digital I/O: GARNET-MM-48. The push button pad inputs are read with this board. Vendor: Diamond Systems



GARNET-MM-48

All computers are enclosed in a PC/104 mounting enclosure, Can-Tainer. Vendor: Tri-M Engineering.



Can-Tainer

More detailed descriptions of the board configurations are found in Appendix A.

3.2.2 Magnetometer Sensors and Sensor Bars

The development of a reliable and accurate lateral referencing system is crucial to the success of the lateral guidance system for any lane assist systems of heavy vehicles.

Since the bus precision docking system has the tightest accuracy requirements, it was used as the benchmark system for the magnetometer sensor design. For the precision docking system, the accuracy requirement for the lateral sensing system is directly proportional to the required docking accuracy especially along the docking platform. The lateral sensing accuracy requirement was set to be better than 1 centimeter for the docking accuracy targeted to be better than the same value. The assumption is that the installation and measurement errors are randomly and evenly distributed relative to the correct position.

PATH has proposed and developed a lateral referencing and sensing system that is based on magnetic markers embedded in the road center to provide the lateral position and road geometric information. The automatic steering guidance system based on such technology provides the control system with the following two fundamental pieces of information: the vehicle position with respect to the roadway, and the current and future road geometry. Two arrays of magnetometers, one located just behind the front bumper and the other at about 4-6 meters behind the front sensors, were used to “simultaneously” obtain front and rear lateral offset measurements.

Extensive development and experiments have been performed on magnetic marker-based lateral sensing systems for many PATH vehicles equipped with automated steering control. The vast knowledge available about this lateral sensing technique as well as its high reliability were two of the primary reasons that this technology was first chosen to support the heavy vehicle automation including Precision Docking steering guidance system. Other positive characteristics of this lateral sensing technique include good accuracy (better than one centimeter), insensitivity to weather conditions, and support for binary coding. The requirement of modifying the infrastructure (installing magnets) and the inherent “look-down” nature (the sensor measures the lateral displacement at locations within the vehicle physical boundaries, versus look-ahead ability) of the sensing system are two known limitations of this technology. The principal idea for this sensing system is straightforward. Magnetic markers are installed under the roadway delineating the center of each lane or any other appropriate lines for the specific applications. Magnetometers mounted under the vehicle sense the strength of the magnetic field as the vehicle passes over each magnet. Onboard signal processing software calculates the relative displacement from the vehicle to the magnet based on the magnetic strength and the knowledge of the magnetic characteristics of the marker. This computation is designed to be insensitive to the vehicle bouncing (e.g., heave and pitch) and the ever-present natural and man-made magnetic noises. Furthermore, the road geometric information, such as bus docking locations, road curvatures, lane change areas, can be encoded as a sequence of bits, with each bit corresponding to a magnet. The polarity of each magnet represents either 1 (one) or 0 (zero) in the code. In addition to the lateral displacement measurement and road geometry preview information, other vehicle measurements such as yaw rate, lateral acceleration, and steering wheel angle may also be used to improve the performance of such a lateral guidance system.

3.2.2.1 Magnetic Noise Effects

Four major noise sources are usually present in the magnetic signal measurements in a typical vehicle operational environment: ambient geomagnetic field, local magnetic field distortion, vehicle internal electromagnetic field, and electrical noise.

The most frequent external disturbance is the ever-present earth's permanent magnetic field, which is usually on the order of 0.5 Gauss. The value of the earth field measured by the magnetometers on the vehicle depends on the location of the vehicle on earth as well as the altitude and orientation of the vehicle. Although the earth magnetic field usually changes slowly, sharp turns and severe braking can quickly change the field measurements along the vehicle axes.

The most serious noise problems are caused by local anomalies due to the presence of roadway structural supports, reinforcing rebar, and the ferrous components in the vehicle or under the roadway. Underground power lines are another source of such local field distortion. Rebar or structural support usually creates a sharp change in the background magnetic field and sometimes is difficult to identify. Most signal processing algorithms will have some difficulty recovering from such sharp distortions. The ferrous components in the vehicle, on the other hand, can be isolated as long as their locations are fixed with respect to the magnetometers, or are located at a significant distance from the sensors.

A third source of noise comes from the alternating electric fields generated by various motors or rotating permanent magnets or magnetized materials operating in the vehicle. These rotating "magnets" may include alternator, fan, electric pump, steel belts inside tires, compressor and other actuators. However, their effects vary according to the rotational speed and distance from the magnetometers. The higher the rotating speed or the farther it is placed away from the magnetometers, the less the resultant noise. Sometimes modest changes in sensor placement can alter the size of such disturbances.

The last common noise source arises from the electronic noise in the measurement signal itself. Such noise can be created by the voltage fluctuations in the electrical grounding or from the power source. It can also be a result of poor wiring insulation against electromagnetic disturbances. Usually, the longer the wire, the higher such noise. Although low-pass filtering can reduce the magnitude of such disturbances, noticeable degradation of the magnetic sensor signal processing algorithm occurs when such noise level exceeds 0.04 Gauss. Digital transmission of magnetic field measurements or local embedded processing are two possible approaches that can significantly reduce such noise.

3.2.2.2 Magnetic Sensing Algorithm

One of the important attributes of the lateral sensing system is its reliability. Currently, there exist several algorithms designed to detect the relative position between the marker and sensor (magnetometer), as well as to read the code embedded within a sequence of these markers. Three magnetic marker detection and mapping algorithms have been

experimented with by PATH. The first is called the “peak-mapping” method, which utilizes a single magnetometer to estimate the marker’s relative lateral position when the sensor is passing over the magnet. The second algorithm is the “vector ratio” method that requires a pair of magnetometers to sample the field at two locations. It returns a sequence of lateral estimates in a neighborhood surrounding, but not including the peak. The third is the “differential peak-mapping” algorithm that compares the magnetic field measurements at two observation points to eliminate the common-mode contributions and reconstructs a functional relationship between the differential sensor readings and the lateral position using the knowledge of the sensor geometry. The “peak-mapping” algorithm was selected for the precision docking project because it has been proven effective over a wide range of speeds and has been widely applied in many experimental applications conducted by PATH.

In the heavy vehicle operational environment, the magnetic field maps can deviate quite significantly from the theoretical dipole equation prediction because of the massive amount of ferrous material from the body structure located just above the magnetometers. Numerical mapping created by empirical data gathering (calibration) is used to create the associated inverse maps. Figures 3.2.6 and 3.2.7 show the front and rear magnetic tables for the 40’ bus (C1), respectively. The figures consist of tables of the seven magnetometers starting from the right side of the bus to the left, designated as follows: right-right, right, center-right, center, center-left, left and left-left. Each table is obtained with two sets of calibration data, one at a lower sensor height (at around 7 inches from the magnetometer to the magnet) and the other at a higher sensor height (at 11 inches from the magnetometer to the magnet). Each half-circle in the table consists of vertical and horizontal fields of the marker that are collected at 2-cm interval of lateral displacement. The magnetic tables clearly depict the nonsymmetrical nature of the magnetic field due the adjacent ferrous material. The calibration process was repeated for every bus and every truck to ensure that the static local magnetic effects for each heavy duty vehicles were accounted for.

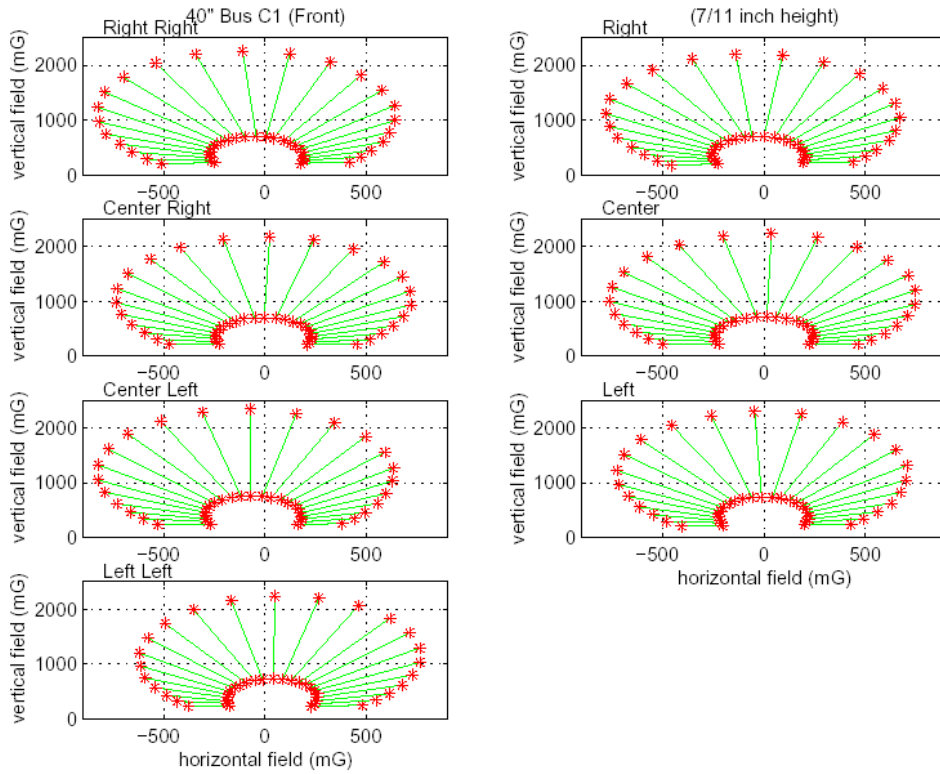


Figure 3.2.6: 40' Bus (C1) Front Magnetometer Calibration Tables

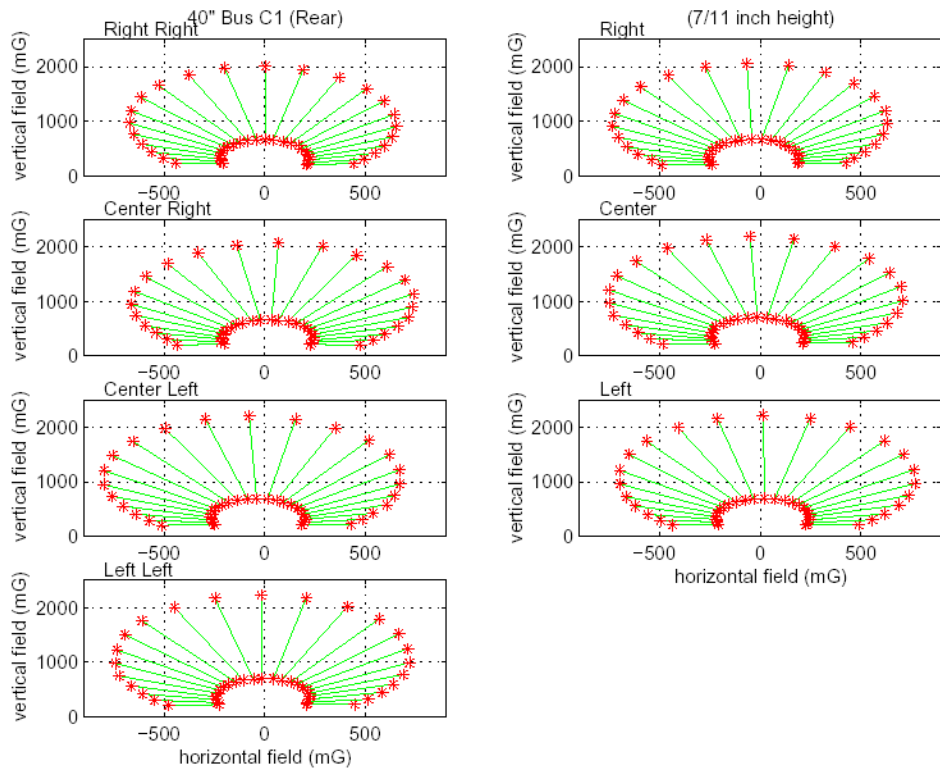


Figure 3.2.7: 40' Bus (C1) Rear Magnetometer Calibration Table

3.2.2.3 Signal Processing

The magnetometers signal processing for the “peak-mapping” method involves three procedures: peak detection, earth field removal and lateral displacement table look-up (see Figure 3.2.8 for block diagram of signal processing algorithm). Although it is straightforward in principle, it becomes complicated when the reliability of the process is the major concern. Many parameters in the lateral sensing signal processing software need to be tuned in order to provide consistent lateral displacement information regardless of vehicle speeds, orientations, operating lateral offsets and vehicle body motions. Debugging can become very time consuming when failure conditions cannot be recreated. To improve the reliability of the lateral sensing system with the magnetic road markers, PATH has developed a “reconstructive” software system for the lateral sensing signal processing that supports the tuning of the parameters using stored real-time data. In such a setup, any erroneous situation can be recreated in a lab environment and debugged with ease.

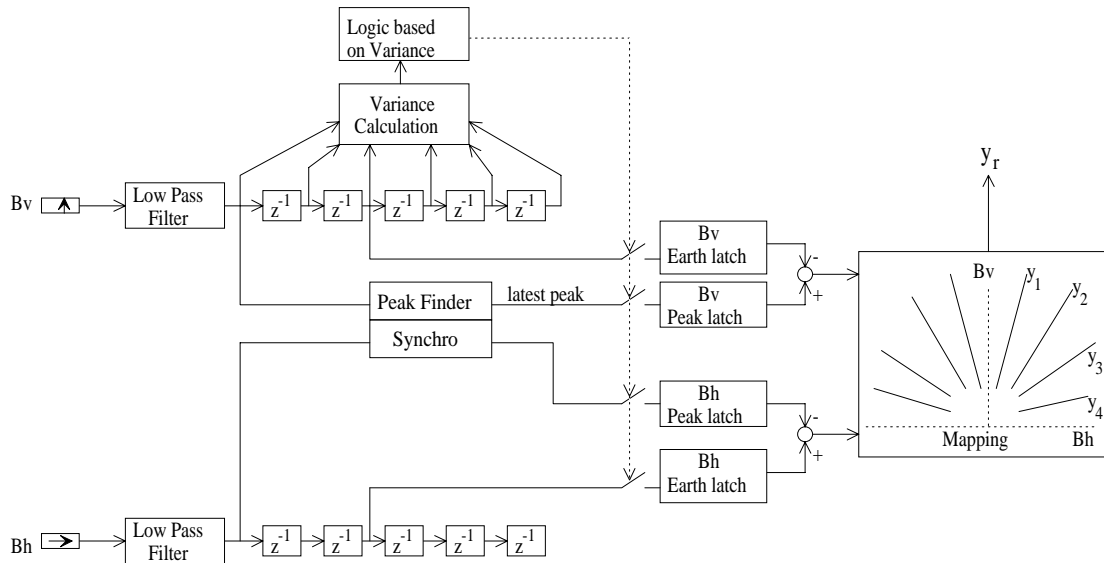


Figure 3.2.8: “Peak-Mapping” Magnetometer Signal Processing Block Diagram

3.2.3 Steering Actuators

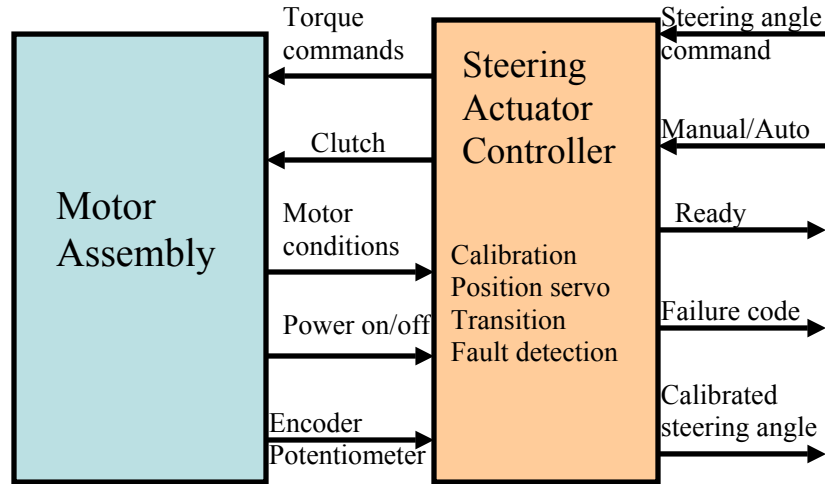


Fig. 3.2.9 Schematic of Steering Actuator Hardware

3.2.3.1 System Configuration

Fig.3.2.9 shows the block diagram of the PATH steering actuator. The motor assembly is manufactured by NSK. As shown in Fig. 3.2.10, the steering actuator motor assembly consists of a steering column, DC motor actuating steering column, an electromagnetic clutch and angle sensors measuring steering wheel position. The DC motor connects to the steering column through a clutch and reduction gear. An incremental encoder is mounted on the motor shaft to measure the relative position of the steering wheel. A multi-turn potentiometer is connected with the column shaft via a pulley gear and belt to measure the absolute position of the steering wheel. Motor current and clutch ON/OFF are controlled by the ECU. The ECU receives the torque command from the upper-level computer and issues a corresponding current command so that the DC motor will generate the required torque. The clutch can also be controlled by the upper-level computer by issuing clutch command to the ECU. The ECU has built-in self-diagnostic features. The health condition of the motor is fed back to the upper-level computer through the motor condition signal.

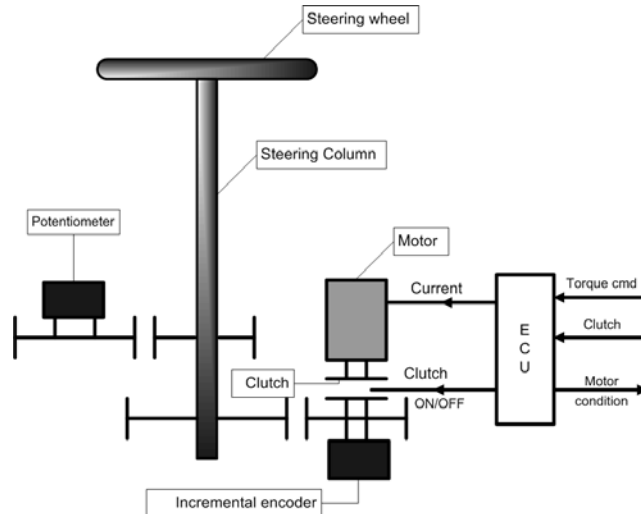


Fig. 3.2.10 - Schematic of Steering Actuator Motor Assembly

The steering actuator controller is a software package designed by PATH to have the following functions:

1. Calibration: The function of calibration is to find zero steering angle of the bus.
2. Position servo: Position servo is a closed loop controller. It receives the steering angle command and issues the torque command to the steering actuator hardware so that the steering wheel will turn to the desired steering wheel angle position.
3. Smooth transition between manual and automatic control.
4. Fault detection for sensors and motor.

3.2.3.2 Position Servo Design

The position servo is the key function of the steering actuator system. Successful lateral controller design requires at least 4-5 Hz closed servo loop bandwidth with 1 degree accuracy on the steering wheel. Before the servo design could be carried out, extensive experiments were conducted to study the open-loop characteristics of the steering actuator. The experimental results revealed a quite challenging servo design problem. Since the Freightliner trucks have a more powerful hydraulic power steering system and the trucks' steering mechanism does not have the significant backlash problem that the buses do, the design of the truck steering actuator was not that challenging compared with the design of the bus steering actuator. Therefore, the design of the steering actuator system for the 40-foot bus will be used as an example for illustration purposes.

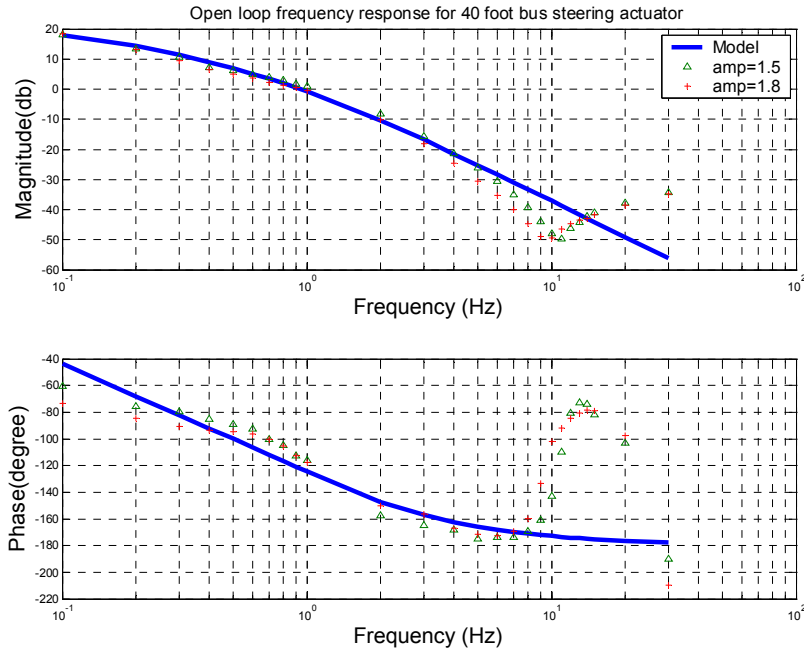


Fig. 3.2.11 - Steering Actuator Open Loop Frequency Response for 40-foot Bus

First, sine sweep tests were used to identify the open-loop frequency response from torque command (V) to steering wheel angle (degree) with different input amplitudes. As shown in Fig. 3.2.11, the open-loop bandwidth of the 40 foot bus steering actuator is less than 1 Hz. The experimental data in the range of 10 Hz and above are not usable because at these higher frequencies the steering wheel hardly moves and the signal-to-noise ratio is poor. The model represented by the solid line in Figure 3.2.11 was used for designing the inner-loop steering actuator controller.

Second, a slow ramp input was used to study the effect of friction on the road. As shown in Fig. 3.2.12, the friction effect is so dominant that the steering wheel starts moving only when the torque command reaches almost half of its full capacity (2V). This means that the actuating motor is seriously “under powered”. Although this may facilitate the driver taking over under emergency situations, the “under powered” motor poses significant difficulty for servo loop design. This is especially true for low-speed applications such as precision docking, where the friction effect is dominant. Third, the steering mechanism of the original bus has about 20 degrees of backlash at the steering wheel. Such a hard nonlinearity, if not properly treated, may lower tracking accuracy, introduce limit cycles or even destabilize the entire control loop.

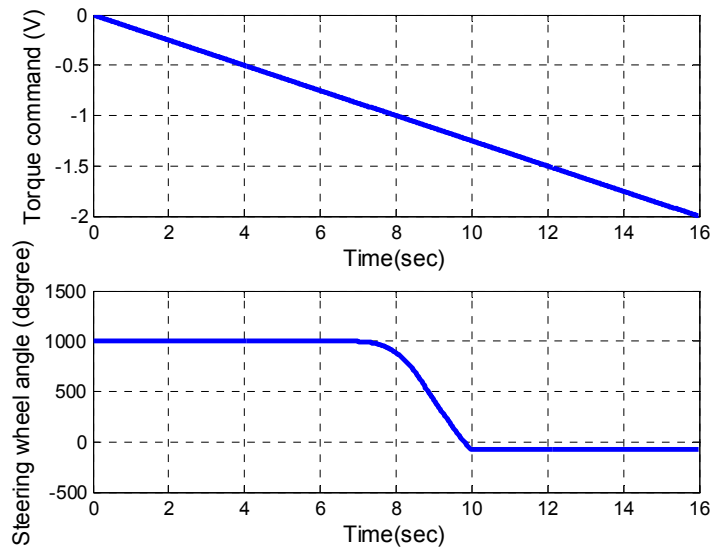


Fig. 3.2.12 – Torque Command and Steering Wheel Response to Ramp Input

To address the design difficulties mentioned above, different strategies were adopted. First, loop shaping was used to increase closed-loop bandwidth as much as possible. Second, a low-and-high gain design technique was used to address actuator saturation introduced by the “under powered” steering motor. Third, nonlinear compensation is used to address the nonlinear friction effect and to avoid a limit cycle due to the backlash of the steering mechanism.

The results of the final design are shown in the following figures. Fig. 3.2.13 shows the closed-loop frequency response, again recalling that the results at 10 Hz and above are not valid.

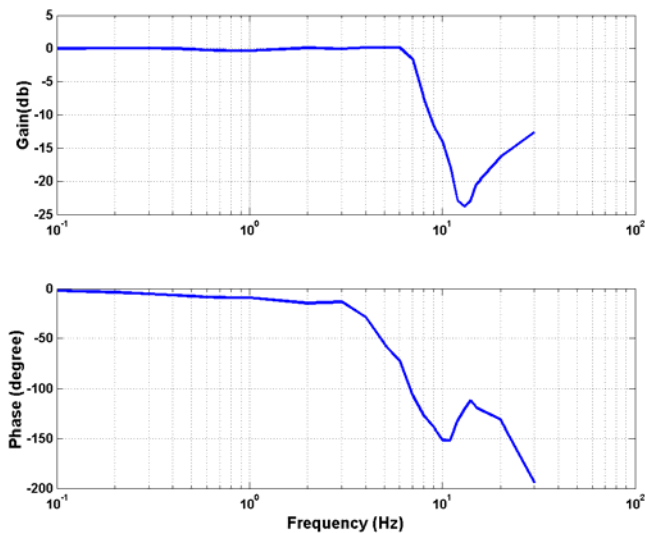


Fig. 3.2.13 – Closed-Loop Steering System Frequency Response

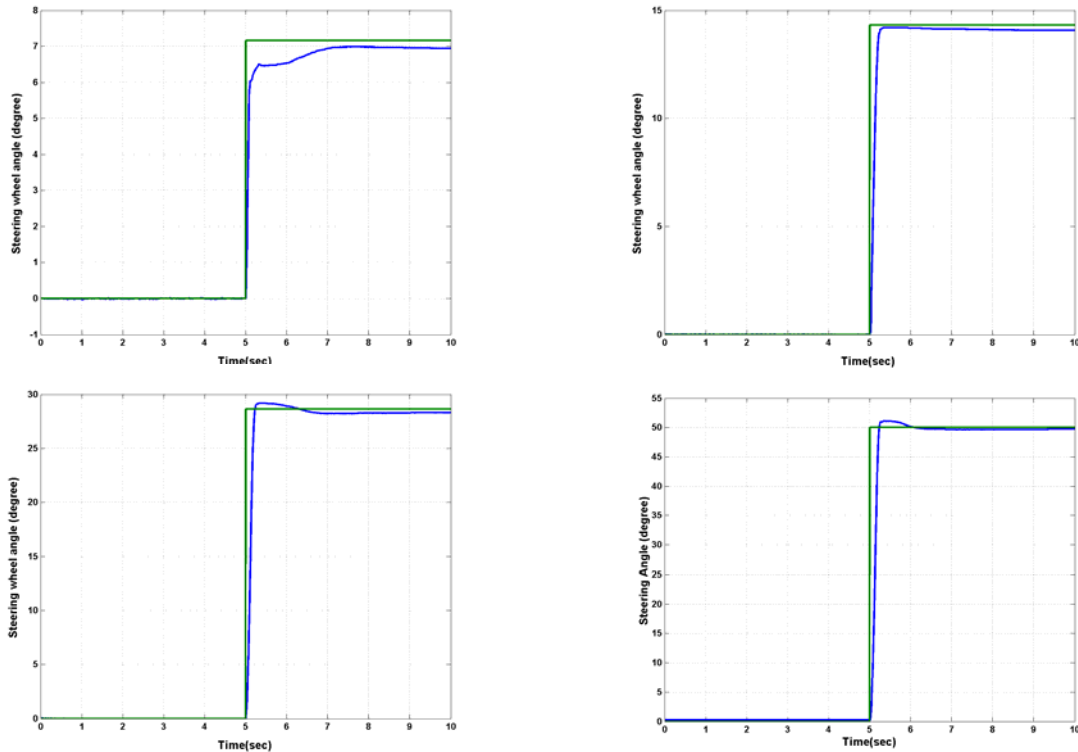


Fig. 3.2.14 – Steering Actuator Step Responses

Fig. 3.2.14 shows the step responses for different magnitude step commands, indicating the success of the closed-loop actuator design.

3.2.4 Body Sensors (acceleration, yaw rate)

It was important to obtain accurate inertial measurements of vehicle body motions in order to provide feedback of these motions to the vehicle control laws. The key inertial measurement systems that were installed on each vehicle were:

- Summit Instruments 2-axis accelerometer, with analog output and 1 g maximum range, to provide lateral and longitudinal vehicle accelerations.
- KVH Industries E-Core 2000 fiber-optic gyro, with 30 degrees per second maximum range and digital output, to provide yaw rate sensing.

3.2.5 Wireless Communication System

An integral part of the PATH longitudinal vehicle control system is the use of a wireless communications link to pass vehicle state information among the vehicles within a platoon. The vehicle state information consists primarily of inter-vehicle distance, velocity, and acceleration measurements, along with platoon coordination information,

such as fault status and operating mode. The requirements for the wireless communication system are strict, requiring real-time operation with transmission of the state information for all vehicles within the platoon within one control cycle (a period of 20 milliseconds). At the beginning of this project, a number of potential wireless communication technologies were investigated for application to this problem.

3.2.5.1 Potential Technologies

The ubiquity of commercial off-the-shelf wireless systems over the past few years gave a range of potential technologies that could have been used for the communications system. Several key requirements were used to narrow the potential candidates, including performance, availability of hardware, and the existence/feasibility of developing device drivers for the QNX operating system. The IEEE 802.11 standard for wireless LAN's was chosen due to its compatibility with the specified requirements and potential for continued support. Within this standard, several subgroups were commercially available and viable for use, including 802.11a and 802.11b. Although IEEE 802.11a had been chosen as the basis for developing the new standard for Dedicated Short Range Communications (DSRC), the technology was not sufficiently mature for experimental vehicle testing throughout the vehicle development process. On the other hand, the IEEE 802.11b standard had been widely accepted for a range of applications and offered the greatest amount of hardware implementations, availability of device drivers, and technical support.

While the IEEE 802.11b standard provides a physical specification and fundamental wireless networking capabilities, a higher-level communications protocol is required to provide an applications-layer interface among wireless nodes. Several protocols were investigated, including standard TCP/IP and the Wireless Token Ring Protocol (WTRP). WTRP was initially chosen as the communications protocol, including a port of its existing Linux implementation to QNX, however unresolved timing issues necessitated the development of a simple token ring protocol termed Real Time Token Ring (RTTR).

The following sections present an overview of the Orinoco 802.11b driver and RTTR protocol developed for QNX, as well as some representative characterizations of their performance. The interested reader can find detailed reference manuals for both in the Appendices of this report.

3.2.5.2 Orinoco 802.11b Driver

The Orinoco 802.11b PCMCIA cards are not officially supported under QNX, so a significant portion of this project involved development of a device driver. The open source `orinoco_cs` driver for Linux was used as a starting point, and a rudimentary networking stack was also created that mimics its Linux counterpart. Direct use of the default QNX network stack was not incorporated into the driver, since the operating system interface is proprietary.

The device driver runs as a process called `ori`, and access to the driver is conducted through atomic query-reply sequences using QNX inter-process messaging. This type of interface was used for efficiency, modularity, and flexibility. Although only the RTTR and WTRP protocols are currently supported by the driver process, other protocols could easily be included through this interface.

3.2.5.3 Real Time Token Ring (RTTR)

The Real Time Token Ring (RTTR) protocol is a simple token ring protocol that relies on implicit token passing to ensure that each node in a ring has an allocated time slot on which to transmit. The topology of a ring includes a single Master node and one or more Slave nodes. Each node is also given a prespecified node number, indicating its relation to the other nodes in the ring. A diagram of a three-node ring is shown below in Figure 3.2.15.

A single rotation in an RTTR ring is initiated by the broadcast of a message by the Master node after an a priori chosen desired rotation time termed `rotation_time`. During this period, all other nodes in the ring listen for incoming messages. The first Slave node, assigned node number 2, takes the token and transmits its own messages when either the Master node's message has been received or a timeout of $\text{rotation_time}/N$ milliseconds has occurred, where N is the total number of nodes in the ring. Similarly, all other Slave nodes in the ring transmit when either the previous node's message has been received or a timeout of $(n-1)*\text{rotation_time}/N$ milliseconds has occurred, where n is the Slave's node number.

Like the Orinoco driver, the RTTR protocol also runs as a separate process called `rttr`. It accesses the physical radio through the interprocess messaging interface described in the previous section. Control application processes, primarily the longitudinal controller, interact with the communications system through a set of data structures maintained within the PATH Database. These data structures are all instances of a fixed packet format that contains all information described in the introduction, along with additional fields used for timing and error detection. Prior to transmission, the RTTR process reads the current packet to transmit from the `DB_COMM_TX` variable in the database. Similarly, any packet received over the wireless link is stored in the database variable `DB_COMM_RXn`, where n is the originating node number of the packet.

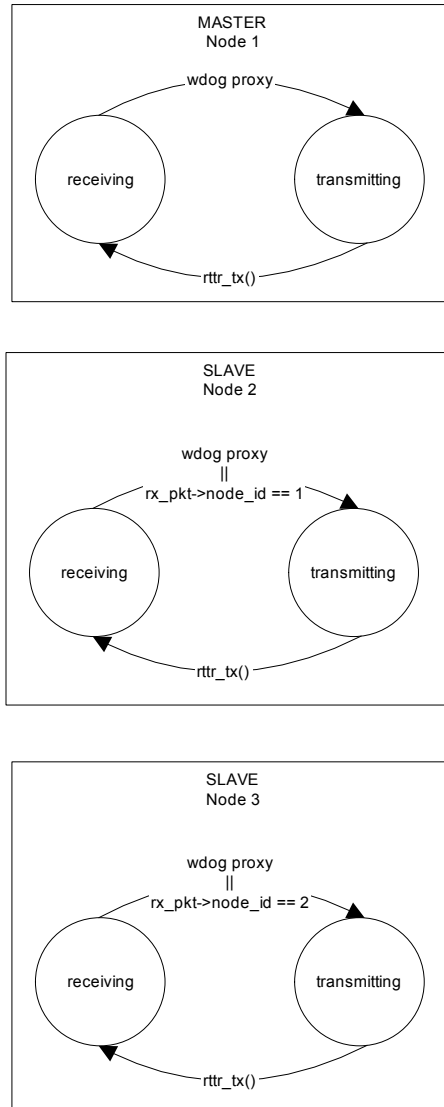


Figure 3.2.15 Example RTTR ring topology

3.2.5.4 Experimental Results

Validation and performance testing of the wireless communication system were conducted at several test facilities, including the Richmond Field Station, Crow's Landing, and I-15 in San Diego. The entire communications system is composed of an Orinoco 802.11b wireless card, a Hypergain omnidirectional antenna (HG2408U), and a Hyperamp 1 Watt AGC amplifier (HA2401-AGC1000).

To demonstrate the typical performance of the wireless communications system, results for high-speed tests of a two-vehicle platoon on I-15 in San Diego will be presented. The first set of plots in Figure 3.2.16 below shows the occurrence of lost packets and the round-trip time for the token as a function of elapsed time during the run.

As seen in the top plot, a minimal number of packets are lost during the run, and the average round trip time of the ring is around 3 milliseconds. A histogram of the round trip time is shown in Figure 3.2.17.

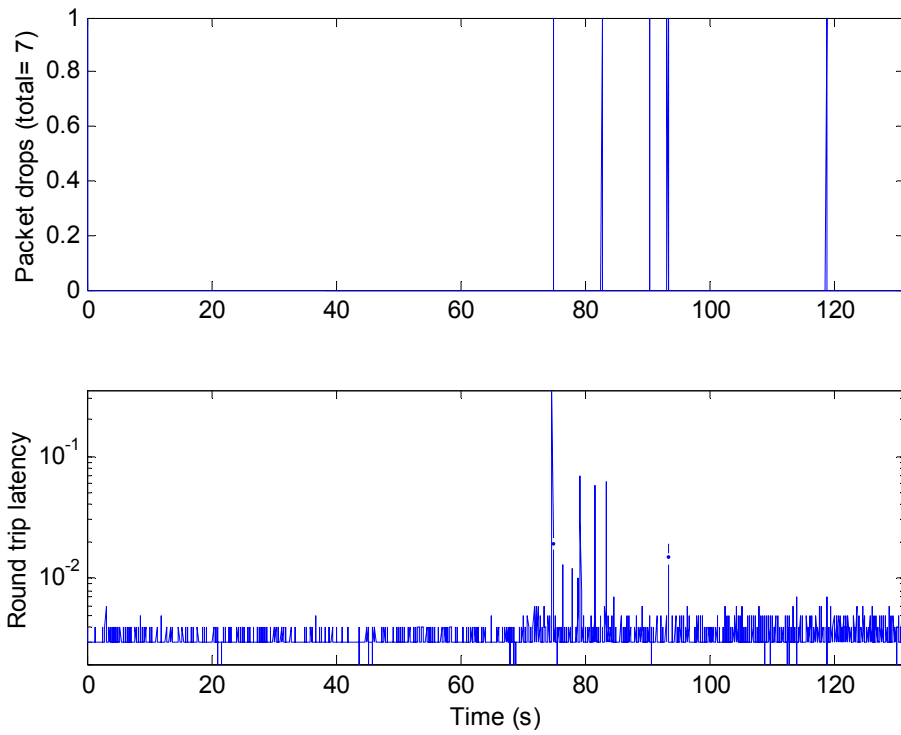


Figure 3.2.16 Communication Performance During an Experimental Run on I-15

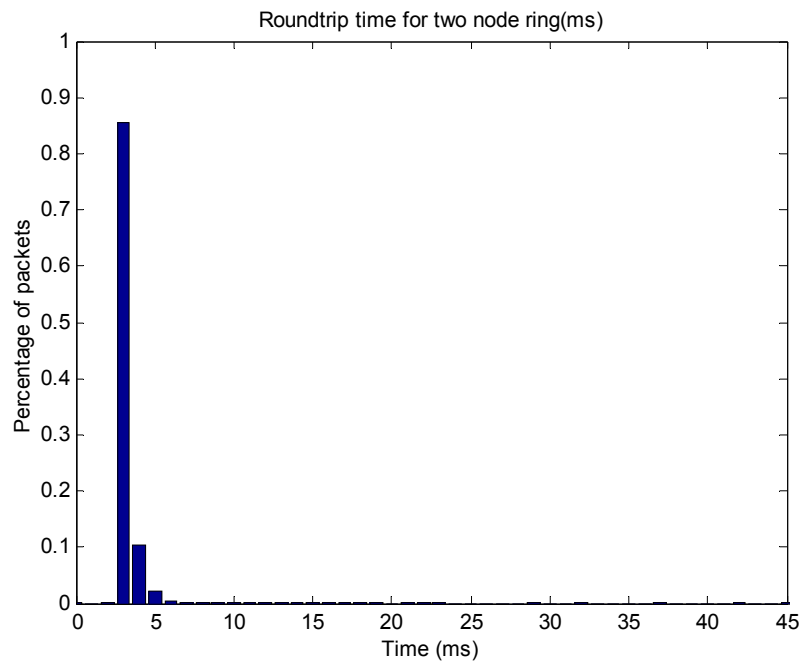


Figure 3.2.17 Histogram of Round-Trip Time

To give an indication of the reliability of maintaining the 21 millisecond update time for the communicated variables, a histogram of the latency between database updates for a given node is shown in 3.2.18.

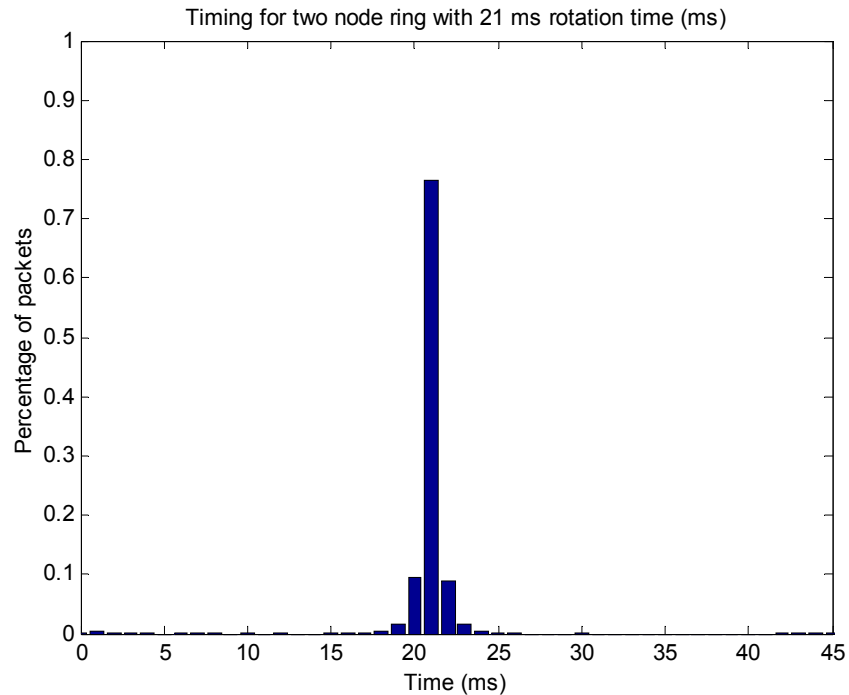


Figure 3.2.18 Histogram of Database Update Latencies

3.2.5.5 Conclusions on Wireless System

The implementation of the RTTR protocol on top of the Orinoco 802.11b driver in QNX 4 provides a high-performance wireless communication system suitable for real-time application. Although the protocol does not easily scale to large numbers of nodes, its simplicity and flexibility for small numbers of nodes makes it ideal for development and prototyping environments. For longer-term application of vehicle-to-vehicle communication, further research and development of WTRP is suggested because of the protocol's superiority in terms of dynamically changing the ring topology and providing transparent TCP/IP services to user applications.

3.2.6 Range and Range Rate Measurement Sensors

An essential part for longitudinal control of automated transit buses is the measurement of the range and range rate between vehicles. In realistic operating conditions, the quality of these measurements can be highly variable due to the time-varying environment, the operating principles of the sensors, and the potential for sensor

failures. One means of improving both the reliability and accuracy can be obtained through the use of multiple sensors in conjunction with sensor fusion and fault diagnostic techniques. Although the design of the sensor fusion and fault diagnostic systems are commonly discussed independently in the literature [1-3], it is quite natural to consider the combination of two systems since they have both complementary goals and methodology. Recent interest in the development of distributed estimation and signal processing in wireless sensor networks has fostered similar research [4,5].

To obtain the measurement of the range and range rate for the automated transit buses, a *virtual range rate sensor* was created by combining local sensor measurements and a wireless communication link mentioned above between vehicles is fused with measurements from a radar and lidar. The radar mounted on the experimental vehicle is an Eaton VORAD EVT-300 24 GHz Doppler radar. This radar is a component of a commercial adaptive cruise control (ACC) system developed primarily for heavy vehicle applications. The radar has a 12 degree beam width, and a range of 100 m. It tracks up to seven simultaneous targets, providing the range, range rate, and azimuth of each target at a sample rate of 65 (msec) [6]. A 2D scanning lidar manufactured by Denso is used for automotive ACC applications as well. The lidar has a 16 degree lateral field of view, a 4.4 vertical degree field of view, and a range of 120 (m). It can track up to eight simultaneous targets, and provides a wealth of information about each target at a sample rate of 100 (msec). The third measurement of the relative speed is computed by a virtual range rate sensor composed of the difference between the following vehicle speed with that of the preceding vehicle. The preceding vehicle information is obtained via wireless communication, while the vehicle speed is measured directly using the vehicle stock wheel speed sensors.

The sensor fusion is conducted by using a sequential variant of the nonparametric probabilistic data association filter (NPDAF) with validation gating. Fault diagnostics are incorporated into the sensor fusion by thresholding the Mahalanobis distance computed in the validation stage. Verification of the proposed integrated sensor fusion and diagnostic (ISFD) system was conducted using experimental data obtained from a 40-foot and a 60-foot New Flyer transit buses. All mathematical explanations in detail can be found in the published paper [7].

Detection range	0-120m
Detection angle	40 deg (lateral, ±20deg)
Detection angle	4.4 deg (elevation)
Update rate	100 ms
Laser wave length	850 nm
Laser beam size	0.2 deg (lateral), 0.9deg (elevation)
Number of detection points	265 (lateral), 6 (elevation) total: 1590points/cycle

Table 3.2.1 Lidar specifications

3.2.7 Range and Range Rate Sensor Fusion

Verification of the integrated sensor fusion and diagnostic (ISFD) system was conducted using experimental data obtained from both 40-foot and 60-foot New Flyer transit buses at both Crow's Landing and I-15 in San Diego. The following plots give representative results of the sensor fusion algorithms' performance.

Figure 3.2.19 shows the range and range rate measurements of the primary Lidar target during a typical experimental run on I-15 in San Diego, CA. The estimate produced by the sensor fusion algorithm is overlaid on the plots. As is easily seen in both plots, the fused estimate tracks the lidar measurements quite well, with a slight reduction of the high frequency noise present in the raw signal. The sensor measurements are quite clean because an additional prefiltering stage was added to ignore targets that have an indicated width different from that of the bus (a range of 2 - 4 m was used). The target width is an additional measurement provided by the lidar's internal signal processing, and the estimated width during this run is shown in Figure 3.2.20. This additional stage was added to reduce false returns from overpasses, reflective signs, and guard rails that were encountered during testing.

The Radar sensor measurements for the same run are shown in Figure 3.2.21 and Figure 3.2.22. As shown in both of these plots, the radar tends to have numerous dropouts and false returns throughout the run. Many of the dropouts are due to the low relative velocity between the vehicles, since the radar loses any target with a relative velocity less than 0.1 m/s. The false returns were due to the numerous roadside objects that were present in the environment. Although the Radar performance is poor while the following vehicle is maintaining a constant spacing from the lead vehicle (at $t > 30$ s in these figures), the sensor measurements are quite good during the initial transient period when the follower joins the platoon.

To give further indication of the performance of the sensor fusion algorithm, the number of targets detected and validated by each of the sensors is shown in Figure 3.2.23.

Conclusions

The performance of the sensor fusion algorithm was verified and demonstrated in the real-time application of heavy vehicle control. The fused estimates were shown to be tolerant to dropouts and false returns in both the radar and lidar. Furthermore, the estimates are of reasonable quality and inherently account for multiple possible targets.

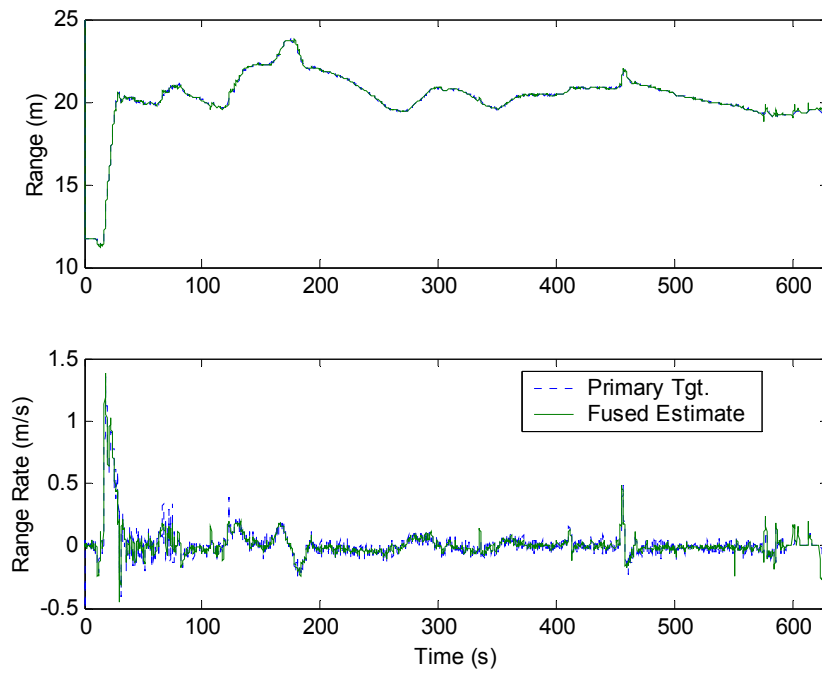


Figure 3.2.19: Lidar Measurements for the Following Vehicle on an I-15 Experimental Run

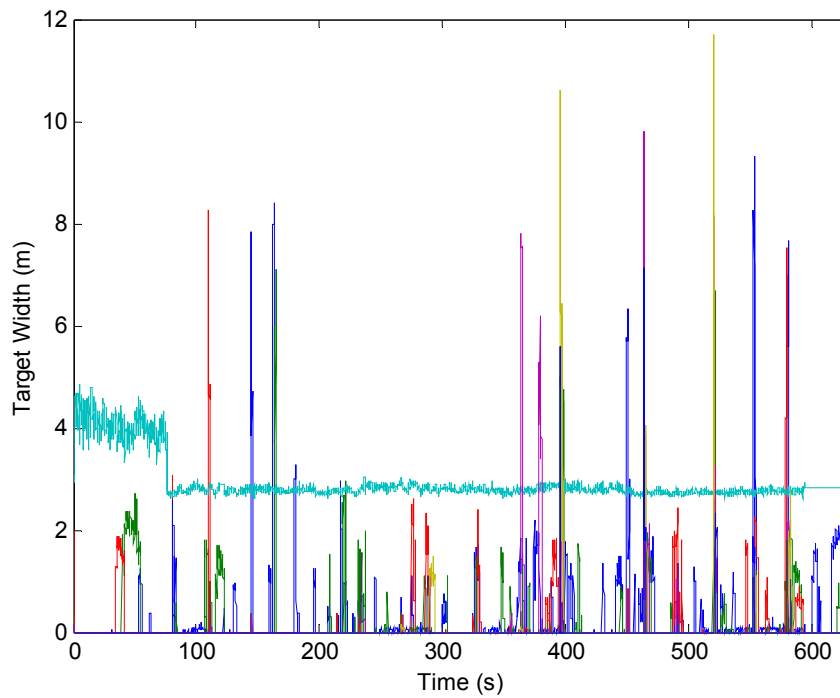


Figure 3.2.20: Target Width as Indicated by the Lidar Internal Signal Processing

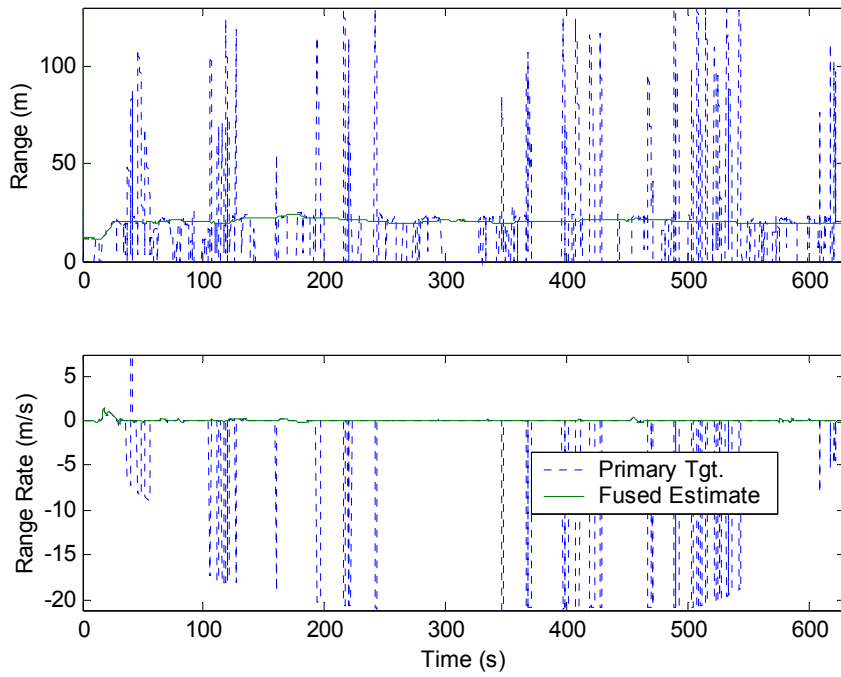


Figure 3.2.21: Radar Measurements of the Following Vehicle on an I-15 Experimental Run

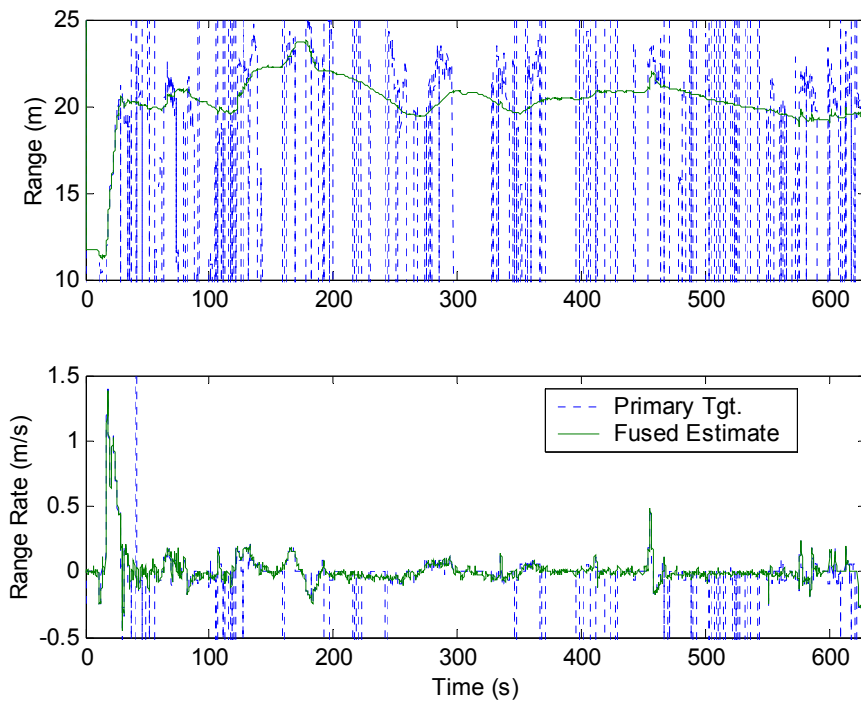


Figure 3.2.22: A Close-up View of the Radar Measurements

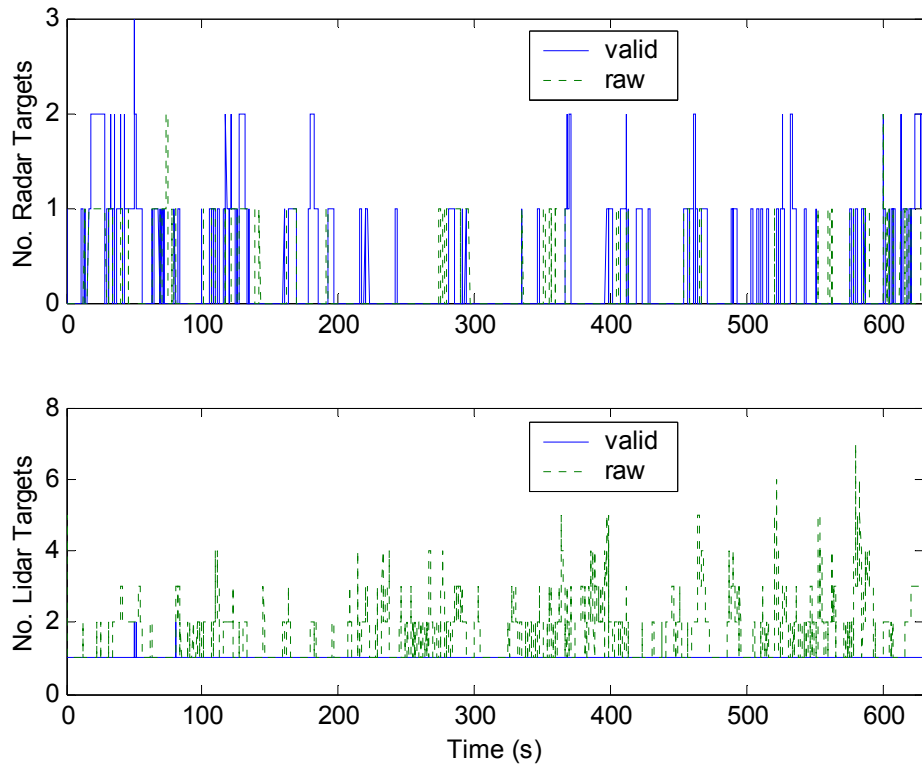


Figure 3.2.23: Number of Targets Detected by both the Radar and Lidar

3.2.8 Brake Actuation System

The brake actuation design considered two main options, one operating in parallel with the pedal, and the other a wheel by wheel design. Wheel by wheel brake application would reduce the braking response time by eliminating most of the delays in the existing pneumatic braking system, but would not incorporate the extensive safety backup and ABS systems available with the stock braking system. It was decided to install an automated braking system in parallel with the brake pedal that would be equivalent in response to a rapid manual brake and release, thus preserving all existing backup, safety and ABS system designs.

Commercial off-the-shelf (COTS) air brake parts were used exclusively to ensure safety and reliability, reduce costs, and speed implementation. Three- and four-way manifolds were installed at various points along the existing brake system to supply air for the automated system, to return command line pressure to the existing system, and to add air pressure transducers along the braking system.

Pressure transducers were placed on both the front and rear braking systems to measure brake pressure at the brake reservoir, the pedal command line, and at each wheel.

The locations of transducers along the brake pneumatic path allowed for rapid characterization of the existing system response necessary for controller algorithm development. This also provided a benchmark for comparison with automated brake system design. An additional pressure transducer signal was available between the electro-pneumatic and the pneumatic volume booster sections of the automated braking system.

The automated braking command unit selected is a COTS unit supplied by Proportion-Air Inc. These units consist of a QB series proportional valve mated to a PSR volume booster. The following is the supplier's description of Proportion-Air Inc. QB series valves:

The QB series valve uses closed loop technology for pressure control. It gives an output pressure proportional to an electrical command signal input.

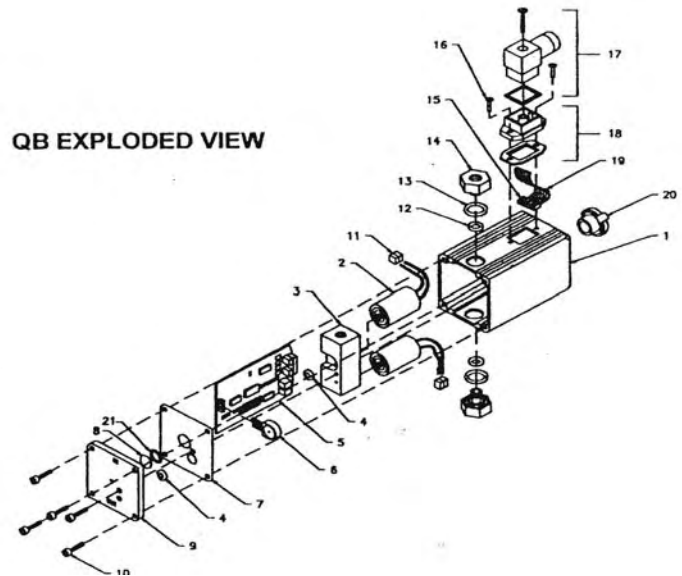
The QB1 is a complete closed-loop servo system consisting of valves, manifold, housing, and electronic controls. Pressure is controlled by the use of two solenoid valves. One valve functions as inlet control, the other as exhaust. The pressure output is measured by a pressure transducer internal to the QB1 and provides a feedback signal to the electronic controls. This feedback signal is compared with the command signal input. A difference between the two signals causes one of the solenoid valves to open, allowing flow in or out of the system. Accurate pressure is maintained by controlling these two valves.

PARTS LIST

1. Housing
2. Valve (2)
3. Manifold
4. O-ring (2)
5. Electronic board
6. Sensor
7. Gasket
8. Filter/Breather
9. Lid
10. Screw (5)
11. Connector (2)
12. O-ring (2)
13. O-ring (2)
14. Fitting (2)
15. Connector
16. Screw (2)
17. Connector assembly
18. Receptacle assembly
19. Wire harness
20. Cap
21. O-ring

PART NUMBERS FOR REPLACEMENT ITEMS*

2. Consult factory
4. H134
6. Consult factory
7. H1054
10. H1049
12. H2014
13. H011
14. H1048
17. H615
20. H368
21. H040



* Include complete QB part number & any modification numbers when ordering replacement parts

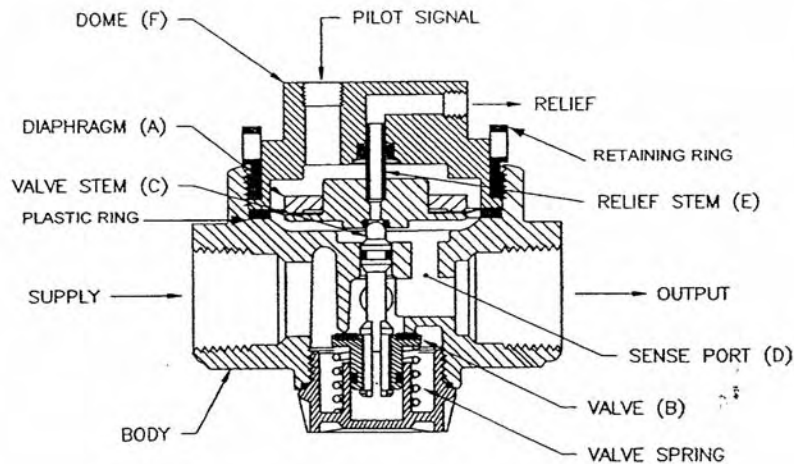
Figure 3.2.24 – Air brake PSR volume booster

Figure 3.2.24 is a cut-away view of the PSR volume booster with an explanation of its function. Proportion-Air Inc. has added an input manifold for the pilot signal with a minimized volume and optimized flow design to decrease response time.

Refer to figure for ports location. With pressure supplied to the volume booster supply port and no pilot signal, valve (B) is closed. Pressurizing the pilot port applies a load to diaphragm (A). This load causes diaphragm (A), valve stem (C), and valve (B) to move downward allowing flow across the seat area between valve (B) and the body. Pressure in the output line is sensed below diaphragm (A), via the sense port (D), and offsets the load on diaphragm (A). As output pressure rises, valve (B), valve stem (C), and diaphragm (A) move upward until the area is closed causing the pilot pressure load on diaphragm (A) and pressure under diaphragm (A) to be in balance. A reduced output pressure has now been obtained.

Creating a demand downstream results in a reduced pressure under diaphragm (A). The load on diaphragm (A) now causes the valve stem (C), and valve (B) to move downward opening seat area and allowing air to flow to the output. The flow of output air is metered by the amount of opening. During low flow requirements, the amount of opening at the seat is small, while at high flows it is large. The output pressure signal requires correction over the flow range, in order to give a constant pressure output when under flow conditions. This adjustment is caused by sense port (D) which is located in such a manner as to provide compensation to the output pressure signal transmitted to diaphragm (A). This effect is called aspiration. The purpose is to maintain output pressure nearly constant over a wide range of flow demands.

Should downstream pressure exceed the desired regulated pressure, the excess pressure will cause diaphragm (A) to move upward, opening the vent hole sealed between diaphragm (A) and valve stem (C), venting the excess pressure to atmosphere through the relief stem (E) and out the relief port on the dome (F).



The first units tested had a good brake-apply response, equal to a fast hard manual brake application, but were much slower in releasing brake pressure than achieved with manual braking. A faster brake release response is necessary to allow for tight longitudinal control of brake application during platooning maneuvers. Analysis of pressure data indicated that the bottleneck in the system was at the PSR volume booster. A larger flow PSR unit was installed and release response was greatly improved.

Initial testing indicated that with loss of power to the QB1 unit the brake command would stay fixed at the last commanded pressure. It was decided to redesign the unit to include a vent to atmospheric pressure at power off to increase safety during testing.

A COTS double check valve was installed on both the front and rear existing brake command lines to allow both automated brake application pressure and manual brake application pressure to be applied to the braking system. The higher of the two brake pressure commands would control the brake system.

The installations of the automated brake units were restricted by space around the brake pedal pressure application unit. Locations were picked to minimize pneumatic hose length, and thus minimize air volume in the brake command system. Care was taken to limit flow restrictions when considering unit placement.

3.2.9 Engine Control System Interfaces

Engineering design for the engine control of the research busses was divided into two tasks. The C-40 bus CNG engines were controlled through the existing analog demand signal and driver gas pedal activation switch. The switch was interrupted and an analog command signal between 0.5 and 4.5 volts was applied to the interface circuit. An isolation diode was placed in series with the command line to separate the manual command from the automated command. The interface circuitry was placed close to the pedal under the dashboard.

The diesel engine on the 60' bus was controlled by way of messages sent on the J1939 CAN interface. A dual CAN PC104 card was installed in the control computer and the throttle was controlled by means of speed and torque command messages sent from the control computer to the standard engine control unit (ECU) already installed on the bus.

3.2.10 Driver-Vehicle Interface

The term driver-vehicle interface (DVI) refers not only the physical switches, status LEDs, or graphic displays that were added to the vehicles, but refers also to the more global concept of information flow between the driver and the vehicle and how transitions between vehicle states occur. Under normal driving circumstances, the driver constantly receives information about the vehicle by watching the road, proprioceptively sensing the steering wheel angle, listening to the engine and road noise, and sensing vehicle accelerations. Drivers then use that information in a feedback loop comparing it to the commands that are being given on the steering wheel, gas pedal, and brake. Once the control of the vehicle becomes automated, the driver still receives the output or visual information from the road, but loses any sense of the input side of the equation or what the vehicle was commanded to do. Thus, the display aspect of the driver-vehicle interface attempts to replace the missing input in the driver feedback loop by providing answers to the following three questions:

1. What is the current status of the vehicle or what does the vehicle think that it is doing?
2. What has the vehicle been commanded to do now or in the near future?
3. How can the driver effect a change in the system?

3.2.10.1 *DVI Physical Components*

System Overview and Driver Controls

Figure 3.2.25 shows a diagram of the physical components that made up the DVI. These components can be categorized as either driver inputs or vehicle status displays. There were three driver inputs added to the vehicle, a kill switch, a transition switch, and six

multifunction buttons along the right side of the graphical display unit. Three vehicle status displays were also added to the vehicle, an array of four LEDs (green, blue, yellow, and red) on the instrument panel, a graphical VGA computer display unit, and a speaker to provide audio.

The kill switch was a standard IDEC Type AYW401-R push-pull kill switch, typically used for automotive and industrial applications. Although a commercial system would not likely provide the driver with a kill switch, safety concerns with the prototype nature of the system dictated that the driver have a quick, simple, and reliable way of disengaging any automated systems in the event of a problem.

The transition switch was a typical three-position, momentary-action, rocker switch, which measured 15 mm wide by 25 mm long. The center position was neutral. Pressing the switch forward sent a request to the control computer to engage any automated system that was ready to be engaged. When the rocker switch was released, it returned to the neutral position. Pressing the switch backwards or to the rear sent a request to the control computer to disengage any automated systems that were currently engaged. This type of switch and method of interaction was selected since it had already been used successfully for the automatic steering control system of the Advanced Rotary Plow.

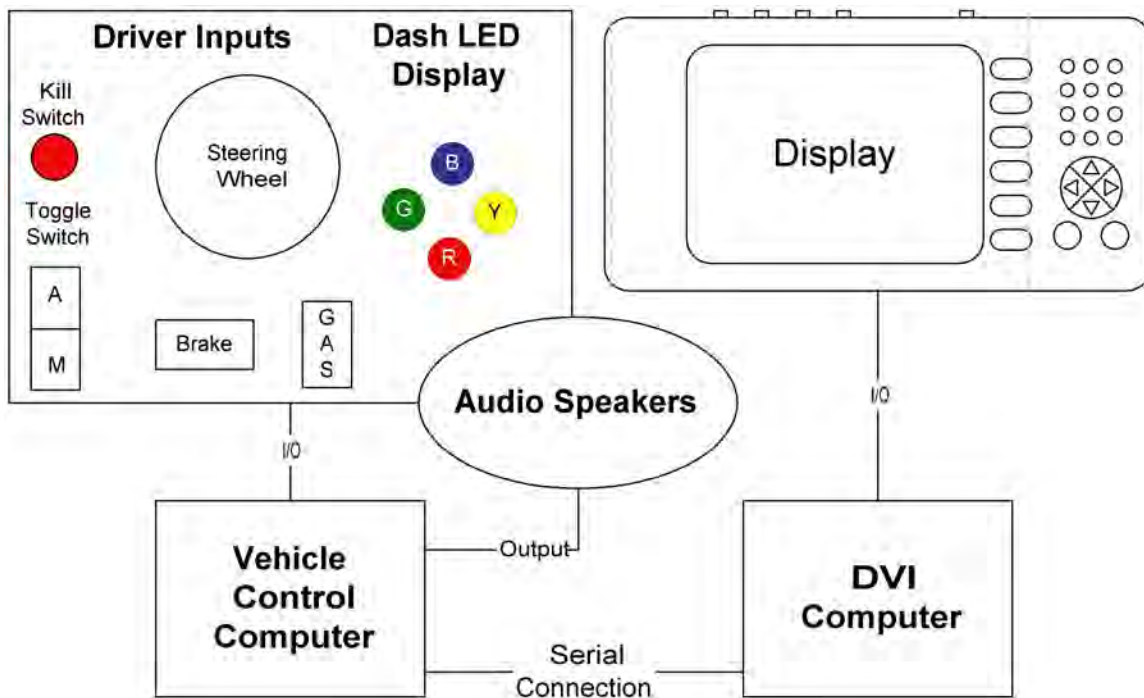


Figure 3.2.25 Driver-Vehicle Interface Component Diagram.

The transition rocker switch and the kill switch were both mounted on a plate in-between the left side of the driver's seat and the control panel to the left of the driver as shown in Figure 3.2.26. This placed both the kill switch and the transition switch within easy reach

of the driver's left hand. The control size, distinct shapes, and separation also allowed for easy activation without glancing down to the control panel.

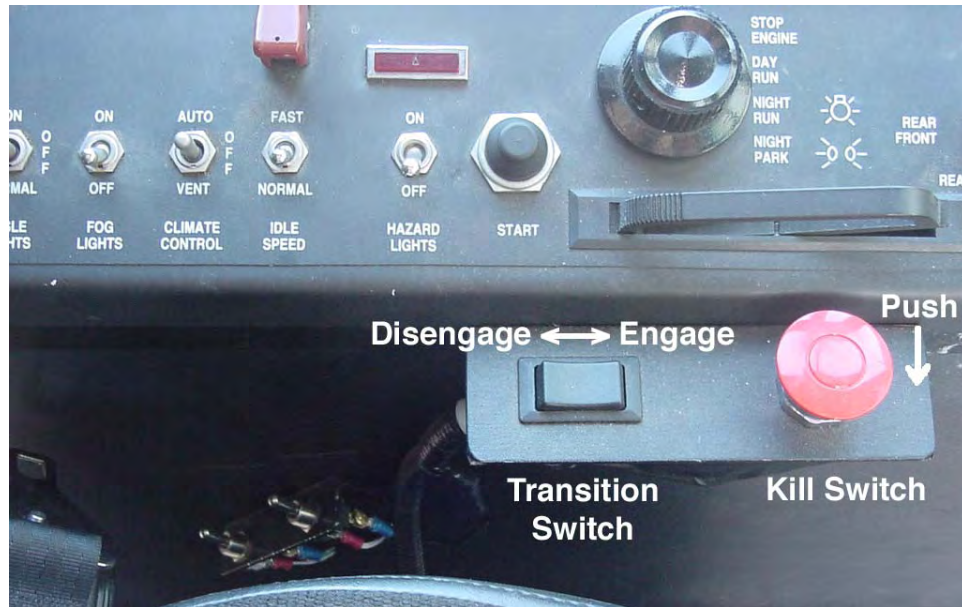


Figure 3.2.26 Transition Switch and Kill Switch Location.

Instrument Panel LED Display

The instrument panel LED display, Figure 3.2.27, consisted of four colored LEDs mounted on the instrument panel to the right of the steering wheel, below the speedometer, and arranged in a diamond pattern to provide easily perceptible coding through position as well as color. The general color meaning is described in Table 3.2.2, and the actual LED states are described in Table 3.2.3. Each LED was an LEDTRONICS PF50-T, sunlight-visible, panel mount unit with a lens diameter of 15.8 mm and a viewing angle of 12-15 degrees. The dashboard LED display was connected directly to one of the digital I/O boards of the vehicle control computer. This setup was chosen for several reasons. First, the dashboard LED display was meant to provide a simplistic overview of the system state, whereas the graphical display provided more detailed information. Second, the LED display was implemented for redundancy in case of a DVI computer failure. Finally, the LED display provided faster, nearly instantaneous updates, of the system state, whereas the architecture for the graphical display contained several potential bottlenecks.



Figure 3.2.27 Dashboard Four-Color LED Display Located Below the Speedometer.

Table 3.2.2 Color Stereotypes, Common Automotive Uses, and DVI LED Use.

Color	Common Stereotypes	Common Automotive Uses	DVI LED Use
Green	General On/Off Go OK, Good	Turn signals, some indicator lights, and cruise control	Ready for the transition to automatic control
Blue	General On/Off	High-beam headlight and air conditioner indicators	Automation is engaging or is in control
Amber Yellow	General On/Off Slow Caution	Check engine, low windshield washer fluid, fog lights, traction control, rear window defroster, and cruise control	Speed fault or speed control disabled and end of magnets approaching
Red	General On/Off Stop Warning or Failure	Warning lights and icons for seat belts, door ajar, oil pressure, temperature, etc.	System failure or driver has overridden the automation

Table 3.2.3. LED Display State Table.

Event	Green	Blue	Yellow	Red
System Start-up	Test	Test	Test	Test
Manual Driving or Automation Not Ready	Off	Off	Off	Off
<i>Transition to Automation</i>				
Automation Ready	Solid	Off	Off	Off
System Failure	Off	Off	Off	Solid
Transfer in Progress	Solid	Blinking	Off	Off
Transfer Complete (Automation On)	Off	Solid	Off	Off
Transfer Failed - Manual Driving	Off	Off	Off	Off
<i>All Automation Modes - Overrides and Faults</i>				
Driver Overrides Steering Initially	Off	Solid	Off	Blinking
Driver Overrides Steering for 2 seconds	Off	Off	Off	Off

Kill Switch Depressed	Off	Off	Off	Solid
Lateral Control System Failure	Off	Off	Off	Solid
<u>Precision Docking</u>				
Driver Pressed Brake - Disengages Speed	Off	Solid	Solid	Off
<u>Automated Driving</u>				
Approaching End of Magnets	Off	Solid	Blinking	Off
Lidar Target Lost Fault	Any	Any	Blinking	Off

Graphic Multifunction Display Unit

The graphic multifunction display unit was a custom made box containing a 6.4” VGA LCD display with a maximum resolution of 800x600 pixels, and it was connected to the graphics output of the DVI computer using standard VGA monitor cables. The display produced about 420 Nits of brightness. For reference, 1000 Nits is considered sunlight visible for an LCD and 200 Nits is typical for a bright contemporary laptop display. Although the display was situated inside a vehicle and not in direct sunlight, there were concerns about glare and indirect sunlight washing out the display given the large windows in the front of the bus and the relatively unprotected enclosure. During the daytime testing, the 420 Nit displays provided for an exceptional display and there were no issues with glare or lack of brightness. If the displays were more shielded from the sunlight, such as being embedded in the dashboard, or only used at night, then a 200 Nit display would have sufficed.

The multifunction display was mounted to the right of the driver and the instrument panel, but still within arms reach, as shown in Figure 3.2.28. The display height was kept low enough such that it did not interfere with the driver’s forward view out the windshield. The location also kept the display within about 30 to 35 inches of the driver’s eye, which is fairly typical for automotive dashboard viewing distances. All text (character heights) were kept well above the 18 minutes of visual angle which is the recommended minimum for in-vehicle applications, making all of the text used on the VGA display easily readable.



Figure 3.2.28 Graphic Multifunction Display Unit.

The graphic display unit also utilized six, E-Switch Series 320, rectangular pushbuttons along the right side of the display. These buttons were attached to a digital I/O card of the DVI computer. The buttons were approximately 10x20 mm in size with an average center to center spacing of 25 mm, well within the recommended size and spacing guidelines for automotive switches. The button functions varied corresponding to the menu items shown on the graphic display. These buttons were used for tasks that did not involve transitioning in and out of automatic control such as changing settings, selecting the testing scenario, and requesting a lane change.

Auditory Warnings

Auditory warnings (or sounds) provide an extremely powerful means to get the driver's attention, since sound is independent of where the driver's eyes are currently focused. However, this also means that the driver can't ignore them, and the use of too many sounds can quickly become distracting or annoying. Additionally, the association between a warning sound and its meaning is often abstract, and thus, the use of too many sounds can become confusing for the driver, especially given that there are already multiple warning sounds on buses. In keeping the number of auditory warnings to a minimum, it was determined that only three events required an auditory component as described in Table 3.2.4.

Of the three sounds, the automation engaged sound was the least necessary and probably would not be used in a real-world implementation; however, the alert was useful during

testing as a confirmation that the automated systems had taken over. The transition ready sound was also not always necessary, and in fact, was only required when the system was set to automatically transition to automated control once the vehicle detected the magnets. This mode was used in the precision docking scenario due to the short test track, which only allowed for about 2.5 seconds of warning before the automated system engaged. In practice, this was a little too short and future implementations should allow at least a full 3 seconds of warning for the driver to cancel the transition.

Table 3.2.4 Events With an Associated Sound

Event	Dashboard LED	Meaning
Transition Ready	Green	Signaled that the green LED had illuminated and the system was ready to transition to automated control. When the automatic transition option was enabled, this sound was played three times with a half-second interval before the automated systems engaged, allowing the driver about 2.5 seconds to cancel the impending transition.
Automation Engaged	Blue	Signaled that the blue LED had illuminated and that the transition to automatic control was successful.
System Failure End of Magnets	Red	Signaled that the driver needed to take over vehicle control immediately due to a system failure or driving off the magnets. The sound consisted of three short bursts followed by a half second pause and repeated continuously until the driver shut off the system.

3.2.10.2 Vehicle Modes and Transitions From the Driver's Perspective

The system as implemented had 3 scenarios from the driver's perspective, precision docking, lane assist, and automated driving. As shown in Figure 3.2.29, each scenario had multiple modes and each mode could have multiple maneuvers. Deciding what exactly constituted a mode or maneuver from the driver's point of view was an iterative process. The distinction between a mode and maneuver was never clearly defined. However, the designation of a mode generally indicated that a transition or fundamental shift in how the vehicle behaved had taken place. A maneuver, on the other hand, indicated that the vehicle was operating in a similar manner but some parameter had changed. As an example, a vehicle transitioning from cruise control to active following was classified as a mode change, but opening or closing the gap was only considered a maneuver since the only change was the vehicle's desired gap clearance.

Although an attempt was made to describe the modes in generic terms scalable to future conceptual systems, Figure 3.2.29 reflects the actual needs of the implemented system in the context of the testing that was being performed, which produced a multitude of artifacts. As an example, the first step required by the driver was the selection of a

driving mode (docking, lane assist, or automated driving) so that the system would know which scenario it should prepare and what actions would be mapped to the rocker transition switch. In a real world implementation, the system would not require this action as it would know where the vehicle was and what automated functions would currently be available.

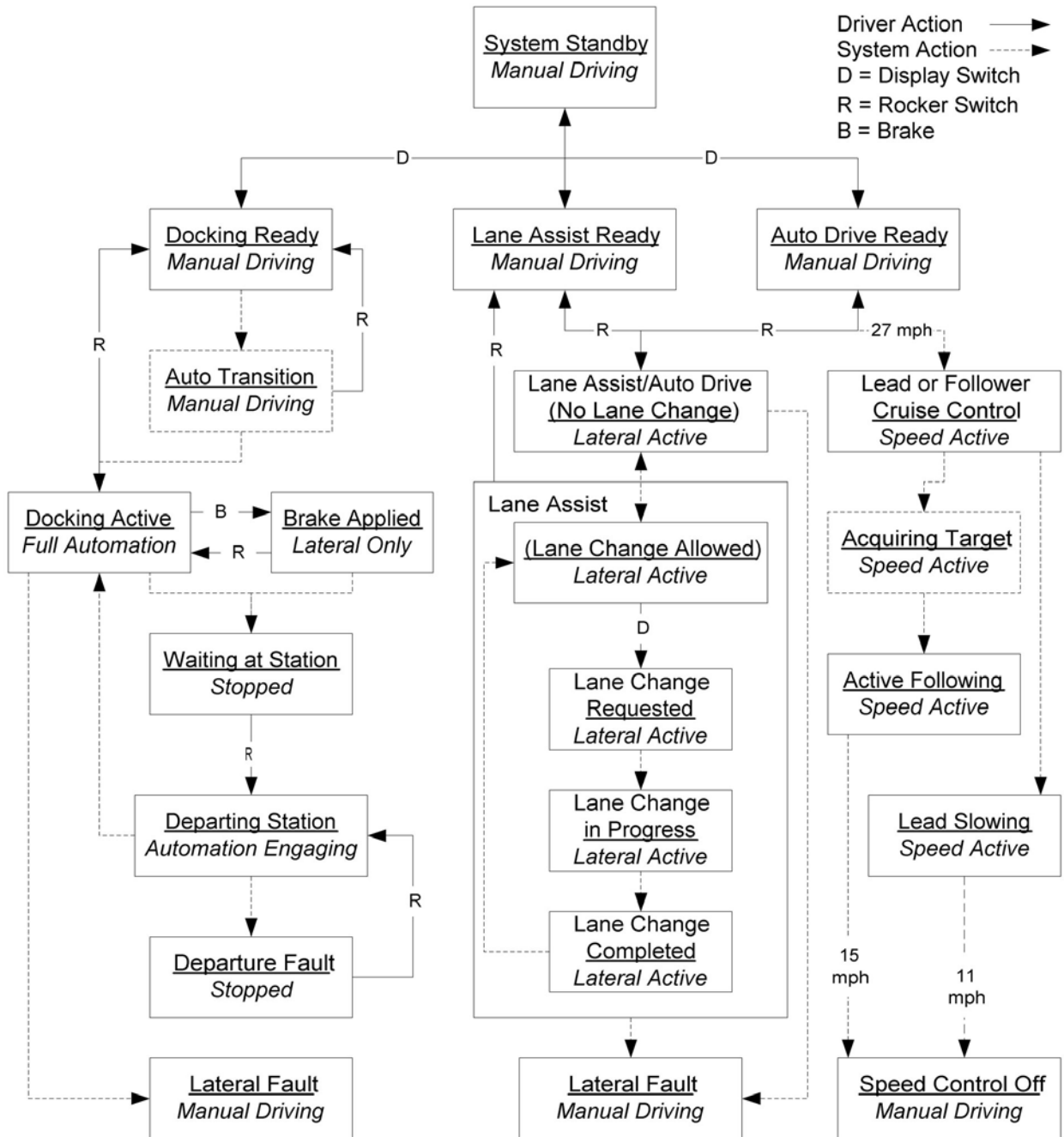


Figure 3.2.29. Vehicle Modes and Transitions from the Driver's Point of View.

3.2.10.3 Graphical Information Display

The graphical information display was composed of 5 different areas as shown in Figure 3.2.30. The top of the screen was dedicated to providing the best description of the current driving scenario or mode. The right side of the screen was dedicated to providing menus of functions accessible by pressing the corresponding physical button next to the menu item. The bottom of the screen provided three different status displays: one for the radar and lidar, one for the wireless communications, and one for the lateral control system. The left side of the screen was dedicated to either a forward collision warning display or gap information if the vehicle was in an active following mode. The center panel of the display was used to provide detailed scenario specific information, and thus changed depending on the scenario, driving mode, and maneuver.

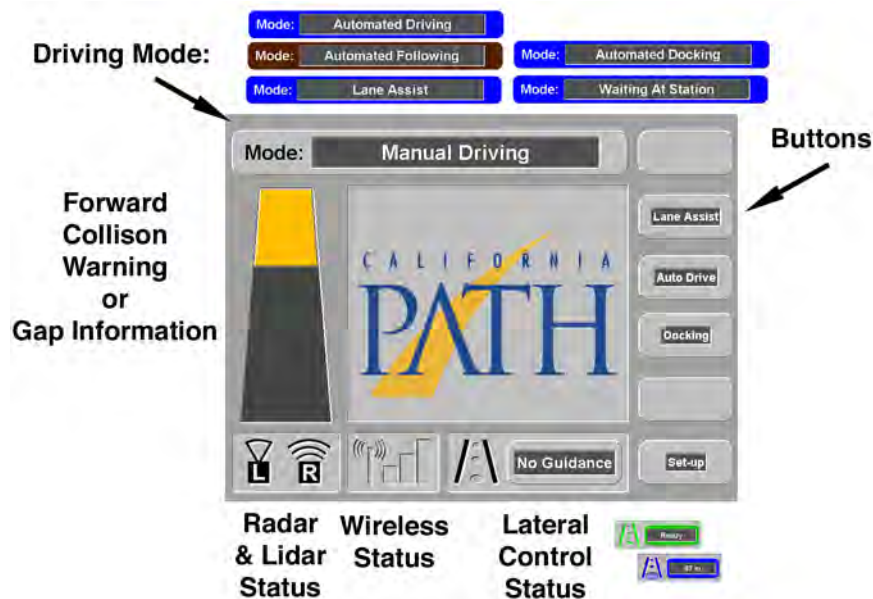


Figure 3.2.30 Graphical Information Display Overview.

With this basic layout, the top, bottom, and right sides of the display were kept constant in their format regardless of the scenario or driving mode, giving the display a consistent and familiar feeling in all modes. The left side of the display, although it did change depending on the driving mode, was always associated with the forward sensors and vehicles in front of the driver. This use of consistency kept screen changes from appearing too abrupt, and allowed the driver to quickly find information regardless of the current driving mode.

Color use in the graphic display was kept as consistent as possible with the color use on the dashboard LED display. Green was used for the lateral control status information, in conjunction with the green LED on the dashboard, to indicate that the system was ready for a transition to automated control. Blue indicated that the automated control systems (specifically lateral control) was currently active. Since the amber LED had multiple meanings, the graphic display often provided clarity. As an example, when the amber

LED indicated the end of magnets nearing, the lateral control status display used amber colors, whereas when the LED indicated a Lidar fault, the Lidar status icon would turn amber. Red was used sparingly and only to indicate critical warnings or that a fault had occurred. In all cases color was used only as a means of redundant coding since about ten percent of the population is color blind.

Precision Docking Scenario

As shown in Figure 3.2.31, the precision docking scenario resulted in six typical “screens” as the scenario progressed. The first screen transition occurred after the vehicle detected magnets for a docking maneuver, at which time the center panel informed the driver that a station was approaching, and the lateral control status information indicated that the system was ready for a transition to automatic control. Once under automatic control (either through an automatic transition or by the driver using the rocker switch to request a transition), the lateral status provided a distance countdown in meters to the docking station. Providing any more information than this might result in the potential for driver distraction, which would be undesirable given the amount of pedestrian traffic at the station and the current inability of any vehicle sensors to reliably detect them. Once docked at the station, the display was merely used to provide reminders to the driver to set the parking brake and place the vehicle into neutral, or to release the brake, select drive, and press the transition switch to depart the station.

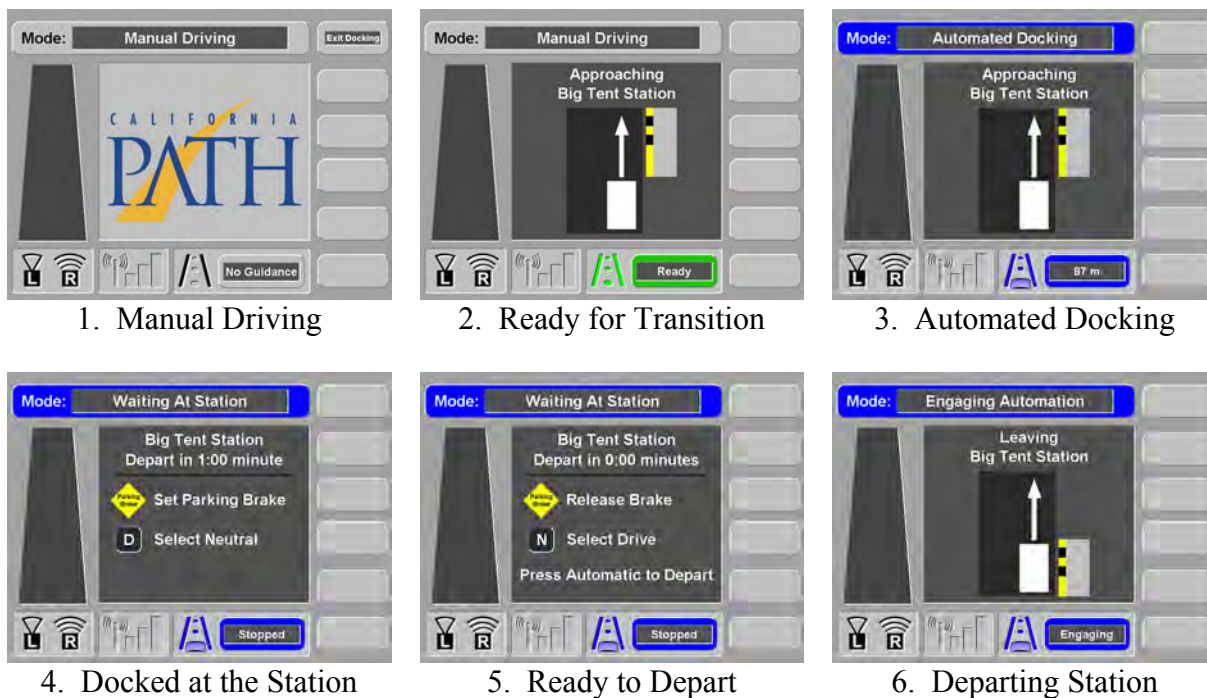


Figure 3.2.31 Typical Precision Docking Screens.

In addition to the normal docking modes, there were two additional modes. First, as shown on the left of Figure 3.2.32, the driver could hit the brake while approaching the docking station, which disengaged speed control placing the vehicle in a lane assist

docking state. Second, the lateral control system might fail completely, resulting in the fault screen shown on the right.

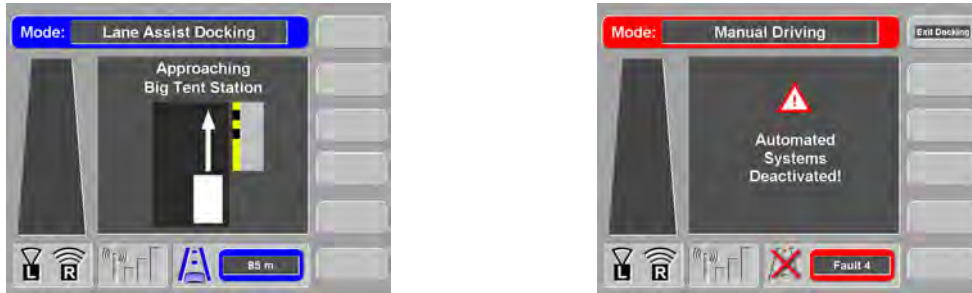


Figure 3.2.32 Docking Driver Override and Fault Screens.

Lane Assist Scenario

During the lane assist scenario, the driving mode and lateral control status portions of the display functioned exactly the same as they did in the docking scenario as shown in Figure 3.2.33. The center panel provided speed, travel direction, an icon for lane ID, and a real-time graph of lane position. With the exception of the lane ID icon and the lane change messages, the majority of the elements displayed on the center panel were displayed to help visitors understand the performance capabilities of the control systems, rather than being for the benefit of the driver. The button shown in the upper right corner of the screen number 3 in Figure 3.2.33 could be used by the driver to invoke an automatic lane change.

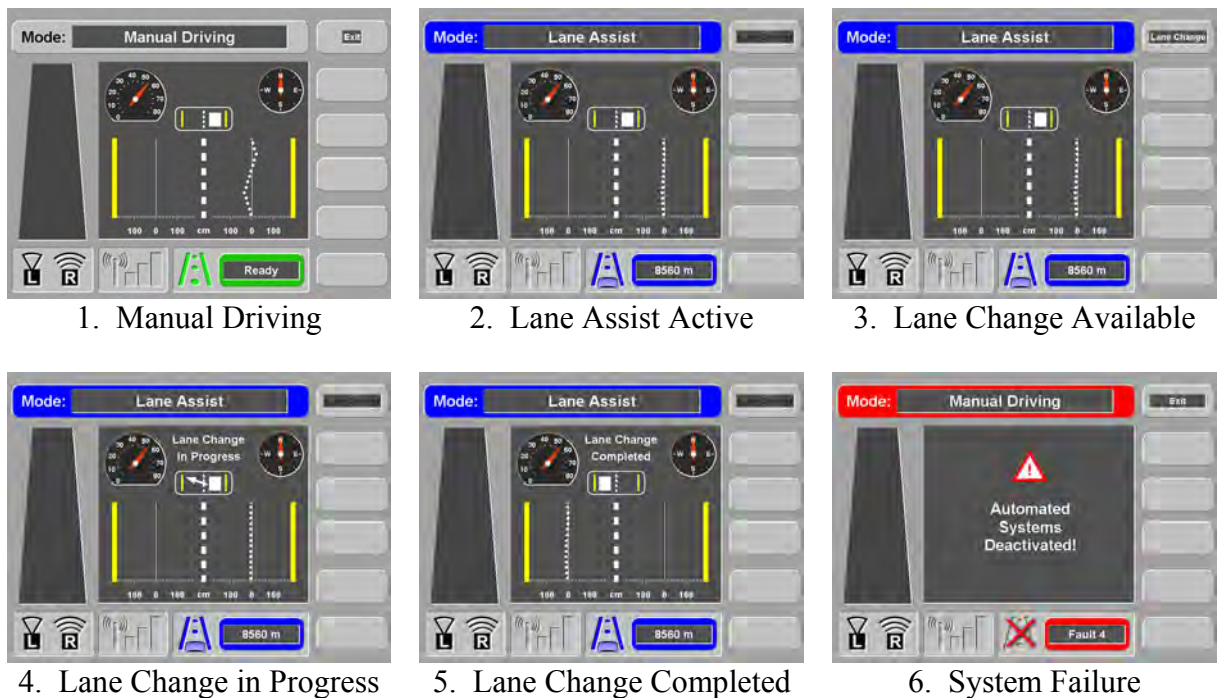


Figure 3.2.33 Typical Lane Assist Screens.

Automated Driving Scenario

The automated driving scenario combined the lane assist functionality with automatic speed and gap control. For lane assist, the most important information that needs to be provided to the driver dealt with transitions between manual and automatic control. Once a reliable lane assist system is active, there is little information that needs to be given to the driver because the vehicle appears to be in a steady state of lane tracking. However, once the speed control functionality is added, there are a host of potential situations where the driver would want clarification about what the vehicle is doing. Events such as the speeding up or slowing down or the closing or opening of the gap with a lead vehicle become very perceptible to the driver and may require explanation. Table 3.2.5 lists some of the information needs that were considered when designing the driver display for the automated driving scenario. The typical progression of screens for an automated driving test is then illustrated in Figure 3.2.34.

Table 3.2.5 Information Needs by Function for the Automated Driving Scenario.

Lane Assist	Speed Control
<ol style="list-style-type: none"> 1. Is the lane assist system in manual or automatic control? 2. Is the system ready for transition to automated lane keeping? 3. Which lane is the vehicle in? 4. What direction is the vehicle traveling? 	<ol style="list-style-type: none"> 1. Is speed control manual or automatic? 2. At what speed will the speed control automatically engage or disengage? 3. Is the system currently tracking a constant speed or a constant gap? 4. Why is the vehicle speeding up or slowing down? What is the current speed vs. the desired or commanded speed? 5. Does the system see that vehicle in front of me? What is the current gap vs. the desired gap or commanded gap?

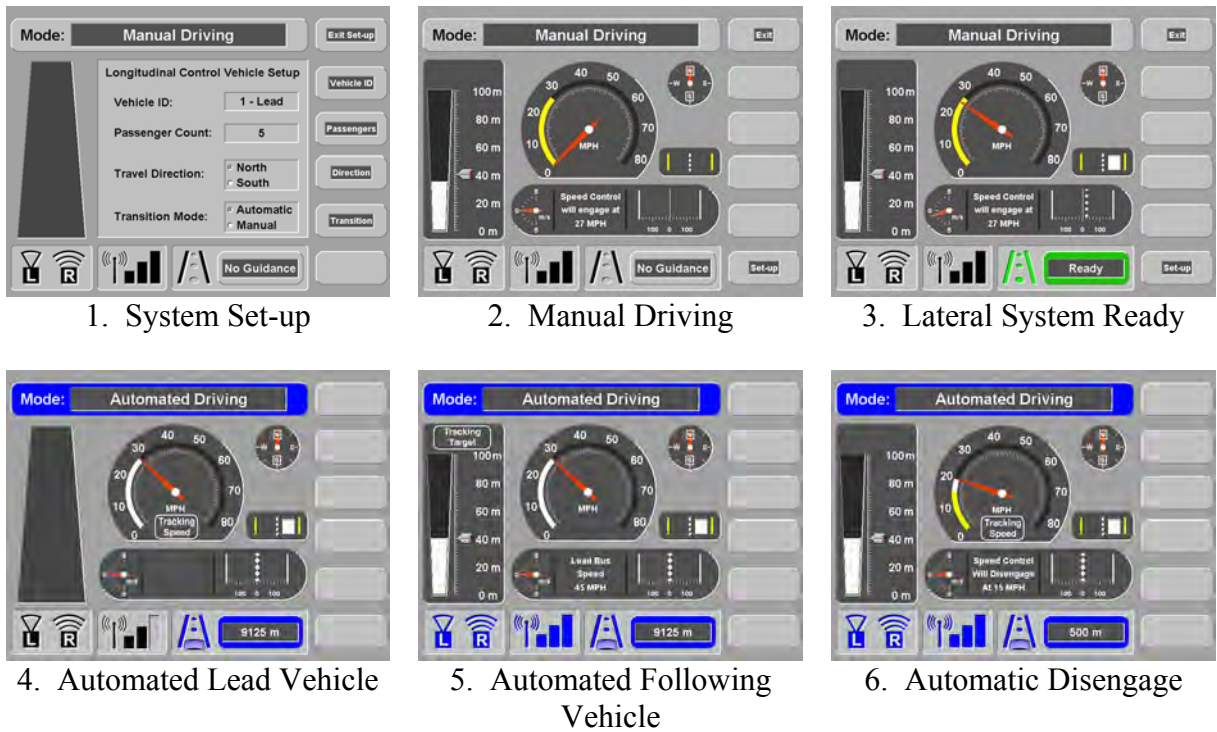


Figure 3.2.34 Typical Automated Driving Screens.

All of the elements from the lane assist screen, the lane ID icon, the travel direction, a graph of the current lane position, and a speedometer were retained. However, the real-time lane position graph was downplayed since this was provided mainly as information for visitors, and the focus of the display was the speedometer in the center panel and the gap information provided on the left side of the screen.

Before the scenario got under way, there was a set-up screen (Figure 3.2.34, Frame 1) which allowed several scenario parameters to be set such as the vehicle ID, the passenger count, and the travel direction. The transition to automated speed control occurred once the driver exceeded 27 mph, which was indicated both in the text message below the speedometer and graphically on the speedometer using a yellow arc (see Figure 3.2.34, Frame 2). Once speed control had engaged, the arc changed color to white (Figure 3.2.34, Frame 4) and represented the desired or commanded speed providing a direct visual comparison between the current and the system-desired speed.

The lateral control system could be activated or deactivated independently of the speed control by the driver using the transition switch, and thus, the lateral control status area of the screen functioned exactly the same as it did during the docking or lane assist scenarios by turning green when a transition was ready (Figure 3.2.34, Frame 3) and blue once the lateral control was active.

With speed control active, the vehicle could be in two different sub-modes, either tracking a speed or tracking the gap to a lead vehicle. This mode was indicated using text icons located in the center of the speedometer (Figure 3.2.34, Frame 4) and above the gap

display (Figure 3.2.34, Frame 5). For the lead vehicle, forward gap information was only given in the form of a forward collision warning display. However, for the following vehicle (Figure 3.2.34, Frame 5), gap information was provided using a bar graph corresponding to the actual gap and a pointer to the right of the graph corresponding to the system desired gap. Future versions of this graph should inverse the fill colors on the current distance bar graph, so that it fills from the top down, instead of from the bottom up. This optical trick, which was used with the forward collision warning display, would provide the effect of a brighter display (more fill area) as the danger or vehicle gets closer.

Two maneuvers, gap closing and opening, were performed during the automated driving scenario. As shown in Figure 3.2.35, the closing and opening gap maneuvers were graphically represented by the change in the desired gap indicator on the gap display. However, since this was a small change graphically, the maneuver was also noted in the text display below the speedometer. A range-rate display, located below the speedometer and to the left of the text message, showed the closing rate in m/s, but this display did not provide any particularly critical information to the driver since this information was more naturally perceived by watching the rate of change of current gap.

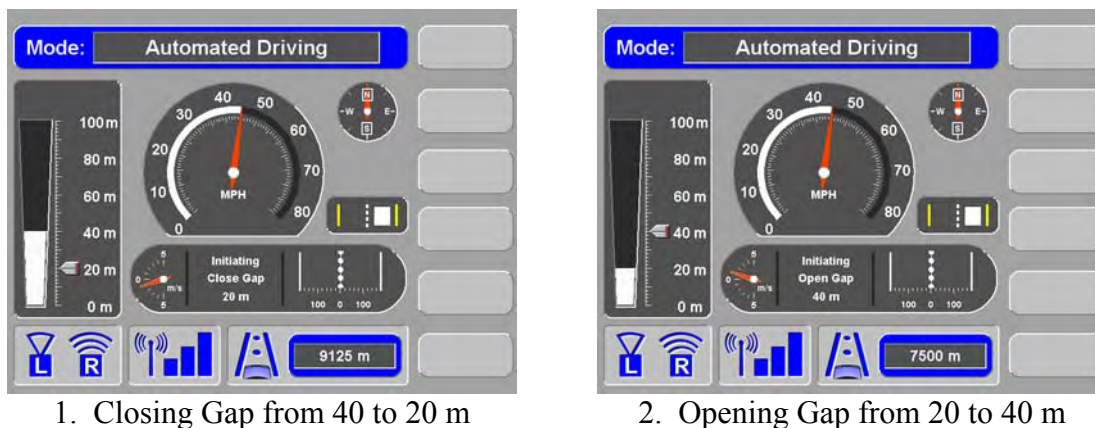


Figure 3.2.35 Opening and Closing Gap Maneuvers.

The relative independence of the lateral control system and the speed control system allowed for several systems modes that were an artifact of this architecture. First, as shown in Figure 3.2.36, Frame 1, lateral control could be activated before the transition to automated speed control, placing the vehicle into a lane assist mode. Likewise, the automated speed control as shown in Figure 3.2.36, Frame 2, could be activated without the lateral control, placing the vehicle in an adaptive cruise control or automated following mode. Finally, a fault in the lateral control system did not deactivate speed control, resulting in the display shown in Figure 3.2.36, Frame 3.



Figure 3.2.36 Additional Automated Driving Scenario Screens.

3.2.10.4 System Performance

Overall, the DVI architecture and graphic display functioned adequately for the system testing that was performed. However, the weakest link in the system was the serial connection used to transfer data between the control computer and the DVI computer. Updated system information was sent every 150 ms over the serial connection which was set to 19.2 Kb/s. Given that the update message was only around 70 Bytes, there should have been plenty of bandwidth in the serial line for the data transfer, but there were frequent and unexplained problems with information becoming backed up in the serial line, especially when bi-directional communications were used, such as when the DVI computer requested a scenario mode change or a lane change.

The mean display update cycle was around 100 ms which was adequate given that the system information was only updated every 150 ms. However, combining both the delay in the information transfer and the delay in the screen update produced a maximum potential delay of 250 ms, which could easily be perceived by a driver under certain circumstances. During testing, it was found this delay showed up most prominently when rapidly switching between manual driving and lane assist (on the order of several times a second). Several optimizations could be explored in the future to eliminate these delays, including event driven, rather than time driven, data transfers and prioritized screen updates.

3.2.11 Video Display for Visitors (quad splitter and screen)

To allow for Driver Vehicle Interface (DVI) development, Human Factors studies, technological demonstrations, and added safety during initial testing stages, a passenger video display was designed and installed using commercial off-the-shelf (COTS) products. This system consisted of: a) three color bullet cameras, b) a composited video quad splitter, c) SVGA splitter/NTSC converter, and d) an 18" color LCD SVGA and TV monitor combination.

This design setup was switchable between full screen DVI display and a quad split-view including three camera views and the DVI display. Cameras were mounted to view the

bus driver and DVI, the lead or following vehicle, and the passenger door to docking station alignment. The display monitor was positioned atop the radio cabinet directly behind the driver's seat for full visibility and viewing by all passengers in the forward portion of the bus.

3.2.12 GPS Positioning and Bus Stop Display

Bus Stop Display

The bus stop display used GPS information to provided a graphic depiction of where the bus was along the I-15 route as shown in Figure 3.2.37. Additionally, the current bus speed (in mph), travel direction, and approximate distance from the bus stop were provided.

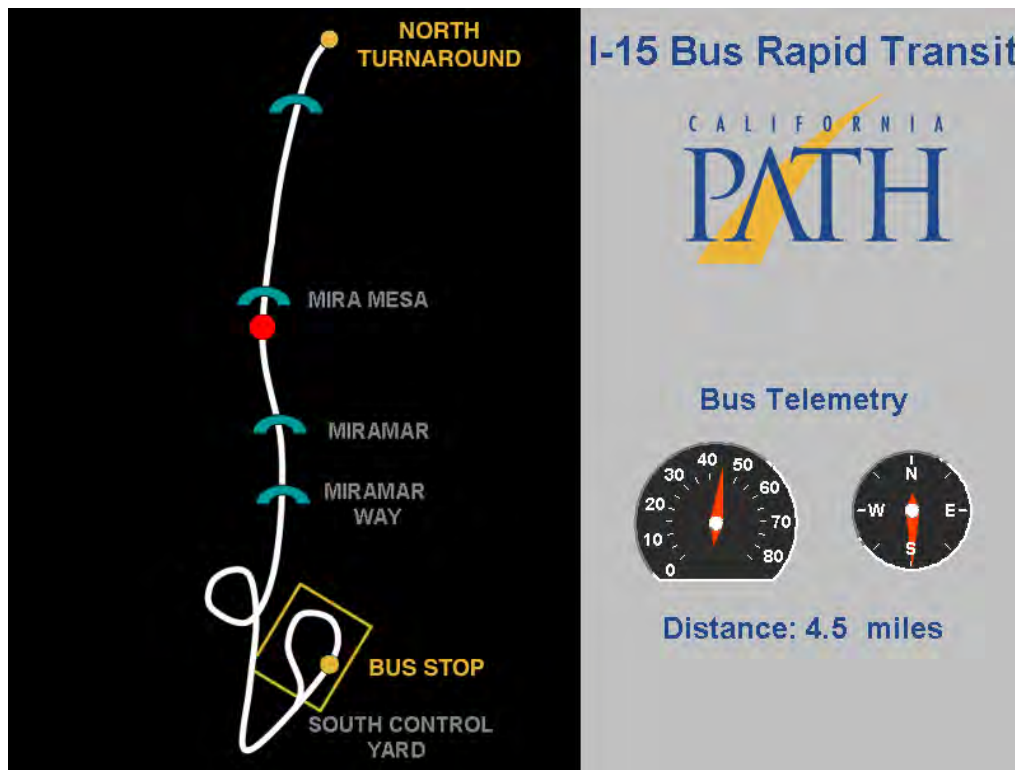


Figure 3.2.37 Bus Stop Display.

3.3 Modifications to Freightliner Trucks for Automation and Data Acquisition

To the maximum extent possible, the modifications to the Freightliner trucks were designed to be similar to those for the buses, but some differences between the vehicles and their applications required that things be done differently. In this section, the focus is on the things that were done differently from the way they were on the buses, rather than repeating descriptions of things that were done in generally the same way.

Note that the Euro-EBS enhancements to the trucks were already described in Section 2. as part of the general description of the truck characteristics, even though these were modifications explicitly needed for automation.

3.3.1 Distance Sensing: Eaton Vorad Radar, Lidar and Their Data Fusion

The Eaton-Vorad EVT-300 is a Doppler radar, which measures primarily relative speed (using the Doppler effect) while relative distance is estimated based on the relative speed measurement. In principle, it can simultaneously track seven targets and provides the following information for each target: target ID, range (relative distance), range rate (relative speed), azimuth, magnitude, and target lock. In theory, information about Target ID and target lock can be used for multiple target tracking. In practice, due to the Doppler effect, which causes much range measurement dropping (zero relative distance estimation when relative speed is nearly zero), it is difficult to associate a measurement with the correct target among the seven.

The Denso lidar can track 8 targets in principle. It has relatively better distance (both longitudinal and azimuth) measurements compared to the Eaton-Vorad radar. However, it is affected severely by weather such as rain, snow, fog and dust. The characteristics of those two types of ranging sensor systems are listed in Table 3.3.1.

	Effective range	Measurement principle	View angle	Azimuth resolution	Latitude resolution	Weather effect
Denso Lidar	150 m	Distance based	± 20 [deg]	~ 0.01	0.01 [m]	severe
Eaton-Vorad radar EVT-300	120 m	Relative speed based	± 6 [deg]	0.1 [deg]	Slightly distance dependent	weak

Table 3.3.1 – Characteristics of Ranging Sensors

3.3.1.1 Target Association

The main problem for vehicle following using radar is to detect the target in the front. Here we do not describe multiple target tracking algorithms. Instead, we need to track the vehicle in front reliably and accurately in distance measurement. Because we do not use relative speed measurements for vehicle following with communication of speed

information from the leading vehicle, we will mainly focus on distance filtering. The Doppler radar distance measurement will drop to zero when the relative speed is nearly zero. This characteristic will be used in radar target association. There is a difference between lidar and radar for target association, based on their measurement principle.

Algorithm for radar target association:

Let $range[I]$, $rate[I]$, and $az[I]$, $I = \dots, 7$ denote radar distance, relative speed and azimuth measurement. $Track_ID$ denotes the track number which catches the front vehicle.

Step 1: To choose initial track:

For $i=1:7$

 If $range[J] > 2.0$ and $range[J] < 100.0$ and $rate[J] \neq 0.0$

 Then $track_ID=J$

If more than one track number satisfies these conditions, then use the smallest one.

Step 2: Target association:

For radar, start from the initial track. At each step, let $rate(track_ID)$ and $az(track_ID)$ represent the detected front vehicle range and azimuth. For sufficiently small parameter $\varepsilon_1, \varepsilon_2 > 0$, if

$$|rate(track_ID) - rate(J)| = \min_i \{|rate(track_ID) - rate(i)|\}$$

$$|az(track_ID) - az(J)| = \min_i \{|az(track_ID) - az(i)|\}$$

$$|rate(track_ID) - rate(J)| \leq \varepsilon_1$$

$$|az(track_ID) - az(J)| \leq \varepsilon_2$$

then $range(J)$, $rate(J)$ and $az(J)$ are considered as the new measures of the track of the front target.

Step 3: Set $track_ID = J$ and go to the next Step 2.

If at least one of the last two conditions are violated, then a measurement error will be reported, which may indicate that the radar measurement has a problem.

For lidar, the above algorithm still applies except that, (a) the total number of tracks is 8; (b) It is distance based, which means “rate” is changed to “longitudinal distance” and “az” is changed to “lateral distance”. The choice of parameters $\varepsilon_1, \varepsilon_2 > 0$ also depends on design requirement.

For both lidar and radar distance measurements, low-pass digital filters [LYN] are used for smoothing the measurements. In particular, the following filters are used:

(a) Recursive type:

$$\bar{x}_n = \lambda x_n + (1 - \lambda)\bar{x}_{n-1}$$

$$0 < \lambda < 1$$

(b) Frequency based (Butterworth):

$$x[0] = 0.4320 * x_old[0] - 0.3474 * x_old[1] + 0.1210 * lidar_rg;$$

$$x[1] = 0.3474 * x_old[0] + 0.9157 * x_old[1] + 0.0294 * lidar_rg;$$

$$lidar_rg_out = 0.4984 * x[0] + 2.7482 * x[1] + 0.0421 * lidar_rg;$$

Figure 3.3.1 shows the raw and filtered radar data after using the above filter, in a manually-driven test that deliberately created multiple large variations in separation between vehicles. The left plots show the raw range data (lower) and filtered range data (upper), indicating how the multiple drop-outs have been eliminated. These drop-outs are associated in some cases with loss of reflections from the target and in other cases with loss of Doppler effect when the speed difference approached zero. The importance of speed difference can be seen in the plots of range rate on the right side of the figure (raw data, lower and filtered, upper). In the case of the range rate, note also that a significant zero-offset had to be removed.

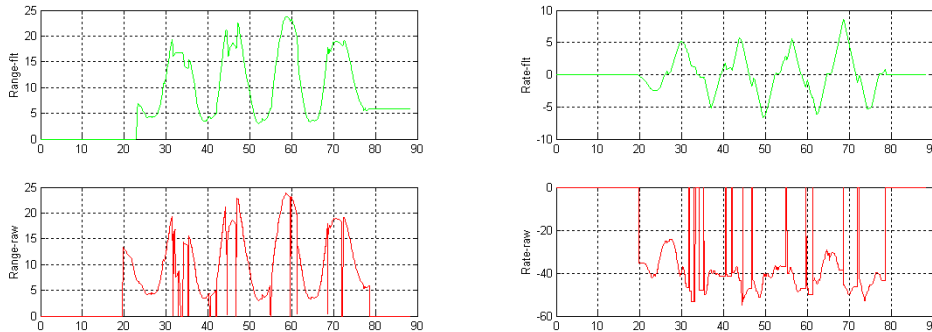


Figure 3.3.1 Eaton-VORAD EVT300 Radar Data for Range and Range Rate, Raw and Filtered

3.3.1.2 Data Fusion

The purpose for radar and lidar data fusion is to achieve more reliable and accurate measurements by means of sensor complementarity and redundancy. The following three techniques are used in data fusion of radar and lidar.

- (a) Using lidar distance measurement to compensate for radar distance measurement when relative vehicle speed is zero. In this case, the radar measurement will drop to zero while lidar still has a good measurement if the weather is reasonable. It is

- simply to use the average of the previous step's radar measurement and the current lidar measurement to replace the lost radar measurement.
- (b) If one sensor reports error status, the measurement naturally shifts to the other.
- (c) A static Kalman filter is used to fuse the two distance measurements in normal cases [8].

Figure 3.3.2 shows the filtering and fusion of radar and lidar distance measurements.

The following notations are used in the figure:

- vr_d_rg: Eaton Vorad radar rage
- vr_d_rt: Eaton Vorad radar range rate
- lid_rg: Lidar range
- flt: filtered
- K-F: Distance fusion after Kalman filtering

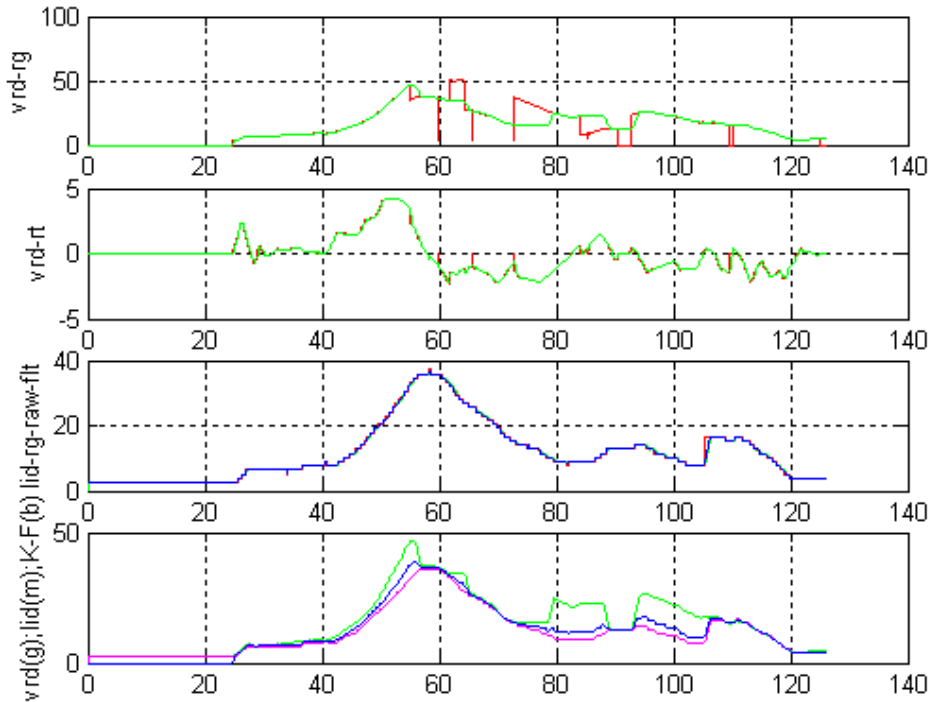


Fig. 3.3.2 Fusion of Radar and Lidar data to Obtain Improved Measurement of Range

The top plot shows how multiple drop-outs in the radar range measurement have been filtered to produce the improved radar outputs, while the second plot shows less of difference between the raw and filtered radar range rate data. The third plot shows both raw and filtered lidar range data, indicating very little difference between the two. The lower plot shows the combination of the filtered radar range (green), filtered lidar range (magenta) and fused radar and lidar (dark blue).

3.3.2 Control System Design

In general, four factors mainly affect control system design and its robust performance: model mismatch, measurement noise, time delay and external disturbances. Smaller model mismatch will achieve larger robust stability margin and better performance, which can be described in terms of response time and tracking error, etc. On one hand, this requires that the model be complicated enough to capture the intrinsic dynamics of the vehicle, which usually leads to a high dimension and highly nonlinear model. On the other hand, it is required that the model be simple enough for control design and synthesis, for example, that it be feedback linearizable. External disturbances can be expelled by robustness of the controller. Response to measurement noise depends on filtering and data fusion techniques which are to achieve the smallest estimation error, the strongest noise rejection property and the least time delay.

It is recognized that the longitudinal control problems here are quite different from those encountered in the control design for passenger cars [9–14]. The differences arise mainly from vehicle dynamics, which directly affect modeling and control design. In fact, longitudinal control of heavy-duty vehicles for vehicle following is more difficult compared to that of passenger cars. The main difficulties are caused by the following characteristics of heavy trucks:

(a) *Low power/mass ratio*: Fuel rate control variable is easy to saturate and the vehicle has very limited acceleration capability. For example, a fully loaded heavy-duty truck has maximum acceleration of $0.05[m/s^2]$ on a flat road when vehicle speed is over $25[mph]$. This is the main difficulty in control design. Control saturation means the loss of controllability, and thus the automatic driving stability in longitudinal motion as well as the string stability for vehicle following [15-17]. This can be relieved by properly reducing performance demands, imposing limited maneuver profiles. The difficulty is how to achieve the best maneuver performance and robust stability, including string stability simultaneously.

(b) *Large actuator time delays*: The delays appear in both engine control and brake control. (1) For engine control, it is the delay from pedal deflection to fuel rate. This delay is as large as $200[ms]$ sometimes. (2) For brake system control, large delays appear in both pneumatic brake and transmission retarder. For example, pneumatic brake activation delay is about $600[ms]$ and release delay is about $800[ms]$. These delays must be dealt with properly. As reported in [18], using predictor approach for brake control design does not bring significant advantages in response. Here, a different but effective approach to compensate for time delay is presented.

(c) *Mass dominant*: It is due to the appearance of the term $Mgh_r \sin\theta$ in vehicle dynamics. This makes the feedback controller very sensitive to road grade. A small fluctuation in road grade $\theta \pm \Delta\theta$ will cause a large fluctuation $\pm Mgh_r \cdot \Delta\theta$ in torque demand. This may destroy closed-loop stability. But too slow a response to road grade

changes may lead to a similar result. Thus road grade knowledge is very important for a heavy truck to move up/down a hill. An obvious suggestion is to use road preview by GPS linked to a map database.

The following strategies are used in model simplification for control design:

1. For turbocharged diesel engine, turbocharger dynamics is separated from the engine dynamics under the assumption that the booster pressure (manifold pressure between turbocharger and the cylinders) is measured.
2. A static nonlinear engine map which gives a functional relationship among engine speed, booster pressure, fuel rate and indicated torque is used to replace the engine dynamics in closed-loop control.
3. Brake dynamics is separated from drive-line dynamics.

Here the control variables include acceleration pedal deflection, pneumatic brake applied pressure, engine brake switching ON/OFF for 2, 4, 6 cylinders, and transmission retarder applied voltage. The whole control system structure can be divided into two layers: (a) Upper level dynamics is composed of the second order linearizable vehicle dynamics. Control design in this level is to generate desired torque from the distance tracking error and speed tracking error through vehicle dynamics. The global feedback linearizability is used for robust closed-loop stability considerations as well as for control synthesis; (b) Lower level control has two branches: Engine control is to generate desired acceleration pedal deflection from desired positive torque; Brake control is to generate and properly distribute a desired braking torque on the three components: Jake brake, pneumatic brake and transmission retarder. To properly synthesize brake control based on the characteristics of the three components is one of the key factors to achieve good performance of longitudinal control for heavy trucks.

Modeling & Control System Structure

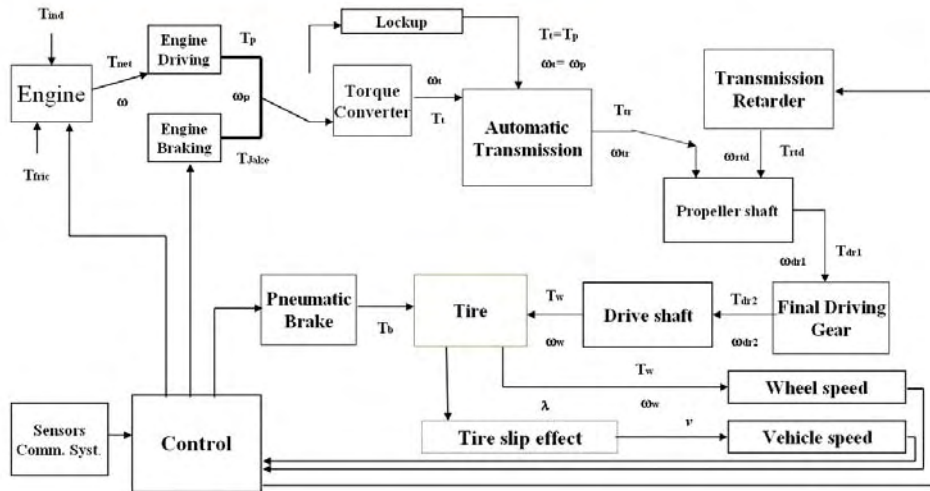


Fig. 3.3.3 Overall Control Systems Structure

3.3.2.1 Disturbances to Control

(a) Accessories

The following components, when activated, produce large engine power (torque) demand disturbances, which need compensation:

Engine cooling fan: $\max = 46.2[hp]$

Generator: $\max = 2.2[hp]$

Water pump: $\max = 2.6[hp]$

Compressor: $\max = 2.6[hp]$

Air conditioner compressor: $\max = 5.2[hp]$

According to [19], they are approximately proportional to $(\omega - \omega_{idle})$

(b) Positive Feedback by Grade Sensor

Grade measurement is very important for longitudinal control of heavy trucks because the term $Mgh_r \sin \theta$ makes a substantial contribution to the desired torque due to large M . In practice, unless road survey information is available and can be matched to the current position of the vehicle, one has to estimate θ using a pitch sensor in real time. However, problems arise in the following two aspects:

- (a) There is persistent measurement noise;
- (b) Vehicle acceleration causes pitch up and deceleration causes pitch down, which gives positive feedback to the closed-loop control, in turn destroying the closed-loop stability. To compensate for this, vehicle acceleration measurement can be used to reduce this effect as

$$\theta_e = \mu(\theta_m) - \sigma(a_m)$$

θ_e – estimated pitch angle

θ_m – measured pitch angle

$\mu(\theta_m)$ – a filtered pitch angle

a_m – a measured acceleration

$\sigma(a_m)$ – filtered acceleration

Then θ_e is used to replace θ .

(c) Gear Shifting

The typical symptom of gear shifting is large vehicle jerking, which is a prominent disturbance to string stability. Although the string stability of the controller in the ideal case guarantees that the uncertainties would be attenuated downstream along the platoon, in practice, there is always a time delay caused by distance measurement and filtering. For example, the Eaton-Vorad Doppler radar has an update interval of 65[ms]. Adding the time delay caused by filtering, the total practical delay would be about 100[ms]. Such delay will accumulate downstream along the platoon, destroying the attenuation capability of the closed-loop controller and thus the string stability.

The transmission gear shifting point depends on both vehicle speed and acceleration. For multiple-vehicle platooning, it is very difficult to synchronize the shifting for all the vehicles because there are always differences between speed tracking errors as well as distance tracking errors between those vehicles.

3.3.2.2 Upper Level Control Design

The upper level control design uses the sliding mode approach to generate desired torque from inter-vehicle distance tracking error and speed tracking error based on vehicle dynamics [13,14,20].

To avoid confusion, it is assumed that the starting point of the subject vehicle is the initial point, which is set to 0 and the forward direction is positive for distance (x -coordinate). (x, v, a) and $(x_{pre}, v_{pre}, a_{pre})$ represent the moving distance, speed and acceleration of current vehicle and preceding vehicle respectively. Then

$$\begin{aligned}
x_e &= x_{pre} - x \\
v_e &= v_{pre} - v \\
a_e &= a_{pre} - a \\
x(0) &= 0, \quad x_{pre}(0) = L_{pre} > 0 \\
v(0) &= 0, \quad v_{pre}(0) = 0
\end{aligned}$$

With this notation, there hold

$$\dot{x}_e = v_e, \dot{v}_e = a_e$$

An error model for upper-layer longitudinal control design is

$$\begin{aligned}
\dot{x}_e &= v_e \\
\dot{v}_e &= a_{pre} - \frac{r_d r_g T_{des} - (r_d T_{rd} + T_b + F_a h_r + F_r h_r + M g h_r \sin \theta)}{\bar{I}} \\
x_e(0) &= L_{pre} \\
v_e(0) &= 0
\end{aligned} \tag{3.1}$$

Suppose the desired inter-vehicle distance is $L = const.$ Then choose the sliding surface as

$$s = v_e + k_1(x_e - L), k_1 > 0$$

From any sliding reachability condition $\dot{s} = -\gamma(s)$ [20], the desired torque T_{des} can be solved out as

$$T_{des} = \frac{\bar{I}(\gamma(s) + k_1 v_e + a_{pre})}{r_d r_g} + \frac{(r_d T_{rd} + T_b + F_a h_r + F_r h_r + M g h_r \sin \theta)}{r_d r_g}$$

3.3.2.3 Lower Level Control Design

Due to the built-in engine controller, it is impossible to directly access the fuel rate command. Instead, pedal deflection is used as input. From the upper level controller, desired speed and desired torque are calculated as (v_{des}, T_{des}) . Using the engine mapping, the desired fuel percentage rate α_{des} is obtained from a lookup table, which is similar to the approach in [4]. The rest of this section emphasizes brake control.

(a) Braking System

The braking system for automatic control is composed of three parts: Jake brake, pneumatic brake and transmission retarder. Each part has its own characteristics. To understand these characteristics for braking system control strategy is the key factor for good performance of the control system and safety.

The total braking torque at the wheels is:

$$T_{total} = \begin{cases} T_b + T_{tr}^{(w)} + T_{jk}^{(w)}, & \omega \geq 800[rpm] \\ T_b + T_{tr}^{(w)}, & \omega < 800[rpm] \end{cases}$$

Jake Brake

A variable structure model is used:

$$T_{jk} = \begin{cases} T_{jk_0}, & 0 \text{ cylinder on} \\ T_{jk_2}, & 2 \text{ cylinder on} \\ T_{jk_4}, & 4 \text{ cylinder on} \\ T_{jk_6}, & 6 \text{ cylinder on} \end{cases}$$

$$T_{jk_0} = T_{eng-brk} = \frac{C_{eng-brk}(\omega - \omega_{idle})}{R_g}$$

$$T_{jk_i} = \frac{C_{jk_i}(\omega - \omega_{idle})}{R_g}, \quad \omega \geq 800[rpm]$$

$$R_g = r_g r_d$$

$$i=2,4,6$$

The following relationships are obviously true:

$$T_{jk_0} < T_{jk_2} < T_{jk_4} < T_{jk_6}$$

Pneumatic Brake

The pneumatic (air) brake provides continuous and large braking torque on all wheels. Its response is slow. The pneumatic brake model for control design was proposed as [6]

$$\dot{P}_b = \begin{cases} \frac{1}{\tau_{ba}}(-P_b + A_b P_{app}(t - \tau_a)), & \text{activation} \\ \frac{1}{\tau_{br}}(-P_b + A_b P_{app}(t - \tau_r)), & \text{release} \end{cases}$$

To design the controller, one expands the applied brake pressure $P_{app}(t - \tau_a)$ and $P_{app}(t - \tau_r)$ using Taylor series approach as follows

$$P_{app}(t - \tau_a) \approx P_{app}(t) - \dot{P}_{app}(t)\tau_a$$

$$P_{app}(t - \tau_r) \approx P_{app}(t) - \dot{P}_{app}(t)\tau_r$$

which is replaced into (1.2) to obtain

$$\dot{P}_b = \begin{cases} \frac{1}{\tau_{ba}} \left[-P_b + A_b(P_{app}(t) - \dot{P}_{app}(t)\tau_a) \right], & \text{activation} \\ \frac{1}{\tau_{br}} \left[-P_b + A_b(P_{app}(t) - \dot{P}_{app}(t)\tau_r) \right], & \text{release} \end{cases}$$

To implement it, $\dot{P}_{app}(t)$ can be calculated from $P_{app}(t)$ in real-time by a difference method or an integral filter [21].

Transmission Retarder

To deal with the time delay part in the model, $\varsigma(V_{rtd}(t - \kappa_{tr}))$ is approximated by its first order Taylor series

$$\begin{aligned} \varsigma(V_{rtd}(t - \kappa_{tr})) &= \varsigma(V_{rtd}(2t - \tau_{tr0})) \\ &= \varsigma\left(V_{rtd}\left[2\left(t - \frac{\tau_{tr0}}{2}\right)\right]\right) \\ &= \varsigma\left(\bar{V}_{rtd}\left(t - \frac{\tau_{tr0}}{2}\right)\right) \\ \bar{V}_{rtd}(p) &= V_{rtd}(2p) \end{aligned}$$

Thus

$$\varsigma(V_{rtd}(t - \kappa_{tr})) = \varsigma(\bar{V}_{rtd}(t)) - \zeta(\bar{V}_{rtd}(t)) \frac{\tau_{tr0}}{2}$$

which is replaced into (1.7) as

$$\dot{T}_{rtd} = \begin{cases} \frac{1}{\tau_{tr}} \left[-T_{rtd} + \varsigma(\bar{V}_{rtd}(t)) - \zeta(\bar{V}_{rtd}(t)) \frac{\tau_{tr0}}{2} \right], & t < \tau_{tr0} \\ \frac{1}{\tau_{tr}} \left[-T_{rtd} + \varsigma(V_{rtd}(t)) \right], & t \geq \tau_{tr0} \end{cases}$$

for control design.

Braking System Control Strategy

Suppose the total desired braking torque on all wheels is T_{brk_total} .

A variable structure braking system control strategy is proposed as follows.

If $T_{brk_total} \leq T_{jk_0}$, no pneumatic brake nor Jake brake is necessary, but engine fueling is relaxed.

If $T_{jk_0} < T_{brk_total} < T_{jk_2}$,

$$T_b^{(des)} + T_{rd}^{(des)} = T_{brk_total} - T_{jk_0}$$

If $T_{jk_2} \leq T_{brk_total} < T_{jk_4}$,

$$T_b^{(des)} + T_{rd}^{(des)} = T_{brk_total} - T_{jk_2}$$

If $T_{jk_4} \leq T_{brk_total} < T_{jk_6}$,

$$T_b^{(des)} + T_{rd}^{(des)} = T_{brk_total} - T_{jk_4}$$

If $T_{jk_6} \leq T_{brk_total}$,

$$T_b^{(des)} + T_{rd}^{(des)} = T_{brk_total} - T_{jk_6}$$

The distribution of the desired braking torque among the air brake and transmission retarder is completely up to the designer.

Based on the relationship between applied brake pressure and brake torque,

$$T_b = r_b \sigma_b P_b$$

$$P_{app} = \sigma_0 V_b$$

$$P_{app} \leq P_{res}$$

the desired brake pressure is calculated as

$$P_b^{(des)} = \frac{T_b^{(des)}}{r_b \sigma_b}$$

from which $\dot{P}_b^{(des)}$ can be calculated in principle.

Based on (1.4), the applied brake pressure should drive the error system

$$\dot{P}_b - \dot{P}_b^{(des)} = \begin{cases} \frac{1}{\tau_{ba}} \left[-P_b + A_b (P_{app}(t) - \dot{P}_{app}(t) \tau_a) \right] - \dot{P}_b^{(des)} & \text{activation} \\ \frac{1}{\tau_{br}} \left[-P_b + A_b (P_{app}(t) - \dot{P}_{app}(t) \tau_r) \right] - \dot{P}_b^{(des)} & \text{release} \end{cases}$$

global asymptotically stable. To achieve this, it is sufficient to choose

$$\frac{1}{\tau_{ba}} A_b (P_{app}(t) - \dot{P}_{app}(t) \tau_a) = \frac{1}{\tau_{ba}} P_b^{(des)} + \dot{P}_b^{(des)}$$

activation

$$\frac{1}{\tau_{br}} A_b (P_{app}(t) - \dot{P}_{app}(t) \tau_r) = \frac{1}{\tau_{br}} P_b^{(des)} + \dot{P}_b^{(des)}$$

release

With such a choice, the error dynamics is

$$\dot{P}_b - \dot{P}_b^{(des)} = \begin{cases} \frac{-1}{\tau_{ba}} (P_b - P_b^{(des)}), & \text{activation} \\ \frac{-1}{\tau_{br}} (P_b - P_b^{(des)}), & \text{release} \end{cases}$$

which is globally asymptotically stable.

Or equivalently,

$$\dot{P}_{app}(t) - \frac{1}{\tau_a} P_{app}(t) = \frac{-\tau_{ba}}{\tau_a A_b} \left[\frac{1}{\tau_{ba}} P_b^{(des)} + \dot{P}_b^{(des)} \right],$$

activation

$$\dot{P}_{app}(t) - \frac{1}{\tau_r} P_{app}(t) = \frac{-\tau_{br}}{\tau_r A_b} \left[\frac{1}{\tau_{br}} P_b^{(des)} + \dot{P}_b^{(des)} \right],$$

release

These equations can be integrated as

$$P_{app} = \begin{cases} \frac{-\tau_{ba}}{\tau_a A_b} \exp\left(\frac{t}{\tau_a}\right) \int_0^t \left[\frac{1}{\tau_{ba}} P_b^{(des)} + \dot{P}_b^{(des)} \right] e^{\sigma \tau_a} d\sigma, & \text{activation} \\ \frac{-\tau_{br}}{\tau_r A_b} \exp\left(\frac{t}{\tau_r}\right) \int_0^t \left[\frac{1}{\tau_{br}} P_b^{(des)} + \dot{P}_b^{(des)} \right] e^{\sigma \tau_r} d\sigma, & \text{release} \end{cases}$$

which can be implemented in real-time.

Similarly, one can get $\bar{P}_{rtid}(t)$ from the transmission retarder model (1.8), which is the voltage to be applied.

3.4 Coordination Layer Control Implementation

The coordination layer control for the automated buses is primarily formed of two separate, but interacting, hybrid automata; the Trajectory Planner and the Coordinator. The Coordinator interfaces with the supervisory layer to select a complex maneuver for the bus to execute. This supervisory layer represents either the Driver Vehicle Interface (DVI) or a scripting mechanism (dubbed the “Fake DVI”). At the lowest level, the Trajectory planner computes a specific desired trajectory for the regulation layer controller to track based on the current maneuver. Each of these components will now be discussed in more detail.

3.4.1 Coordinator

The Coordinator’s purpose is to enforce constraints about the physics of the vehicle and the safety of logical transitions. The primary physical constraints to be imposed are limiting the maximum commanded acceleration/deceleration to within the capabilities of the vehicles. The safety constraints include checks on the vehicle/system status prior to transitioning between control modes, such as verifying that the FLS have acquired a target vehicle before switching to distance tracking mode.

The Coordinator hybrid automaton has three operating modes: human, speed regulation/tracking, and distance regulation/tracking. All transitions, regardless of state, result in an event being propagated to the Trajectory planner and the regulation layer controller to indicate the mode switch, desired tracking profile parameters, etc. The human mode indicates that the operator is driving the vehicle under manual control, and is always the initial state of the Coordinator. The transition from human to speed tracking mode can be completed once a set speed has been reached or the operator initiates the mode switch via the DVI. Within the speed tracking mode, the current set speed can be changed or control can be returned to the operator at any time. Distance tracking can be engaged only when the sensor processing algorithm and communications systems verify that an automated bus is preceding the vehicle. Within the distance tracking mode, transitions to either human or speed tracking can be taken at any time.

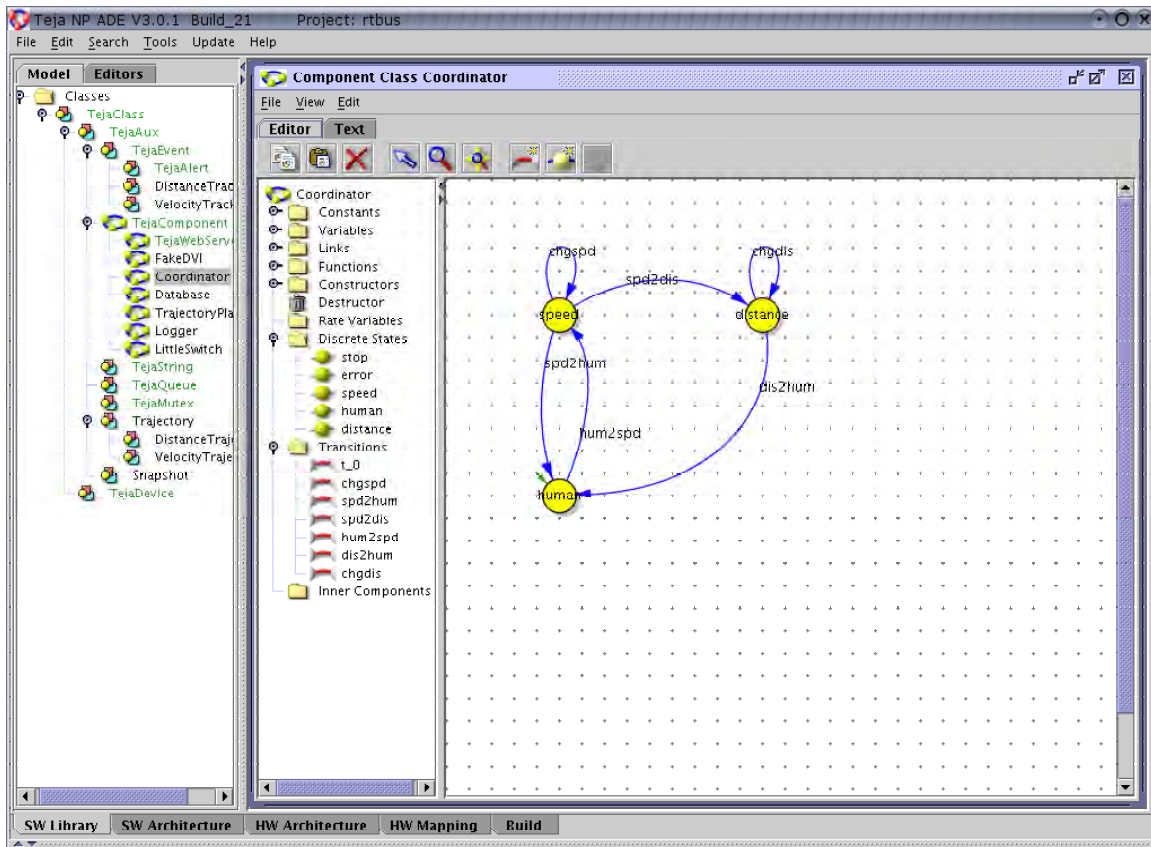


Figure 3.4.1 Coordinator State Machine

3.4.2 Trajectory Planner

The purpose of the Trajectory Planner component is to generate a smooth speed or distance profile for the regulation layer to track when notified of a change in maneuver. The profile is calculated on-demand using the current and final desired conditions as well as a maximum acceleration capability. Currently, several different curve fit routines are available to provide flexibility in implementation and performance. This formulation allows for smooth transient behavior when the desired speed or distance changes, regardless of the current operating condition.

The hybrid automaton for the Trajectory Planner has two states representing the possible operating modes of the regulation layer, corresponding to speed tracking and distance tracking. Within each state, two fundamental operations can occur; generation of the overall profile and computation of the current desired variable. Generation of the trajectory occurs either at initialization or upon receipt of certain events from a Coordinator component. The Coordinator can force the trajectory to be recomputed because of a change in the final desired condition (the `chng_*` transitions) or when the tracking mode changes (the `to_*_tracking` transitions). The trajectory is then stored as a set of coefficients and an estimate of the overall time, t_{final} , required to achieve the final set point. In either state, the default flow is to compute the desired distance, speed, and

acceleration at the current time step, which is subsequently passed to the regulation layer via the PATH database.

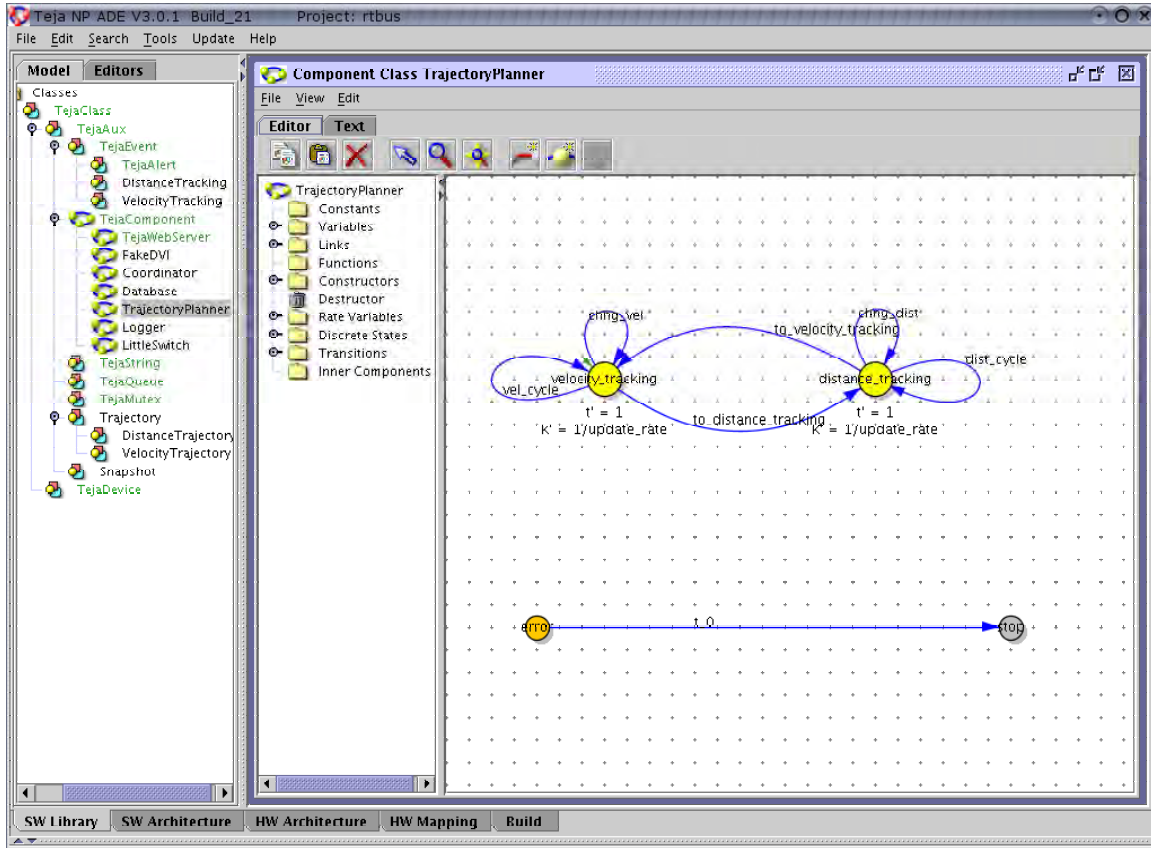


Figure 3.4.2 Trajectory Planner State Machine

For the speed tracking state, the trajectory is computed using a different method depending on the relative difference between the current and final speeds. This is due to the significant difference in vehicle performance under acceleration compared to deceleration (essentially fuel and brake control, respectively). If the final speed is greater than the current speed, a first order system is used to compute the desired speed. The governing equations are:

$$\tau = \frac{(v_f - v_i)}{a_{\max}}$$

$$t_{\text{final}} = 3\tau$$

$$v(t) = v_i + \left(1 - e^{-\frac{t}{\tau}}\right)(v_f - v_i)$$

$$a(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}}(v_f - v_i)$$

where v_f , v_i , and a_{\max} are the final velocity, initial velocity, and maximum absolute acceleration, respectively. Example plots of a typical acceleration trajectory are shown in Figure 3.4.3.

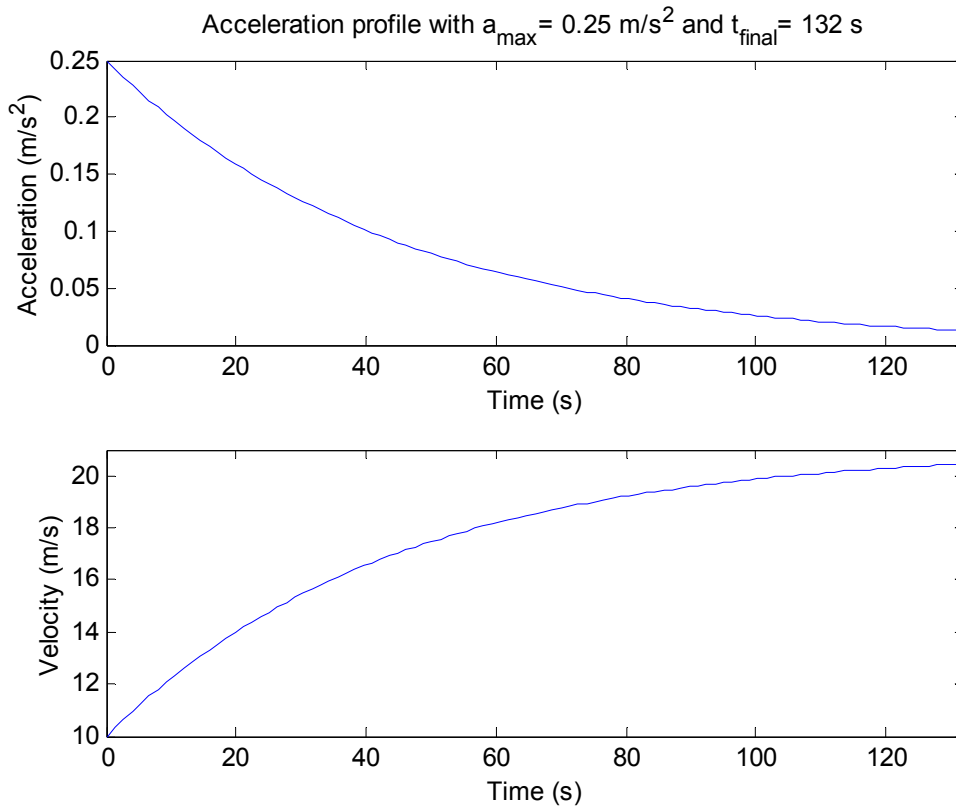


Figure 3.4.3 Desired Trajectory for Vehicle Acceleration

If the final speed is less than the current speed, a second-order polynomial fit is used to compute the desired speed. The governing equations are:

$$t_{\text{final}} = -2 \frac{(v_f - v_i)}{a_{\max}}$$

$$v(t) = v_i - \frac{a_{\max}}{2t_{\text{final}}} t^2$$

$$a(t) = -\frac{a_{\max}}{t_{\text{final}}} t$$

Example plots of a typical vehicle deceleration trajectory are shown in Figure 3.4.4.

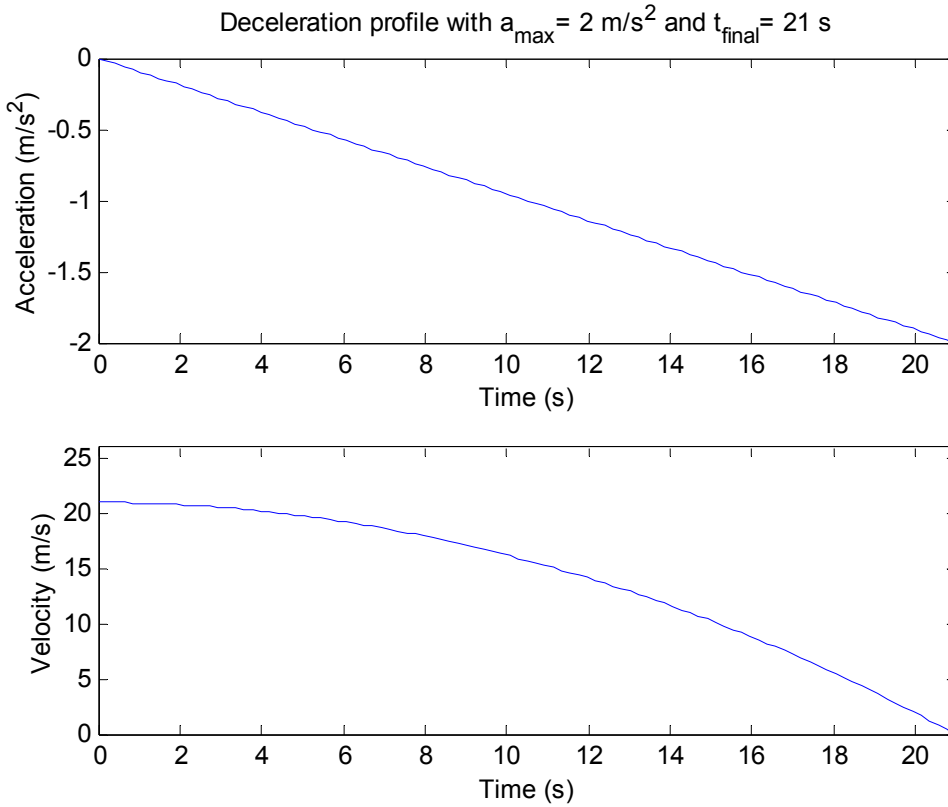


Figure 3.4.4 Desired Trajectory for Vehicle Deceleration

For distance tracking, a quintic polynomial distance profile is used in order to guarantee smoother transient behavior at the endpoints of the maneuver, while limiting the maximum absolute relative acceleration required for the trajectory. This characteristic of the implemented method was attractive because of the strict limitations on the acceleration capabilities of the buses due to roadway grades, variable mass, and the limitations of the engine performance. Based on the algebraic method described in Nickalls [22], a cubic polynomial for the relative acceleration between the current vehicle (denoted with subscript i) and its predecessor in the platoon (denoted with subscript $i-1$) can be determined such that

- The relative acceleration $da = a_{i-1} - a_i$ is bounded by a_{\max} .
- The relative velocity at the beginning and end of the trajectory is zero.
- The initial and final distances are specified a priori as d_i and d_f , respectively.

The mathematical equations describing the time of maneuver, polynomial coefficients and relative acceleration are as follows:

$$t_{\text{final}} = \left(\frac{10}{\sqrt{3}} \frac{|d_f - d_i|}{a_{\max}} \right)^{\frac{1}{2}}$$

$$c_0 = 12\sqrt{3} \frac{a_{\max} \text{sign}(d_f - d_i)}{t_{\text{final}}^3}$$

$$c_1 = -\frac{c_0}{4} t_{\text{final}}^2$$

$$c_2 = \frac{c_0}{64} t_{\text{final}}^4$$

$$c_3 = \frac{(d_f - d_i)}{2}$$

$$\delta a(t) = c_0 t_d^3 + c_1 t_d$$

where $t_d = t - t_{\text{final}}/2$. Similarly, the governing equations for the relative velocity δv and distance δ are;

$$\delta v(t) = \frac{c_0}{4} t_d^4 + \frac{c_1}{2} t_d^2 + c_2$$

$$\delta(t) = \frac{c_0}{20} t_d^5 + \frac{c_1}{6} t_d^3 + c_2 t_d + c_3$$

Example plots of a typical distance trajectory for a closing the distance between vehicles are shown in Figure 3.4.5.

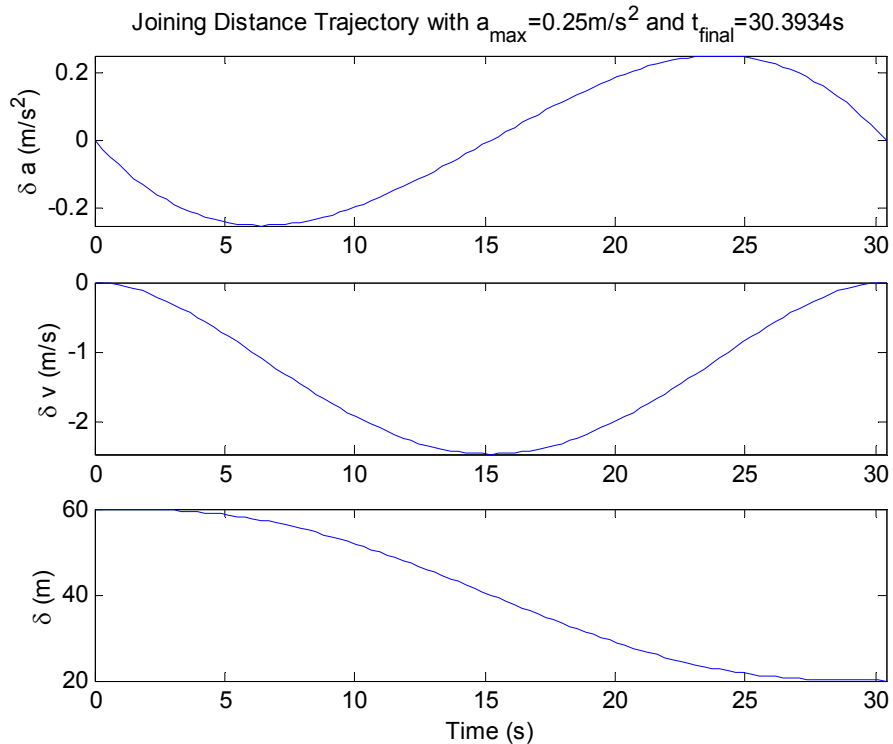


Figure 3.4.5 Desired Trajectory for Closing Gap Maneuver

3.4.3 Mixed Initiative: Human-Machine Interactions

Human-machine interactions happen at the supervisory layer through the Driver-Vehicle Interface (DVI), which was already described in detail in Section 3.2.10. The question of the role of the driver must be considered carefully when designing automated vehicle systems. Transit operators and professional bus drivers tend to favor more limited human-machine interactions, and would prefer the system be completely automated. However, the realities of day-to-day operations in an imperfect world, in mixed traffic, with imperfect sensors, in all weather conditions, make driver intervention an important safety tool.

Scripting Mechanism (Fake DVI)

In case the driver or transit operator chooses for the vehicle to operate automatically, a scripting mechanism is provided to allow easy programming of scenarios. Emphasis was placed on easy programming and plug-and-play scenarios, as the buses are sometimes used by non-expert programmers. The programming of scripts is meant to resemble utilization of the DVI as much as possible. Transitions in the script language correspond to the pressing of buttons on the DVI, except that when running scripts they are triggered based on time or vehicle speed rather than on driver input. States in the script correspond to modes of operation of the buses (manually driven, speed tracking, and vehicle

following, that is, distance tracking, adaptive cruise control and/or platooning). The basic operation of the Fake DVI for both the lead and follower vehicle cases will now be described in more detail.

The Fake DVI state machine begins in the idle state to allow the operator to select several configuration options through the DVI. The configuration options include the vehicle's role in the platoon (leader or follower), the direction of travel along I-15 (northbound or southbound), and the number of passengers (for feed-forward grade compensation). Once the operator finishes selecting the desired configuration, the operator can select the initiation of automatic control from a button press on the DVI.

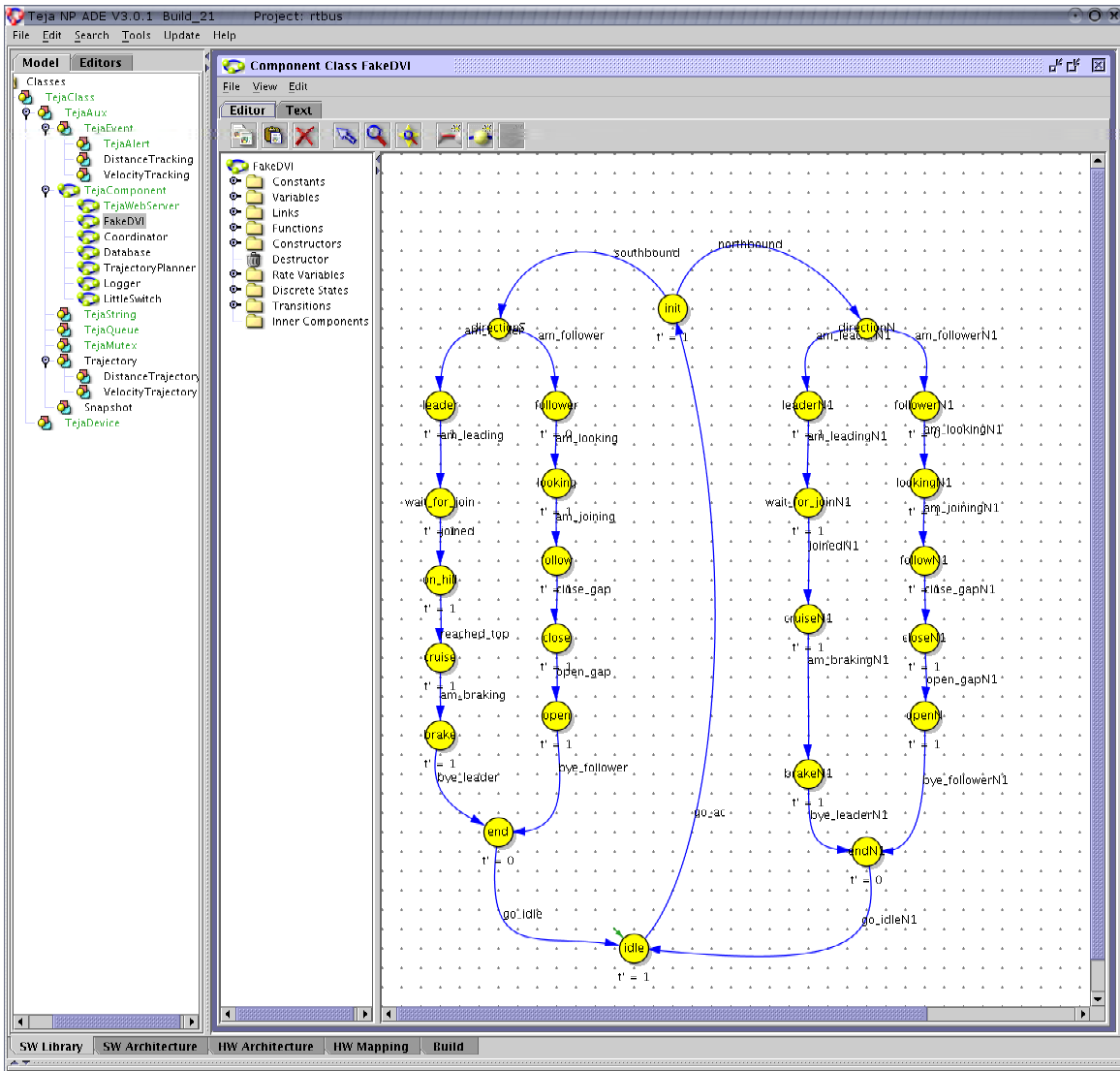


Figure 3.4.61 Fake DVI State Machine

The initiation of automatic control forces the Fake DVI to transition into one of four scripts based on the vehicle role and direction of travel configuration options. For the

southbound direction, the lead vehicle progresses through six consecutive states: leader, wait_for_join, on_hill, cruise, brake, and end. Each of the transitions occurs based on a priori specified time intervals unless otherwise noted. It should also be noted that this method of enabling transitioning could easily be modified to be driven either by handshake protocols and sensor information or commands from the operator via the DVI. Furthermore, each transition between these Fake DVI states is synchronized with the hum2spd, chng_spd, or spd2hum transition of the Coordinator to enforce the correct desired trajectory and regulation layer operating mode.

The Fake DVI of the lead vehicle remains in the leader state until one of two events occurs; either the vehicle speed increases above an a priori transition speed threshold or the operator flips off the manual override switch. These events transition the Fake DVI into the wait_for_join state, where the lead vehicle tracks a constant low speed to allow the following vehicle to reach the desired spacing. The implemented scripts then wait for a specified period of time before transitioning to the on_hill state, where the lead vehicle tracks a cruise speed of 18 m/s until the top of the hill is reached. For the southbound run, this is the maximum sustainable cruise speed due to the steep incline on the I-15 track. For the northbound run, this additional on_hill state is not required to maintain stability of the platoon. After the top of the hill is reached, the Fake DVI transitions to the cruise state and tracks a constant speed of 21 m/s until the end of the track is reached. At this point the Fake DVI transitions to the brake state, thus forcing the generation of a deceleration trajectory. Once the automatic transition speed is reached, or the manual override switch is turned on, the Fake DVI transitions to the end state, where the automatic controller is disabled and the operator must resume control of the vehicle. Finally, the Fake DVI automatically transitions back to the idle state to prepare for the operator to initiate another run.

The follower vehicle progresses through six states as well, regardless of travel direction. These six states are follower, looking, follow, close, open, and end. Similar to the lead vehicle, each of the transitions occurs based on a priori specified time intervals unless otherwise noted. Also, each transition between these Fake DVI states is synchronized with one of the Coordinator transitions to enforce the correct desired trajectory and regulation layer operating mode.

The Fake DVI of the follower vehicle remains in the follower state until one of two events occurs; either the vehicle speed increases above an a priori transition speed threshold or the operator flips off the manual override switch. These events transition the Fake DVI into the looking state, and the Coordinator into the speed state, where the follower vehicle determines if a preceding vehicle is present. A preceding vehicle is determined to be present if the FLS sensor processing process has acquired a target and the wireless communication system hears packets from a master node. If a vehicle is present, the implemented scripts transition into the follow state, the Coordinator transitions to distance control, and a trajectory is generated to reach to the desired spacing using the current range estimate as the initial condition. To demonstrate the performance of the regulation layer at tracking a desired range, the desired distance is decreased and subsequently increased by the transitions to the close and open states. The specific

timing of these maneuvers is dependent on the travel direction to ensure that the desired distance is not significantly changed while on the steep incline. Once the automatic transition speed is reached, or the manual override switch is turned on, the Fake DVI transitions to the end state, where the automatic controller is disabled and the operator must resume control of the vehicle. Finally, the Fake DVI automatically transitions back to the idle state to prepare for the operator to initiate another run.

Chapter 3 References

- [1] D. Hall, *Mathematical techniques in multisensor data fusion*, Artech House, 1992.
- [2] J. Gertler, Survey of model-based failure detection and isolation in complex plants, *IEEE Control System Magazine*, December 1988, pp. 3-11
- [3] R. Isermann, Fault diagnosis of machines via parameter estimation and knowledge processing, *Automatica*, vol. 29, no. 4, 1993, pp. 815-835
- [4] F. Koushanfar, S. Slijepcevic, M. Potkonjak and A. Sangiovanni-Vincentelli, Error-tolerant multimodal sensor fusion, *IEEE CAS Workshop on Wireless Communication and Networking*, Pasadena, CA, Sep. 5-6, 2002
- [5] T. Clouqueur, P. Ramanathan, K. Saluja and K.-C. Wang, Value-fusion versus decision-fusion for fault-tolerance in collaborative target detection in sensor networks, in *FUSION 2001*
- [6] D. Swaroop, J. K. Hedrick, P. P. Yip, and J. C. Gerdes, "Dynamic surface control for a class of nonlinear systems," *IEEE Trans. Automatic Control*, vol. 45, no. 10, 2000, pp. 1893-1899.
- [7] A. Howell, B. Song, and J.K. Hedrick, Cooperative range estimation and sensor diagnostics for vehicle control, *Proceedings of ASME: Dynamic Systems and Control Division*, To appear in Nov. 2003
- [8] Chui, C. K. and Chen, G, 1999, *Kalman Filtering, with Real-Time Applications*, Third Edition, Springer, Berlin. [CHU]
- [9] Cho, D. and J. K. Hedrick, 1989, Automotive power train modeling for control, *Transaction of ASME, J. Dynamic Systems, Measurement and Control*, **Vol. 111**, p568-576. [CHO]
- [10] Hedrick, J. K., Nonlinear controller design for automated vehicle applications, *Proc. UKACC Int. Conf. on Contr. '98, 1-4 Sept. 1998, Swansea, U. K.*, p 23-31. [HDK]
- [11] Rajamani, R., S. B. Choi, J. K. Hedrick and B. Law, 1998, Design and experimental implementation for a platoon of automated vehicles, *Proc. of the ASME Dynamic Systems and Control Division*. [RAJ1]
- [12] Rajamani, R., H.-S. Tan, B. Law and Zhang, 2000, Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons, *IEEE Trans. on Control Systems Technology*, **Vol. 8, No. 4**, p. 695-708. [RAJ2]
- [13] X. Y. Lu and J. K. Hedrick, Longitudinal control design and experiment for Heavy-Duty Trucks, *Proc. of 2003 American Control Conference*, June 4-6, Denver, Colorado. [XYL6]

- [14] X. Y. Lu, H. S. Tan, S. E. Shladover, J. K. Hedrick, 2000, Implementation and comparison of nonlinear longitudinal controllers for car platooning, *Transaction of ASME, Journal of Dynamic Systems, Measurement and Control*, Vol. 123, p161-167. [XYL8]
- [15] X. Y. Lu and J. K. Hedrick, A panoramic view of fault management for longitudinal control of automated vehicle platooning, accepted to present at 2002 ASME IMECE, Dynamic Systems and Control Division, Advanced Automotive Technologies Symposium, Nov. 17-22, 2002, New Orleans. [XYL4]
- [16] X. Y. Lu and J. K. Hedrick, Practical string stability, 18th IAVSD Symposium on Dynamics of Vehicles on roads and Tracks (Int. Association of Vehicle System Dynamics), August 25-29, 2003, Atsugi, Kanagawa, Japan. [XYL7]
- [17] D. Swaroop and J. K. Hedrick, String stability of interconnected systems, *IEEE Trans. Auto. Contr.*, 40, no.3, p 349-357. [SWA]
- [18] E. J., Yanakiev, J. Eyre, D. and Kanellakopoulos, 1998, of AVCS for Analysis, Design and Evaluation for Heavy-Duty Vehicles with actuator delays, California PATH Research Report, UCB-ITS-PRR-98-18. [YAN]
- [19] X. Y. Lu and J. K. Hedrick, Modeling of heavy-duty vehicles for longitudinal control, *Proc. of The 6th Int. Symposium on Advanced Vehicle Control*, Hiroshima, Japan, Sept. 9-13, 2002, Paper No. 109. [XYL3]
- [20] X. Y. Lu and J. K. Hedrick, 2001, H_∞ Sub-optimization in dynamic back-stepping multiple surface control, *Proc. of American Control Conference*, Arlington VA, June 25-27, p4986 – 4991. [XYL5]
- [21] X. Y. Lu and J. K. Hedrick, 2000, Integral filters from a new viewpoint and their application in nonlinear control, *Proc. IEEE Conference on Control Application*, Anchorage, Alaska, p 501-506. [XYL2]
- [22] R. W. D. Nickalls, 1993, “A new approach to solving the cubic: Cardan’s solution revealed,” *The Mathematical Gazette*, Vol. 77, pp. 354-360.

Chapter 4. Performance of Automated Heavy Vehicle Systems

In this section, the actual test performance of the automated heavy vehicles is presented. The first three subsections address the performance of the buses in precision docking, and then lateral and longitudinal control at highway speeds. The later subsections address the performance of the trucks in longitudinal and then lateral control at highway speeds.

4.1 Precision Docking of Buses

The most important control function for precision docking is lateral control, but longitudinal control of the stopping bus has also been implemented in order to provide a fully automated docking capability. The lateral control results are described first, followed by those for longitudinal control.

4.1.1 Docking Lateral Control

4.1.1.1 Control System Description

The bus docking system provides a special function within bus lateral control systems. The system employs almost identical vehicle components to the general automatic steering capabilities. In the control system software, except for certain “docking” control algorithm specifics and the “docking” accuracy requirements, those of the precision docking system are designed to be identical to the automatic steering control system. A typical bus docking/automatic steering system includes the following basic elements and functions: lateral guidance system, steering actuator system, lane-keeping algorithm and transition control algorithm, driver vehicle interface control system, fault detection and control system, and associated longitudinal control and sensing system (if applicable).

Figure 4.1.1.1 illustrates the system block diagram of the PATH precision docking system. It includes hardware, and software with various functions. The hardware consists of switches, steering DC motor, vehicle sensors, computer, displays and audible units. The software consists of various software drivers and control algorithms. The control algorithms include steering actuator algorithm, lateral control algorithm, and DVI controls. The steering actuator control algorithm provides functions such as self-calibration, tight position servo, mode transitions and fault detection. The lateral control algorithms include lane-tracking control, transition control, trajectory planning, mode switching and failure detection and fault mode control. The DVI control tracks the control states and driver reactions, and provides appropriate interfaces for manual, automated, and transition controls. It also includes warning and emergency support.

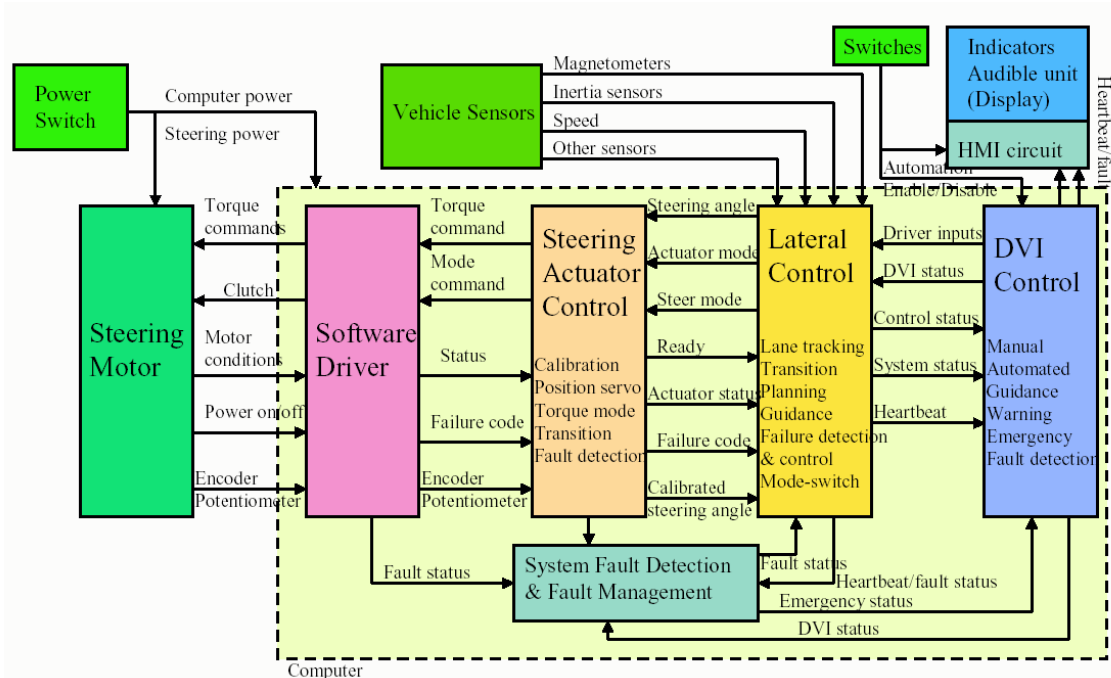


Figure 4.1.1.1: PATH Precision Docking/Automatic Steering Control System Block Diagram

4.1.1.2 Control Performance Requirements

A bus precision docking control system is a subset of an automated lateral control system for bus lane-assist application. An automated lateral control system targeted toward a an automatic steering application is required to perform all normal steering functions from leaving a bus station to arriving at a bus station with extremely high reliability. It should be robust against different roadway geometries, unknown vehicle loading, various speeds, and changing roadway surface conditions. An ideal element of such an automated steering control system is a high-gain robust “vehicle lateral servo” that “steers” the vehicle to follow any desired trajectory as long as such trajectory is defined within the limitations of the vehicle capabilities. The closed-loop performance requirements for a general bus automated lateral control algorithm are defined as follows:

1. 0.2 meter maximum tracking error for highway driving without any prior knowledge of the roadway
2. 0.5 meter maximum tracking error for 0.3-g automated steering maneuver without any prior knowledge of the roadway
3. 0.02 meter maximum tracking error for vehicle speed less than 5 m/s on straight sections of the roadway for docking accuracy
4. No noticeable oscillations at frequencies above 0.3 Hz for passenger comfort, and 0.4 minimum damping coefficient for any mode at lower frequencies

5. 1 m/s^2 maximum lateral acceleration deviation between the lateral acceleration created by the vehicle and that from the road geometry
6. Consistent performance under various vehicle-operating conditions

Requirements 1, 2 and 6 address scenarios other than precision docking. They are included simply because a “precision docking ready” controller should also be “automatic steering ready.” They use virtually the same vehicle and infrastructure components.

4.1.1.3 Lane-Keeping and Docking Controller Design

The results in [11, 12] indicate that a simple “look-ahead” controller combining constant feedback gain and constant look-ahead distance may achieve all performance requirements described in Section 4.1.1.2 except for the high accuracy in precision docking. The low speed precision docking requires much higher gain to satisfy the high tracking accuracy requirement. Furthermore, several practical constraints limit the feasibility of such a simple implementation. The amplification of the measurement noise limits the length of the look-ahead distance to a couple of car lengths; high feedback gain with large look-ahead distance can easily excite the non-linearity or unmodeled dynamics in the steering actuators or mechanism; large look-ahead distance creates noticeable steady-state tracking error during curves.

In order to address both the practical limitations and the specific phase lead requirement associated with the constant look-ahead distance depicted in [11], a frequency shaped look-ahead controller law as shown in Figure 4.1.1.2 is proposed as

$$V_s(s) = sG_{\dot{y}_r, \delta}(s) + d_s(v)G_{ds}(s)sG_{\dot{\psi}, \delta}(s) + h_s sG_{\dot{\phi}, \delta}(s), \quad (4.1.1.1)$$

along with a feedback compensator that mainly compensates for the actuator dynamics:

$$C(s) = k_c(v)G_c(s) \quad (4.1.1.2)$$

where $G_{\dot{y}_r, \delta}(s)$, $G_{\dot{\psi}, \delta}(s)$ and $G_{\dot{\phi}, \delta}(s)$ are transfer functions from steering angle to \dot{y}_r , $\dot{\psi}$ and $\dot{\phi}$, (the derivative of lateral displacement at CG w.r.t. road reference frame, the yaw rate, and the roll rate) respectively; and h_s is the distance from the sensor to the roll axis. Furthermore, Figure 4.1.1.2 depicts a generic block diagram of a steering feedback system consisting of five subsystems: actuator dynamics ($A(s)$), road reference (desired lateral acceleration at the sensor location: $\ddot{y}_{ref} = v^2 \rho$), vehicle dynamics at sensor ($V_s(s)$), vehicle kinematics ($1/s^2$) and control law ($C(s)$).

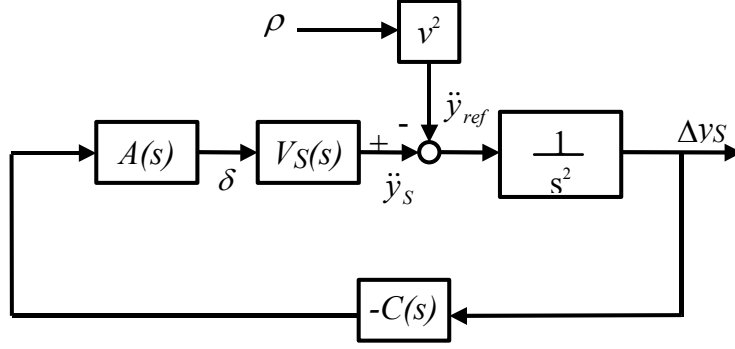


Figure 4.1.1.2: Bus Steering Control Block Diagram

Two speed-independent filters, $G_{ds}(s)$ and $G_c(s)$, are chosen based on the noise characteristics as well as the bus roll and yaw dynamics coupling as in Eqs. (4.1.1.3) and (4.1.1.4). In order to reduce both the effects of the steady state tracking bias and the unwanted excitation of the high frequency unmodeled actuator dynamics, $G_c(s)$ consists of a low-frequency integrator and high-frequency roll-off. Similarly, $G_{ds}(s)$ is made of a high frequency roll-off portion and a mid-frequency lead-lag filter to limit the look-ahead amplification and to provide extra “look-ahead” between 0.1 and 0.6 Hz.

$$G_c(s) = \frac{12\pi(s + 0.3\pi)}{(s + 0.02\pi)(s + 12\pi)} \quad (4.1.1.3)$$

$$G_{ds}(s) = \frac{(s + 80\pi)(s + 0.014\pi)}{10(s + 8\pi)(s + 0.14\pi)} \quad (4.1.1.4)$$

Inserting $G_{ds}(s)$ into $V_s(s)$ as in Eq. (4.1.1.1) and appending $G_c(s)$ to all open-loop transfer functions immediately after $V_s(s)$ as in the optimization dynamic programming described in [11], the corresponding optimal control gain pair $(\bar{k}_c(v), \bar{d}_s(v))$ can be computed to satisfy the stability requirements for any given vehicle speed v . The optimal control gain pair $(\bar{k}_c(v), \bar{d}_s(v))$ corresponding to the control filters $(G_c(s), G_{ds}(s))$ are plotted in Figure 4.1.1.3. This optimal gain pair guarantees at least 50 degrees phase margin and 6 db gain margin for any vehicle speed. As the result of the additional phase lead created by the larger look-ahead distance between 0.1 and 0.6 Hz, the frequency-shaped look-ahead scheme renders a more desirable optimal gain pair characteristics. Generally speaking, the look-ahead distance increases and the feedback gain decreases as the vehicle speed increases. More specifically, $(\bar{k}_c(v), \bar{d}_s(v))$ remain almost constant for vehicle speeds between 15 and 30 m/s. This indicates that a “constant” controller can work almost optimally from medium to highway speeds. The relatively “flat” gains with respect to velocity also imply that the controller has high tolerance to velocity errors. Furthermore, the small look-ahead and high gains at lower speed also guarantee high accuracy for precision docking.

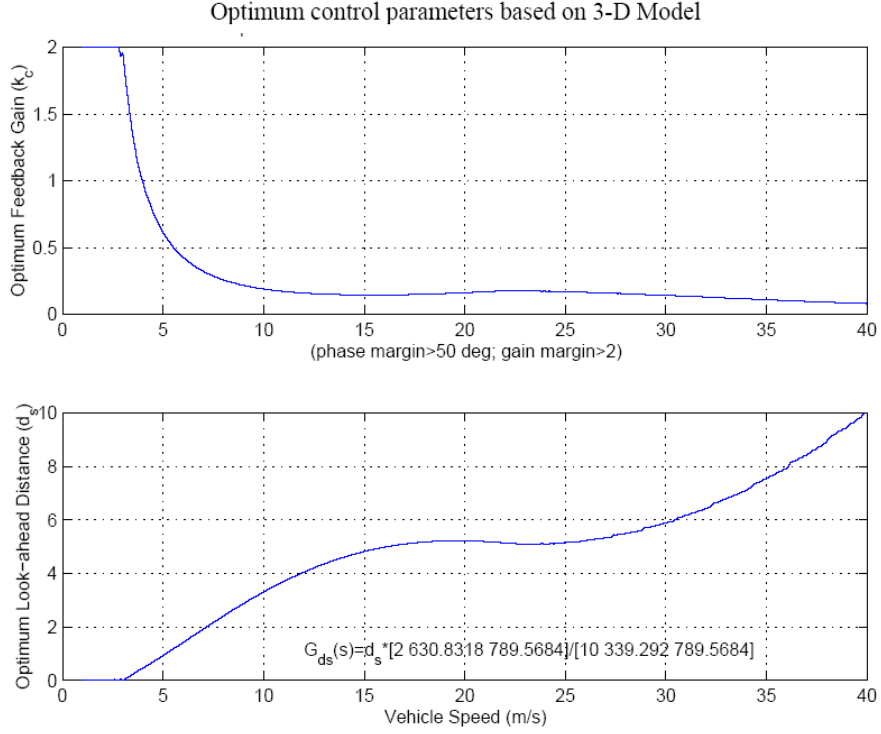


Figure 4.1.1.3: Optimal control gain pair (\bar{k}_c, \bar{d}_s) based on 3 DOF model with frequency shaped look-ahead control ($pm=50$ deg, $gm=2$)

The final steering control algorithm implemented in the bus needs to satisfy both tracking accuracy and ride comfort requirements for all operational scenarios at various bus speeds regardless of the following uncertainties: road adhesion variations, incorrect road curvature information, sensor noise, actuator bandwidth, vehicle dynamics changes, soft suspension modes, and vehicle parameters. The following final frequency shaped virtual look-ahead lane-keeping control algorithm was developed and implemented:

$$\delta_c = -k_c(v)G_c(s)(k_{int}(s)y + d_s(v)G_{ds}(s)\psi) \quad (4.1.1.5)$$

with

$$K_{int}(s) = \frac{(s + 0.3\pi)}{(s + 0.02\pi)} \quad (4.1.1.6)$$

$$G_c(s) = \frac{12\pi}{(s + 12\pi)} \quad (4.1.1.7)$$

$$\psi \cong \frac{y_f - y_b}{L} \quad (4.1.1.8)$$

where δ_c is the steering command, k_{int} an additional integrator to keep the steady state tracking error small, G_{ds} the virtual sensor look-ahead filter, G_c the compensator at the virtual sensor location; y_f and y_b are the lateral measurements in front and back of the vehicle, respectively, and L is the distance between these two sensors. The two gain-

scheduled coefficients, $k_c(v)$ and $d_s(v)$, approximate the velocity dependent relationships in Figure 4.1.1.3.

4.1.1.4 Precision Docking Test Results

Two test results are shown in this report:

- Initial docking preparation results at Richmond Field Station
- Docking test results at the South Control Yard at San Diego

4.1.1.4.1 Precision docking preparation at Richmond Field Station

These were the initial runs when the system was first successfully tested at the Richmond Field Station. The bus starts at the beginning of the docking curve driven manually. During the straight-line segment, transition to automatic control mode is initiated. The transitions were initiated by driver by pushing the “Auto” button. Automatic control mode could be lateral (steering) control only or both lateral control and longitudinal control depending on driver selection. Once transitioned to the automatic mode, the bus will steer itself along the predetermined magnetic track. If longitudinal automatic mode is selected, the bus will accelerate and decelerate to about 10 mph and cruise at this speed. Before stopping at the bus station, the bus will make a full lane change (S-curve) following the magnetic track. If longitudinal automatic mode is activated, the bus will stop at the docking station automatically. Otherwise the driver should control the bus longitudinally to stop at the docking station. The short docking station was placed at the bus’s front door.

Figures 4.1.1.4a to 4.1.1.4c illustrate 26 consecutive precision docking preparation runs for the C2 40-ft bus at the Richmond station docking test track. More than half of the runs were conducted without automated longitudinal control. Figure 4.1.1.4a shows the time traces of all the runs. By observing the speed variations, it is clear that the driver controlled the throttle and brake for most of the runs. In one of the runs, the driver has slowed down the bus to almost a stop and go traffic situation.

Figure 4.1.1.4b illustrates the same plots as those in Figure 4.1.1.4a except that the data were plotted against marker numbers. Plotting against the marker numbers normalizes the data so that they can be compared at every marker location for consistency and easy variations. These plots indicate clearly that the tracking repeatability is very high, to within a couple of centimeters, once the bus enters the lane-change curve. On the other hand, the steering command had variance of more than 50 degrees. This further confirmed the design intuition that a high gain controller is essential for precision docking applications since a control method based on an “open-loop” type command would not produce consistent precision docking as indicated by Figure 4.1.1.4b.

Figure 4.1.1.4c shows the blow-up plot of Figure 4.1.1.4b around the docking station. The blow-up plots clearly show that the bus has never touch the station, and the maximum error after the bus is approaching the station is basically within 2.5 centimeter peak-to-peak.

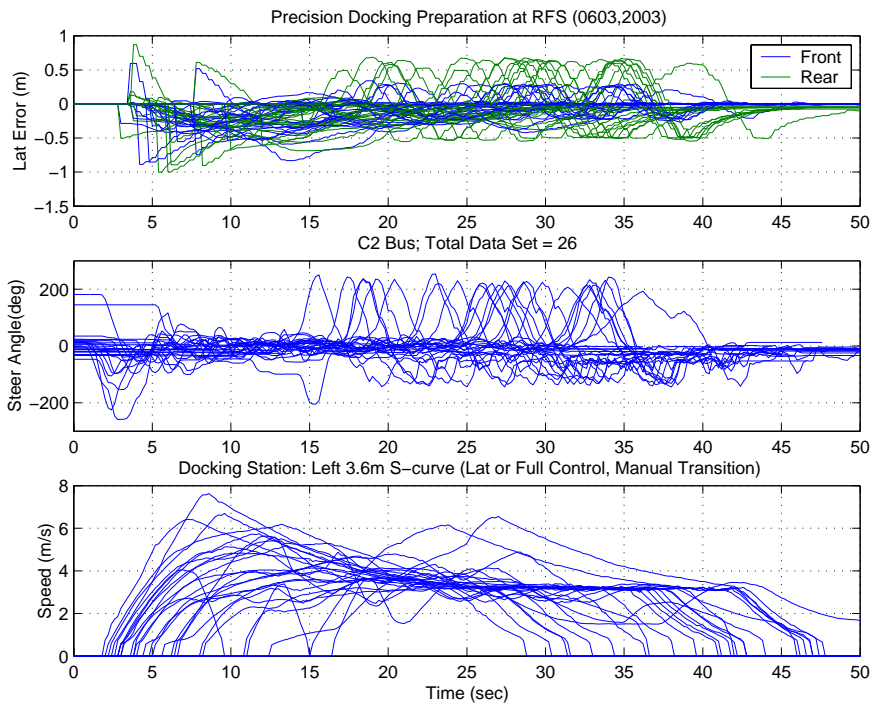


Figure 4.1.1.4a: Precision Docking at RFS (Time based plot) on C2 40' Bus (26 runs)

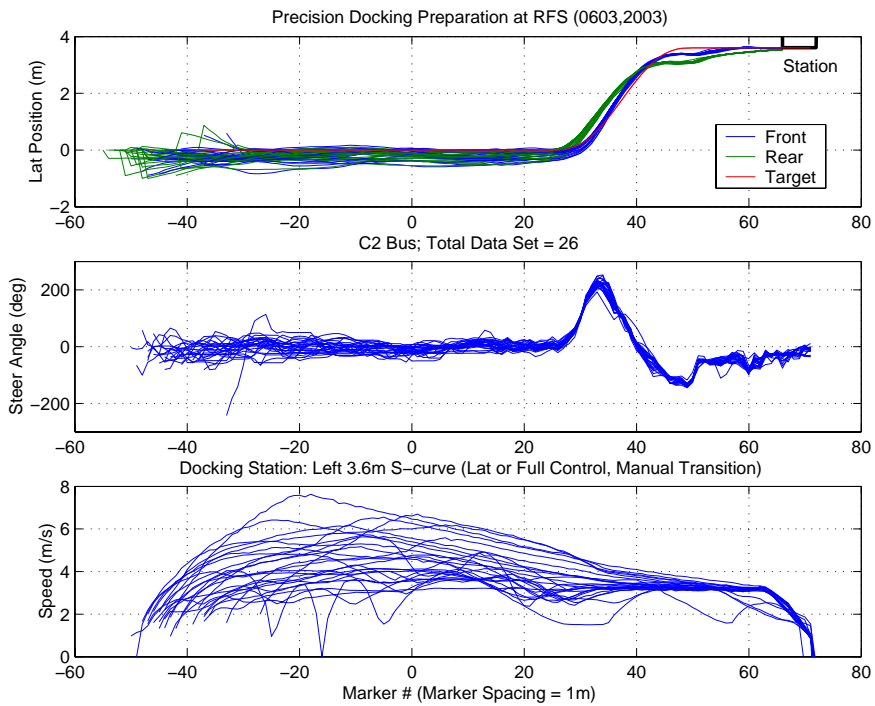


Figure 4.1.1.4b: Precision Docking at RFS (Marker based) on C2 40' Bus (26 runs)

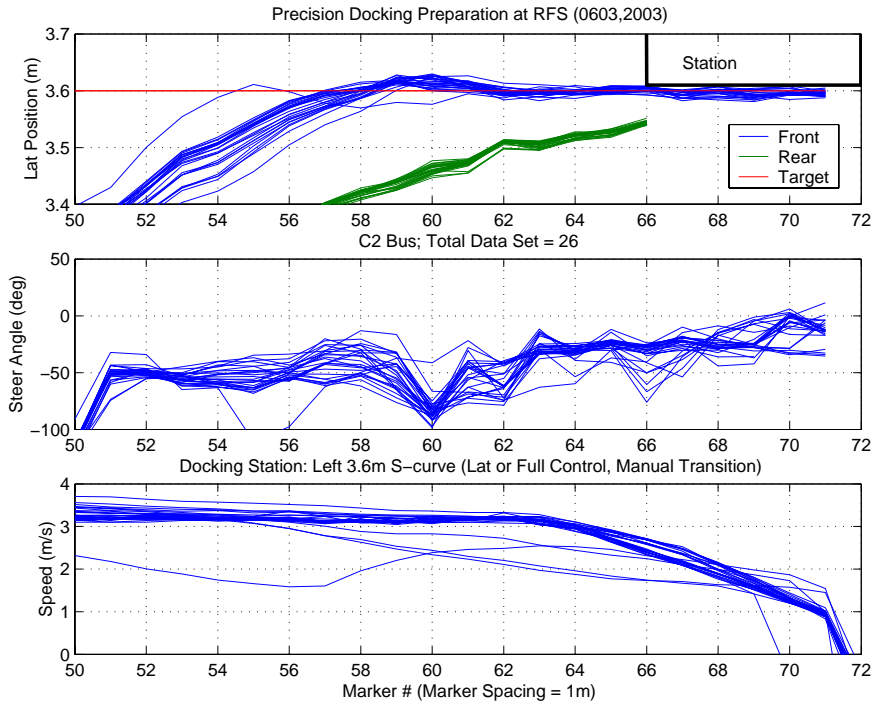


Figure 4.1.1.4c: Precision Docking at RFS (Marker based) on C2 40' Bus (26 runs)

4.1.1.4.2 Precision docking testing at San Diego

The precision docking demo scenario for San Diego is shown in Figure 4.1.1.5. The bus starts at the beginning of the docking curve manually. During the straight-line segment, transition to automatic control mode is initiated. The transition could be initiated by the driver or by the control system itself. Automatic control mode could be lateral (steering) control only or both lateral control and longitudinal control depending on driver selection. Once transitioned to the automatic mode, the bus will steer itself along the predetermined magnetic track. If the automatic longitudinal mode is also selected, the bus will accelerate or decelerate to about 10 mph and cruise at this speed. The bus will first dock at the inline docking station. If the automatic longitudinal mode is activated, the bus will stop at the inline docking station automatically. Otherwise, the driver should control the bus longitudinally to stop at the inline docking station. After passenger unloading and boarding, the driver has two options to pull the bus out of the inline docking station. First, the driver could steer the bus manually out of the inline docking station. Second, the driver could just push the automatic button and the control system will steer the bus out of the inline docking station automatically without driver interference. The docking stations face both the front and rear doors of the bus.

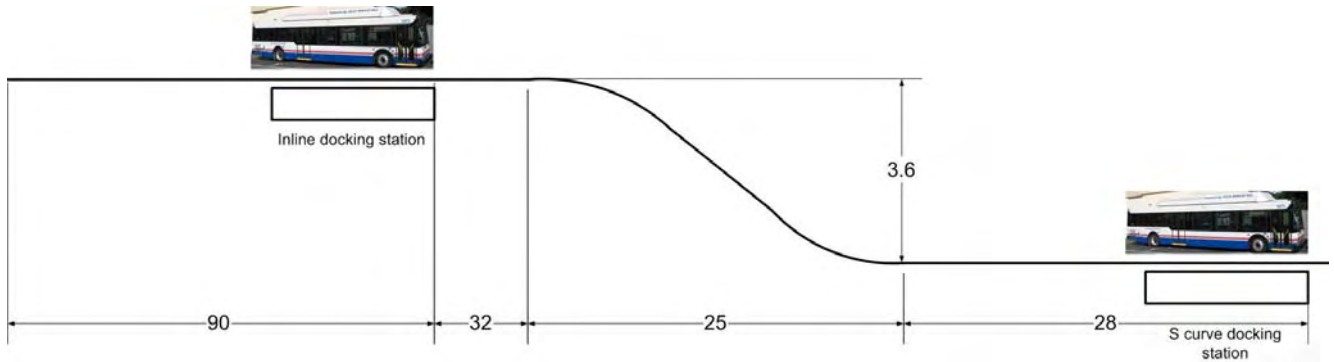


Fig. 4.1.1.5 Docking Scenario in San Diego (Distances shown in meters)

Figures (4.1.1.6a) to (4.1.1.6c) show 17 consecutive precision docking test runs for the C2 40-ft bus at the South Control Yard test track in San Diego. While Figures (4.1.1.6d) to (4.1.1.6f) plot 14 consecutive runs for the C1 40-ft bus at the same test track. The controllers employed by these two buses were identical. No verification was done to verify whether the two buses were identical in their dynamic characteristics. As described before, the test track in the SCY has two docking stations connected by the magnetic markers: first an in-line station and then an S-curve station. For most of the runs, fully automated control with automatic transition from manual to automatic was conducted. The driver drove toward the magnet track, the bus then transitioned to fully automated controls, and stopped at the in-line docking station. When all the passengers got in or out of the bus, the driver pushed the “auto” button and the bus resumed automated control and stopped again automatically at the S-curve docking station. In each set of test runs, several runs were conducted differently to demonstrate the capabilities of the system. These non-normal runs involved stop and go, switching off speed control by stepping on the brake, manually stopping at the station, and manually driving off the station. Observing the speed plots in Figures (4.1.1.6a) to (4.1.1.6f) can easily identify these runs.

Figure 4.1.1.6a and 4.1.1.6d illustrate the lateral positions, steering angle, and speeds with respect to the marker numbers for Bus C2 and C1, respectively. The two buses exhibited similar performance characteristics and accuracy. The two stations were also identified on the lateral position plot in figure. It is worthwhile noticing that the buses automatically left the bus station and kept away from the platform until the rear of the bus cleared away from the in-line bus station for safety. This can also be observed on the blow-up plots for the first in-line docking station as in Figures 4.1.1.6b and 4.1.1.6e.

Similarly, Figure 4.1.1.6c and 4.1.1.6f show the blow-up plot of Figure 4.1.1.6a and 4.1.1.6d around the S-curve docking station. Again, the blow-up plots clearly show that the bus has never touched the station, and the maximum error after bus approaching the station is basically within 1.5 centimeter peak-to-peak for front and 1 centimeter peak-to-peak for rear. The docking accuracy is about 1.5 centimeter peak-to-peak for all runs.

Figure 4.1.1.6g exhibits the status of (1) the transitional switch (-1: none, 0: manual, and 1: auto), (2) the transitional status (0: manual, 1: automated), and (3) the control status (0: manual, 1: lateral control only, 2: lateral and longitudinal controls). It can be seen that the

docking performance is highly consistent despite the fact that the driver has frequently transitioned in and out of the automated control regardless that the speed was either controlled by the driver or by the automated system.

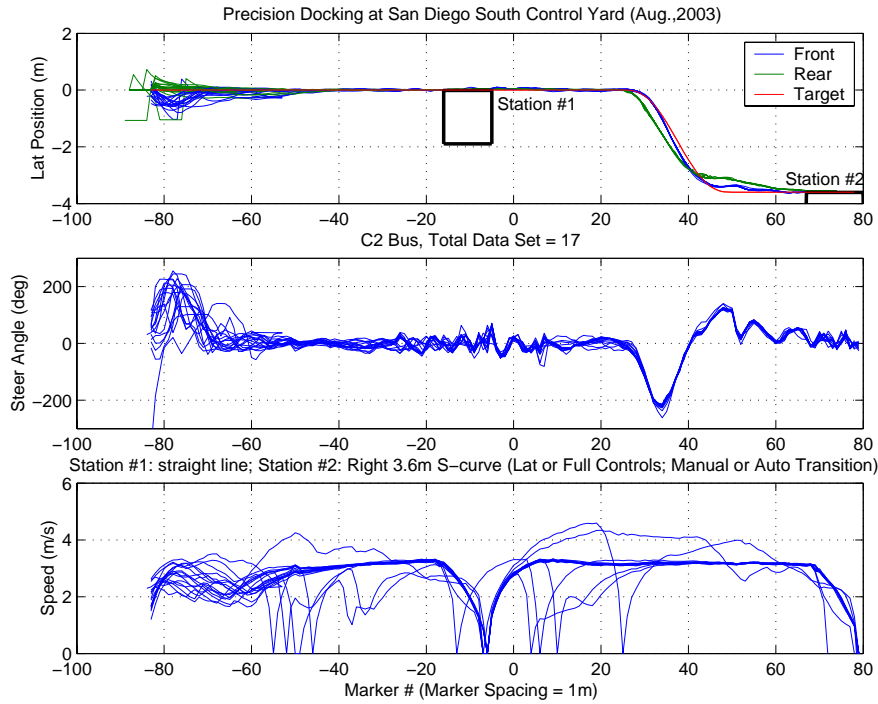


Figure 4.1.1.6a: Precision Docking at SCY (Marker based plot) C2 40' bus (17 runs)

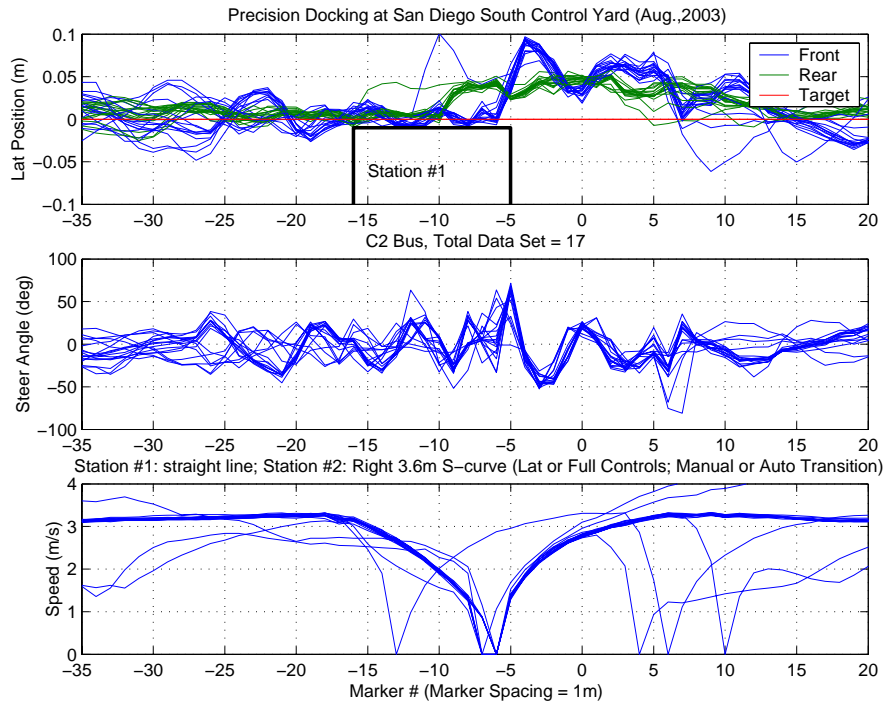


Figure 4.1.1.6b: Precision Docking at SCY (Station #1) C2 40' bus (17 runs)

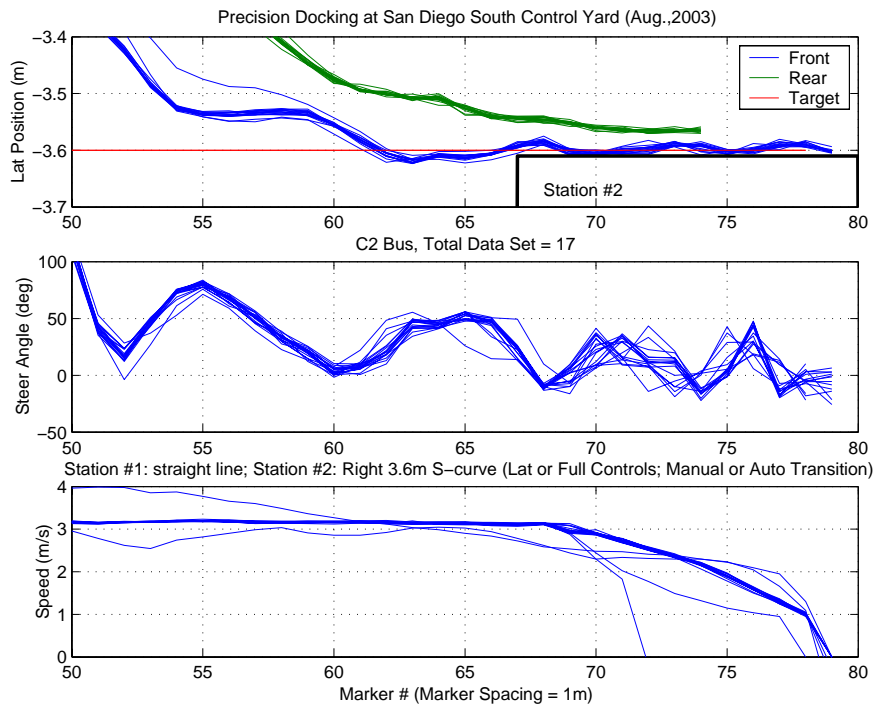


Figure 4.1.1.6c: Precision Docking at SCY (Station #2) C2 40' bus (17 runs)

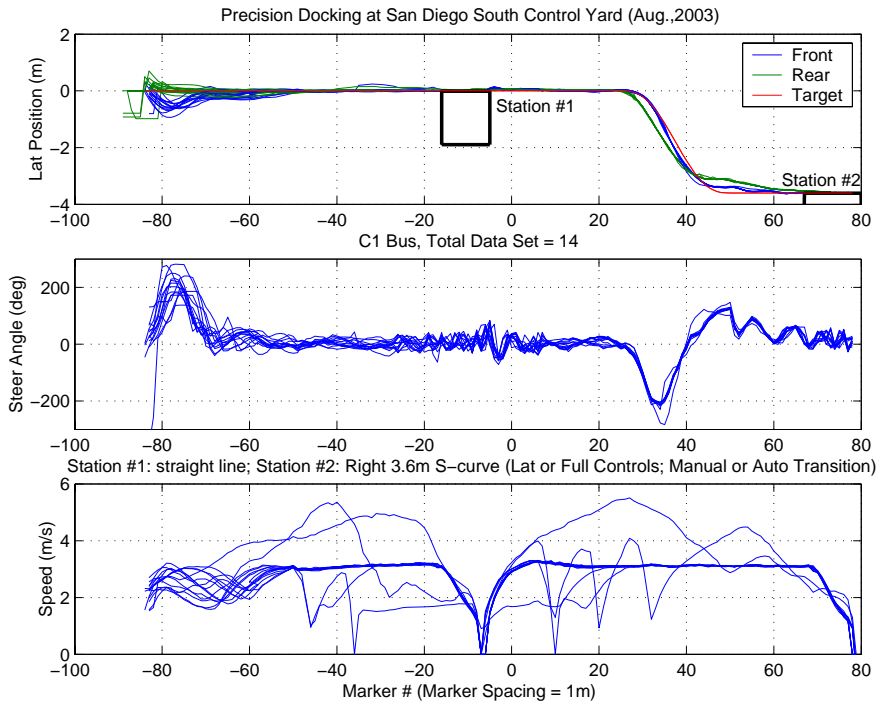


Figure 4.1.1.6d: Precision Docking at SCY (Marker based plot) C1 40' bus (14 runs)

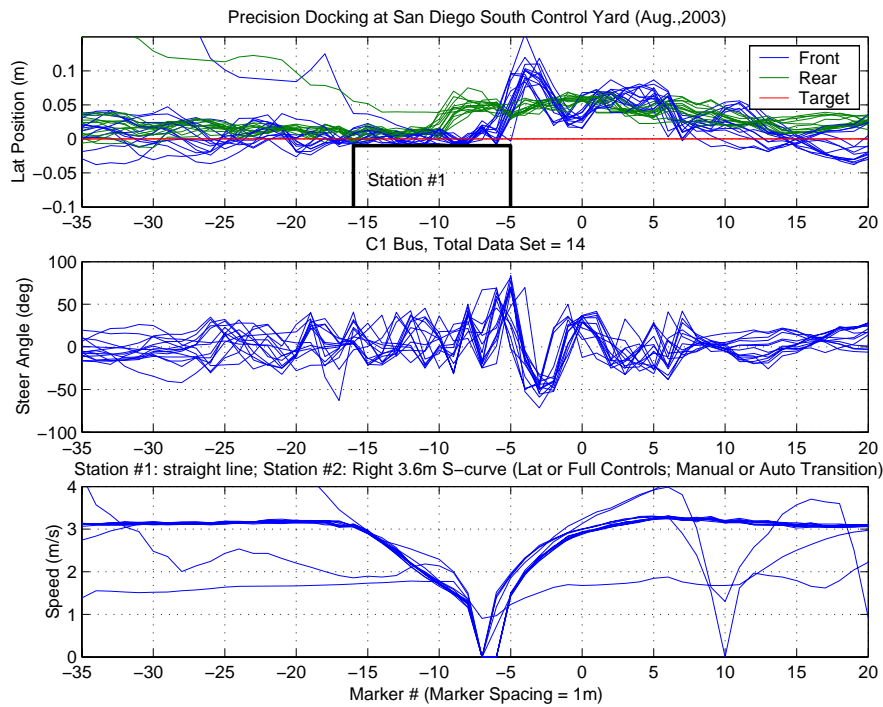


Figure 4.1.1.6e: Precision Docking at SCY (Station #1) C1 40' bus (14 runs)

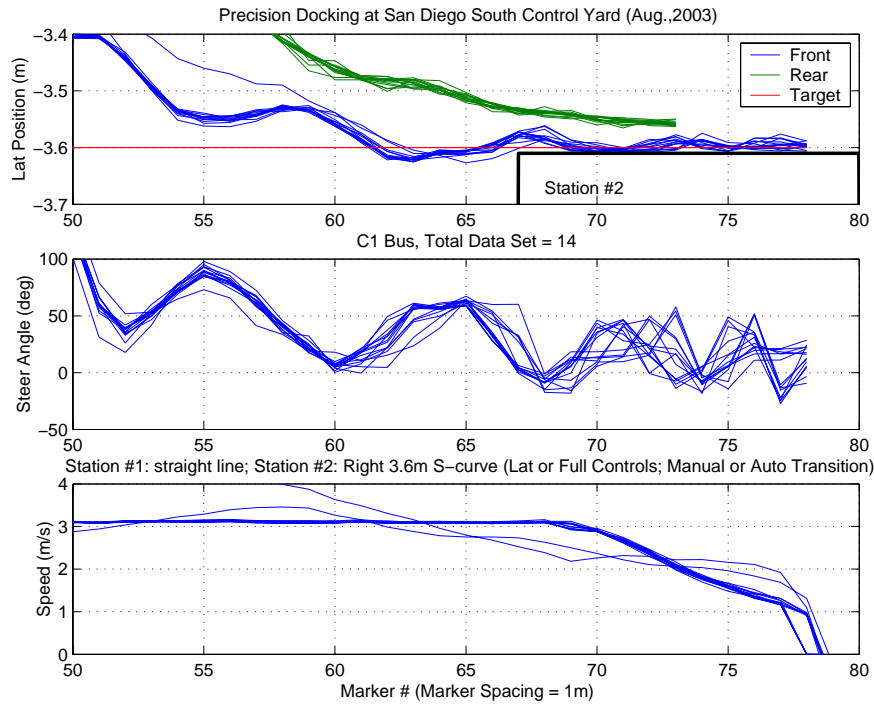


Figure 4.1.1.6f: Precision Docking at SCY (Station #2) C1 40' bus (14 runs)

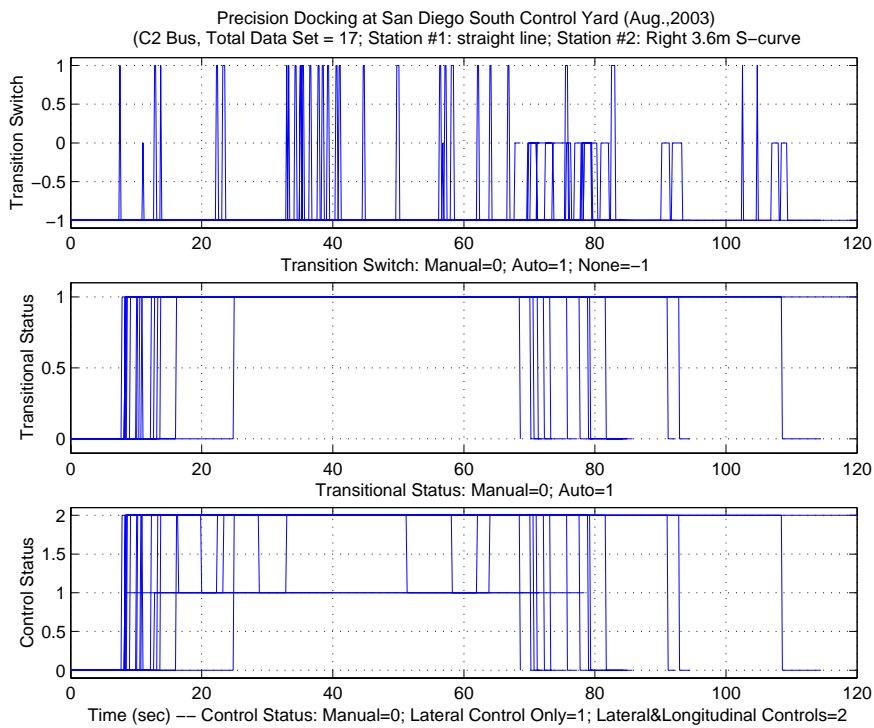


Figure 4.1.1.6g: Precision Docking at SCY (Marker based) on C2 40' Bus (17 runs)

4.1.2 Low-Speed Stopping Control of 40-foot CNG Bus

As previously shown in Fig. 4.1.1.5, two different docking scenarios were tested and proven in San Diego. A 40 foot CNG bus drives straight about 90 meters and stops at the in-line docking station, then the bus starts again, makes a full lane change and stops at the S-curve docking station. Two requirements of docking longitudinal control are: 1) maintain smooth operation to ensure passenger's comfort; 2) stop within 10 centimeter of desired location. In this chapter, the software structure of docking longitudinal control will be introduced in Section 4.1.2.1. Both cruise control and stopping control will be presented in Sections 4.1.2.2 and 4.1.2.3 respectively. Experimental data will also be presented to verify the effectiveness of longitudinal control development.

4.1.2.1 Software structure

From the longitudinal control point of view, both inline docking and S-curve docking maneuvers can be divided into three steps naturally. The bus starts from beginning, cruises at a predetermined speed, reduces speed gradually and then stops at the desired location. Therefore, the docking longitudinal control software can be treated as a state machine switching among its major states: cruise control, stopping control and stop. Fig. 4.1.2.1 shows the block diagram of longitudinal docking control software. Sensor information like engine transmission states and brake pressure is received from the J-Bus and brake pressure sensor. Coordination commands such as longitudinal automatic enable and stop position (location of docking stations) are sent to the docking longitudinal control software by docking coordination (part of docking lateral control program which coordinates docking longitudinal and lateral control). Absolute longitudinal position is also sent to longitudinal control software by docking coordination.

After receiving the sensor information and control command, the longitudinal control state machine will switch the control program to an appropriate state as shown in Fig. 4.1.2.2. Longitudinal control status is sent back to docking coordination together with longitudinal fault status generated by self diagnostics.

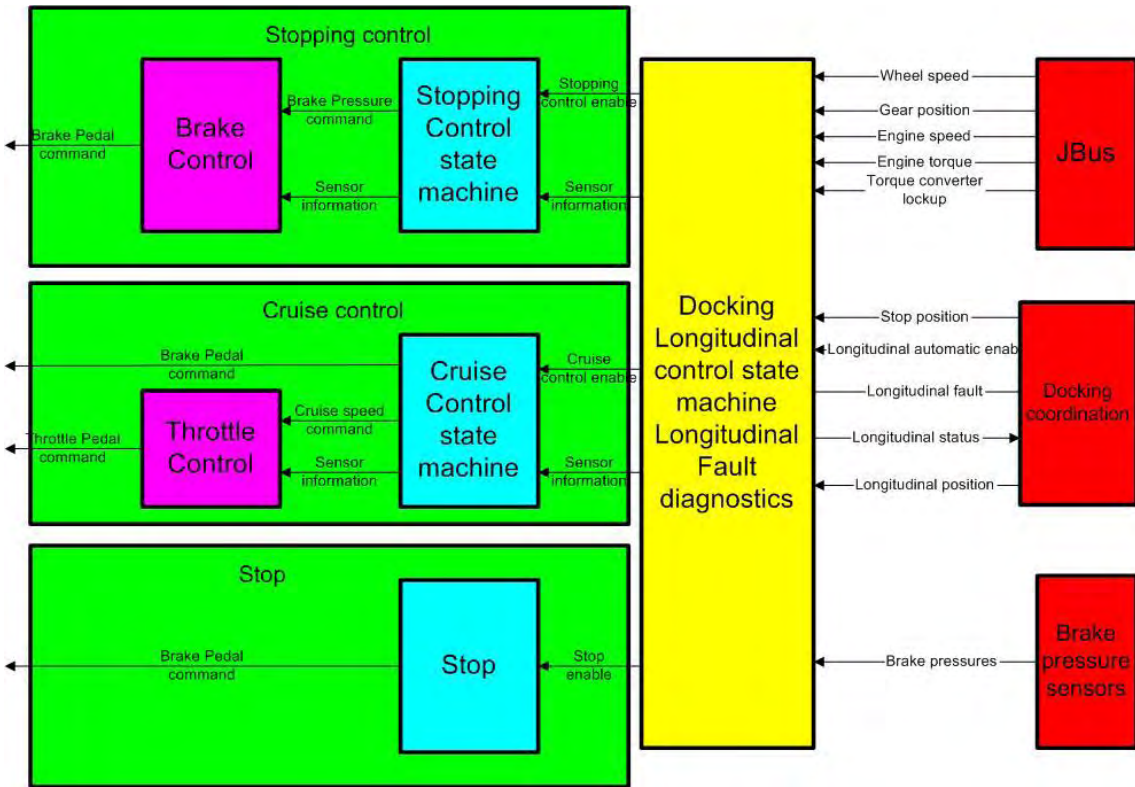


Fig. 4.1.2.1 Docking Longitudinal Control Block Diagram

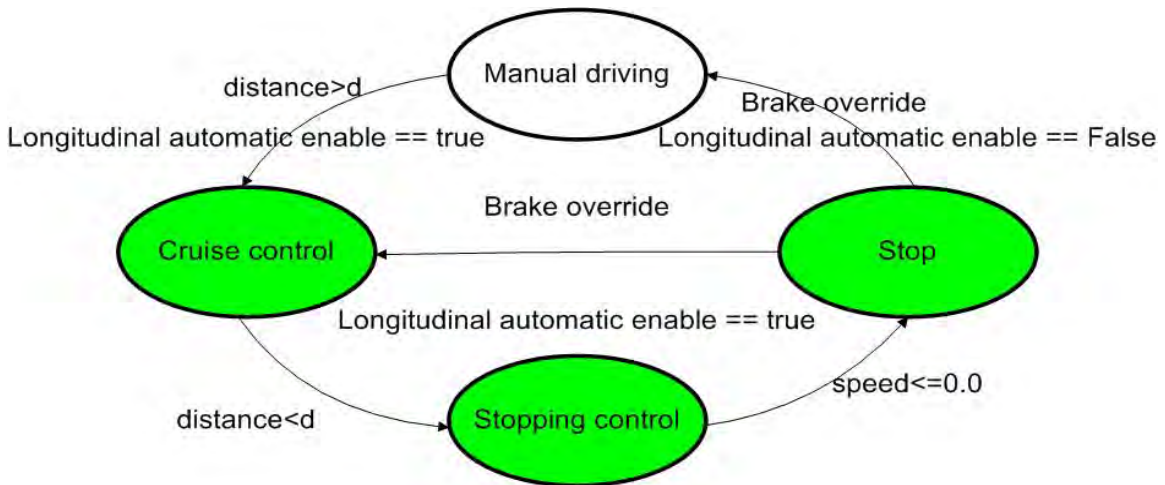


Fig. 4.1.2.2 Longitudinal Control State Machine

4.1.2.2 Cruise Control

A. Controller design

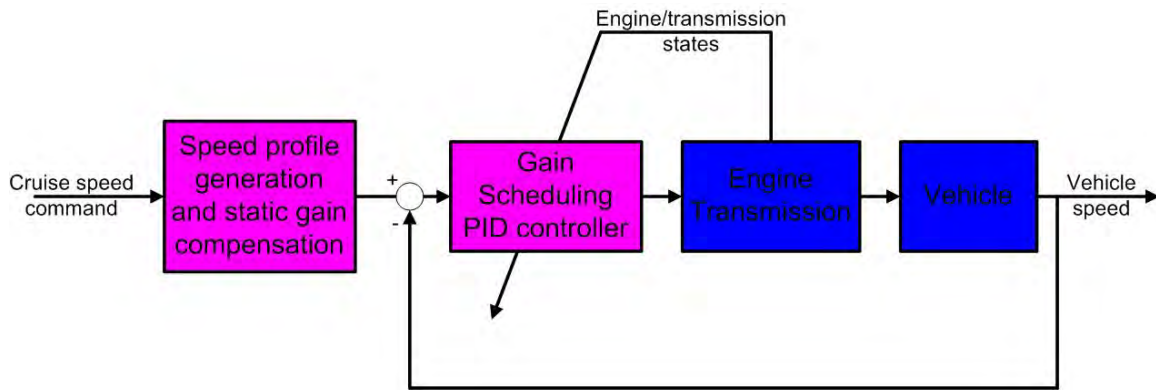


Fig. 4.1.2.3 Cruise Control Block Diagram

During the cruise control period, the bus first accelerates/decelerates to the cruise speed and maintains such speed until it is within a certain distance of the desired stopping location. The cruise control can be formulated into a speed tracking control problem as shown in Fig. 4.1.2.3. A gain scheduled PID controller is designed to track the received cruise speed command. The parameters of PID controller are scheduled according to the engine/transmission state changes (e.g. gear position and torque converter lockup). One thing that needs to be pointed out is that the vehicle speed smoothness is much more important than the speed tracking accuracy in our application here. To ensure a smooth vehicle speed response, two approaches are implemented. First, the parameters of the PID controller are chosen in such a way that bus speed response is always “undershoot”. The explanation can be illustrated as follows. Both brake actuator and throttle actuator are needed in order to get a fast and accurate speed tracking response in a speed tracking controller design. However, the switching of two different actuators may not be smooth for passenger comfort if not designed properly. This is true especially under low speed operation such as docking. An “undershoot” speed response will avoid unnecessary actuator switching and achieve smooth speed response. The speed tracking error can be compensated through static gains. Although this may slow system response, bandwidth and accuracy are not the most important concern here. Second, a smooth speed command profile generation is implemented. It is essentially a second order filter with initial conditions matched to the vehicle starting speed and acceleration.

B. Experimental results

To facilitate illustration, data from eight S-curve docking runs with automatic start are plotted in Fig. 4.1.2.4. The results are quite consistent with 0.3 m/s speed tracking error. Although it is a little bit slow, it is quite smooth according to passengers’ responses.

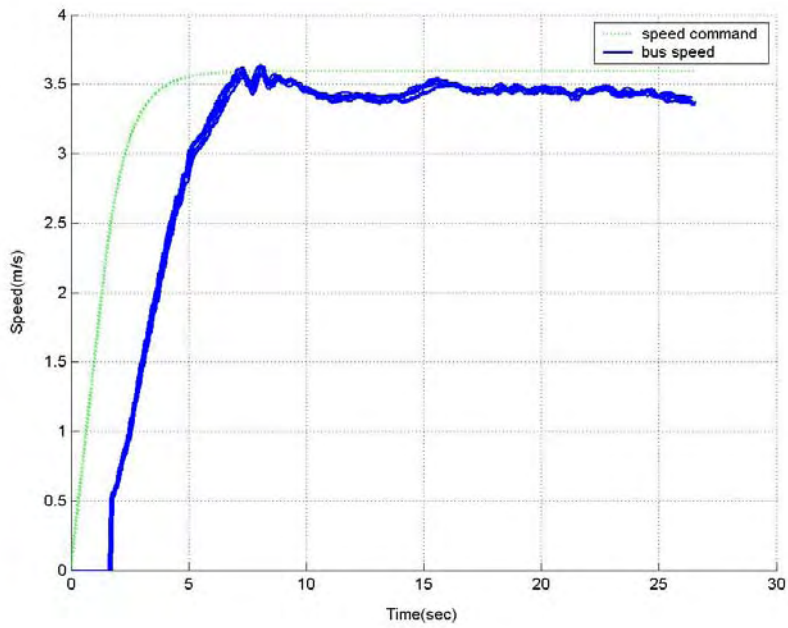


Fig. 4.1.2.4 Speed Tracking Results from San Diego Docking

4.1.2.3 Stopping Control

A. Controller design

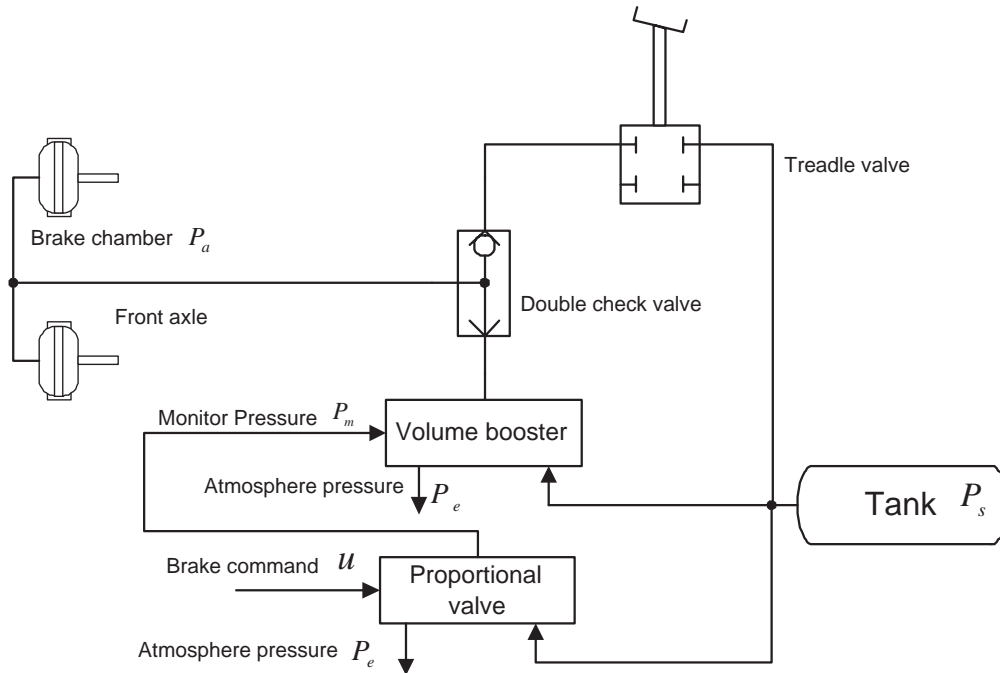


Fig. 4.1.2.5 Retrofitted Brake Actuator

Once the bus enters within 12 meters of the desired stopping location, the longitudinal control system enters the stopping control mode. The purpose of stopping control is to stop the bus at the desired location within 10 centimeters accuracy. This is quite a challenging job due to the following reasons. First, the brake actuator is retrofitted on the existing pneumatic brake system as shown in Fig. 4.1.2.5. This brings in all the inherent nonlinearities of the pneumatic brake system. Fig. 4.1.2.6 shows the stair step response and ramp command response from input command to brake chamber pressure. The stair step response shows a slow pressure response, especially when the brake is in release. Furthermore, the ramp command response shows a 4 psi deadband when the brake is releasing. Second, the bus weight could change significantly from empty to full of passengers. The empty 40 ft CNG bus weighs about 29,500 lb. The full capacity of the bus seating is 38. Assuming 170 lb per person, this will add 6460 lb, about one fifth of the bus' empty weight. Third, the absolute longitudinal position is computed by combining magnet counts and integration of bus speed. However, the magnets are buried in the road with 1 meter spacing and the lowest speed which can be sensed by the bus speed sensor is 0.65 m/s. This means that the absolute position information will be lost at the final stage of stopping, when the speed is below 0.65m/s. This poses significant design difficulty in meeting the 10 centimeter stop accuracy requirement.

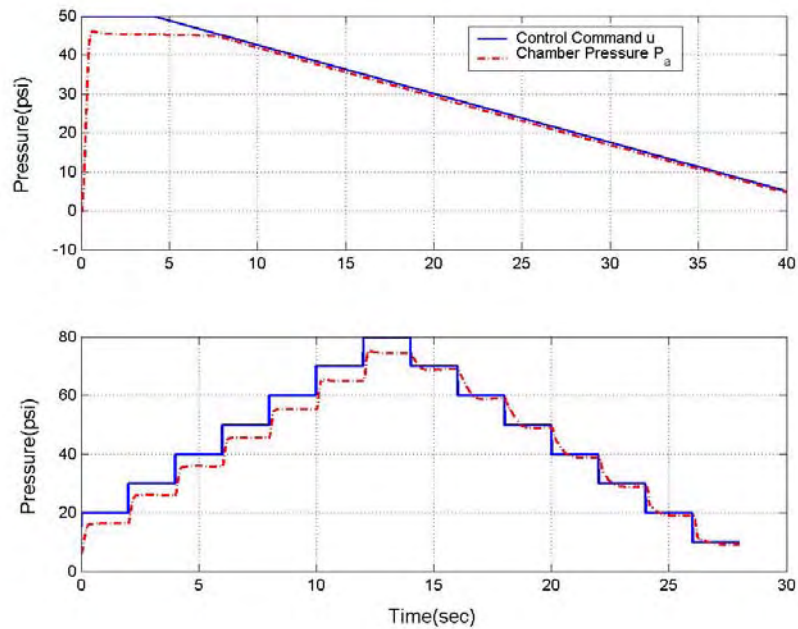


Fig. 4.1.2.6 Nonlinear Response of Brake Actuator

To address the design difficulties mentioned above, control strategies are proposed as shown in Fig. 4.1.2.7. The vehicle states in Fig. 4.1.2.7 include position, speed and acceleration. First, an inner brake pressure servo loop is designed to increase the bandwidth and reduce nonlinearities of the pneumatic brake system. The brake servo controller regulates the brake chamber pressure to the desired command input. Second,

we assume that brake force generated by the pneumatic brake is approximately proportional to the brake chamber pressure.

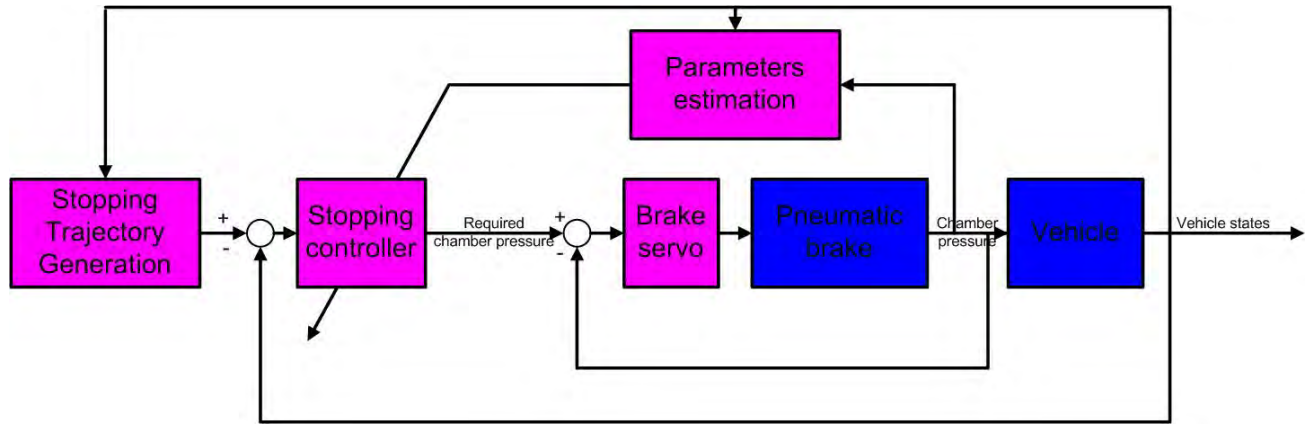


Fig. 4.1.2.7 Block Diagram of Stopping Control

Fig. 4.1.2.8 shows the relationship between vehicle deceleration and brake chamber pressure from experimental data from the two CNG buses, labeled C1 and C2. Despite the fact that these were essentially identical buses from the same manufacturer, it can be seen that the braking performance is noticeably different. As shown in Fig. 4.1.2.8, the brake force (bus deceleration) remains proportional to the brake chamber pressure for chamber pressures larger than 7 psi, which is good enough for our application.

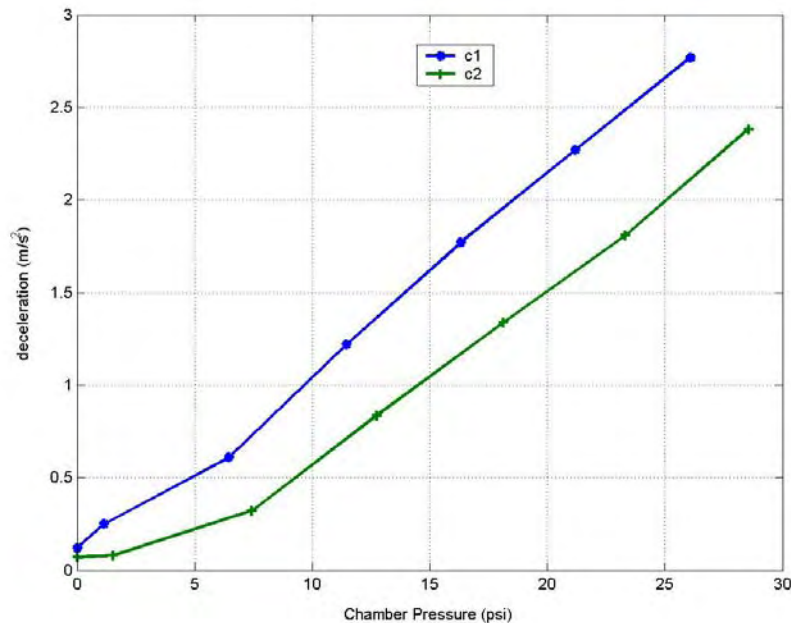


Fig. 4.1.2.8 Bus Deceleration vs. Brake Chamber Pressure

Therefore, the braking force generated by the pneumatic brake system can be expressed by:

$$F_b = -(\eta P_a + d)$$

where P_a is brake chamber pressure, η and d are unknown constants. The bus motion equation can be expressed by

$$\ddot{x} = -\frac{b}{m} \dot{x} - \frac{\eta}{m} P_a - \frac{d}{m}$$

where x is bus displacement, b is unknown viscous damping coefficient and m is bus mass. This shows that the bus equation of motion can be linearly parameterized by a set of unknown constants such as b/m , η/m and d/m . Then different online identification algorithms such as recursive least squares can be used to identify those unknown parameters. Third, the identified parameters will be used for the bus's "last minute" correction when position information is lost.

B. Experimental results

Fig. 4.1.2.9 and Fig. 4.1.2.10 show the experimental data from 10 runs of docking in San Diego. During the remaining 12 meters of stopping control, the speed trajectories converge together gradually as parameter identification picks up for both in-line docking and S- curve docking. Although data shows that bus speed becomes zero at about 0.3 meter before the desired stopping location, this does not mean that the bus stops at 0.3 meter before desired stopping location. This is just because the bus speed is too low for the speed sensor at that time. Measurements after each docking run show that 10 centimeter accuracy is achieved successfully.

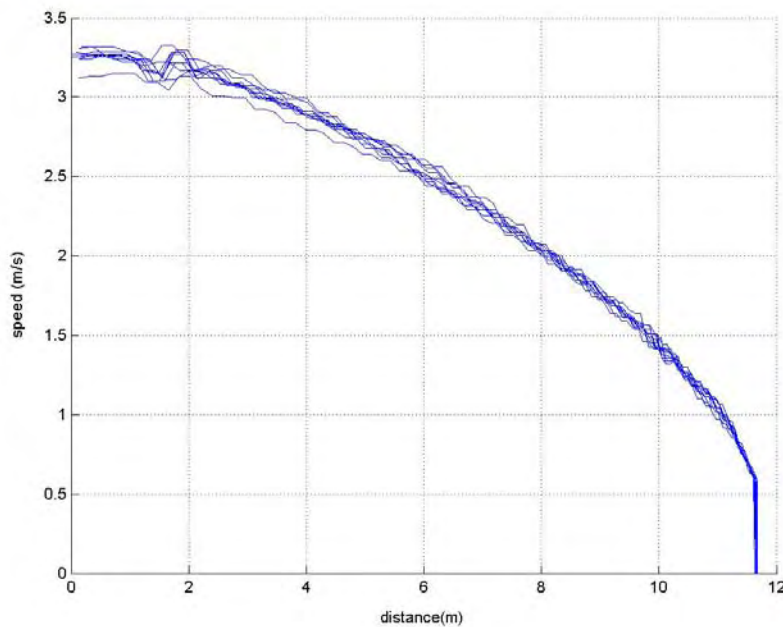


Fig. 4.1.2.9 Speed vs Distance for In-line Docking

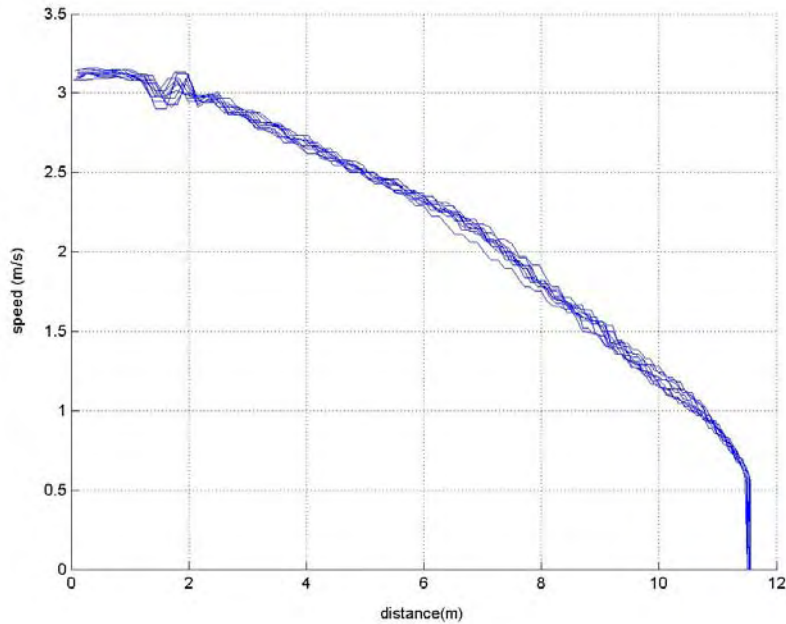


Fig. 4.1.2.10 Speed vs Distance for S-curve Docking

4.2 Bus Lateral Control at Highway Speed

4.2.1 Lane tracking

In principle, the bus lane tracking control system is identical to the bus docking system described in Section 4.1.1. They employ identical vehicle components and have the same control system architecture as shown in Figure 4.1.1.1. The control requirements specified in Section 4.1.1.2 were targeted for both lane tracking and precision docking operations; among them, requirements 1, 2, 4, 5 and 6 in fact address more the performance of lane tracking control. The control design methodology described in Section 4.1.1.3 indicates that a simple frequency-shaped “look-ahead” controller combining a gain scheduled feedback gain and a gain scheduled look-ahead distance scheduled with respect to vehicle speed will achieve all performance requirements in Section 4.1.1.2 for both lane tracking and precision docking control as shown in [11].

4.2.1.1 Lane Tracking Controller

The final lane tracking control algorithm implemented in the bus needs to satisfy both tracking accuracy and ride comfort requirements for all operational scenarios at various bus speeds from precision docking to highway speeds regardless of the following uncertainties: road adhesion variations, incorrect road curvature information, sensor noise, actuator bandwidth, vehicle dynamics changes, soft suspension modes, and vehicle

parameters. The following modified frequency shaped virtual look-ahead lane-keeping control algorithm was developed and implemented for lane tracking control; they are basically identical to Eqs. (4.1.1.5) to (4.1.1.8) for the precision docking control in Section 4.1.3 with an additional nonlinear term, as in the Eq. (4.2.1):

$$\delta_c = -k_c(v)G_c(s)(k_{int}(s)y + d_s(v)G_{ds}(s)\psi) + \delta_{op}(v, \delta_c(t_{i=1,\dots,n}), \dot{\delta}_c(t)) \quad (4.2.1)$$

with

$$K_{int}(s) = \frac{(s + 0.3\pi)}{(s + 0.02\pi)}, G_c(s) = \frac{12\pi}{(s + 12\pi)} \text{ and } \psi \cong \frac{y_f - y_b}{L},$$

and

$$t - \Delta t(v) < t_i < t.$$

The added nonlinear control term, $\delta_{op}(v, \delta_c(t_{i=1,\dots,n}), \dot{\delta}_c(t))$, deals with the 20 to 22 degrees backlash in the steering actuator hydraulic assist system. It injected an open loop command based on the past steering angle behaviors as well as the bus velocity. The two speed dependent gain-scheduled coefficients, $k_c(v)$ and $d_s(v)$, approximate the velocity dependent relationships in Figure 4.1.1.3. Notice that as the vehicle speed is increased, the look-ahead distance is also increased, whereas the feedback gain is reduced, and these gains move moderately with respect to speed between 15 m/s to 25 m/s.

4.2.1.2 San Diego I-15 HOV Lane Test Results

The lane keeping controller was implemented on three buses, two CNG 40-ft buses (C1 & C2), and one diesel 60-ft articulated bus (D1), and tested on the I-15 HOV lanes under general automatic steering scenarios (speed controlled by the drivers) as well as in the platoon scenarios (fully automated).

Four different data comparisons are presented in this report to address the effectiveness of the lane-keeping steering controller presented in Section 4.2.1: (1) three consecutive automatic steering control runs for one CNG bus (C1-40ft) on I-15 HOV north bound lane on 08-03-03; (2) two-bus platoon (C1-CNG 40ft & D1-articulated 60ft) under full automatic control on I-15 HOV south bound lane on 08-18-03; (3) two different runs on two different buses (C1-CNG 40ft & C2-CNG 40ft) under fully automatic platoon control on I-15 HOV north bound lane on 08-18-03 and 08-19-03; and (4) two different runs on the articulated bus (D1-60ft) under fully automatic platoon control on I-15 south bound HOV lane on 08-19-03.

Figure 4.2.1a shows the lateral deviations and bus speeds for three consecutive lane-keeping control runs for one CNG bus (C1-40ft). The driver manually controlled the speeds in all three cases, where the maximum speeds for the Runs 1, 2, and 3 were 60 mph, 46 mph, and 52 mph, respectively. The driver started with manual steering and switched to automatic steering control while the bus started moving. The driver intentionally varied the speeds continuously during each run to demonstrate the

robustness of the controller; notice that there were harsh braking for all three runs close to the end of the track. The standard deviations of the tracking errors under automatic steering control for this CNG bus in Run 1, 2 and 3 are 5.3, 4.5 and 5.1 centimeters, respectively. In all three runs, 99% of the time, the tracking errors are within 15 centimeters. Figure 4.2.1b shows the corresponding steering angles and lateral accelerations during these three runs. The standard deviations of the steering angles commanded by the controller in Run 1, 2 and 3 are 10.2, 10.0 and 10.1 hand-wheel degrees, respectively. It is worthwhile comparing these standard deviations to the roughly 22 degrees of steering wheel backlash. It shows a very tight control under such large nonlinearity. The standard deviations of the lateral accelerations resulted from the automated steering control in Run 1, 2 and 3 are 0.19, 0.13 and 0.19 m/s^2 , respectively. The maximum lateral acceleration is under 0.1 g, including those from the road curvature changes. Figure 4.2.1c repeats Figure 4.2.1.a, but with the Y-axis changing to the milepost where 0-milepost represents the south end of the instrumented HOV lanes. This figure shows that the tracking performances among these three runs are almost identical at every location with different speeds, which also demonstrates the consistency of the automated steering control algorithm.

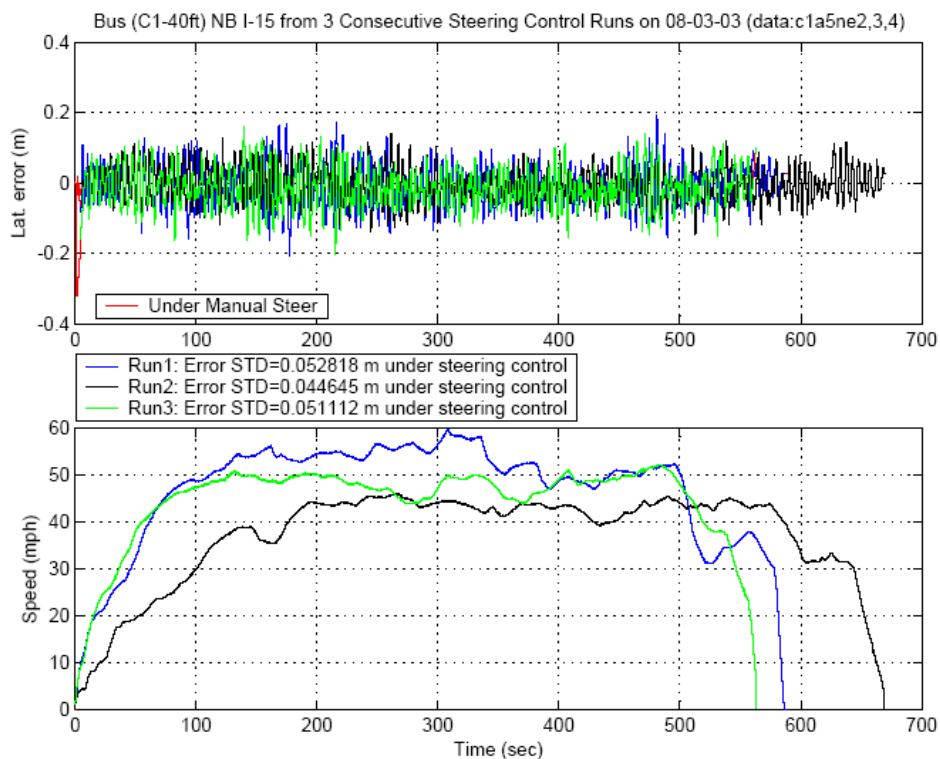


Figure 4.2.1a: CNG Bus (C1-40ft) 3 Consecutive Automatic Steering Control Runs on I-15 HOV lane on 08-03-03 (lateral tracking error & speed w.r.t. time)

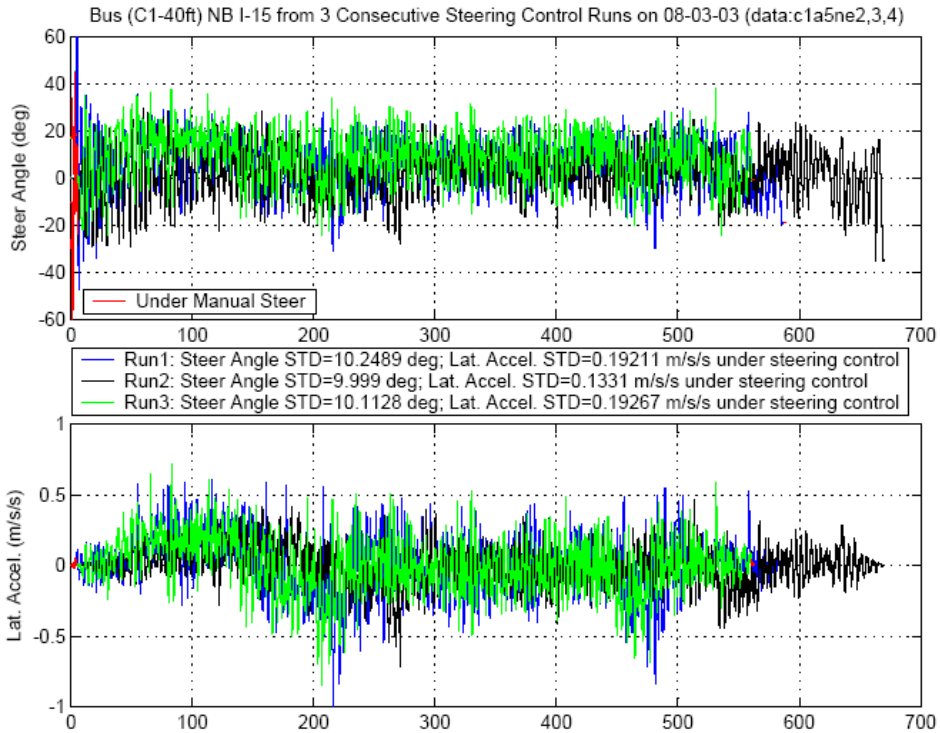


Figure 4.2.1b: CNG Bus (C1-40ft) 3 Consecutive Automatic Steering Control Runs on I-15 HOV lane on 08-03-03 (steering angle & lateral acceleration w.r.t. time)

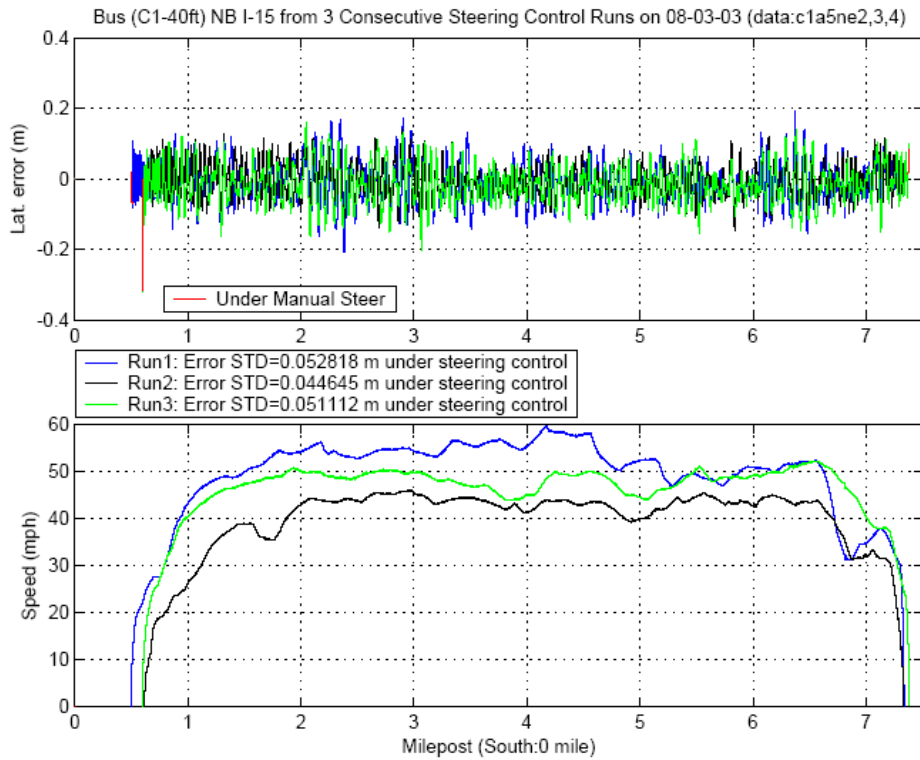


Figure 4.2.1c: CNG Bus (C1-40ft) 3 Consecutive Automatic Steering Control Runs on I-15 HOV lane on 08-03-03 (lateral tracking error & speed w.r.t. milepost (south end=0))

Figure 4.2.2 shows the lateral tracking errors and the speeds resulting from the lane keeping control under the fully automated platoon scenario on two different kinds of buses. The lead bus was a different 40-ft CNG bus (C2) compared to C1 in Figure 4.2.1; the following bus was a diesel 60-ft articulated bus (D1). The distance between the two buses was maintained at 40 meters; the driver switched on automated steering control after the bus reached around 30 mph and switched off at around 40 mph before reaching the south end of the HOV lanes. The standard deviations of the tracking error under automatic steering control for the CNG bus and the articulated bus are 4.6 and 6.8 centimeters, respectively. In all three runs, 99% of the time, the tracking errors are within 15 and 20 centimeters for C2 and D1 bus, respectively.

Figure 4.2.3 shows the lateral deviations and speeds of two different CNG 40-ft buses under fully automated platoon control scenarios from two different platoon runs on I-15 HOV northbound lane. In these two different runs, C1 was the following bus for run 2 (a shorter run), and C2 was the lead bus on run 1. Identical steering control parameters were implemented on both buses. The standard deviations of the tracking error under automatic steering control for the C1 and C2 CNG buses are 5.8 and 4.2 centimeters, respectively. These standard deviations are consistent with the corresponding ones in Figure 4.2.1 and Figure 4.2.2. Figure 4.2.3 also shows that the tracking performance among these two buses on two different runs is almost identical at every location with different speeds, which again demonstrates the consistency of the automated steering control algorithm.

Figure 4.2.4 shows the lateral deviations and speeds of the same Diesel articulated 60-ft bus (D1) under fully automated platoon control from two different runs on I-15 HOV southbound lane. In these two different runs, D1 was always the following bus. The distance between the lead bus and D1 changed from 40 m to 20 m and back to 40 m on Run 1, whereas the distance in Run 2 remained 40 m. The standard deviations of the tracking error under automatic steering control for the D1 articulated bus are 5.3 and 6.1 centimeters for Run 1 and Run 2, respectively. In both runs, the maximum tracking error was under 20 centimeters 95% of the time, and the tracking errors are within 15 centimeters.

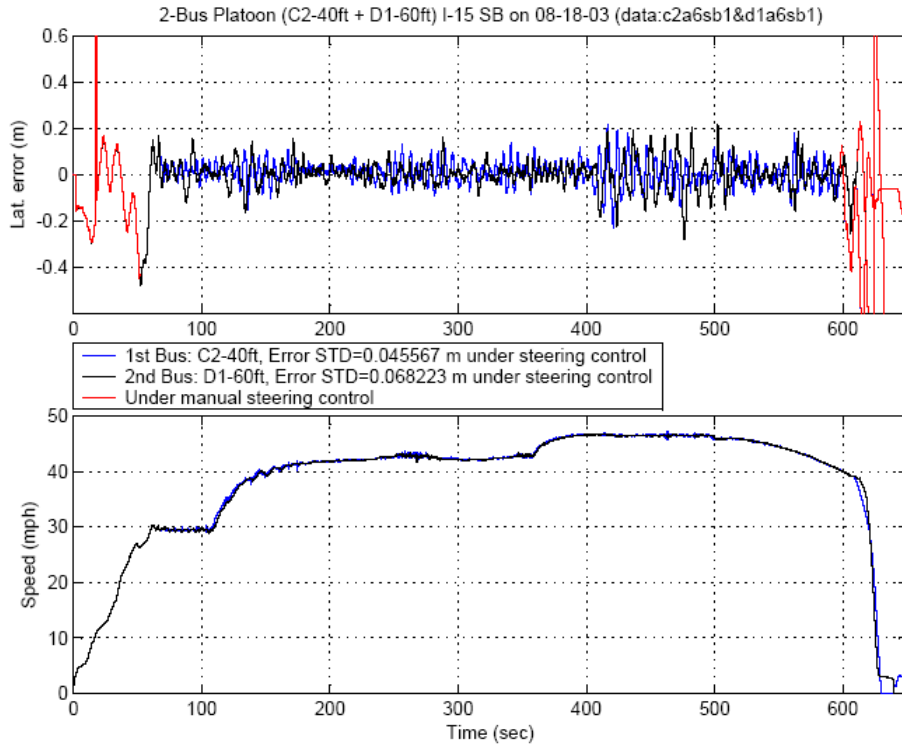


Figure 4.2.2: 2-Bus Platoon (C1-CNG 40ft & D1-articulated 60ft) under Full Automatic Control on I-15 HOV lane on 08-18-03 (lateral tracking error & speed w.r.t. time)

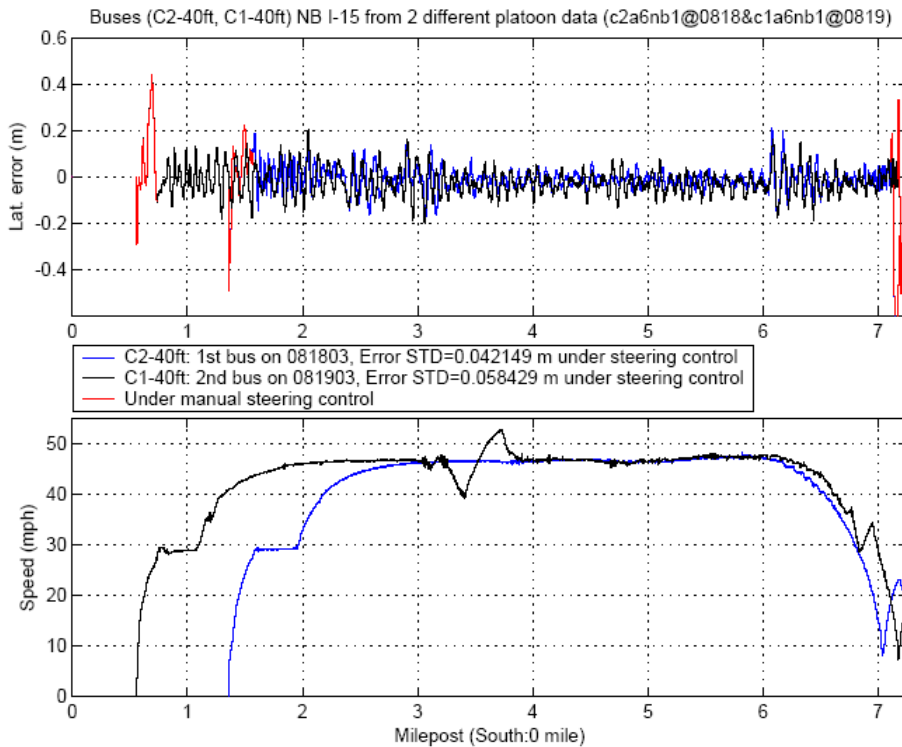


Figure 4.2.3: 2 Buses (C1-CNG 40ft & C2-CNG 40ft) under Automatic Platoon from 2 different runs on I-15 HOV lane (lateral tracking error & speed w.r.t. milepost)

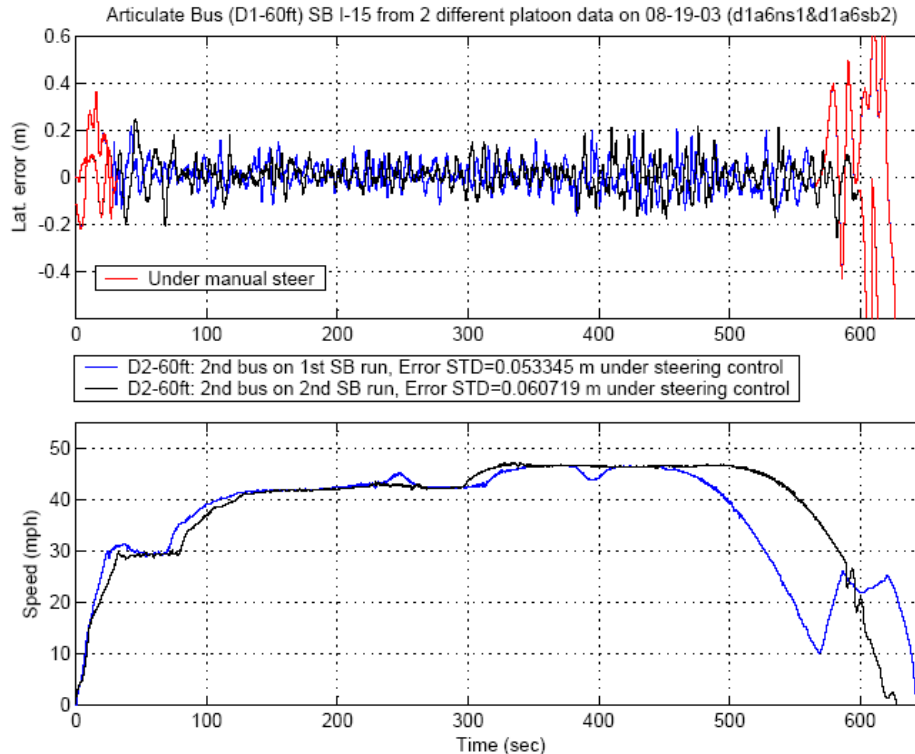


Figure 4.2.4: Articulated Bus (D1-60ft) under Automatic Platoon from 2 different runs on I-15 HOV lane (lateral tracking error & speed w.r.t. time)

4.2.2 Lane changing

The fundamental difference between using look-down or look-ahead sensors for automated lane change maneuver is a result of the difference in the range of the lateral sensors. The look-ahead sensor typically has a larger sensor range. Lane-change and lane-keeping maneuvers become virtually identical to a trajectory-following problem when the lateral sensor sees both lanes simultaneously. The control problem becomes more complicated when the lateral sensor cannot acquire information about either lane and the vehicle must travel a certain distance without sensing any markers during a lane-change. An automated lane-change maneuver therefore involves changing between various control states based on the availability of the lateral sensing measurement. The most uncertain, and thus the most difficult phase of this maneuver usually occurs during the attempt to sense the new marker line in the target lane.

Three basic methods can be devised for conducting automated lane-change maneuvers using range-limited lateral sensors: open-loop lane-change, infrastructure-guided lane-change, and free lane-change using inertial sensors. The difference between these methods lies in the different choices of guiding technique used during the period when no direct road measurement is available between lanes. The open-loop method uses a prescribed steering command; the infrastructure-guided method uses additional pre-installed road markers between lanes at predetermined locations; and the free lane-change method employs dead reckoning with inertial sensors that estimate the vehicle's lateral

states. Except when the “gap” between two lanes is very small, the open-loop lane-change method has proved to be the least robust method because of its extreme sensitivity to vehicle and road uncertainties. Therefore the two approaches that satisfy the above lane-change requirements are infrastructure-guided lane-change and free lane-change using inertial sensors. The free lane-change control is chosen to be the primary lane change method for the bus transit system simply because it provides flexibility to allow an automated bus to conduct lane changes without prior infrastructure development and support.

4.2.2.1 Lane Change Controller Design

An automated vehicle should be able to conduct free lane-changes at any location that is designated for lane-changing even without detailed road information. The major difficulty in performing automated free lane-changes in practice is the high performance sensitivity with respect to sensor noise, vehicle/road parameter variations and any environmental disturbances. Although several methods have been proposed for free lane-change steering control when the lateral sensor senses neither lane during part of the lane-change period, most do not explicitly address concerns about robustness and reliability over the wide range, high noise and large variability of the vehicle operating conditions. Most methods require knowledge of vehicle parameters or exact geometric road information and clean sensor signals for a successful maneuver.

To improve the reliability of the automated free lane-change maneuver, a free lane-change control algorithm was developed and successfully tested in [13] by PATH. It consists of the following four cooperative control schemes:

- (1) a lateral displacement observer/estimator using a yaw-rate sensor (and/or accelerometer) that results in bounded estimation error;
- (2) an adaptive lane-change and lane-catching trajectory-planning scheme that guarantees a smooth final arrival at the target marker line;
- (3) a robust and high damping lane-keeping control algorithm that is capable of tracking the desired trajectory under the worst case scenario; and
- (4) a state machine that quickly coordinates the above schemes and determines the proper vehicle operational-state based on sensor signals, available road information and maneuver demands.

Because a lateral accelerometer is usually noisier and more sensitive to the sensor location, a yaw-rate sensor was chosen to provide the input to the lateral displacement estimator. The purpose of such an estimator is to determine the lateral displacement using the yaw-rate measurement when the lateral sensors detect no markers between lanes. To reduce the sensitivity to dynamic vehicle variations, a more reliable kinematic relationship between lateral displacement and yaw rate is used for the design. The basic estimator is developed in [13] which accounts for three major sources of bias: (1) the yaw-rate sensor bias or drifting; (2) the yaw-rate contribution from the road’s curvature and from the road’s super-elevation; and (3) a slight variation in the estimate of the initial

conditions as the vehicle leaves the marker line. The estimator consists of the following equations and is described as follows:

$$\hat{y} = \hat{y}_0 + V\hat{\theta}_0 t + \frac{1}{2}V\hat{\omega}_0 t^2 + V \iint \omega dt \quad (4.2.2)$$

Equation (4.2.2) describes a “kinematic” estimate of \hat{y} (the lateral displacement) that includes the contributions from bias and the initial conditions, where y_0 , θ_0 and ω_0 are the initial conditions or biases in lateral displacement, vehicle-angle and yaw-rate, respectively. In order to use Eq. (4.2.2) as the displacement estimator during the short “gap” period, the initial conditions (y_0 , θ_0 and ω_0) should be identified. Among them, ω_0 contributes the most toward the estimation error. The vehicle traveling at angle θ_V can be described as:

$$\dot{\theta}_V = \omega_V - \omega_R \quad (4.2.3)$$

where ω_V is the true vehicle yaw-rate, and ω_R is the yaw rate corresponding to the road geometry and vehicle speed. The vehicle angle estimator $\hat{\theta}_V$ can be written as:

$$\dot{\hat{\theta}}_V = \omega_M + K_e(\bar{\theta}_V - \hat{\theta}_V), \text{ with} \quad (4.2.4)$$

$$\omega_M = \omega_V + \omega_N, \text{ and } \bar{\theta}_V = \theta_V + \theta_n,$$

where ω_N is the measurement noise (white noise) combined with the bias (slow varying) of the yaw-rate sensor, $\bar{\theta}_V$ is the computed value of θ_V (e.g., based on displacement measurements), θ_n is the computational error, and K_e is the observer feedback gain. The estimate of ω_0 can be constructed from the following equation by combining Eqs. (4.2.3) and (4.2.4):

$$K_e(\bar{\theta}_V - \hat{\theta}_V) = \frac{-K_e}{s + K_e}(\omega_R + \omega_N) + \frac{K_e s}{s + K_e}\theta_n. \quad (4.2.5)$$

The right hand side of Eq. (4.2.5) consists of a first-order low-pass filtered ($\omega_R + \omega_N$) and a first-order high-pass filtered θ_n . Both filters possess the same corner frequency at K_e . Since ($\omega_R + \omega_N$) is exactly the yaw-rate “bias” (road contribution and sensor bias) that is used in Eq. (4.2.2) after the vehicle leaves the lateral sensor’s range, a low-pass filtered $K_e(\bar{\theta}_V - \hat{\theta}_V)$ as in Eq. (4.2.6) would provide a good estimate for the yaw-rate bias as long as the high frequency portion of the computational noise (from θ_n) is removed.

$$\hat{\omega}_0 = -G_{lp}(s)K_e(\bar{\theta}_V - \hat{\theta}_V). \quad (4.2.6)$$

To effectively reduce the impact from θ_n , the bandwidth of the additional low-pass filter ($G_{lp}(s)$) is chosen to be between $K_e/5$ and $K_e/10$. The delay from this low-pass filter creates the only restriction of such automated lane-change: the lane change should not be initiated right after a change in the road’s curvature until a time span corresponding to the time constant of this low-pass filter is passed. The same technique can also be applied to the kinematic relationship between y and θ_V to obtain the estimate $\hat{\theta}_0$ when the direct

calculation of vehicle angle is not accurate enough. The estimate \hat{y}_0 is usually obtained using the last available displacement measurement. In order to further increase the robustness in practice, additional filters may be added to provide bounds for estimates or to remove specific noises (such as spikes) in the measurements.

Two controller states are needed for the free lane-change scenario: lane-change and lane-catching, as shown in Figure 4.2.5. Although the lane-change state can be subdivided into lane-splitting (using the lateral measurement) and yaw-rate supported lane-change (using the lateral estimate), the trajectory-planning is identical. The difference lies in which lateral displacement (measurement or estimate) is selected for the trajectory tracking control.

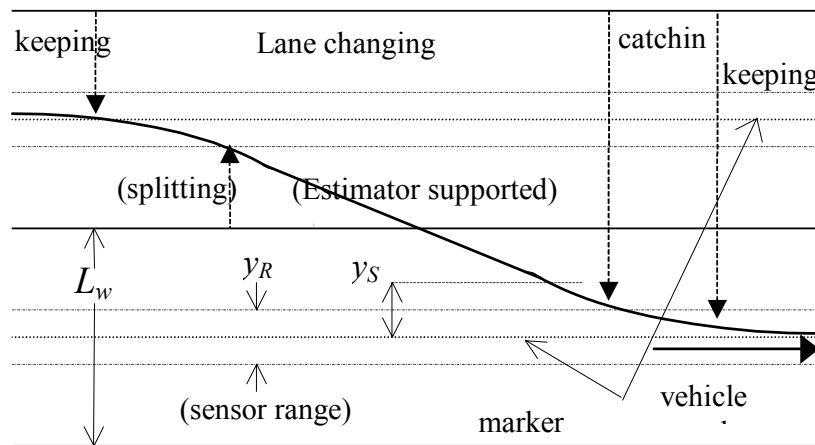


Figure 4.2.5: Controller States for Automated Free Lane-Change

Trajectory-planning is a crucial component in completing the automated free lane-change maneuver. Two trajectories need to be designed:

- (1) Lane-change trajectory: the trajectory that the vehicle follows to leave the traveling lane (based on marker measurement offset) and perform the lane-change maneuver (based on lateral displacement estimate) until the lateral sensor receives the first marker signal from the target lane;
- (2) Lane-catching trajectory: the trajectory that smoothly leads the vehicle to align itself to the new marker line once it senses the first marker.

Although fifth-order polynomials provide a smoother trajectory, third-order polynomials were chosen for field-testing because of their simpler real-time computation and faster initial correction during lane-catching. Since the guided lane-change maneuvers are performed based on marker locations, the trajectories are defined as functions of the longitudinal positions as shown in Equation (4.2.7). The bus tracks the desired trajectory ($y = y_d(x)$) instead of the road center ($y = 0$). The desired trajectory is described as:

$$y_d(x) = a_3x^3 + a_2x^2 + a_1x + a_0, \text{ with} \quad (4.2.7)$$

$$\begin{aligned}
a_0 &= y_d(0), \\
a_1 &= \dot{y}_d(0) = \frac{dy_d}{dx}(0), \\
a_2 &= -\frac{2(y_d(x_f) - y_d(0))}{x_f^3} + \frac{\dot{y}_d(x_f) + \dot{y}_d(0)}{x_f^2}, \\
a_3 &= \frac{3(y_d(x_f) - y_d(0))}{x_f^2} - \frac{\dot{y}_d(x_f) + 2\dot{y}_d(0)}{x_f}.
\end{aligned}$$

Table 4.2.1 shows the coefficients of a typical application of Equation (4.2.7) to lane-change, and lane-catching scenarios, with L_w the lane width and ψ the designated final approaching angle. The length of the trajectory planning, x_f , is usually adaptive to vehicle speed so that the time of dead reckoning is limited to 3-5 seconds. However for simplicity, a fixed number between 150 to 200 m for lane-changes and 20 to 40 m for lane-catching is also applicable. Furthermore, the lane-change trajectory modifies itself if the vehicle does not reach the target lane after a prescribed time interval.

	$y_d(0)$	$\dot{y}_d(0)$	$y_d(x_f)$	$\dot{y}_d(x_f)$
lane-change	0	0	$\pm L_w$	$\pm \psi$
lane-catching	$y(t_a)$	θ_a	0	0

Table 4.2.1 Trajectory-planning for free lane-change

The lane-keeping controller is essential for the robustness of the lane change maneuver. Not only it is the final safeguard of this automated maneuver, but the strength of the lane-keeping controller also determines the allowable maximum estimator errors. The controller has to bring the vehicle back to the lane-keeping function even at the largest possible arrival angle created by the estimator errors.

4.2.2.2 San Diego I-15 HOV Lane Test Results

The lane change controller were implemented on all buses: two CNG 40-ft buses (C1 & C2), and one diesel 60-ft articulated bus (D1), and tested on the I-15 HOV lane under the general automatic steering scenarios where the bus speed was controlled by the drivers. The lane change controller is designed to function at any speeds, as well as under severe braking conditions.

Figure 4.2.6a – 4.2.6c illustrate a nine-minute test run for one CNG 40-ft bus (C1) under automatic steering control with driver controlling the throttle and brake on I-15 HOV southbound lane on 08-20-03. This test run demonstrates the following lateral control functions: Manual/Auto Transition, Automated Lane Change, and Automated Lane Keeping, under various speed variations including Hard Braking.

Figure 4.2.6a illustrates the field test results of automated free lane-change maneuvers on the I-15 test track. The bus started from the right lane on the southbound travel lanes. Twelve consecutive free lane-changes were recorded as the driver pushed a sequence of lane-change requests using the driver DVI button after an automated lane change was completed and as he desired. After a request was made, an automated lane change would be commanded at the next “Lane Change Permit” area. As an example, the first automated lane change was a left lane change at 45 mph (at 109 second). The bus first followed a desired lane-change trajectory (third-order polynomials as in Eq. 4.2.7) guided by the magnetic measurements until it left the sensor range at 1.12 m (right side of the bus). The bus continued to track the same desired trajectory using the estimated displacement instead of the measurement until the sensor saw the first magnet on the target lane (right lane) at -1.19 m (left side of the bus) away from the new lane center. At this time, the estimator reading was 2.23 m. The error of the estimate was 0.18 m since the correct estimation (at the first marker reading) should be about ± 2.45 m (the lane width, 3.6 m, minus the sensor range, ~ 1.15 m). The bus then switched to the lane catching function with a new desired trajectory calculated by third-order polynomials whose coefficients are determined by the initial catching condition as in Table 4.2.1. Finally, the vehicle resumed the lane-keeping function after finishing lane-catching. According to Figure 4.2.6a, the resulting estimator errors for the twelve consecutive lane changes were: 0.18, 0.14, 0.39, 0.14, 0.25, 0.10, 0.20, 0.30, 0.15, 0.24, 0.31, and 0.39 meters. The controller needs to be robust enough to recover from these estimation errors. Furthermore, these lane changes were performed with speeds ranging from 40 mph to 53 mph. Notice also that #3, #11 and #12 lane changes were conducted under emergency braking; #4 and #6 were under full throttle; and #2 were done while the driver changed from braking to throttling.

Figure 4.2.6b shows the corresponding steering angles and the control states for the same test run. The steering angles have a standard deviation of 11.7 degrees which is somewhat larger than the roughly 10 degrees of the three cases in Figure 4.2.1b. The 1.7 degree increase is likely due to the fact that (1) there were 56 automatic/manual transitions and (2) there were also 12 lane changes.

Figure 4.2.6c illustrates the blowout plots for the lateral tracking error, steering angle, control state, and driver transition switch status for the same test run. The figures in Figure 4.2.6c show clearly the responsiveness of the control state with respect to the driver demand. The controller exhibits its “on-demand” property by changing the state as soon as the driver pushes the switch on or switch off buttons. The “controller” also responds to the control state immediately and smoothly. The smoothness can be easily observed in the smoothness in both the steering angles and the lateral deviations in between different control states. In addition, the lane-keeping controller always brought the lateral displacement toward zero (smoothly) whenever the driver transitions to automatic control.

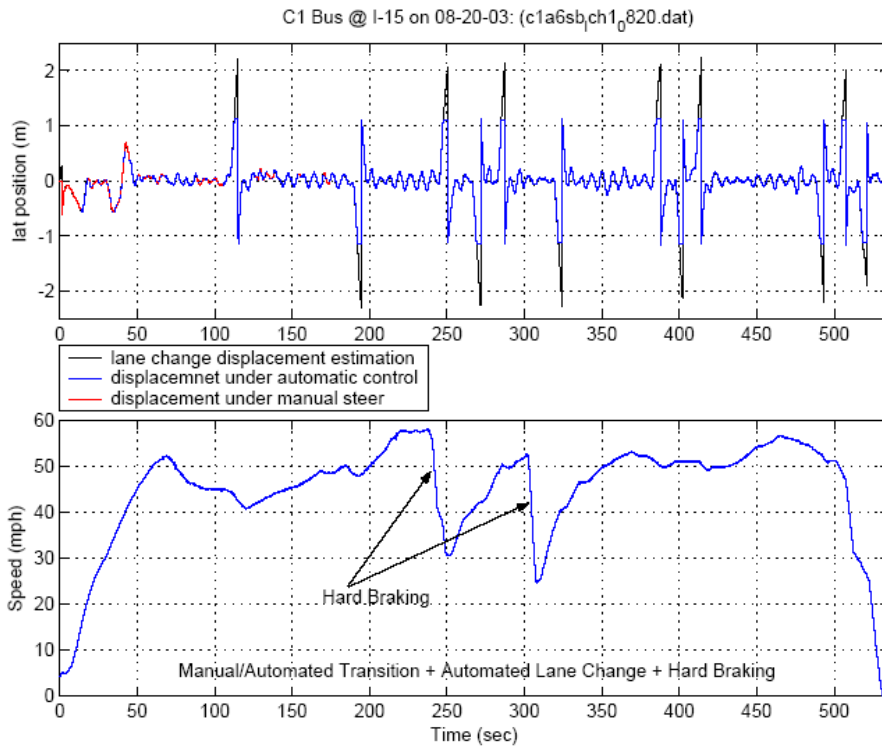


Figure 4.2.6a: CNG Bus (C1 40ft) under Automatic Steering Control with Manual/Auto Transition, Automated Lane Change, with Hard Braking on I-15 HOV lane on 08-20-03 (lateral tracking error & speed w.r.t. time)

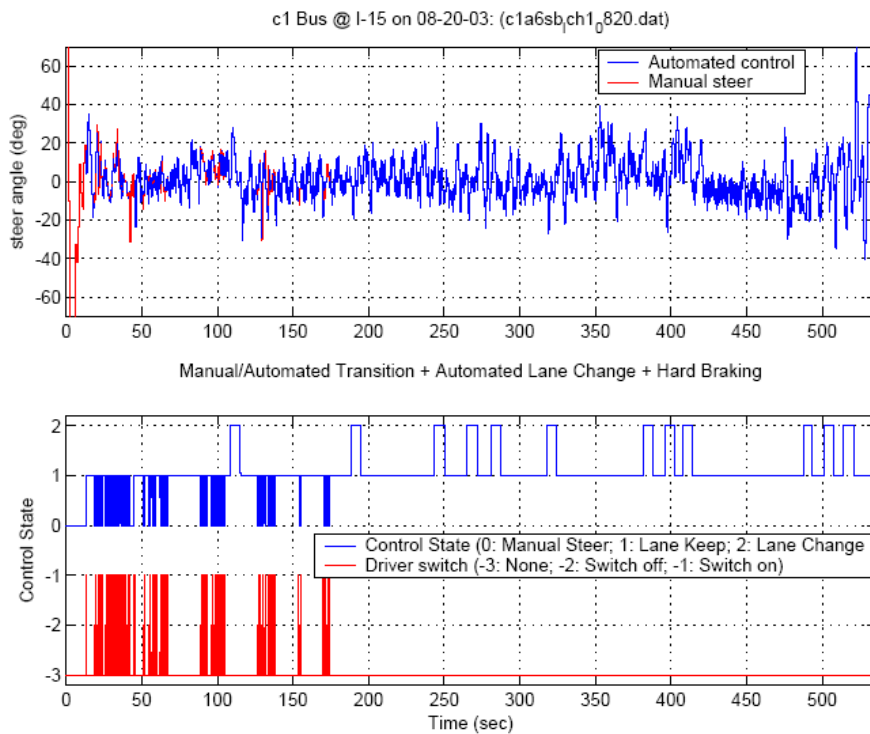


Figure 4.2.6b: CNG Bus (C1 40ft) under Automatic Steering Control with Manual/Auto Transition, Automated Lane Change, with Hard Braking on I-15 HOV lane on 08-20-03 (steering angle, control state, and driver transition switch w.r.t. time)

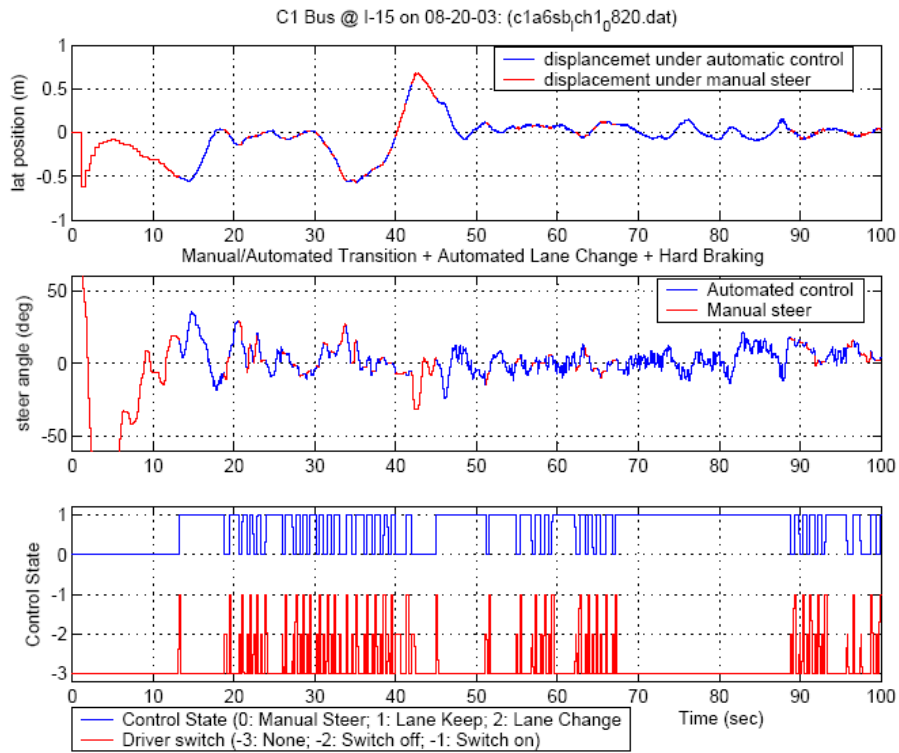


Figure 4.2.6c: CNG Bus (C1 40ft) under Automatic Steering Control with Manual/Auto Transition, Automated Lane Change, with Hard Braking on I-15 HOV lane on 08-20-03 (lateral tracking error, steering angle, control state & driver transition switch w.r.t. time)

4.3 Bus Longitudinal Control

The objective of the longitudinal controller is to follow either a given desired velocity or distance profile using the three control inputs, namely the engine torque, transmission retarder braking torque, and pneumatic braking torque. Based on the vehicle model defined in Section 2.1.3.1, the longitudinal controller is designed to achieve desired control objectives.

4.3.1 Tracking a speed profile

In this section, the control laws for determining the desired engine and brake torque will be designed using dynamic sliding control (DSC). Due to the successful implementation of DSC previously on the PATH passenger vehicles [1,2], and its extensibility to the transit bus models, the speed control law will be derived briefly without detailed descriptions.

① Preliminary design of speed controller

Using the terminology of sliding mode control, the sliding error surface representing the velocity tracking error is defined as $S_{1e} := v - v_{des}$. Then, after differentiating S_{1e} and using equation (2.1.1), we can write the forcing term \bar{T}_e and filter dynamics as follows:

$$\dot{S}_{1e} = \frac{T_e - T_{acc} - R_g \cdot T_b}{J_{eq}} - f_1 - \dot{v}_{des}$$

$$\bar{T}_e = T_{acc} + R_g \cdot T_b + J_{eq} (f_1 + \dot{v}_{des} - \lambda_{1e} S_{1e}) \quad (4.3.1a)$$

$$\tau_{2e} \dot{T}_{edes} + T_{edes} = \bar{T}_e, \quad T_{edes}(0) = \bar{T}_e(0) \quad (4.3.1b)$$

where T_b is zero for engine control and λ_{1e} is the control gain of the first sliding surface. Subsequently, after differentiating $S_{2e} := T_e - T_{edes}$ and using equation (2.1.3), the subsequent surface error dynamics and desired engine torque command are:

$$\dot{S}_{2e} = \dot{T}_e - \dot{T}_{edes} = \frac{1}{\tau_e} (-T_e + T_{ctrl}) - \dot{T}_{edes} \Rightarrow T_{ctrl} = T_e + \tau_e (\dot{T}_{edes} - \lambda_{2e} S_{2e}) \quad (4.3.1c)$$

$$T_{ctrl} = T_e + \tau_e (\dot{T}_{edes} - \lambda_{2e} \tilde{S}_{2e}) \quad (4.3.1d)$$

where $\tilde{S}_{2e} := T_{e,meas} - T_{edes}$, i.e., the engine torque value defined in equation (2.1.2) is used to calculate \tilde{S}_{2e} , which might be slightly different from S_{2e} , and λ_{2e} is the control gain of the second sliding surface. Finally, desired commands for engine control are following:

$$\begin{cases} T_{cmd} = T_{ctrl} & \text{for a diesel engine} \\ u_\alpha = q_e(\omega_e, T_{ctrl}) & \text{for a CNG engine} \end{cases} \quad (4.3.1e)$$

where q_e is an inverse map of $T_{map}(\omega_e, u_\alpha)$ in equation (2.1.3). It is noted that the desired engine torque command (T_{ctrl}) is used as TCC for the diesel engine and is fed into the inverse engine map to calculate the pedal position command for the CNG engine.

A control law for the brake system can be derived similarly by defining $S_{1b} := S_{1e}$. After following the similar procedure, the corresponding equations for the brake system are:

$$\bar{T}_b = \frac{1}{R_g} [T_{ect} - T_{acc} - J_{eq} (f_1 + \dot{v}_{des} - \lambda_{1b} S_{1b})] \quad (4.3.2a)$$

$$\tau_{2b} \dot{P}_{bdes} + P_{bdes} = \bar{T}_b / K_b, \quad P_{bdes}(0) = \bar{T}_b(0) / K_b \quad (4.3.2b)$$

$$\bar{P}_{bv}(u_\beta) = P_b + \tau_b (\dot{P}_{bdes} - \lambda_{2b} S_{2b}) \Rightarrow u_\beta = q_b(\bar{P}_{bv}) \quad (4.3.2c)$$

where T_{ect} is the minimum or closed throttle torque, $S_{2b} := P_b - P_{bdes}$, and q_b is the inverse function of P_{bv} . Once the preliminary design of the longitudinal controller is completed, the appropriate choice of the controller gains, $\{\lambda_{1i}, \lambda_{2i}, \tau_{2i}\}$ for $i = e, b$, will be discussed next.

② Analysis and design of speed controller

Consider the closed-loop error dynamics with DSC law for engine control. After subtracting and adding T_{edes} and \bar{T}_e in equation (2.3.1) and using the definition of \bar{T}_e and T_{edes} shown in equations (4.3.1a) and (4.3.1d), the closed-loop system equations are written as

$$\begin{cases} \dot{v} = \frac{\{\bar{T}_e + (T_e - T_{edes}) + (T_{edes} - \bar{T}_e)\} - T_{acc} - f_1 + \Delta f_1}{J_{eq}} \\ \dot{T}_e = \frac{1}{\tau_e} \{-T_e + T_{ctrl}\} + \Delta f_{2e} \\ \dot{\xi}_{2e} = \dot{T}_{edes} - \dot{\bar{T}}_e = -\xi_{2e} / \tau_{2e} - J_{eq} (\dot{f}_1 + \ddot{v}_{des} - \Lambda_{1e} \dot{S}_{1e}) \end{cases} \quad (4.3.3)$$

where $\xi_{2e} = T_{edes} - \bar{T}_e$ is the low-pass filter error of the DSC. Then, using the definition of S_{1e} , S_{2e} , and T_{ctrl} shown in Eq. (6), the augmented error dynamics shown above can be rewritten in terms of the surface and filter errors as follows:

$$\begin{cases} \dot{S}_{1e} = -\Lambda_{1e} S_{1e} + (S_{2e} + \xi_{2e}) / c_{1e} + \Delta f_1 \\ \dot{S}_{2e} = -\Lambda_{2e} S_{2e} + \Delta f_{2e} \\ -c_{1e} \Lambda_{1e} \dot{S}_{1e} + \dot{\xi}_{2e} = -\xi_{2e} / \tau_{2e} - c_{1e} \cdot \dot{f}_1 - c_{1e} \cdot \ddot{v}_{des} \end{cases} \quad (4.3.4)$$

where the parameter $c_{1e} = J_{eq}$. Similarly, when the brake pressure is greater than the push-out pressure (P_o), the longitudinal equation for brake control is presented as follows

$$\begin{cases} \dot{v} = \frac{T_{ect} - T_{acc} - R_g \cdot K_b \cdot [\bar{P}_b - P_o + (P_b - P_{bdes})]}{J_{eq}} - \frac{R_g \cdot K_b \cdot (P_{bdes} - \bar{P}_b)}{J_{eq}} + \Delta f_1 \\ \dot{P}_b = \frac{1}{\tau_b} [-P_b + P_{bv}(u_\beta)] + \Delta f_{2b} \\ \dot{\xi}_{2b} = \dot{P}_{bdes} - \dot{\bar{P}}_b = -\frac{\xi_{2b}}{\tau_{2b}} + \frac{J_{eq}}{K_b \cdot R_g} (\dot{f}_1 + \ddot{v}_{des} - \Lambda_{1b} \dot{S}_{1b}) \end{cases} \quad (4.3.5)$$

where $\xi_{2b} = P_{bdes} - \bar{P}_b$. Then, using the definition of S_{1b} , S_{2b} , and u_β shown in equation (4.3.2c), the error dynamics for the case of the braking control are

$$\begin{cases} \dot{S}_{1b} = -\Lambda_{1b}S_{1b} + (S_{2b} + \xi_{2b})/c_{1b} \\ \dot{S}_{2b} = -\Lambda_{2b} \cdot S_{2b} \\ -c_{1b}\Lambda_{1b}\dot{S}_{1b} + \dot{\xi}_{2b} = -\xi_{2b}/\tau_{2b} - c_{1b} \cdot \dot{f}_1 - c_{1b} \cdot \ddot{v}_{des} \end{cases} \quad (4.3.6)$$

where $c_{3b} = -\frac{J_{eq}}{K_b \cdot R_g}$. Finally, combining the equations (4.3.4) with (4.3.6), we can write

the switched error dynamics as follows: for $i = e, b$,

$$T_i \dot{z}_i = \bar{A}_i z_i + \bar{B}_{w,i} w_i + \bar{B}_{r,i} r \Rightarrow \dot{z}_i = A_i z_i + B_{w,i} w_i + B_{r,i} r \quad (4.3.7)$$

where $z_i = [S_{1i} \ S_{2i} \ \xi_{2i}]^T \in \mathfrak{R}^3$, $w_i = [\Delta f_1 \ \Delta f_{2i} \ \dot{f}_1]^T \in \mathfrak{R}^3$,

$$r = [v_{des} \ -v_{des}(0) \ \dot{v}_{des} - \dot{v}_{des}(0) \ \ddot{v}_{des} - \ddot{v}_{des}(0) \ 1]^T \in \mathfrak{R}^4$$

and the system matrices are following:

$$T_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -c_{1i}\Lambda_{1i} & 0 & 1 \end{bmatrix}, \bar{A}_i = \begin{bmatrix} -\Lambda_{1i} & 1/c_{1i} & 1/c_{1i} \\ 0 & -\Lambda_{2i} & 0 \\ 0 & 0 & -1/\tau_{2i} \end{bmatrix}, \bar{B}_{w,i} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -c_{1i} \end{bmatrix} \text{ and}$$

$$\bar{B}_{r,i} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -c_{1i} & c_{1i}\ddot{v}_{des}(0) \end{bmatrix}$$

In consequence, it is said that the augmented error dynamics with DSC can be described as linear error dynamics subject to perturbation terms, w_i . In the case of robust tracking for the uncertain nonlinear system, the DSC does not drive the error to zero, but the error will be quadratically bounded within a small region around the desired trajectory [3].

The quadratic boundedness for a tracking problem of a nonlinear system via DSC was proposed in [4]. Moreover, based on simultaneous stability in linear control theory [5], the results in [4] can be extended to consider a switched nonlinear system, e.g., engine and brake control for automated transit bus systems here. Furthermore, these were applied for a set of nonlinear systems to consider possible faults [6]. Since all mathematical details can be referred to [6], the algorithm to obtain appropriate control gains (λ_i and τ_i) is written without descriptions as follows:

Algorithm:

- i. Choose a set of surface gains (λ_i) and filter time constants (τ_i) which makes at least A_i shown in equation (4.3.7) be Hurwitz and satisfies design constraints, e.g., all τ should be greater than a minimum value.
- ii. Choose the matrices $C_{z,i}$ and $D_{r,i}$ such that $|w_i| \leq |C_{z,i} z + D_{r,i} r|$ for all $z \in D$.
- iii. Find the smallest ε_P by solving the following problem for a fixed κ :

maximize t
subject to $P \geq tI, \Sigma \geq 0$, and

$$\begin{bmatrix} A_i^T P + P A_i + \delta P + C_{z,i}^T \Sigma C_{z,i} & P B_{w,i} & P \bar{B}_{r,i} + C_{z,i}^T \Sigma \bar{D}_{r,i} \\ B_{w,i}^T P & -\Sigma & 0 \\ \bar{B}_{r,i}^T P + \bar{D}_{r,i}^T \Sigma C_z & 0 & \bar{D}_{r,i}^T \Sigma \bar{D}_{r,i} - \kappa I \end{bmatrix} \leq 0 \quad (4.3.8)$$

where the reference input has bounded peaks, i.e. $r^T r \leq r_0$ where r_0 is the bound on the peak, $\bar{B}_r = \sqrt{r_0} B_r$, and $\bar{D}_r = \sqrt{r_0} D_r$.

In order to provide an effective controller performance analysis, the above problem (4.3.8) in step (iii) is to find the ellipsoidal error bound for the DSC. However, the ellipse should also be “small” in some sense, so that the tracking performance will be estimated. The measure of size used in this paper is the largest semi-axis length of the ellipse [19]. Using this measure, the minimization of the largest semi-axis length can be posed as the semi-definite programming problem shown in (4.3.8). Repeating procedure of step (i) through (iii) provides analysis and design method for choosing surface gains and time constants of DSC.

③ Switching criterion

The specific control mode of the vehicle is determined by switching logic based on the desired and residual acceleration computed by the engine control law. The residual acceleration is defined as

$$a_{resid} = \frac{T_{ect} - T_{acc}}{J_{eq}} - f_1$$

and represents the acceleration of the vehicle as a result of closed-throttle-torque, rolling resistance, and aerodynamic drag. For example, if the engine controller computes $a_{syn} \geq a_{resid}$ where $a_{syn} := \dot{v}_{des} - \lambda_{1e} S_{1e}$, then engine control is used. However, if $a_{syn} < a_{resid}$, then brake control is used [7, 2]. Once the brake control is activated, it should be decided whether the transmission retarder braking torque is enough or additionally a pneumatic braking torque is necessary. One of the simplest methods is to use the maximum torque of the transmission retarder (T_{max}) as follows:

$$\begin{cases} T_{tr} = \bar{T}_b \\ u_\beta = 0 \end{cases} \text{ if } \bar{T}_b \leq T_{max}, \quad \begin{cases} T_{tr} = T_{max} \\ u_\beta = q_b(\bar{P}_{bv} - T_{max} / K_b) \end{cases} \text{ otherwise}$$

It is remarked that small hysteresis for the switching criterion was used to prevent the potential chatter due to sensor noise, finite sampling rate, and model uncertainties [7].

④ Experimental validation and performance

As the model validation was performed in the flat open space mentioned in Section 2.1.3.1, high speed tests using two transit buses were conducted at Crows Landing in California. As shown in Figure 4.3.1, the desired velocity is given ranging from 0 to 23 (m/s) with the maximum acceleration, 0.3 (m/s²). When the controller gains are assigned

as $\{\lambda_{1e}, \lambda_{2e}, \tau_{2e}\} = \{1.2, 25, 0.02\}$ and $\{\lambda_{1b}, \lambda_{2b}, \tau_{2b}\} = \{1.0, 20, 0.02\}$ for the longitudinal control of a 40-ft transit bus, the velocity tracking performance is shown in the bottom plot of Figure 4.3.1. Moreover, to validate robustness of the proposed longitudinal controller, a lane-change maneuver was conducted several times and an air conditioner was switched on and off during the experimental test. Once the torque converter is locked (after 23 seconds in Figure 4.3.1), the tracking error remains within 0.6 (m/s) even under system parameter disturbances, and it is on the order of the wheel speed sensor resolution, ± 0.28 (m/s).

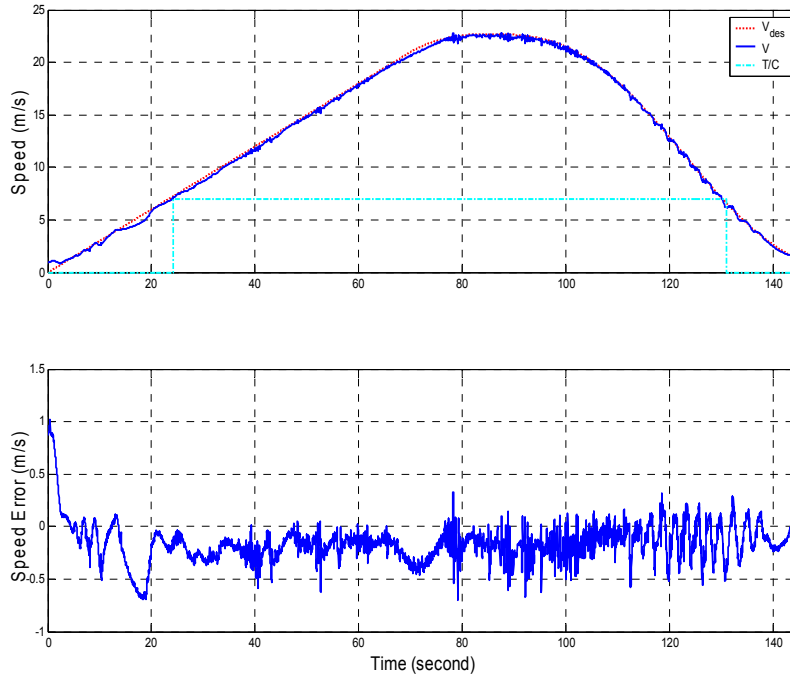


Figure 4.3.1 Time Responses of Speed and Speed Tracking Error at Crows Landing: a 40-ft CNG Transit Bus

Similarly, when the controller gains are assigned as $\{\lambda_{1e}, \lambda_{2e}, \tau_{2e}\} = \{1.2, 25, 0.02\}$ and $\{\lambda_{1b}, \lambda_{2b}, \tau_{2b}\} = \{1.0, 20, 0.02\}$ for a 60-ft articulated bus, the velocity tracking performance is shown in the bottom plot of Figure 4.3.2, and its tracking error remains within 0.7 (m/s). There are two interesting remarks from the experimental results. One is that the controller gains used above were obtained from the algorithm proposed in the previous section and their small variation was applied to improve the performance. The other is that the unified speed tracking controller was working well for the two different types of transit buses using the same gains.

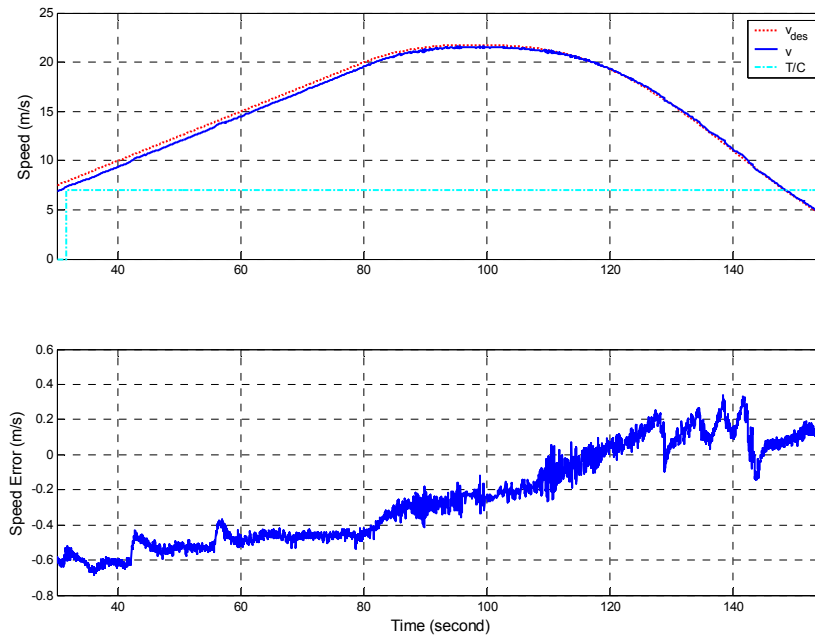


Figure 4.3.2 Time Responses of Speed and Speed Tracking Error at Crows Landing: a 60-ft Diesel Articulated Bus

While the longitudinal controller was tested on the flat test road mentioned above, its performance and robustness with respect to road grade should be considered from the viewpoint of real implementation. For instance, Figure 4.3.3 shows road elevation and grade of a highway section on I-15 in San Diego, California, which was used for the demonstration in August of 2003. Figure 4.3.4 shows time responses of a 40-foot CNG transit bus in terms of velocity and accelerator pedal position on I-15. The results shown in Figure 4.3.4 demonstrate how well the longitudinal control algorithm makes the bus track a desired speed profile even in the presence of road grade. From these, it is concluded experimentally that the proposed longitudinal controller is robust enough to track the desired speed without considering road grade. More specifically, once the torque converter is locked, the tracking error remains less than 0.5 (*m/s*), although a magnitude of speed error reflects the road grade (also compare Figure 4.3.3(b) with Figure 4.3.4(b)).

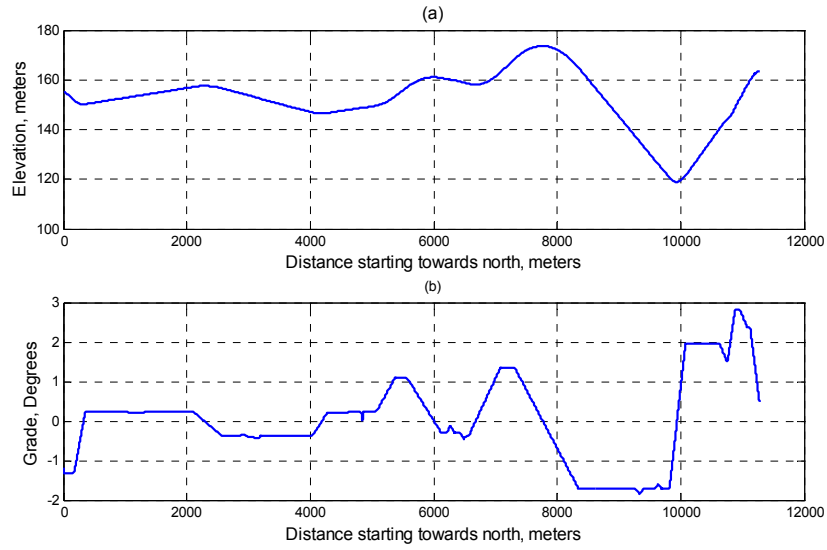


Figure 4.3.3 Road Elevation and Grade on I-15 at San Diego

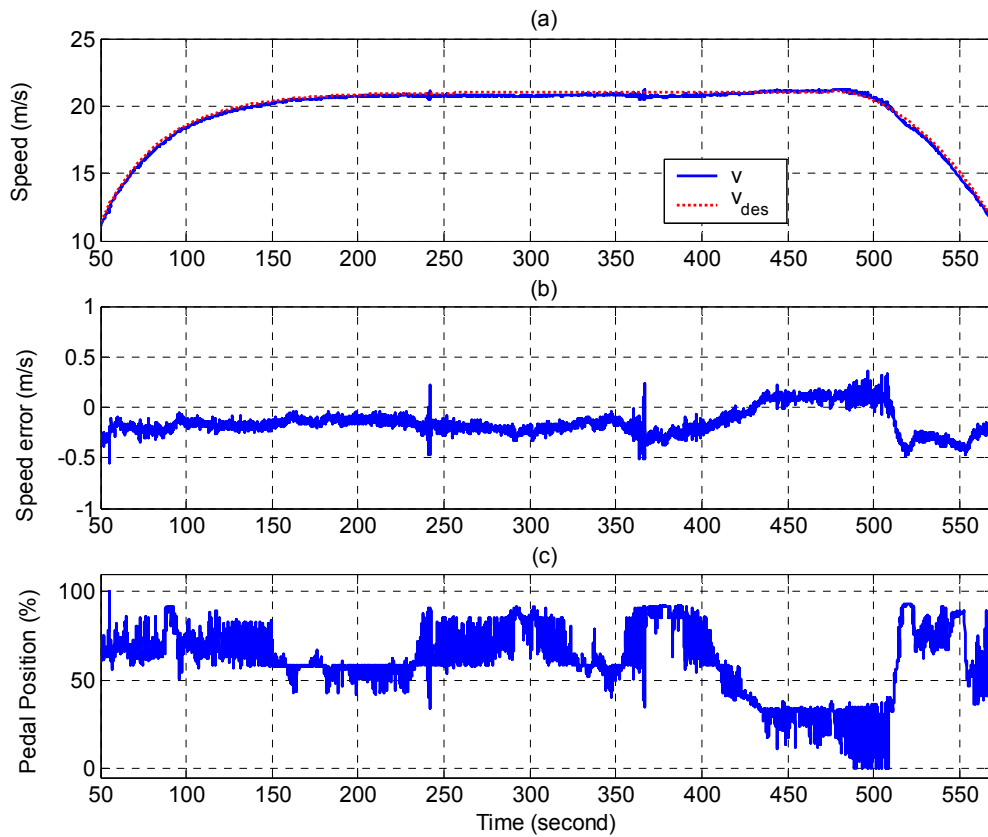


Figure 4.3.4 Speed Tracking on I-15 at San Diego: a 40-ft Transit Bus, Driving Northbound

4.3.2 Vehicle-Following in Platoon, with Both Constant and Varying Spacings

① Distance controller

The distance following control law can be derived similarly by extending the definition of S_1 . That is, if there are two vehicles, i.e., the leading and following vehicle, the first sliding surface can be defined as

$$S_1 = \dot{\varepsilon}_1 + q_1 \varepsilon_1$$

where $\varepsilon_1 = R_{des} - R_1$, R_{des} is the desired spacing, and R_1 is the distance between the lead and following vehicle. As derived in equation (4.3.1) and (4.3.2), both engine and brake control laws can be obtained similarly. Furthermore, it is interesting to remark that both R_1 and \dot{R}_1 are obtained through the range and range rate measurement sensors. The range and range rate data are obtained from a millimeter-wave radar, a laser radar (lidar), and a wireless communication system. The detailed discussion of the sensor processing and fusion algorithm can be found in [8].

If there are more than two vehicles to consider, the first sliding surface can be extended for guaranteeing the string stability [11] as follows: for $i = 2, \dots$,

$$S_1 = \dot{\varepsilon}_i + q_1 \cdot \varepsilon_i + q_2 (\dot{x}_{i+1} - \dot{x}_{lead}) + q_3 \left(x_{i+1} - x_{lead} - \sum_{j=1}^i L_j \right)$$

where $\varepsilon_i = R_{des} - R_i$, R_i is the range of $(i+1)$ th vehicle, x_i is the position of i th vehicle, and x_{lead} is the position of the first vehicle, and L_j is the length of j th vehicle.

② Experimental validation and performance

As done above for the experimental validation of the proposed speed controller, high speed tests for distance following were conducted using two transit buses at Crows Landing in California. Figure 4.3.5 shows the desired velocity ranging from 0 to 22 (m/s) with the maximum acceleration, 0.25 (m/s^2). When the controller gains are assigned as $\{\lambda_{1e}, \lambda_{2e}, \tau_{2e}, q_1\} = \{1.2, 25, 0.02, 0.7\}$ and $\{\lambda_{1b}, \lambda_{2b}, \tau_{2b}, q_1\} = \{1.0, 20, 0.02, 0.7\}$ for the longitudinal control of a 40-ft transit bus, the distance tracking performance is shown in the bottom plot of Figure 4.3.5. Once the torque converter is locked (after 29 second in Figure 4.3.5), the tracking error remains within 1.5 (m) when the desired constant distance is 15 (m). Similarly, Figure 4.3.6 shows the distance tracking performance for a 60-ft articulated bus. Its tracking error of the distance following controller is less than 1 (m) under a flat road condition.

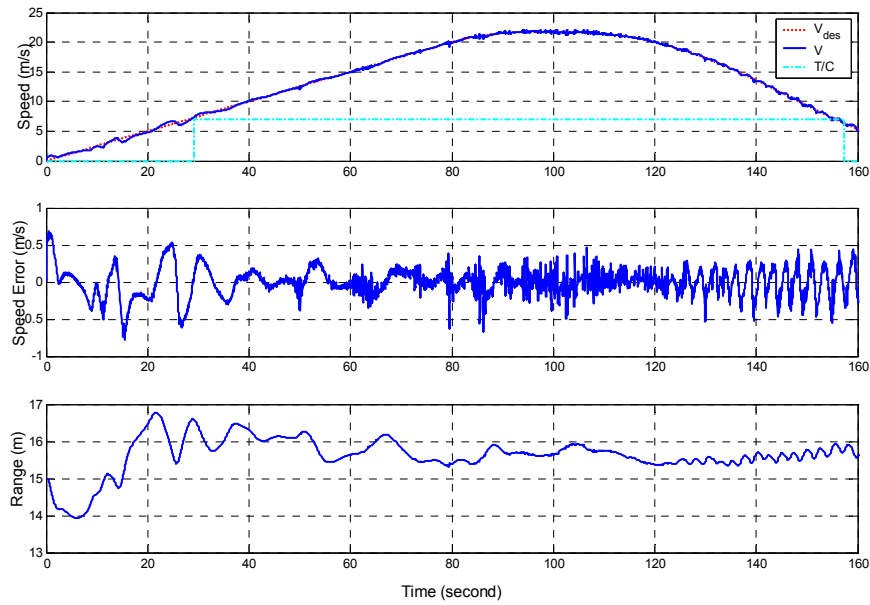


Figure 4.3.5 Time Responses of Speed, Speed Error, and Range at Crows Landing: a 40-ft Transit Bus

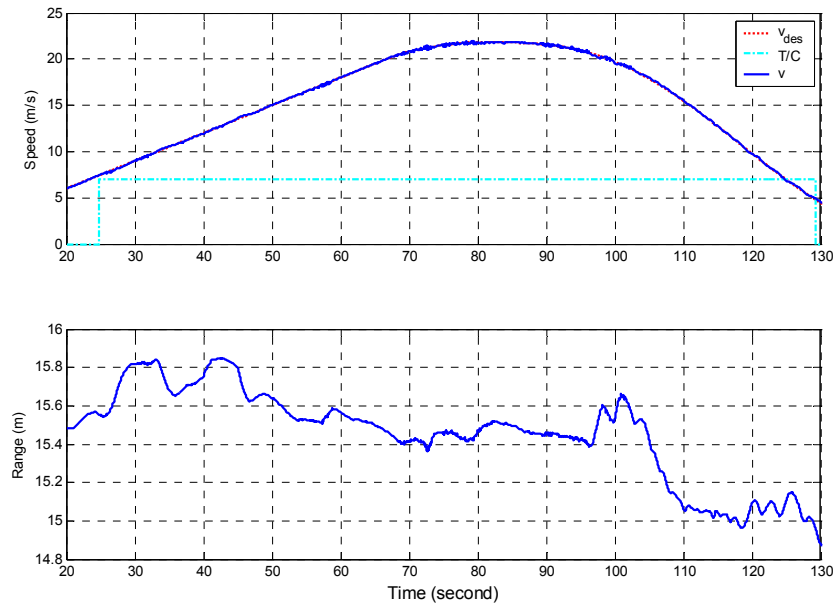


Figure 4.3.6 Time Responses of Speed and Range at Crows Landing: a 60-ft Articulated Bus

While a virtual leading vehicle was used above to validate the distance following control law, the two-vehicle following scenario on I-15 was considered next. That is, the 40-foot CNG bus was driven automatically as a leading vehicle, and the automated 60-foot

articulated bus followed with a given desired spacing. Figure 4.3.7(a) shows speed responses of the two transit buses as well as a given desired speed profile with respect to time. More specifically speaking, the operation of both vehicles was switched to automatic control initially about the speed of 13 (m/s) by a driver through DVI (after about 40 second in the figure), and the following distance was closing from the current distance to 15 (m), which is set to be a desired distance. This is why there is a discrepancy in the term of speed between 40 and 90 second in Figure 4.3.7(a). For the given scenario, Figure 4.3.7 (b) and (c) show range and distance tracking error between two automated transit buses. It is noted that there is a fairly steep hill during the period from 90 to 220 seconds (also refer to Figure 4.3.3), and it results in larger distance tracking error up to 2 (m). Except this period, the distance tracking error remains within 0.5 (m) during the constant distance following. It is suggested that it may be good to have a road grade estimation algorithm and the corresponding hardware and/or software (GPS, map database) to improve the distance following performance in the presence of a steep hill on a highway.

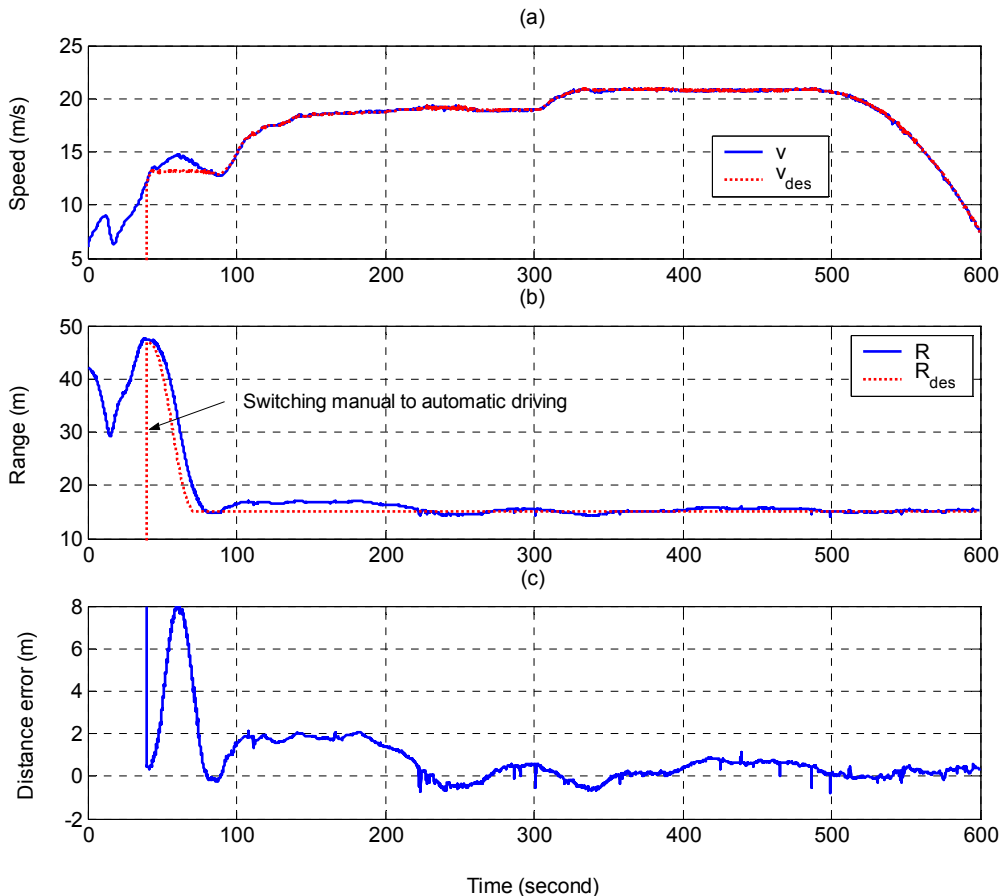


Figure 4.3.7 Switching Manual to Automatic Driving and Distance Tracking: Driving Southbound on I-15

Next, the closing and opening gap scenario was added, i.e., the following distance was closing from 40 (m) to 20 (m) after about 210 second and opening to 40 (m) after about

360 second. Figure 4.3.8(b) shows that the velocity tracking error of the lead vehicle stays less than 0.5 (m/s), while the distance tracking error of the following vehicle is within 2 (m) as shown in Figure 4.3.7(c).

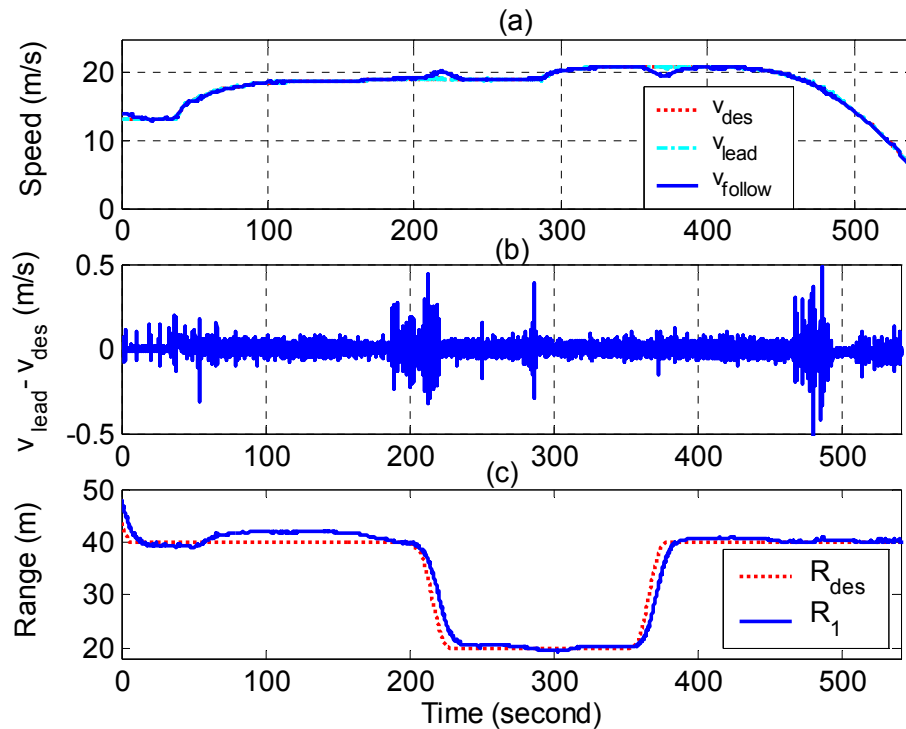


Figure 4.3.8 Closing and Opening Gap Maneuver: Driving Southbound on I-15

Finally, a platoon maneuver of three automated transit buses was conducted by placing another 40-ft CNG transit bus as a third vehicle. After about 45 seconds, all three vehicles were switched to automatic driving (by driver use of the DVI) as shown in Figure 4.3.9. Then, they were closing a gap to 40 (m) and maintained the desired distance. Figure 4.3.9(b) shows that distance error does not propagate backward. That is, even when the distance error of the second bus reaches up to 2 (m), the error of the third bus still remains within 2 (m). This validated that the overall system with the distance following controller is *string stable* [2]. Furthermore, it shows experimentally the potential feasibility of a platoon maneuver of more than three transit buses.

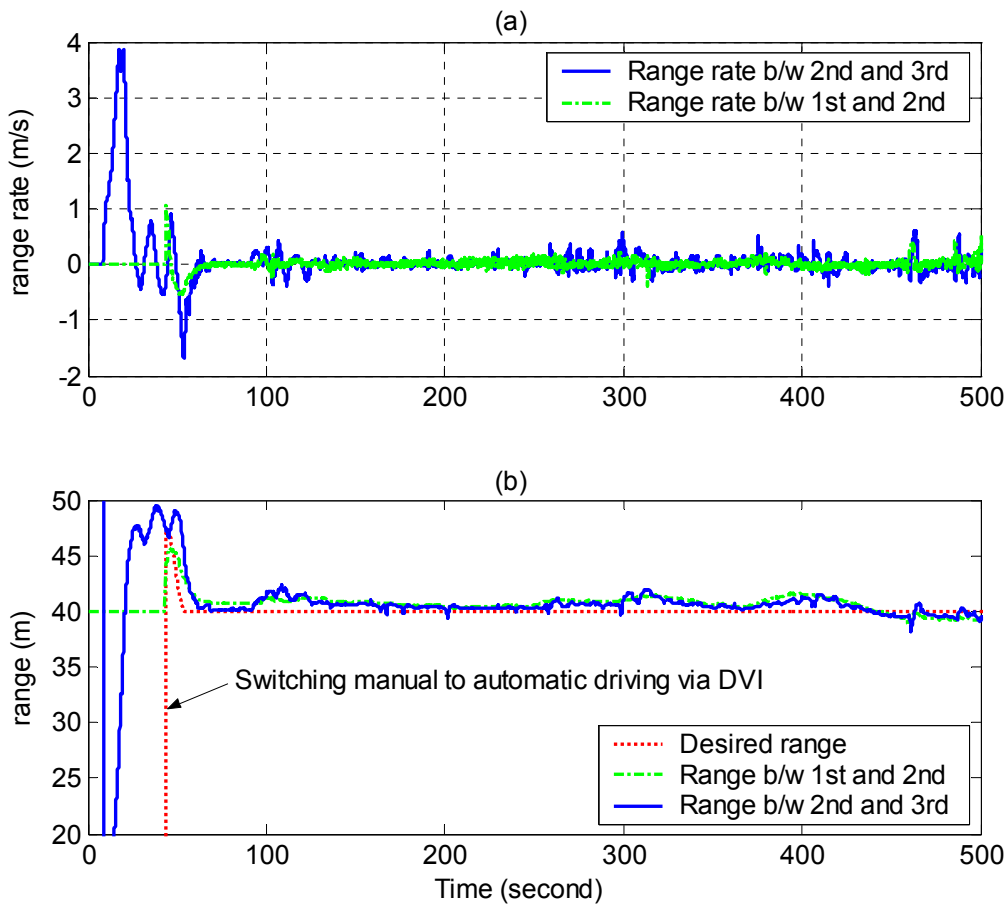


Figure 4.3.9 Platoon Maneuver of Three Automated Transit Buses on I-15: Driving Southbound

4.4 Truck Longitudinal Control

The primary focus of the automation development work on the trucks has been for longitudinal control, because this is what provides the largest potential benefits in terms of increasing capacity of a truck lane and reducing fuel consumptions and emissions. The development sequence began with causing each truck to follow a commanded speed profile, addressing the lower-level vehicle dynamics issues, and then proceeded to vehicle-following of trucks within a platoon, which involves the integration of forward sensing and communication systems.

4.4.1 Tracking a speed profile

The commanded speed profiles for testing combinations of tractor and empty trailer and with fully loaded trailer at the Crows Landing track are depicted in Fig. 4.4.1 below. These profiles are governed by the desire be able to test up to 55 mph within the limited length of that track (approximately 2.4 km) and the limited power of the truck engines. Smooth acceleration commands are generated within these constraints to obtain the illustrated profiles. The fully loaded truck is particularly limited in its acceleration capability at the higher speeds (engine torque declines above 1800 rpm and drag increases), so its profile differs noticeably from that of the empty truck.

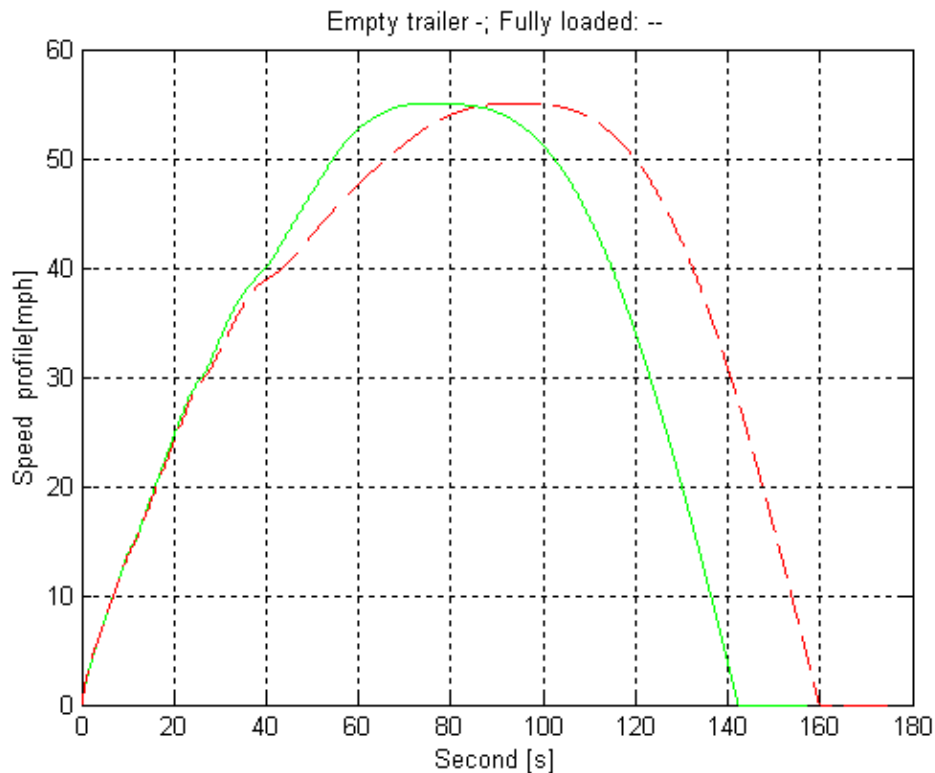


Fig. 4.4.1 Speed Command Profiles for Testing Empty and Full Trucks at Crows Landing

Test Conditions for Single-Vehicle Speed Tracking

- Vehicle combined weight tested
 - Empty flat trailer ($M=14061[kg]$)
 - Half loaded ($M=22226[kg]$)
 - Fully loaded ($M=31795[kg]$)
- Speed range tested: $10 \sim 55[mpg]$
- Flat test track with total length $\sim 2100[m]$
- Combined braking system tested
 - Air brake (EBS) + Jake brake + Transmission retarder
 - Air brake (EBS) + Jake brake
 - Air brake (EBS) + Transmission retarder
 - Air (EBS) brake only

Hardware:

Cummins *N14+ 435[hp]* Turbo diesel engine

With Jake (compression) brake capability

400[N.m] \sim 2 cylinder

800[N.m] \sim 4 cylinder

1200[N.m] \sim 6 cylinder

Transmission retarder

Euro-EBS/ABS

Most data reading and commanding are through J1939 Bus

Acceleration and deceleration test:

Vehicle speed to follow a predefined profile (following a virtual vehicle)

Maximum acceleration tested for different loads

$a=0.55 [m/(s^2)]$ when $v=2.0 [m/s]$

$a=0.24 [m/(s^2)]$ when $v=14.0 [m/s]$

$a=0.06 [m/(s^2)]$ when $v=25.0 [m/s]$

Maximum deceleration range tested

$0.4 \sim 1.0 [m/(s^2)]$

Notations in figures

Each run has 3 figures.

Units & terminologies used in the following figures:

spd: Speed [mph]

dist: distance
 dist_err: distance error [m]
 spd_err: speed error [m/s]
 max_acc: Maximum acceleration [m/s²]
 max_dec: Maximum deceleration [m/s²]
 acceleration [m/s²]
 deceleration [m/s²]
 EBS: Euro-EBS/ABS
 Jake: Jake (compression) brake
 trtd: Transmission retarder

A comprehensive collection of test data from these experiments can be seen in Appendix B. A few representative examples are described here to show the key performance issues.

The first case, shown in the three figures following, is for the empty trailer loading, with a maximum speed of 55 mph, maximum commanded deceleration rate of 1.2 m/s² and all three braking systems in action. Figure 4.4.2 shows the commanded and actual speed profiles to be virtually indistinguishable from each other. The transmission gear shifts and distance and speed tracking errors are shown on the other two traces of this figure. They show distance errors remaining within about +/- 1 m and very small speed errors except for the very start and conclusion of the maneuver.

Empty Flat Trailer, Run 1: max_spd=55[mph], max_dec=1.2[m/s²], EBS + Jake + trtd

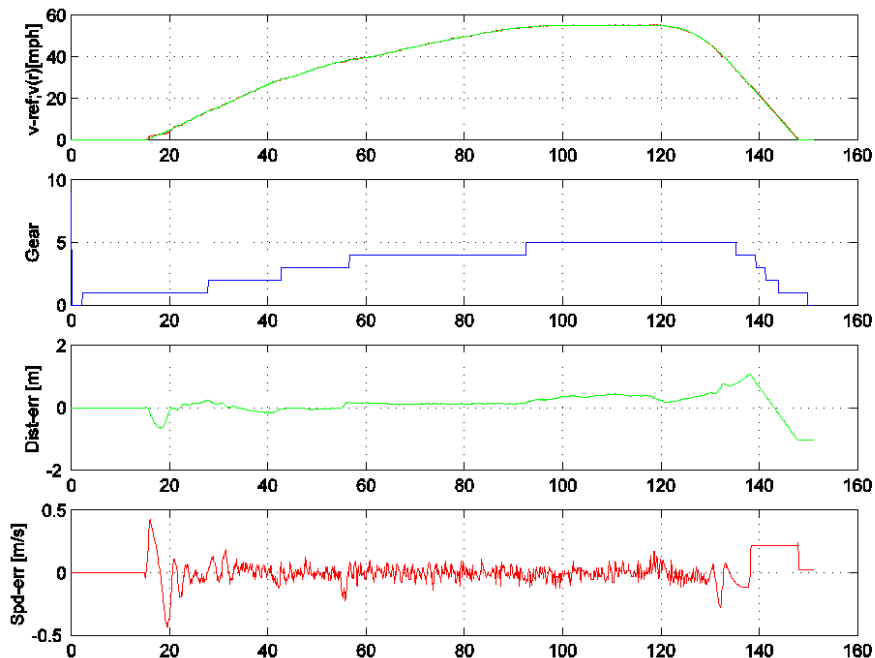


Fig. 4.4.2 – Speed Profile Tracking by Single Empty Truck (1 of 3)

Empty Flat Trailer, Run 1: $\text{max_spd}=55[\text{mph}]$, $\text{max_dec}=1.2[\text{m/s}^2]$, EBS + Jake + trtd

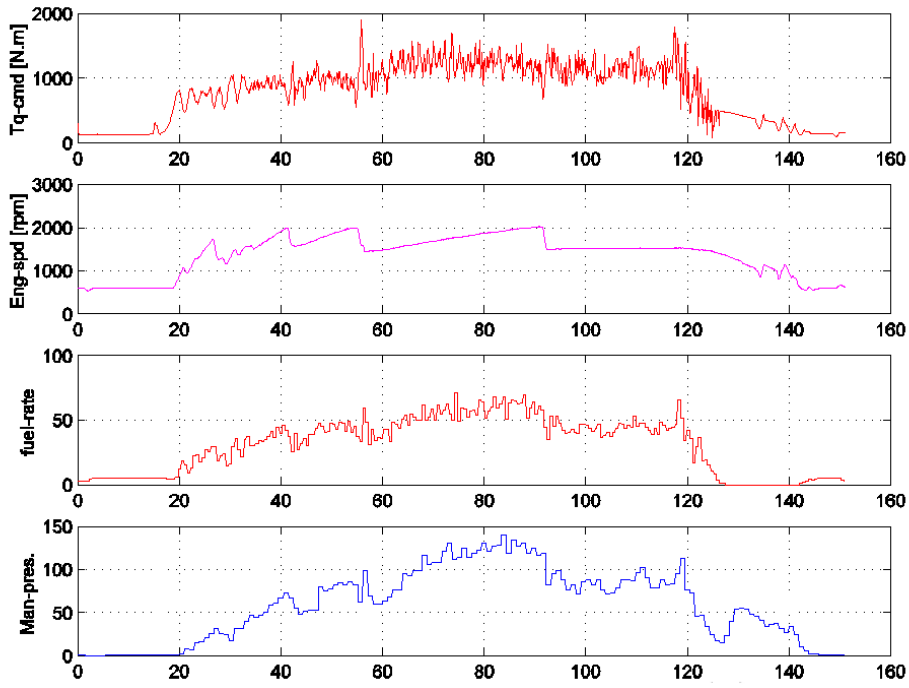


Fig. 4.4.3 – Speed Profile Tracking by Single Empty Truck (2 of 3)

Empty Flat Trailer, Run 1: $\text{max_spd}=55[\text{mph}]$, $\text{max_dec}=1.2[\text{m/s}^2]$, EBS + Jake + trtd

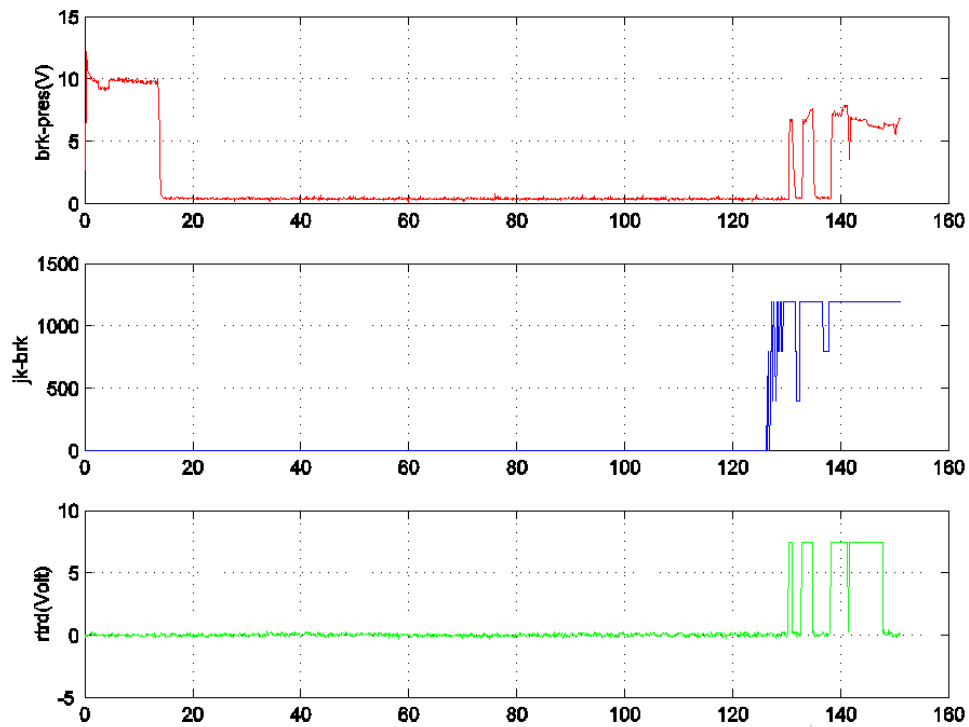


Fig. 4.4.4 – Speed Profile Tracking by Single Empty Truck (3 of 3)

The remaining two figures provide more of the internal performance measures needed to identify how the longitudinal controller is working, such as the engine torque command profile, engine speed, fueling rate and manifold pressure, followed by the operation of the three braking systems. The transmission shifting effects can be seen clearly in the jumps in the engine speed profile. The complementary actions of the three braking systems can be seen in the final plot, with the compression brake acting noticeably earlier than the other two braking systems.

The fully loaded truck has more than twice the mass of the empty truck, so it is important to verify how robust the control system is with respect to mass variations. The results for a similar test up to 55 mph of the fully loaded truck are shown in the following Figures 4.4.5 – 4.4.7. In this particular run, the retarder was not used and the maximum commanded deceleration rate was 1 m/s^2 . Note that the distance error is still very small until well into the braking maneuver, when it oscillates around approximately 1 m of error, and this is accompanied by some oscillations of the speed error as well (but those are contained within less than 0.5 m/s). These oscillations can be traced to the dynamics of the EBS and compression brakes, which can be seen to switch on and off periodically in Figure 4.4.7.

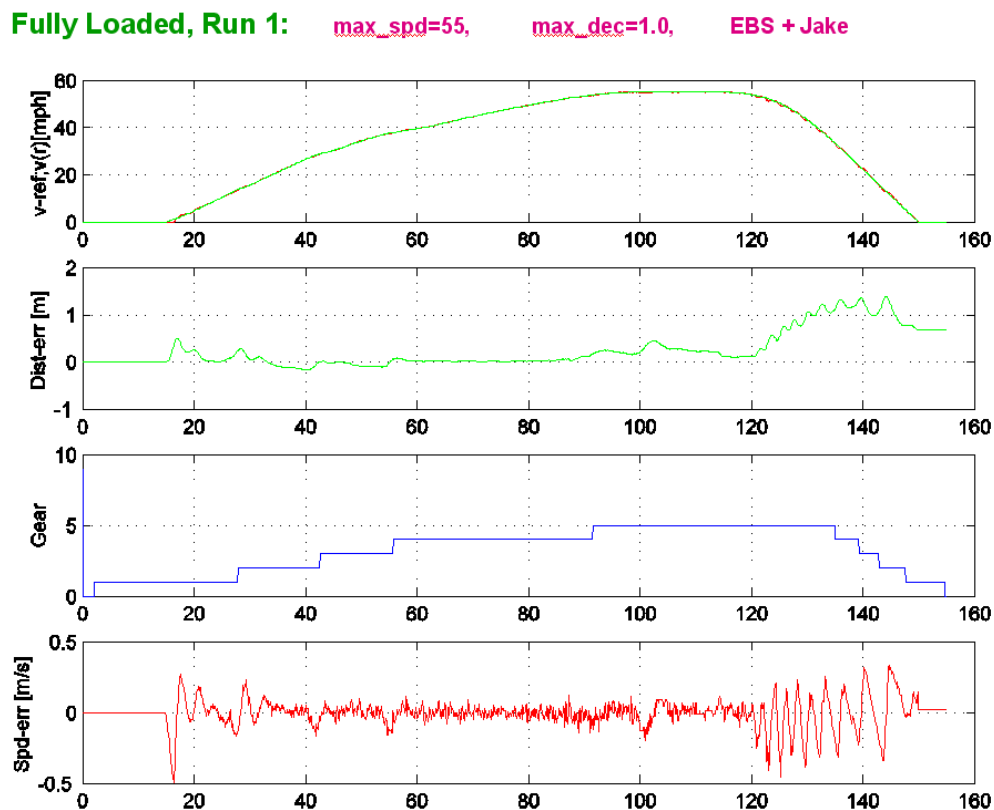


Fig. 4.4.5 – Speed Profile Tracking by Single Fully Loaded Truck (1 of 3)

Fully Loaded, Run 1: max_spd=55, max_dec=1.0, EBS + Jake

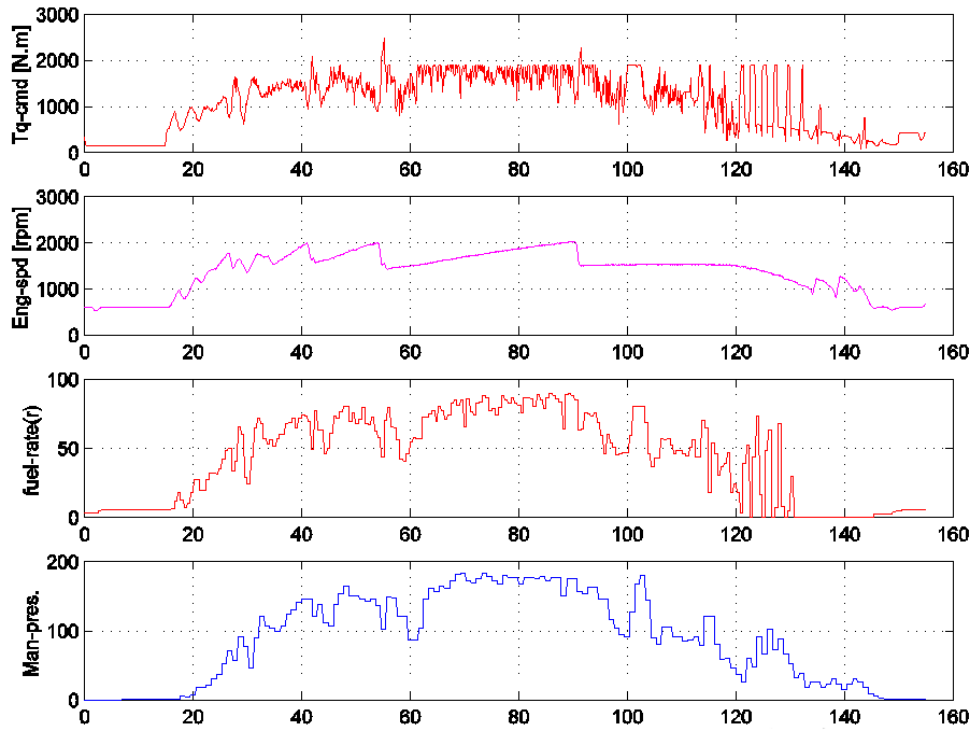


Fig. 4.4.6 – Speed Profile Tracking by Single Fully Loaded Truck (2 of 3)

Fully Loaded, Run 1: max_spd=55, max_dec=1.0, EBS + Jake

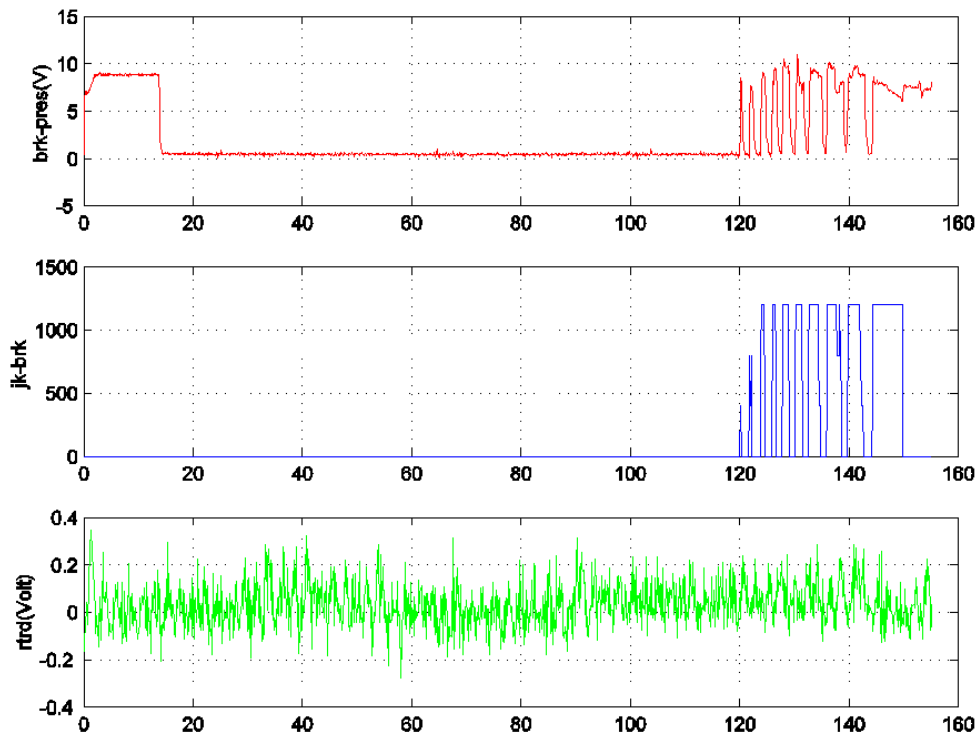


Fig. 4.4.7 – Speed Profile Tracking by Single Fully Loaded Truck (3 of 3)

A fortuitous error in the experimental procedures made it possible to assess the robustness of the control system to unknown loading deviations, when some of the tests were run for the fully-loaded truck when the controller parameters were defined based on the assumption that the truck was empty. Results of one of these tests are shown in Figures 4.4.8 – 4.4.10 below.

In these runs, only the EBS braking system was active, which limited both the level and speed of the braking action that were available. The distance errors were still confined within the range below 1 m, and the speed errors less than 0.5 m/s, but this test case was for a commanded deceleration rate of only 0.5 m/s, which is lower than in the previously reported test cases. The oscillations in action of the EBS can be seen clearly, since this test was performed before the EBS controller was modified.

Fully loaded, Run 6: max_spd=55, max_dec=0.5, EBS only
Empty trailer Mass is used in feedback control

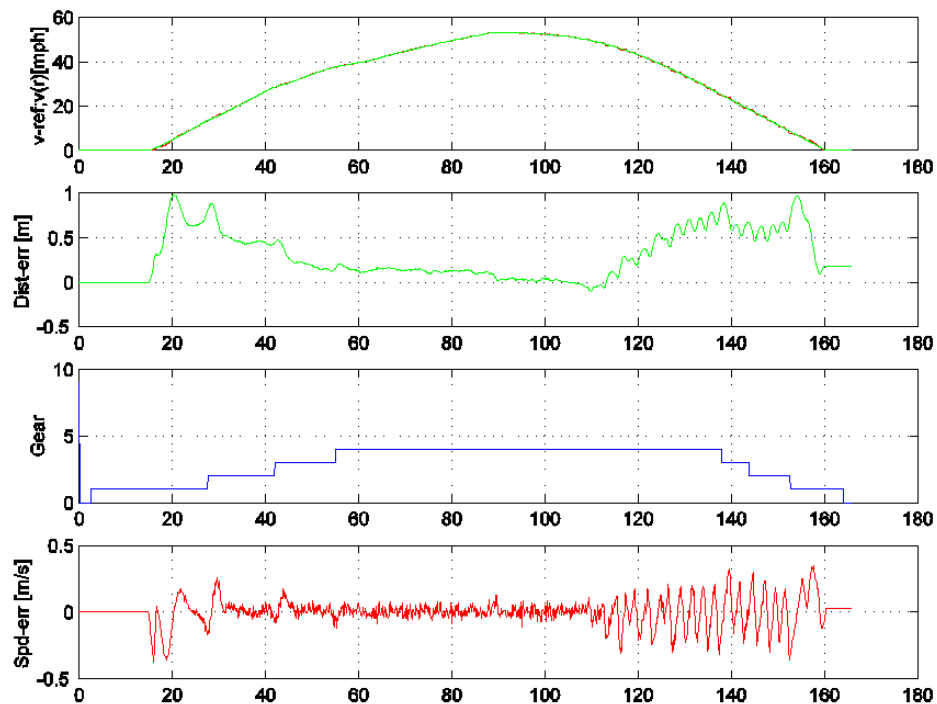


Fig. 4.4.8 – Speed Profile Tracking by Single Fully-Loaded Truck Using Controller Parameters for Empty Truck (1 of 3)

Fully Loaded, Run 6: $\text{max_spd}=55$, $\text{max_dec}=0.5$, **EBS only**
 Empty trailer Mass is used in feedback control

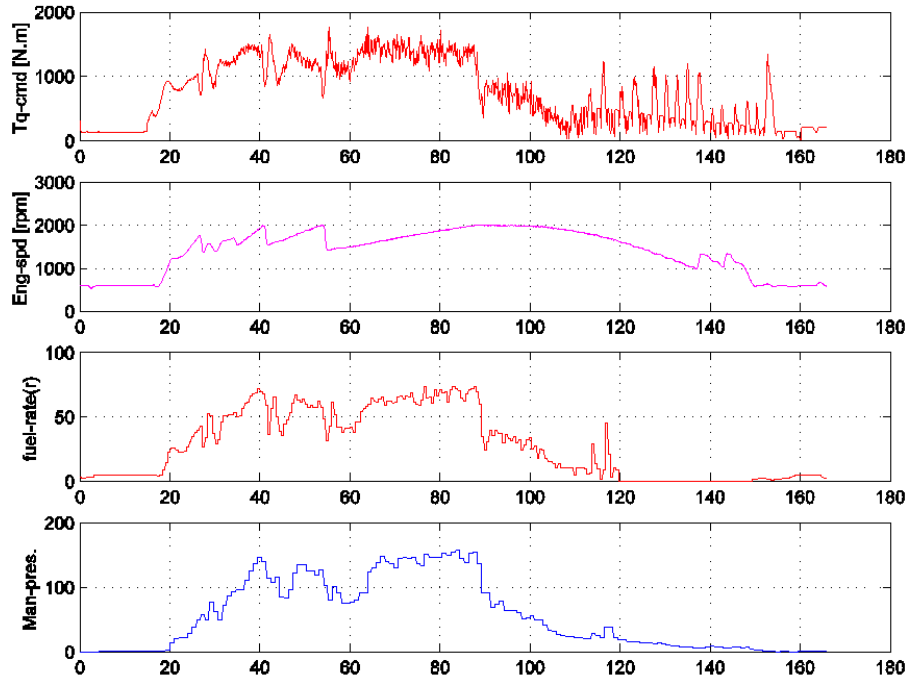


Fig. 4.4.9 – Speed Profile Tracking by Single Fully-Loaded Truck Using Controller Parameters for Empty Truck (2 of 3)

Fully Loaded, Run 6: $\text{max_spd}=55$, $\text{max_dec}=0.5$, **EBS + Jake + trtd**
 Empty trailer Mass is used in feedback control

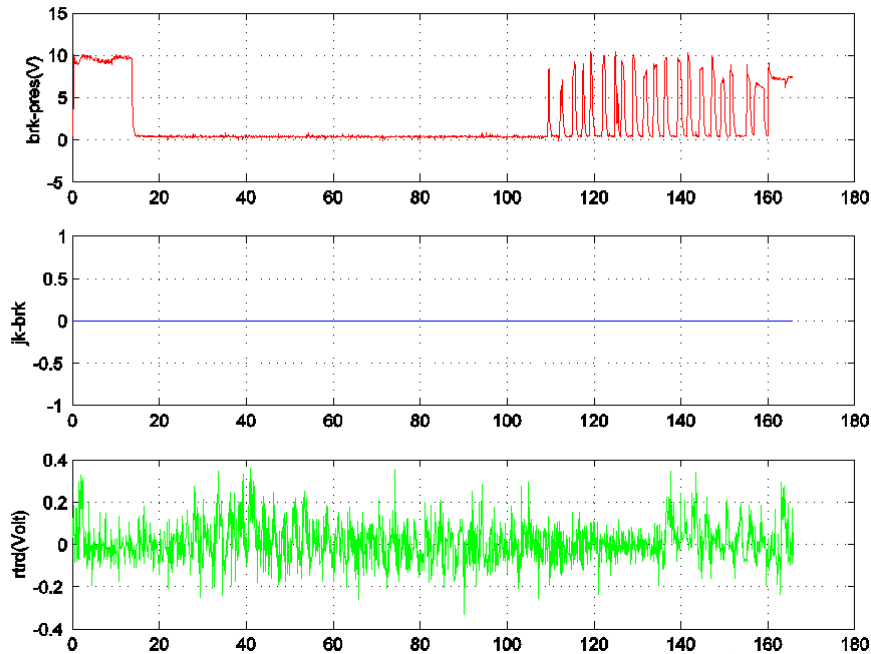


Fig. 4.4.10 – Speed Profile Tracking by Single Fully-Loaded Truck Using Controller Parameters for Empty Truck (3 of 3)

4.4.2 Vehicle-Following in Platoon

Control design for vehicle following to ensure string stability is reported in [9]. Preliminary experimental work for heavy-duty-truck following has been reported by the CHAUFFEUR project in [10]. However, the trucks they used are not as heavy as those used by PATH. Besides, the following distance they used was between 8~10[m]. With our developed longitudinal controller, we have tested following distance for different truck loads from empty trailer to fully loaded with following distances between 3~10[m]. Qualitative results about fuel consumption are also indicated.

Test Scenarios:

- Vehicle following
 - 1st Truck: Fully loaded ($M=31795[kg]$)
 - 2nd Truck: Half loaded ($M=22226[kg]$)
- Speed range tested: 45 ~ 55[mph]
- Inter-vehicle distance: 4~10[m]
- Flat test track with total length ~ 2250[m]
- Combined braking system tested
 - Air brake (EBS) + Jake brake + Transmission retarder
 - 2nd has modified EBS Box with 0 initial value
 - 1st has default initial value for deceleration of $0.25[m/s^2]$

Maneuvers Tested:

1st vehicle speed to follow a predefined profile (following a virtual vehicle)

2nd vehicle to follow the 1st to keep constant inter-vehicle distance

Maximum acceleration tested

$a=0.55 [m/(s^2)]$ when $v=2.0 [m/s]$

$a=0.24 [m/(s^2)]$ when $v=14.0 [m/s]$

$a=0.06 [m/(s^2)]$ when $v=25.0 [m/s]$

Maximum deceleration range tested

$0.9 [m/(s^2)]$

Each run has 3 figures.

Units & notations used in the following figures are:

spd: Speed [mph]

dist: distance
dist_err: distance error [m]
spd_err: speed error [m/s]

Color used in plotting:

red - 1st vehicle
green - 2nd vehicle

The x-coordinate is time in second.

A substantial collection of test data from these runs is shown in Appendix C. A representative example is described here to show the key performance characteristics for one of the most demanding conditions, with the maximum speed of 55 mph and the trucks operating at a separation of only 4 m. The summary of top-level performance attributes in Figure 4.4.11 shows the virtually indistinguishable speed profiles of the two trucks, and their speed errors in the second-row plot confirm how small and very similar these errors are. The transmission shifts are seen to occur nearly simultaneously for the two trucks in the next plot, and their distance errors are similarly small, with the exception of some growth in the distance error of the following truck during the final stage of the braking maneuver.

Run 6: Max speed 55 [mph] ; Des_dist: 4 [m]

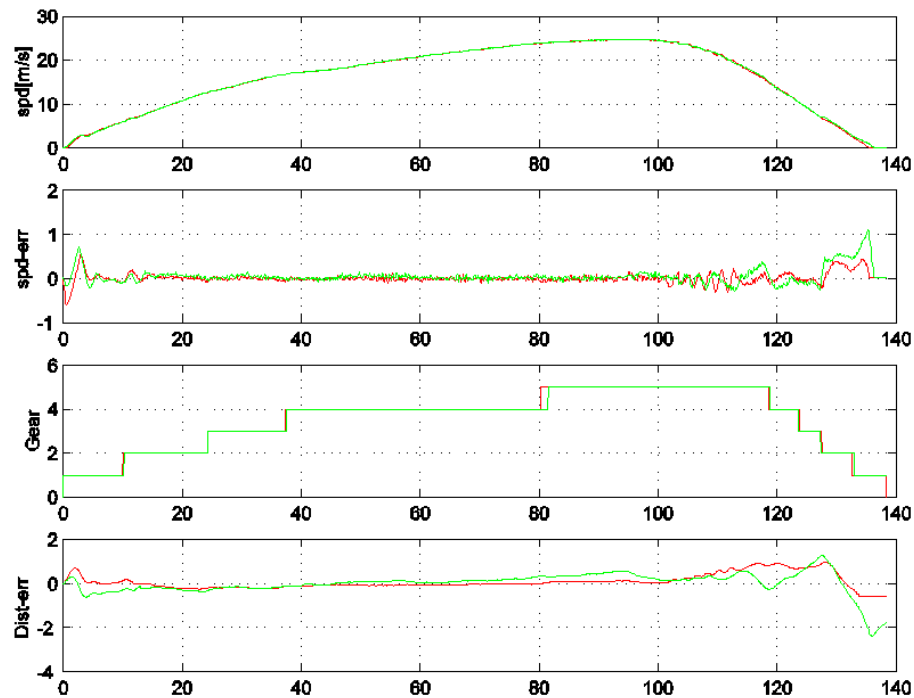


Fig. 4.4.11 – Two-Truck Platoon Operating at 4 m Separation (1 of 3)

Run 6: Max speed 55 [mph] ; Des_dist: 4 [m]

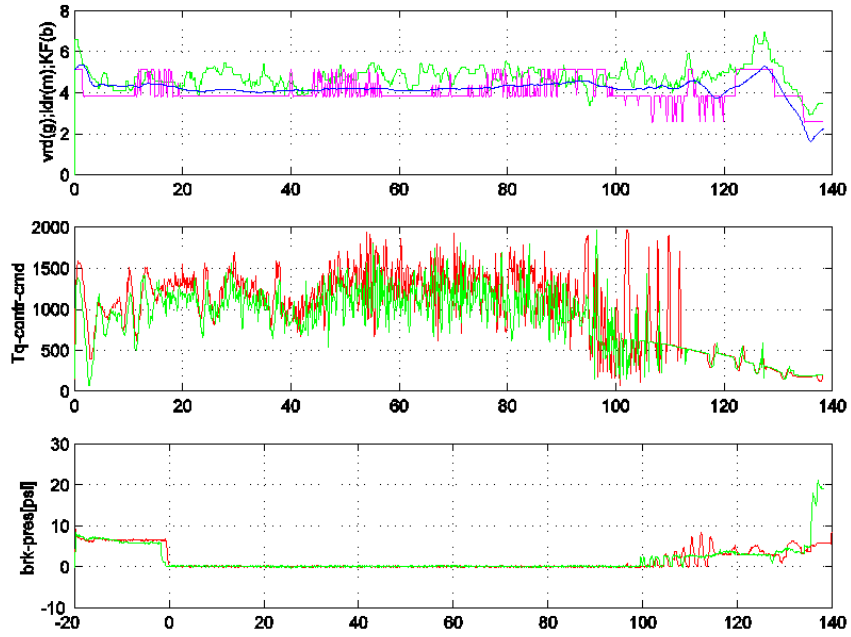


Fig. 4.4.12 – Two-Truck Platoon Operating at 4 m Separation (2 of 3)

Run 6: Max speed 55 [mph] ; Des_dist: 4 [m]

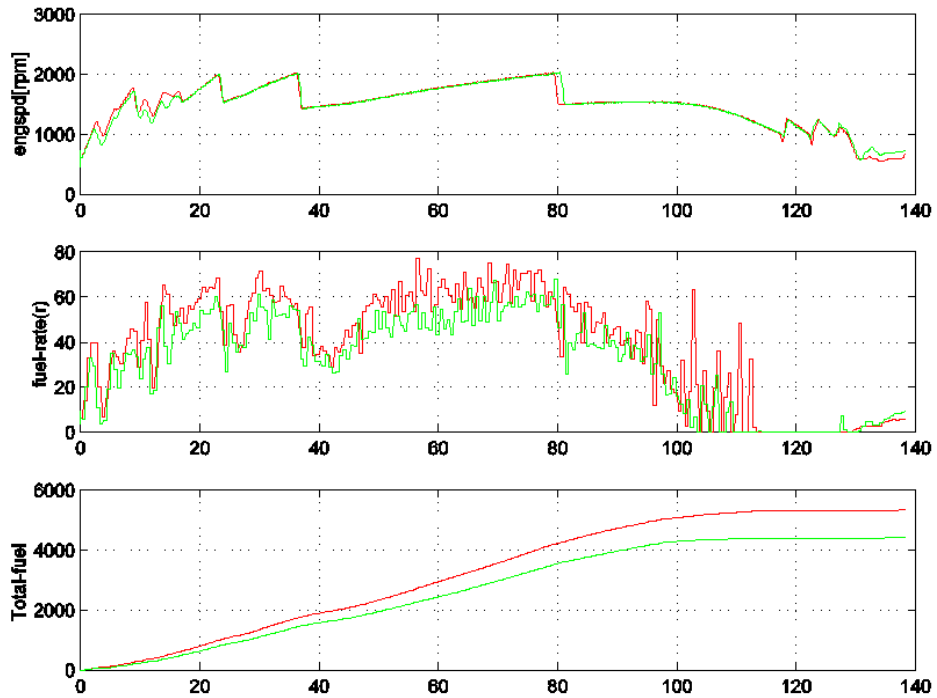


Fig. 4.4.13 – Two-Truck Platoon Operating at 4 m Separation (3 of 3)

Figure 4.4.12 shows the ranging sensor data used for controlling the separation between the trucks, with the relatively noisy traces from the individual radar and lidar sensors being fused into a much smoother combined range estimate. The torque command and brake pressure measurements from the two trucks are seen to be fairly similar to each other, and where there are differences, the following truck shows less oscillatory measurements than the leading truck, which is an important stability consideration.

Finally, Figure 4.4.13 shows the engine speed, fueling rate and total fuel consumption of both trucks through the test maneuver. The similarities in engine speed and in the general shape of the fueling rate plot are quite evident. The overall fuel consumption is clearly higher for the first truck than for the second, but this truck was also more heavily loaded in this test run, so conclusions about fuel economy effects cannot be drawn from this particular test case.

Later experiments were conducted under more carefully controlled conditions, and with identically loaded trucks, in order to compare fuel consumption of the first and second trucks in the platoon, as well as comparing with an individual truck following the same speed profile.

4.4.3 Testing for Changes in Fuel Consumption and Emissions

After the two-truck platoon longitudinal control was working satisfactorily, it was used for two series of controlled experiments at Crows Landing to make direct measurements of the potential savings in fuel consumption and emissions, based on the aerodynamic drag reductions from close-formation platoon driving. These tests were performed in cooperation with Prof. Frederick Browand from USC (for drag and fuel consumption) and Prof. Matthew Barth from UCR (for emissions).

The Crows Landing tests had to be performed under several challenging practical constraints, which limited the opportunities for obtaining the most comprehensive and authoritative data. The first was the limited length of the runway there (7500 feet), which made it impossible to accelerate the trucks to speeds above 55 mph, and only provided limited cruising time at that speed before it was necessary to decelerate. The second important constraint was the lack of lateral control on the trucks, because no magnetic reference markers were available on the longest runway at Crows Landing. This meant that the drivers had to work hard to steer their trucks accurately to follow a line along the edge of the runway, but they were still subject to lateral deviations of as much as 30 cm, reducing the effectiveness of the aerodynamic “drafting”.

The first series of tests, in October 2003, included use of the UCR Modal Emissions Research Laboratory (MERL) trailer, to provide direct measurements of the emissions of one truck tractor at a time. Tests were performed for an individual truck following a commanded speed profile, and then with the MERL trailer attached to the leading and following trucks of two-truck platoons following the same speed profile, to provide data for comparable driving conditions. The MERL trailer includes a heavy-duty diesel

engine to generate power for its onboard instrumentation, which requires that the back end of the trailer remain open during the tests for ventilation. The weight of this trailer limited the maximum speed that could be reached within the available runway length, and the open back end changed the aerodynamics of this trailer compared to “standard” box trailers of similar dimension, introducing some distortion into the test results.

The detailed results of the emissions testing are being reported by Prof. Barth under the MOU that funded his participation in these tests. The top-level summary of these results for NOx emissions is shown in Table 4.4.1 and for CO2 in Table 4.4.2. The results were not consistent for intermediate spacing values where they were better for the rear truck and worse for the front truck, so it is difficult to draw meaningful conclusions on the basis of these tests, beyond noting that the savings are quite small, but somewhat more significant for the front truck than for the rear truck. The CO2 reductions, which should be directly related to fuel consumption savings, are larger, and clearly more significant for the rear truck than for the front truck. However, these results did not follow smooth trends at the intermediate spacings.

Table 4.4.1 – Savings in NOx emissions from truck operations in close-formation platoons

Spacing	Front Truck	Rear Truck
10 m	5.6 %	1.4 %
4 m	4.4 %	1.1%

Table 4.4.2 – Savings in CO2 emissions from truck operations in close-formation platoons

Spacing	Front Truck	Rear Truck
10 m	8.1 %	15.5%
4 m	11.3 %	17.7%

The tests for fuel consumption were repeated in December 2003, without use of the MERL trailer, but with two identical leased trailers on the two trucks, and under somewhat more carefully controlled test conditions, based on the lessons learned from the October testing. Prof. Browand compared the results with his prior results from wind-tunnel testing of scale model trucks.

The Crows Landing tests involved direct comparisons of the real-time measurements of fuel being consumed by the engines of both trucks in tandem, compared with the same trucks driving individually. In order to minimize variability in the experimental conditions, data were only analyzed for the periods when the trucks were cruising at constant speed, not during their acceleration or deceleration maneuvers, and data were only recorded when the ambient winds were modest. Multiple runs were performed for

each test condition, and the runs were balanced between the two directions of travel on the runway in order to compensate for ambient winds and for a slight grade along the runway (about 0.4%). The results are summarized in Figure 4.4.14 below.

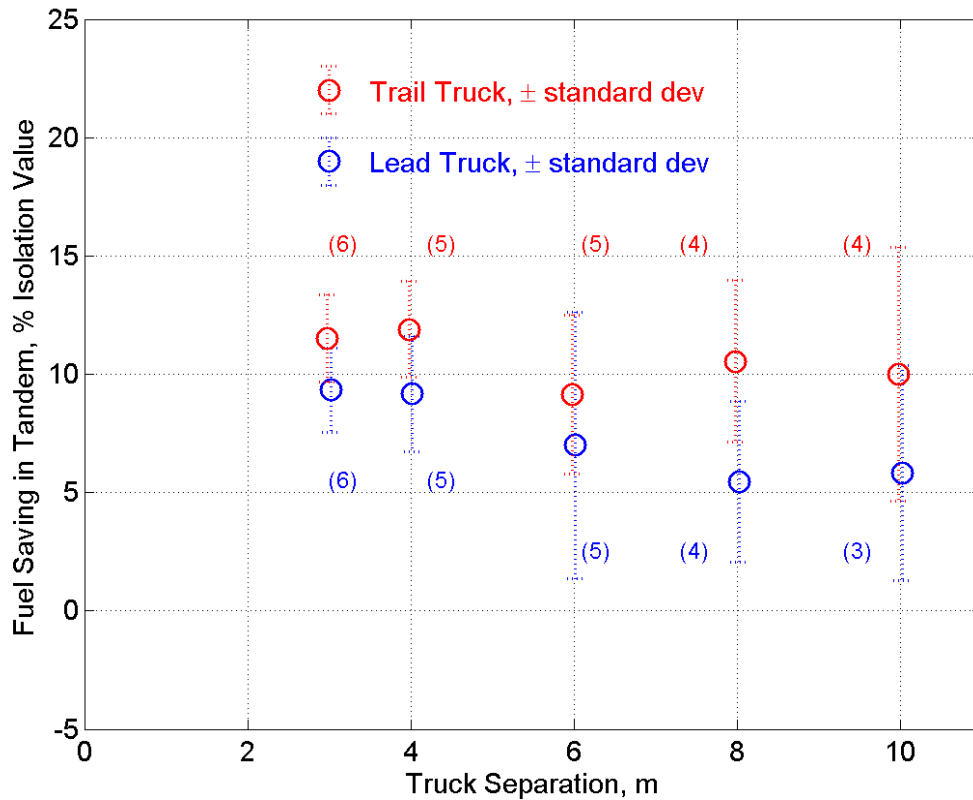


Figure 4.4.14 Comparison of Fuel Consumption of Trucks Operating in Tandem in Platoon, Versus Operating in Isolation

As the uncertainty bands on this figure show, there is still considerable variability in the data, but it is clear that the trailing truck saves somewhat more than the leading truck. However, the trend with respect to truck separation is not as strong as was expected based on the earlier wind tunnel results. The data are plotted together in Figure 4.4.15 to illustrate this contrast. In particular, the trailing truck does not save as much as expected at the shorter gaps. The most likely explanation for this discrepancy is that the wind-tunnel model had a blunt front end comparable to a cab-over-engine tractor design, while the trucks used for the field tests had an engine-forward design, which meant that even when the separation between the front of the second truck and the rear of the leading truck's trailer was small, the separation to the large cross section of the second truck's trailer was considerably larger. Thus, with the typical engine-forward tractor design, it is not physically possible to get down to a really small gap between the successive trailers to enable the substantial drag savings. The cab-over design is likely to be much more amenable to fuel savings from close-formation platoon driving.

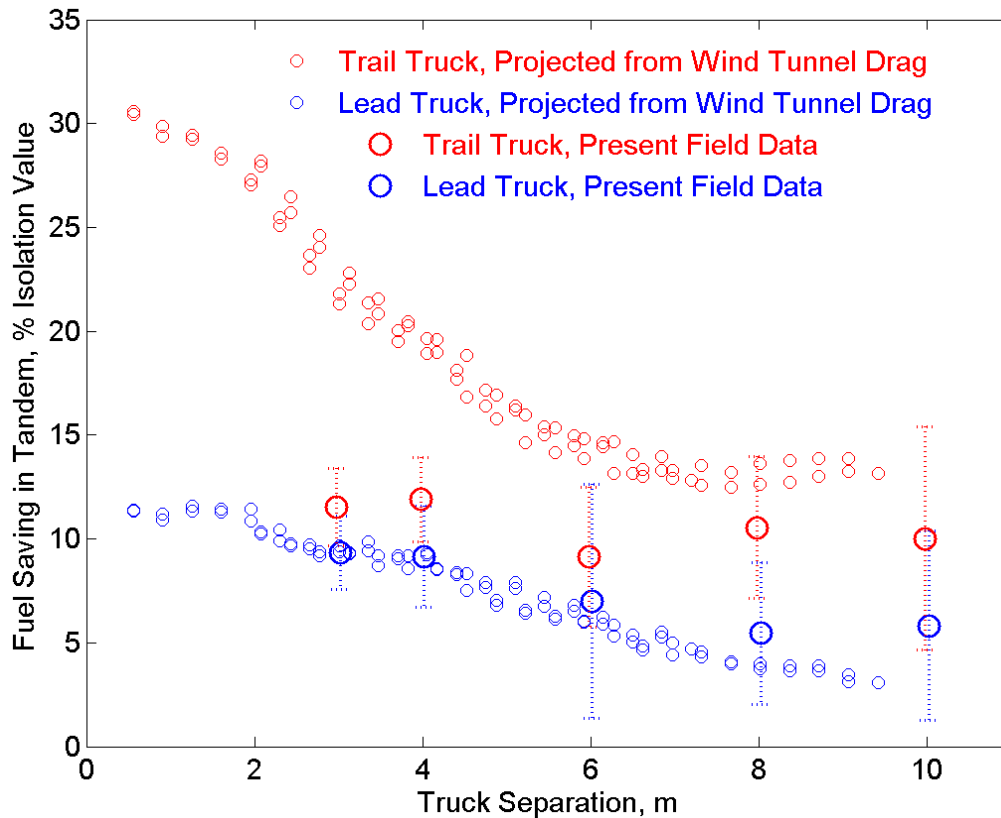


Figure 4.4.15 Comparison of Direct Field Test Measurements with Scale-Model Wind Tunnel Estimates of Fuel Savings from Close-Following Platoon Operations

4.5 Truck Lateral Control at Highway Speed

Limited resources were available for designing and testing the lateral controller for trucks when the hardware components and software drivers were ready, since the development focuses had shifted to the transit buses at that time. However, the development team did use the only trip for initial lateral dynamics and sensor testing at Crows Landing to test out some interesting truck lateral control hypotheses.

As pointed out by several papers, conventional wisdom suggests that steering a tractor semi-trailer is more difficult than steering a passenger vehicle; the challenges in controlling heavy vehicle lateral dynamics result from:

- Large inertia, which causes slower response to steering.
- Inherent non-linearity in the vehicle model such as longitudinal velocity and tire lateral force saturation.
- Unusually large uncertainties in vehicle parameters and environmental disturbances.

However, many truck drivers will argue that the tractor semi-trailer is generally easier to steer (in the stability sense) under normal highway driving conditions. The major challenges in steering control of such vehicles arise when:

- Controlling tractor semi-trailer under severe braking scenario (jack-knifing).

- Backing up or parallel parking tractor semi-trailer into a restricted location. Both challenges result from the inherent poor stability conditions of these two vehicle configurations, similar to buckling or inverse pendulum.

We then set up to substantiate the above engineering intuition by comparing the model of tractor semi-trailers with that of the passenger car under forward driving condition. The comparison reveals that an appropriately designed robust steering controller for passenger car could be a good candidate for controlling the tractor semi-trailer. A “look-ahead” steering controller [12] that specifically aims at desensitizing vehicle dynamics by projecting forward is one such candidate. The resultant steering performance during normal forward driving conditions is superior to other controllers that have been tested to date in the same tractor semi-trailer.

4.5.1 Vehicle and Truck Model Comparison

Figure 4.5.1 shows the comparison of transfer functions from steering angle to lateral acceleration at tractor CG for tractor semi-trailer, tractor only, and passenger car. According to Figure 4.5.1, passenger vehicle may not be easier to steer than tractor semi-trailers (as long as the tractor semi-trailer is properly loaded and maintained). Figure 4.5.2 shows the lateral acceleration transfer functions at 1-tractor or 1-car length ahead of vehicle for the corresponding tractor semi-trailer, tractor only, and passenger car. One can observe that a merely 1-tractor/car length looking-ahead appears to be enough to create similar steering dynamics for all the above vehicle configurations. This observation helps explaining the fact that many truck drivers describe that, except for the technique of properly positioning the trail end of the trailer during turning, steering a tractor semi-trailer is generally more stable than a passenger car under relatively constant speed forward driving conditions.

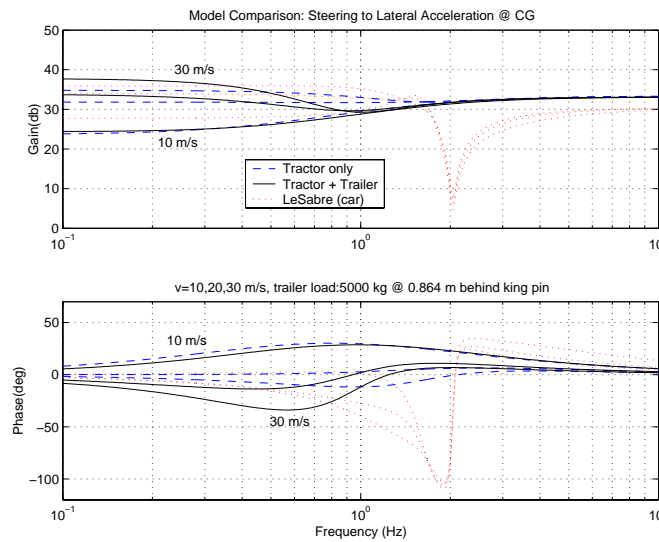


Figure 4.5.1: Transfer function comparison from steering angle to lateral acceleration at tractor or car CG for tractor semi-trailer, tractor only, and passenger car

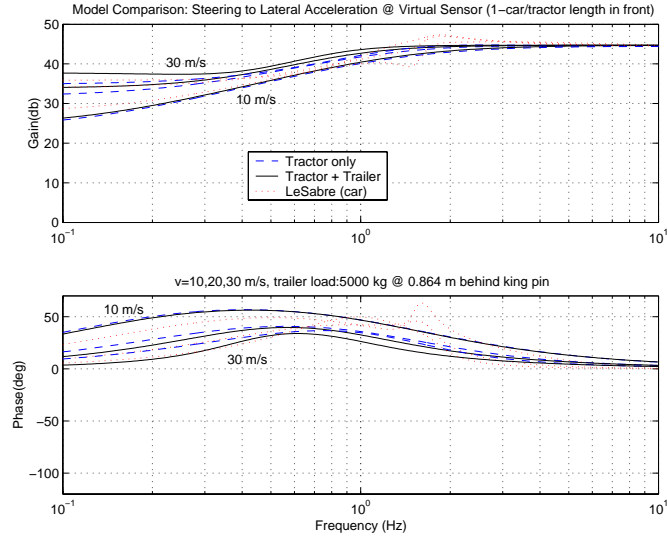


Figure 4.5.2: Transfer function comparison from steering angle to lateral acceleration at 1-tractor/car length ahead of vehicle for tractor semi-trailer, tractor only, passenger car

4.5.2 Controller Design

Based on the discussions in Section 4.5.1 and 2.2.2.2, one would suspect that a well designed “look-ahead” steering controller for a passenger car may be a good candidate for the steering control of both configurations: tractor only and tractor semi-trailer. This engineering intuition was investigated and verified in [14]. A simple linear “look-ahead” controller that was designed for the Buick LeSabre lane-keeping demonstration [12] was implemented on the tractor semi-trailer as well as on the tractor only configurations. The lateral dynamics of these three configurations were generally considered to be so far apart that they warrant different steering controllers. The linear lane-keeping controller for the steering command δ_c has a very similar structure as the one for the transit bus lane-keeping controller (as in Eq. 4.1.1.5):

$$\delta_c = -k_c G_c ((k_{int} + k_e k_{ext}) y_f - k_e k_{ext} y_r) \quad (4.5.1)$$

where k_{int} is the integrator at front sensor location, k_{ext} the virtual sensor extension filter, G_c the compensator at the virtual sensor location, k_e and k_c constants that can be tuned, y_f and y_r the lateral measurements at front and rear sensors respectively. The three filters and their primary functions are described as follows: (a) An integral control, $k_{int}(s)$, that keeps the steady state tracking error at the front sensor to zero. (b) A frequency shaped look-ahead distance, $G_{ds}(s)$, that provides more look-ahead distance around the vehicle lateral modes and roll-off of the look-ahead distance at higher frequencies. (c) A servo controller, $G_c(s)$, that uses the frequency shaped virtual displacement as input and compensates it for the actuator and noises.

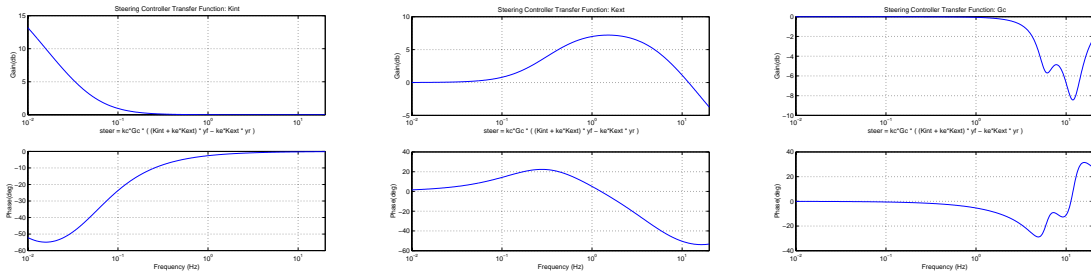


Figure 4.5.3: Filter frequency responses (Left: $k_{int}(s)$, Center: $G_{ds}(s)$, Right: $G_c(s)$)

4.5.3: Truck Experimental Results at Crows Landing

Without modifying any coefficients in the LeSabre controller, the identical controller with identical controller parameters was implemented on the Freightliner Century tractor, and an older Freightliner FLD 120 tractor, with different trailer combinations, and tested at the Crows Landing test track (shown in Figure 4.5.4). Table 4.5.1 lists all the possible tractor and trailer combinations for the tests. The tractor was manually sped up from rest to about 73 mph. Hard braking was applied in order to stop the vehicle before the end of the track. The following are the equivalent curve transitions based on the longitudinal position along the track: tangent to +800 m radius transition at 740 m; -800 m radius transition at 974 m; +800 m radius transition at 1442 m; and transition to tangent at 1676 m.

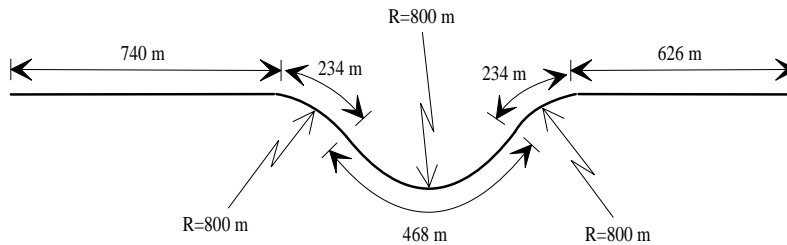


Figure 4.5.4: Crows Landing Test Track

Test Configuration	Tractor		Trailer			Controller	
	Weight (kg)	Wheel base (m)	Type	Weight (kg)	Load (kg)	Type	Parameters
FLD120 Tractor w/o trailer	7727	5.35				Look-ahead (LSB)	Same as LeSabre
FLD120 Tractor w trailer	7727	5.35	Box	5455	5000	Look-ahead (LSB)	Same as LeSabre
2001 Century Tractor w/o trailer	8400	5.82				Look-ahead (LSB)	Same as LeSabre
2001 Century Tractor w empty trailer	8400	5.82	Lowboy	6465		Look-ahead (LSB)	Same as LeSabre
2001 Century Tractor w loaded trailer	8400	5.82	Lowboy	6465	8200	Look-ahead (LSB)	Same as LeSabre
LeSabre Passenger car	1740	2.81				Look-ahead (LSB)	Based on LeSabre

Table 4.5.1: Tractor and Semi-trailer Combination Controller Test Configurations

Figures 4.5.5 and 4.5.6 show the results of a typical closed-loop test using the LeSabre “look-ahead” linear controller (with constant coefficients) for the Freightliner FLD120 Class 8 tractor with the Great Dane semi-trailer (with 5000 kg load at trailer center) as well as for the tractor only configuration. Similarly, Figures 4.5.7 shows the results (lateral displacement, steering angle, and speed) for the same tests for the following three configurations: the 2001 Freightliner Century tractor only, Century tractor with empty lowboy trailer, and Century tractor with a lowboy trailer loaded with an 8200 kg tractor. Figure 4.5.8 plots the blowup lateral displacements and steering angles starting from the longitudinal position at 980 m to 1600 m. Figure 4.5.9 compares the tracking error for all tractor semi-trailer (top) and passenger car (bottom) configurations in Table 4.5.1.

The following are several interesting observations:

- The tracking errors exhibit extremely similar characteristics, regardless of which tractor and semi trailer combinations, or trailer loading conditions, or with or without trailers were used for testing, at the same longitudinal locations along the test track. It appears that a robust “rail-like” characteristic was obtained based on this simple control algorithm.
- The tracking error is generally less than 5 cm at the equilibrium conditions (on the straight and on the curves); and the maximum transition error is less than 25 cm at speeds over 70 mph with 400 m radius transitions. This is the best performance to date

that has been achieved on this particular tractor semi-trailer. And it was achieved without any parameter tuning process.

- The steering angle characteristics are also similar with different tractors and trailer combinations, or with and without trailer, which reinforces the analysis that steering dynamics are similar under a proper “look-ahead” controller. Furthermore, the steering angle for the tractor semi-trailer is more biased toward the right (positive angle) than for the tractor only configuration. This is explained by the fact that the test track is tilted toward the left (positive error) and thus a tractor with trailer will need to steer more to the right to compensate for the additional lateral force of dragging the trailer along.
- The passenger vehicle exhibits very similar steering and tracking characteristics to those from the tractor or tractor trailer combinations except that its tracking errors and steering angles are smaller in magnitude. The higher bandwidth (smaller mass with faster actuator) of the passenger vehicle generates faster lateral response and thus results in smaller tracking error.

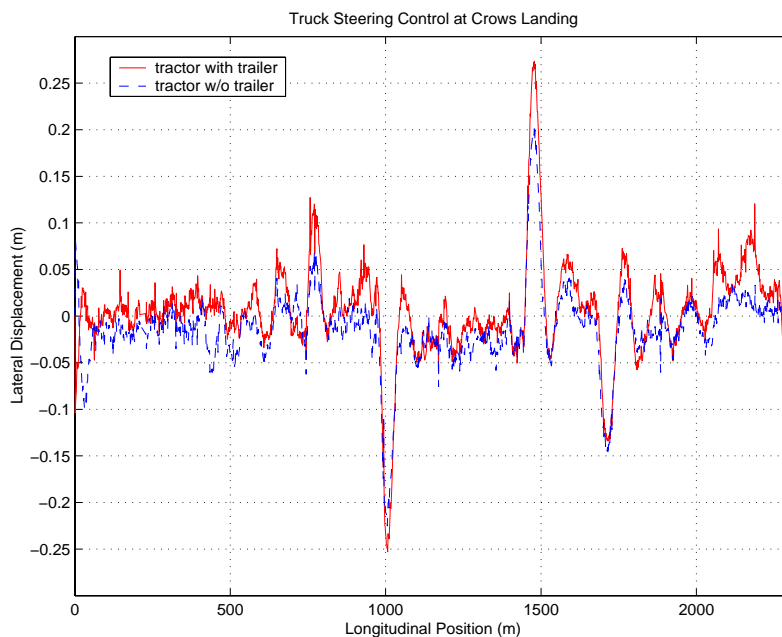


Figure 4.5.5: Tracking error for tractor semi-trailer (with 5000 kg load) and FDL120 tractor only (with respect to longitudinal position along the track)

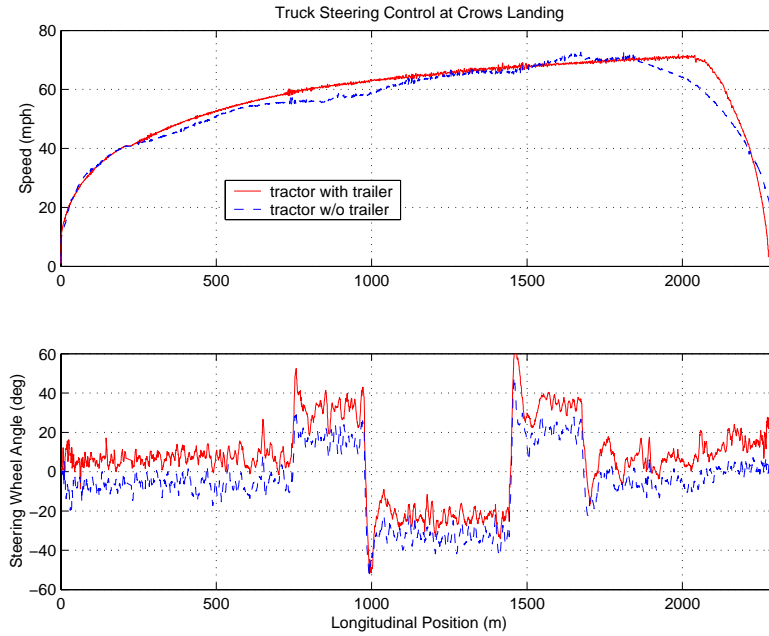


Figure 4.5.6: Speed and steering angle for tractor semi-trailer (with 5000 kg load) and FDL120 tractor only configuration (along longitudinal position at the track)



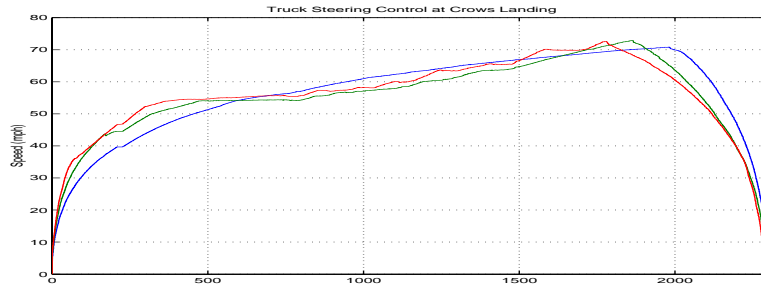


Figure 4.5.7: Tracking errors, steering angles, and speeds for Century tractor with and without empty or loaded lowboy semi-trailer (with 8200 kg load) configurations (along longitudinal position at the track)

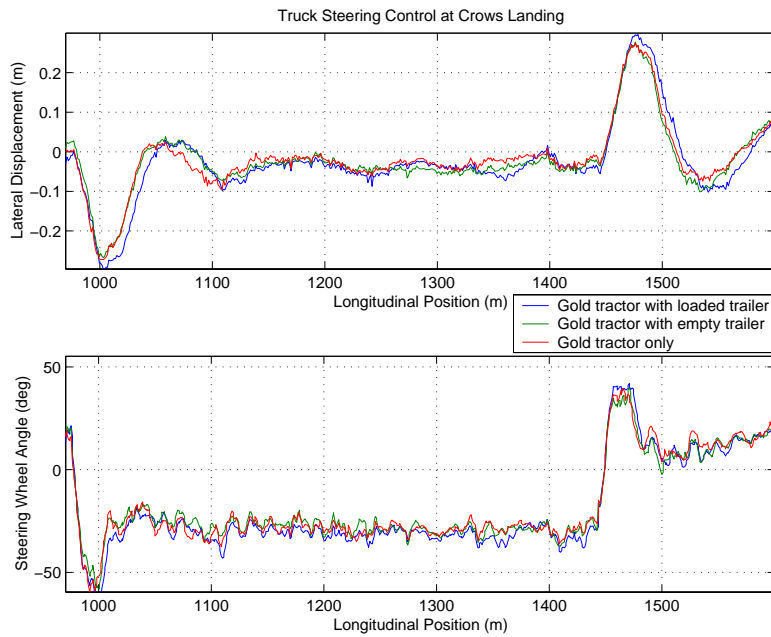


Figure 4.5.8: Tracking errors and steering angles for Century tractor with and without empty or loaded lowboy semi-trailer (with 8200 kg load) configurations (along longitudinal position from 980m to 1600 m at the track)

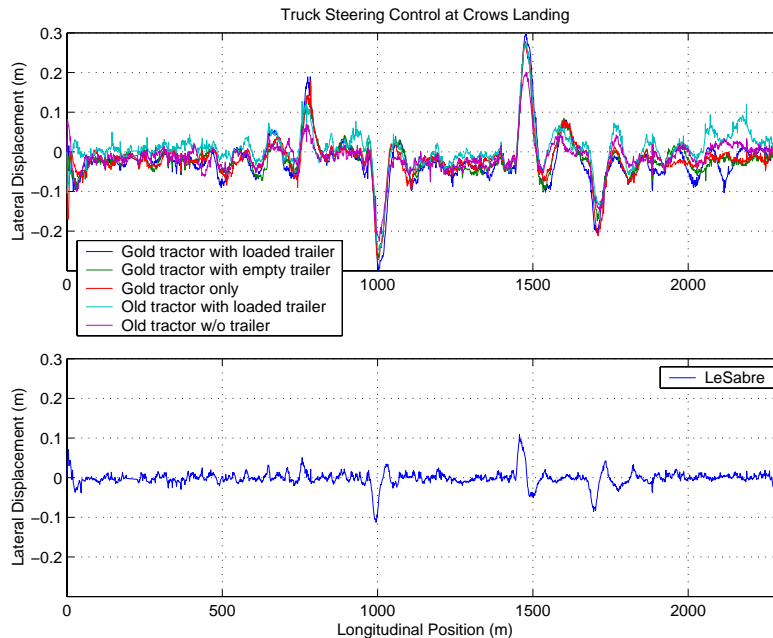


Figure 4.5.9: Tracking error for all tractor semi-trailer (top) and passenger car (bottom) configurations in Table 4.5.1

References for Chapter 4

- [1] R. Rajamani, H. Tan, B.K. Law, and W. Zhang, Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons. *IEEE Trans. Control Systems Technology*, vol. 8, no. 4, 2000, pp 695-708.
- [2] J.K. Hedrick and P.P. Yip, Multiple sliding surface control: theory and application, *Journal of Dynamic System, Measurement, and Control*, vol. 122, December, 2000, pp 586-593.
- [3] D. Swaroop, J. K. Hedrick, P. P. Yip, and J. C. Gerdes, "Dynamic surface control for a class of nonlinear systems," *IEEE Trans. Automatic Control*, vol. 45, no. 10, 2000, pp. 1893-1899.
- [4] B. Song, J. K. Hedrick, and A. Howell, "Robust Stabilization and Ultimate Boundedness of Dynamic Surface Control Systems via Convex Optimization," *International Journal of Control*, vol. 75, no. 12, 2002, pp. 870-881.
- [5] M. Vidyasagar, 1985, "Control system synthesis: a factorization approach," Cambridge, Mass., MIT Press.
- [6] B. Song, J.K. Hedrick, and A. Howell, Fault tolerant control and classification for longitudinal vehicle control, *Journal of Dynamic Systems, Measurement, and Control*, vol. 125, September, 2003, pp 320-329.
- [7] J.C. Gerdes, Decoupled design of robust controllers for nonlinear systems: as motivated by and applied to coordinated throttle and brake control for automated highways, *Ph. D. thesis*, U.C. Berkeley, 1996.

- [8] A. Howell, B. Song, and J.K. Hedrick, Cooperative range estimation and sensor diagnostics for vehicle control, *Proceedings of ASME: Dynamic Systems and Control Division*, To appear in Nov. 2003.
- [9] X. Y. Lu, S. Shladover and J. K. Hedrick, 2004, Heavy-Duty Truck Control: Short Inter-vehicle Distance Following, accepted to American Control Conference, (ACC-04), Boston, MA. [XYL9]
- [10] Fritz, H., Longitudinal and lateral control of heavy-duty trucks for automated vehicle following in mixed traffic: experimental results from the CHAUFFEUR project, *Proc. of IEEE Int. Conf. on Contr. Applications*, Kohala Coast-Island of Hawai'i, Aug. 22-17, 1999. [FRI]
- [11] H.-S. Tan, B. Bougler and W.-B. Zhang, "Automatic Steering Based on Roadway Markers - From Highway Driving to Precision Docking," *Vehicle System Dynamics*, vol. 37, no. 5, March, 2002, pp. 315-339.
- [12] H.-S. Tan, J. Guldner, S. Patwardhan, C. Chen and B. Bougler, "Development of an Automated Steering Vehicle Based on Roadway Magnets - A Case Study of Mechatronic System Design," *IEEE/ASME Transactions on Mechatronics*, vol. 4, no. 3, Sept., 1999, pp. 258-272.
- [13] H.-S. Tan, J. Guldner, C. Chen, S. Patwardhan and B. Bougler, "Lane Changing with Look-down Reference Systems on Automated Highways," *IFAC Journal, Control Engineering Practice*, vol. 8, issue 9, Sept., 2000, pp. 1033-1043.
- [14] H.-S. Tan, B. Bougler, and P. Hingwe, "Automatic Steering Control of Tractor Semi-Trailer Using Controller Designed for Passenger Vehicles," in *Proceedings of the 2nd IFAC Conference on Mechatronic Systems*, Berkeley, CA, USA, Dec. 2002, pp. 63-68.

Appendix A

In-Vehicle PC/104 Computer Systems for Trucks and Transit Buses

I. HARDWARE OVERVIEW

Currently, UC PATH has adopted the PC/104 computer standard for control of heavy vehicles which includes three Freightliner trucks and three New Flyer transit buses. The PC/104 was chosen since it is rapidly becoming a standard for computers often found in factories, laboratories, and machines to provide programmable control of complex systems. PC/104 is a standard for PC-compatible modules (circuit boards) that can be stacked together to create a complete computer system. These types of systems are often found in factories, laboratories, and machinery to provide programmable control of a complex system. PC/104 systems are very similar to standard desktop PCs but with a different form factor. The name "PC/104" is derived from this likeness and the special stackable bus connector having 104 pins (Figures 1 and 2).



Figure 1. Typical PC/104 Stack.

These systems can be programmed with the same development tools used with full-size PCs which reduces the need and cost of custom development efforts. Although only about 100 cm x 100 cm, PC/104 boards are very powerful for their size. PC/104 products are designed for minimal power consumption, small foot print, modularity, expandability, and ruggedness.

The computer system display configuration used during the 2003 demo on the transit buses is shown in figure 2 and consists of four major components:

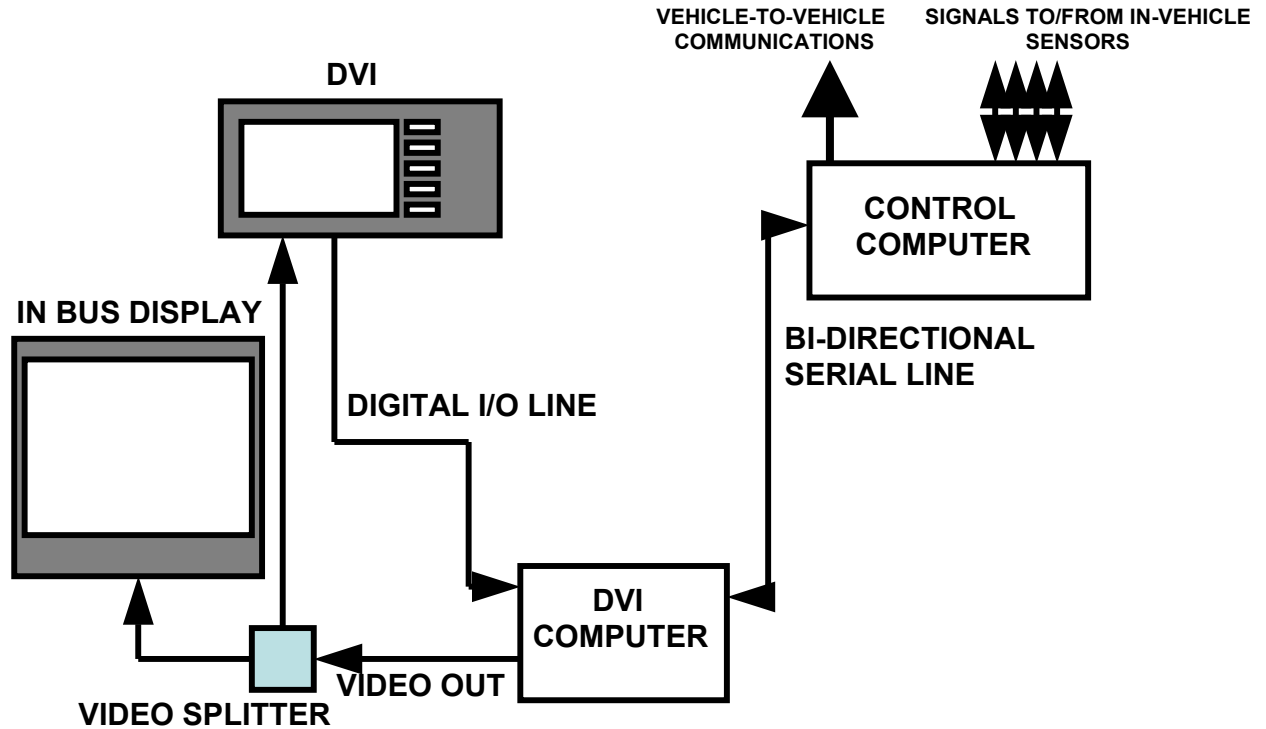


Figure 2. Computer system Configuration

1. **The Control Computer.** Used to read all sensors, issue speed and steering commands, and communicate state information to the other buses.

2-3. **The DVI (Driver Vehicle Interface) Computer/DVI Control Box.** This system is designed to generate a graphical interface of the system state to the driver. The state information is transferred to the DVI computer from the control computer using a bi-directional serial line. The DVI not only shows vehicle information, but also allows control mode/menu switching using a push button pad next to the display.

4. **In Bus Display.** A 19" computer monitor was installed viewable to the passengers showing the display from the DVI control box. This was used since the control box display is small and viewable only to the driver.

The vehicle control computers used in the PATH heavy vehicles consist of 10 stacked boards shown in Figure 3, and the DVI computer stack is shown in Figure 4.

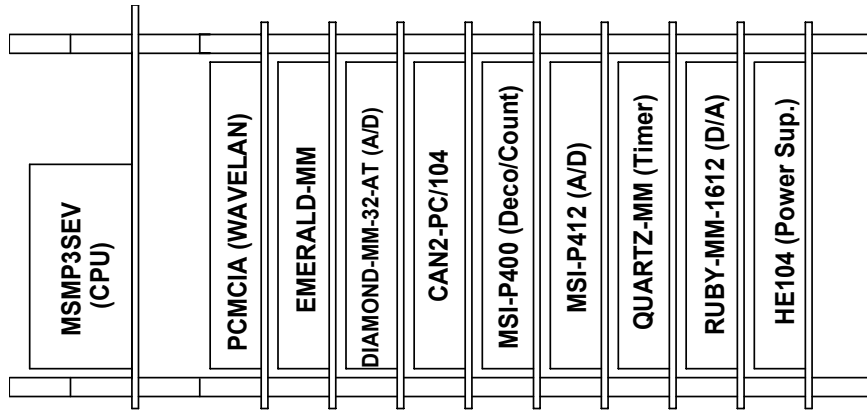


Figure 3. PC/104 Stack for the Control Computer.

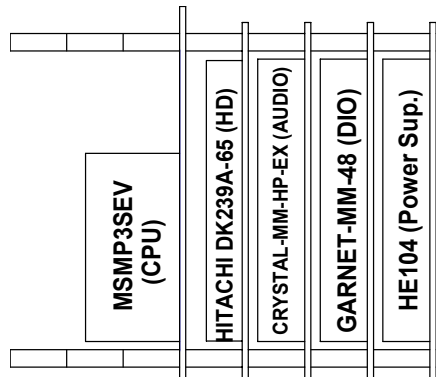


Figure 4. PC/104 Stack for the DVI Computer.

A photograph of the control computer as installed in the New Flyer transit bus is shown in Figure 5.



Figure 5. PC/104 Control Computer Installed in New Flyer Transit Bus.

The control computer is located in a compartment above the front window and behind the destination sign. The computer and associated cables/terminal blocks are mounted to a shelf that can be moved in and out of the compartment for ease of maintenance.

II. INDIVIDUAL BOARD BOARD DESCRIPTIONS AND FUNCTIONALITY

A brief description of each board and function is given below:

1. CPU: Intel 400 MHz P3Celeron MSM-P3SEV. The board includes VGA, 100/10Base-T LAN ethernet, E-IDE hard disk interface, 3.5" micro Floppy disk interface, and COM1, COM2 serial ports. Vendor: Advanced Digital Logic [1].



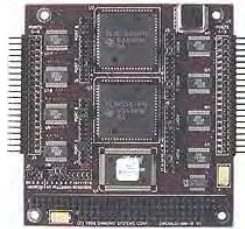
MSM-P3SEV

2. PCMCIA: 2-slot PCMCIA board MSMJ104D. Used for holding an Orinoco 802.11b vehicle-to-vehicle communications card. Vendor: Advanced Digital Logic.



MSMJ104D

3. Serial ports: 8-port RS-232 serial port board EMERALD-8232-XT. Includes 8 programmable digital I/O lines. The ports are used for reading the radar, lidar, rate gyroscope, J1587 bus, and GPS. It also communicates information to and from the DVI computer. Vendor: Diamond Systems [2].



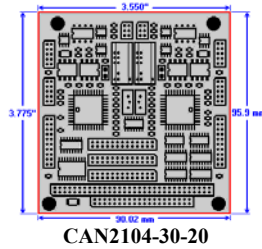
EMERALD-8232-XT

4. 32-channel 16-bit 200KHz analog to digital board DIAMOND-MM-32-AT. Includes 4 12-bit analog output channels, 24 programmable digital I/O lines, and 1 32-bit counter/timer. This board reads the analog signals from the magnetometers, accelerometer, steering potentiometer. Vendor: Diamond Systems.

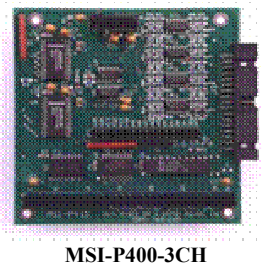


DIAMOND-MM-32-AT

5. Dual CAN field bus interface board: CAN2104-30-20. Includes 32 32-bit digital I/O lines. The J1939 data bus is read with this board. Vendor: SSV Embedded Systems [3].



6. 3-channel quadrature decoder/counter board MSI-P400-3CH. The steering wheel angular position is read by this card .Vendor: MicroComputer Systems [4].



7. 16-channel 12-bit digital to analog output board RUBY-MM-1612-XT. Includes 24 digital I/O lines. The analog outputs are used for the steering torque, accelerator pedal, and brake valve commmands, while the digital I/O lines monitor the magnetometer health states, and send control transitions. Vendor: Diamond Systems.



8. 50 watt DC/DC power supply board: HE104-512-V512. Vendor: Diamond Systems.



The following are PC/104 boards are used in the DVI computer:

9. SoundBlaster Pro compatible full-duplex, high-power 16-Bit stereo audio board CRYSTAL-MM-HP-EX. Vendor: Diamond Systems



CRYSTAL-MM-HP-EX

10. 48-line high current digital I/O: GARNET-MM-48. The push button pad inputs are read with this board. Vendor: Diamond Systems





GARNET-MM-48

All computers are enclosed in a PC/104 mounting enclosure, Can-Tainer. Vendor: Tri-M Engineering [5].



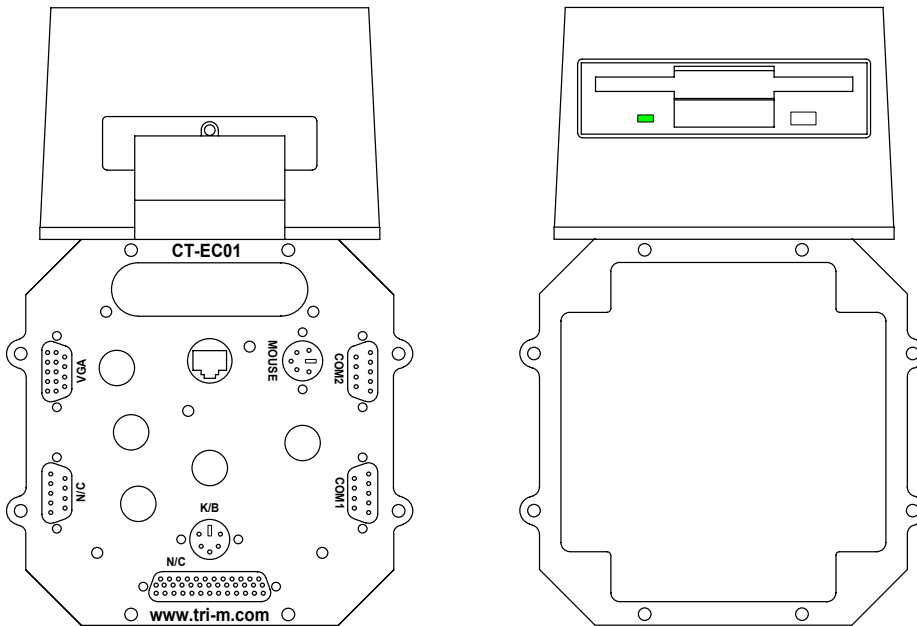
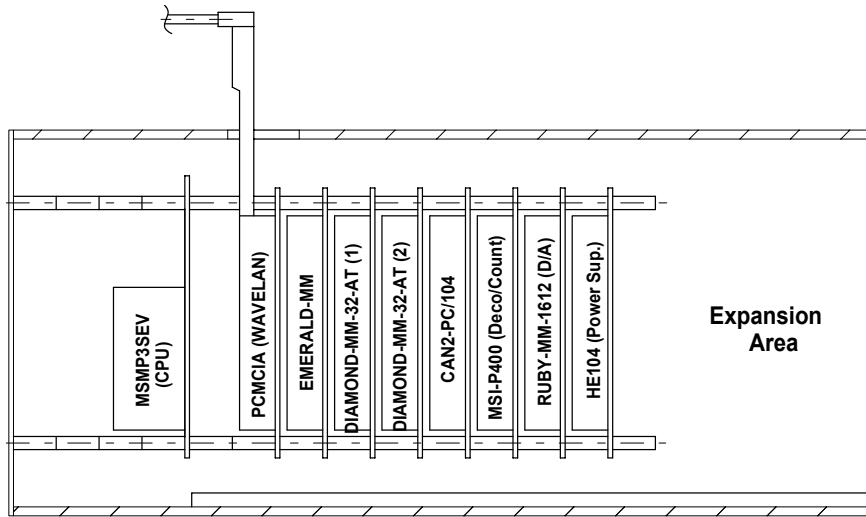
Can-Tainer

III. COMPUTER BOARD CONFIGURATION DOCUMENTATION FORMATS

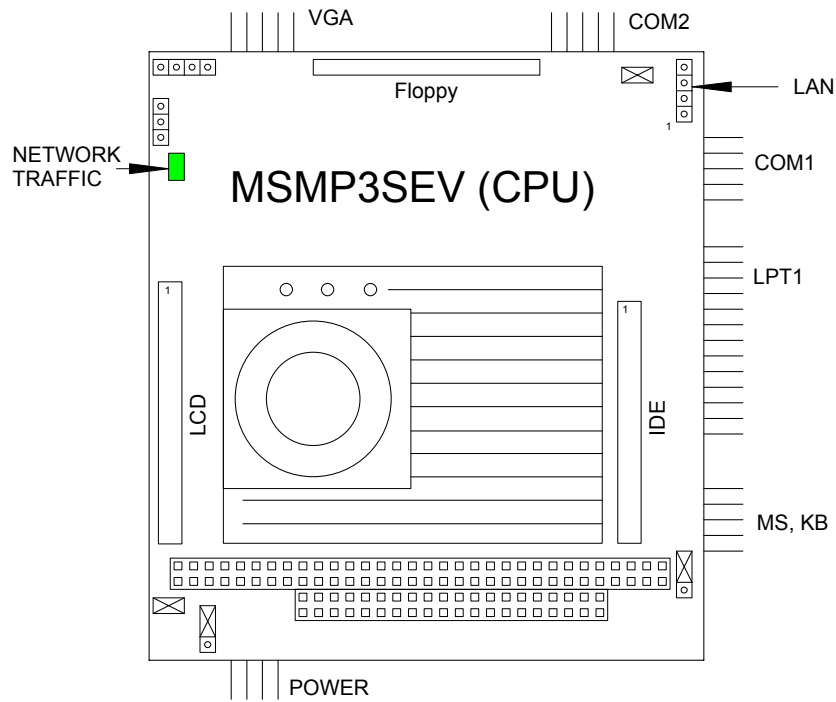
The remainder of this document provides a detailed configuration description of each PC/104 board. This includes to scale diagram of the board showing the location of each jumper for setting the base address, interrupt, etc. All pertinent I/O headers and bus connectors are clearly labeled. An installed jumper is denoted by the following symbol , and an uninstalled jumper by the symbol . It was decided to use a graphic description of the jumper placements since it is much easier to configure a board by simply viewing a diagram as opposed to using a purely text based representation. This greatly reduces the risk of mistakes and is much more efficient.

The second part of each board description includes a table describing the function of each I/O header pin. Each table usually includes the pin number, board function, and wiring color if applicable. Special implementation notes are also included for certain boards.

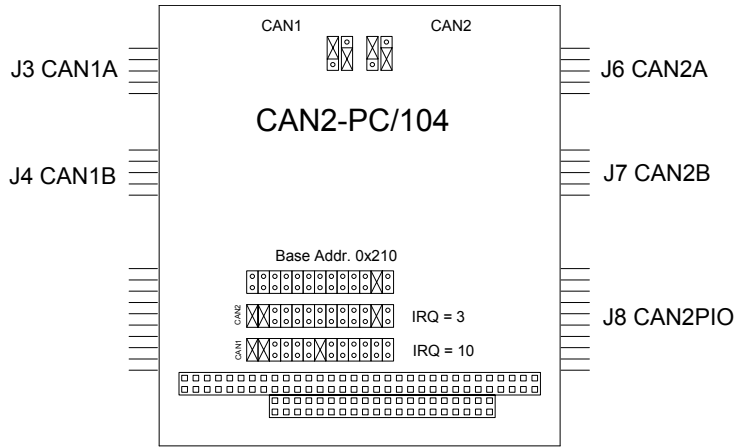
III. TRANSIT BUS PC-104 BOARD CONFIGURATION



MSMP3SEV (CPU) CONFIGURATION ^{Bus}



CAN2-PC/104 CONFIGURATION ^{Bus}

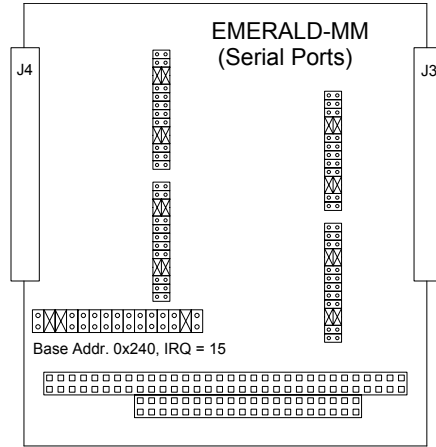


J3 CAN1A - Channel 1 Serial bus I/O			
J4 CAN1B - Channel 1 Serial bus I/O			
J6 CAN2A - Channel 2 Serial bus I/O			
J7 CAN2B - Channel 2 Serial bus I/O			
Pin	Name	Function	Direction
1	N/C	---	---
2	GND	Signal Ground	---
3	CAN-L	Signal LOW	---
4	CAN-H	Signal HIGH	---
5	GND	Signal Ground	---
6	N/C	---	---
7	N/C	---	---
8	N/C	---	---
9	N/C	---	---
10	N/C	---	---

J5 CAN1PIO - Channel 1 Parallel I/O				
Pin	Name	Function	Direction	Description
1	P1.0	Port 1 Bit 0	I/O	
2	P2.0	Port 2 Bit 0	I/O	
3	P1.1	Port 1 Bit 1	I/O	
4	P2.1	Port 2 Bit 1	I/O	
5	P1.2	Port 1 Bit 2	I/O	
6	P2.2	Port 2 Bit 2	I/O	
7	P1.3	Port 1 Bit 3	I/O	
8	P2.3	Port 2 Bit 3	I/O	
9	P1.4	Port 1 Bit 4	I/O	
10	P2.4	Port 2 Bit 4	I/O	
11	P1.5	Port 1 Bit 5	I/O	
12	P2.5	Port 2 Bit 5	I/O	
13	P1.6	Port 1 Bit 6	I/O	
14	P2.6	Port 2 Bit 6	I/O	
15	P1.7	Port 1 Bit 7	I/O	
16	P2.7	Port 2 Bit 7	I/O	
17	GND	Sig. Ground	---	
18	Vcc	+5 VDC	---	
19	GND	Sig. Ground	---	
20	Vcc	+5 VDC	---	

J8 CAN2PIO - Channel 2 Parallel I/O				
Pin	Name	Function	Direction	Description
1	P1.0	Port 1 Bit 0	I/O	
2	P2.0	Port 2 Bit 0	I/O	
3	P1.1	Port 1 Bit 1	I/O	
4	P2.1	Port 2 Bit 1	I/O	
5	P1.2	Port 1 Bit 2	I/O	
6	P2.2	Port 2 Bit 2	I/O	
7	P1.3	Port 1 Bit 3	I/O	
8	P2.3	Port 2 Bit 3	I/O	
9	P1.4	Port 1 Bit 4	I/O	
10	P2.4	Port 2 Bit 4	I/O	
11	P1.5	Port 1 Bit 5	I/O	
12	P2.5	Port 2 Bit 5	I/O	
13	P1.6	Port 1 Bit 6	I/O	
14	P2.6	Port 2 Bit 6	I/O	
15	P1.7	Port 1 Bit 7	I/O	
16	P2.7	Port 2 Bit 7	I/O	
17	GND	Sig. Ground	---	
18	Vcc	+5 VDC	---	
19	GND	Sig. Ground	---	
20	Vcc	+5 VDC	---	

EMERALD-MM (Serial Ports) CONFIGURATION

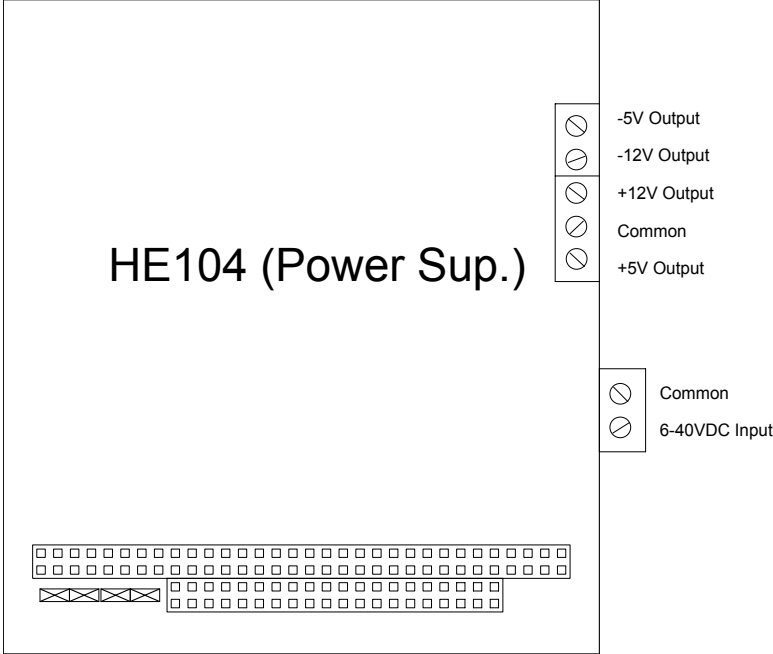


J3					J4				
Description	DCD	PIN	PIN	DSR	Description	DCD	PIN	PIN	DSR
Port 1 /dev/ser2 Eaton Vorad Radar (19200 Baud) 0x248	RXD 1	1	2	DSR 1	Port 5 /dev/ser6 J1587 Bus (9600 Baud) 0x268	RXD 5	1	2	DSR 5
	TXD 1	3	4	RTS 1		TXD 5	3	4	RTS 5
	DTR 1	5	6	CTS 1		DTR 5	5	6	CTS 5
	Grnd	7	8	RI 1		Grnd	7	8	RI 5
		9	10	DIO A		Grnd	9	10	DIO E
Port 2 /dev/ser3 Denso Lidar (19200 Baud) 0x250	RXD 2	11	12	DSR 2	Port 6 /dev/ser7 DVI Comms (115200 Baud) 0x270	RXD 6	11	12	DSR 6
	TXD 2	13	14	RTS 2		TXD 6	13	14	RTS 6
	DTR 2	15	16	CTS 2		DTR 6	15	16	CTS 6
	Grnd	17	18	RI 2		Grnd	17	18	RI 6
		19	20	DIO B		Grnd	19	20	DIO F
Port 3 /dev/ser4 KVH E-Core 2000 Gyro (9600 Baud) 0x258	RXD 3	21	22	DSR 3	Port 7 /dev/ser8 0x278	RXD 7	21	22	DSR 7
	TXD 3	23	24	RTS 3		TXD 7	23	24	RTS 7
	DTR 3	25	26	CTS 3		DTR 7	25	26	CTS 7
	Grnd	27	28	RI 3		Grnd	27	28	RI 7
		29	30	DIO C		Grnd	29	30	DIO G
Port 4 /dev/ser5 AshTech G-12 GPS (9600 Baud) 0x260	RXD 4	31	32	DSR 4	Port 8 /dev/ser9 0x280	RXD 8	31	32	DSR 8
	TXD 4	33	34	RTS 4		TXD 8	33	34	RTS 8
	DTR 4	35	36	CTS 4		DTR 8	35	36	CTS 8
	Grnd	37	38	RI 4		Grnd	37	38	RI 8
		39	40	DIO D		Grnd	39	40	DIO H

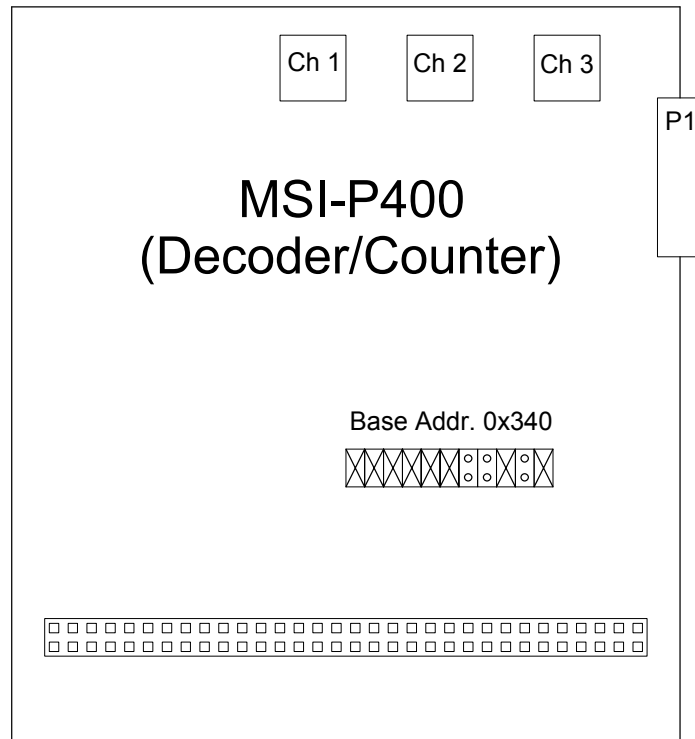
DIO CHANNEL	Description	DB15 Connector Pin
DIO A	Red LED (Output) (1=ON, 0 = OFF)	1
DIO B	Green LED (Output) (1=ON, 0 = OFF)	2
DIO C	Blue LED (Output) (1=ON, 0 = OFF)	3
DIO D	Amber LED (Output) (1=ON, 0 = OFF)	4
DIO E	Heart Beat (Output) (Toggles)	5
DIO F		6
DIO G		7
DIO H		8

NOTE: Pins 9-15 on DB15 are Grounds.

HE104 (POWER SUPPLY) CONFIGURATION Bus



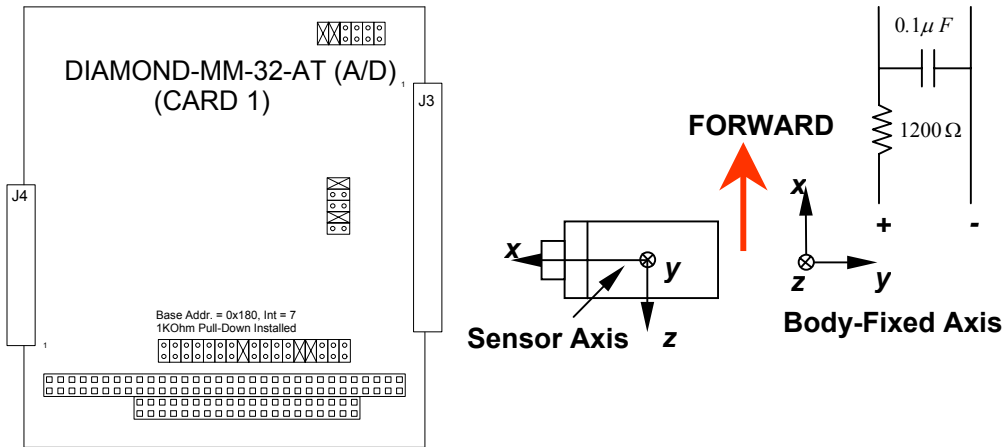
MSI-P400 (DECODER/COUNTER) CONFIGURATION ^{Bus}



NOTE: The First Color of Each TP Corresponds to the Sensor Signal and Underlined Colors are the Return Lines.

Header P1							
Color	Description		PIN	PIN		Description	Color
<u>Wh/Blk</u>	Steering Position Signal (A Ch Output)	FREQ1	<u>1</u>	<u>2</u>	REF1	Steering Position Signal (B Ch Output)	<u>Grn/Blk</u>
		FREQ2	<u>3</u>	<u>4</u>	REF2		
		FREQ3	<u>5</u>	<u>6</u>	REF3		
		N/C	<u>7</u>	<u>8</u>	N/C		
		Gnd	<u>9</u>	<u>10</u>	N/C		

DIAMOND-MM-32-AT (A/D) (CARD 1) CONFIGURATION



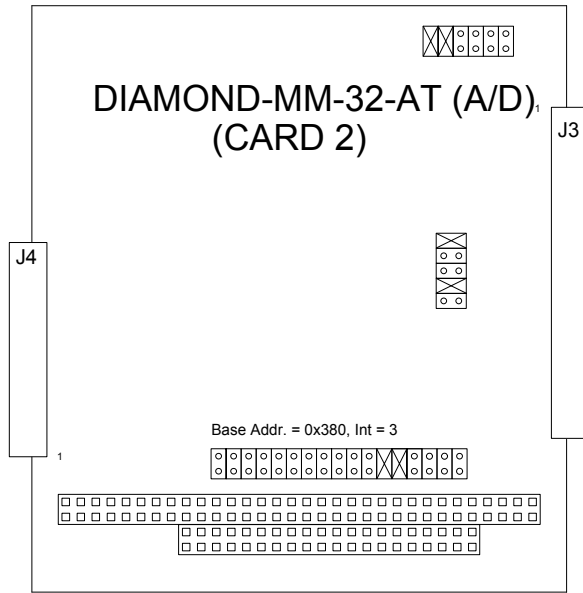
NOTE: Wire Sensor y-axis to Vertical Channel, Sensor x-axis to Lateral Channel

NOTE: The First Color of Each TP Corresponds to the Sensor Signal and Underlined Colors are the Return Lines.

P1

Color	Description	Func	Pin	Pin	Func	Description	Color
		Agnd	1	2	Agnd		
<u>Brn/Blu</u>	Frt-Left-Left Mag, Lat. Axis	Vin 0	3	4	Vin 16	Rear-Left Mag, Lat Axis	<u>Blu/Org</u>
<u>Yel/Blu</u>	Frt-Left-Left Mag, Vert. Axis	Vin 1	5	6	Vin 17	Rear-Left Mag, Vert Axis	<u>Blk/Org</u>
<u>Blu/Org</u>	Frt-Left Mag, Lat Axis	Vin 2	7	8	Vin 18	Rear-Cent-Left Mag, Lat. Axis	<u>Yel/Brn</u>
<u>Blk/Org</u>	Frt-Left Mag, Vert Axis	Vin 3	9	10	Vin 19	Rear-Cent-Left Mag, Vert. Axis	<u>Org/Brn</u>
<u>Yel/Brn</u>	Frt-Cent-Left Mag, Lat. Axis	Vin 4	11	12	Vin 20	Rear-Cent Mag, Lateral Axis	<u>Wh/Grn</u>
<u>Org/Brn</u>	Frt-Cent-Left Mag, Vert. Axis	Vin 5	13	14	Vin 21	Rear-Cent Mag, Vertical Axis	<u>Org/Grn</u>
<u>Wh/Grn</u>	Frt-Cent Mag, Lat Axis	Vin 6	15	16	Vin 22	Rear-Cent-Right Mag, Lat. Axis	<u>Wh/Blk</u>
<u>Org/Grn</u>	Frt-Cent Mag, Vert Axis	Vin 7	17	18	Vin 23	Rear-Cent-Right Mag, Vert. Axis	<u>Brn/Blk</u>
<u>Wh/Blk</u>	Frt-Cent-Right Mag, Lat. Axis	Vin 8	19	20	Vin 24	Rear-Right Mag, Lat. Axis	<u>Brn/Grn</u>
<u>Brn/Blk</u>	Frt-Cent-Right Mag, Vert. Axis	Vin 9	21	22	Vin 25	Rear-Right Mag, Vert. Axis	<u>Blk/Grn</u>
<u>Brn/Grn</u>	Frt-Right Mag, Lat Axis	Vin 10	23	24	Vin 26	Rear-Right-Right Mag, Lat. Axis	<u>Red/Brn</u>
<u>Blk/Grn</u>	Frt-Right Mag, Vert Axis	Vin 11	25	26	Vin 27	Rear-Right-Right Mag, Vert. Axis	<u>Red/Yel</u>
<u>Red/Brn</u>	Frt-Right-Right Mag, Lat. Axis	Vin 12	27	28	Vin 28	Potentiometer Connected to Steering	
<u>Red/Yel</u>	Frt-Right-Right Mag, Vert. Axis	Vin 13	29	30	Vin 29	Steering Motor Condition (<2V = ERROR)	
<u>Brn/Blu</u>	Rr-Left-Left Mag, Lat Axis	Vin 14	31	32	Vin 30	Accel. 1-axis (Longitudinal)	
<u>Yel/Blu</u>	Rr-Left-Left Mag, Vert Axis	Vin 15	33	34	Vin 31	Accel. 2-axis (-Lateral)	
		Vout 3	35	36	Vout 2		
		Vout 1	37	38	Vout 0		
		Vref Out	39	40	Agnd		
		A/D Convert	41	42	Ctr 2 Out/Dout 2		
		Dout 1	43	44	Ctr 0 Out/Dout 0		
		Extclk/Din 3	45	46	Extgate/Din 2		
		Gate 0/Din 1	47	48	Clk 0/Din 0		
		+5V	49	50	Dgnd		

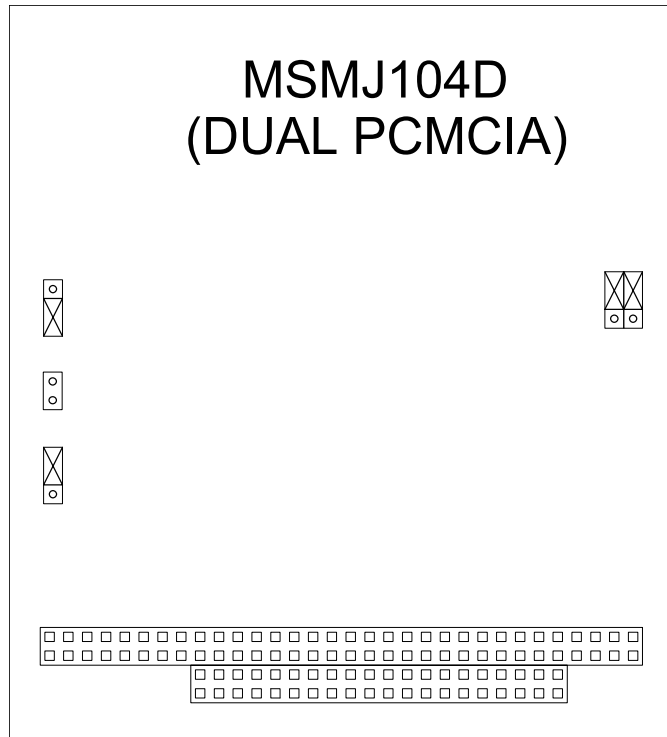
DIAMOND-MM-32-AT (A/D) (CARD 2) CONFIGURATION



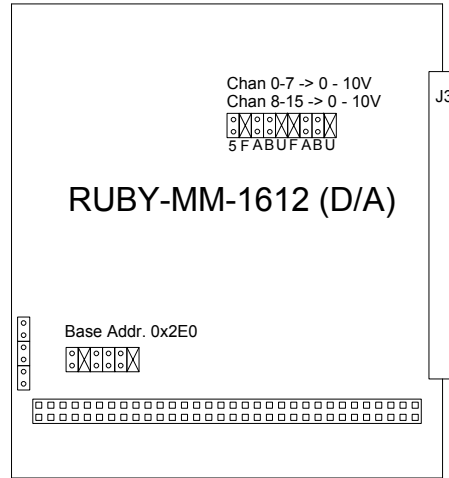
P1

Color	Description	Func	Pin	Pin	Func	Description	Color
		Agnd	1	2	Agnd		
	Rear Brake Axle Press	Vin 0	3	4	Vin 16		
	Middle Brake Axle Press (60')	Vin 1	5	6	Vin 17		
	Front Brake Axle Press	Vin 2	7	8	Vin 18		
	Rear Brake Monitor Press	Vin 3	9	10	Vin 19		
	Front Brake Monitor Press	Vin 4	11	12	Vin 20		
	Rear Brake Applied Press	Vin 5	13	14	Vin 21		
	Front Brake Applied Press	Vin 6	15	16	Vin 22		
	Accelerator Pedal Position	Vin 7	17	18	Vin 23		
	Accelerometer Temp.	Vin 8	19	20	Vin 24		
		Vin 9	21	22	Vin 25		
		Vin 10	23	24	Vin 26		
		Vin 11	25	26	Vin 27		
		Vin 12	27	28	Vin 28		
		Vin 13	29	30	Vin 29		
		Vin 14	31	32	Vin 30		
		Vin 15	33	34	Vin 31		
		Vout 3	35	36	Vout 2		
		Vout 1	37	38	Vout 0		
		Vref Out	39	40	Agnd		
		A/D Convert	41	42	Ctr 2 Out/Dout 2		
		Dout 1	43	44	Ctr 0 Out/Dout 0		
		Extclk/Din 3	45	46	Extgate/Din 2		
		Gate 0/Din 1	47	48	Clk 0/Din 0		
		+5V	49	50	Dgnd		

MSMJ1040D (DUAL PCMCIA) CONFIGURATION ^{Bus}



RUBY-MM-1612 (D/A) CONFIGURATION ^{Bus}

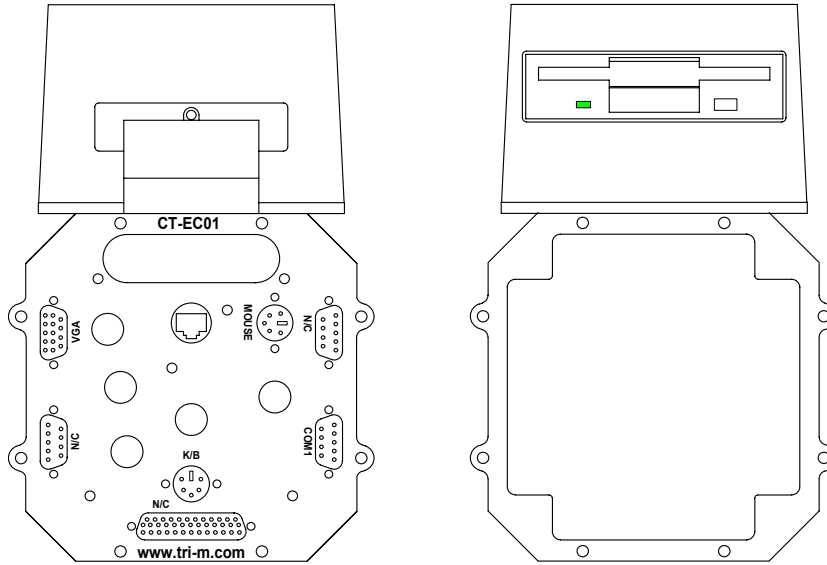
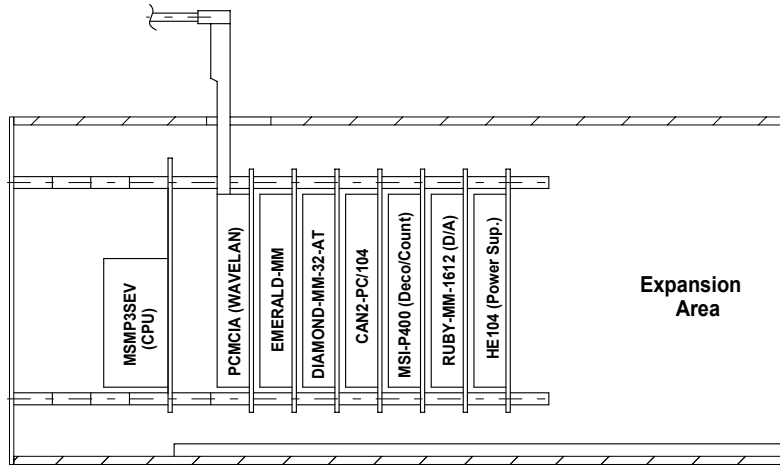


Header J3

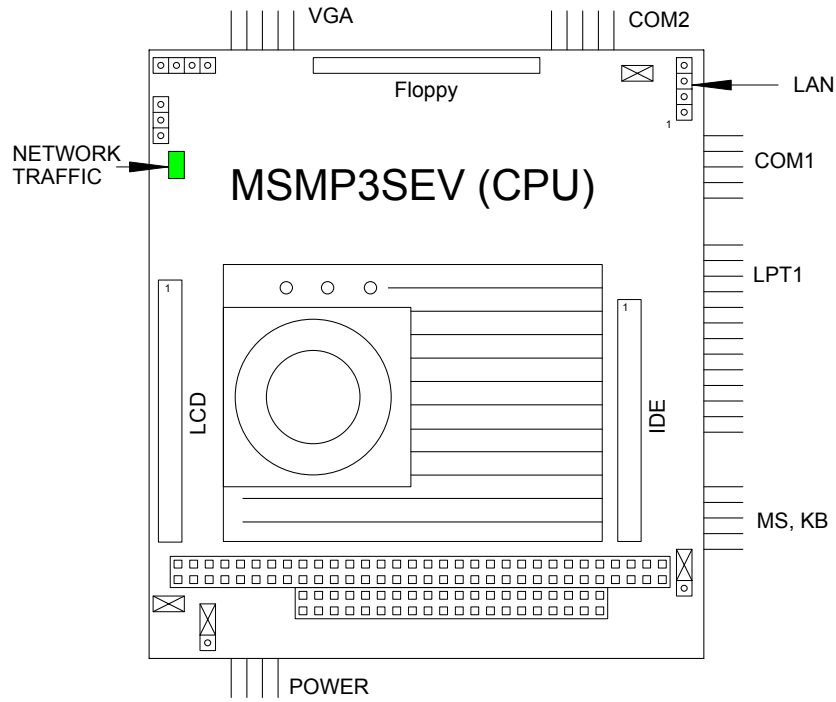
Color	Description		Pin	Pin		Description	Color
		Agnd	1	2	Vout 0	Steering Clutch (0=OFF, >9V=ON)	
		Agnd	3	4	Vout 1	Steering Torque Command	
		Agnd	5	6	Vout 2	Accelerator Pedal Command	
		Agnd	7	8	Vout 3	Front Brake Valve Command	
		Agnd	9	10	Vout 4	Rear Brake Valve Command	
		Agnd	11	12	Vout 5		
		Agnd	13	14	Vout 6		
		Agnd	15	16	Vout 7		
		Vout 8	17	18	Vout 9		
		Vout 10	19	20	Vout 11		
		Vout 12	21	22	Vout 13		
		Vout 14	23	24	Vout 15		
Yel/Wh	Rear-Left-Left Magnetometer Health Signal (Input)	DIO A7	25	26	DIO A6	Front-Right-Right Magnetometer Health Signal (Input)	Wh/Brn
Blu/Blk	Front-Right Magnetometer Health Signal (Input)	DIO A5	27	28	DIO A4	Front-Center-Right Magnetometer Health Signal (Input)	Org/Wh
Yel/Grn	Front-Center Magnetometer Health Signal (Input)	DIO A3	29	30	DIO A2	Front-Center-Left Magnetometer Health Signal (Input)	Blu/Grn
Yel/Blk	Front-Left Magnetometer Health Signal (Input)	DIO A1	31	32	DIO A0	Front-Left-Left Magnetometer Health Signal (Input)	Yel/Wh
	Auto Steering ON, (Relay State) (Input)	DIO B7	33	34	DIO B6	Steering Actuator Status, 1=ON, 0 = OFF, (Input)	
Wh/Brn	Rear-Right-Right Magnetometer Health Signal (Input)	DIO B5	35	36	DIO B4	Rear-Right Magnetometer Health Signal (Input)	Blu/Blk
Org/Wh	Rear-Center-Right Magnetometer Health Signal (Input)	DIO B3	37	38	DIO B2	Rear-Center Magnetometer Health Signal (Input)	Yel/Grn
Blu/Grn	Rear-Center-Left Magnetometer Health Signal (Input)	DIO B1	39	40	DIO B0	Rear-Left Magnetometer Health Signal (Input)	Yel/Blk
		DIO C7	41	42	DIO C6		
		DIO C5	43	44	DIO C4		
	AutoTransition Request (1=ON)	DIO C3	45	46	DIO C2	Manual Transition Request (1=ON)	
	Auto Throttle ON, (Relay State) (Input)	DIO C1	47	48	DIO C0 / Ext Trig	Auto Brake ON, (Relay State) (Input)	
		+ 5V	49	50	DIO Dgnd		

NOTE: For Magnetometer Health Signals 1 = OK, 0 = Fault

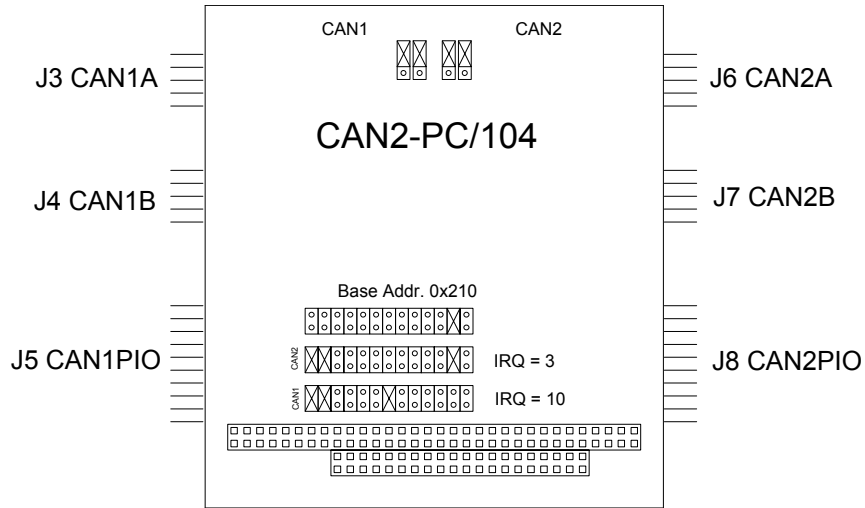
IV. TRUCK PC-104 BOARD CONFIGURATION



MSMP3SEV (CPU) CONFIGURATION Truck



CAN2-PC/104 CONFIGURATION Truck

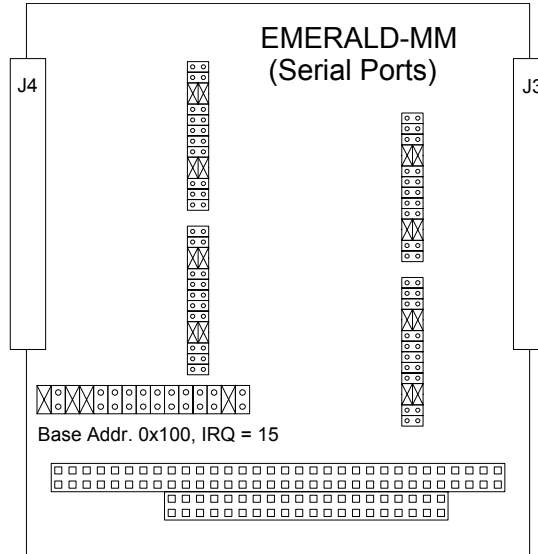


J3 CAN1A - Channel 1 Serial bus I/O			
J4 CAN1B - Channel 1 Serial bus I/O			
J6 CAN2A - Channel 2 Serial bus I/O			
J7 CAN2B - Channel 2 Serial bus I/O			
Pin	Name	Function	Direction
1	N/C	---	---
2	GND	Signal Ground	---
3	CAN-L	Signal LOW	---
4	CAN-H	Signal HIGH	---
5	GND	Signal Ground	---
6	N/C	---	---
7	N/C	---	---
8	N/C	---	---
9	N/C	---	---
10	N/C	---	---

J5 CAN1PIO - Channel 1 Parallel I/O				
Pin	Name	Function	Direction	Description
1	P1.0	Port 1 Bit 0	I/O	
2	P2.0	Port 2 Bit 0	I/O	
3	P1.1	Port 1 Bit 1	I/O	
4	P2.1	Port 2 Bit 1	I/O	
5	P1.2	Port 1 Bit 2	I/O	
6	P2.2	Port 2 Bit 2	I/O	
7	P1.3	Port 1 Bit 3	I/O	
8	P2.3	Port 2 Bit 3	I/O	
9	P1.4	Port 1 Bit 4	I/O	
10	P2.4	Port 2 Bit 4	I/O	
11	P1.5	Port 1 Bit 5	I/O	
12	P2.5	Port 2 Bit 5	I/O	
13	P1.6	Port 1 Bit 6	I/O	
14	P2.6	Port 2 Bit 6	I/O	
15	P1.7	Port 1 Bit 7	I/O	
16	P2.7	Port 2 Bit 7	I/O	
17	GND	Sig. Ground	---	
18	Vcc	+5 VDC	---	
19	GND	Sig. Ground	---	
20	Vcc	+5 VDC	---	

J8 CAN2PIO - Channel 2 Parallel I/O				
Pin	Name	Function	Direction	Description
1	P1.0	Port 1 Bit 0	I/O	
2	P2.0	Port 2 Bit 0	I/O	
3	P1.1	Port 1 Bit 1	I/O	
4	P2.1	Port 2 Bit 1	I/O	
5	P1.2	Port 1 Bit 2	I/O	
6	P2.2	Port 2 Bit 2	I/O	
7	P1.3	Port 1 Bit 3	I/O	
8	P2.3	Port 2 Bit 3	I/O	
9	P1.4	Port 1 Bit 4	I/O	
10	P2.4	Port 2 Bit 4	I/O	
11	P1.5	Port 1 Bit 5	I/O	
12	P2.5	Port 2 Bit 5	I/O	
13	P1.6	Port 1 Bit 6	I/O	
14	P2.6	Port 2 Bit 6	I/O	
15	P1.7	Port 1 Bit 7	I/O	
16	P2.7	Port 2 Bit 7	I/O	
17	GND	Sig. Ground	---	
18	Vcc	+5 VDC	---	
19	GND	Sig. Ground	---	
20	Vcc	+5 VDC	---	

EMERALD-MM (SERIAL PORTS) CONFIGURATION Truck

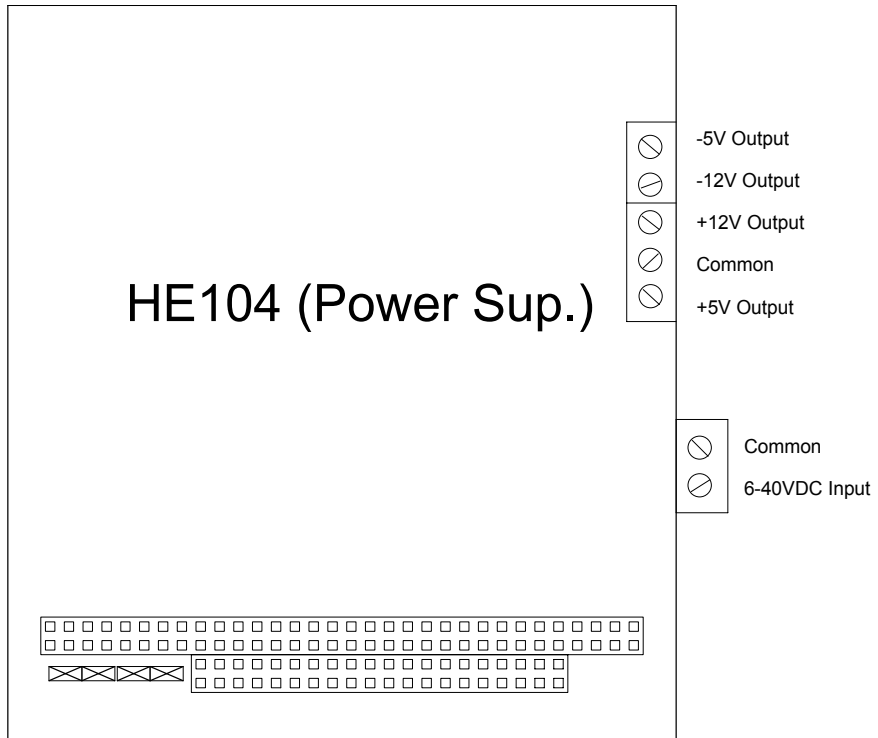


Header J3					Header J4				
Description	PIN	PIN		Description	PIN	PIN			
Port 1 /dev/ser2 Eaton Vorad Radar (19200 Baud)	DCD 1	1	2	DSR 1	Port 5 /dev/ser6 J1587 Bus (9600 Baud)	DCD 5	1	2	DSR 5
	RXD 1	3	4	RTS 1		RXD 5	3	4	RTS 5
	TXD 1	5	6	CTS 1		TXD 5	5	6	CTS 5
	DTR 1	7	8	RI 1		DTR 5	7	8	RI 5
	Grnd	9	10	DIO A		Grnd	9	10	DIO E
Port 2 /dev/ser3 Denso Lidar (19200 Baud)	DCD 2	11	12	DSR 2	Port 6 /dev/ser7	DCD 6	11	12	DSR 6
	RXD 2	13	14	RTS 2		RXD 6	13	14	RTS 6
	TXD 2	15	16	CTS 2		TXD 6	15	16	CTS 6
	DTR 2	17	18	RI 2		DTR 6	17	18	RI 6
	Grnd	19	20	DIO B		Grnd	19	20	DIO F
Port 3 /dev/ser4 KVH E-Core 2000 Gyro (9600 Baud)	DCD 3	21	22	DSR 3	Port 7 /dev/ser8	DCD 7	21	22	DSR 7
	RXD 3	23	24	RTS 3		RXD 7	23	24	RTS 7
	TXD 3	25	26	CTS 3		TXD 7	25	26	CTS 7
	DTR 3	27	28	RI 3		DTR 7	27	28	RI 7
	Grnd	29	30	DIO C		Grnd	29	30	DIO G
Port 4 /dev/ser5 AshTech G-12 GPS (9600 Baud)	DCD 4	31	32	DSR 4	Port 8 /dev/ser9	DCD 8	31	32	DSR 8
	RXD 4	33	34	RTS 4		RXD 8	33	34	RTS 8
	TXD 4	35	36	CTS 4		TXD 8	35	36	CTS 8
	DTR 4	37	38	RI 4		DTR 8	37	38	RI 8
	Grnd	39	40	DIO D		Grnd	39	40	DIO H

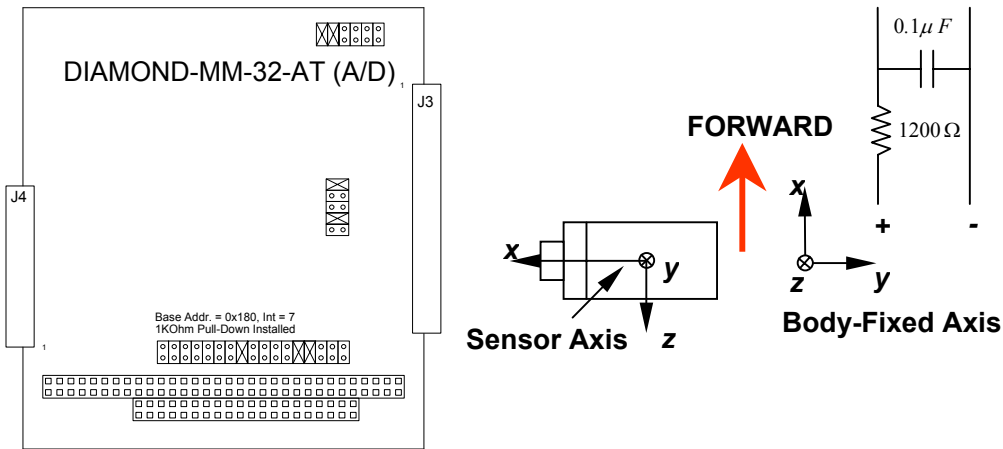
DIO CHANNEL	Description	DB15 Connector Pin
DIO A	Engine Fan Override (Output) (1=Override; 0=NOT)	1
DIO B	Engine Fan ON/OFF (Output) (1=ON; 0=OFF)	2
DIO C		3
DIO D		4
DIO E		5
DIO F		6
DIO G		7
DIO H		8

NOTE: Pins 9-15 on DB15 are Grounds

HE104 (POWER SUPPLY) CONFIGURATION ^{Truck}



DIAMOND-MM-32-AT (A/D) CONFIGURATION ^{Truck}



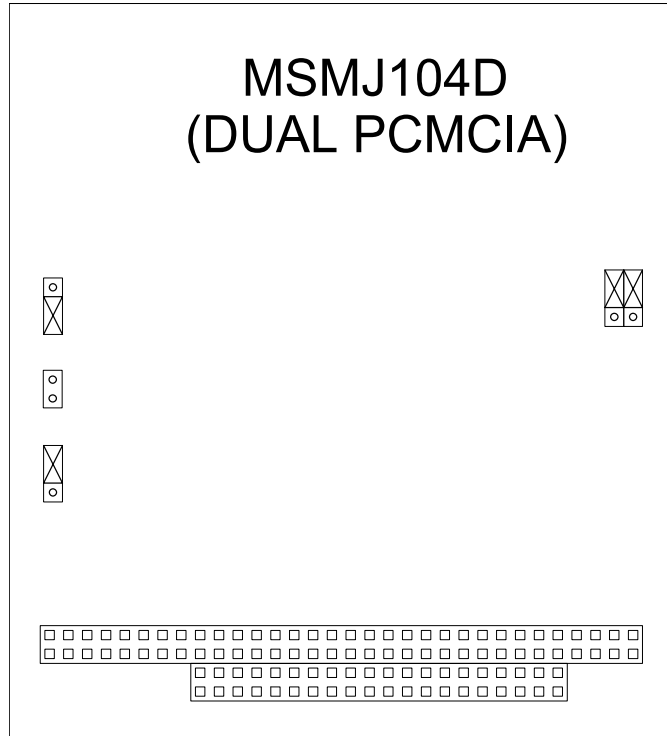
NOTE: Wire Sensor y-axis to Vertical Channel, Sensor x-axis to Lateral Channel

NOTE: The First Color of Each TP Corresponds to the Sensor Signal and Underlined Colors are the Return Lines.

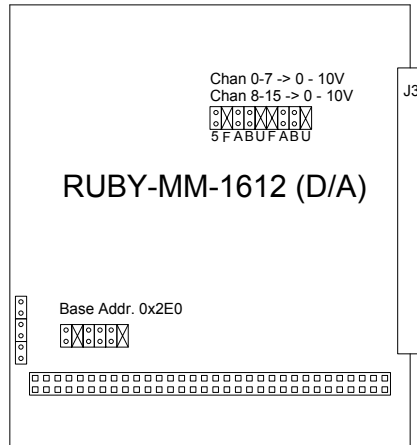
P1

Color	Description	Func	Pin	Pin	Func	Description	Color
		Agnd	1	2	Agnd		
	Front-Left Mag, Lat. Axis	Vin 0	3	4	Vin 16	Rear-Center-Right Mag, Lat. Axis	
	Front-Left Mag, Vert. Axis	Vin 1	5	6	Vin 17	Rear-Center-Right Mag, Vert. Axis	
	Front-Center-Left Mag, Lat. Axis	Vin 2	7	8	Vin 18	Rear-Right Mag, Lat. Axis	
	Front-Center-Left Mag, Vert. Axis	Vin 3	9	10	Vin 19	Rear-Right Mag, Vert. Axis	
	Front-Center Mag, Lat. Axis	Vin 4	11	12	Vin 20	Accel. 1-axis (Longitudinal)	
	Front-Center Mag, Vert. Axis	Vin 5	13	14	Vin 21	Accel. 2-axis (-Lateral)	
	Front-Center-Right Mag, Lat Axis	Vin 6	15	16	Vin 22	Throttle Pedal Position	
	Front-Center-Right Mag, Vert Axis	Vin 7	17	18	Vin 23	Front Brake Axle Pressure	
	Front-Right Mag, Lat. Axis	Vin 8	19	20	Vin 24	Middle Brake Axle Pressure	
	Front-Right Mag, Vert. Axis	Vin 9	21	22	Vin 25	Rear Brake Axle Pressure	
	Rear-Left Mag, Lat. Axis	Vin 10	23	24	Vin 26	Transmission Retarder	
	Rear-Left Mag, Vert. Axis	Vin 11	25	26	Vin 27		
	Rear-Center-Left Mag, Lat. Axis	Vin 12	27	28	Vin 28		
	Rear-Center-Left Mag, Vert. Axis	Vin 13	29	30	Vin 29		
	Rear-Center Mag, Lat. Axis	Vin 14	31	32	Vin 30	Potentiometer Connected to Steering	
	Rear-Center Mag, Vert. Axis	Vin 15	33	34	Vin 31	Steering Motor Condition (<2V = ERROR)	
		Vout 3	35	36	Vout 2		
		Vout 1	37	38	Vout 0		
		Vref Out	39	40	Agnd		
		A/D Convert	41	42	Ctr 2 Out/Dout 2		
		Dout 1	43	44	Ctr 0 Out/Dout 0		
		Extclk/Din 3	45	46	Extgate/Din 2		
		Gate 0/Din 1	47	48	Clk 0/Din 0		
		+5V	49	50	Dgnd		

MSMJ1040D (DUAL PCMCIA) CONFIGURATION Truck



RUBY-MM-1612 (D/A) CONFIGURATION Truck

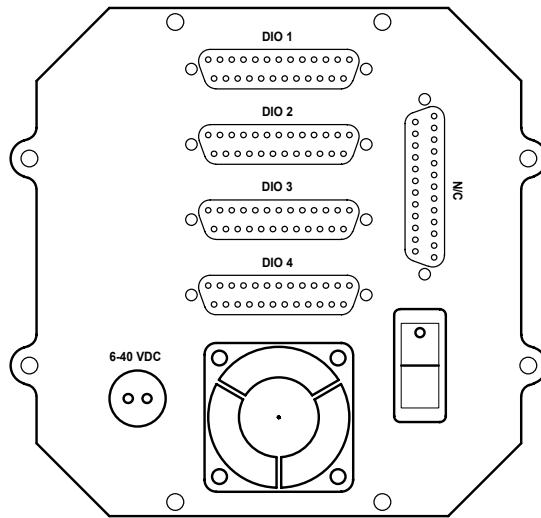
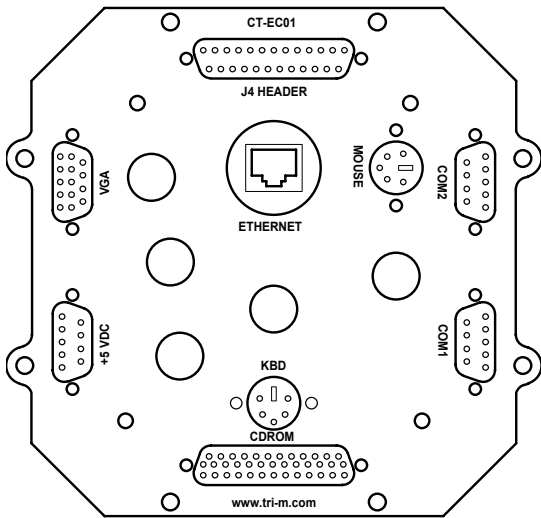
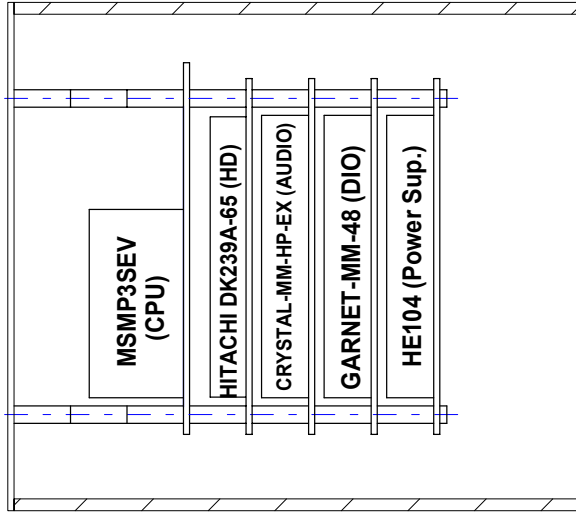


Header J3

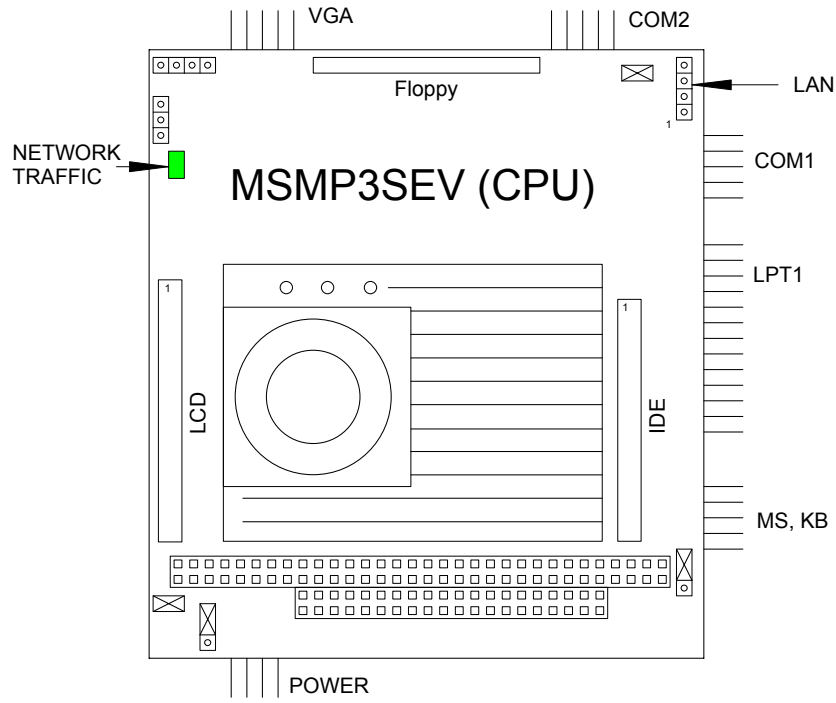
Description		Pin	Pin		Description
	Agnd	1	2	Vout 0	Steering Clutch (0=OFF, >9V=ON)
	Agnd	3	4	Vout 1	Steering Torque Command
	Agnd	5	6	Vout 2	Transmission Retarder
	Agnd	7	8	Vout 3	
	Agnd	9	10	Vout 4	
	Agnd	11	12	Vout 5	
	Agnd	13	14	Vout 6	
	Agnd	15	16	Vout 7	
	Vout 8	17	18	Vout 9	
	Vout 10	19	20	Vout 11	
	Vout 12	21	22	Vout 13	
	Vout 14	23	24	Vout 15	
Rear Center Magnetometer Health Signal (Input)	DIO A7	25	26	DIO A6	Rear Center Left Magnetometer Health Signal (Input)
Rear Left Magnetometer Health Signal (Input)	DIO A5	27	28	DIO A4	Front Right Magnetometer Health Signal (Input)
Front Center Right Magnetometer Health Signal (Input)	DIO A3	29	30	DIO A2	Front Center Magnetometer Health Signal (Input)
Front Center Left Magnetometer Health Signal (Input)	DIO A1	31	32	DIO A0	Front Left Magnetometer Health Signal (Input)
Steering Actuator Status (0 = OFF, 1 = ON) (Input)	DIO B7	33	34	DIO B6	Trailer Right Magnetometer Health Signal (Input)
Trailer Center Right Magnetometer Health Signal (Input)	DIO B5	35	36	DIO B4	Trailer Center Magnetometer Health Signal (Input)
Trailer Center Left Magnetometer Health Signal (Input)	DIO B3	37	38	DIO B2	Trailer Left Magnetometer Health Signal (Input)
Rear Right Magnetometer Health Signal (Input)	DIO B1	39	40	DIO B0	Rear Center Right Magnetometer Health Signal (Input)
	DIO C7	41	42	DIO C6	
	DIO C5	43	44	DIO C4	
Auto Transition Request (1=ON) (Input)	DIO C3	45	46	DIO C2	Manual Transition Request (1=ON) (Input)
Auto Steering ON (Relay State) (Input) (1=ON)	DIO C1	47	48	DIO C0 / Ext Trig	Engine Desired Fan State (Input) (1=ON; 0=OFF)
	+ 5V	49	50	DIO Dgnd	

NOTE: For Magnetometer Health Signals 1 = OK, 0 = Fault

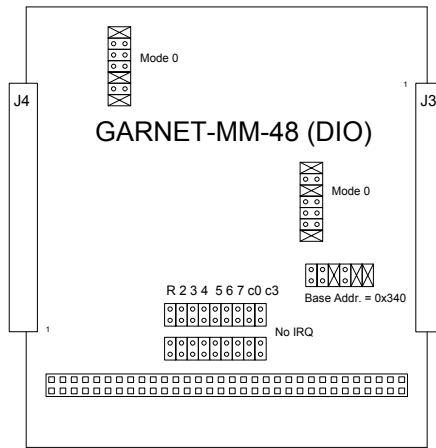
BUS PC-104 DVI BOARD CONFIGURATION



MSMP3SEV (CPU) CONFIGURATION ^{DVI}

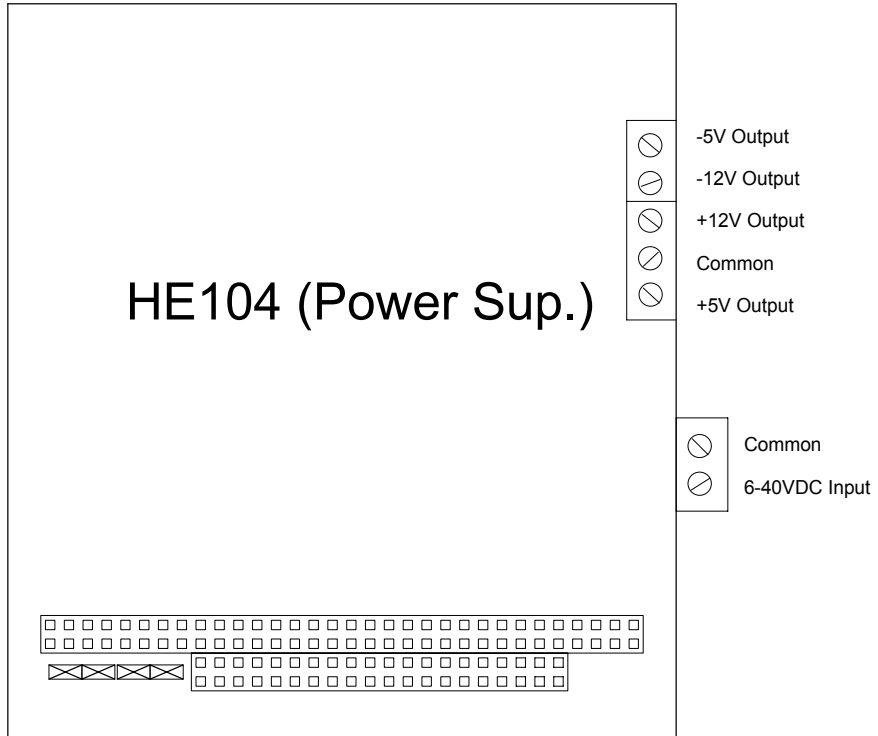


GARNET-MM-48 (Digital I/O) ^{DVI}



J3					J4				
DB25 Connector 1 (DIO 1)					DB25 Connector 3 (DIO 3)				
Description	Pin	Pin	Pin	Gnd	Description	Pin	Pin	Pin	Gnd
Number Pad – 8	A7	1/1	2/14	Gnd		A7	1/1	2/14	Gnd
Number Pad – 7	A6	3/2	4/15	Gnd		A6	3/2	4/15	Gnd
Number Pad – 6	A5	5/3	6/16	Gnd		A5	5/3	6/16	Gnd
Number Pad – 5	A4	7/4	8/17	Gnd		A4	7/4	8/17	Gnd
Number Pad – 4	A3	9/5	10/18	Gnd		A3	9/5	10/18	Gnd
Number Pad – 3	A2	11/6	12/19	Gnd		A2	11/6	12/19	Gnd
Number Pad – 2	A1	13/7	14/20	Gnd		A1	13/7	14/20	Gnd
Number Pad – 1	A0	15/8	16/21	Gnd		A0	15/8	16/21	Gnd
Select – Yellow	C7	17/9	18/22	Gnd		C7	17/9	18/22	Gnd
Select – Green	C6	19/10	20/23	Gnd		C6	19/10	20/23	Gnd
Scrn Sel – 6 (Top)	C5	21/11	22/24	Gnd		C5	21/11	22/24	Gnd
Scrn Sel – 5	C4	23/12	24/25	Gnd		C4	23/12	24/25	Gnd
DB25 Connector 2 (DIO 2)					DB25 Connector 4 (DIO 4)				
Description	Pin	Pin	Pin	Gnd	Description	Pin	Pin	Pin	Gnd
Scrn Sel – 4	C3	25/1	26/14	Gnd		C3	25/1	26/14	Gnd
Scrn Sel – 3	C2	27/2	28/15	Gnd		C2	27/2	28/15	Gnd
Scrn Sel – 2	C1	29/3	30/16	Gnd		C1	29/3	30/16	Gnd
Scrn Sel – 1 (Bottom)	C0	31/4	32/17	Gnd		C0	31/4	32/17	Gnd
Direction Pad – Up	B7	33/5	34/18	Gnd		B7	33/5	34/18	Gnd
Direction Pad – Left	B6	35/6	36/19	Gnd		B6	35/6	36/19	Gnd
Direction Pad – Right	B5	37/7	38/20	Gnd		B5	37/7	38/20	Gnd
Direction Pad - Down	B4	39/8	40/21	Gnd		B4	39/8	40/21	Gnd
Number Pad – #	B3	41/9	42/22	Gnd		B3	41/9	42/22	Gnd
Number Pad – 0	B2	43/10	44/23	Gnd		B2	43/10	44/23	Gnd
Number Pad – *	B1	45/11	46/24	Gnd		B1	45/11	46/24	Gnd
Number Pad – 9	B0	47/12	48/25	Gnd		B0	47/12	48/25	Gnd
	+5	49	50	Gnd		+5	49	50	Gnd

HE104 (Power Supply) CONFIGURATION ^{DVI}



VENDOR REFERENCES

- [1] Advanced Digital Logic, Inc
4411 Morena Blvd., Suite 230
San Diego, CA 92117
Tel. (858) 490-0597
Fax: (858) 490-0599
www.adlogic-pc104.com

- [2] Diamond Systems Corporation
8430-D Central Avenue
Newark, CA 94560
Tel: (510) 456-7800
Fax: (510) 456-7878
www.diamondsystems.com

- [3] SSV Embedded Systems
Heisterbergallee 72
D-30453 Hannover, Germany
Tel.: +49-(0) 511-400000
Fax.: +49-(0) 511-4000040
www.ssv-embedded.de

- [4] MicroComputer Systems
1814 Ryder Drive
Baton Rouge, LA 70808
Tel: (225) 769-2154
Fax: (225) 769-2155
www.microcomputersystems.com

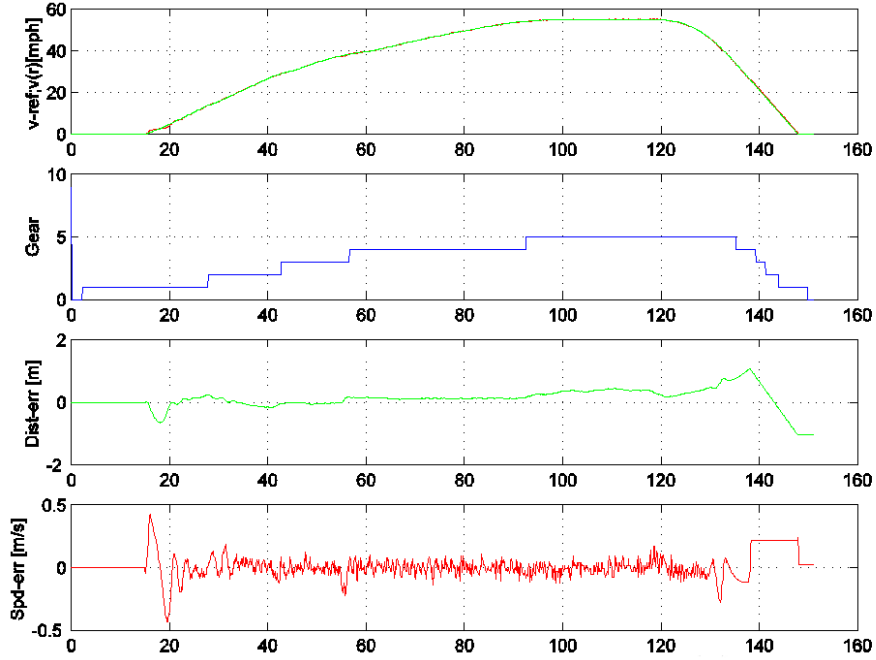
- [5] Tri-M Systems and Engineering Inc.
Unit 100
1407 Kebet way
Port Coquitlam, BC V3C 6L3
Canada
Tel: 604.945.9565
Fax: 604.945.9566
www.tri-m.com

Appendix B

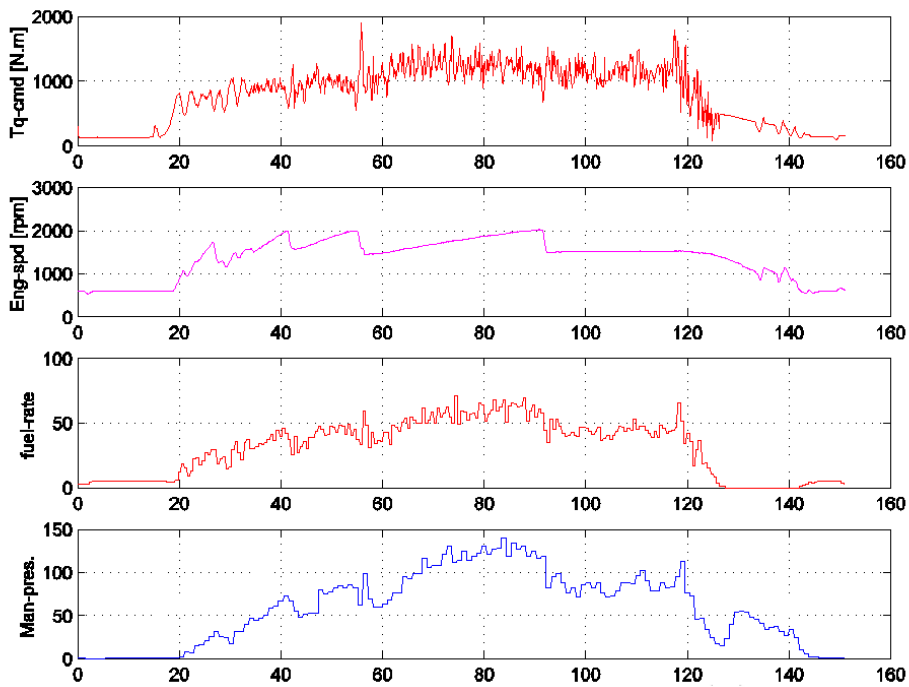
Example Truck Speed Tracking Test Data

This Appendix includes plots of example data sets from multiple test runs of a single Freightliner Century truck automatically following prescribed speed profiles on the Crows Landing test track. These cases cover a variety of truck loading conditions (empty, half loaded and fully loaded), maximum commanded speed (55, 30, 25 and 10 mph), maximum deceleration rate (1.2, 1.0, 0.9, 0.8, 0.7, and 0.5 m/s²) and braking system activations (EBS, compression brake and transmission retarder in various combinations).

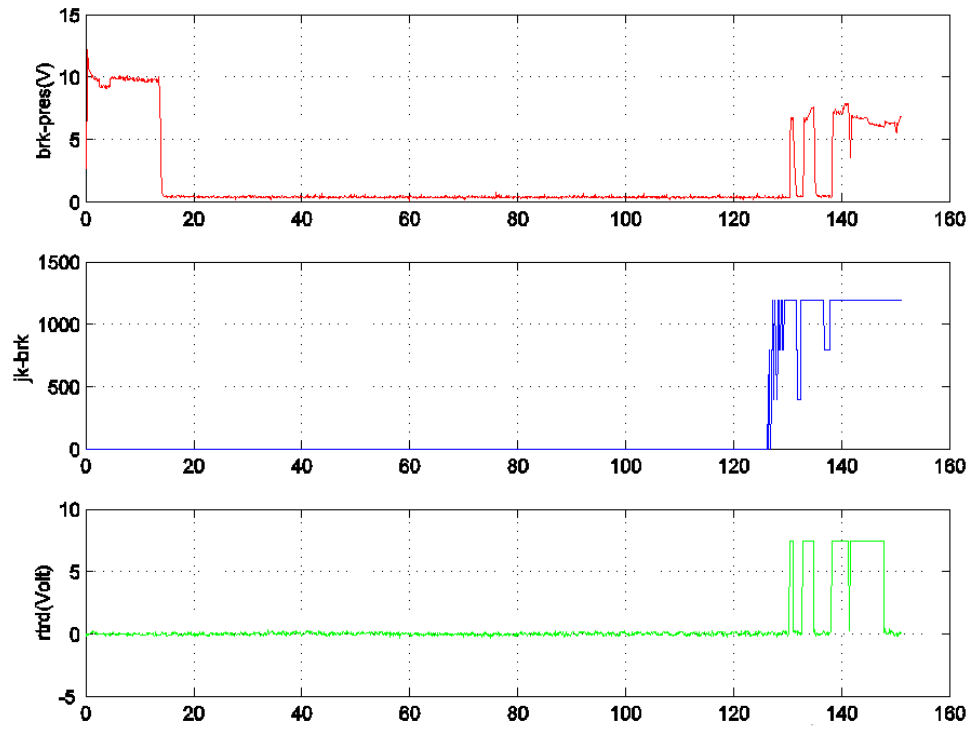
Empty Flat Trailer, Run 1: max_spd=55[mph], max_dec=1.2[m/s^2], EBS + Jake + trtd



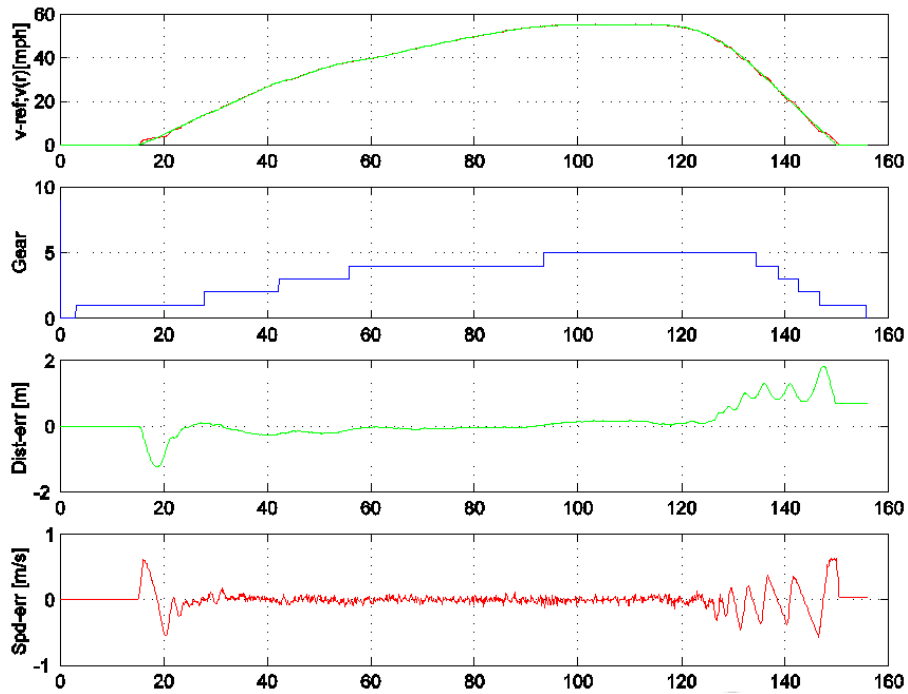
Empty Flat Trailer, Run 1: max_spd=55[mph], max_dec=1.2[m/s^2], EBS + Jake + trtd



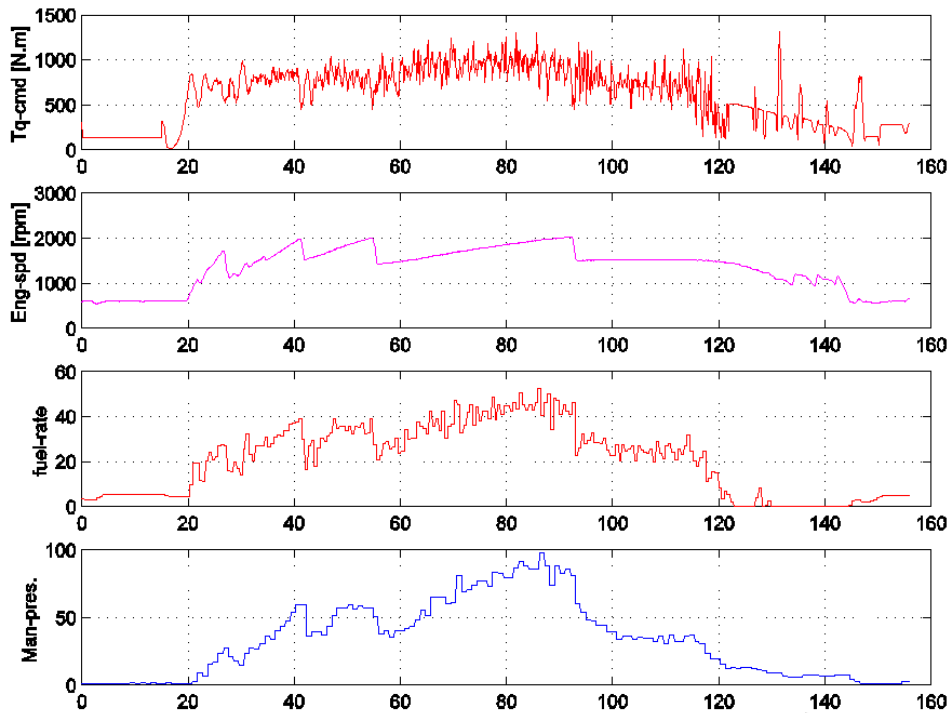
Empty Flat Trailer, Run 1: max_spd=55[mph], max_dec=1.2[m/s^2], EBS + Jake + trfd



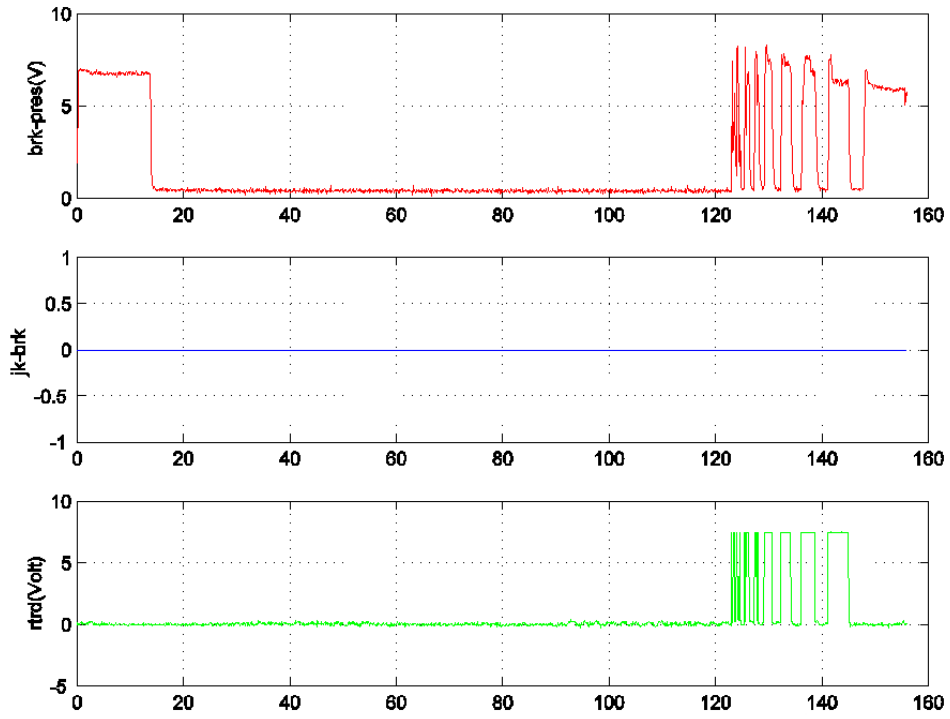
Empty Flat Trailer, Run 2: max_spd=55, max_dec=1.0, EBS + trtd



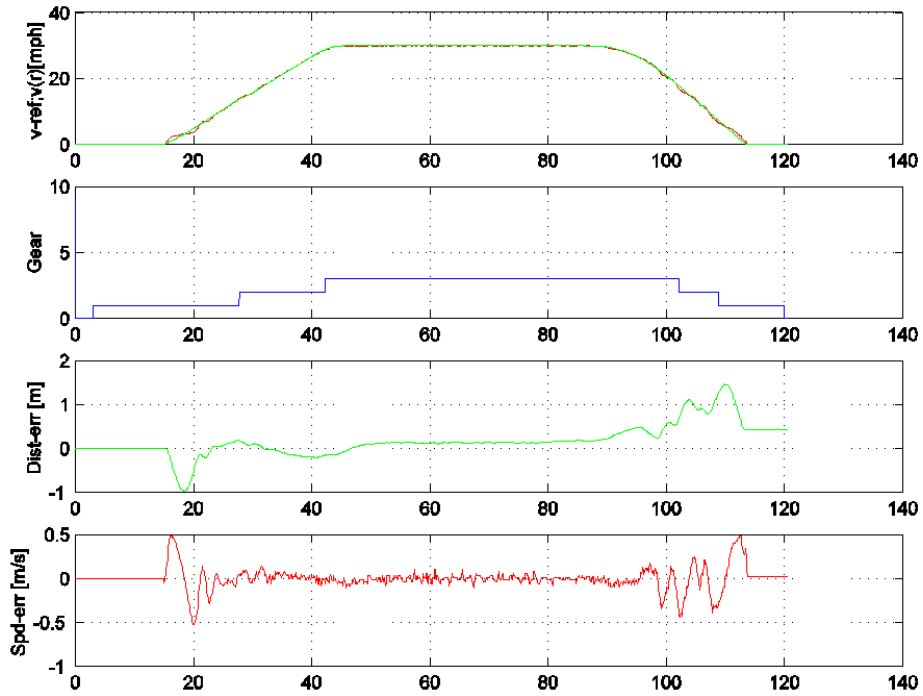
Empty Flat Trailer, Run 2: max_spd=55, max_dec=1.0, EBS + trtd



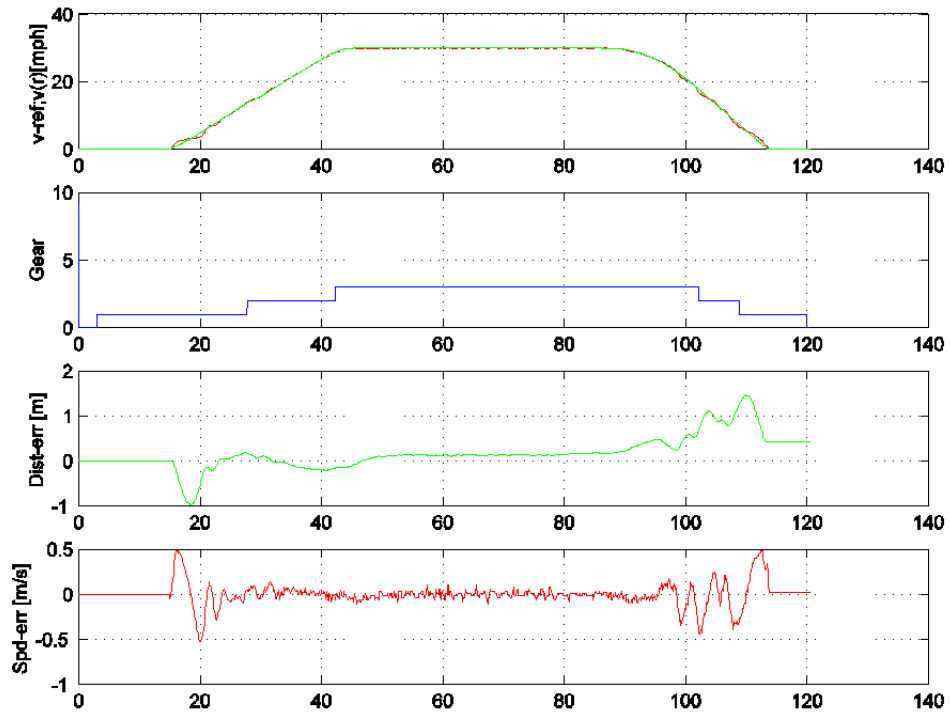
Empty Flat Trailer, Run 2: max_spd=55, max_dec=1.0, EBS + trtd



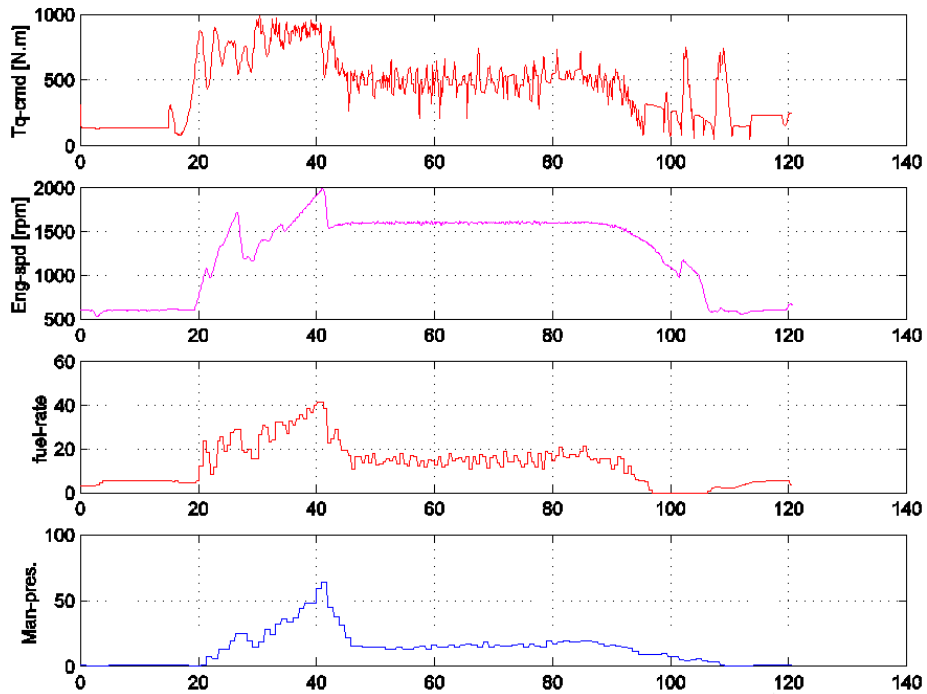
Empty Flat Trailer, Run 3: max_spd=30, max_dec=0.7, EBS + trtd



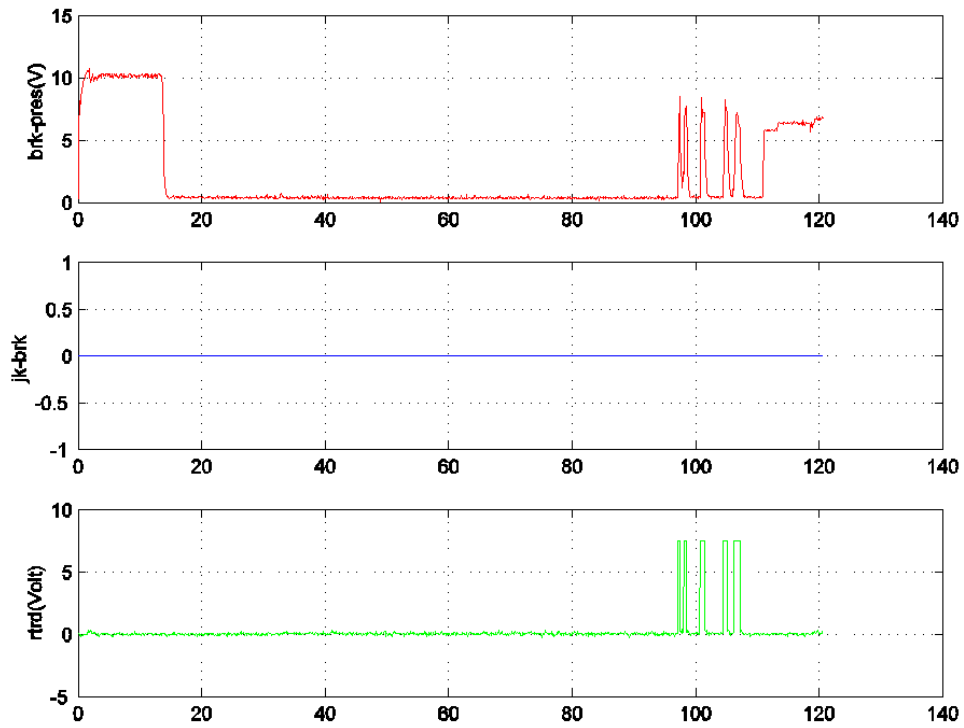
Empty Flat Trailer, Run 3: max_spd=30, max_dec=0.7, EBS + trtd



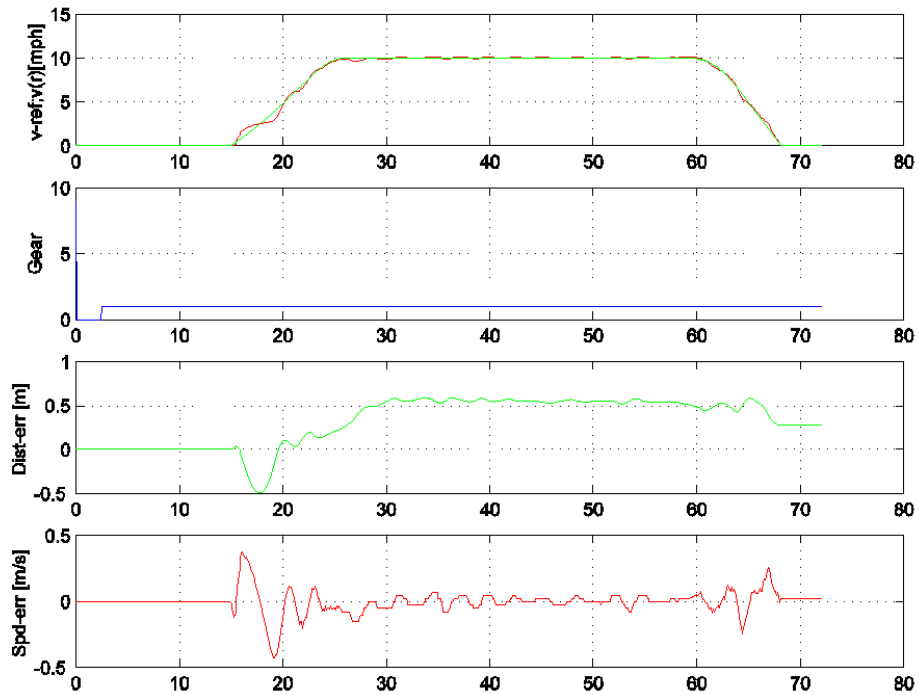
Empty Flat Trailer, Run 3: max_spd=30, max_dec=0.7, EBS + trtd



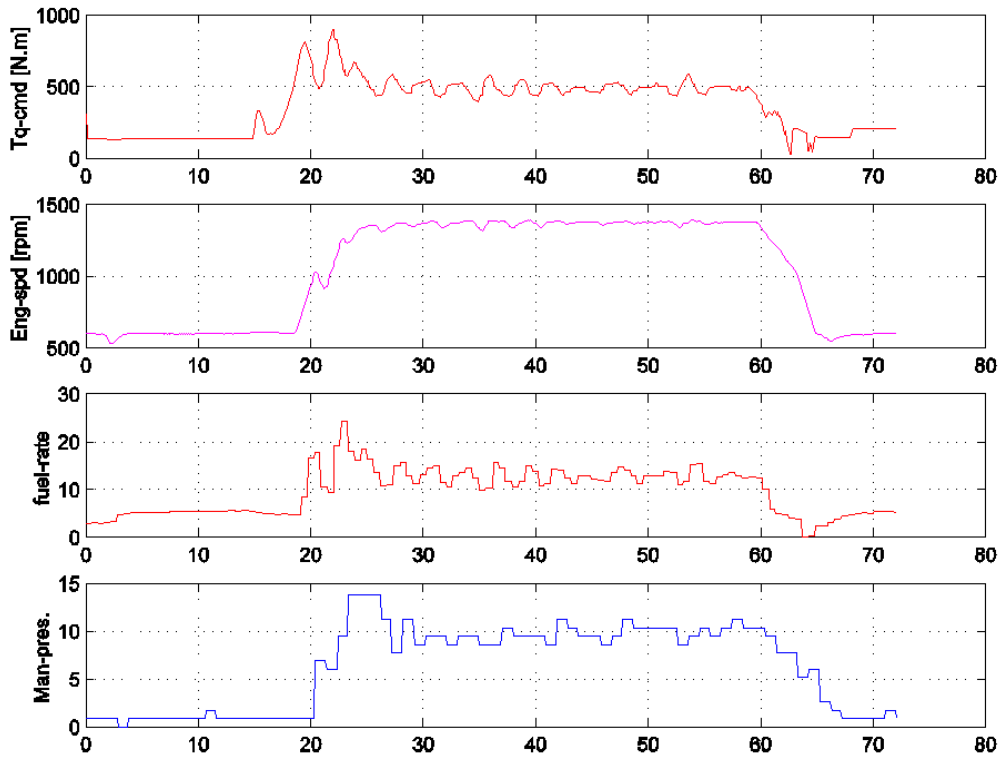
Empty Flat Trailer, Run 3: $\text{max_spd}=30$, $\text{max_dec}=0.7$, EBS + trtd



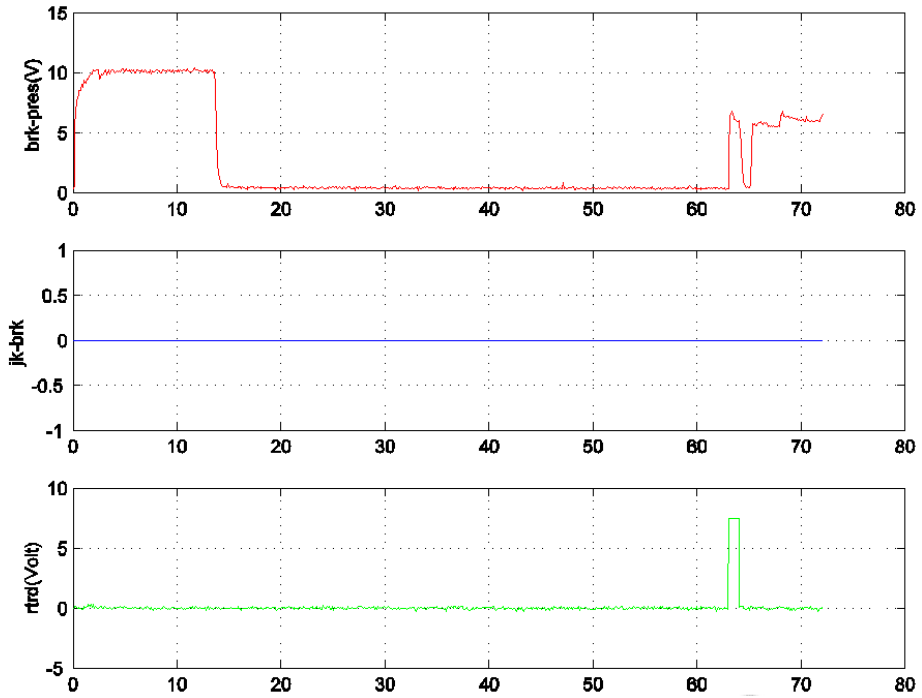
Empty Flat Trailer, Run 4: $\text{max_spd}=10$, $\text{max_dec}=0.7$, EBS + trtd



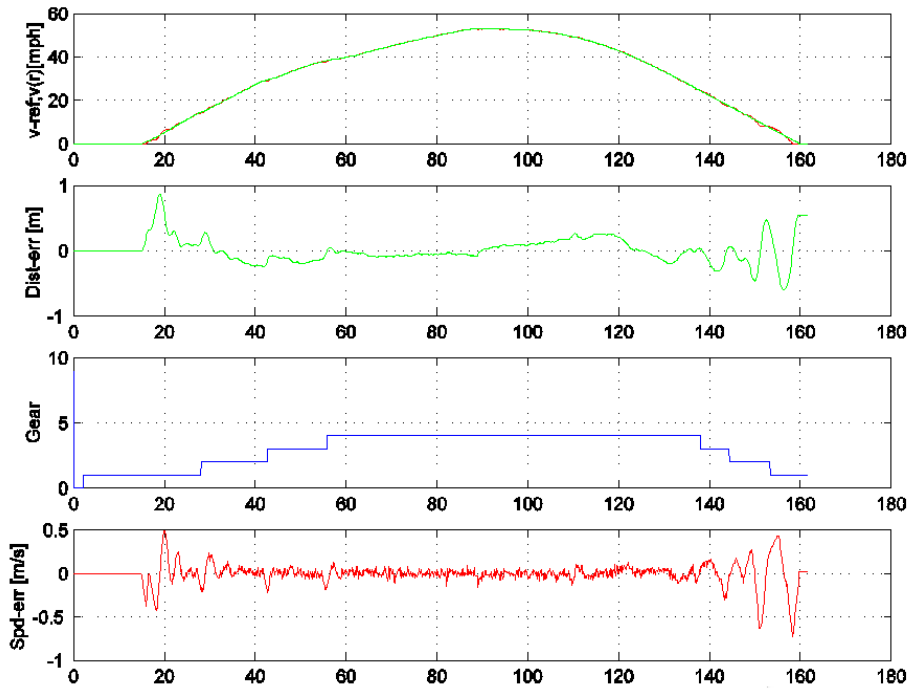
Empty Flat Trailer, Run 4: $\text{max_spd}=10$, $\text{max_dec}=0.7$, EBS + trtd



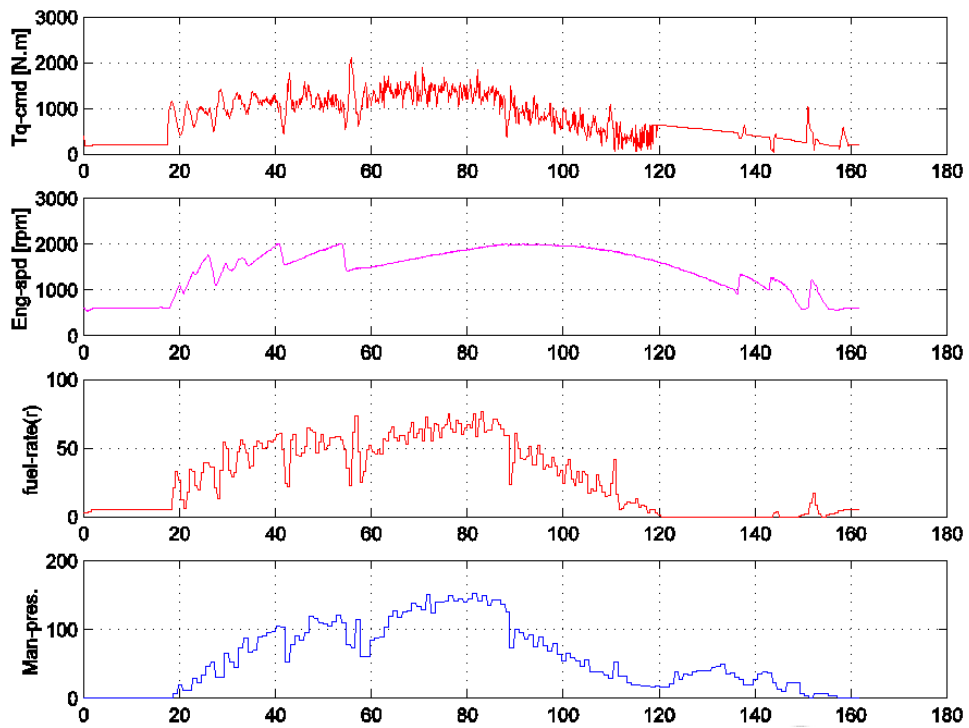
Empty Flat Trailer, Run 4: $\text{max_spd}=10$, $\text{max_dec}=0.7$, EBS + trtd



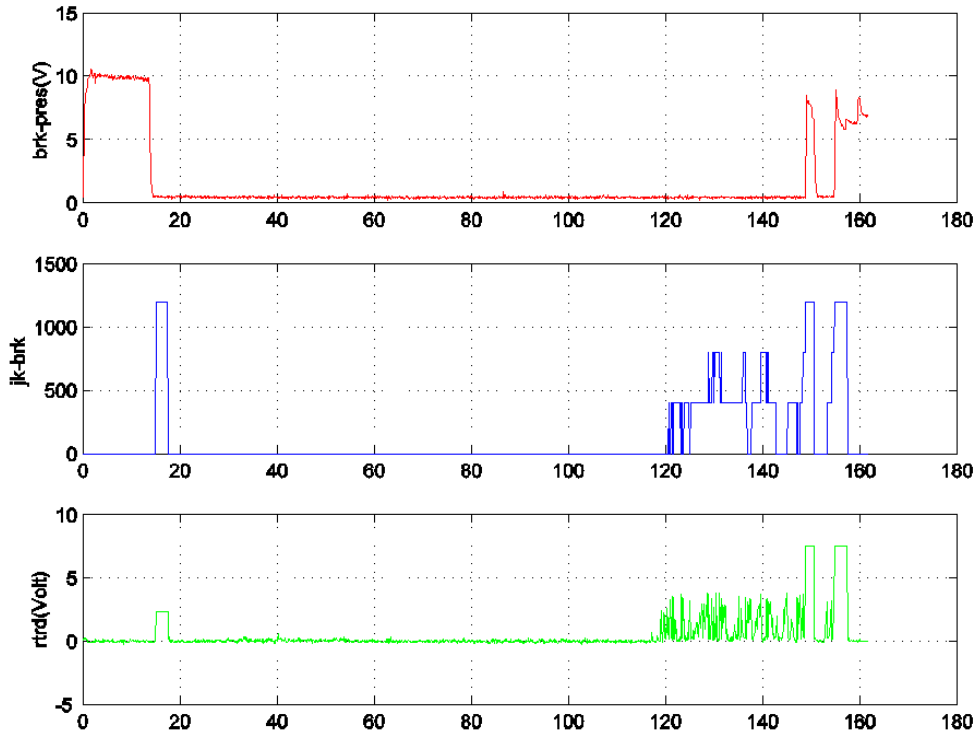
Half Loaded, Run 1: `max_spd=55, max_dec=0.5, EBS + Jake + trtd`



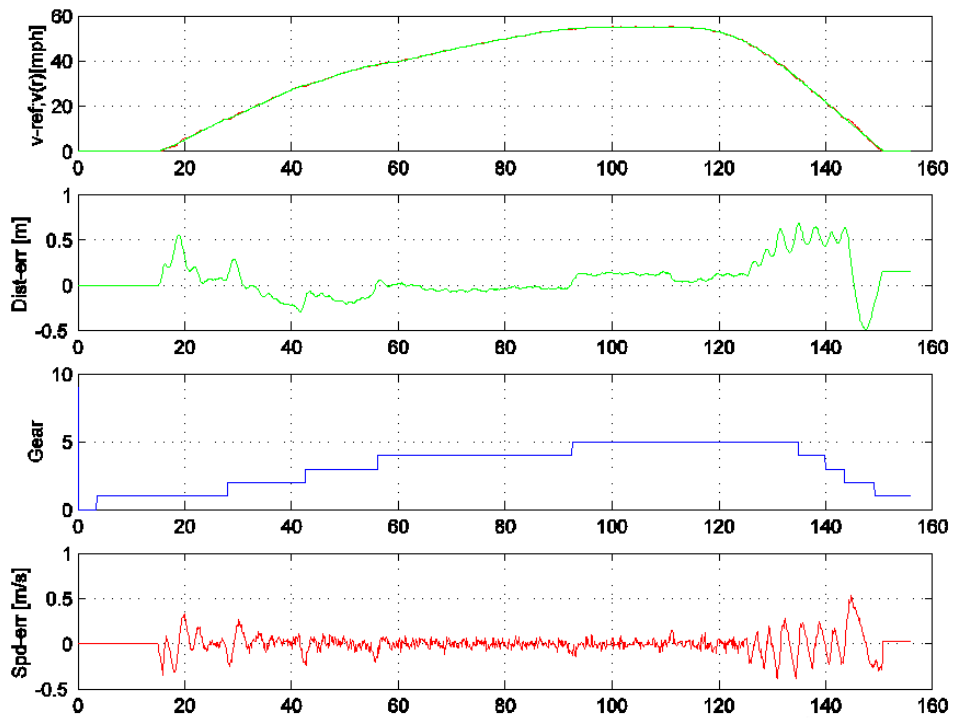
Half Loaded, Run 1: `max_spd=55, max_dec=0.5, EBS + Jake + trtd`



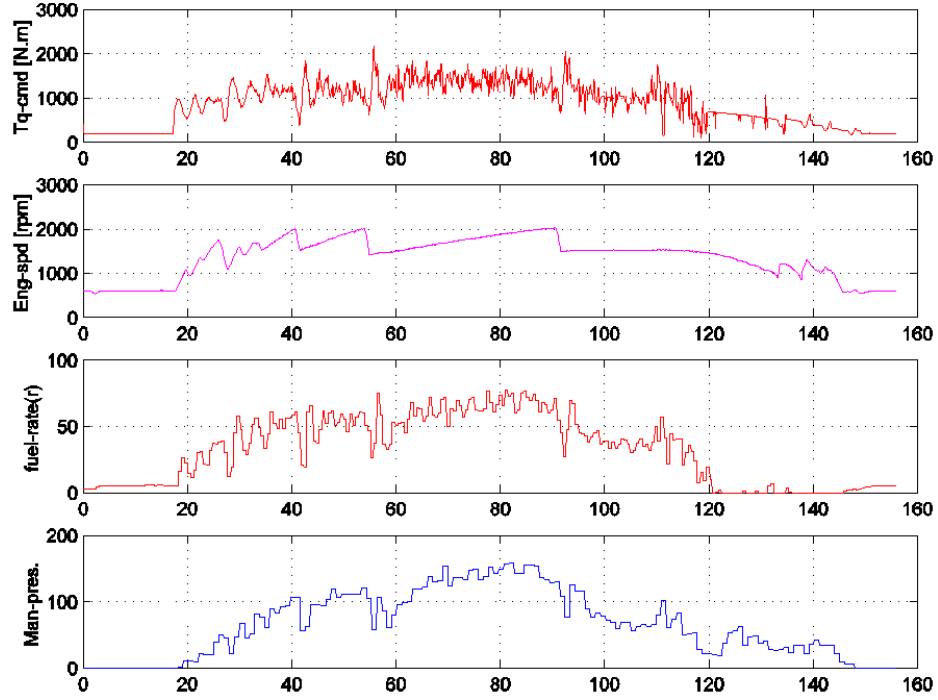
Half Loaded, Run 1: `max_spd=55, max_dec=0.5, EBS + Jake + trtd`



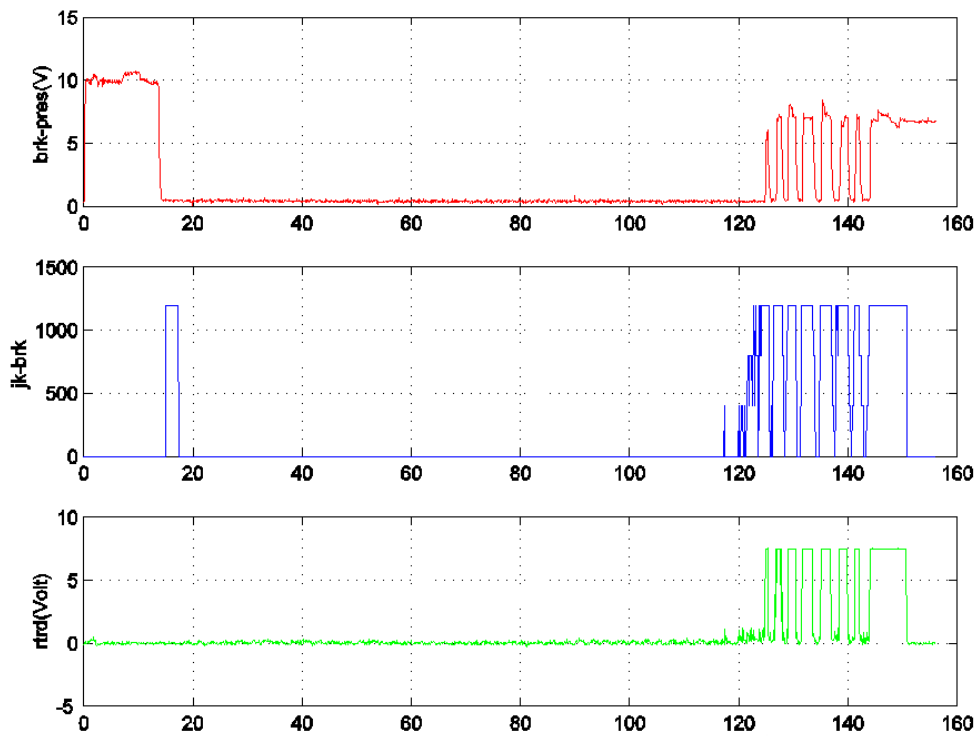
Half Loaded, Run 2: `max_spd=55, max_dec=0.9, EBS + Jake + trtd`



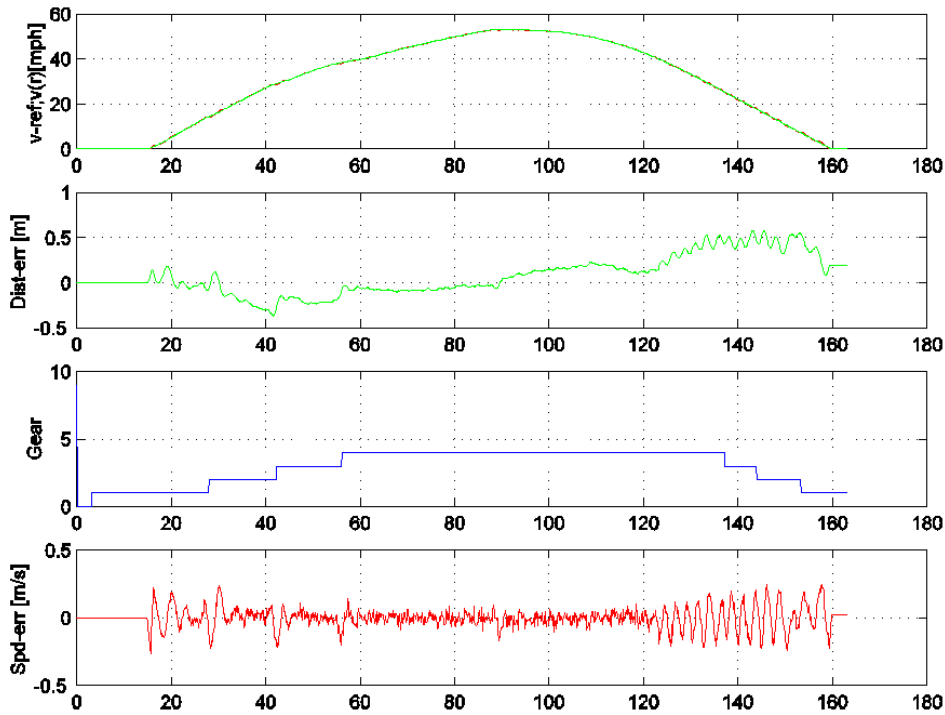
Half Loaded, Run 2: $\text{max_spd}=55$, $\text{max_dec}=0.9$, EBS + Jake + trtd



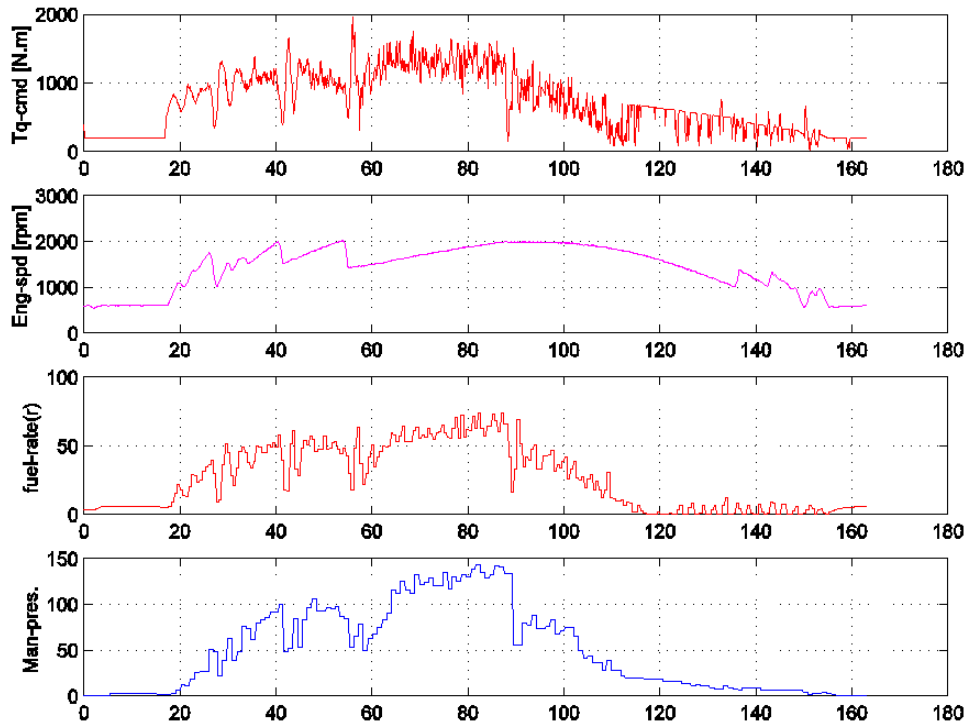
Half Loaded, Run 2: $\text{max_spd}=55$, $\text{max_dec}=0.9$, EBS + Jake + trtd



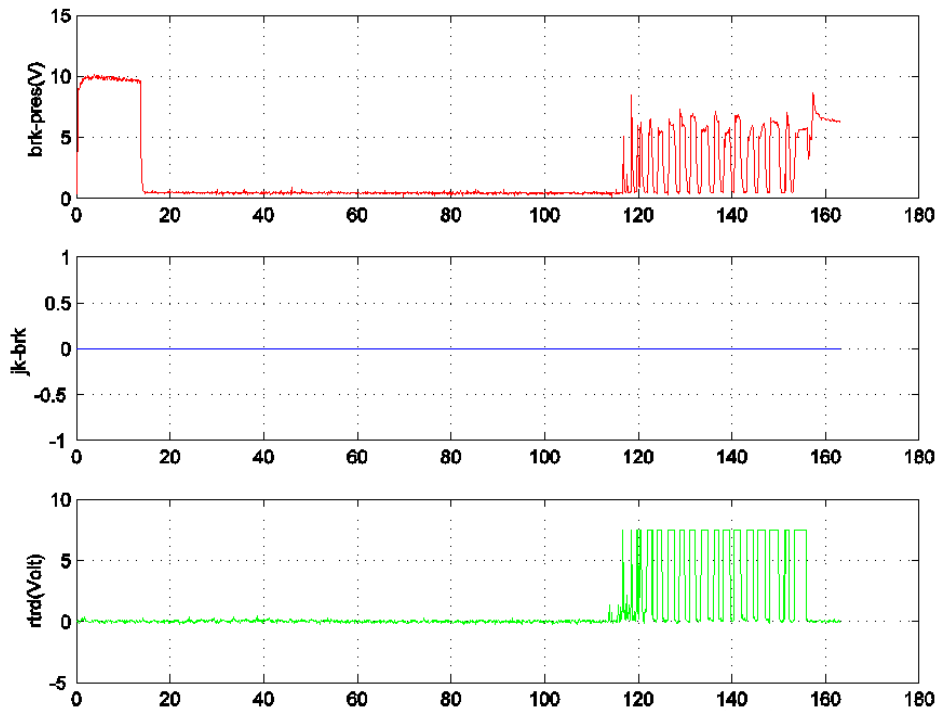
Half Loaded, Run 3: $\text{max_spd}=55$, $\text{max_dec}=0.5$, EBS + trtd



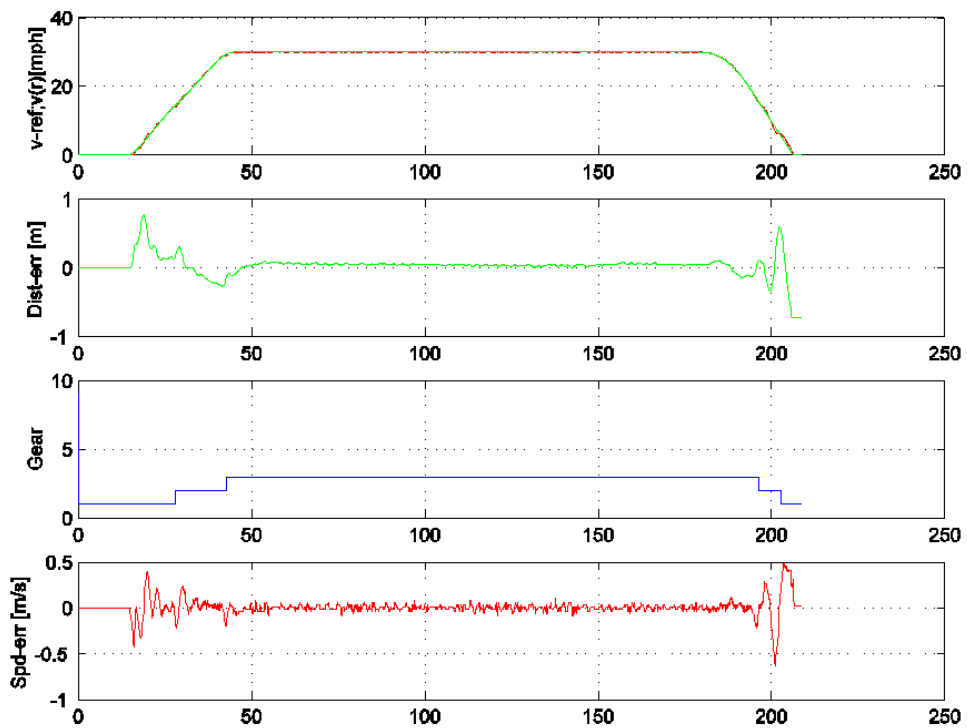
Half Loaded, Run 3: $\text{max_spd}=55$, $\text{max_dec}=0.5$, EBS + trtd



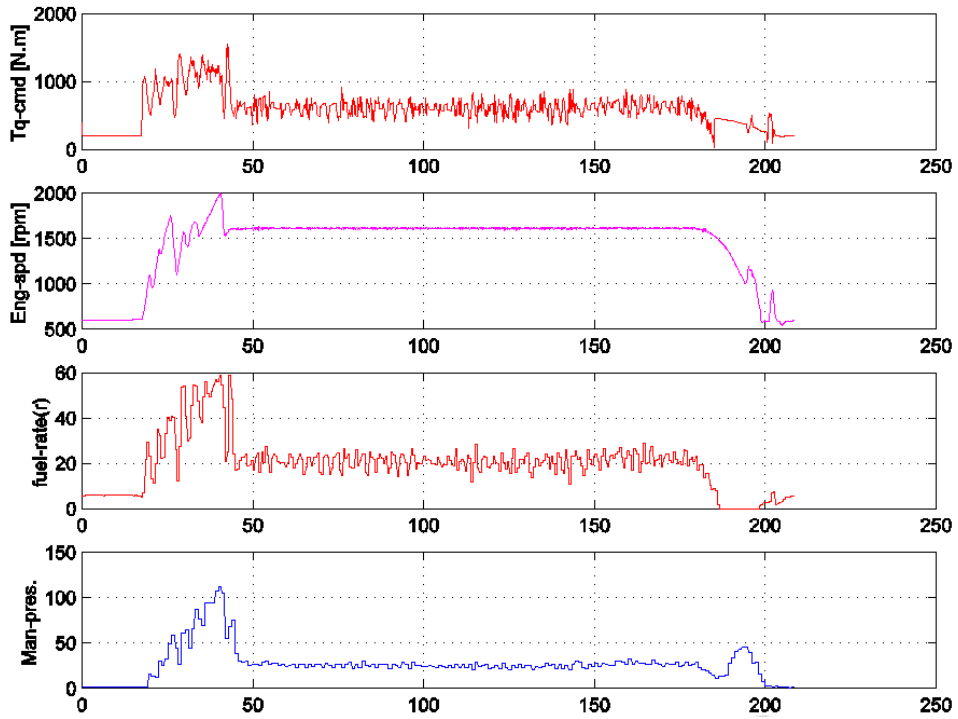
Half Loaded, Run 3: $\text{max_spd}=55$, $\text{max_dec}=0.5$, EBS + trtd



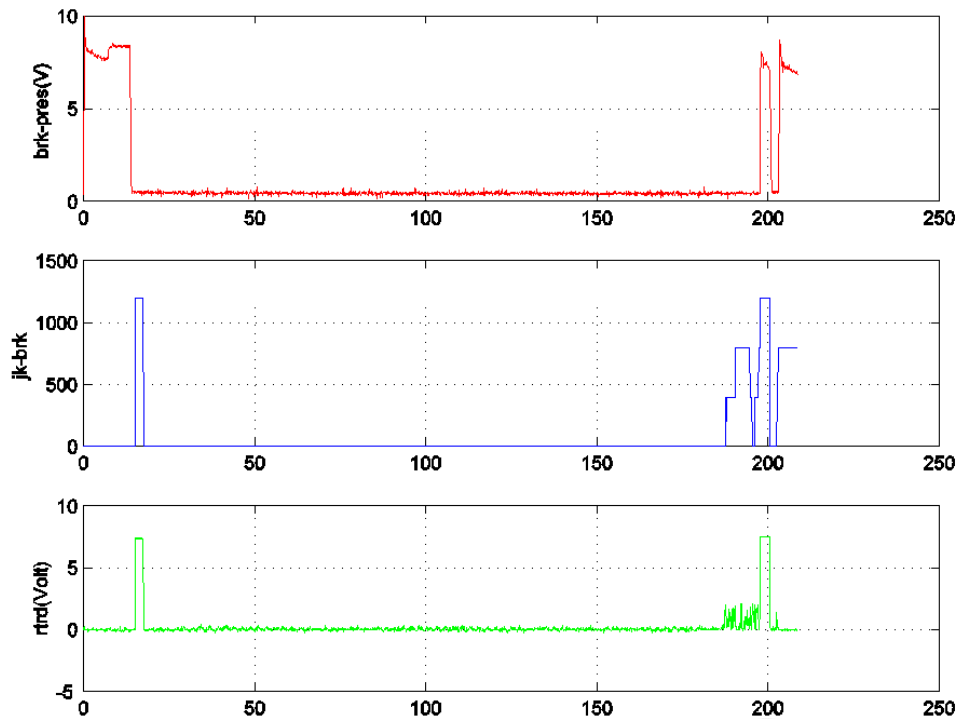
Half Loaded, Run 4: $\text{max_spd}=30$, $\text{max_dec}=0.7$, EBS + Jake + trtd



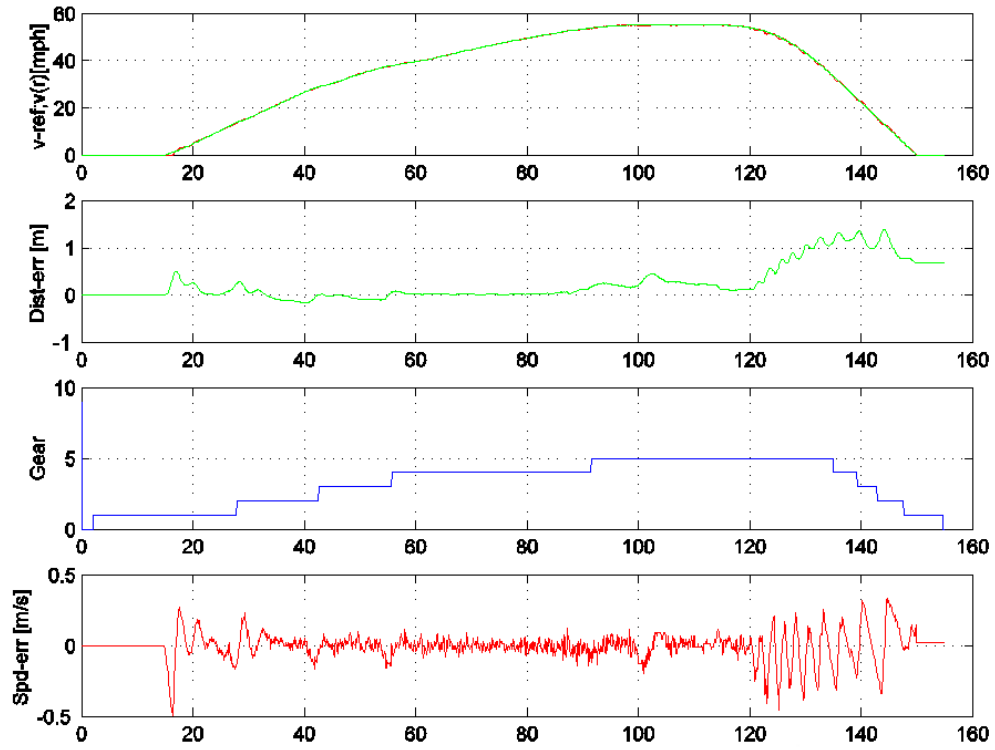
Half Loaded , Run 4: `max_spd=30`, `max_dec=0.7`, `EBS + Jake + trtd`



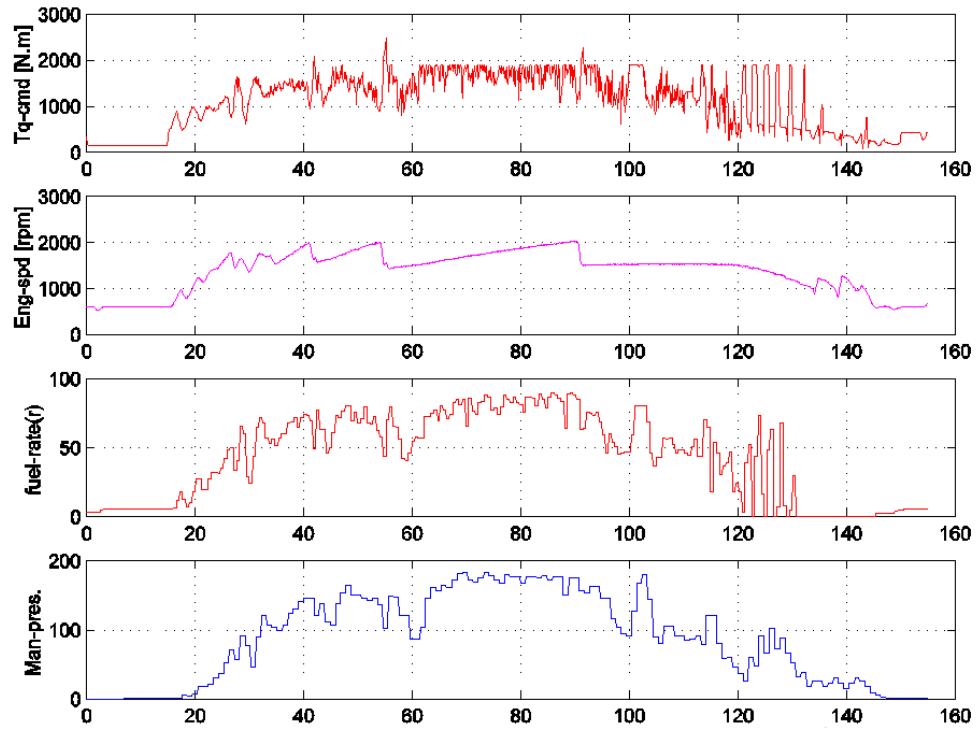
Half Loaded , Run 4: `max_spd=30`, `max_dec=0.7`, `EBS + Jake + trtd`



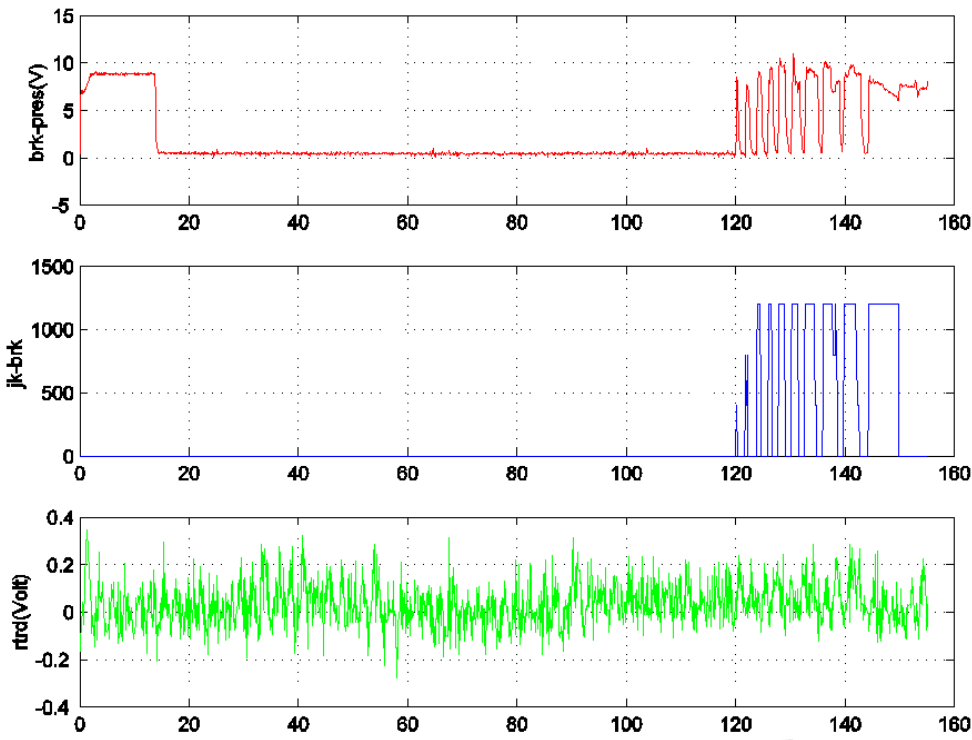
Fully Loaded, Run 1: max_spd=55, max_dec=1.0, EBS + Jake



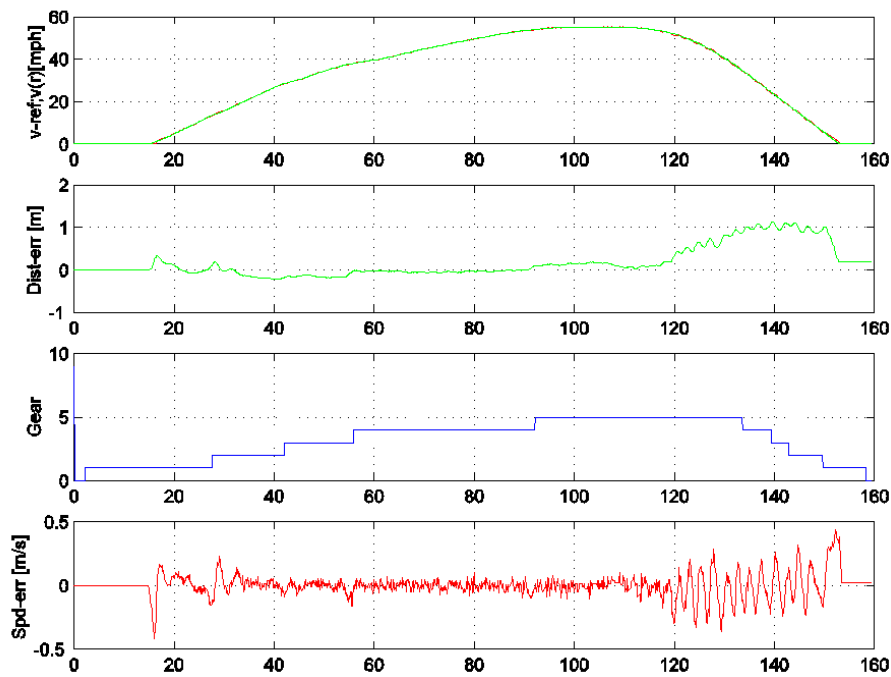
Fully Loaded, Run 1: max_spd=55, max_dec=1.0, EBS + Jake



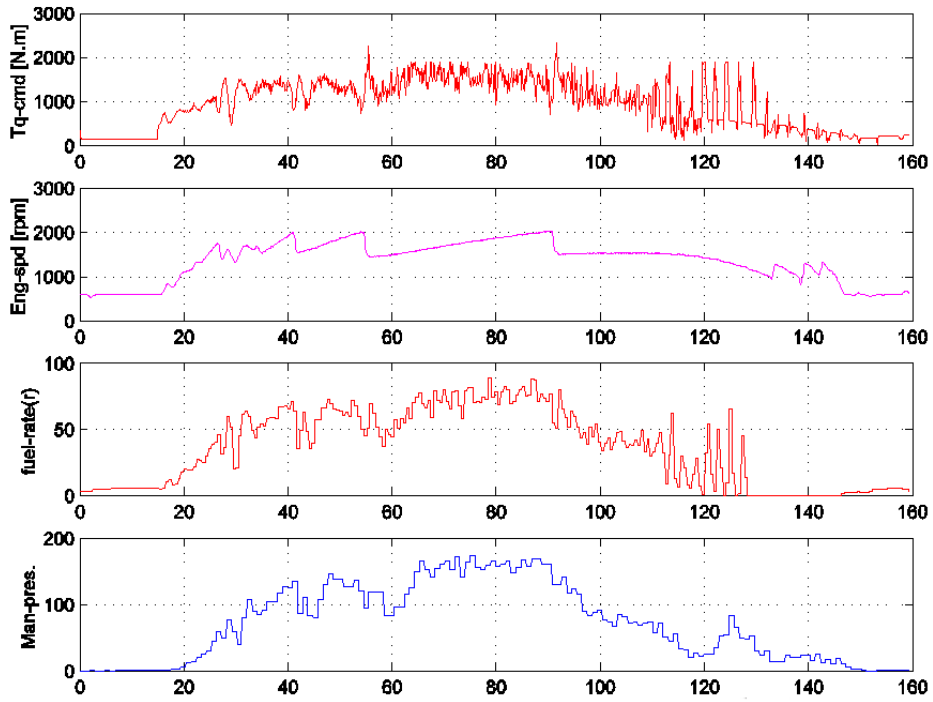
Fully Loaded, Run 1: max_spd=55, max_dec=1.0, EBS + Jake



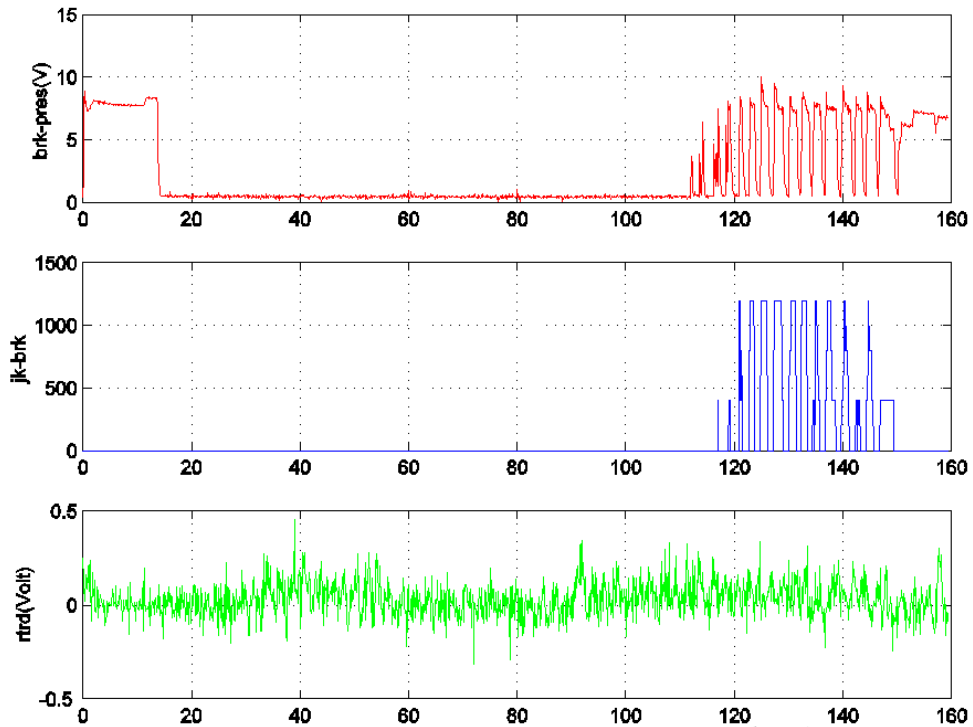
Fully Loaded, Run 2: max_spd=55, max_dec=0.8, EBS + Jake



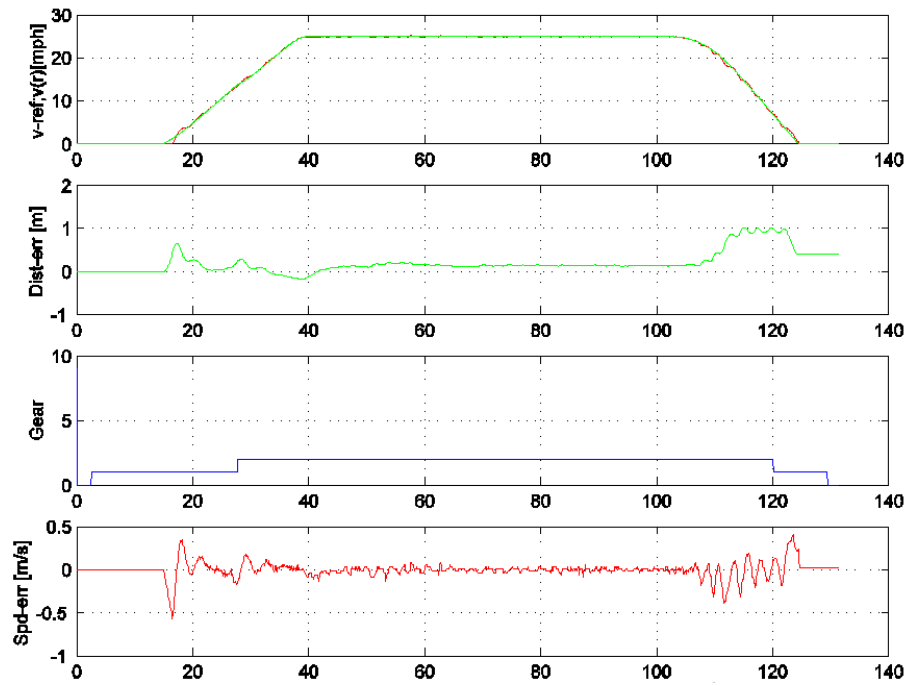
Fully Loaded, Run 2: `max_spd=55`, `max_dec=0.8`, **EBS + Jake**



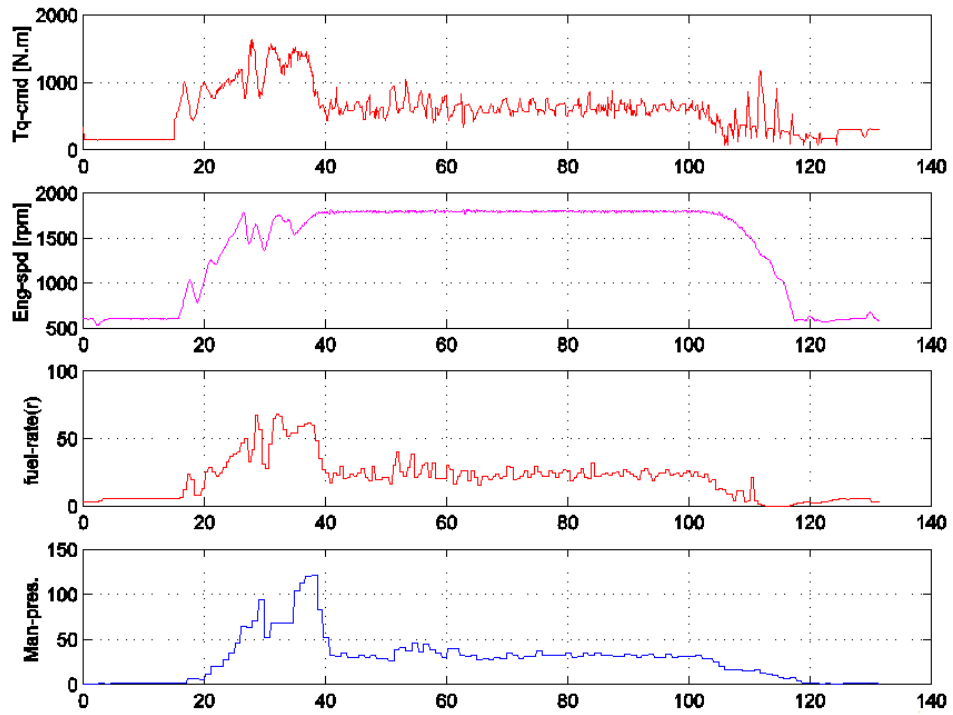
Fully Loaded, Run 2: `max_spd=55`, `max_dec=0.8`, **EBS + Jake**



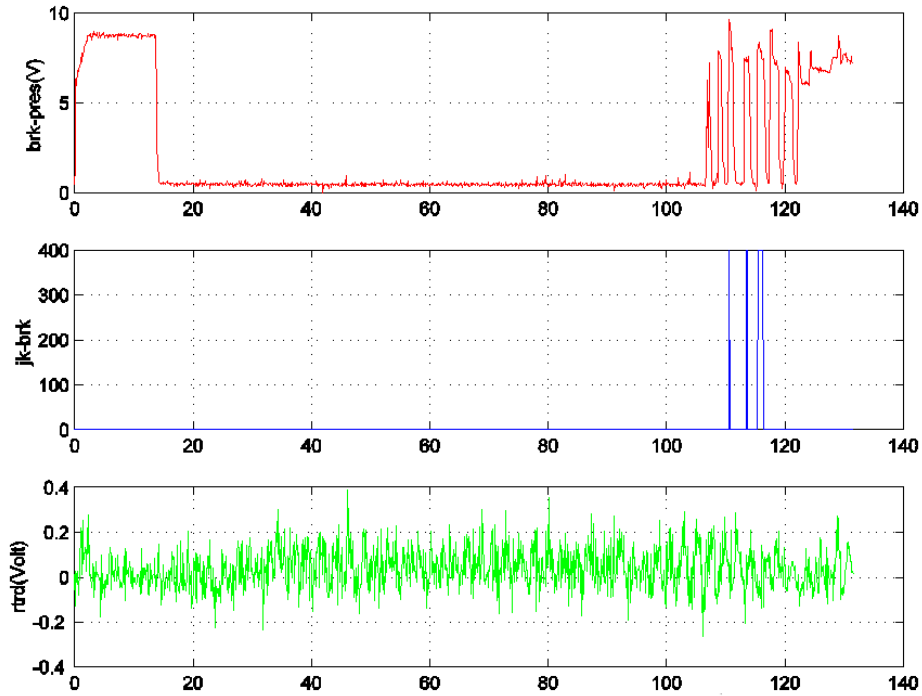
Fully Loaded , Run 3: `max_spd=25,` `max_dec=0.7,` `EBS + Jake`



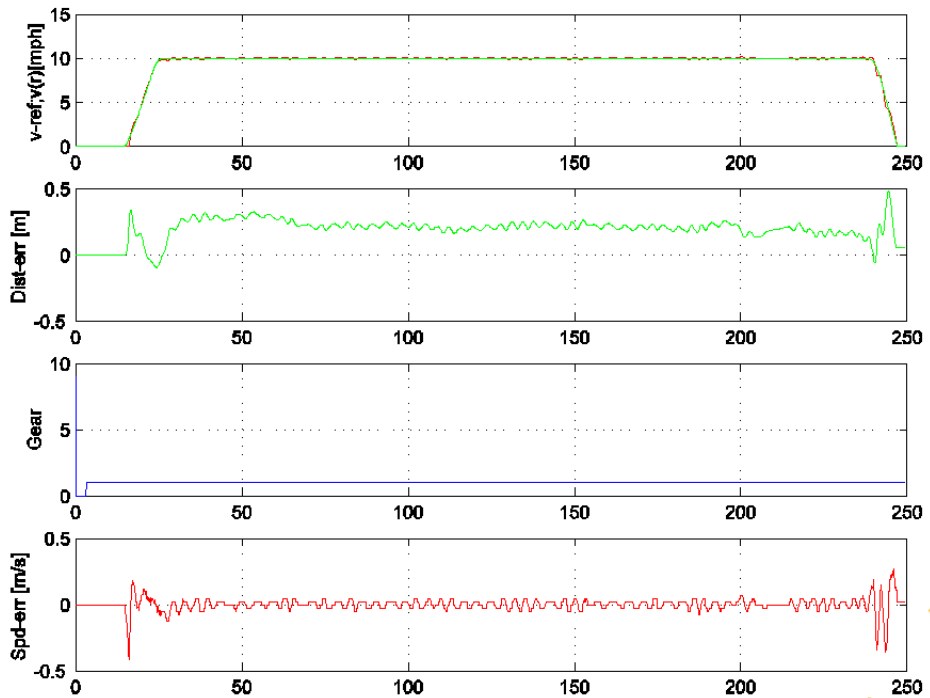
Fully Loaded , Run 3: `max_spd=25,` `max_dec=0.7,` `EBS + Jake`



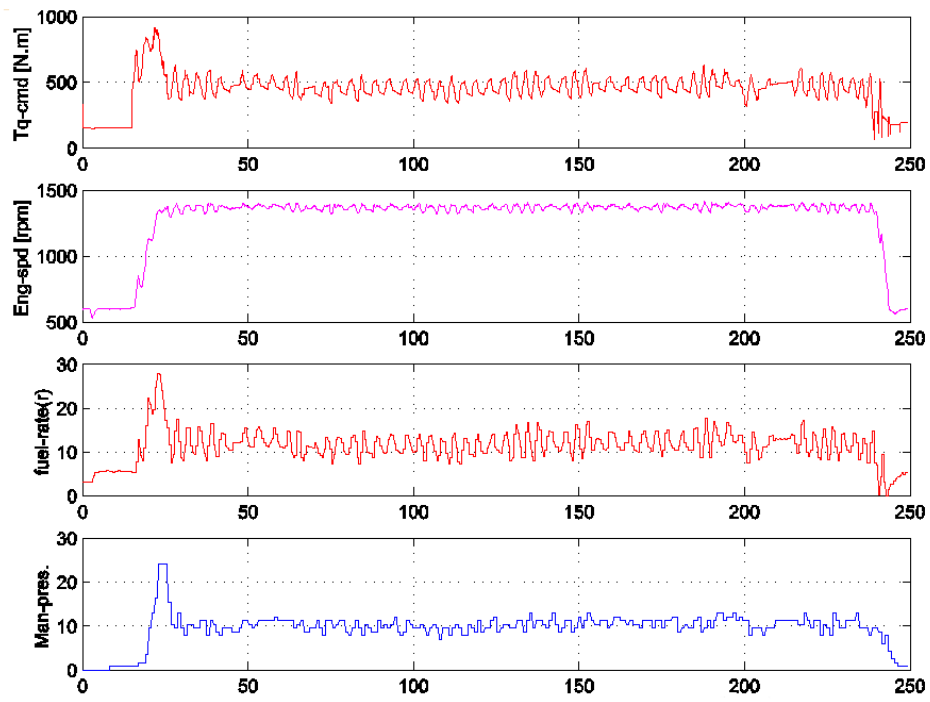
Fully Loaded , Run 3: max_spd=25, max_dec=0.7, EBS + Jake



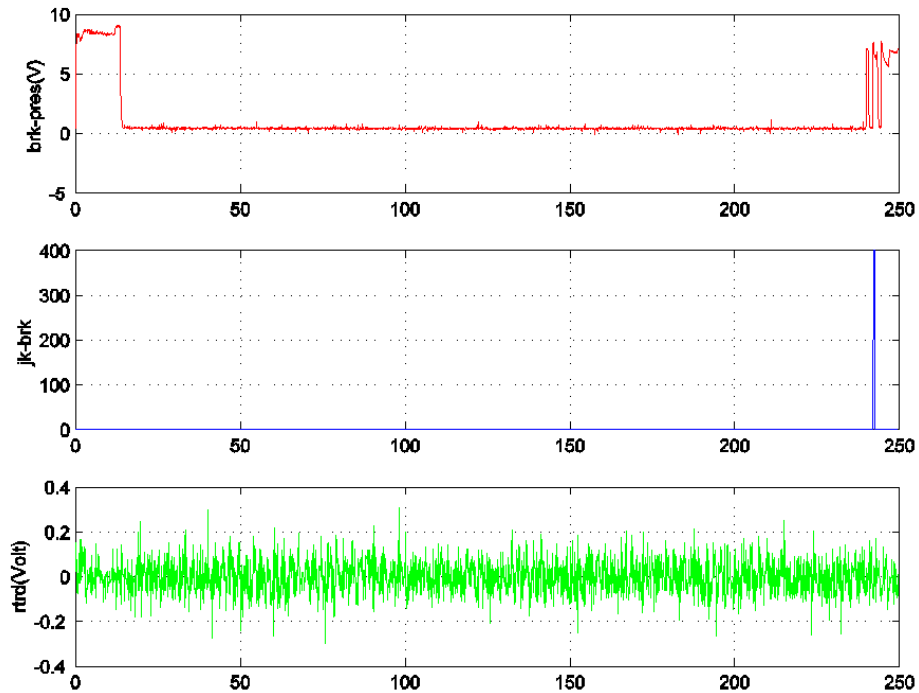
Fully Loaded , Run 4: max_spd=10, max_dec=0.7, EBS + Jake



Fully Loaded , Run 4: max_spd=10, max_dec=0.7, EBS + Jake

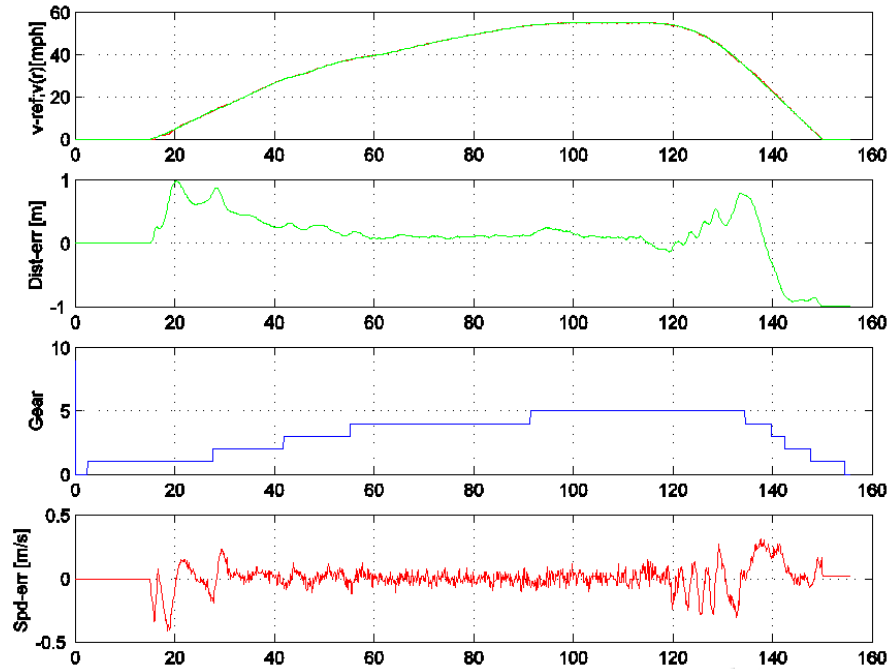


Fully Loaded, Run 4: max_spd=10, max_dec=0.7, EBS + Jake

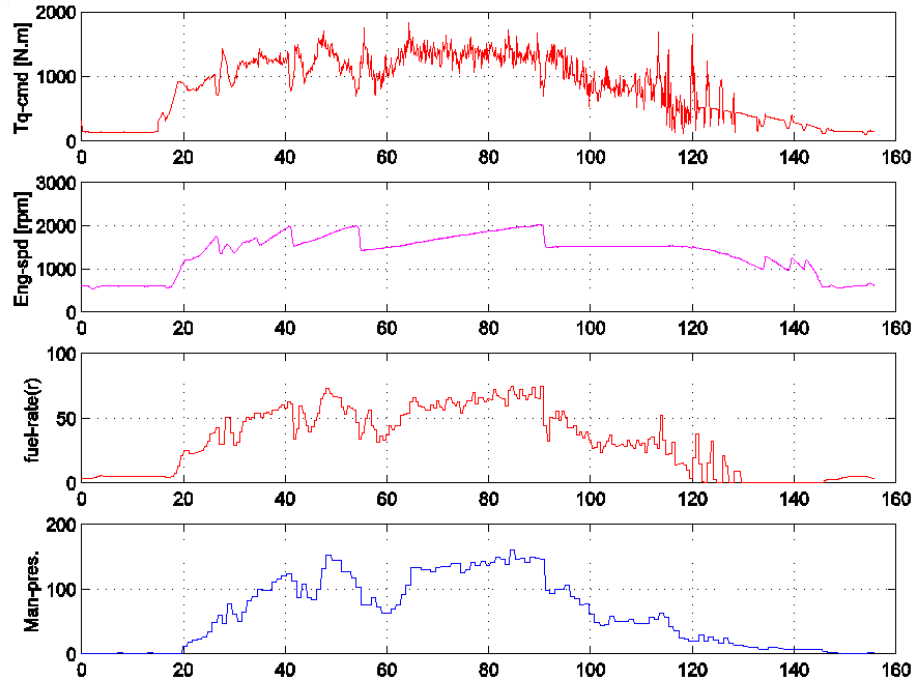


The next two runs show the robustness of the controller performance to imperfect knowledge of vehicle mass. During these tests, the control system parameters were defined based on an assumed empty truck, but in fact the truck was fully loaded.

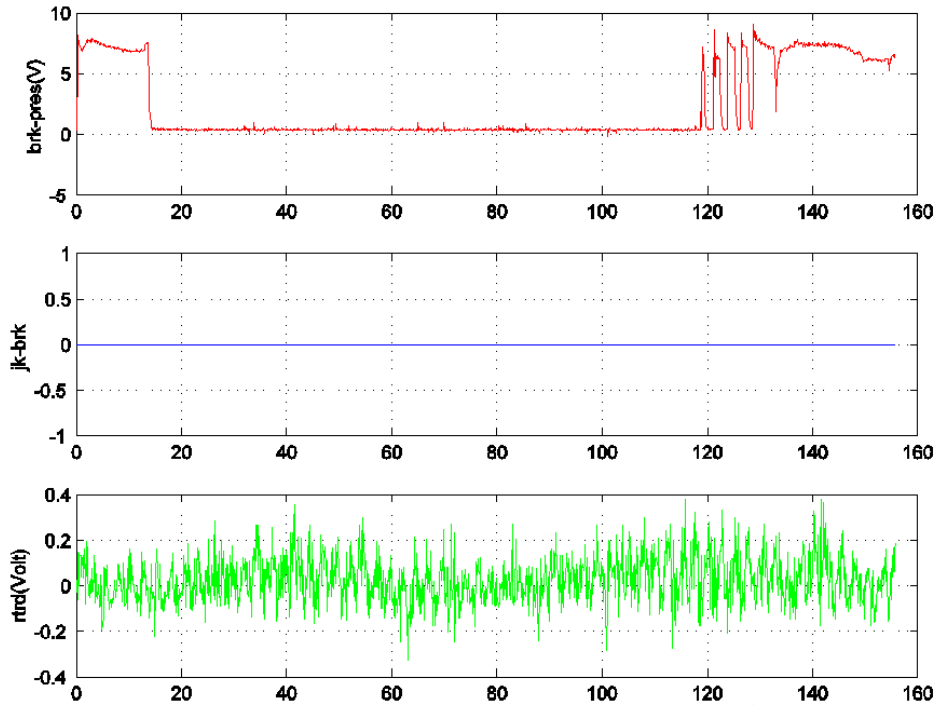
Fully Loaded, Run 5: $\text{max_spd}=55$, $\text{max_dec}=1.0$, EBS only
Empty trailer Mass is used in feedback control



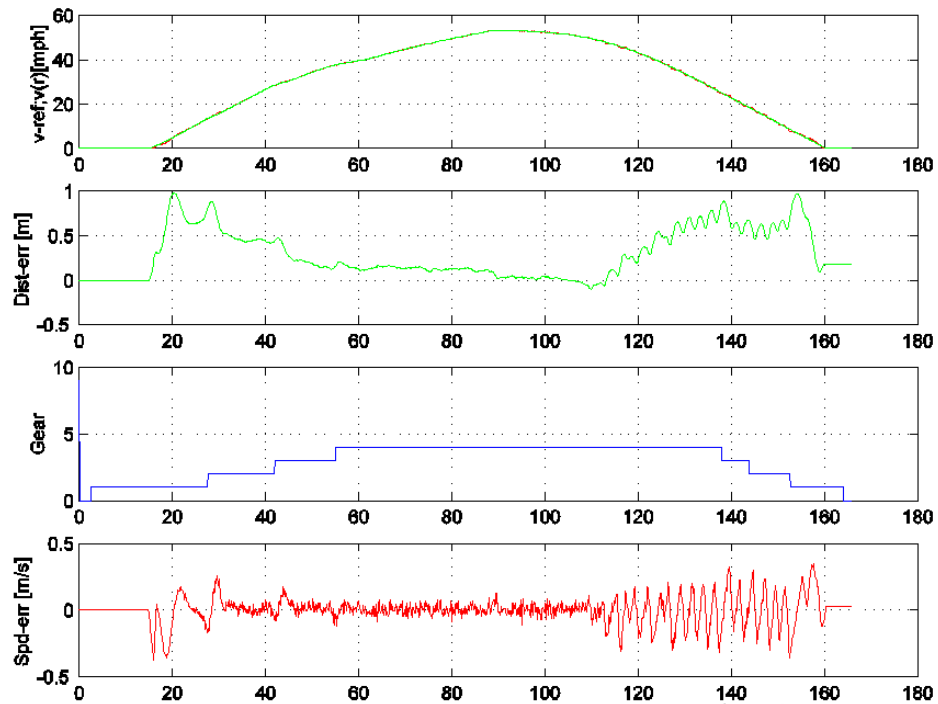
Fully Loaded, Run 5: `max_spd=55`, `max_dec=1.0`, **EBS only**
Empty trailer Mass is used in feedback control



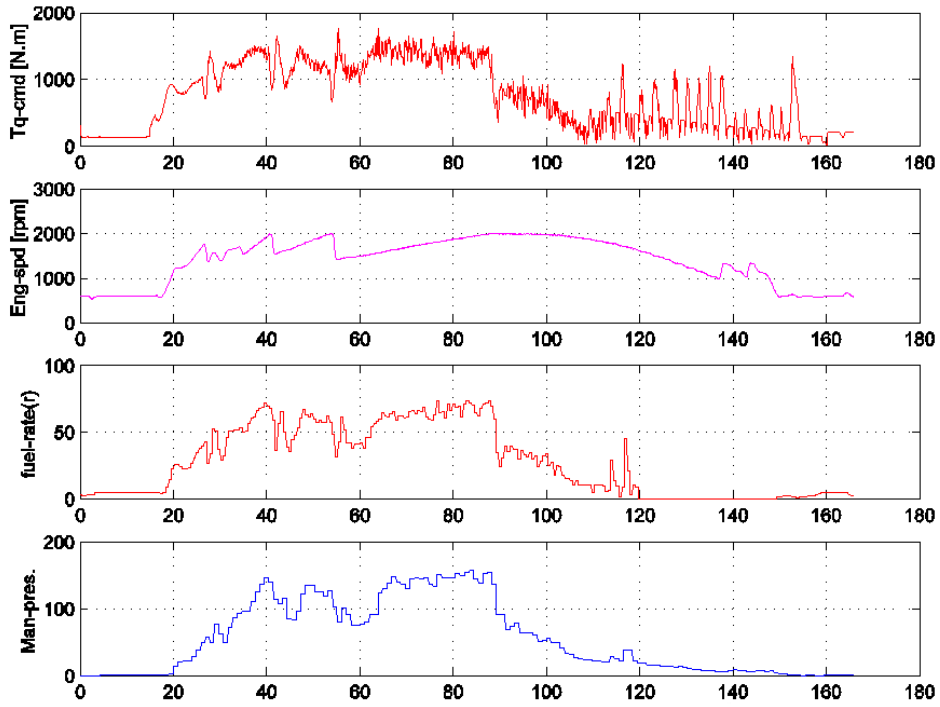
Fully Loaded, Run 5: $\text{max_spd}=55$, $\text{max_dec}=1.0$, EBS only
Empty trailer Mass is used in feedback control



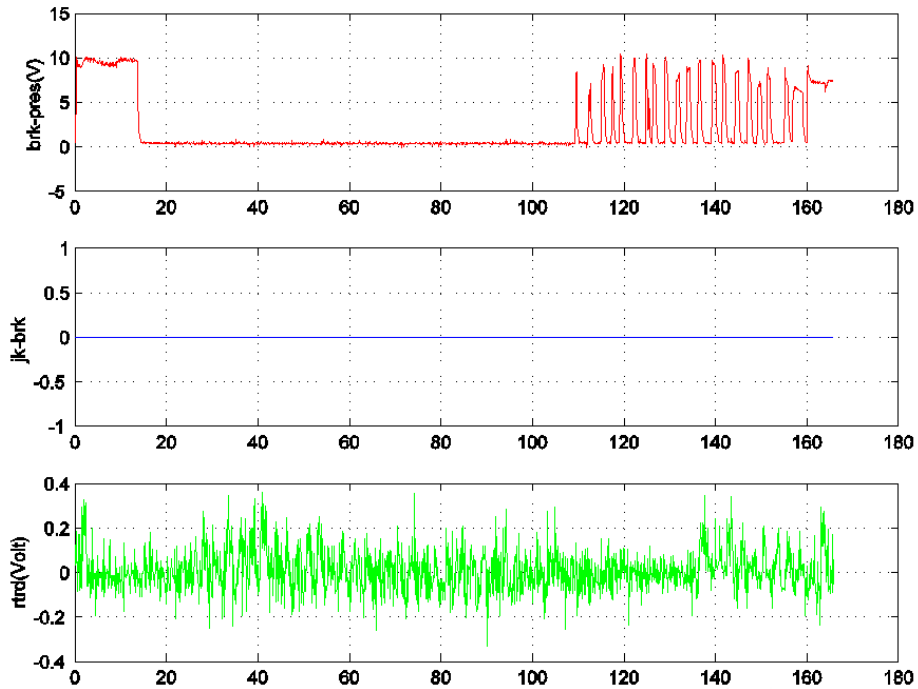
Fully loaded, Run 6: $\text{max_spd}=55$, $\text{max_dec}=0.5$, EBS only
Empty trailer Mass is used in feedback control



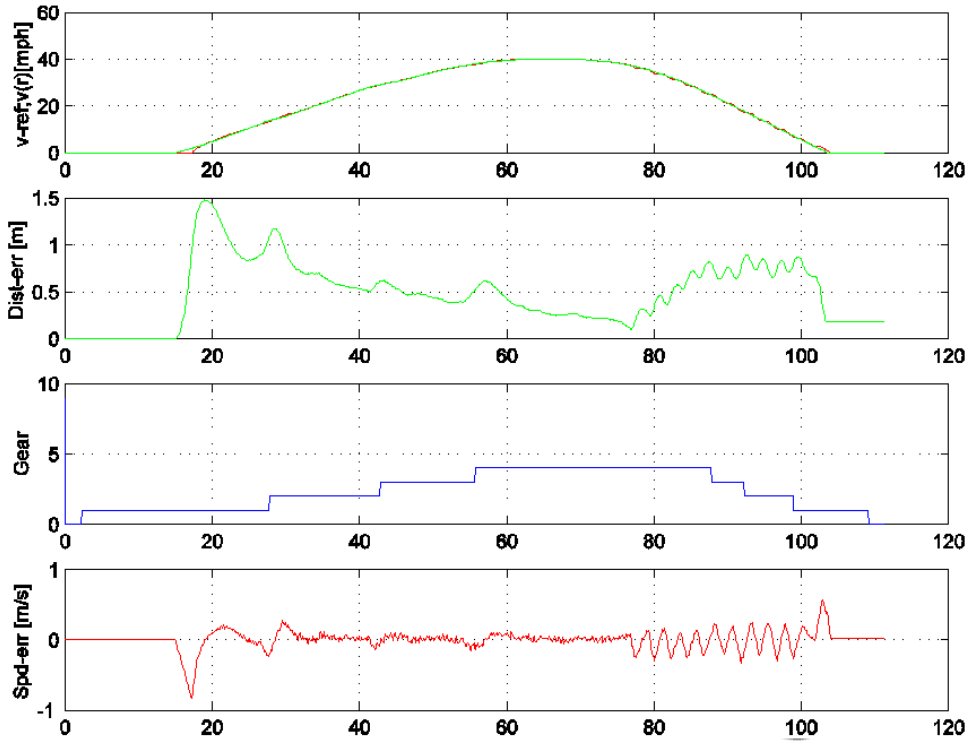
Fully Loaded, Run 6: $\text{max_spd}=55$, $\text{max_dec}=0.5$, **EBS only**
Empty trailer Mass is used in feedback control



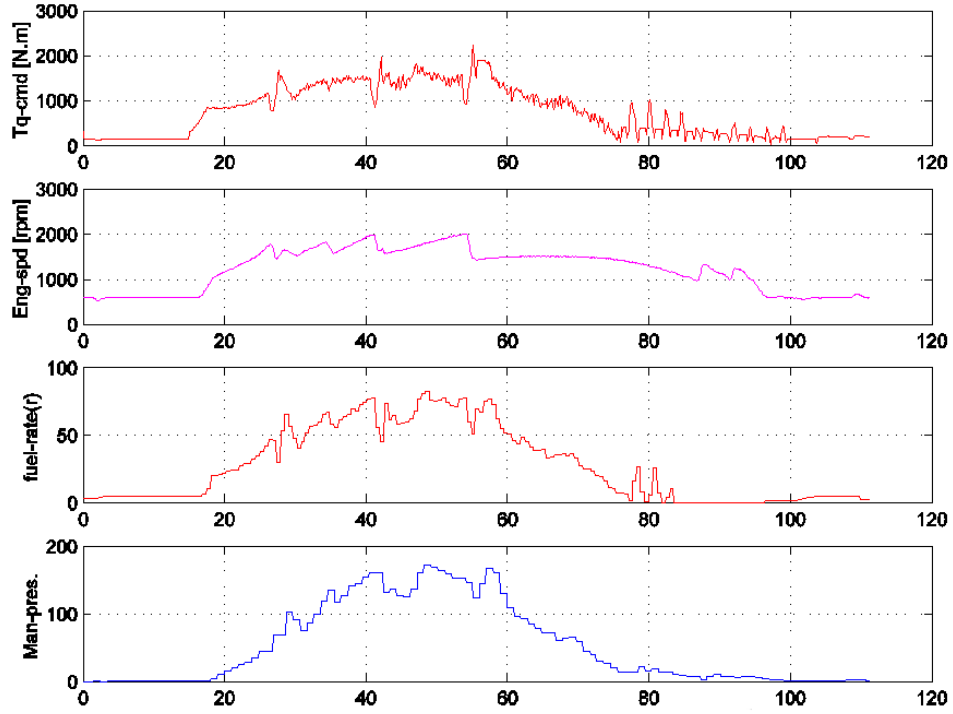
Fully Loaded, Run 6: $\text{max_spd}=55$, $\text{max_dec}=0.5$, **EBS + Jake + trtd**
Empty trailer Mass is used in feedback control



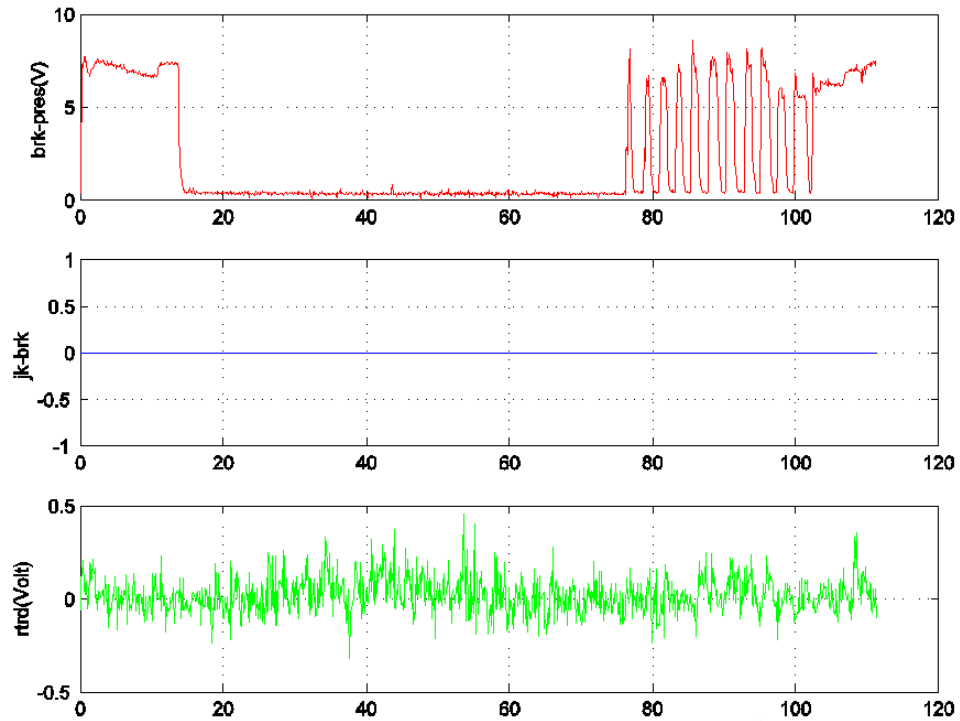
Fully Loaded, Run 7: $\text{max_spd}=40$, $\text{max_dec}=0.7$, EBS only
Empty trailer Mass is used in feedback control



Fully Loaded, Run 7: $\text{max_spd}=40$, $\text{max_dec}=0.7$, EBS only
Empty trailer Mass is used in feedback control



Fully Loaded, Run 7: $\text{max_spd}=40$, $\text{max_dec}=0.7$, EBS only
Empty trailer Mass is used in feedback control

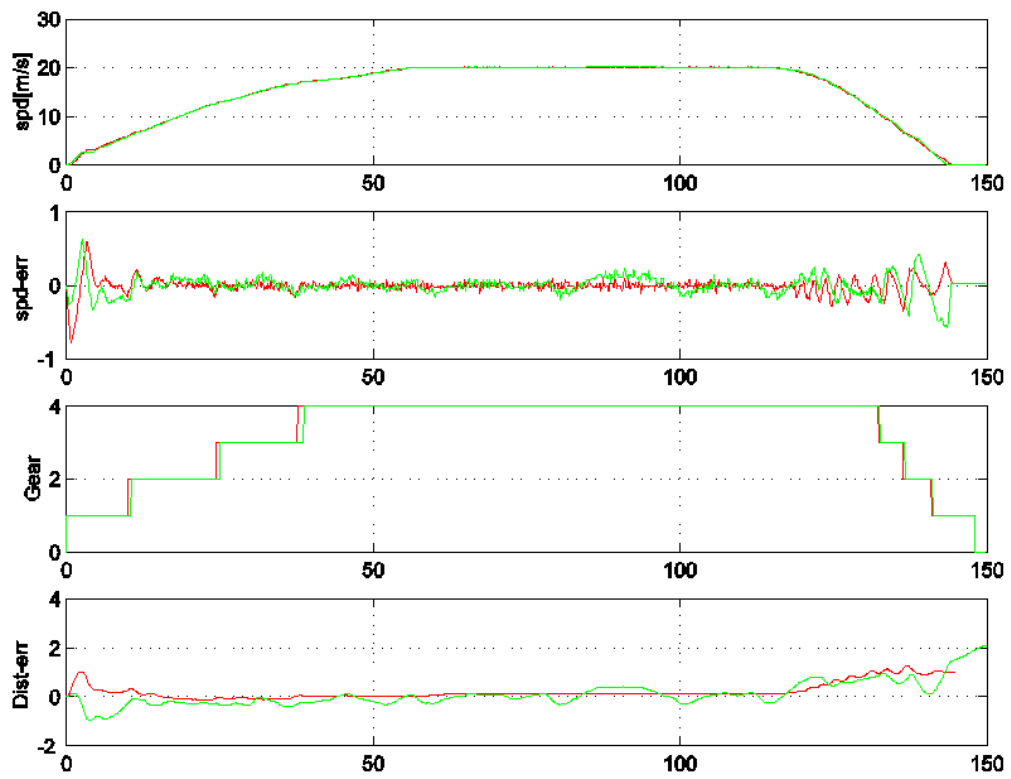


Appendix C

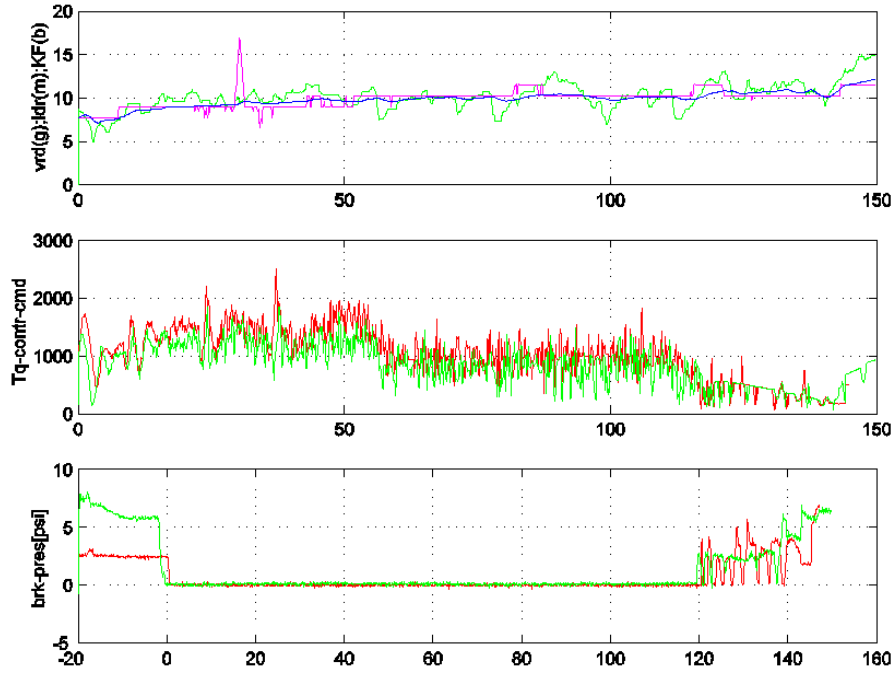
Example Experimental Results of Two-Truck Platoon Tests

This appendix includes a collection of representative test data from the tests that were performed for two-truck platoon following at Crows Landing. These tests included variations in maximum speed (55, 50 and 45 mph) and in the nominal separation between the trucks (10, 8, 6, 4, and 3 m).

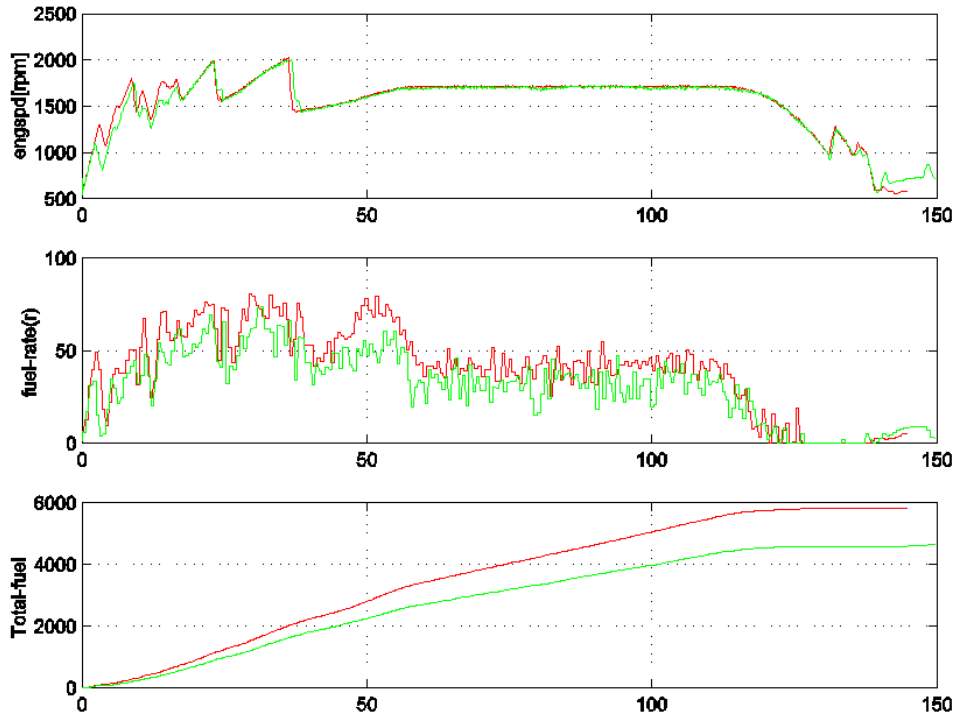
Run 1: Max speed **45 [mph]** ; Des_dist: 10 [m]



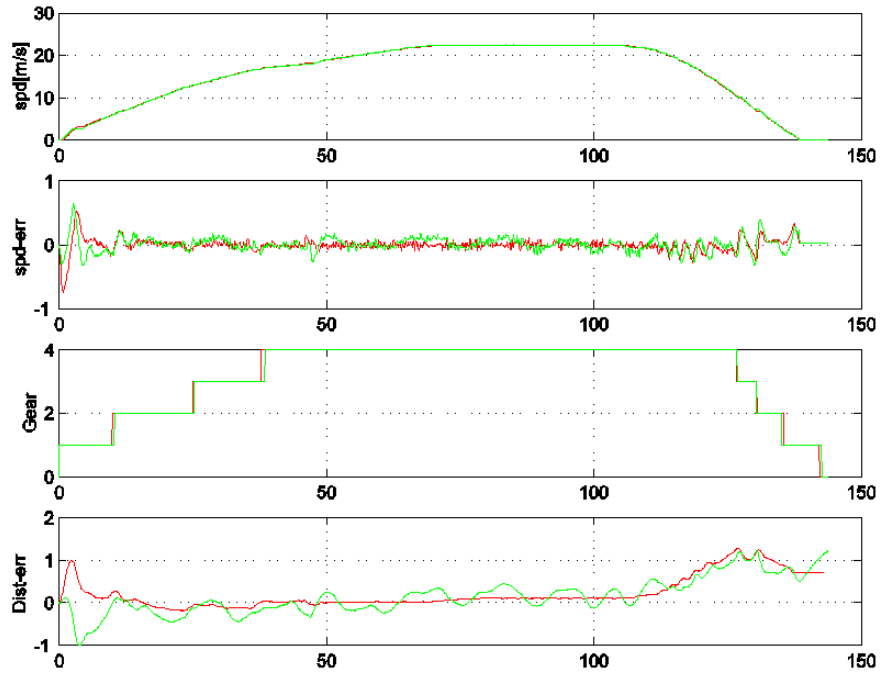
Run 1: Max speed 45 [mph] ; Des_dist: 10 [m]



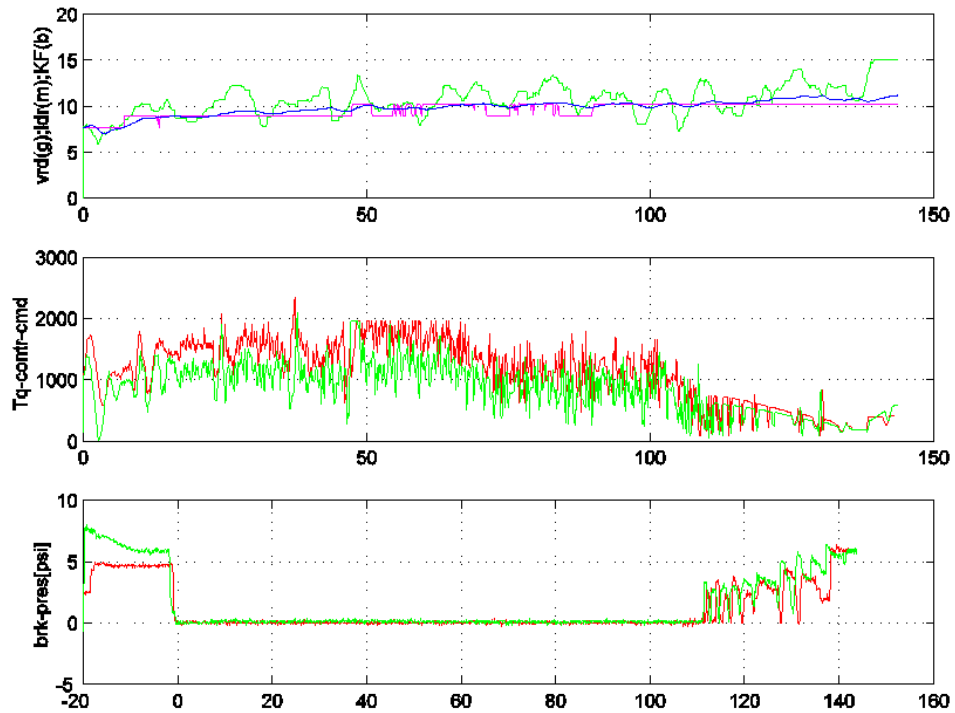
Run 1: Max speed 45 [mph] ; Des_dist: 10 [m]



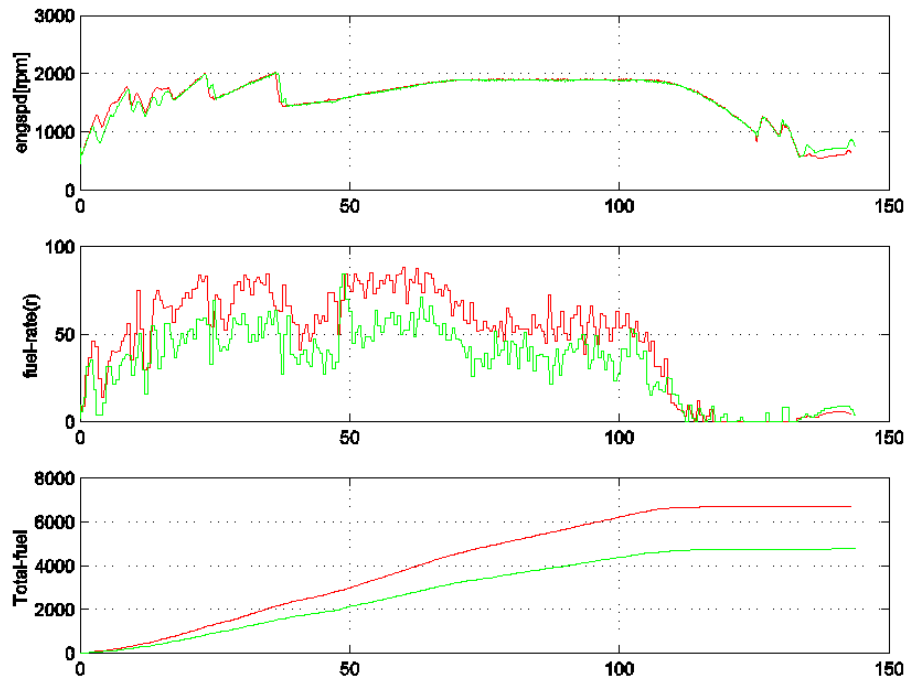
Run 2: Max speed 50 [mph] ; Des_dist: 10 [m]



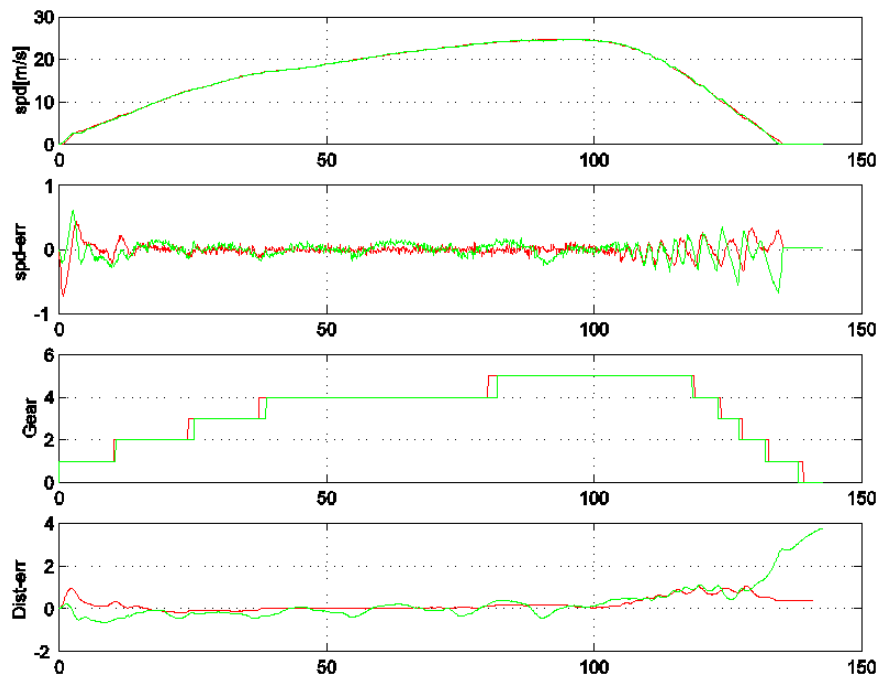
Run 2: Max speed 50 [mph] ; Des_dist: 10 [m]



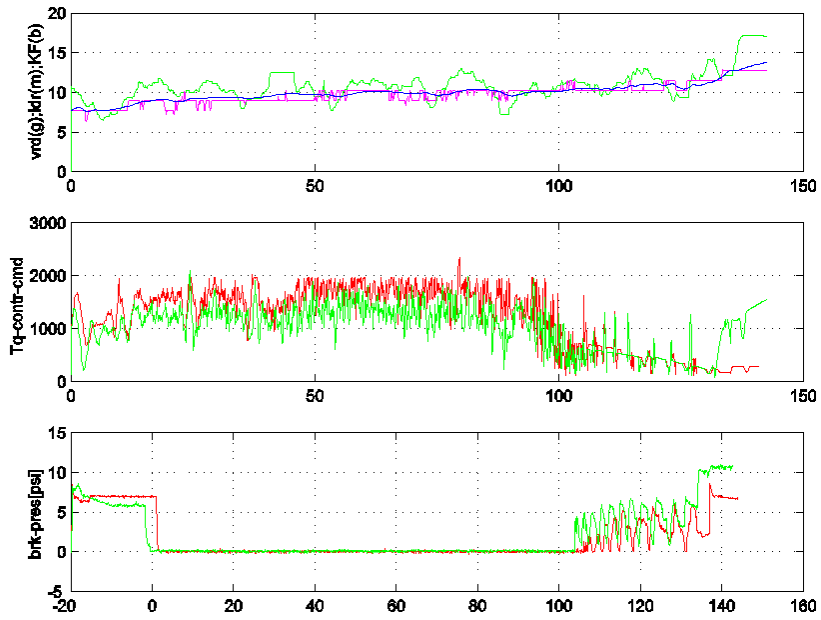
Run 2: Max speed 50 [mph] ; Des_dist: 10 [m]



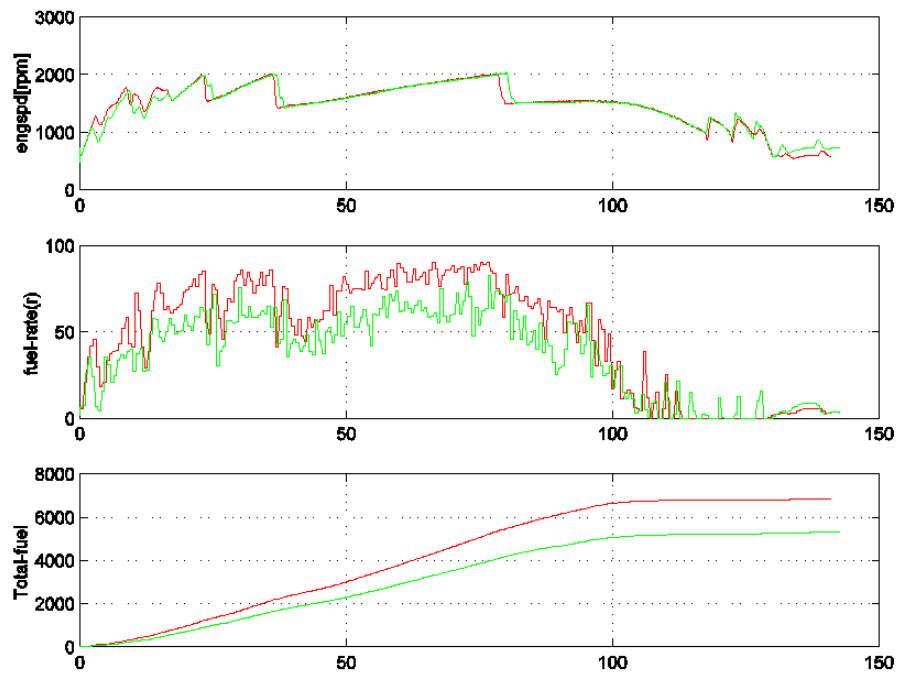
Run 3: Max speed 55 [mph] ; Des_dist: 10 [m]



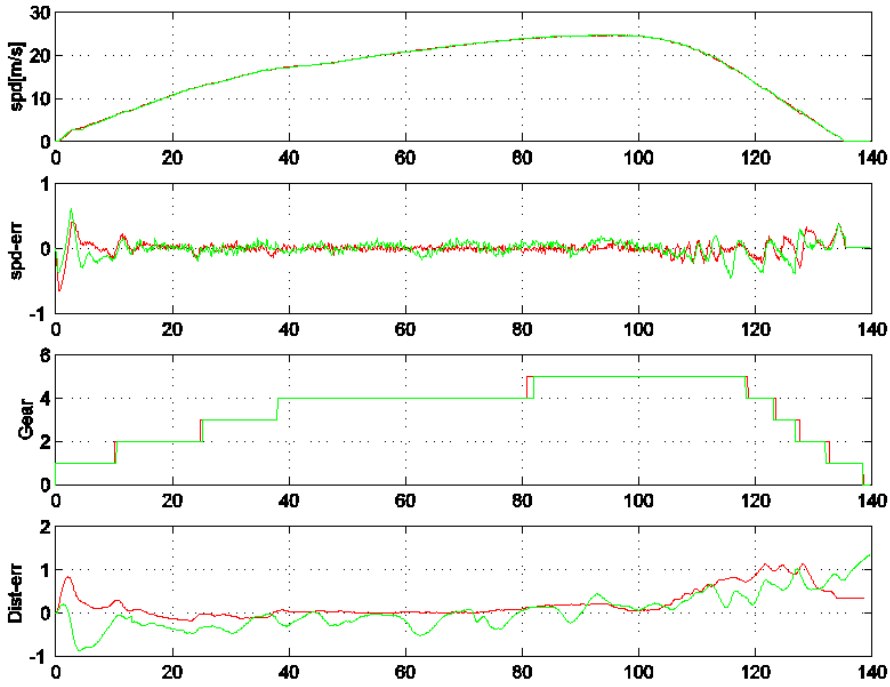
Run 3: Max speed 55 [mph] ; Des_dist: 10 [m]



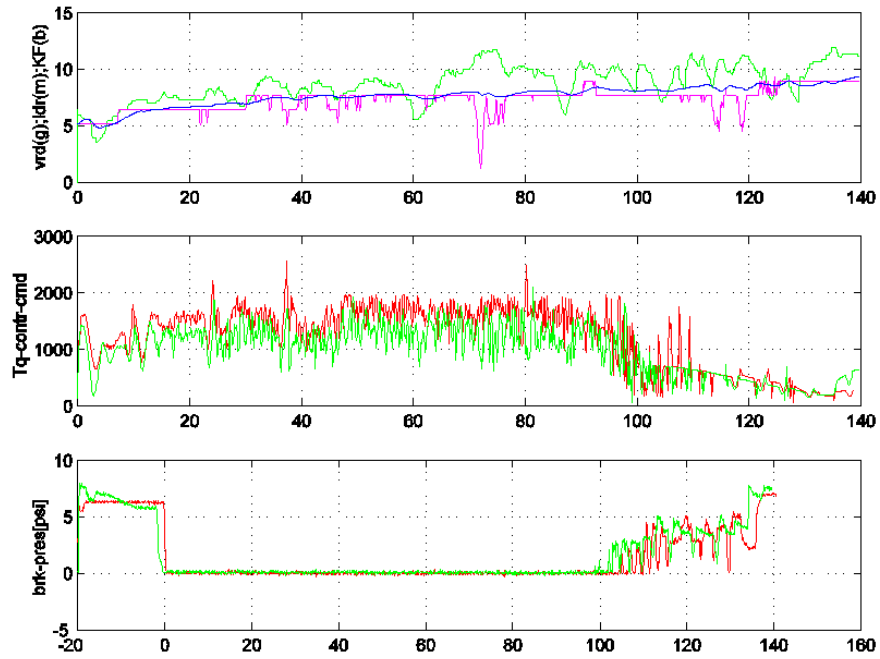
Run 3: Max speed 55 [mph] ; Des_dist: 10 [m]



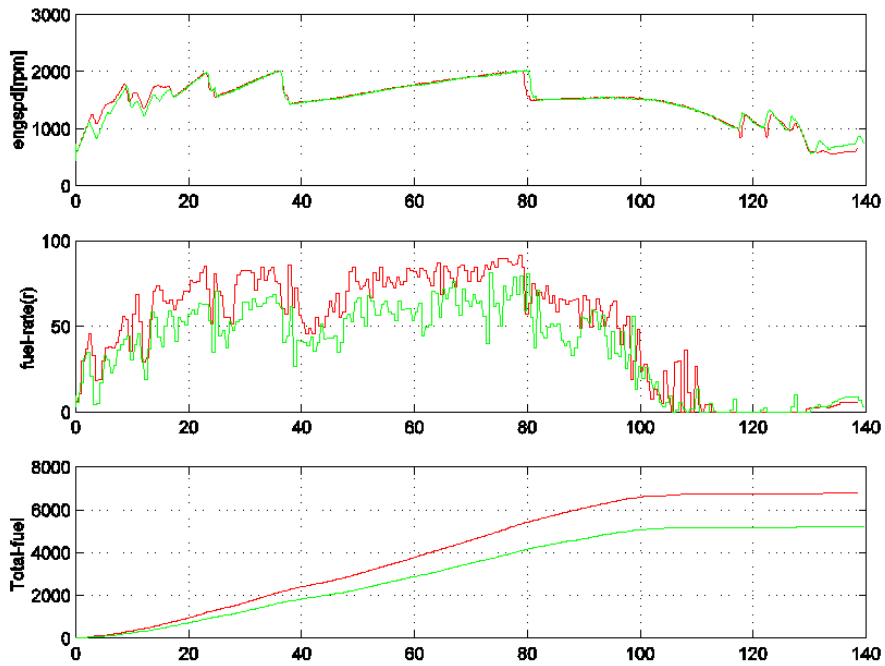
Run 4: Max speed 55 [mph] ; Des_dist: 8 [m]



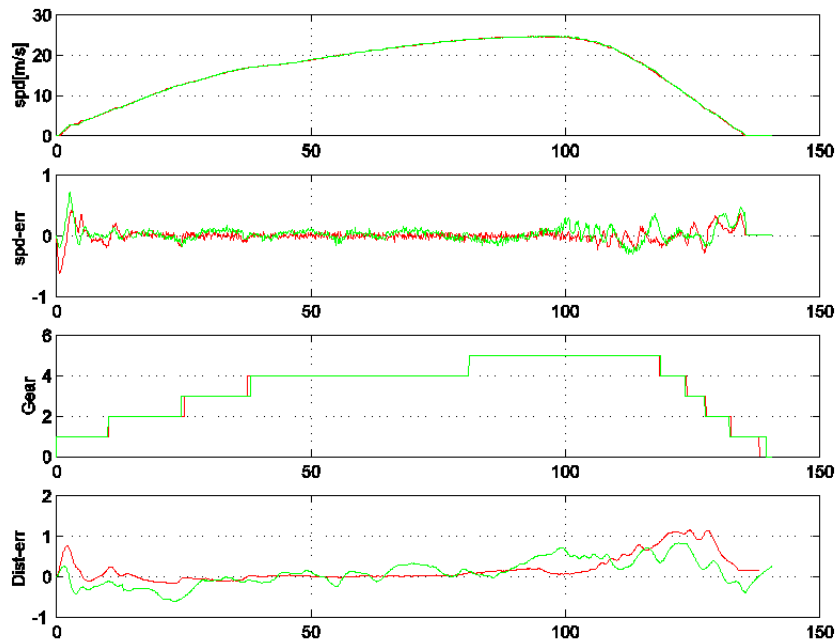
Run 4: Max speed 55 [mph] ; Des_dist: 8 [m]



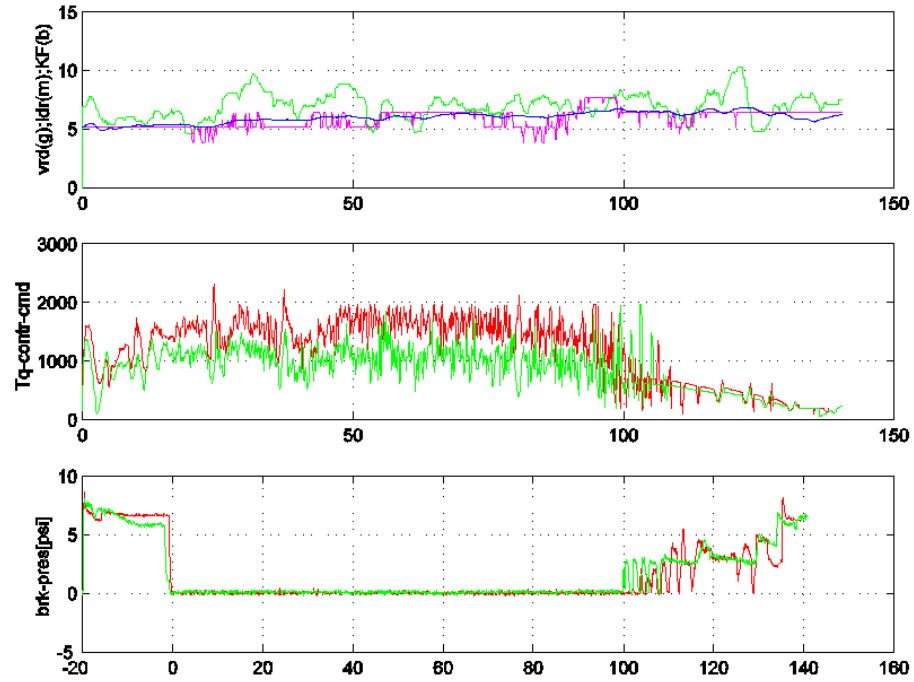
Run 4: Max speed 55 [mph] ; Des_dist: 8 [m]



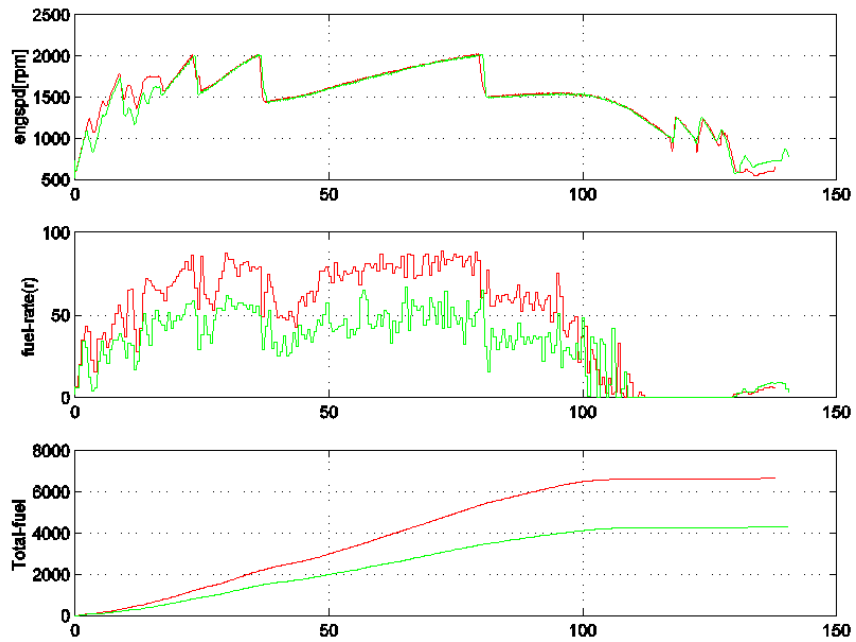
Run 5: Max speed 55 [mph] ; Des_dist: 6 [m]



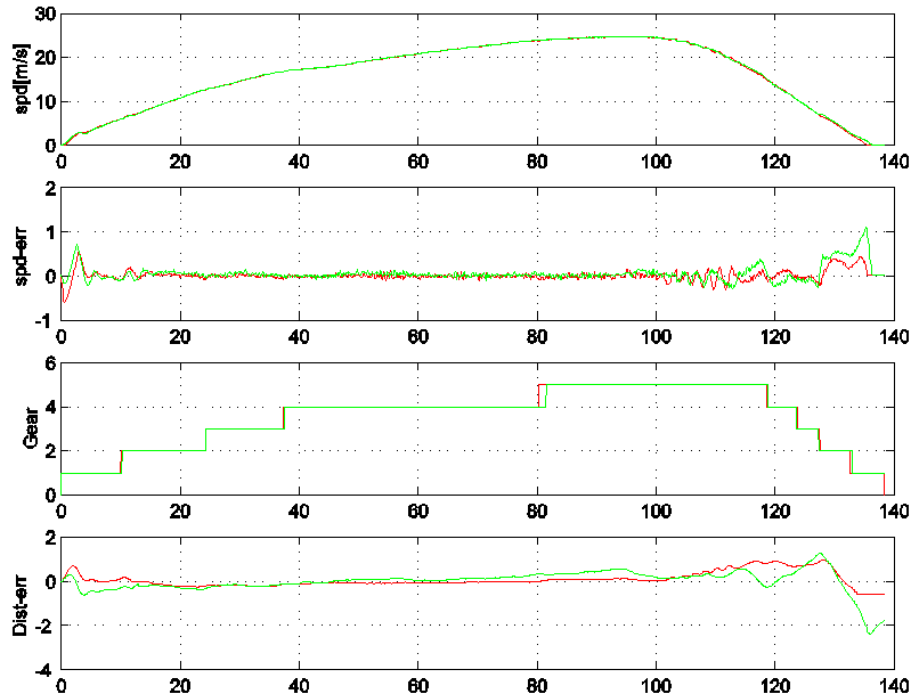
Run 5: Max speed 55 [mph] ; Des_dist: 6 [m]



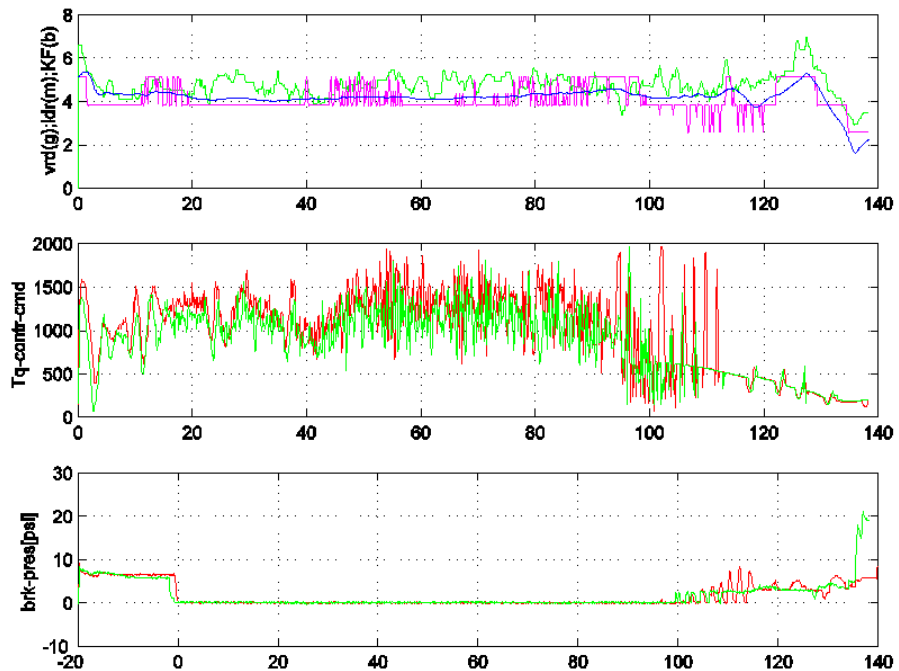
Run 5: Max speed 55 [mph] ; Des_dist: 6 [m]



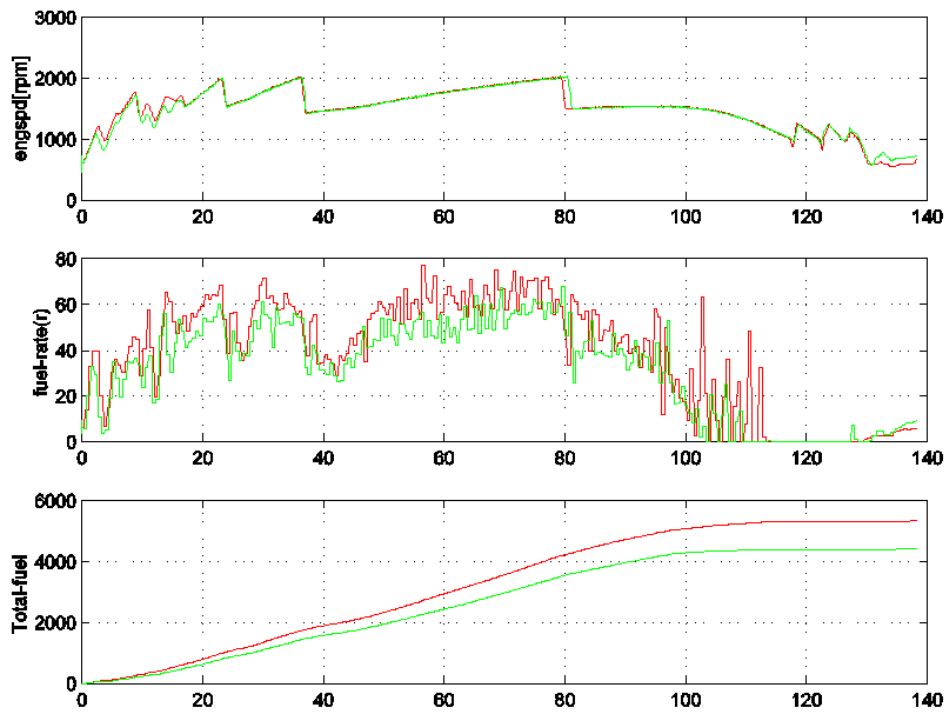
Run 6: Max speed 55 [mph] ; Des_dist: 4 [m]



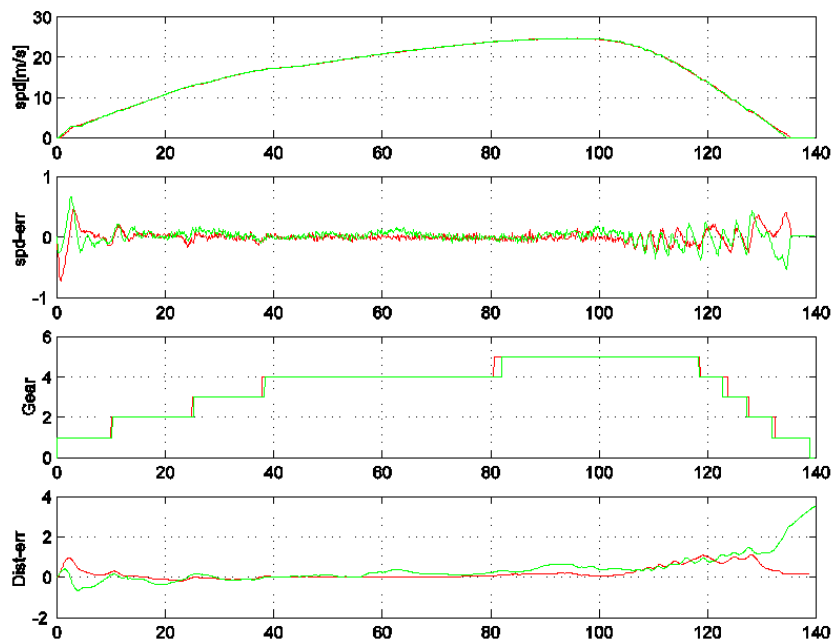
Run 6: Max speed 55 [mph] ; Des_dist: 4 [m]



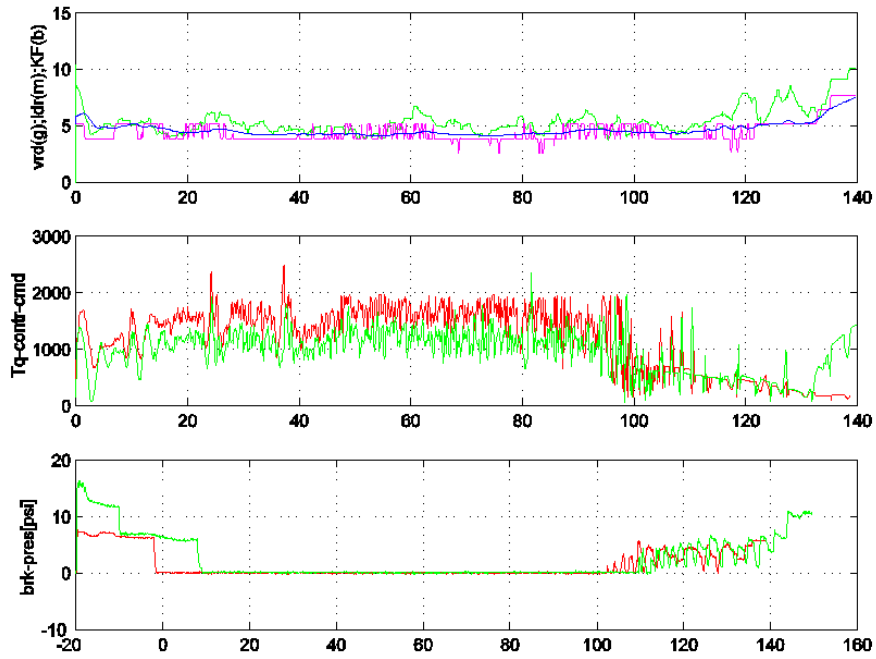
Run 6: Max speed 55 [mph] ; Des_dist: 4 [m]



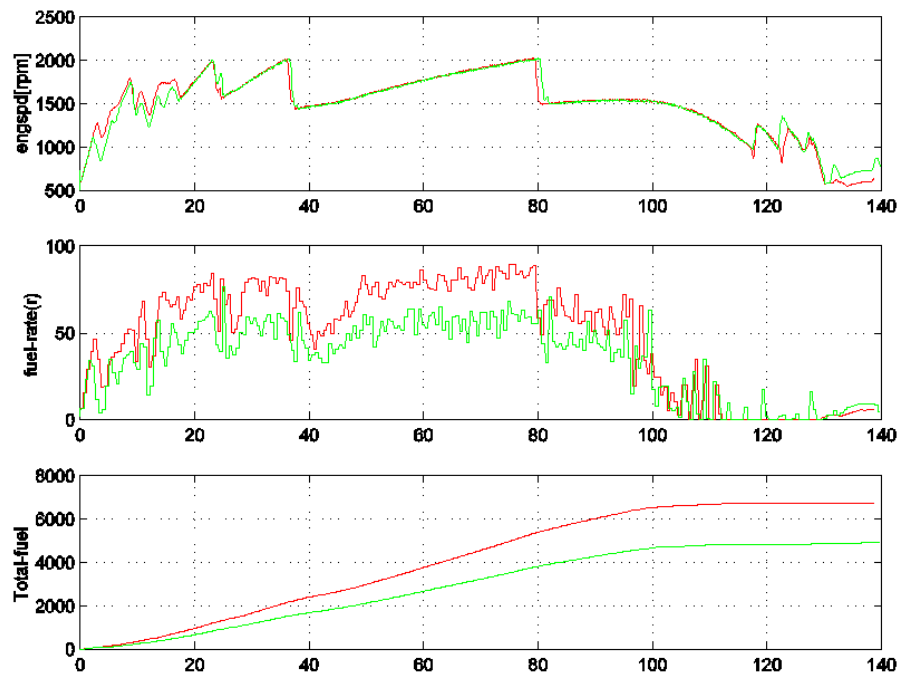
Run 7: Max speed 55 [mph] ; Des_dist: 4 [m]; Different direction



Run 7: Max speed 55 [mph] ; Des_dist: 4 [m]; Different direction

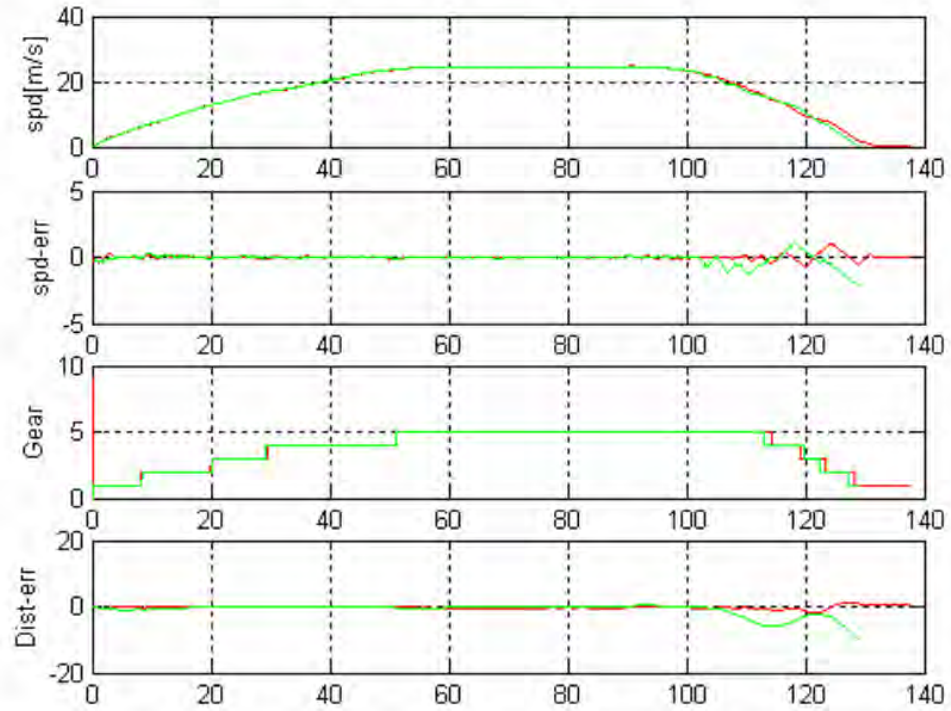


Run 7: Max speed 55 [mph] ; Des_dist: 4 [m]; Different direction

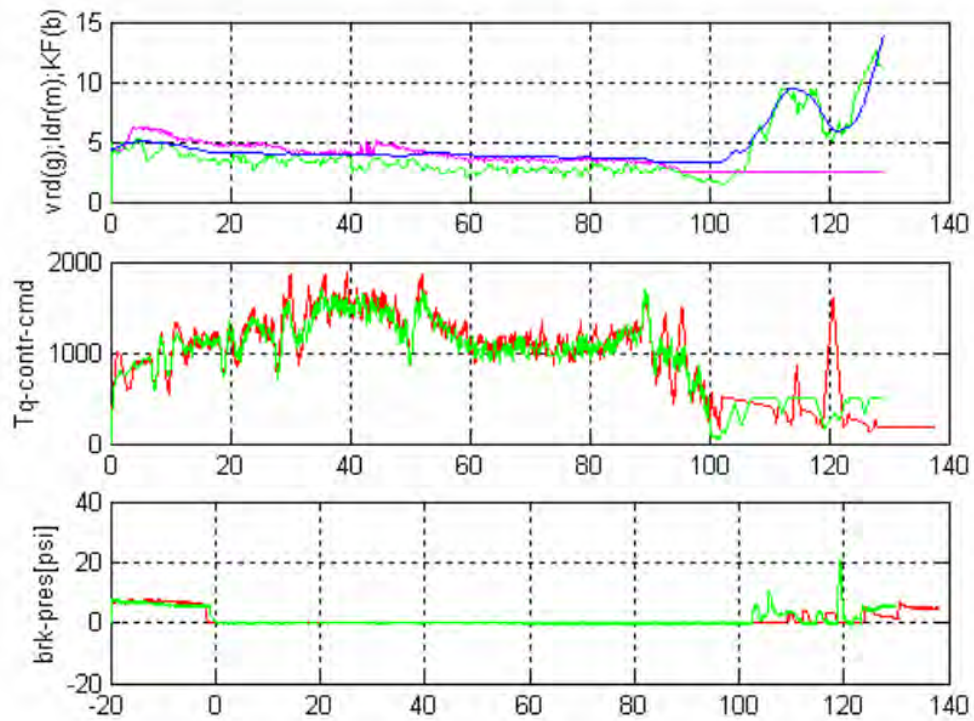


In the following run for 3m following distance, the two trucks are with empty trailers.

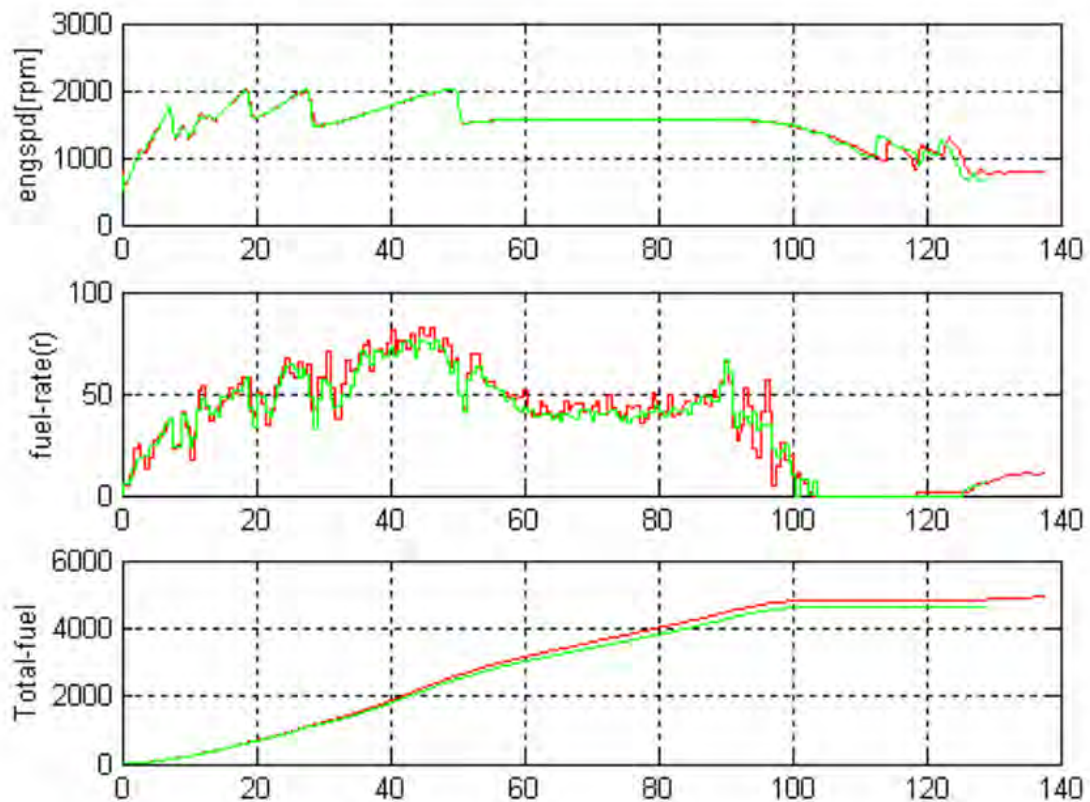
Run 8: Max speed 55 [mph] ; Des_dist: 3[m]



Run 8: Max speed **55 [mph]** ; Des_dist: **3[m]**



Run 8: Max speed 55 [mph] ; Des_dist: 3[m]



Appendix D

In-Vehicle Network Use on AVCSS Demo Heavy-Duty Vehicles

Contents

1	Introduction	3
2	Functions of in-vehicle networks	4
3	Technical specifications of in-vehicle networks	6
3.1	SAE J1587 and SAE J1922 networks	6
3.2	SAE J1939 communications and control network	7
3.3	Use of J1587 and J1939 on PATH AVCSS HDVs	8
3.3.1	Transmission ECMs	8
3.3.2	Engine ECMs	8
3.3.3	Braking system ECM	9
3.4	J1939 network capacity	10
4	Hardware and software architecture of PATH in-vehicle network subsystems	11
4.1	Connections of PATH computer to in-vehicle networks	11
4.1.1	Freightliner truck connections to in-vehicle networks	11
4.1.2	Bus connections to in-vehicle networks	12
4.2	Software architecture for J1939 actuation	12
4.3	Processor and bus utilization	14
5	Role of in-vehicle networks in overall system architecture for automation	16
5.1	Background on longitudinal control of HDVs	16
5.2	Longitudinal control actuation with J1939	18
6	Validation processes in the development process	20

7 Self-testing features of in-vehicle network

21

Chapter 1

Introduction

This report documents the use of in-vehicle networks in the experimental work being conducted by the PATH Advanced Vehicle Control and Safety Systems (AVCSS) group on Heavy Duty Vehicles.

In Chapter 2 we discuss the general functions of in-vehicle networks in Heavy Duty Vehicles (HDVs). In Chapter 3 we give the technical specifications for these networks, and describe which of the networks are available on each of our vehicles. In Chapter 4 we describe the hardware and software components of the in-vehicle network subsystem and how they are connected. In Chapter 5 we specify how the in-vehicle networks and associated software fit into the overall architecture for vehicle automation. In Chapter 6 we describe validation processes in the development process, and in the sixth chapter the self-testing capabilities of in-vehicle networks and associated software are discussed.

On each of the vehicles a PC104 computer, described in another document, has been installed and communicates directly with the in-vehicle networks, taking the role of another subsystem Electronic Control Module (ECM). This computer will be referred to as the *PATH control computer* in this document.

Chapter 2

Functions of in-vehicle networks

The PATH AVCSS group has three Freightliner trucks with Cummins diesel engines, two 40-foot buses with Cummins Compressed Natural Gas (CNG) engines, and one 60-foot articulated bus with a Detroit Diesel engine. On modern HDVs such as these, the engine, transmission and braking systems are each controlled by a separate Electronic Control Module (ECM). These ECMs communicate via in-vehicle serial networks. These in-vehicle networks have several important functions:

1. *Broadcast*: information about engine speed, wheel speed, current gear and many other vehicle system parameters is regularly broadcast by each ECM and may be used by other ECMs for control or for display of information.
2. *Command*: the transmission or an anti-locking braking system may command or inhibit engine speed or torque by sending a message on these networks; advanced cruise control systems may also use these capabilities. Commands can also be sent to activate airbrakes, transmission retarders and engine retarders.
3. *Fault reporting*: special messages report faults. These messages can activate dashboard "blink code" or error number systems for fault analysis. They can also be read by the PATH control computer during real-time control.

4. *Off-line diagnostics and information reporting:* the in-vehicle networks can be used for communication with a variety of service tools to report system settings and trip information, and in some cases can be used to recalibrate the ECM.

Chapter 3

Technical specifications of in-vehicle networks

On our HDVs three types of in-vehicle network are used: SAE J1587, SAE J1922, and SAE J1939.

3.1 SAE J1587 and SAE J1922 networks

The SAE J1587 [14] and J1922 [15] protocols are both based on the J1708 serial data communications link standard, which uses RS-485 transceivers and receivers. The SAE J1708 standard, which defines basic hardware and conditions required for data exchange between vehicle components, was developed in 1986 [16]. The J1708-based networks are limited to a bandwidth of less than 10Kb its/second, can have up to 20 nodes, and are limited to about 40 feet for a single network.

The J1708 message format is very broad, providing for 21-byte messages with a single byte Message ID (MID) and a check sum (included in the 21-bytes). MIDs are loosely grouped by equipment type (engines, brakes, tires, etc.) but the content of the message is left completely up to the manufacturer. It is the job of manufacturer to define and document the message corresponding to an MID, and of the system integrator to make sure that two different pieces of equipment on the same network aren't using the same MID to mean different things.

The J1587 protocol was developed soon after J1708 to sit on top of it with standardized definitions for information sharing and diagnostic functions.

Originally intended for heavy-duty trucks, J1708/J1587 was adapted for bus use in 1992 [16]. J1708 reserves MIDs greater than 127 for J1587. For J1587, MIDs represent the source ECM of the message, and standard Parameter Ids (PIDs) are used within a message to indicate that the following data is of the specified type. Units on the network are encouraged to send messages with multiple PIDs per message to avoid message overhead.

J1922 was developed as an interim standard, until J1939 is widely available, for power train information. MIDs 69-86 are reserved by the J1708 standard for J1922 usage, and have defined formats, with a message length and fixed field types corresponding to each MID. J1587 is designed to continue as a low-cost alternative to J1939 for electronic data interchange of less crucial information.

3.2 SAE J1939 communications and control network

The SAE J1939 standard [13] is a higher-level protocol designed for use in HDVs with a Controller Area Network (CAN). CAN is a serial bus protocol for non-destructive collisions originally developed for use in automobiles by a German company, Bosch GmbH, in the 1980s. J1939 network have bandwidths of up to 1M bit/second. and can have up to 30 nodes. Like J1708 networks, they are limited to about 40 feet for a single network. Standards for bridging multiple J1939 networks have been defined. CAN is now an international standard (ISO 11898), and in HDVs the CAN-based SAE J1939 vehicle network is replacing the slower SAE J1587 and J1922 protocols for both diagnostic and component control applications (such as anti-lock BRAking systems and cruise control). CAN was designed to implement real-time, distributed control systems [18], and the ISO 11898 standard is currently under development to improve its performance in safety-critical applications [2, 4].

3.3 Use of J1587 and J1939 on PATH AVCSS HDVs

The in-vehicle network connections and functions they control are described in the following for each type of ECM in our vehicles. The PATH AVCSS group has access to proprietary information about messages broadcast and interpreted by these ECMs which will not be detailed here.

3.3.1 Transmission ECMs

All of our vehicles use Allison automatic transmissions with WTEC-III Electronic Controls [1]. The Allison transmission ECM has both J1587 and J1939 ports, and autodetects which of these can be used to obtain throttle information used by the transmission. On our trucks, the transmission J1939 port is not connected to the engine J1939 port, and throttle information is obtained using the J1587 network. On our buses, the transmission and engine are connected using J1939.

On all our vehicles, a transmission retarder is present, and has been configured to respond to Torque/Speed Control commands sent to the transmission ECM on the J1939 network. The PATH control computer is connected to the transmission J1939 network on all vehicles, in order to read data (including current gear, input shaft speed and output shaft speed) broadcast by the transmission and control the transmission retarder.

3.3.2 Engine ECMs

Our three Freightliner trucks were built with identical Cummins CelectPlus electronic subsystems; the engine ECMs have connections for all three types of in-vehicle network. J1587 provides information used by dashboard electronics to provide blink codes for fault conditions and also is used for communication with the Electronic Braking System (EBS) and the transmission. While some messages are broadcast on the J1922, the information is redundant with that on the J1939, and in our trucks no other devices are connected to J1922.

The engine J1939 network on our trucks is connected to the EBS ECM, but not to the transmission ECM. The PATH control computer is separately connected to each of the engine and transmission J1939 networks. The ECMs

on our trucks are calibrated to respond to the J1939 Torque/Speed Control message, when it is sent from an approved address. Both torque and speed control commands can be sent.

An engine retarder is present on all our trucks. It can be controlled during manual driving by setting a switch on the dashboard to high or low. A message sent to the J1939 address for engine retarders is received and responded to by the engine ECM as a request that a percentage of maximum retardation torque be applied by the engine retarder.

Our 40-foot CNG buses have a Cummins C8.36+ CM556 electronic control system, which features only the J1587 and J1939 serial networks. A somewhat different set of broadcast messages on these networks is supported than with the CelectPlus on the trucks. The transmission, engine and braking system ECMs are all connected together by both the J1939 and the J1587 serial networks; the PATH control computer is also connected to this network. On the CNG buses, the engine ECM is not calibrated to respond to the J1939 Torque/Speed Control message, and no engine retarder is available.

Our 60-foot bus has a Detroit Diesel engine with an ECM that broadcasts on both J1587 and J1939 networks, and also responds to J1939 Torque/Speed Control command requests for engine torque and engine speed. No engine retarder is configured in our engine model, and engine retarder messages sent to the engine ECM are ignored. Transmission, engine and braking systems are all connected by both J1587 and J1939.

3.3.3 Braking system ECM

Our Freightliner trucks, originally configured with a standard WABCO Electronic Braking System (EBS), have been outfitted with a *brake-by-wire* EBS that was signal compatible with the original EBS. The new EBS allows WABCO-proprietary braking commands to be sent over the J1939 network. This EBS was intended for the European market, and is not street-legal in the United States.

The braking systems on our buses are Anti-Lock Braking Systems (ABS) without the centralized electronic control of an EBS[20]. Instead of re-doing the braking systems on the vehicle for EBS in order to use brake-by-wire, we have modified the pneumatic control system to accept a proportional voltage control set by the computer. The in-vehicle networks are not involved in braking system control on these systems except for some reports of brake and ABS status.

3.4 J1939 network capacity

On all our vehicles, the J1939 data link is configured at 250 Kbit/second, giving a maximum bandwidth of approximately 1-2 J1939 messages per millisecond [7]). On the Freightliner trucks, J1939 bus loading due to standard broadcast messages from the Cummins engine ECM as shown by Canalyzer software [19] has been measured at about 6 percent. There is excess capacity that the PATH control computer can use to send Torque/Speed Control messages to the engine and the retarder, at the J1939 standard update rates of 10 and 50 ms, respectively, as well as to send a proprietary brake demand message to the WABCO EBS.

Chapter 4

Hardware and software architecture of PATH in-vehicle network subsystems

The first subsection below describes the details of connections that have been made from the PATH control computer to the in-vehicle networks on different systems. The second subsection describes the software architecture for in-vehicle network control.

4.1 Connections of PATH computer to in-vehicle networks

4.1.1 Freightliner truck connections to in-vehicle networks

The J1587 serial network signal is taken from the Diagnostic Connector (Freightliner Part No. DUFHD10 6 12P) located at the forward foot of the left hand (driver's side) pillar. The signals are derived from vampire taps with the green wire corresponding to CAN High and the orange wire corresponding to CAN Low.

The engine J1939 serial network signal is taken from Cummins' Service Tool Connection which is located on the driver's side of the engine and is part of the N14 Plus Engine Harness. It is a three pin connector with Pin A

corresponding to CAN High, Pin B corresponding to CAN Low, and Pin C is the shield.

The Allison transmission J1939 serial network signal is taken from pins on the Transmission ECM. Pin 13 corresponds to CAN High, Pin 29 corresponds to CAN Low, and Pin 12 is the shield. It should be noted that without modification this is a Lite CAN which cannot be connected to the main J1939 Vehicle Bus; consequentially, our computer uses a separate CAN connection for this signal.

The WABCO EBS J1939 serial network signal is taken from connector EC1 on the Central Module located in the left pillar. Pin 3 corresponds to CAN High and Pin 1 corresponds to CAN Low, replacing the J1922 serial network connections on the original WABCO system. This signal is routed through a switch so that the EBS can be disconnected from the engine J1939 network. This may be necessary because of a requirement for continuous communication to the EBS that the engine ECM does not satisfy without assistance from the PATH control computer.

4.1.2 Bus connections to in-vehicle networks

On the buses, the J1587 and J1939 connections between the engine, transmission and braking ECMs were made as part of the OEM wiring. On the CNG buses, we tap into the 18-pin connector for ECU-4S/4M located above the main door at pin 13 (dark blue wire) for J1587B(-) and pin 14 (yellow wire) for J1587A(+), and at pin 1 (mint green wire) J1939B CAN_L, pin2 (black wire) J1939 SHLD and pin 3 (lime green wire) J1939A CAN_H. On the 60-foot bus, we tap into the 18-pin connector for ECU-6S/6M located above the main door at pin 13 (dark blue wire) for J1587B(-) and pin 14 (yellow wire) for J1587A(+), and at pin 1 (green wire) J1939B CAN_L, pin2 (white wire) J1939 SHLD and pin 3 (yellow wire) J1939A CAN_H. For both buses, on the cable to the computer, wire colors are: red J1939 CAN_H, black J1939 CAN_L, white J1587A(+), green J1587B(-).

4.2 Software architecture for J1939 actuation

Software development has been based on PATH's previous passenger car automation work using the QNX 4 Real-time Operating System [11]. A in-memory *publish-subscribe* database is used for communication between

Process name	Processor utilization
dbslv	12.72
jbussend	0.08
longctl	1.88
rdj1587	0.31
rdj1939	0.32

Table 4.1: Processor utilization of control processes.

sensor, actuator and control processes [17]. While the passenger cars were instrumented with standard ISA-bus PCs, a PC104 stack is being used in our HDVs. A package of about 9,000 lines of C code has been written to support reading and sending J1939 commands while interfacing with the database. Details about the functions and data structures are documented in the *SAE J1939 and J1587 Support Package Reference Manual*, which is included as part of this report.

The J1939 support routines are designed to interface with different low-level CAN send and receive routines, depending on the hardware interface to the serial network. The QNX4 CAN driver used on our PATH control computers was written for us by the Underwater Systems and Technology Lab, Institute of Systems and Robotics, FEUP. Details about this driver are documented in the *QNX4 Can Driver Reference Manual* also included as part of this report.

The CAN support provided is simple and does not use any of the special features for receiving into multiple message objects and filtering messages in hardware that the CAN protocol provides. Currently every J1939 message present on the vehicle network is received by the control computer and written to the database. If this is too expensive the software can be easily modified to screen out uninteresting messages.

The PATH J1939 software is very portable from vehicle to vehicle. The same routines that are used to command engine torque in the Cummins Engine in our Freightliner trucks are also used for the Detroit Diesel Engine in our 60-foot bus. The same code controls transmission retarder actuation on all our HDVs.

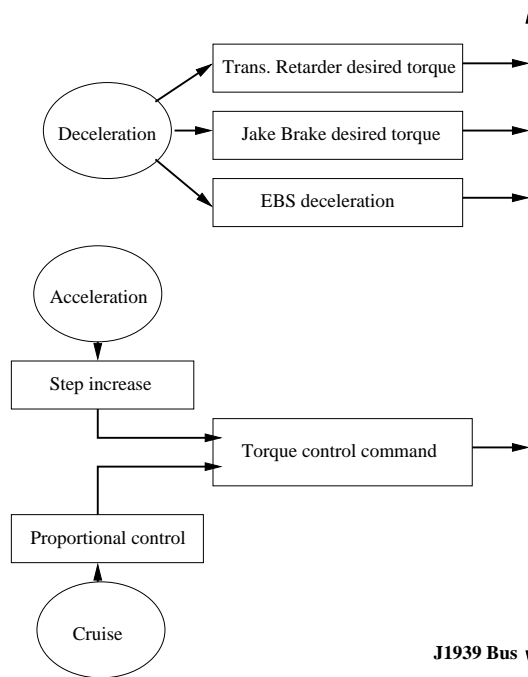


Figure 4.1: J1939 commands issued in braking tests

4.3 Processor and bus utilization

The control, sensor and actuation software runs on a PC104 500 MHz Intel processor system. The processor utilization for each process, measured as process time divided by real time for execution, is shown in Table 4.1.

The processes described in Table 4.1 are `dbslv`, an in memory database server process that is also handling information from radar, GPS, accelerometer and gyro processes, `jbussend`, which issues J1939 command messages to the CAN driver, `longctl`, the actuation tests longitudinal controller, `rdj1587`, which reads messages from J1587 network and writes interesting messages to the database, and `rdj1939`, which does the same for the J1939 network. It is important that processor utilization for these low-level processes be small because the over-all system architecture includes processor-intensive lateral control as well as inter-vehicle communication processes that must all run in real-time.

Loading of the engine J1939 network increased significantly with the ad-

dition of broadcast messages from the EBS system and the sending of commands from the PATH control computer. The bus load of about 6 percent due to the engine's broadcast messages increased to about 10-11 percent average, with a maximum of 16-18 percent as measured using Canalyzer software. About half of this increase is due to J1939 commands from the PATH control computer, the other half is due to broadcast messages from the EBS system. The traffic increase resulted in an increase of CAN bus error frames from about two percent to about 10 percent.

Chapter 5

Role of in-vehicle networks in overall system architecture for automation

Sensor information from the in-vehicle networks is being used instead of add-on sensors for lateral control and fault detection as well as for longitudinal control. However, the biggest role of the in-vehicle J1939 network in our overall system architecture is for longitudinal control actuation, and that is discussed below.

5.1 Longitudinal control of HDVs

Longitudinal control of fully automated road vehicles can be divided into two layers: The upper layer is composed of the vehicle dynamics and feedback control loop. The lower layer is control actuation including throttle and braking systems. Dynamic models of those systems and the corresponding control designs are considered in [9, 10]. This paper concentrates on control actuation using the SAE J1939 vehicle network widely available in commercial HDVs. This longitudinal control actuation of HDV is different from that for fully automated passenger cars developed at PATH [5, 8, 12] in the following aspects. For passenger cars:

- Both throttle and brake control actuators were developed by PATH.
- There is no complicated control feedback loop between the engine con-

trol system and the throttle actuator, which means that one can directly access the fuel rate control.

- Measurement of signals are directly from sensors installed by PATH.
- Braking system uses hydraulic brakes only, and was redesigned by GM Research for PATH [5].

For HDV Century Freightliner:

- With the Freightliner's Cummins N-435 turbo-diesel engine [3], the built-in Electronic Control Module (ECM) controls fuel rate actuation, using complicated algorithms and information from the vehicle network. This means that it is very difficult to directly access fuel rate actuation. Instead, the engine is controlled by overriding the command from the accelerator pedal with a command sent to the engine ECM using the J1939 network. The engine ECM listens to the SAE J1939 Torque/Speed Control command [13] and uses that instead of the accelerator pedal signal to determine fuel rate actuation.
- Measurements of many signals used for longitudinal control are from built-in sensors available to the engine, transmission or braking systems. These measurements are broadcast on the J1939 network by the engine, transmission or braking ECM at certain standard frequencies.
- The braking system contains an Engine Retarder (also called Jake Brake [6]) and a Transmission Retarder as well as Air Brakes controlled by a WABCO Electronic Braking System (EBS) [20]. The operation of each of these braking systems can be directed by commands sent to the engine, transmission or EBS ECMs over the J1939 network.

The increasing sophistication of the electronic control systems in HDVs and their integration with the J1939 vehicle network allows us to implement fully automated longitudinal control with fewer hardware modifications than in earlier vehicles. However, the independent operation of ECMs for each of the different vehicle components (engine, transmission and EBS) increases the overall complexity of the control problem.

5.2 Longitudinal control actuation with J1939

Figure 5.1 shows a system model of longitudinal control activation for PATH's three Freightliner trucks. Each truck is being instrumented with additional sensors (including radar, lidar, magnetometer, GPS, and wireless vehicle-to-vehicle communication for platooning) as well as the information broadcast on the J1939 network by the engine, transmission and EBS ECMs. Vehicle dynamics modeling uses all this information to determine vehicle velocity and acceleration. Lower-level control uses this and the desired speed specified by higher-level control to decide what commands to issue.

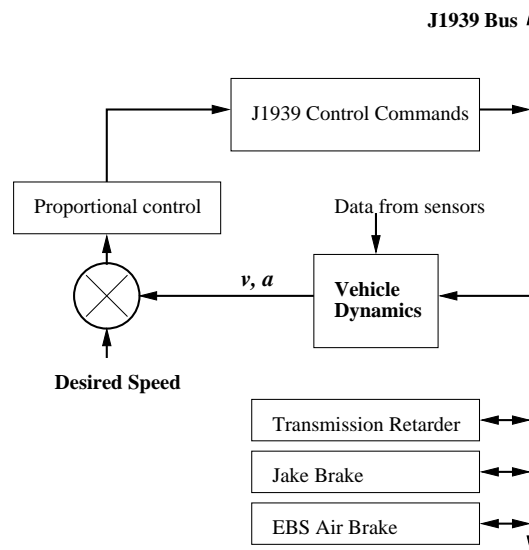


Figure 5.1: Sensing and activating for longitudinal control using J1939 Serial Network

Actuation commands for the engine, jake brake, transmission retarder and EBS are all issued to the J1939 network from a single process. This process is triggered every 5 ms, checks an in-memory database process to see what commands are active and scheduled to be issued, and writes a command to the device if required. Messages to the different ECMs are thus staggered so that they can all be issued at the required frequencies without tying up the J1939 network.

On the 60-foot trucks, the longitudinal control actuation model is similar

for engine torque/speed control and for the transmission retarder, but there is no engine retarder available and there is no J1939 control of the air brakes. On the 40-foot buses, only the transmission retarder can be controlled by J1939, so the in-vehicle networks play a role only in providing feedback from sensors read by subsystem ECM.

Details about the common software used to interface with the J1939 support package and with other actuators can be found in the *Longitudinal Shared Actuation Code Reference Manual* included as part of this report.

Chapter 6

Validation processes in the development process

Standalone test code was developed to monitor in-vehicle network messages. This code was used during manual driving to explore the characteristics of different messages present on the in-vehicle networks on our vehicles, and to determine the correlation of different parameters to vehicle operation and conditions. These programs have been validated and can now also be used for hardware connection tests and for checks on correct operation of the in-vehicle networks.

In-memory traces were also taken of parameters read from the in-vehicle networks while running profiling and prototype control applications under computer control. These traces are saved to disk at the end of a run, so that disk access does not interfere with the control applications. This data was used to develop models of vehicle behavior. Further tests and data gathering were used to validate the models; the models are then used to provide checks of correct in-vehicle network operation.

Chapter 7

Self-testing features of in-vehicle network

Self-testing features of in-vehicle networks currently in place include:

- Dashboard display of J1587 fault codes, including fault counts and blink codes.
- Control software fails to proceed to automated control if required messages from in-vehicle networks have not been read during initialization.

As the control software matures, we plan to add the following features that depend on information from the in-vehicle networks:

- Software access to fault messages.
- Timeouts to detect bus-off condition (absence of messages).
- Validation of correct control actuation based on values reported by broadcast messages on the in-vehicle network.

The proposal for further research into this area *Intelligent Transducers* by Dr. Adam Howell has been included as part of this report.

Bibliography

- [1] Allison Transmission, *3000MH, 4000MH (WTEC III) Owner's Manual (OM3349EN)*, October 1, 2002, order from www.allisontransmission.com.
- [2] Bosch Microelectronics, *TTCAN IP Module Technical Data*, <http://www.can.bosch.com>.
- [3] Cummins Customer Assistance Center, 1999, *Operation and Maintenance Manual - N-14 Plus series Engines*, Cummins Engine Company
- [4] Fredriksson, L., 2002, CAN for Critical Embedded Automotive Networks, *IEEE Micro*, July-August, p28-35.
- [5] J. K. Hedrick, Nonlinear controller design for automated vehicle applications, *Proc. UKACC Int. Conf. on Contr. '98, 1-4 Sept. 1998, Swansea, U. K.*, p23-31
- [6] Jacobs Vehicle Systems, Engine Brake Theory, <http://www.jakebrake.com/askus/faq/ebt.htm>
- [7] Kvaser, 2001, *CAN overview*, zipped pdf, [http://www.cankingdom.org/download/Download files/Basics/can.zip](http://www.cankingdom.org/download/Download%20files/Basics/can.zip).
- [8] Lu, X. Y., Tan, H.-S., Shladover, S. E. and Hedrick, J. K., 2001, Nonlinear longitudinal controller implementation and comparison for automated cars, *Trans. of ASME, Journal of Dynamic Systems, Measurement and Control*, Vol. 123, p161-167.
- [9] X. Y. Lu and J. K. Hedrick, Modeling of heavy-duty vehicles for longitudinal control, *Proc. of The 6th Int. Symposium on Advanced Vehicle Control*, Hiroshima, Japan, Sept. 9-13, 2002, **Paper No. 109**
- [10] X. Y. Lu and J. K. Hedrick, 2003, Longitudinal Control Design and Experiment for Heavy-Duty Trucks, submitted to *2003 American Control Conference*

- [11] QNX RTOSv4, <http://www.qnx.com>.
- [12] Rajamani, R., H.-S. Tan, B. Law and Zhang, 2000, Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons, *IEEE Trans. on Control Systems Technology*, **Vol. 8, No. 4**, p. 695-708
- [13] Society of Automotive Engineers, 2001, *SAE Truck and Bus Control and Communications Network Standards Manual*.
- [14] SAE J1587, *Joint SAE/TMC Recommended Practices for Electronic Data Interchange Between Microcomputer Systems in Heavy-Duty Vehicle Applications*, revised February 2002, www.sae.org.
- [15] SAE J1922, *Powertrain Control Interface for Electronic Controls Used in Medium and Heavy-Duty Diesel and Highway Vehicle Applications*, September 2000, www.sae.org.
- [16] Schiavone, John J., *Transit Cooperative Research Program Report 43*, "Understanding and Applying Advanced On-Board Bus Electronics", Transportation Research Board, National Research Council, National Academy Press, Washington, D.C. 1999.
- [17] Tripakis, S., 2002, Description and Schedulability Analysis of the Software Architecture of an Automated Vehicle Control System, *EMSOFT 2002*, Grenoble, France, p123-137.
- [18] Torngren, M. 1995, A perspective to the design of distributed real-time control applications based on CAN, *Proceedings of 2nd Int. CiA CAN conference*, London-Heathrow, 3-4 October 1995.
- [19] Vector Informatik GmbH, CANalyzer Manual, Version 4.0, <http://www.canalyzer.com>.
- [20] WABCO, 1998, *EBS (EPB) Description of System and Functions*, http://www.wabco.info/intl/pdf/815/000/262/815_262.pdf.

Appendix E

Real Time Token Ring (RTTR) for QNX 4 Reference Manual 1.0

Generated by Doxygen 1.3.4

Tue Dec 16 10:13:04 2003

Contents

1 Real Time Token Ring (RTTR) for QNX	1
2 Real Time Token Ring (RTTR) for QNX 4 Data Structure Index	5
3 Real Time Token Ring (RTTR) for QNX 4 File Index	5
4 Real Time Token Ring (RTTR) for QNX 4 Data Structure Documentation	6
5 Real Time Token Ring (RTTR) for QNX 4 File Documentation	14

1 Real Time Token Ring (RTTR) for QNX

This document is a detailed manual of the Real Time Token Ring protocol (RTTR) implementation for the Orinoco 802.11b wireless PCMCIA card under the QNX realtime operating system. The Linux `orinoco_cs` driver was used as a basis for the device-level support of the Orinoco PCMCIA 802.11b wireless card. Minimal changes were made to the device-level driver source, in order to be able to track future changes in the GNU Public License `orinoco_cs` driver. For more information about the `orinoco` driver, see the QNX 4 Driver for Orinoco Wireless Card reference manual appended to the end of this document and the documentation for the Linux `orinoco_cs` driver [Tourrilhes, 2003].

1.1 Overview

The RTTR protocol has been implemented as a single process under QNX that provides a bidirectional flow of information between a device driver process for the Orinoco wireless card and user-defined application code. Figures ??, ??, and ?? illustrate the implementation of the three main tasks of the RTTR protocol: configuration, transmission, and reception. The RTTR protocol and device driver processes interact with each other via interprocess (I-P) messaging. The I-P messaging support under QNX is extremely efficient and fast, so there is only minor additional overhead in both the source and executable code. This division of protocol and device driver into separate processes has its own advantages and disadvantages, however the main reasons for this setup are to allow for easier debugging, development, and future porting to other operating systems.

For the current PATH application of automated vehicle control, the application code effectively provides an interface to several variables within the PATH publish-subscribe database. The simplicity of this code did not require separation into another process, so the application code has been embedded in the `rttr` process in the current version.

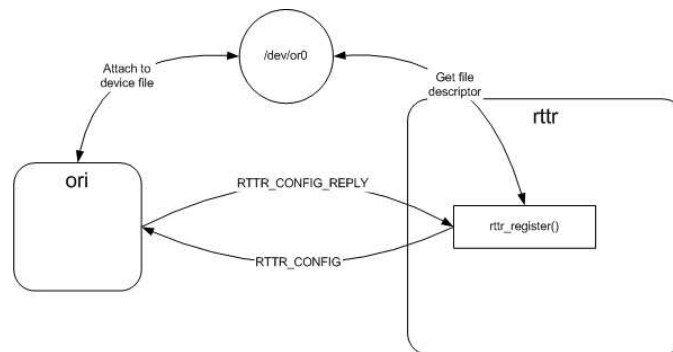


Figure 1: Schematic of RTTR Configuration task

However, the interface between the protocol and application is well defined to allow for future modifications towards separate protocol and application processes. The remainder of this section will cover the following topics in more detail: the interface between the RTTR and Orinoco driver processes, the interface between the RTTR and application code, and finally the PATH automated vehicle control application code. For brevity throughout this document, the RTTR and Orinoco processes will be termed *rttr* and *ori*, respectively.

1.2 RTTR and Orinoco driver Inter-Process Communication

Although there are several methods of providing I-P messaging in QNX, the chosen method was through the use of the file descriptor `/dev/or0` due to its standardized behavior across Unix-based systems. The specific types of I-P communication between *rttr* and *ori* are limited to three essentially atomic operations: configuration, packet reception, and packet transmission. The configuration operation runs once both of the processes have been started to share information required for both I-P and Wireless communication. First, the *ori* process is started, attaches to the `/dev/or0` file descriptor through QNX system calls, and receive-blocks waiting for a `RTTR_CONFIG` message from *rttr*. Once the *rttr* process is started, it gets the `/dev/or0` file descriptor using the `open()` system call, and transmits an `RTTR_CONFIG` message to *ori* through `/dev/or0`. This message contains two proxy IDs that will be used for flow control in the other atomic operations.

These two proxies are called data rx proxy and ok tx proxy, and their specific roles will be described later. The *rttr* process will then receive-block waiting for a `RTTR_CONFIG_REPLY` message from *ori*. Once *ori* processes the `RTTR_CONFIG` message, it replies with a `RTTR_CONFIG_REPLY` message containing the MAC address of the orinoco card. This address is needed by the RTTR protocol to generate appropriate packet headers.

Once the configuration operation is complete, the *rttr* process is receive-blocked with respect to the *ori* process until one of two events occur: a packet is received by the

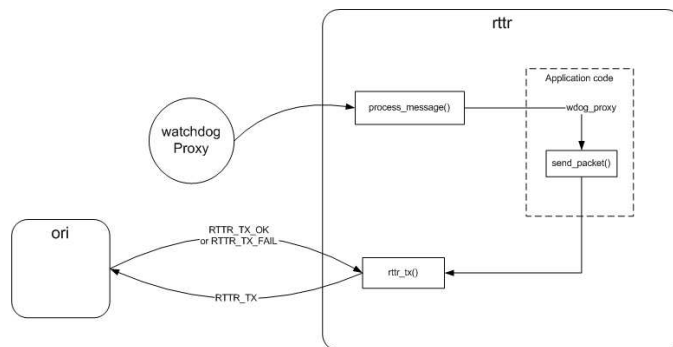


Figure 2: Schematic of RTTR Transmission task

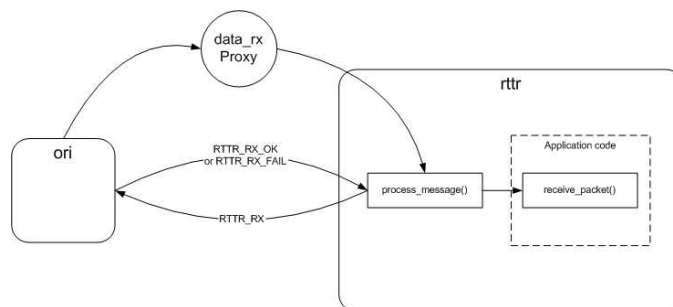


Figure 3: Schematic of RTTR Reception task

Orinoco card or the watchdog timer times out. The reception event is interrupt driven within the ori process, while the time out is an OS timer, however both are indicated to the rttr process through proxies; the `data_rx` proxy mentioned above in the configuration operation for packet reception, and a `wdog` proxy for the timeout. Currently, the RTTR protocol does not use the `ok_tx` proxy, although it has been included for future implementation of error detection and flow control. Each of these cases will now be described in more detail.

When a packet has been received by the Orinoco card, the ori process notifies the rttr process by activating the `data_rx` proxy. Once the `data_rx` proxy is received by the rttr process, it immediately requests the received packet by sending a `RTTR_RX` message to the ori process. The ori process in turn replies with either a `RTTR_RX_OK` message containing the received packet, or a `RTTR_RX_FAIL` message if there was a problem. The received packet is then processed by the RTTR protocol through the function `receive_packet()`.

The transmission of packets occurs when a mode-dependent event occurs. If the node is a MASTER, then a new packet is transmitted upon receipt of the `wdog` proxy. For SLAVE nodes, a new packet is transmitted after successful reception of a packet from the preceding node in the ring. In either case, the `rttr_tx()` function is called to pass the

raw packet to the ori process via an RTTR_TX I-P message. The ori process in turn replies with either an RTTR_TX_OK message if the packet was transmitted successfully, or an RTTR_TX_FAIL message if there was a problem.

1.3 RTTR and Application Interface

The interface between the RTTR protocol and the application specific code is defined by a two functions, `send_packet()` and `receive_packet()`, that deal with the transmission and reception of packets, respectively. The function `send_packet()` is used to transmit data using the RTTR protocol. This function encapsulates the raw data with 802.2 ethernet and RTTR headers, then sends the packet on to the ori driver for transmission using the function `rtrr_tx()`. The `receive_packet()` function hook is called when a data packet has been received. This function should be written by the user, with the input to `receive_packet()` being the raw data. Since there are no headers on the input data, this function should at least make the raw data available to the application code in some form.

1.4 PATH Vehicle-to-Vehicle Communication Application

An integral part of the PATH automated vehicle control system is the use of intervehicle wireless communication to pass vehicle state information throughout a platoon. The vehicle state information consists primarily of vehicle distance, velocity, and acceleration measurements along with platoon coordination information, such as fault status and operating mode. The requirements for this wireless communication are strict; real-time operation with transmission of the state information for all vehicles within the platoon in a period of 20 milliseconds. The periodic transmission functionality was implemented by creating a periodic QNX timer that activates a proxy called wdog. When the wdog proxy is received, the `send_packet()` function hook is called by the rtrr process. Currently, the `send_packet()` function reads the required state information from the PATH database directly, and sends a transmission request to the driver through the `rtrr_tx()`. In parallel to this transmission path, the reception of data packets is passed through `process_message()` which in turn calls `receive_packet()`. The `receive_packet()` function parses the incoming data packet, and directly writes the other vehicles state information to the PATH database. Validation of the overall vehicle-to-vehicle communications implementation on the in-vehicle PC/104 computing platform has been conducted and detailed in [Demo Report, 2003].

References

[Tourrilhes, 2003] The Linux Wavelan Drivers, http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Wavelan.html.

[Demo Report, 2003] *Development and Demonstration of Automated Bus Rapid Transit and Automated Truck Operations*, California PATH Technical Report, 2003.

2 Real Time Token Ring (RTTR) for QNX 4 Data Structure Index

2.1 Real Time Token Ring (RTTR) for QNX 4 Data Structures

Here are the data structures with brief descriptions:

buffer_item (Circular buffer item structure)	6
message_t (IP messages Inter-process (I-P) messages between the orinoco and rttr processes are a union of the various supported message types)	7
packet (RTTR raw packet I-P message)	8
packet_t (PATH communications packet definition Standard definition of data included in a wireless communication packet)	8
rttr_config (RTTR configuration I-P message)	10
rttr_config_reply (RTTR configuration reply I-P message)	11
station_t (Tokenring station structure)	11

3 Real Time Token Ring (RTTR) for QNX 4 File Index

3.1 Real Time Token Ring (RTTR) for QNX 4 File List

Here is a list of all documented files with brief descriptions:

lib.h	??
long_comm.h	??
ori_inc.h	??
rttr.c (Main QNX code for the rttr process)	14
rttr.h	??

4 Real Time Token Ring (RTTR) for QNX 4 Data Structure Documentation

4.1 `buffer_item` Struct Reference

Circular buffer item structure.

```
#include <rttr.h>
```

Data Fields

- char `type` [10]
Packet type: tx: or rx:.
- char `hour`
hour of packet timestamp
- char `minute`
minute of packet timestamp
- char `second`
second of packet timestamp
- double `rtrip`
for MASTER node, the roundtrip time of the packet
- unsigned short `millisec`
millisecond of packet timestamp
- int `strength`
current signal strength
- `packet_t` `packet`
Raw packet.

4.1.1 Detailed Description

Definition of structure used to store process information in the circular buffer.

The documentation for this struct was generated from the following file:

- `rttr.h`

4.2 message_t Union Reference

IP messages Inter-process (I-P) messages between the orinoco and rttr processes are a union of the various supported message types.

```
#include <ori_inc.h>
```

Data Fields

- unsigned short `type`
Type of message.
- packet `pack`
Data packet message.
- `_io_open` `open`
Open message to ori.
- `_io_open_reply` `ropen`
Reply to open message from ori.
- `_io_close` `close`
Close message to ori.
- `_io_close_reply` `rclose`
Reply to close message from ori.
- `rttr_config` `config`
Inter-process communication configuration message.
- `rttr_config_reply` `rconfig`
Reply to inter-process communication configuration from ori.

4.2.1 Detailed Description

Note that the standard `open`, `ropen`, `close`, `rclose` message types provide support for accessing the orinoco card through the file handle `/dev/or0`.

The documentation for this union was generated from the following file:

- `ori_inc.h`

4.3 packet Struct Reference

RTTR raw packet I-P message.

```
#include <ori_inc.h>
```

Data Fields

- msg_t [type](#)
Message type (RTTR_TX or RTTR_RX).
- unsigned int [len](#)
Length of data array.
- unsigned char [data](#) [MAX_PACKET_LEN]
Raw packet data.

4.3.1 Detailed Description

Inter-process message that contains a raw RTTR packet. The packet includes the data plus encapsulating ethernet and RTTR headers. Note that the ethernet header is required for correct address resolution via the 802.11 wireless protocol.

The documentation for this struct was generated from the following file:

- [ori_inc.h](#)

4.4 packet_t Struct Reference

PATH communications packet definition Standard definition of data included in a wireless communication packet.

```
#include <long_comm.h>
```

Data Fields

- int [node](#)
Node number of packet transmitter.
- char [hour](#)
hour of tx timestamp

- char **minute**
minute of tx timestamp
- char **second**
second of tx timestamp
- unsigned short **millisec**
millisecond of tx timestamp
- unsigned long **id**
absolute packet ID number
- float **global_time**
platoon global time
- float **acc_traj**
Desired acceleration from profile (m/s^2).
- float **vel_traj**
Desired velocity from profile (m/s).
- float **velocity**
Current velocity (m/s).
- float **accel**
Current acceleration (m/s^2).
- float **range**
*Range from *dar fusion (m).*
- float **rate**
*Relative velocity from *dar fusion (m/s).*
- float **latitude**
Latitude from GPS (if available).
- float **longitude**
Longitude from GPS (if available).
- float **altitude**
Altitude from GPS (if available).

- int [marker_number](#)
Last marker number passed.
- int [marker_counter](#)
Current marker count.
- unsigned short [my_id](#)
Current vehicle ID.
- unsigned short [maneuver_id](#)
Current vehicle maneuver ID.
- unsigned short [fault_mode](#)
Current fault mode ID.
- unsigned short [maneuver_des_1](#)
Desired maneuver of lead vehicle.
- unsigned short [maneuver_des_2](#)
Desired maneuver of preceding vehicle.

The documentation for this struct was generated from the following file:

- [long_comm.h](#)

4.5 rttr_config Struct Reference

RTTR configuration I-P message.

```
#include <ori_inc.h>
```

Data Fields

- [msg_t type](#)
Message type (RTTR_CONFIG).
- [pid_t tx_proxy](#)
Proxy called by ori to indicate ready to transmit.
- [pid_t rx_proxy](#)
Proxy called by ori to indicate packet recieved.

4.5.1 Detailed Description

Inter-process message that contains I-P messaging configuration information that will be sent to orinoco driver process.

The documentation for this struct was generated from the following file:

- `ori_inc.h`

4.6 rttr_config_reply Struct Reference

RTTR configuration reply I-P message.

```
#include <ori_inc.h>
```

Data Fields

- `msg_t type`
Message type (RTTR_CONFIG_REPLY).
- unsigned char `mac_addr` [ETH_ALEN]
MAC address of orinoco card.

4.6.1 Detailed Description

Inter-process message that contains I-P communication configuration information sent back from the orinoco driver process.

The documentation for this struct was generated from the following file:

- `ori_inc.h`

4.7 station_t Struct Reference

Tokenring station structure.

```
#include <rttr.h>
```

Data Fields

- `int fd`
File descriptor for access to orinoco driver.

- pid_t [ok_tx_pid](#)
Transmit done interrupt from ori.
- pid_t [data_rx_pid](#)
Receive packet interrupt from ori.
- timer_t **tid**
- pid_t [wdog_pid](#)
Watchdog timer proxy id.
- unsigned char [mac_address](#) [ETH_ALEN]
MAC address of card.
- int [mode](#)
Station mode = MASTER, SLAVE.
- int [node_id](#)
logical position in ring
- int [num_node](#)
number of nodes in ring
- int [status](#)
status of comm
- int [strength](#)
signal strength for the node
- long [rotation_time](#)
rotation time (msec)
- int [fault](#)
fault status: -1 emergency, 0 none, 1 silent
- db_clt_typ * [pclt](#)
rotation time (msec)
- data_buffer [dbuff](#)
rotation time (msec)
- unsigned long [tx_id](#)

packet id of last transmitted packet

- unsigned long `rx_id`
packet id of last received packet
- unsigned long `dropouts`
no
- unsigned long `oorders`
no
- unsigned long `tx_errors`
no
- unsigned long `rx_errors`
no
- float `rtrip_time`
roundtrip time (sec)
- unsigned char * `data`
transmission data buffer
- unsigned int `data_size`
size of transmitted data buffer
- timeb `last_tx`
time of last transmission
- `packet_t rx`
last received packet

4.7.1 Detailed Description

Main data structure that contains rtrr node and ring information.

4.7.2 Field Documentation

4.7.2.1 unsigned long `station_t::dropouts`

of dropped packets

4.7.2.2 unsigned long `station_t::orders`

of packets out of order

4.7.2.3 unsigned long `station_t::rx_errors`

of reception errors (hardware)

4.7.2.4 int `station_t::status`

link = LINK_UP, LINK_DOWN

4.7.2.5 unsigned long `station_t::tx_errors`

of transmission errors (hardware)

The documentation for this struct was generated from the following file:

- `rttr.h`

5 Real Time Token Ring (RTTR) for QNX 4 File Documentation

5.1 `rttr.c` File Reference

Main QNX code for the `rttr` process.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/proxy.h>
#include <sys/types.h>
#include <sys/kernel.h>
#include <sys/io_msg.h>
#include <signal.h>
#include <setjmp.h>
#include <time.h>
```

```
#include <sys/timeb.h>
#include <linux_compat.h>
#include <linux/netdevice.h>
#include <linux/if_ether.h>
#include <local.h>
#include "lib.h"
#include "rttr.h"
```

Packet transmission functions

- int `rttr_tx` (`station_t *pst`, unsigned char *data, unsigned long data_len)
Send transmitted packet to orinoco driver.

Application specific functions

- void `send_packet` (`station_t *pst`)
Send rttr packet over wireless link.

Protocol setup/cleanup functions

- int `rttr_register` (`station_t *pst`)
Setup interprocess communication between orinoco and rttr processes.
- int `rttr_unregister` (`station_t *pst`)
Cleanup RTTR process resources.
- void `general_setup` (`station_t *pst`)
RTTR general process setup.
- void `watchdog_setup` (`station_t *pst`)
Setup watchdog timer for RTTR process.

Functions

- void `receive_packet` (`station_t *pst`, unsigned char *data, unsigned int data_size)

Process received rttr packets.

- void `process_message` (`station_t *pst`, `pid_t sender_id`)
Inter-process message handling.
- int `main` (`int argc`, `char *argv[]`)
Main loop of the rttr process.

Variables

- int `verbose` = 0

5.1.1 Detailed Description

This file contains the main QNX code for the rttr process. Primarily, the functions within provide inter-process communication support between the orinoco driver (ori) and the real tokenring protocol code (rttr).

Author:

A. Howell

5.1.2 Function Documentation

5.1.2.1 void `general_setup` (`station_t *pst`)

This function provides general setup routines for the rttr process. This includes:

- proxies for handling inter-process communication.
- logging into the PATH database and initialization of database variables.
- creation of a circular buffer for logging.

Parameters:

pst Pointer to tokenring station.

5.1.2.2 int `main` (`int argc`, `char * argv[]`)

This function implements the main loop of the rttr process. The process runs the initialization functions, then waits in a Receive-blocked state until an inter-process message is received.

See also:

[general_setup](#)
[rttr_register](#)
[master_setup](#)
[watchdog_setup](#)
[process_message](#)
[rttr_unregister](#)

Returns:

EXIT_SUCCESS on successful termination, EXIT_FAILURE otherwise

5.1.2.3 void process_message (station_t * *pst*, pid_t *sender_id*)

This function is the main handler for received inter-process messages. Essentially, it handles two types of messages, watchdog timer proxies and proxies called by the orinoco driver process.

When a watchdog timer proxy is received, the action taken depends on whether the node is a MASTER or SLAVE. If the station is a MASTER node, it is used to create the token at a fixed time interval. For a SLAVE node, it is used to detect dropped packets, provide QoS data through signal strength, and warn the driver through the DVI when too many packets are dropped (see FAULT_EMER and FAULT_SILENT macros).

The inter-process proxies initiated by the orinoco driver process are used to control the flow of transmitted/received packets between the rttr and orinoco processes. An ok_tx proxy is received once the orinoco card is available for accepting packets to transmit, but is currently unused by the protocol because of the small packet sizes transmitted. The data_rx proxy is received to notify rttr that a packet has been received by the card, and is available for processing. In response, the rttr process sends an inter-process message to request the packet from the driver, and the received packet is returned in the inter-process reply message.

Parameters:

pst Pointer to tokenring station.
sender_id PID of sending process.

See also:

[send_packet](#)
[receive_packet](#)

5.1.2.4 void receive_packet (station_t * *pst*, unsigned char * *data*, unsigned int *data_size*)

This function accepts the raw data transmitted over the wireless link, and updates the appropriate PATH database variables.

Parameters:

- pst* Pointer to tokenring station.
- data* Pointer to raw data from application code.
- data_len* Length of data array.

5.1.2.5 int rttr_register (station_t * pst)

This function performs several initialization tasks once the rttr process is started. These tasks include:

- opening file descriptor connection to orinoco through /dev/or0.
- initialize inter-process communication between ori and rttr processes.

Parameters:

- pst* Pointer to tokenring station.

Returns:

EXIT_SUCCESS on successful termination, EXIT_FAILURE otherwise

5.1.2.6 int rttr_tx (station_t * pst, unsigned char * data, unsigned long data_len)

This function accepts the raw data to be transmitted from the application specific code, encapsulates it with a 802.2 header, then passes the packet to the orinoco driver process using QNX interprocess messaging.

Parameters:

- pst* Pointer to tokenring station.
- data* Pointer to raw data from application code.
- data_len* Length of data array.

Returns:

EXIT_SUCCESS on successful transmission, EXIT_FAILURE otherwise

5.1.2.7 int rttr_unregister (station_t * pst)

This function cleans up the RTTR processes allocated resources. This includes writing logging information to file and logging out of the database.

Parameters:

- pst* Pointer to tokenring station.

Returns:

EXIT_SUCCESS on successful termination, EXIT_FAILURE otherwise

5.1.2.8 void send_packet (station_t * pst)

This application specific function reads the packet data from the PATH database, and sends it over the wireless link by calling [rttr_tx\(\)](#).

Parameters:

pst Pointer to tokenring station.

See also:

[rttr_tx](#)

5.1.2.9 void watchdog_setup (station_t * pst)

This function creates a watchdog timer for the rttr process.

Parameters:

pst Pointer to tokenring station.

Index

- buffer_item, 6
- dropouts
 - station_t, 13
- general_setup
 - rttr.c, 16
- main
 - rttr.c, 16
- message_t, 7
- orders
 - station_t, 13
- packet, 8
- packet_t, 8
- process_message
 - rttr.c, 17
- receive_packet
 - rttr.c, 17
- rttr.c, 14
 - general_setup, 16
 - main, 16
 - process_message, 17
 - receive_packet, 17
 - rttr_register, 18
 - rttr_tx, 18
 - rttr_unregister, 18
 - send_packet, 18
 - watchdog_setup, 19
- rttr_config, 10
- rttr_config_reply, 11
- rttr_register
 - rttr.c, 18
- rttr_tx
 - rttr.c, 18
- rttr_unregister
 - rttr.c, 18
- rx_errors
 - station_t, 14
- send_packet
 - rttr.c, 18
- station_t, 11
 - dropouts, 13
 - orders, 13
 - rx_errors, 14
 - status, 14
 - tx_errors, 14
- status
 - station_t, 14
- tx_errors
 - station_t, 14
- watchdog_setup
 - rttr.c, 19

Appendix F

WTRP for QNX Reference Manual

1.0

Generated by Doxygen 1.2.16

Thu Apr 10 08:25:41 2003

Contents

1	Wireless Token Ring Protocol (WTRP) for QNX	1
2	WTRP for QNX Data Structure Index	7
3	WTRP for QNX File Index	7
4	WTRP for QNX Data Structure Documentation	8
5	WTRP for QNX File Documentation	17

1 Wireless Token Ring Protocol (WTRP) for QNX

This document is a detailed manual of the Wireless Token Ring protocol (WTRP) implementation for the Orinoco 802.11b wireless PCMCIA card under the QNX realtime operating system. Only the specifics of the WTRP implementation will be covered within this document, however the reader interested in more details about the protocol itself is referred to the references [[Berkeley WOW, 2003](#), [Lee et. al., 2001](#)].

The Linux *orinoco_cs* driver was used as a basis for the device-level support of the Orinoco PCMCIA 802.11b wireless card. Minimal changes were made to the device-level driver source, in order to be able to track future changes in the GNU Public License *orinoco_cs* driver. For more information about the orinoco driver, see the *QNX 4 Driver for Orinoco Wireless* reference manual appended to the end of this document and the documentation for the Linux *orinoco_cs* driver [[Tourrilhes, 2003](#)].

1.1 Overview

The WTRP protocol has been implemented as a single process under QNX that provides a bidirectional flow of information between a device driver process for the Orinoco wireless card and user-defined application code. Figures 1, 2, and 3 illustrate the implementation of the three main tasks of the WTRP protocol: configuration, transmission, and reception.

The WTRP protocol and device driver processes interact with each other via inter-process (I-P) messaging. The I-P messaging support under QNX is extremely efficient and fast, so there is only minor additional overhead in both the source and executable code. This division of protocol and device driver into separate processes has its own advantages and disadvantages, however the main reasons for this setup are to allow for easier debugging, development, and future porting to other operating systems.

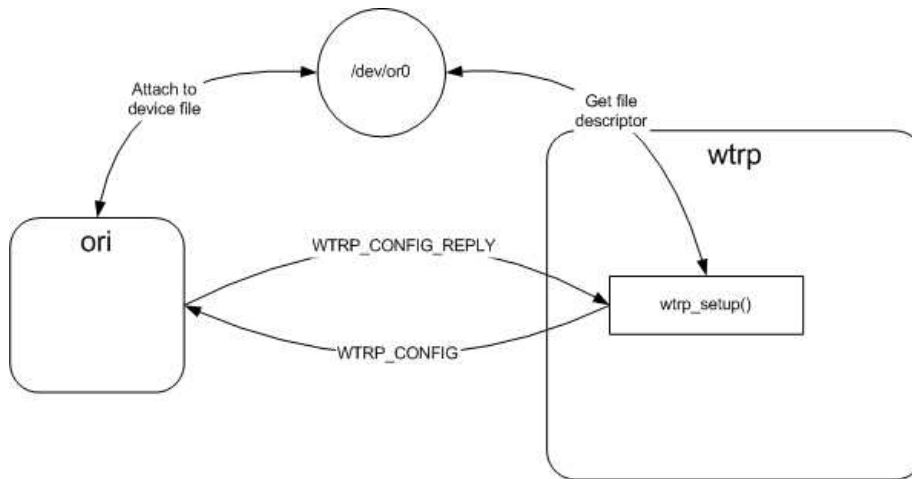


Figure 1: Schematic of WTRP Configuration task

For the current PATH application of automated vehicle control, the application code effectively provides an interface to several variables within the PATH publish-subscribe database. The simplicity of this code did not require separation into another process, so the application code has been embedded in the *wtrp* process in the current version. However, the interface between the protocol and application is well defined to allow for future modifications towards separate protocol and application processes.

The remainder of this section will cover the following topics in more detail: the interface between the WTRP and Orinoco driver processes, the interface between the WTRP and application code, and finally the PATH automated vehicle control application code. For brevity throughout this document, the WTRP and Orinoco processes will be termed *wtrp* and *ori*, respectively.

1.2 WTRP and Orinoco driver Inter-Process Communication

Although there are several methods of providing I-P messaging in QNX, the chosen method was through the use of the file descriptor `/dev/or0` due to its standardized behavior across Unix-based systems. The specific types of I-P communication between *wtrp* and *ori* are limited to three essentially atomic operations: configuration, packet reception, and packet transmission.

The configuration operation runs once both of the processes have been started to share information required for both I-P and Wireless communication. First, the *ori* process is started, attaches to the `/dev/or0` file descriptor through QNX system calls, and receive-blocks waiting for a `WTRP_CONFIG` message from *wtrp*. Once the *wtrp* process is started, it gets the `/dev/or0` file descriptor using the `open()` system call, and transmits a `WTRP_CONFIG` message to *ori* through `/dev/or0`. This message contains two proxy ID's that will be used for flow control in the other atomic operations.

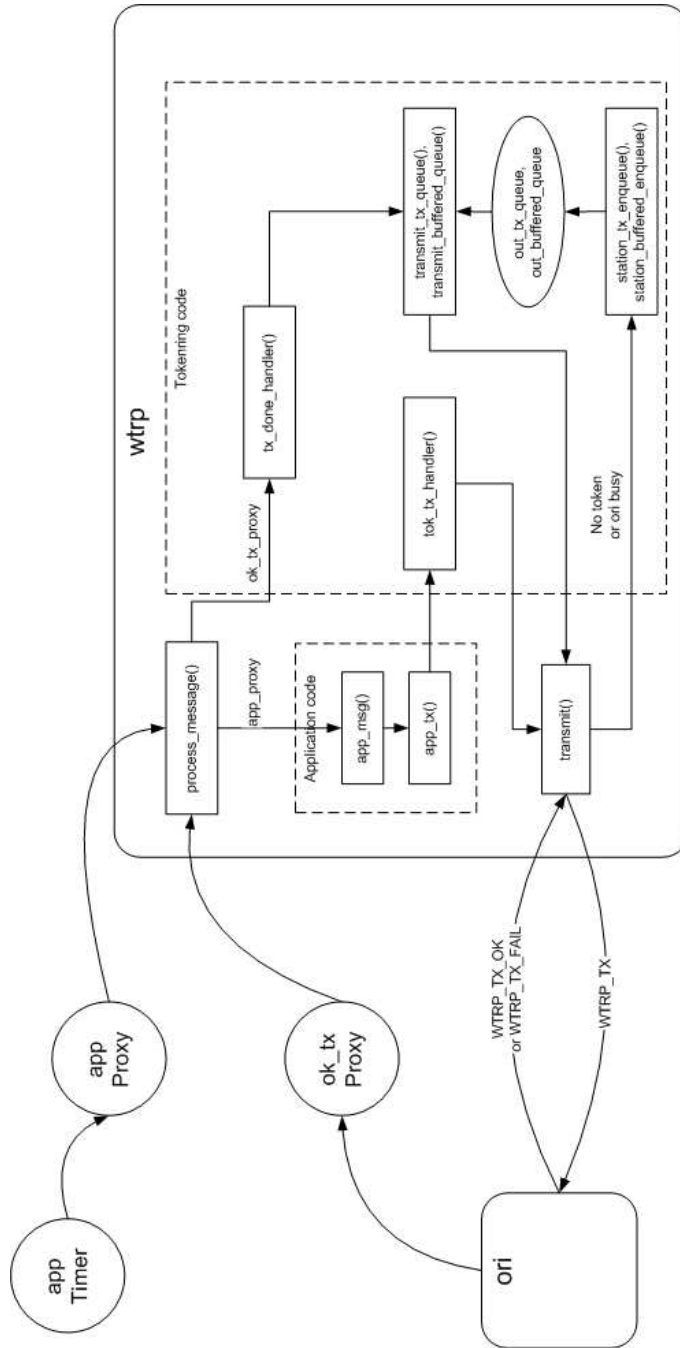


Figure 2: Schematic of WTRP Transmission task

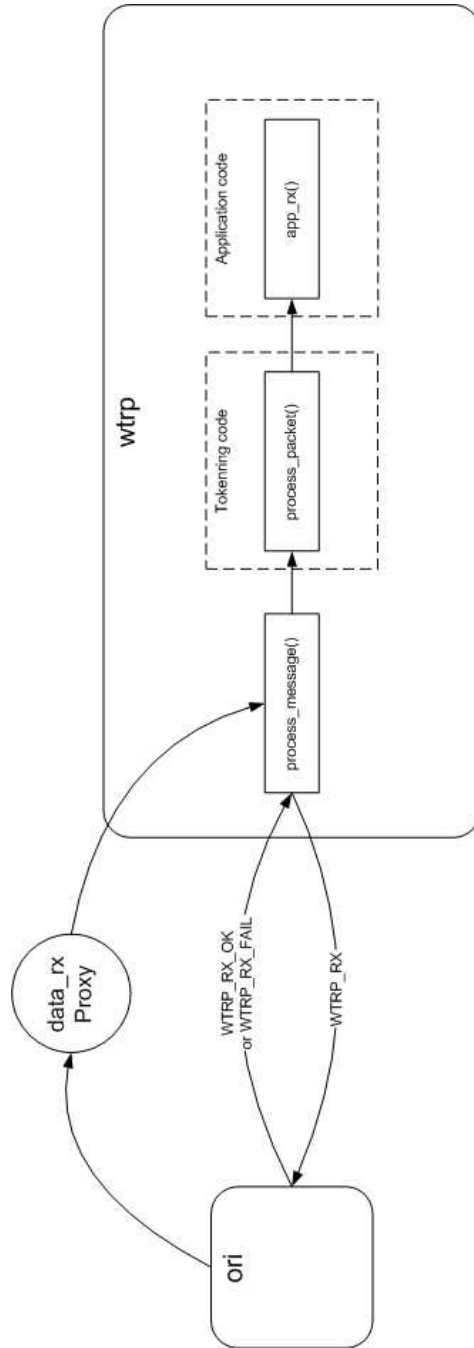


Figure 3: Schematic of WTRP Reception task

These two proxies are called `data_rx_proxy` and `ok_tx_proxy`, and their specific roles will be described later. The `wtrp` process will then receive-block waiting for a `WTRP_CONFIG_REPLY` message from `ori`. Once `ori` processes the `WTRP_CONFIG` message, it replies with a `WTRP_CONFIG_REPLY` message containing the MAC address of the orinoco card. This address is needed by the WTRP protocol to generate appropriate packet headers.

Once the configuration operation is complete, the `wtrp` process is receive-blocked with respect to the `ori` process until one of two events occur: a packet is received by the Orinoco card or the Orinoco card has completed transmission of a packet over the wireless medium. Both of these events are interrupt driven within the `ori` process, and they are also communicated to the `wtrp` process through the two proxies mentioned above in the configuration operation. Each of these cases will now be described in more detail.

When a packet has been received by the Orinoco card, the `ori` process notifies the `wtrp` process by activating the `data_rx` proxy. Once the `data_rx` proxy is received by the `wtrp` process, it immediately requests the received packet by sending a `WTRP_RX` message to the `ori` process. The `ori` process in turn replies with either a `WTRP_RX_OK` message containing the received packet, or a `WTRP_RX_FAIL` message if there was a problem. The received packet is then processed by the WTRP through the function `process_packet()`.

Once the Orinoco card has completed a transmission, the `ori` process notifies the `wtrp` process that it is ready for more packets to transmit by activating the `ok_tx` proxy. Once the `ok_tx` proxy is received by the `wtrp` process, it checks if it currently has the token. If so, the `wtrp` process sends the next packet in the transmission queue within a `WTRP_TX` message to the `ori` process. If the `wtrp` process does not have the token or there are no queued packets, no message is sent. The `ori` process in turn replies with either a `WTRP_TX_OK` message if the packet was transmitted successfully, or a `WTRP_RX_FAIL` message if there was a problem.

1.3 WTRP and Application Interface

The interface between WTRP and the application specific code is defined by a two functions, `app_tx()` and `app_rx()`, that are deal with the transmission and reception of packets, respectively.

The function `app_tx()` is used to transmit data using the WTRP protocol. This function encapsulates the raw data with 802.2 ethernet and WTRP headers, then sends the packet on to the tokenring code for transmission using the function `tok_tx_handler()`. The tokenring code then attempts to send the packet to the `ori` process. If the packet cannot be transmitted, or the WTRP node does not have the token, then the packet is placed on a transmission queue for later transmission.

The `app_rx()` function hook is called when a data packet has been received and processed by the tokenring code. This function should be written by the user, with the

input to `app_rx()` being the raw data. Since there are no headers on the input data, this function should at least make the raw data available to the application code in some form.

1.4 PATH Vehicle-to-Vehicle Communication Application

An integral part of the PATH automated vehicle control system is the use of inter-vehicle wireless communication to pass vehicle state information throughout a platoon. The vehicle state information consists primarily of vehicle distance, velocity, and acceleration measurements along with platoon coordination information, such as fault status and operating mode. The requirements for this wireless communication are strict; real-time operation with transmission of the state information for all vehicles within the platoon in a period of 20 milliseconds.

The periodic transmission functionality was implemented by creating a periodic QNX timer that activates a proxy called `app_proxy`. When the proxy is received, the `app_msg()` function hook is called by the `wtrp` process. Currently, the `app_msg()` function reads the required state information from the PATH database directly, and sends a transmission request to the tokenring code through `tok_tx_handler()`. If the node has the token, the packet is sent to the driver through the `transmit()` function, otherwise it is placed on the transmission queue.

In parallel to this transmission path, the reception of data packets is passed through the tokenring function `process_packet()` which in turn calls `app_rx()`. The `app_rx()` function parses the incoming data packet, and directly writes the other vehicles' state information to the PATH database.

Validation of the overall vehicle-to-vehicle communications implementation is currently ongoing. The first stage of validation involves testing WTRP on a set of laptops to mimic a static platoon. This stage is used to determine proper timing parameters for WTRP, and to verify the stability of the developed code over a longer "burn-in" period. The second stage of validation will involve installation of the communications system in the on-vehicle PC/104 computing platform. This stage will focus on performance tests of the communications system under realistic operating conditions, and on determining the systems robustness to multipath and other environmental interference.

References

- [Berkeley WOW, 2003] Berkeley Web Over Wireless (WOW) webpage, <http://wow.eecs.berkeley.edu/WTRP/>. 1
- [Lee et. al., 2001] Duke Lee, Roberto Attias, Anuj Puri, Raja Sengupta, Stavros Tripakis, and Pravin Varaiya. *A wireless token ring protocol for Intelligent Transportation Systems*. Proc. of the IEEE Intelligent Transportation Systems, 2001. 1

[Tourrilhes, 2003] Jean Tourrilhes, “MPL/GPL drivers for the Wavelan IEEE/Orinoco, and others,” http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Orinoco.html

2 WTRP for QNX Data Structure Index

2.1 WTRP for QNX Data Structures

Here are the data structures with brief descriptions:

ctrl_packet (PATH control packet)	8
message_t (IP messages Inter-process (I-P) messages between the orinoco and wtrp processes are a union of the various supported message types)	8
packet (WTRP raw packet I-P message)	9
private (Application specific data)	10
qos_struct (QOS structure)	10
station_struct (Tokenring station structure)	12
wtrp_config (WTRP configuration I-P message)	16
wtrp_config_reply (WTRP configuration reply I-P message)	17

3 WTRP for QNX File Index

3.1 WTRP for QNX File List

Here is a list of all documented files with brief descriptions:

app.c (PATH WTRP application program)	17
ctable.c (Connectivity table functions)	20
lib.c (Library of general utility functions)	22
params.c (Parameter support functions)	23
tokenring.c (Tokenring protocol implementation)	25

[wtrp.c](#) (Main QNX code for the wtrp process)

35

4 WTRP for QNX Data Structure Documentation

4.1 ctrl_packet Struct Reference

4.1.1 Detailed Description

Packet structure for periodic automated control message between vehicles.

Data Fields

- float [timestamp](#)
Time of packet formation.
- float [dummy_float](#) [14]
Dummy floats used to match size of packet.
- int [dummy_int](#) [5]
Dummy integers used to match size of packet.
- unsigned short [dummy_short](#) [5]
Dummy shorts used to match size of packet.

4.2 message_t Union Reference

4.2.1 Detailed Description

Note that the standard open, reopen, close, rclose message types provide support for accessing the orinoco card through the file handle /dev/or0.

Data Fields

- unsigned short [type](#)
Type of message.

- [packet pack](#)
Data packet message.
- [_io_open open](#)
Open message to ori.
- [_io_open_reply ropen](#)
Reply to open message from ori.
- [_io_close close](#)
Close message to ori.
- [_io_close_reply rclose](#)
Reply to close message from ori.
- [wtrp_config config](#)
Inter-process communication configuration message.
- [wtrp_config_reply rconfig](#)
Reply to inter-process communication configuration from ori.

4.3 packet Struct Reference

4.3.1 Detailed Description

Inter-process message that contains a raw WTRP packet. The packet includes the data plus encapsulating ethernet and WTRP headers. Note that the ethernet header is required for correct address resolution via the 802.11 wireless protocol.

Data Fields

- [msg_t type](#)
Message type (WTRP_TX or WTRP_RX).
- [unsigned int len](#)
Length of data array.
- [unsigned char data](#) [MAX_PACKET_LEN]

Raw packet data.

4.4 private Struct Reference

4.4.1 Detailed Description

Structure to contain local data used in the application-specific code.

Data Fields

- [station_struct * station](#)
WTRP station data.
- [pid_t tx_proxy](#)
Proxy called to initiate transmission of data packet.
- [timer_t tid](#)
Timer to force periodic transmissions.
- [ctrl_pkt_t txpacket](#)
Local copy of pending transmission packet.
- [ctrl_pkt_t rxpacket](#)
Local copy of recieved packet.
- [db_clt_typ * clt](#)
PATH database pointer.

4.5 qos_struct Struct Reference

4.5.1 Detailed Description

WTRP application-specific quality of service (QOS) (not implemented)

Data Fields

- unsigned long `max_token_rotation_time`
Desired max.
- unsigned long `max_token_holding_time`
Desired max.
- unsigned long `solicit_successor_prob`
Desired solicit successor probability.
- unsigned long `max_num_token_pass_try`
Desired max.
- `qos_struct * next`
Linked list pointer to next qos_struct.
- unsigned short `user_max_num_node`
User defined max.

4.5.2 Field Documentation

4.5.2.1 unsigned long max_num_token_pass_try

number of token attempts to pass

4.5.2.2 unsigned long max_token_holding_time

token holding time

4.5.2.3 unsigned long max_token_rotation_time

token rotation time

4.5.2.4 unsigned short user_max_num_node

number of nodes in ring

4.6 station_struct Struct Reference

4.6.1 Detailed Description

Main data structure that contains WTRP node and ring information.

Data Fields

- `comm_timing_t comm_tmng`
Protocol timing structure.
- unsigned short `state`
Current state.
- unsigned short `last_state`
Previous state.
- unsigned short `last_last_state`
Previous previous state :(.
- int `is_selfring`
Flag for checking if currently in single node ring.
- int `was_selfring`
Flag for checking if previously in single node ring.
- int `was_was_selfring`
Flag for checking if previously in single node ring.
- unsigned char `TS` [TOKEN_ALEN]
MAC address of this node.
- unsigned char `PS` [TOKEN_ALEN]
MAC address of the previous node.
- unsigned char `NS` [TOKEN_ALEN]
MAC address of the next node.
- unsigned char `RA` [TOKEN_ALEN]
Ring address of this node(MAC address of the ring owner).
- unsigned long `num_token_pass_try`

No.

- token_type [last_tx_forward_token](#)
For retransmission in case no implicit acknowledgement.
- unsigned long [ul_parms](#) [NUM_UL_PARMS]
Protocol parameters to be read from the process command line or computed.
- int [freeze_my_ctable](#)
Flag used to preserve a topology information when it was stable.

QNX specific members

- int [fd](#)
File descriptor for access to orinoco driver.
- pid_t [ok_tx_pid](#)
Transmit done interrupt from ori.
- pid_t [data_rx_pid](#)
Receive packet interrupt from ori.
- pid_t [app_pid](#)
Application proxy id.

Sequence numbers.

The generation sequence number is the generation sequence number of the last token accepted.

A station may increment genseq number when transmitting the token, but without modification of station's own genseq number. The same applies to sequence number

- unsigned long [seq](#)
Sequence Number of this node.
- unsigned long [genseq](#)
Generation Sequence Number of this node.

Protocol timers

- timer_list [solicit_wait_timer](#)
Wait for solicitor timer.

- timer_list [contention_timer](#)
Contention timer.
- timer_list [claim_token_timer](#)
Claim token timer.
- timer_list [solicit_successor_timer](#)
Solicit successor timer.
- timer_list [idle_timer](#)
Idle timer.
- timer_list [offline_timer](#)
Offline timer.
- timer_list [inring_timer](#)
Inring timer.
- timer_list [token_pass_timer](#)
Token pass timer.
- timer_list [token_holding_timer](#)
Token holding timer.

Connectivity table variables

- unsigned char [my_ctable](#) [NUM_TABLE_HISTORY][MAX_NUM_NODE][TOKEN_ALEN]
Connectivity table for this node.
- unsigned short [table_index](#)
points to the table that is being built
- unsigned short [my_node_index](#)
index of current nodes address in connectivity table
- unsigned char [other_ctable](#) [NUM_MONITOR_NODE][TOKEN_ALEN]
Connectivity table for another ring.
- unsigned short [other_node_index](#)
Index of another nodes address in connectivity table (used for joining/leaving ring).

Ring statistics

- unsigned short `num_node`
No.
- unsigned long `last_accepted`
For calculation of token rotation time.

Joining variables

- int `wants_to_join`
Flag for station wanting to join ring.
- unsigned char `soliciting_station` [TOKEN_ALEN]
MAC address of soliciting station.
- unsigned char `soliciting_station_successor` [TOKEN_ALEN]
MAC address of soliciting stations successor.

Transmission variables

- sk_buff_head `out_buffered_queue`
Data packets that are buffered for later transmission.
- sk_buff_head `out_tx_queue`
Packets that are ready to be transmitted, but blocked off due to hardware being busy.
- spinlock_t `out_queue_lock`
Spinlock for output transmission queue.
- sk_buff * `pending_skb_to_transmit`
Pointer to socket buffer that is next in line for transmission.

Timing variables

- timespec `begin_rotation_time`
Beginning of last token rotation time measurement.
- timespec `begin_processing_time`
Beginning of last processing time measurement.
- timespec `begin_join_time`

Beginning of last join time measurement.

- timespec `begin_db_read`
Beginning of last database read time measurement.
- timespec `begin_db_write`
Beginning of last database write time measurement.

4.6.2 Field Documentation

4.6.2.1 unsigned char my_ctable[NUM_TABLE_HISTORY][MAX_NUM_NODE][TOKEN_ALEN]

This array holds information about the transmission order of the own ring, the first entry of the my_ctable is the successor of this node

4.6.2.2 unsigned short num_node

of nodes in this ring

4.6.2.3 unsigned long num_token_pass_try

of times a station tries before giving up passing token and exiting

4.6.2.4 unsigned char other_ctable[NUM_MONITOR_NODE][TOKEN_ALEN]

This array holds information about transmissions from nodes not in the stations current ring, it is a circular buffer holding transmission records.

4.6.2.5 struct sk_buff_head out_tx_queue

(i.e. due to carrier sensing)

4.7 wtrp_config Struct Reference

4.7.1 Detailed Description

Inter-process message that contains I-P messaging configuration information that will be sent to orinoco driver process.

Data Fields

- `msg_t type`
Message type (WTRP_CONFIG).
- `pid_t tx_proxy`
Proxy called by ori to indicate ready to transmit.
- `pid_t rx_proxy`
Proxy called by ori to indicate packet recieved.

4.8 wtrp_config_reply Struct Reference

4.8.1 Detailed Description

Inter-process message that contains I-P communication configuration information sent back from the orinoco driver process.

Data Fields

- `msg_t type`
Message type (WTRP_CONFIG_REPLY).
- unsigned char `mac_addr` [ETH_ALEN]
MAC address of orinoco card.

5 WTRP for QNX File Documentation

5.1 app.c File Reference

5.1.1 Detailed Description

The application program is the interface between the orinoco/wtrp communication processes and the OS/system processes. In the case of the PATH automated control code,

this application has two main tasks. The first task is to parse any recieved packet, and to place the control packet information into the proper database variables. The second task is to periodically (at a specified sample rate) read a set of vehicle information from the database, form a control packet, and initiate the packets transmission through the orinoco/wtrp processes.

While the current application code has been written for this sole purpose, it can easily be modified to provide other functionality (such as interfacing with the standard TCP/IP stack) through a set of function hooks. To add other functionality, the following four functions should be overwritten:

- int `app_rx`(unsigned char*, unsigned int): called when a WTRP data packet has been received.
- int `app_msg`(pid_t sender_id): called when an application specific proxy received.
- pid_t `app_setup`(struct station_struct*): setup any application specific structures/code/etc.
- int `app_cleanup`(struct station_struct*): cleanup any application specific structures/code/etc.

To send a data packet through the orinoco/wtrp processes, use the function call `app_tx(station, data, data_len)` where:

- station(struct station_struct*): pointer to station structure
- data(unsigned char*): pointer to data array to put in packet
- data_len(unsigned int): length of data array

Author:

A. Howell

Defines

- #define `MSEC` 1E+06
Conversion from nanosec to millisec.

Functions

- int `app_rx` (unsigned char *inpacket, unsigned long inpacket_len)
WTRP reception hook.
- pid_t `app_setup` (struct station_struct *pst)
Application specific initialization hook.

- int `app_msg` (pid_t sender_pid)
Application-specific message processing hook.
- int `app_cleanup` (struct `station_struct` *pst)
Application specific clean up hook.

Variables

- private_t `ctrl_comm`
Instantiation of private structure for application specific code.

5.1.2 Function Documentation

5.1.2.1 int `app_cleanup` (struct `station_struct` * *pst*)

Clean up any application specific resources. This currently entails logging out of the PATH database only.

Parameters:

pst pointer to WTRP station

5.1.2.2 int `app_msg` (pid_t *sender_pid*)

Once a message has been received, this function is called to process it based on the PID of the sender. Currently, only transmission based on receiving a proxy is implemented.

Parameters:

sender_pid PID of process that sent message (not used currently)

5.1.2.3 int `app_rx` (unsigned char * *inpacket*, unsigned long *inpacket_len*)

Called when a WTRP data packet has been received.

Parameters:

inpacket pointer to received packet data array

inpacket_len length of the received packet

5.1.2.4 pid_t app_setup (struct station_struct * pst)

Runs any necessary initialization routines for the application specific code. This includes the following tasks:

- Creation of a timer/proxy pair to send control packets every 100 msec (for now).
- Logging in to the PATH database and creating database variables (if not already there).

Parameters:

pst pointer to WTRP station

Returns:

PID of process for messages to be processed using `app_msg()`

5.2 ctable.c File Reference

5.2.1 Detailed Description

This file provides a set of functions to support the connectivity tables (ctables) for the WTRP protocol.

Author:

A. Howell , Jeff Ko , Duke Lee

Functions

- int `search_other_ctable` (struct `station_struct` *station, unsigned char *address)
Search the stations other_ctable for the given address.
- void `update_other_ctable` (struct `station_struct` *station, unsigned char *packet)
Copy the source address of the given packet into the stations other_ctable.
- int `update_my_ctable` (struct `station_struct` *station, unsigned char FC, unsigned char *RA, unsigned char *DA, unsigned char *SA, unsigned long genseq, unsigned long seq, int forward_moving, int retransmission)
Updates this nodes ctable based on the given token information.

- void `print_my_htable` (struct `station_struct` *station)
Print my_htable to kernel log.
- unsigned char * `next_my_htable` (struct `station_struct` *station, unsigned char *address)
Get the next address in node ctable.
- int `index_from_my_htable` (struct `station_struct` *station, unsigned char *address)
Get the index of the given address from the nodes ctable.

5.2.2 Function Documentation

5.2.2.1 int `index_from_my_htable` (struct `station_struct` * station, unsigned char * address)

Returns:

The index of the element with the address from my_htable if it exists, otherwise return ERROR

5.2.2.2 unsigned char* `next_my_htable` (struct `station_struct` * station, unsigned char * address)

Returns:

If the given address is at the end of the table or the connectivity table is uninitialized return this station's address, if the given address is not in the table return NULL, otherwise return the next address.

5.2.2.3 int `search_other_htable` (struct `station_struct` * station, unsigned char * address)

Returns:

TRUE if the address is in the other_htable, FALSE otherwise

5.2.2.4 int `update_my_ctable` (struct `station_struct` * *station*, unsigned char *FC*, unsigned char * *RA*, unsigned char * *DA*, unsigned char * *SA*, unsigned long *genseq*, unsigned long *seq*, int *forward_moving*, int *retransmission*)

Returns:

ERROR if loop in ctable detected, RESET if token is not forward moving and a claim token message is received, and SUCCESS otherwise.

5.3 lib.c File Reference**5.3.1 Detailed Description**

This file contains a set of general utility functions used by the protocol code.

Author:

Duke Lee , Mustafa Ergen , Jeff Ko

Integer encoding/decoding functions

- void `encode_ushort` (unsigned char *bytes, unsigned short n)
- void `encode_ulong` (unsigned char *bytes, unsigned long n)
- unsigned short `decode_ushort` (const unsigned char *bytes)
- unsigned long `decode_ulong` (const unsigned char *bytes)

Randomization functions

- unsigned long `net_random` (void)
- void `net_srandom` (unsigned long entropy)
- unsigned long `tok_random` (unsigned char *addr)
- int `flip_biased_coin` (int bias, unsigned long random)

Address related functions

- int `is_broadcast_addr` (unsigned char *addr)
- void `printaddr` (unsigned char *addr)
- void `_printaddr` (unsigned char *addr)
- void `copyaddr` (unsigned char *dest, unsigned char *src)
- void `makenulladdr` (unsigned char *addr)
- void `initaddr` (unsigned char *addr)
- int `is_null_addr` (unsigned char *addr)

- void **sprintaddr** (unsigned char *buffer, unsigned char *addr)
- int **prioaddr** (unsigned char *first, unsigned char *second)
- void **make_broadcast** (unsigned char *addr)

Math functions

- unsigned long **diff** (unsigned long a, unsigned long b)
- int **is_lesseq** (unsigned long a, unsigned long b)
- int **is_leqseq** (unsigned long a, unsigned long b)
- unsigned long **next_seq** (unsigned long seq)

WTRP header printing/debugging functions

- void **get_header_info** (struct [station_struct](#) *station, unsigned char *inpacket, unsigned char *FC, unsigned char **pRA, unsigned char **pDA, unsigned char **pSA, unsigned char **pNS, unsigned short *pnnum_node, unsigned long *pseq, unsigned long *pgenseq, unsigned short *pnetwork_proto, unsigned char *pusing_checksum, unsigned int *pchecksum)

Print WTRP header information for both control and data packets given a packet with only WTRP encapsulation.

- void **validate_eth_header** (struct [station_struct](#) *station, unsigned char *eth)
Verify correctness of 802.2 ethernet header.
- void **DEBUG_HEADER** (struct net_device *dev, unsigned char *outpacket, int size)
Print WTRP header information given a raw packet (ETH+WTRP headers).

Functions

- int **is_ownpacket** (struct [station_struct](#) *station, struct sk_buff *skb)
- void **printstate** (struct [station_struct](#) *station)

5.4 params.c File Reference

5.4.1 Detailed Description

This file provides a number of support functions that automatically compute protocol parameters based on a user-definable subset.

Author:

A. Howell , Duke Lee , Mustafa Ergen , Ruchira Datta

Parameter randomization functions

All randomization functions use the station address combined with the current clock time as the seed.

- unsigned long **randomize_token_pass_timer** (struct [station_struct](#) *station)
- unsigned long **randomize_claim_token_timer** (struct [station_struct](#) *station)
- unsigned long **randomize_solicit_successor_timer** (struct [station_struct](#) *station)
- unsigned long **randomize_idle_timer** (struct [station_struct](#) *station)
- unsigned long **randomize_inring_timer** (struct [station_struct](#) *station)
- unsigned long **randomize_offline_timer** (struct [station_struct](#) *station)

Parameter bounding functions

- unsigned long **upper_claim_token_tx_time** (struct [station_struct](#) *station)
- unsigned long **upper_proc_time** (struct [station_struct](#) *station, unsigned char *addr)
- unsigned long **upper_token_holding_time** (struct [station_struct](#) *station, unsigned char *addr)
- unsigned long **upper_total_token_holding_time** (struct [station_struct](#) *station)
- unsigned long **upper_total_management_time** (struct [station_struct](#) *station)
- unsigned long **max_token_pass_time** (struct [station_struct](#) *station)
- unsigned long **min_token_pass_time** (struct [station_struct](#) *station)
- unsigned long **MTRT** (struct [station_struct](#) *station)
- unsigned long **max_idle_time** (struct [station_struct](#) *station)
- unsigned long **max_solicit_successor_time** (struct [station_struct](#) *station)
- unsigned long **upper_bw** (struct [station_struct](#) *station)
- unsigned long **lower_bw** (struct [station_struct](#) *station)

Parameter calculation functions

- unsigned long **TRT** (struct [station_struct](#) *station)
- unsigned long **inring_time** (struct [station_struct](#) *station)
- unsigned long **offline_time** (struct [station_struct](#) *station)
- unsigned long **contention_time** (struct [station_struct](#) *station)
- unsigned long **claim_token_time** (struct [station_struct](#) *station)
- unsigned long **solicit_interval** (struct [station_struct](#) *station)

Defines

- #define **MAX_NUM_ERRORS** 2

Functions

- unsigned long **round_ul_parm_value** (unsigned long value, int i)
Round parameters to the jiffy.
- int **wtrp_is_ul_parm_set_by_user** (int i)
Determine if the parameter the parameter is user-definable.
- void **calculate_params** (struct **station_struct** *station)
Compute all non-configurable parameters based on all current parameter values.
- void **get_params** (struct **station_struct** *station)
Compute, print, and check all parameters.
- void **check_params** (struct **station_struct** *station)
- void **print_params** (struct **station_struct** *station)
Print all parameters to kernel log.

5.4.2 Function Documentation

5.4.2.1 int wtrp_is_ul_parm_set_by_user (int i)

Parameters:

i Parameter index in wtrp_ul.

Returns:

TRUE if the parameter can be set by the user, FALSE otherwise.

5.5 tokenring.c File Reference

5.5.1 Detailed Description

This file is the actual tokenring protocol implementation. The code is a modified version of a hybrid system model developed using Teja software tool.

Some notes about the protocol:

- Genseq and seq are initialized to be 0, and 0 is reserved to be the initial state.
- Token generation is equivalent to the station receiving a token.
- The genseq and seq numbers pertain to the last token accepted, NOT the genseq and seq number of the last token transmitted.
- Implicit acknowledgements are any transmission that has the ring address of the current node, or ring address of any node in connectivity table.

Author:

A. Howell , Duke Lee , Ruchira Datta

Station transmission queue functions

- void `clean_tx_queue` (struct `station_struct` *station)
Remove all tokens from the tx queue.
- void `station_tx_enqueue` (struct `station_struct` *station, struct `sk_buff` *skb)
Append skb to end of tx queue.
- `sk_buff` * `station_tx_dequeue` (struct `station_struct` *station)
Pull first skb from tx queue.
- void `station_buffered_enqueue` (struct `station_struct` *station, struct `sk_buff` *skb, int len)
Append skb to end of buffered tx queue.
- `sk_buff` * `station_buffered_dequeue` (struct `station_struct` *station)
Pull first skb from buffered tx queue.
- void `transmit_tx_queue` (struct `station_struct` *station)
Transmit packet in tx queue.
- void `transmit_buffered_queue` (struct `station_struct` *station)
Transmit packet in buffered tx queue.

Timer functions

All state-specific timer functions assume that the station is in the appropriate state (i.e. in `idle_state` to call `reset_idle` timers).

- void `reset_idle_timers` (struct `station_struct` *station, unsigned char *RA, unsigned char *SA, unsigned char *DA, int addressed_to_me)
Reset all timers associated with idle_state.
- void `delete_idle_timers` (struct `station_struct` *station)
Delete all timers for the idle state.
- void `delete_have_token_timers` (struct `station_struct` *station)
Delete all timers for have_token_state.
- void `delete_floating_timers` (struct `station_struct` *station)
Delete all timers for floating_state.
- void `delete_offline_timers` (struct `station_struct` *station)
Delete all timers for offline_state.
- void `delete_solicit_timers` (struct `station_struct` *station)
Delete all timers for soliciting_state.
- void `delete_monitoring_timers` (struct `station_struct` *station)
Delete all timers for monitoring_state.
- void `delete_joining_timers` (struct `station_struct` *station)
Delete all timers for joining_state.
- void `delete_state_timer` (struct `station_struct` *station)
Delete all timers for current state.

State transition functions

All `is_*` functions return true if the station is in the specified state, while all `was_*` functions return true if the station was just in the specified state.

For all `go_*` functions the procedure is to update station parameters, attach the new state timers, and transition to the new state. Furthermore, it is assumed that all station state-timers are detached before being called.

- int `is_floating` (struct `station_struct` *station)
Determine if the station is in the floating state.
- int `is_monitoring` (struct `station_struct` *station)
Determine if the station is in the monitoring state.

- int `is_idle` (struct `station_struct` *station)
Determine if the station is in the idle state.
- int `is_soliciting` (struct `station_struct` *station)
Determine if the station is in the soliciting state.
- int `was_soliciting` (struct `station_struct` *station)
Determine if the station was just in the soliciting state.
- int `was_selfring` (struct `station_struct` *station)
Determine if the station was just in a self ring.
- int `is_joining` (struct `station_struct` *station)
Determine if the station is in the joining state.
- int `was_joining` (struct `station_struct` *station)
Determine if the station was just in the joining state.
- int `is_offline` (struct `station_struct` *station)
Determine if the station is in the offline state.
- void `record_states` (struct `station_struct` *station)
Record all prior state information.
- void `go_floating` (struct `station_struct` *station)
Transition to floating state.
- void `go_idle` (struct `station_struct` *station)
Transition to idle state.
- void `go_soliciting` (struct `station_struct` *station)
Transition to soliciting state.
- void `go_monitoring` (struct `station_struct` *station)
Transition to monitoring state.
- void `go_have_token` (struct `station_struct` *station)
Transition to have_token state.
- void `go_joining` (struct `station_struct` *station)
Transition to joining state.

- void `go_offline` (struct `station_struct` *station)

Transition to offline state.

Transmission utilities.

- int `tx_normal_token` (struct `station_struct` *station, unsigned char *RA, unsigned char *DA, unsigned char *SA, unsigned short num_node, unsigned long genseq, unsigned long seq)

Transmit normal token control packet.

- int `tx_token_deleted` (struct `station_struct` *station, unsigned char *RA, unsigned char *SA, unsigned char *DA)

Transmit token_deleted packet.

- int `tx_set_successor` (struct `station_struct` *station, unsigned char *RA, unsigned char *DA, unsigned char *SA, unsigned char *NS)

Transmit set_successor packet.

- int `tx_set_predecessor` (struct `station_struct` *station, unsigned char *RA, unsigned char *DA, unsigned char *SA, unsigned short num_node, long genseq, long seq)

Transmit set_predecessor packet.

- int `tx_solicit_successor` (struct `station_struct` *station, unsigned char *RA, unsigned char *SA, unsigned char num_node, unsigned char *NS)

Transmit solicit_successor packet.

- int `tx_claim_token` (struct `station_struct` *station, unsigned char *RA, unsigned char *SA)

Transmit claim_token packet.

Handler functions

- void `offline_timer_handler` (struct `station_struct` *station)

Handle offline_timer expiration.

- void `claim_token_handler` (struct `station_struct` *station)

Handle claim_token_timer expiration.

- void `solicit_successor_handler` (struct `station_struct` *station)

Handle solicit_successor_timer expiration.

- void `solicit_wait_handler` (struct `station_struct` *station)
Handle solicit_wait_timer expiration.
- void `token_holding_timer_handler` (struct `station_struct` *station)
Handle token_holding_timer expiration.
- void `idle_timer_handler` (struct `station_struct` *station)
Handle idle_timer expiration.
- void `inring_timer_handler` (struct `station_struct` *station)
Handle inring_timer expiration.
- int `tok_tx_handler` (struct `station_struct` *station, struct `sk_buff` *skb)
Handle request from wtrp process code to transmit data.
- void `token_pass_timer_handler` (struct `station_struct` *station)
Handle token_pass_timer expiration.
- void `contention_timer_handler` (struct `station_struct` *station)
Handle contention_timer expiration.
- void `tx_done_handler` (struct `station_struct` *station)
Handle completion of packet transmission to the driver process.

Defines

- #define `__NO_VERSION__`
- #define `TOKENRING_C`
- #define `jiffies` 0

Functions

- void `set_ethernet_header` (unsigned char *SA, unsigned char **dptrptr)
Fill the ethernet header correctly for the WTRP protocol (correct addresses, etc.).
- void `enforce_no_timer_pending` (struct `station_struct` *station)
ASSERT that no timers of this station is pending.
- void `check_skb` (struct `sk_buff` *skb)

Checks conditions on skb to detect any errors.

- void `check_station_conditions` (struct `station_struct` *station)
Checks conditions on station structure to detect any errors.
- int `is_have_token` (struct `station_struct` *station)
Determine if the node has the token.
- int `is_selfring` (struct `station_struct` *station)
Determine if the node is a self ring.
- void `set_station_offline` (struct `station_struct` *station)
Set station settings for offline state.
- unsigned long `next_genseq` (struct `station_struct` *station)
Return the generation sequence number of the transmitted token.
- unsigned long `updated_num_node` (struct `station_struct` *station, unsigned long num_node, unsigned long seq)
Return the estimated number of nodes.
- int `is_implicitack` (struct `station_struct` *station, unsigned char *RA)
Determine if received token is implicit acknowledgement.
- void `make_selfring` (struct `station_struct` *station)
Create a one node ring.
- int `decide_to_solicit_successor` (struct `station_struct` *station)
Make a decision to solicit successors based on token rotation time.
- void `process_packet` (struct `station_struct` *station, unsigned char *inpacket, unsigned int inpacket_len)
Process a packet received by the wtrp process.
- void `init_station` (struct `station_struct` *station, unsigned char *addr)
Initialize the station structure.

5.5.2 Function Documentation

5.5.2.1 void check_skb (struct sk_buff * skb)

Checks conditions on skb to detect any errors.

5.5.2.2 void check_station_conditions (struct station_struct * station)

Checks conditions on station structure to detect any errors.

5.5.2.3 void claim_token_handler (struct station_struct * station)

Indicates there is no station found on the medium. The token is claimed, and the station transitions to the idle state.

5.5.2.4 void clean_tx_queue (struct station_struct * station)

Remove all tokens from the transmission queue.

5.5.2.5 void contention_timer_handler (struct station_struct * station)

Indicates that the node did not receive a set_predecessor packet from the soliciting node during the contention period. It is assumed that the node has lost the contention process, so transition to the floating state.

5.5.2.6 void idle_timer_handler (struct station_struct * station)

Indicates there is no activity on the medium (token must be lost), so claim a token by creating a new normal token and become a new ring owner.

5.5.2.7 void inring_timer_handler (struct station_struct * station)

Indicates that there is activity in the ring (idle_timer did not expires), but the node has not received a token for a while. In this case, kick out of the ring and transition to the floating state.

5.5.2.8 int is_have_token (struct station_struct * station)**Returns:**

One if the station has the token, zero otherwise.

5.5.2.9 int is_implicitack (struct station_struct * station, unsigned char * RA)

Returns:

TRUE if RA field of received token corresponds to implicit acknowledgement for the station.

5.5.2.10 int is_selfring (struct station_struct * station)**Returns:**

One if the station is a self ring, zero otherwise.

5.5.2.11 unsigned long next_genseq (struct station_struct * station)**Returns:**

If owner of the token, 1 + genseq of station is returned. Otherwise, the returned genseq number is the same as the station.

5.5.2.12 void offline_timer_handler (struct station_struct * station)

Indicates the node is able to join a ring again.

5.5.2.13 void set_station_offline (struct station_struct * station)

Set station settings for offline state by initializing connectivity tables and zeroing ring statistics.

5.5.2.14 void solicit_successor_handler (struct station_struct * station)

Indicates that the station has the token and is willing to invite other nodes to join its ring. It is also called directly to solicit successors. A solicit successor packet is transmitted and the station transitions to the soliciting state.

5.5.2.15 void solicit_wait_handler (struct station_struct * station)

Indicates the station was not successful in soliciting. If the station is not a self ring, then pass the token to the successor and transition to the monitoring state. If the node is a self ring, transition to the idle state.

5.5.2.16 struct sk_buff* station_buffered_dequeue (struct station_struct * station)

Pull the first socket buffer from the buffered transmission queue.

Returns:

Pointer to the pulled skb.

5.5.2.17 void station_buffered_enqueue (struct station_struct * station, struct sk_buff * skb, int len)

Append socket buffer to the end of the buffered transmission queue.

5.5.2.18 struct sk_buff* station_tx_dequeue (struct station_struct * station)

Pull the first socket buffer from the transmission queue.

Returns:

Pointer to the pulled skb.

5.5.2.19 void station_tx_enqueue (struct station_struct * station, struct sk_buff * skb)

Append the socket buffer to the end of the transmission queue.

5.5.2.20 int tok_tx_handler (struct station_struct * station, struct sk_buff * skb)

If the node has the token, any queued packets are sent first, followed by the input skb. If the node does not have the token, the input skb is put on the buffered transmission queue.

5.5.2.21 void token_holding_timer_handler (struct station_struct * station)

Indicates the allotted token transmission time has passed before being able to transmit all the data, so pass the token and transition to the idle state.

5.5.2.22 void token_pass_timer_handler (struct station_struct * station)

Indicates that no implicit acknowledgement was received after sending forward moving token. It is assumed that the token is lost, and the last token is retransmitted. If the

successor is unresponsive for two consecutive retransmissions, then transmit the token to the next nodes in the ring. Finally, if the node is unsuccessful in passing the token for `num_token_pass_try` times, give up on passing token and go offline.

5.5.2.23 void transmit_buffered_queue (struct [station_struct](#) * station)

Send latest packet in transmission queue to driver using the wtrp process transmit hook. Assumes that `out_buffered_queue` is not empty, and `out_tx_queue` is empty.

5.5.2.24 void transmit_tx_queue (struct [station_struct](#) * station)

Send latest packet in transmission queue to driver using the wtrp process transmit hook. Assumes that `out_tx_queue` is not empty.

5.5.2.25 void tx_done_handler (struct [station_struct](#) * station)

If the node has the token, continue to transmit if there are more to be transmitted in `out_tx_queue`.

5.5.2.26 unsigned long updated_num_node (struct [station_struct](#) * station, unsigned long *num_node*, unsigned long *seq*)

Returns:

If owner of the ring, recalculate the number of nodes by subtracting the sequence number of this rotation and last accepted token, otherwise return the `num_node` reported by the previous node.

5.6 wtrp.c File Reference

5.6.1 Detailed Description

This file contains the main QNX code for the wtrp process. Primarily, the functions within provide inter-process communication support between the orinoco driver (`ori`) and the real tokenring protocol code (in [tokenring.c](#)).

Author:

A. Howell

Timing analysis functions

- void [begin_timing_record](#) (struct timespec *begin)

Begin a timing record.

- void `end_timing_record` (struct timespec *begin, stats_t *pstat)
End a timing record, and update timing stats.

Packet transmission functions

- int `app_tx` (struct `station_struct` *station, unsigned char *data, unsigned long data_len)
Transmission request from application code.
- int `transmit` (struct `station_struct` *pst, struct sk_buff *skb)
Send transmitted packet to orinoco driver.

Protocol setup/cleanup functions

- int `wtrp_register` (station_t *pst)
Setup WTRP process upon creation.
- int `wtrp_unregister` (station_t *pst)
Cleanup WTRP process resources.
- void `proxy_setup` (station_t *pst)
Setup proxies associated with WTRP process.

User-definable protocol parameters

Default user definable protocol parameters

- long `processing_time` = 22
- long `max_num_token_pass_try` = 3
- long `solicit_successor_prob` = 1
- long `max_token_holding_time` = 28
- long `transmission_rate` = 70190
- long `max_token_rotation_time` = 307

Functions

- int `wtrp_set_ul_parm` (struct `station_struct` *station, int i, unsigned long parm)
Set a user-definable WTRP parameter.
- int `is_destined_for_me` (int addressed_to_me, unsigned char *DA)
Checks if packet destination is addressed to this node.
- void `notify_mac_busy` (struct `station_struct` *station)
Not used.
- void `notify_mac_available` (struct `station_struct` *station)
Not used.
- void `process_message` (station_t *pst, pid_t sender_id)
Inter-process message handling.
- int `main` (int argc, char *argv[])
Main loop of the WTRP process.

5.6.2 Function Documentation

5.6.2.1 int `app_tx` (struct `station_struct` * station, unsigned char * data, unsigned long data_len)

This function accepts the raw data to be transmitted from the application specific code, encapsulates it with 802.2 and WTRP DATA headers, then passes the resulting socket buffer to the tokenring transmission handler.

Parameters:

- station* Pointer to tokenring station.
- data* Pointer to raw data from application code.
- data_len* Length of data array.

Returns:

- Return value of `tok_tx_handler`.

5.6.2.2 void begin_timing_record (struct timespec * begin)

This function records the beginning of a timing record is received

5.6.2.3 void end_timing_record (struct timespec * begin, stats_t * pstat)

This function records the end of a timing record, and update the appropriate stats using update_stats()

Parameters:

- pstat* Pointer to comm_tmng stat to update
- begin* Output of begin_timing_record

5.6.2.4 int is_destined_for_me (int addressed_to_me, unsigned char * DA)

This function determines if a packet is addressed for the current station (seems unnecessary).

Returns:

- One if true

5.6.2.5 int main (int argc, char * argv[])

This function implements the main loop of the WTRP process. The process runs the initialization functions, then waits in a Receive-blocked state until an inter-process message is received.

See also:

- [proxy_setup](#) , [wtrp_register](#) , [app_setup](#) , [process_message](#) , [wtrp_unregister](#)

5.6.2.6 void notify_mac_available (struct station_struct * station)

Tell the physical layer that the datalink layer is ready (not used)

5.6.2.7 void notify_mac_busy (struct station_struct * station)

Tell the physical layer that the datalink layer is busy (not used)

5.6.2.8 void process_message (station_t * *pst*, pid_t *sender_id*)

This function is the main handler for received inter-process messages. Essentially, it handles two types of messages, timer proxies, application specific proxies, and proxies called by the orinoco driver process.

When a timer proxy is received, the timers associated function handler is called.

When an application specific proxy is received, the hook [app_msg\(\)](#) is called to handle the event.

Finally, the proxies from the driver process are used to control the flow of transmitted/received packets between wtrp and the orinoco driver. An `ok_tx` proxy is received once the orinoco card is available for accepting packets to transmit. The `tx_done_` handler is called to notify wtrp of this event. The `data_rx` proxy is received to notify wtrp that a packet has been received by the card, and is available for processing. In response, the wtrp process sends an inter-process message to request the packet from the driver, and the received packet is passed to the wtrp process, and subsequently the tokenring processing, in the inter-process reply message.

Parameters:

- pst* Pointer to tokenring station.
- sender_id* PID of sending process.

See also:

[process_packet](#) , [tx_done_handler](#)

5.6.2.9 void proxy_setup (station_t * *pst*)

This function attaches a set of proxies used by the WTRP process. This includes a set of proxies attached to protocol-related timers and proxies for handling inter-process communication.

Parameters:

- pst* Pointer to tokenring station.

5.6.2.10 int transmit (struct [station_struct](#) * *pst*, struct [sk_buff](#) * *skb*)

This function passes transmitted packet to the orinoco driver process using QNX inter-process messaging. Processing time is also estimated.

Parameters:

- pst* Pointer to tokenring station.

skb Pointer to socket buffer to be transmitted.

Returns:

Returns one on failure to transmit for any reason, and zero for success.

5.6.2.11 int wtrp_register (station_t * pst)

This function performs several initialization tasks once the wtrp process is started. These tasks include:

- Set user-definable parameters.
- Initialize inter-process communication between ori and wtrp processes.
- Initialize tokenring station structure.

Parameters:

pst Pointer to tokenring station.

See also:

[wtrp_set_ul_parm](#) , [init_station](#)

5.6.2.12 int wtrp_set_ul_parm (struct station_struct * station, int i, unsigned long parm)

This function sets a user-definable WTRP protocol parameter in the given station struct.

Parameters:

station Pointer to tokenring station.

i Index of parameter in ul_parms array.

parm Desired value of the parameter.

See also:

[wtrp_ul](#) , [station_struct](#)

5.6.2.13 int wtrp_unregister (station_t * pst)

This function cleans up the WTRP processes allocated resources. This includes writing timing information to file and removing allocated timers.

Parameters:

pst Pointer to tokenring station.

5.6.3 Variable Documentation

5.6.3.1 long max_num_token_pass_try = 3

Max. number of tries to pass token.

5.6.3.2 long max_token_holding_time = 28

Max. token holding time (jiffies).

5.6.3.3 long max_token_rotation_time = 307

Max. token rotation time (jiffies).

5.6.3.4 long processing_time = 22

Processing time (jiffies).

5.6.3.5 long solicit_successor_prob = 1

Probability of soliciting a successor.

5.6.3.6 long transmission_rate = 70190

Transmission rate (bytes/jiffy).

Index

`__NO_VERSION__`
 tokenring.c, 30

`_printaddr`
 lib.c, 22

`app.c`, 17
 `app_cleanup`, 18
 `app_msg`, 19
 `app_rx`, 19
 `app_setup`, 19
 `ctrl_comm`, 18
 MSEC, 18

`app_cleanup`
 app.c, 18

`app_msg`
 app.c, 19

`app_pid`
 station_struct, 13

`app_rx`
 app.c, 19

`app_setup`
 app.c, 19

`app_tx`
 wtrp.c, 37

`begin_db_read`
 station_struct, 15

`begin_db_write`
 station_struct, 15

`begin_join_time`
 station_struct, 15

`begin_processing_time`
 station_struct, 15

`begin_rotation_time`
 station_struct, 15

`begin_timing_record`
 wtrp.c, 37

`calculate_params`
 params.c, 24

`check_params`
 params.c, 25

`check_skb`
 tokenring.c, 31

`check_station_conditions`
 tokenring.c, 31

`claim_token_handler`
 tokenring.c, 31

`claim_token_time`
 params.c, 24

`claim_token_timer`
 station_struct, 13

`clean_tx_queue`
 tokenring.c, 31

`close`
 message_t, 8

`clt`
 private, 10

`comm_tmng`
 station_struct, 11

`config`
 message_t, 9

`contention_time`
 params.c, 24

`contention_timer`
 station_struct, 13

`contention_timer_handler`
 tokenring.c, 32

`copyaddr`
 lib.c, 22

`ctable.c`, 20
 `index_from_my_ctable`, 20
 `next_my_ctable`, 21
 `print_my_ctable`, 20
 `search_other_ctable`, 21
 `update_my_ctable`, 21
 `update_other_ctable`, 20

`ctrl_comm`
 app.c, 18

`ctrl_packet`, 7
 dummy_float, 8
 dummy_int, 8
 dummy_short, 8
 timestamp, 8

`data`

packet, 9

data_rx_pid
station_struct, 13

DEBUG_HEADER
lib.c, 23

decide_to_solicit_successor
tokenring.c, 31

decode_ulong
lib.c, 22

decode_ushort
lib.c, 22

delete_floating_timers
tokenring.c, 26

delete_have_token_timers
tokenring.c, 26

delete_idle_timers
tokenring.c, 26

delete_joining_timers
tokenring.c, 27

delete_monitoring_timers
tokenring.c, 27

delete_offline_timers
tokenring.c, 27

delete_solicit_timers
tokenring.c, 27

delete_state_timer
tokenring.c, 27

diff
lib.c, 22

dummy_float
ctrl_packet, 8

dummy_int
ctrl_packet, 8

dummy_short
ctrl_packet, 8

encode_ulong
lib.c, 22

encode_ushort
lib.c, 22

end_timing_record
wtrp.c, 37

enforce_no_timer_pending
tokenring.c, 30

fd

station_struct, 12

flip_biased_coin
lib.c, 22

freeze_my_ctable
station_struct, 12

genseq
station_struct, 13

get_header_info
lib.c, 22

get_params
params.c, 24

go_floating
tokenring.c, 28

go_have_token
tokenring.c, 28

go_idle
tokenring.c, 28

go_joining
tokenring.c, 28

go_monitoring
tokenring.c, 28

go_offline
tokenring.c, 28

go_soliciting
tokenring.c, 28

idle_timer
station_struct, 13

idle_timer_handler
tokenring.c, 32

index_from_my_ctable
ctable.c, 20

init_station
tokenring.c, 31

initaddr
lib.c, 22

inring_time
params.c, 24

inring_timer
station_struct, 14

inring_timer_handler
tokenring.c, 32

is_broadcast_addr
lib.c, 22

is_destined_for_me

- wtrp.c, 37
- is_floating
 - tokenring.c, 27
- is_have_token
 - tokenring.c, 32
- is_idle
 - tokenring.c, 27
- is_implicitack
 - tokenring.c, 32
- is_joining
 - tokenring.c, 28
- is_leqseq
 - lib.c, 22
- is_lessseq
 - lib.c, 22
- is_monitoring
 - tokenring.c, 27
- is_null_addr
 - lib.c, 22
- is_offline
 - tokenring.c, 28
- is_ownpacket
 - lib.c, 23
- is_selfring
 - station_struct, 12
 - tokenring.c, 32
- is_soliciting
 - tokenring.c, 27
- jiffies
 - tokenring.c, 30
- last_accepted
 - station_struct, 14
- last_last_state
 - station_struct, 12
- last_state
 - station_struct, 12
- last_tx_forward_token
 - station_struct, 12
- len
 - packet, 9
- lib.c, 21
 - _printaddr, 22
 - copyaddr, 22
 - DEBUG_HEADER, 23
 - decode_ulong, 22
 - decode_ushort, 22
 - diff, 22
 - encode_ulong, 22
 - encode_ushort, 22
 - flip_biased_coin, 22
 - get_header_info, 22
 - initaddr, 22
 - is_broadcast_addr, 22
 - is_leqseq, 22
 - is_lessseq, 22
 - is_null_addr, 22
 - is_ownpacket, 23
 - make_broadcast, 22
 - makenulladdr, 22
 - net_random, 22
 - net_srandom, 22
 - next_seq, 22
 - printaddr, 22
 - printstate, 23
 - prioaddr, 22
 - sprintaddr, 22
 - tok_random, 22
 - validate_eth_header, 23
- lower_bw
 - params.c, 24
- mac_addr
 - wtrp_config_reply, 17
- main
 - wtrp.c, 37
- make_broadcast
 - lib.c, 22
- make_selfring
 - tokenring.c, 31
- makenulladdr
 - lib.c, 22
- max_idle_time
 - params.c, 24
- MAX_NUM_ERRORS
 - params.c, 24
- max_num_token_pass_try
 - qos_struct, 11
 - wtrp.c, 40
- max_solicit_successor_time
 - params.c, 24

- max_token_holding_time
 - qos_struct, 11
 - wtrp.c, 40
- max_token_pass_time
 - params.c, 24
- max_token_rotation_time
 - qos_struct, 11
 - wtrp.c, 40
- message_t, 8
 - close, 8
 - config, 9
 - open, 8
 - pack, 8
 - rclose, 8
 - rconfig, 9
 - ropen, 8
 - type, 8
- min_token_pass_time
 - params.c, 24
- MSEC
 - app.c, 18
- MTRT
 - params.c, 24
- my_ctable
 - station_struct, 15
- my_node_index
 - station_struct, 14
- net_random
 - lib.c, 22
- net_srandom
 - lib.c, 22
- next
 - qos_struct, 11
- next_genseq
 - tokenring.c, 32
- next_my_ctable
 - ctable.c, 21
- next_seq
 - lib.c, 22
- notify_mac_available
 - wtrp.c, 38
- notify_mac_busy
 - wtrp.c, 38
- NS
 - station_struct, 12
- num_node
 - station_struct, 15
- num_token_pass_try
 - station_struct, 16
- offline_time
 - params.c, 24
- offline_timer
 - station_struct, 13
- offline_timer_handler
 - tokenring.c, 33
- ok_tx_pid
 - station_struct, 13
- open
 - message_t, 8
- other_ctable
 - station_struct, 16
- other_node_index
 - station_struct, 14
- out_buffered_queue
 - station_struct, 15
- out_queue_lock
 - station_struct, 15
- out_tx_queue
 - station_struct, 16
- pack
 - message_t, 8
- packet, 9
 - data, 9
 - len, 9
 - type, 9
- params.c, 23
 - calculate_params, 24
 - check_params, 25
 - claim_token_time, 24
 - contention_time, 24
 - get_params, 24
 - inring_time, 24
 - lower_bw, 24
 - max_idle_time, 24
 - MAX_NUM_ERRORS, 24
 - max_solicit_successor_time, 24
 - max_token_pass_time, 24
 - min_token_pass_time, 24
 - MTRT, 24

- offline_time, 24
- print_params, 25
- randomize_claim_token_timer, 23
- randomize_idle_timer, 23
- randomize_inring_timer, 23
- randomize_offline_timer, 23
- randomize_solicit_successor_timer, 23
- randomize_token_pass_timer, 23
- round_ul_parm_value, 24
- solicit_interval, 24
- TRT, 24
- upper_bw, 24
- upper_claim_token_tx_time, 24
- upper_proc_time, 24
- upper_token_holding_time, 24
- upper_total_management_time, 24
- upper_total_token_holding_time, 24
- wtrp_is_ul_parm_set_by_user, 25
- pending_skb_to_transmit
 - station_struct, 15
- print_my_ctable
 - ctable.c, 20
- print_params
 - params.c, 25
- printaddr
 - lib.c, 22
- printstate
 - lib.c, 23
- prioaddr
 - lib.c, 22
- private, 9
 - clt, 10
 - rxpacket, 10
 - station, 10
 - tid, 10
 - tx_proxy, 10
 - txpacket, 10
- process_message
 - wtrp.c, 38
- process_packet
 - tokenring.c, 31
- processing_time
 - wtrp.c, 40
- proxy_setup
 - wtrp.c, 38
- PS
 - station_struct, 12
- qos_struct, 10
 - max_num_token_pass_try, 11
 - max_token_holding_time, 11
 - max_token_rotation_time, 11
 - next, 11
 - solicit_successor_prob, 10
 - user_max_num_node, 11
- RA
 - station_struct, 12
- randomize_claim_token_timer
 - params.c, 23
- randomize_idle_timer
 - params.c, 23
- randomize_inring_timer
 - params.c, 23
- randomize_offline_timer
 - params.c, 23
- randomize_solicit_successor_timer
 - params.c, 23
- randomize_token_pass_timer
 - params.c, 23
- rclose
 - message_t, 8
- rconfig
 - message_t, 9
- record_states
 - tokenring.c, 28
- reset_idle_timers
 - tokenring.c, 26
- ropen
 - message_t, 8
- round_ul_parm_value
 - params.c, 24
- rx_proxy
 - wtrp_config, 16
- rxpacket
 - private, 10
- search_other_ctable
 - ctable.c, 21

- seq
 - station_struct, 13
- set_ethernet_header
 - tokenring.c, 30
- set_station_offline
 - tokenring.c, 33
- solicit_interval
 - params.c, 24
- solicit_successor_handler
 - tokenring.c, 33
- solicit_successor_prob
 - qos_struct, 10
 - wtrp.c, 40
- solicit_successor_timer
 - station_struct, 13
- solicit_wait_handler
 - tokenring.c, 33
- solicit_wait_timer
 - station_struct, 13
- soliciting_station
 - station_struct, 14
- soliciting_station_successor
 - station_struct, 15
- sprintaddr
 - lib.c, 22
- state
 - station_struct, 11
- station
 - private, 10
- station_buffered_dequeue
 - tokenring.c, 33
- station_buffered_enqueue
 - tokenring.c, 33
- station_struct, 11
 - app_pid, 13
 - begin_db_read, 15
 - begin_db_write, 15
 - begin_join_time, 15
 - begin_processing_time, 15
 - begin_rotation_time, 15
 - claim_token_timer, 13
 - comm_tmg, 11
 - contention_timer, 13
 - data_rx_pid, 13
 - fd, 12
 - freeze_my_ctable, 12
 - genseq, 13
 - idle_timer, 13
 - inring_timer, 14
 - is_selfring, 12
 - last_accepted, 14
 - last_last_state, 12
 - last_state, 12
 - last_tx_forward_token, 12
 - my_ctable, 15
 - my_node_index, 14
 - NS, 12
 - num_node, 15
 - num_token_pass_try, 16
 - offline_timer, 13
 - ok_tx_pid, 13
 - other_ctable, 16
 - other_node_index, 14
 - out_buffered_queue, 15
 - out_queue_lock, 15
 - out_tx_queue, 16
 - pending_skb_to_transmit, 15
 - PS, 12
 - RA, 12
 - seq, 13
 - solicit_successor_timer, 13
 - solicit_wait_timer, 13
 - soliciting_station, 14
 - soliciting_station_successor, 15
 - state, 11
 - table_index, 14
 - token_holding_timer, 14
 - token_pass_timer, 14
 - TS, 12
 - ul_parms, 12
 - wants_to_join, 14
 - was_selfring, 12
 - was_was_selfring, 12
- station_tx_dequeue
 - tokenring.c, 33
- station_tx_enqueue
 - tokenring.c, 34
- table_index
 - station_struct, 14
- tid
 - private, 10

- timestamp
 - ctrl_packet, 8
- tok_random
 - lib.c, 22
- tok_tx_handler
 - tokenring.c, 34
- token_holding_timer
 - station_struct, 14
- token_holding_timer_handler
 - tokenring.c, 34
- token_pass_timer
 - station_struct, 14
- token_pass_timer_handler
 - tokenring.c, 34
- tokenring.c, 25
 - __NO_VERSION__, 30
 - check_skb, 31
 - check_station_conditions, 31
 - claim_token_handler, 31
 - clean_tx_queue, 31
 - contention_timer_handler, 32
 - decide_to_solicit_successor, 31
 - delete_floating_timers, 26
 - delete_have_token_timers, 26
 - delete_idle_timers, 26
 - delete_joining_timers, 27
 - delete_monitoring_timers, 27
 - delete_offline_timers, 27
 - delete_solicit_timers, 27
 - delete_state_timer, 27
 - enforce_no_timer_pending, 30
 - go_floating, 28
 - go_have_token, 28
 - go_idle, 28
 - go_joining, 28
 - go_monitoring, 28
 - go_offline, 28
 - go_soliciting, 28
 - idle_timer_handler, 32
 - init_station, 31
 - inring_timer_handler, 32
 - is_floating, 27
 - is_have_token, 32
 - is_idle, 27
 - is_implicitack, 32
 - is_joining, 28
 - is_monitoring, 27
 - is_offline, 28
 - is_selfring, 32
 - is_soliciting, 27
 - jiffies, 30
 - make_selfring, 31
 - next_genseq, 32
 - offline_timer_handler, 33
 - process_packet, 31
 - record_states, 28
 - reset_idle_timers, 26
 - set_ethernet_header, 30
 - set_station_offline, 33
 - solicit_successor_handler, 33
 - solicit_wait_handler, 33
 - station_buffered_dequeue, 33
 - station_buffered_enqueue, 33
 - station_tx_dequeue, 33
 - station_tx_enqueue, 34
 - tok_tx_handler, 34
 - token_holding_timer_handler, 34
 - token_pass_timer_handler, 34
 - TOKENRING_C, 30
 - transmit_buffered_queue, 34
 - transmit_tx_queue, 34
 - tx_claim_token, 29
 - tx_done_handler, 34
 - tx_normal_token, 28
 - tx_set_predecessor, 29
 - tx_set_successor, 29
 - tx_solicit_successor, 29
 - tx_token_deleted, 29
 - updated_num_node, 35
 - was_joining, 28
 - was_selfring, 27
 - was_soliciting, 27
- TOKENRING_C
 - tokenring.c, 30
- transmission_rate
 - wtrp.c, 41
- transmit
 - wtrp.c, 39
- transmit_buffered_queue
 - tokenring.c, 34
- transmit_tx_queue
 - tokenring.c, 34

- TRT
 - params.c, 24
- TS
 - station_struct, 12
- tx_claim_token
 - tokenring.c, 29
- tx_done_handler
 - tokenring.c, 34
- tx_normal_token
 - tokenring.c, 28
- tx_proxy
 - private, 10
 - wtrp_config, 16
- tx_set_predecessor
 - tokenring.c, 29
- tx_set_successor
 - tokenring.c, 29
- tx_solicit_successor
 - tokenring.c, 29
- tx_token_deleted
 - tokenring.c, 29
- txpacket
 - private, 10
- type
 - message_t, 8
 - packet, 9
 - wtrp_config, 16
 - wtrp_config_reply, 17
- ul_parms
 - station_struct, 12
- update_my_ctable
 - ctable.c, 21
- update_other_ctable
 - ctable.c, 20
- updated_num_node
 - tokenring.c, 35
- upper_bw
 - params.c, 24
- upper_claim_token_tx_time
 - params.c, 24
- upper_proc_time
 - params.c, 24
- upper_token_holding_time
 - params.c, 24
- upper_total_management_time
 - params.c, 24
- upper_total_token_holding_time
 - params.c, 24
- user_max_num_node
 - qos_struct, 11
- validate_eth_header
 - lib.c, 23
- wants_to_join
 - station_struct, 14
- was_joining
 - tokenring.c, 28
- was_selfring
 - station_struct, 12
 - tokenring.c, 27
- was_soliciting
 - tokenring.c, 27
- was_was_selfring
 - station_struct, 12
- wtrp.c, 35
 - app_tx, 37
 - begin_timing_record, 37
 - end_timing_record, 37
 - is_destined_for_me, 37
 - main, 37
 - max_num_token_pass_try, 40
 - max_token_holding_time, 40
 - max_token_rotation_time, 40
 - notify_mac_available, 38
 - notify_mac_busy, 38
 - process_message, 38
 - processing_time, 40
 - proxy_setup, 38
 - solicit_successor_prob, 40
 - transmission_rate, 41
 - transmit, 39
 - wtrp_register, 39
 - wtrp_set_ul_parm, 39
 - wtrp_unregister, 40
- wtrp_config, 16
 - rx_proxy, 16
 - tx_proxy, 16
 - type, 16
- wtrp_config_reply, 17
 - mac_addr, 17

- type, [17](#)
- wtrp_is_ul_parm_set_by_user
 - [params.c, 25](#)
- wtrp_register
 - [wtrp.c, 39](#)
- wtrp_set_ul_parm
 - [wtrp.c, 39](#)
- wtrp_unregister
 - [wtrp.c, 40](#)

Appendix G

QNX4 driver for Orinoco wireless Reference Manual

1.0

Generated by Doxygen 1.3-rc1

Thu Apr 10 10:44:38 2003

Contents

1 QNX4 driver for Orinoco wireless Data Structure Index	1
2 QNX4 driver for Orinoco wireless File Index	1
3 QNX4 driver for Orinoco wireless Data Structure Documentation	3
4 QNX4 driver for Orinoco wireless File Documentation	5

1 QNX4 driver for Orinoco wireless Data Structure Index

1.1 QNX4 driver for Orinoco wireless Data Structures

Here are the data structures with brief descriptions:

orinoco_private (Linux orinoco_private structure was used with the addition of a few QNX specific fields for inter-process communication)	3
---	---

2 QNX4 driver for Orinoco wireless File Index

2.1 QNX4 driver for Orinoco wireless File List

Here is a list of all documented files with brief descriptions:

etherdevice.h	??
hermes.c (Linux Hermes driver core modified for QNX4)	5
hermes.h (Header file for Hermes driver core)	7
hermes.qnx.c (Replacements for in-line functions in original Linux code)	12
hermes_rid.h (Defines for Hermes Record IDs)	12
ieee802_11.h (IEEE 802.11 frame constants)	16
if.h	??
if_arp.h	??

if_ether.h	??
if_packet.h	??
linux_compat.c (System functions available on Linux, not available on QNX)	18
linux_compat.h (This header file defines a set of macros which allow for some linux-specific functions,types,etc)	19
net_init.c (Net_init.c: Initialization for network devices, This version is slightly modified to compile on QNX4 by Sue Dickey)	22
netdev.c (Routines from Linux netdevice.h and net/core/dev.c needed by the orinoco and wtrp drivers)	24
netdevice.h	??
ori.c (QNX 4 driver for Orinoco card)	25
ori.h (Ori.h – Header file for ori.c, Orinoco device driver for QNX 4)	28
orinoco.c (Linux orinoco.c source modified for QNX4/WTRP)	30
orinoco.h (Linux file ported to QNX4 orinoco.h)	33
show_hermes.c (Standalone program to print Orinoco registers)	35
skbuff.c (Simplified version of Linux skbuff routines for QNX 4)	35
skbuff.h	??
socket.h	??
timer.c (Replacement version of linux timer.c using QNX timers)	43
timer.h	??
trace_init.c (Standalone program to do card reset and print firmware settings)	43
tracehermes.c (Trace routines for Hermes Records ID (RID) tables, calls to the routines can be added as needed during debugging to hermes.c)	44
wireless.h	??

3 QNX4 driver for Orinoco wireless Data Structure Documentation

3.1 orinoco_private Struct Reference

Linux orinoco_private structure was used with the addition of a few QNX specific fields for inter-process communication.

```
#include <orinoco.h>
```

Data Fields

- void * **card**
- int(* **hard_reset**)(struct orinoco_private *)
- spinlock_t **lock**
- long **state**
- net_device * **ndev**
- net_device_stats **stats**
- iw_statistics **wstats**
- hermes_t **hw**
- u16 **txfid**
- int **firmware_type**
- int **has_ibss**
- int **has_port3**
- int **has_ibss_any**
- int **ibss_port**
- int **has_wep**
- int **has_big_wep**
- int **has_mwo**
- int **has_pm**
- int **has_preamble**
- int **has_sensitivity**
- int **nicbuf_size**
- int **broken_cor_reset**
- u16 **channel_mask**
- u32 **iw_mode**
- int **prefer_port3**
- u16 **wep_on**
- u16 **wep_restrict**
- u16 **tx_key**

- orinoco_keys_t **keys**
- int **bitratemode**
- char **nick** [IW_ESSID_MAX_SIZE+1]
- char **desired_essid** [IW_ESSID_MAX_SIZE+1]
- u16 **frag_thresh**
- u16 **mwo_robust**
- u16 **channel**
- u16 **ap_density**
- u16 **rts_thresh**
- u16 **pm_on**
- u16 **pm_mcast**
- u16 **pm_period**
- u16 **pm_timeout**
- u16 **preamble**
- int **spy_number**
- u_char **spy_address** [IW_MAX_SPY][ETH_ALEN]
- iw_quality **spy_stat** [IW_MAX_SPY]
- int **port_type**
- int **allow_ibss**
- int **promiscuous**
- int **mc_count**
- proc_dir_entry * **dir_dev**
- pid_t **tx_proxy**
- pid_t **rx_proxy**
- sk_buff_head * **rx_queue**
- char **host_string** [IW_ESSID_MAX_SIZE+1]

3.1.1 Detailed Description

Linux orinoco_private structure was used with the addition of a few QNX specific fields for inter-process communication.

3.1.2 Field Documentation

3.1.2.1 char orinoco_private::host_string[IW_ESSID_MAX_SIZE+1]

set a name for nick, since may not have higher-level networking.

3.1.2.2 pid_t orinoco_private::rx_proxy

QNX: receive proxy.

3.1.2.3 struct sk_buff_head* orinoco_private::rx_queue

QNX queue to hold received messages.

3.1.2.4 pid_t orinoco_private::tx_proxy

QNX: transmit proxy.

The documentation for this struct was generated from the following file:

- [orinoco.h](#)

4 QNX4 driver for Orinoco wireless File Documentation

4.1 hermes.c File Reference

Linux Hermes driver core modified for QNX4.

```
#include <linux_compat.h>
#include <errno.h>
#include "hermes.h"
```

Defines

- #define **CMD_BUSY_TIMEOUT** (100)
- #define **CMD_INIT_TIMEOUT** (50000)
- #define **CMD_COMPL_TIMEOUT** (20000)
- #define **ALLOC_COMPL_TIMEOUT** (1000)
- #define **DEBUG**(lvl, fmt, var)
- #define **IO_TYPE**(hw) ((hw) → io_space ? "IO " : "MEM ")

Functions

- void **hermes_struct_init** (hermes_t *hw, ulong address, int io_space, int reg-spacing)
- int **hermes_reset** (hermes_t *hw)
- void **hermes_docmd_trace** (int cmd, int err)
- int **hermes_docmd_wait** (hermes_t *hw, u16 cmd, u16 parm0, hermes_response_t *resp)
- int **hermes_allocate** (hermes_t *hw, u16 size, u16 *fid)
- int **hermes_bap_pread** (hermes_t *hw, int bap, void *buf, int len, u16 id, u16 offset)
- int **hermes_bap_pwrite** (hermes_t *hw, int bap, const void *buf, int len, u16 id, u16 offset)
- int **hermes_read_ltv** (hermes_t *hw, int bap, u16 rid, int bufsize, u16 *length, void *buf)
- int **hermes_write_ltv** (hermes_t *hw, int bap, u16 rid, u16 length, const void *value)

4.1.1 Detailed Description

Linux Hermes driver core modified for QNX4.

Driver core for the "Hermes" wireless MAC controller, as used in the Lucent Orinoco and Cabletron RoamAbout cards. It should also work on the hfa3841 and hfa3842 MAC controller chips used in the Prism II chipsets.

This is not a complete driver, just low-level access routines for the MAC controller itself.

Based on the prism2 driver from Absolute Value Systems' linux-wlan project, the Linux wvlan_cs driver, Lucent's HCF-Light (wvlan_hcf.c) library, and the NetBSD wireless driver (in no particular order).

Copyright (C) 2000, David Gibson, Linuxcare Australia
<hermes@gibson.dropbear.id.au> Copyright (C) 2001, David Gibson,
IBM <hermes@gibson.dropbear.id.au>

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

Alternatively, the contents of this file may be used under the terms of the GNU General Public License version 2 (the "GPL"), in which case the provisions of the GPL are applicable instead of the above. If you wish to allow the use of your version of this file

only under the terms of the GPL and not to allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the GPL. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the GPL.

4.2 hermes.h File Reference

Header file for Hermes driver core.

```
#include <linux_compat.h>
```

Data Structures

- struct **hermes**
- struct **hermes_debug_entry**
- struct **hermes_response**
- struct **hermes_rx_descriptor**
- struct **hermes_scan_apinfo**
- struct **hermes_scan_frame**
- struct **hermes_tallies_frame**
- struct **hermes_tx_descriptor**

Defines

- #define **HERMES_ALLOC_LEN_MIN** (4)
- #define **HERMES_ALLOC_LEN_MAX** (2400)
- #define **HERMES_LTV_LEN_MAX** (34)
- #define **HERMES_BAP_DATALEN_MAX** (4096)
- #define **HERMES_BAP_OFFSET_MAX** (4096)
- #define **HERMES_PORTID_MAX** (7)
- #define **HERMES_NUMPORTS_MAX** (HERMES_PORTID_MAX+1)
- #define **HERMES_PDR_LEN_MAX** (260)
- #define **HERMES_PDA_RECS_MAX** (200)
- #define **HERMES_PDA_LEN_MAX** (1024)
- #define **HERMES_SCANRESULT_MAX** (35)
- #define **HERMES_CHINFORESULT_MAX** (8)
- #define **HERMES_MAX_MULTICAST** (16)
- #define **HERMES_MAGIC** (0x7d1f)
- #define **HERMES_CMD** (0x00)
- #define **HERMES_PARAM0** (0x02)
- #define **HERMES_PARAM1** (0x04)

- #define **HERMES_PARAM2** (0x06)
- #define **HERMES_STATUS** (0x08)
- #define **HERMES_RESP0** (0x0A)
- #define **HERMES_RESP1** (0x0C)
- #define **HERMES_RESP2** (0x0E)
- #define **HERMES_INFOFID** (0x10)
- #define **HERMES_RXFID** (0x20)
- #define **HERMES_ALLOCFID** (0x22)
- #define **HERMES_TXCOMPLFID** (0x24)
- #define **HERMES_SELECT0** (0x18)
- #define **HERMES_OFFSET0** (0x1C)
- #define **HERMES_DATA0** (0x36)
- #define **HERMES_SELECT1** (0x1A)
- #define **HERMES_OFFSET1** (0x1E)
- #define **HERMES_DATA1** (0x38)
- #define **HERMES_EVSTAT** (0x30)
- #define **HERMES_INTEN** (0x32)
- #define **HERMES_EVACK** (0x34)
- #define **HERMES_CONTROL** (0x14)
- #define **HERMES_SWSUPPORT0** (0x28)
- #define **HERMES_SWSUPPORT1** (0x2A)
- #define **HERMES_SWSUPPORT2** (0x2C)
- #define **HERMES_AUXPAGE** (0x3A)
- #define **HERMES_AUXOFFSET** (0x3C)
- #define **HERMES_AUXDATA** (0x3E)
- #define **HERMES_CMD_BUSY** (0x8000)
- #define **HERMES_CMD_AINFO** (0x7f00)
- #define **HERMES_CMD_MACPORT** (0x0700)
- #define **HERMES_CMD_RECL** (0x0100)
- #define **HERMES_CMD_WRITE** (0x0100)
- #define **HERMES_CMD_PROGMODE** (0x0300)
- #define **HERMES_CMD_CMDCODE** (0x003f)
- #define **HERMES_STATUS_RESULT** (0x7f00)
- #define **HERMES_STATUS_CMDCODE** (0x003f)
- #define **HERMES_OFFSET_BUSY** (0x8000)
- #define **HERMES_OFFSET_ERR** (0x4000)
- #define **HERMES_OFFSET_DATAOFF** (0x0ffe)
- #define **HERMES_EV_TICK** (0x8000)
- #define **HERMES_EV_WTERR** (0x4000)
- #define **HERMES_EV_INFDDROP** (0x2000)
- #define **HERMES_EV_INFO** (0x0080)
- #define **HERMES_EV_DTIM** (0x0020)
- #define **HERMES_EV_CMD** (0x0010)

- #define **HERMES_EV_ALLOC** (0x0008)
- #define **HERMES_EV_TXEXC** (0x0004)
- #define **HERMES_EV_TX** (0x0002)
- #define **HERMES_EV_RX** (0x0001)
- #define **HERMES_CMD_INIT** (0x0000)
- #define **HERMES_CMD_ENABLE** (0x0001)
- #define **HERMES_CMD_DISABLE** (0x0002)
- #define **HERMES_CMD_DIAG** (0x0003)
- #define **HERMES_CMD_ALLOC** (0x000A)
- #define **HERMES_CMD_TX** (0x000B)
- #define **HERMES_CMD_CLRPRST** (0x0012)
- #define **HERMES_CMD_NOTIFY** (0x0010)
- #define **HERMES_CMD_INQUIRE** (0x0011)
- #define **HERMES_CMD_ACCESS** (0x0021)
- #define **HERMES_CMD_DOWNLD** (0x0022)
- #define **HERMES_CMD_MONITOR** (0x0038)
- #define **HERMES_MONITOR_ENABLE** (0x000b)
- #define **HERMES_MONITOR_DISABLE** (0x000f)
- #define **HERMES_DESCRIPTOR_OFFSET** 0
- #define **HERMES_802_11_OFFSET** (14)
- #define **HERMES_802_3_OFFSET** (14+32)
- #define **HERMES_802_2_OFFSET** (14+32+14)
- #define **HERMES_RXSTAT_ERR** (0x0003)
- #define **HERMES_RXSTAT_BADCRC** (0x0001)
- #define **HERMES_RXSTAT_UNDECRYPTABLE** (0x0002)
- #define **HERMES_RXSTAT_MACPORT** (0x0700)
- #define **HERMES_RXSTAT_MSGTYPE** (0xE000)
- #define **HERMES_RXSTAT_1042** (0x2000)
- #define **HERMES_RXSTAT_TUNNEL** (0x4000)
- #define **HERMES_RXSTAT_WMP** (0x6000)
- #define **HERMES_TXSTAT_RETRYERR** (0x0001)
- #define **HERMES_TXSTAT_AGEDERR** (0x0002)
- #define **HERMES_TXSTAT_DISCON** (0x0004)
- #define **HERMES_TXSTAT_FORMERR** (0x0008)
- #define **HERMES_TXCTRL_TX_OK** (0x0002)
- #define **HERMES_TXCTRL_TX_EX** (0x0004)
- #define **HERMES_TXCTRL_802_11** (0x0008)
- #define **HERMES_TXCTRL_ALT_RTRY** (0x0020)
- #define **HERMES_INQ_TALLIES** (0xF100)
- #define **HERMES_INQ_SCAN** (0xF101)
- #define **HERMES_INQ_LINKSTATUS** (0xF200)
- #define **HERMES_DEBUG_BUFSIZE** 4096
- #define **HERMES_BAP_BUSY_TIMEOUT** (500)

- #define **HERMES_IO** 1
- #define **HERMES_MEM** 0
- #define **HERMES_16BIT_REGSPACING** 0
- #define **HERMES_32BIT_REGSPACING** 1
- #define **hermes_read_reg**(hw, off) (inpw((hw) → iobase + ((off) << (hw) → reg_spacing)))
- #define **hermes_write_reg**(hw, off, val) (outpw((hw) → iobase + ((off) << (hw) → reg_spacing), (val)))
- #define **hermes_read_regn**(hw, name) (hermes_read_reg((hw), HERMES_ ##name))
- #define **hermes_write_regn**(hw, name, val) (hermes_write_reg((hw), HERMES_ ##name, (val)))
- #define **hermes_present**(hw) (hermes_read_regn((hw), SWSUPPORT0) == HERMES_MAGIC)
- #define **hermes_enable_port**(hw, port) (hermes_docmd_wait((hw), HERMES_CMD_ENABLE | ((port) << 8), 0, NULL))
- #define **hermes_disable_port**(hw, port) (hermes_docmd_wait((hw), HERMES_CMD_DISABLE | (port << 8), 0, NULL))
- #define **hermes_inquire**(hw, rid) (hermes_docmd_wait((hw), HERMES_CMD_INQUIRE, (rid), NULL))
- #define **HERMES_BYTES_TO_RECLEN**(n) (((n) % 2) ? (((n)+1)/2)+1 : ((n)/2)+1)
- #define **HERMES_RECLEN_TO_BYTES**(n) (((n)-1) * 2)
- #define **HERMES_READ_RECORD**(hw, bap, rid, buf) (hermes_read_ltv((hw),(bap),(rid), sizeof(*buf), NULL, (buf)))
- #define **HERMES_WRITE_RECORD**(hw, bap, rid, buf) (hermes_write_ltv((hw),(bap),(rid),HERMES_BYTES_TO_RECLEN(sizeof(*buf)),(buf)))
- #define **DO_TRACE_HERMES**

Typedefs

- typedef hermes **hermes_t**
- typedef hermes_response **hermes_response_t**

Functions

- void **hermes_struct_init** (hermes_t *hw, ulong address, int io_space, int reg_spacing)
- int **hermes_reset** (hermes_t *hw)
- int **hermes_docmd_wait** (hermes_t *hw, u16 cmd, u16 parm0, hermes_response_t *resp)
- int **hermes_allocate** (hermes_t *hw, u16 size, u16 *fid)

- int **hermes_bap_pread** (hermes_t *hw, int bap, void *buf, int len, u16 id, u16 offset)
- int **hermes_bap_pwrite** (hermes_t *hw, int bap, const void *buf, int len, u16 id, u16 offset)
- int **hermes_read_ltv** (hermes_t *hw, int bap, u16 rid, int buflen, u16 *length, void *buf)
- int **hermes_write_ltv** (hermes_t *hw, int bap, u16 rid, u16 length, const void *value)
- void **hermes_enable_interrupt** (hermes_t *hw, u16 events)
- void **hermes_set_irqmask** (hermes_t *hw, u16 events)
- void **hermes_read_words** (hermes_t *hw, int off, void *buf, int count)
- void **hermes_write_words** (hermes_t *hw, int off, const void *buf, int count)
- int **hermes_read_wordrec** (hermes_t *hw, int bap, u16 rid, u16 *word)
- int **hermes_write_wordrec** (hermes_t *hw, int bap, u16 rid, u16 word)
- void **hermes_trace_static_cfg** (hermes_t *hw)
- void **hermes_trace_dynamic_cfg** (hermes_t *hw)
- void **hermes_trace_nic_info** (hermes_t *hw)
- void **hermes_trace_mac_info** (hermes_t *hw)
- void **hermes_trace_modem_info** (hermes_t *hw)

Variables

- hermes_rx_descriptor **packed**

4.2.1 Detailed Description

Header file for Hermes driver core.

Driver core for the "Hermes" wireless MAC controller, as used in the Lucent Orinoco and Cabletron RoamAbout cards. It should also work on the hfa3841 and hfa3842 MAC controller chips used in the Prism I & II chipsets.

This is not a complete driver, just low-level access routines for the MAC controller itself.

Based on the prism2 driver from Absolute Value Systems' linux-wlan project, the Linux wvlan_cs driver, Lucent's HCF-Light (wvlan_hcf.c) library, and the NetBSD wireless driver.

Copyright (C) 2000, David Gibson, Linuxcare Australia
<hermes@gibson.dropbear.id.au>

Portions taken from hfa384x.h, Copyright (C) 1999 AbsoluteValue Systems, Inc. All Rights Reserved.

This file distributed under the GPL, version 2.

4.3 hermes_qnx.c File Reference

Replacements for in-line functions in original Linux code.

```
#include <stdio.h>
#include <hermes.h>
```

Functions

- void **hermes_enable_interrupt** (hermes_t *hw, u16 events)
- void **hermes_set_irqmask** (hermes_t *hw, u16 events)
- void **hermes_read_words** (struct hermes *hw, int off, void *buf, int count)
- void **hermes_write_words** (struct hermes *hw, int off, const void *buf, int count)
- int **hermes_read_wordrec** (hermes_t *hw, int bap, u16 rid, u16 *word)
- int **hermes_write_wordrec** (hermes_t *hw, int bap, u16 rid, u16 word)

4.3.1 Detailed Description

Replacements for in-line functions in original Linux code.

4.3.2 Function Documentation

4.3.2.1 void hermes_read_words (struct hermes * hw, int off, void * buf, int count)

Note that for hermes_read_words and hermes_write_words, the count is in 16-bit words, not bytes.

4.4 hermes_rid.h File Reference

Defines for Hermes Record IDs.

Data Structures

- struct **hermes_idstring**
- struct **hermes_multicast**

Defines

- #define **HERMES_RID_CNFPORTTYPE** 0xFC00
- #define **HERMES_RID_CNFWOWNMACADDR** 0xFC01
- #define **HERMES_RID_CNFDESIRESSID** 0xFC02
- #define **HERMES_RID_CNFWOWNCHANNEL** 0xFC03
- #define **HERMES_RID_CNFWOWNSSID** 0xFC04
- #define **HERMES_RID_CNFWOWNATIMWINDOW** 0xFC05
- #define **HERMES_RID_CNFSYSTEMSCALE** 0xFC06
- #define **HERMES_RID_CNFMAXDATALEN** 0xFC07
- #define **HERMES_RID_CNFWDSADDRESS** 0xFC08
- #define **HERMES_RID_CNFPMENABLED** 0xFC09
- #define **HERMES_RID_CNFPMEPS** 0xFC0A
- #define **HERMES_RID_CNFMULTICASTRECEIVE** 0xFC0B
- #define **HERMES_RID_CNFMAXSLEEPDURATION** 0xFC0C
- #define **HERMES_RID_CNFPMHOLDOVERDURATION** 0xFC0D
- #define **HERMES_RID_CNFWOWNNAME** 0xFC0E
- #define **HERMES_RID_CNFWOWNDTIMPERIOD** 0xFC10
- #define **HERMES_RID_CNFWDSADDRESS1** 0xFC11
- #define **HERMES_RID_CNFWDSADDRESS2** 0xFC12
- #define **HERMES_RID_CNFWDSADDRESS3** 0xFC13
- #define **HERMES_RID_CNFWDSADDRESS4** 0xFC14
- #define **HERMES_RID_CNFWDSADDRESS5** 0xFC15
- #define **HERMES_RID_CNFWDSADDRESS6** 0xFC16
- #define **HERMES_RID_CNFMULTICASTPMBUFFERING** 0xFC17
- #define **HERMES_RID_CNFWEPEENABLED_AGERE** 0xFC20
- #define **HERMES_RID_CNFMANDATORYBSSID_SYMBOL** 0xFC21
- #define **HERMES_RID_CNFWEPEDEFAULTKEYID** 0xFC23
- #define **HERMES_RID_CNFDEFAULTKEY0** 0xFC24
- #define **HERMES_RID_CNFDEFAULTKEY1** 0xFC25
- #define **HERMES_RID_CNFMWOROBUST_AGERE** 0xFC25
- #define **HERMES_RID_CNFDEFAULTKEY2** 0xFC26
- #define **HERMES_RID_CNFDEFAULTKEY3** 0xFC27
- #define **HERMES_RID_CNFWEPEFLAGS_INTERSIL** 0xFC28
- #define **HERMES_RID_CNFWEPEKEYMAPPINGTABLE** 0xFC29
- #define **HERMES_RID_CNFAUTHENTICATION** 0xFC2A
- #define **HERMES_RID_CNFMAXASSOCSTA** 0xFC2B
- #define **HERMES_RID_CNFKEYLENGTH_SYMBOL** 0xFC2B
- #define **HERMES_RID_CNFTXCONTROL** 0xFC2C
- #define **HERMES_RID_CNFROAMINGMODE** 0xFC2D
- #define **HERMES_RID_CNFHOSTAUTHENTICATION** 0xFC2E
- #define **HERMES_RID_CNFRVCRCERROR** 0xFC30
- #define **HERMES_RID_CNFMMLIFE** 0xFC31

- #define **HERMES_RID_CNFALTRETRYCOUNT** 0xFC32
- #define **HERMES_RID_CNFBEACONINT** 0xFC33
- #define **HERMES_RID_CNFAPPCFINFO** 0xFC34
- #define **HERMES_RID_CNFSTAPCFINFO** 0xFC35
- #define **HERMES_RID_CNFPRIORITYQUSAGE** 0xFC37
- #define **HERMES_RID_CNFTIMCTRL** 0xFC40
- #define **HERMES_RID_CNFTHIRTY2TALLY** 0xFC42
- #define **HERMES_RID_CNFENHSECURITY** 0xFC43
- #define **HERMES_RID_CNFGROUPADDRESSES** 0xFC80
- #define **HERMES_RID_CNFCREATEIBSS** 0xFC81
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD** 0xFC82
- #define **HERMES_RID_CNFRSTHRESHOLD** 0xFC83
- #define **HERMES_RID_CNFTXRATECONTROL** 0xFC84
- #define **HERMES_RID_CNFPROMISCUOUSMODE** 0xFC85
- #define **HERMES_RID_CNFBASICRATES_SYMBOL** 0xFC8A
- #define **HERMES_RID_CNFPREAMBLE_SYMBOL** 0xFC8C
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD0** 0xFC90
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD1** 0xFC91
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD2** 0xFC92
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD3** 0xFC93
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD4** 0xFC94
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD5** 0xFC95
- #define **HERMES_RID_CNFFRAGMENTATIONTHRESHOLD6** 0xFC96
- #define **HERMES_RID_CNFRSTHRESHOLD0** 0xFC97
- #define **HERMES_RID_CNFRSTHRESHOLD1** 0xFC98
- #define **HERMES_RID_CNFRSTHRESHOLD2** 0xFC99
- #define **HERMES_RID_CNFRSTHRESHOLD3** 0xFC9A
- #define **HERMES_RID_CNFRSTHRESHOLD4** 0xFC9B
- #define **HERMES_RID_CNFRSTHRESHOLD5** 0xFC9C
- #define **HERMES_RID_CNFRSTHRESHOLD6** 0xFC9D
- #define **HERMES_RID_CNFSHORTPREAMBLE** 0xFCB0
- #define **HERMES_RID_CNFWEPKEYS_AGERE** 0xFCB0
- #define **HERMES_RID_CNFECLUDELONGPREAMBLE** 0xFCB1
- #define **HERMES_RID_CNFTXKEY_AGERE** 0xFCB1
- #define **HERMES_RID_CNFAUTHENTICATIONRSPTO** 0xFCB2
- #define **HERMES_RID_CNFBASICRATES** 0xFCB3
- #define **HERMES_RID_CNFSUPPORTEDRATES** 0xFCB4
- #define **HERMES_RID_CNFTICKTIME** 0xFCE0
- #define **HERMES_RID_CNFSCANREQUEST** 0xFCE1
- #define **HERMES_RID_CNFJOINREQUEST** 0xFCE2
- #define **HERMES_RID_CNFAUTHENTICATESTATION** 0xFCE3
- #define **HERMES_RID_CNFCHANNELINFOREQUEST** 0xFCE4
- #define **HERMES_RID_MAXLOADTIME** 0xFD00

- #define **HERMES_RID_DOWNLOADBUFFER** 0xFD01
- #define **HERMES_RID_PRIID** 0xFD02
- #define **HERMES_RID_PRISUPRANGE** 0xFD03
- #define **HERMES_RID_CFIACRANGES** 0xFD04
- #define **HERMES_RID_NICSERNUM** 0xFD0A
- #define **HERMES_RID_NICID** 0xFD0B
- #define **HERMES_RID_MFISUPRANGE** 0xFD0C
- #define **HERMES_RID_CFISUPRANGE** 0xFD0D
- #define **HERMES_RID_CHANNELLIST** 0xFD10
- #define **HERMES_RID_REGULATORYDOMAINS** 0xFD11
- #define **HERMES_RID_TEMPTYPE** 0xFD12
- #define **HERMES_RID_CIS** 0xFD13
- #define **HERMES_RID_STAID** 0xFD20
- #define **HERMES_RID_STASUPRANGE** 0xFD21
- #define **HERMES_RID_MFIACRANGES** 0xFD22
- #define **HERMES_RID_CFIACRANGES2** 0xFD23
- #define **HERMES_RID_SECONDARYVERSION_SYMBOL** 0xFD24
- #define **HERMES_RID_PORTSTATUS** 0xFD40
- #define **HERMES_RID_CURRENTSSID** 0xFD41
- #define **HERMES_RID_CURRENTBSSID** 0xFD42
- #define **HERMES_RID_COMMSQUALITY** 0xFD43
- #define **HERMES_RID_CURRENTTXRATE** 0xFD44
- #define **HERMES_RID_CURRENTBEACONINTERVAL** 0xFD45
- #define **HERMES_RID_CURRENTSCALETHRESHOLDS** 0xFD46
- #define **HERMES_RID_PROTOCOLRSPTIME** 0xFD47
- #define **HERMES_RID_SHORTRETRYLIMIT** 0xFD48
- #define **HERMES_RID_LONGRETRYLIMIT** 0xFD49
- #define **HERMES_RID_MAXTRANSMITLIFETIME** 0xFD4A
- #define **HERMES_RID_MAXRECEIVELIFETIME** 0xFD4B
- #define **HERMES_RID_CFPOLLABLE** 0xFD4C
- #define **HERMES_RID_AUTHENTICATIONALGORITHMS** 0xFD4D
- #define **HERMES_RID_PRIVACYOPTIONIMPLEMENTED** 0xFD4F
- #define **HERMES_RID_CURRENTTXRATE1** 0xFD80
- #define **HERMES_RID_CURRENTTXRATE2** 0xFD81
- #define **HERMES_RID_CURRENTTXRATE3** 0xFD82
- #define **HERMES_RID_CURRENTTXRATE4** 0xFD83
- #define **HERMES_RID_CURRENTTXRATE5** 0xFD84
- #define **HERMES_RID_CURRENTTXRATE6** 0xFD85
- #define **HERMES_RID_OWNNMACADDR** 0xFD86
- #define **HERMES_RID_SCANRESULTSTABLE** 0xFD88
- #define **HERMES_RID_PHYTYPE** 0xFDC0
- #define **HERMES_RID_CURRENTCHANNEL** 0xFDC1
- #define **HERMES_RID_CURRENTPOWERSTATE** 0xFDC2

- #define **HERMES_RID_CCAMODE** 0xFDC3
- #define **HERMES_RID_SUPPORTEDDATARATES** 0xFDC6
- #define **HERMES_RID_BUILDSEQ** 0xFFFFE
- #define **HERMES_RID_FWID** 0xFFFF

Typedefs

- typedef hermes_multicast **hermes_multicast_t**

Variables

- hermes_idstring **packed**

4.4.1 Detailed Description

Defines for Hermes Record IDs.

4.5 ieee802_11.h File Reference

IEEE 802.11 frame constants.

Data Structures

- struct **ieee802_11_hdr**

Defines

- #define **IEEE802_11_DATA_LEN** 2304
- #define **IEEE802_11_HLEN** 30
- #define **IEEE802_11_FRAME_LEN** (IEEE802_11_DATA_LEN + IEEE802_11_HLEN)
- #define **IEEE802_11_FCTL_VERS** 0x0002
- #define **IEEE802_11_FCTL_FTYPE** 0x000c
- #define **IEEE802_11_FCTL_STYPE** 0x00f0
- #define **IEEE802_11_FCTL_TODS** 0x0100
- #define **IEEE802_11_FCTL_FROMDS** 0x0200
- #define **IEEE802_11_FCTL_MOREFRAGS** 0x0400
- #define **IEEE802_11_FCTL_RETRY** 0x0800
- #define **IEEE802_11_FCTL_PM** 0x1000

- #define **IEEE802_11_FCTL_MOREDATA** 0x2000
- #define **IEEE802_11_FCTL_WEP** 0x4000
- #define **IEEE802_11_FCTL_ORDER** 0x8000
- #define **IEEE802_11_FTYPE_MGMT** 0x0000
- #define **IEEE802_11_FTYPE_CTL** 0x0004
- #define **IEEE802_11_FTYPE_DATA** 0x0008
- #define **IEEE802_11_STYPE_ASSOC_REQ** 0x0000
- #define **IEEE802_11_STYPE_ASSOC_RESP** 0x0010
- #define **IEEE802_11_STYPE_REASSOC_REQ** 0x0020
- #define **IEEE802_11_STYPE_REASSOC_RESP** 0x0030
- #define **IEEE802_11_STYPE_PROBE_REQ** 0x0040
- #define **IEEE802_11_STYPE_PROBE_RESP** 0x0050
- #define **IEEE802_11_STYPE_BEACON** 0x0080
- #define **IEEE802_11_STYPE_ATIM** 0x0090
- #define **IEEE802_11_STYPE_DISASSOC** 0x00A0
- #define **IEEE802_11_STYPE_AUTH** 0x00B0
- #define **IEEE802_11_STYPE_DEAUTH** 0x00C0
- #define **IEEE802_11_STYPE_PSPOLL** 0x00A0
- #define **IEEE802_11_STYPE_RTS** 0x00B0
- #define **IEEE802_11_STYPE_CTS** 0x00C0
- #define **IEEE802_11_STYPE_ACK** 0x00D0
- #define **IEEE802_11_STYPE_CFEND** 0x00E0
- #define **IEEE802_11_STYPE_CFENDACK** 0x00F0
- #define **IEEE802_11_STYPE_DATA** 0x0000
- #define **IEEE802_11_STYPE_DATA_CFACK** 0x0010
- #define **IEEE802_11_STYPE_DATA_CFPOLL** 0x0020
- #define **IEEE802_11_STYPE_DATA_CFACKPOLL** 0x0030
- #define **IEEE802_11_STYPE_NULLFUNC** 0x0040
- #define **IEEE802_11_STYPE_CFACK** 0x0050
- #define **IEEE802_11_STYPE_CFPOLL** 0x0060
- #define **IEEE802_11_STYPE_CFACKPOLL** 0x0070
- #define **IEEE802_11_SCTL_FRAG** 0x000F
- #define **IEEE802_11_SCTL_SEQ** 0xFFFF0

Variables

- **ieee802_11_hdr** packed

4.5.1 Detailed Description

IEEE 802.11 frame constants.

4.6 linux_compat.c File Reference

System functions available on Linux, not available on QNX.

```
#include <conio.h>
#include <i86.h>
#include <sys/time.h>
#include <sys/inline.h>
#include <sys/types.h>
#include <errno.h>
#include <linux_compat.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <linux/skbuff.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
#include <linux/etherdevice.h>
#include <linux/wireless.h>
```

Functions

- unsigned long [simulate_jiffies](#) ()
- void [udelay](#) (int interval)
- int [set_bit](#) (int nr, volatile void *ptr)
- int [clear_bit](#) (int nr, volatile void *ptr)
- int [test_bit](#) (int nr, volatile void *ptr)
- void [do_gettimeofday](#) (struct timeval *tv)

4.6.1 Detailed Description

System functions available on Linux, not available on QNX.

4.6.2 Function Documentation

4.6.2.1 void do_gettimeofday (struct timeval * tv)

do_gettimeofday –use clock_gettime to implement

4.6.2.2 int set_bit (int nr, volatile void * ptr)

routines from Linux asm-generic/bitops.h; long is assumed to be 32-bits; later should change these to use assembly code or whatever for atomicity instead of disabling interrupts

4.6.2.3 unsigned long simulate_jiffies ()

Returns Unix time in milliseconds; Linux jiffies is clock ticks since system boot.

Later may want to do something different with QNX time functions for this.

4.6.2.4 void udelay (int interval)

microsecond delay

4.7 linux_compat.h File Reference

This header file defines a set of macros which allow for some linux-specific functions, types, etc.

```
#include <conio.h>
```

```
#include <i86.h>
```

```
#include <sys/time.h>
```

Defines

- #define **ETH_ALEN** 6
- #define **inline**
- #define **inw**(addr) inpw(addr)
- #define **outw_p**(val, addr) outpw(addr, val)
- #define **cpu_to_le16**(x) (x)
- #define **le16_to_cpu**(x) (x)
- #define **le16_to_cpus**(x) do { } while (0)
- #define **ntohs**(x) (((x) & 0xff) << 8) | (((x) & 0xff00) >> 8)

- #define **htons(x)** (((x) & 0xff) << 8) | (((x) & 0xff00) >> 8))
- #define **__attribute__**_(x)
- #define **__cacheline_aligned**
- #define **__init**
- #define **__setup**(a, b)
- #define **NR_CPUS** 1
- #define **smp_processor_id**() 0
- #define **printk** printf
- #define **kmalloc**(size, code) ((void *) malloc(size))
- #define **kfree**(ptr) free(ptr)
- #define **KERN_ERR**
- #define **KERN_DEBUG**
- #define **KERN_WARNING**
- #define **KERN_NOTICE**
- #define **KERN_INFO**
- #define **KERN_CRIT**
- #define **out_of_line_bug**() printf("Out of line bug\n")
- #define **copy_to_user**(a, b, c) (!memcpy(a, b, c))
- #define **copy_from_user**(a, b, c) (!memcpy(a, b, c))
- #define **max_t**(typ, a, b) max((a),(b))
- #define **BUG**() printf("Unexpected error\n")
- #define **access_ok**(a, b, c) 1
- #define **verify_area**(a, b, c) 0
- #define **capable**(x) 1
- #define **create_proc_entry**(a, b, c) 0
- #define **create_proc_read_entry**(a, b, c, d, e) 0
- #define **remove_proc_entry**(a, b)
- #define **EXPORT_SYMBOL**(x)
- #define **HZ** 1000
- #define **spin_lock_bh**(x) _disable()
- #define **spin_unlock_bh**(x) _enable()
- #define **spin_lock_init**(x) do { } while (0)
- #define **spin_lock_irqsave**(x, flags) do { } while (0)
- #define **spin_unlock_irqrestore**(x, flags) do { } while (0)
- #define **cli**() _disable()
- #define **sti**() _enable()
- #define **__cli**() _disable()
- #define **__sti**() _enable()
- #define **br_write_lock_bh**(x)
- #define **br_write_unlock_bh**(x)
- #define **rtnl_lock**(x) _disable()
- #define **rtnl_unlock**(x) _enable()
- #define **atomic_inc**(ptr) ((*ptr)++)

- #define **atomic_read**(ptr) (*(ptr))
- #define **atomic_dec_and_test**(ptr) (((*(ptr))-- == 0)
- #define **GFP_ATOMIC** 0
- #define **KM_SKB_DATA_SOFTIRQ**
- #define **local_irq_save**(x) do { } while (0)
- #define **local_irq_restore**(x) do { } while (0)
- #define **test_and_set_bit**(nr, addr) (set_bit((nr),(addr))?1:0)
- #define **test_and_clear_bit**(nr, addr) (clear_bit((nr),(addr))?1:0)
- #define **jiffies** simulate_jiffies()

Typedefs

- typedef unsigned char **u8**
- typedef unsigned short **u16**
- typedef unsigned long **u32**
- typedef long **s32**
- typedef unsigned char **__u8**
- typedef unsigned short **__u16**
- typedef unsigned long **__u32**
- typedef long **__s32**
- typedef unsigned long **ulong**
- typedef volatile int **atomic_t**
- typedef int **rwlock_t**
- typedef size_t **__kernel_size_t**
- typedef int **spinlock_t**

Functions

- unsigned long [simulate_jiffies](#) ()
- void [udelay](#) (int interval)
- int [set_bit](#) (int nr, void *addr)
- int [clear_bit](#) (int nr, void *addr)
- int [test_bit](#) (int nr, void *addr)

4.7.1 Detailed Description

This header file defines a set of macros which allow for some linux-specific functions, types, etc.

to be used on a QNX 4.25 system 9/3/02 -ASH changed #defines for inw and outw_p to macro #defines 10/25/02 -SRD

4.7.2 Function Documentation

4.7.2.1 unsigned long simulate_jiffies ()

Returns Unix time in milliseconds; Linux jiffies is clock ticks since system boot.

Later may want to do something different with QNX time functions for this.

4.7.2.2 void udelay (int interval)

microsecond delay

4.8 net_init.c File Reference

net_init.c: Initialization for network devices, This version is slightly modified to compile on QNX4 by Sue Dickey.

```
#include <sys/inline.h>
#include <sys/types.h>
#include <errno.h>
#include <linux_compat.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <linux/skbuff.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
#include <linux/etherdevice.h>
#include <linux/wireless.h>
```

Functions

- net_device * [alloc_etherdev](#) (int sizeof_priv)
- **EXPORT_SYMBOL** (init_etherdev)
- **EXPORT_SYMBOL** (alloc_etherdev)
- void **ether_setup** (struct net_device *dev)
- **EXPORT_SYMBOL** (ether_setup)

- **EXPORT_SYMBOL** (register_netdev)
- **EXPORT_SYMBOL** (unregister_netdev)

4.8.1 Detailed Description

net_init.c: Initialization for network devices, This version is slightly modified to compile on QNX4 by Sue Dickey.

Written 1993,1994,1995 by Donald Becker.

The author may be reached as becker@scyld.com, or C/O Scyld Computing Corporation 410 Severn Ave., Suite 210 Annapolis MD 21403

This file contains the initialization for the "pl14+" style ethernet drivers. It should eventually replace most of drivers/net/Space.c. It's primary advantage is that it's able to allocate low-memory buffers. A secondary advantage is that the dangerous NE*000 netcards can reserve their I/O port region before the SCSI probes start.

Modifications/additions by Bjorn Ekwall <bj0rn@blox.se>: ethdev_index[MAX_ETH_CARDS] [register_netdev\(\)](#) / [unregister_netdev\(\)](#)

Modifications by Wolfgang Walter Use dev_close cleanly so we always shut things down tidily.

Changed 29/10/95, Alan Cox to pass sockaddr's around for mac addresses.

14/06/96 - Paul Gortmaker: Add generic [eth_change_mtu\(\)](#) function. 24/09/96 - Paul Norton: Add token-ring variants of the netdev functions.

08/11/99 - Alan Cox: Got fed up of the mess in this file and cleaned it up. We now share common code and have regularised name allocation setups. Abolished the 16 card limits. 03/19/2000 - jgarzik and Urban Widmark: init_etherdev 32-byte align 03/21/2001 - jgarzik: alloc_etherdev and friends

4.8.2 Function Documentation

4.8.2.1 struct net_device* alloc_etherdev (int sizeof_priv)

alloc_etherdev - Allocates and sets up an ethernet device : Size of additional driver-private structure to be allocated for this ethernet device

Fill in the fields of the device structure with ethernet-generic values. Basically does everything except registering the device.

Constructs a new net device, complete with a private data area of size . A 32-byte (not bit) alignment is enforced for this private data area.

4.9 netdev.c File Reference

Routines from Linux netdevice.h and net/core/dev.c needed by the orinoco and wtrp drivers.

```
#include <stdio.h>
#include <sys/types.h>
#include <errno.h>
#include <malloc.h>
#include <string.h>
#include <linux_compat.h>
#include <linux/netdevice.h>
#include <linux/skbuff.h>
```

Functions

- int **netif_device_present** (struct net_device *dev)
- void **netif_device_attach** (struct net_device *dev)
- void **netif_device_detach** (struct net_device *dev)
- void **netif_start_queue** (struct net_device *dev)
- int **netif_wake_queue** (struct net_device *dev)
- void **netif_stop_queue** (struct net_device *dev)
- int **netif_queue_stopped** (struct net_device *dev)
- void **netif_qnx_open** (struct net_device *dev)
- void **netif_qnx_close** (struct net_device *dev)
- int **netif_running** (struct net_device *dev)
- int **dev_alloc_name** (struct net_device *dev, const char *name)
- net_device * **dev_alloc** (const char *name, int *err)
- int **netif_rx** (struct sk_buff *skb)

Variables

- softnet_data softnet_data[NR_CPUS] **__cacheline_aligned**
- int **netdev_max_backlog** = 300
- int **weight_p** = 64
- int **no_cong_thresh** = 10
- int **no_cong** = 20
- int **lo_cong** = 100
- int **mod_cong** = 290
- netif_rx_stats **netdev_rx_stat** [NR_CPUS]

4.9.1 Detailed Description

Routines from Linux netdevice.h and net/core/dev.c needed by the orinoco and wtrp drivers.

Rewritten for QNX 4 by Sue Dickey, Jan 2003.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

4.9.2 Function Documentation

4.9.2.1 struct net_device* dev_alloc (const char * name, int * err)

dev_alloc – allocate a net device structure with a given name

4.9.2.2 int dev_alloc_name (struct net_device * dev, const char * name)

dev_alloc_name - allocate a name for a device : device : name format string

Calls qnx_prefix_attach to name our device.

4.9.2.3 int netif_rx (struct sk_buff * skb)

netif_rx - post buffer to the network code : buffer to post

This function receives a packet from a device driver and queues it for the upper (protocol) levels to process. It always succeeds. The buffer may be dropped during processing for congestion control or by the protocol layers.

return values: NET_RX_SUCCESS (no congestion) NET_RX_CN_LOW (low congestion) NET_RX_CN_MOD (moderate congestion) NET_RX_CN_HIGH (high congestion) NET_RX_DROP (packet was dropped)

4.10 ori.c File Reference

QNX 4 driver for Orinoco card.

```
#include "ori.h"
```

Functions

- pid_t **handler** ()
- int **wtrp_setup** (struct net_device *dev, char *dev_name)
- net_device * **install_orinoco_dev** (int oribase, int irq, char *dev_name, char *host_name, struct sk_buff_head *rx_queue)
- void **config_option_reset** (int configbase)
- int **attach_orinoco_dev** (struct net_device *netdev, int configbase)
- void **process_wtrp_tx** (message_t *msg, struct net_device *netdev)
- void **process_wtrp_rx** (message_t *msg, struct net_device *netdev)
- void **process_wtrp_config** (message_t *msg, struct net_device *netdev, int configbase)
- void **process_ori_open** (message_t *msg, struct net_device *netdev, pid_t pid)
- void **process_ori_close** (message_t *msg, struct net_device *netdev)
- int **main** (int argc, char **argv)

Variables

- pid_t **intr_proxy**
- int **devno**
- int **interrupt_id**
- jmp_buf **env**

4.10.1 Detailed Description

QNX 4 driver for Orinoco card.

Based on Linux GPL orinoco_cs driver.

This driver is not designed to work with the QNX4 networking stack. Instead, it uses support routines ported from the Linux networking code for basic transmit and receive operation; these can be found in the linux_compat directory. It interacts only with the Wireless Token Ring Protocol code, found in the wtrp directory.

Mode of operation: when the ori process is started, a QNX device is created and the Orinoco card is initialized, using the I/O base, IRQ and device name values on the command line. An IO_OPEN message received from the wtrp process causes the device structure to be set-up with the values needed for Independent Basic Service Set (IBSS) operation. Then a WTRP_CONFIG message causes a chip reset, so that the new values take effect, and the values of transmit and receive proxies are returned to the WTRP process in the reply to the WTRP_CONFIG message.

When the wtrp process wants to transmit a message, it sends a WTRP_TX message to the ori process; the reply to this message will be either OK or FAIL (wtrp will need to queue packets for possible retransmission). When the chip has allocated a transmission, the ori process will send the transmit proxy to the wtrp process, to signal that it is OK to send another WTRP_TX. A transmit proxy will also be sent in case of transmission timeout and reset.

When the ori process gets a message reception interrupt from the Orinoco card, it sends the receive proxy to the wtrp process and queues the message. The wtrp process responds with a WTRP_RX message, and the queued data is returned to the wtrp process in the reply.

Written by Adam Howell and Susan Dickey, Feb 2003

4.10.2 Function Documentation

4.10.2.1 int attach_orinoco_dev (struct net_device * netdev, int configbase)

attach_orinoco_dev - in response to WTRP_CONFIG messages, does device-level open

4.10.2.2 void config_option_reset (int configbase)

Configuration Option Register reset is done using CardServices in Linux orinoco driver, but testing showed that if the code that does it is commented out it doesn't seem to make any difference with the Orinoco Silver cards, so we are not currently using this routine.

When used, the configbase is set on the command line from the Configbase value shown in the pin cis output..

4.10.2.3 struct net_device* install_orinoco_dev (int oribase, int irq, char * dev_name, char * host_name, struct sk_buff_head * rx_queue)

install_orinoco_dev - initializes the Orinoco card at I/O port oribase, IRQ irq, to be device with path dev_name.

Attaches interrupt handler to proxy, and sets up orinoco_dev structure.

Have received messages queue in priority order.

DON'T Adjust priority to match that of received message

4.10.2.4 void process_ori_close (message_t * msg, struct net_device * netdev)

Process close request.

4.10.2.5 void process_ori_open (message_t * msg, struct net_device * netdev, pid_t pid)

Process open request (only one allowed); don't start queueing received messages until configuration message with proxy arrives.

4.10.2.6 void process_wtrp_config (message_t * msg, struct net_device * netdev, int configbase)

Process configuration message from WTRP process; later worry about any problems with received this more than once.

4.10.2.7 void process_wtrp_rx (message_t * msg, struct net_device * netdev)

Process request to receive message.

4.10.2.8 void process_wtrp_tx (message_t * msg, struct net_device * netdev)

Process request to transmit message.

4.10.2.9 int wtrp_setup (struct net_device * dev, char * dev_name)

Performs the initial device setup for a specific device to use WTRP MAC and tokenring datalink layer (may do something special later?).

4.11 ori.h File Reference

ori.h – Header file for ori.c, Orinoco device driver for QNX 4

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <unistd.h>
#include <setjmp.h>
#include <signal.h>
```

```
#include <errno.h>
#include <time.h>
#include <sys/timeb.h>
#include <sys/irqinfo.h>
#include <sys/proxy.h>
#include <sys/kernel.h>
#include <sys/name.h>
#include <sys/sched.h>
#include <sys/psinfo.h>
#include <sys/prfx.h>
#include <sys/fd.h>
#include <sys/io_msg.h>
#include <sys/sys_msg.h>
#include <sys/stat.h>
#include <sys/sendmx.h>
#include <sys/inline.h>
#include <sys/types.h>
#include <linux_compat.h>
#include <linux/timer.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <linux/skbuff.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
#include <linux/etherdevice.h>
#include <linux/wireless.h>
#include "hermes.h"
#include "orinoco.h"
#include "packet.h"
#include "params.h"
#include "wtrp.h"
```

Defines

- #define **MAX_DEV_NAME_LENGTH** 80

4.11.1 Detailed Description

ori.h – Header file for ori.c, Orinoco device driver for QNX 4

4.12 orinoco.c File Reference

Linux orinoco.c source modified for QNX4/WTRP.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/kernel.h>
#include <errno.h>
#include <linux/compat.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <linux/skbuff.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
#include <linux/etherdevice.h>
#include <linux/wireless.h>
#include "hermes.h"
#include "hermes_rid.h"
#include "orinoco.h"
#include "ieee802_11.h"
#include "data_tx.h"
#include "params.h"
#include "lib.h"
#include "packet.h"
#include "tokenring.h"
```



```
#include "wtrp.h"
```

Data Structures

- struct **header_struct**
- struct **sta_id**

Defines

- #define **SIOCIWFIRSTPRIV** SIOCDEVPRIVATE
- #define **SPY_NUMBER(priv)** (priv → spy_number)
- #define **ORINOCO_MIN_MTU** 256
- #define **ORINOCO_MAX_MTU** (IEEE802_11_DATA_LEN - ENCAPS_OVERHEAD)
- #define **SYMBOL_MAX_VER_LEN** (14)
- #define **USER_BAP** 0
- #define **IRQ_BAP** 1
- #define **MAX_IRQLOOPS_PER_IRQ** 10
- #define **MAX_IRQLOOPS_PER_JIFFY** (20000/HZ)
- #define **SMALL_KEY_SIZE** 5
- #define **LARGE_KEY_SIZE** 13
- #define **TX_NICBUF_SIZE_BUG** 1585
- #define **DUMMY_FID** 0xFFFF
- #define **MAX_MULTICAST(priv)** (HERMES_MAX_MULTICAST)
- #define **NUM_CHANNELS** (sizeof(channel_frequency) / sizeof(channel_frequency[0]))
- #define **BITRATE_TABLE_SIZE** (sizeof(bitrate_table) / sizeof(bitrate_table[0]))
- #define **ENCAPS_OVERHEAD** (sizeof(encaps_hdr) + 2)
- #define **PROC_LTV_SIZE** 128
- #define **PROC_BUFFER_SIZE** 4096
- #define **PROC_SAFE_SIZE** 3072
- #define **DISPLAY_WORDS** 0
- #define **DISPLAY_BYTES** 1
- #define **DISPLAY_STRING** 2
- #define **DISPLAY_XSTRING** 3
- #define **PROC_REC(name, type)** { HERMES_RID_##name, #name, DISPLAY_##type }
- #define **NUM_RIDS** (sizeof(record_table) / sizeof(record_table[0]))

Functions

- void **orinoco_shutdown** (struct [orinoco_private](#) *priv)
- int **orinoco_reset** (struct [orinoco_private](#) *priv)
- void **orinoco_interrupt** (int irq, void *dev_id, struct pt_regs *regs)
- int **orinoco_proc_dev_init** (struct [orinoco_private](#) *priv)
- void **orinoco_proc_dev_cleanup** (struct [orinoco_private](#) *priv)
- net_device * **alloc_orinocodev** (int sizeof_card)
- int **orinoco_standalone_ioctl** (struct net_device *netdev)
- void **orinoco_standalone_multicast** (struct net_device *netdev)

Variables

- const long **channel_frequency** []
- struct {
 - int **bitrate**
 - int **automatic**
 - u16 **agere_txratectrl**
 - u16 **intersil_txratectrl**
 } **bitrate_table** []
- header_struct **packed**
- u8 **encaps_hdr** [] = {0xaa, 0xaa, 0x03, 0x00, 0x00, 0x00}
- struct {
 - u16 **rid**
 - char * **name**
 - int **displaytype**
 } **record_table** []

4.12.1 Detailed Description

Linux orinoco.c source modified for QNX4/WTRP.

orinoco.c 0.11b - (formerly known as dldwd.cs.c and orinoco_cs.c)

A driver for Hermes or Prism 2 chipset based PCMCIA wireless adaptors, with Lucent/Agere, Intersil or Symbol firmware.

Copyright (C) 2000 David Gibson, Linuxcare Australia
 <hermes@gibson.dropbear.id.au> With some help from : Copyright
 (C) 2001 Jean Tourrilhes, HP Labs <jt@hpl.hp.com> Copyright (C) 2001
 Benjamin Herrenschmidt <benh@kernel.crashing.org>

Based on dummy_cs.c 1.27 2000/06/12 21:27:25

Portions based on wvlan_cs.c 1.0.6, Copyright Andreas
 Neuhaus <andy@fasta.fh-dortmund.de>
<http://www.fasta.fh-dortmund.de/users/andy/wvlan/>

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The initial developer of the original code is David A. Hinds <dahinds@users.sourceforge.net>. Portions created by David A. Hinds are Copyright (C) 1999 David A. Hinds. All Rights Reserved.

Alternatively, the contents of this file may be used under the terms of the GNU General Public License version 2 (the "GPL"), in which case the provisions of the GPL are applicable instead of the above. If you wish to allow the use of your version of this file only under the terms of the GPL and not to allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the GPL. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the GPL.

4.12.2 Variable Documentation

4.12.2.1 `const long channel_frequency[]`

Initial value:

```
{
    2412, 2417, 2422, 2427, 2432, 2437, 2442,
    2447, 2452, 2457, 2462, 2467, 2472, 2484
}
```

4.13 orinoco.h File Reference

Linux file ported to QNX4 orinoco.h.

Data Structures

- struct `orinoco_key`
- struct `orinoco_private`

Linux orinoco_private structure was used with the addition of a few QNX specific fields for inter-process communication.

Defines

- #define **WIRELESS_SPY**
- #define **ORINOCO_MAX_KEY_SIZE** 14
- #define **ORINOCO_MAX_KEYS** 4
- #define **ORINOCO_STATE_INIRQ** 0
- #define **ORINOCO_STATE_DOIRQ** 1
- #define **FIRMWARE_TYPE_AGERE** 1
- #define **FIRMWARE_TYPE_INTERSIL** 2
- #define **FIRMWARE_TYPE_SYMBOL** 3
- #define **DEBUG**(lvl, fmt, var) do { } while (0)
- #define **TRACE_ENTER**(devname) DEBUG(2, "enter %s\n", devname);
- #define **TRACE_EXIT**(devname) DEBUG(2, "exit %s\n", devname);
- #define **RUP_EVEN**(a) ((a) % 2 ? (a) + 1 : (a))

Typedefs

- typedef orinoco_key **orinoco_key_t**
- typedef orinoco_key_t **orinoco_keys_t** [ORINOCO_MAX_KEYS]

Functions

- net_device * **alloc_orinocodev** (int sizeof_card)
- void **orinoco_shutdown** (struct [orinoco_private](#) *dev)
- int **orinoco_reset** (struct [orinoco_private](#) *dev)
- int **orinoco_proc_dev_init** (struct [orinoco_private](#) *dev)
- void **orinoco_proc_dev_cleanup** (struct [orinoco_private](#) *priv)
- void **orinoco_interrupt** (int irq, void *dev_id, struct pt_regs *regs)
- int **orinoco_standalone_ioctl** (struct net_device *netdev)

Variables

- list_head **orinoco_instances**

4.13.1 Detailed Description

Linux file ported to QNX4 orinoco.h.

Common definitions to all pieces of the various orinoco drivers

4.14 show_hermes.c File Reference

Standalone program to print Orinoco registers.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "hermes.h"
#include "hermes_rid.h"
```

Defines

- #define **USER_BAP** 0
- #define **DREG**(name) fprintf(stdout,"%-16s: %04x\n", #name, inpw(base + (HERMES_##name)))

Functions

- int **main** (int argc, char *argv[], char *envp[])

4.14.1 Detailed Description

Standalone program to print Orinoco registers.

4.15 skbuff.c File Reference

Simplified version of Linux skbuff routines for QNX 4.

```
#include <linux_compat.h>
#include <sys/types.h>
#include <linux/netdevice.h>
#include <malloc.h>
#include <string.h>
#include <linux/skbuff.h>
#include <stdio.h>
```

Functions

- void **show_net_buffers** (void)
- sk_buff * **alloc_skb** (unsigned int size, int priority)
- void **__kfree_skbmem** (struct sk_buff *skb)
- void **kfree_skbmem** (struct sk_buff *skb)
- int **skb_queue_empty** (struct sk_buff_head *list)
- sk_buff * **skb_get** (struct sk_buff *skb)
- void **__kfree_skb** (struct sk_buff *skb)
- void **kfree_skb** (struct sk_buff *skb)
- int **skb_cloned** (struct sk_buff *skb)
- int **skb_shared** (struct sk_buff *skb)
- sk_buff * **skb_peek** (struct sk_buff_head *list_)
- sk_buff * **skb_peek_tail** (struct sk_buff_head *list_)
- __u32 **skb_queue_len** (struct sk_buff_head *list_)
- void **skb_queue_head_init** (struct sk_buff_head *list)
- void **__skb_queue_head** (struct sk_buff_head *list, struct sk_buff *newsk)
- void **skb_queue_head** (struct sk_buff_head *list, struct sk_buff *newsk)
- void **__skb_queue_tail** (struct sk_buff_head *list, struct sk_buff *newsk)
- void **skb_queue_tail** (struct sk_buff_head *list, struct sk_buff *newsk)
- sk_buff * **__skb_dequeue** (struct sk_buff_head *list)
- sk_buff * **skb_dequeue** (struct sk_buff_head *list)
- void **__skb_insert** (struct sk_buff *newsk, struct sk_buff *prev, struct sk_buff *next, struct sk_buff_head *list)
- void **skb_insert** (struct sk_buff *old, struct sk_buff *newsk)
- void **__skb_append** (struct sk_buff *old, struct sk_buff *newsk)
- void **skb_append** (struct sk_buff *old, struct sk_buff *newsk)
- void **__skb_unlink** (struct sk_buff *skb, struct sk_buff_head *list)
- void **skb_unlink** (struct sk_buff *skb)
- sk_buff * **__skb_dequeue_tail** (struct sk_buff_head *list)
- sk_buff * **skb_dequeue_tail** (struct sk_buff_head *list)
- int **skb_is_nonlinear** (const struct sk_buff *skb)
- int **skb_headlen** (const struct sk_buff *skb)
- unsigned char * **__skb_put** (struct sk_buff *skb, unsigned int len)
- unsigned char * **skb_put** (struct sk_buff *skb, unsigned int len)
- unsigned char * **__skb_push** (struct sk_buff *skb, unsigned int len)
- unsigned char * **skb_push** (struct sk_buff *skb, unsigned int len)
- char * **__skb_pull** (struct sk_buff *skb, unsigned int len)
- unsigned char * **skb_pull** (struct sk_buff *skb, unsigned int len)
- int **skb_headroom** (const struct sk_buff *skb)
- int **skb_tailroom** (const struct sk_buff *skb)
- void **skb_reserve** (struct sk_buff *skb, unsigned int len)
- void **__skb_trim** (struct sk_buff *skb, unsigned int len)

- void `skb_trim` (struct `sk_buff *skb`, unsigned int `len`)
- void `skb_orphan` (struct `sk_buff *skb`)
- void `skb_queue_purge` (struct `sk_buff_head *list`)
- void `__skb_queue_purge` (struct `sk_buff_head *list`)
- `sk_buff * __dev_alloc_skb` (unsigned int `length`, int `gfp_mask`)
- `sk_buff * dev_alloc_skb` (unsigned int `length`)
- `sk_buff * skb_clone` (struct `sk_buff *skb`, int `priority`)
- `sk_buff * skb_copy` (struct `sk_buff *skb`, int `priority`)

Variables

- atomic_t `net_skbcount` = 0
- atomic_t `net_locked` = 0
- atomic_t `net_allocs` = 0
- atomic_t `net_fails` = 0
- atomic_t `net_free_locked` = 0
- atomic_t `ip_frag_mem`

4.15.1 Detailed Description

Simplified version of Linux skbuff routines for QNX 4.

Derived by Sue Dickey, from the routines in the Linux net/core source by Alan Cox and Florian La Roche. Simplified to compile under WATCOM and remove Linux kernel specific memory handling.

4.15.2 Function Documentation

4.15.2.1 struct `sk_buff*` `__dev_alloc_skb` (unsigned int *length*, int *gfp_mask*)

`__dev_alloc_skb` - allocate an skbuff for sending : `length` to allocate : `get_free_pages` mask, passed to `alloc_skb`

Allocate a new `&sk_buff` and assign it a usage count of one. The buffer has unspecified headroom built in. Users should allocate the headroom they think they need without accounting for the built in space. The built in space is used for optimisations.

NULL is returned in there is no free memory.

4.15.2.2 void `__kfree_skbmem` (struct `sk_buff *skb`)

Free an skbuff by memory.

4.15.2.3 struct sk_buff* __skb_dequeue (struct sk_buff_head * list)

`__skb_dequeue` - remove from the head of the queue : list to dequeue from

Remove the head of the list. This function does not take any locks so must be used with appropriate locks held only. The head item is returned or NULL if the list is empty.

4.15.2.4 struct sk_buff* __skb_dequeue_tail (struct sk_buff_head * list)

`__skb_dequeue_tail` - remove from the tail of the queue : list to dequeue from

Remove the tail of the list. This function does not take any locks so must be used with appropriate locks held only. The tail item is returned or NULL if the list is empty.

4.15.2.5 void __skb_queue_head (struct sk_buff_head * list, struct sk_buff * newsk)

`__skb_queue_head` - queue a buffer at the list head : list to use : buffer to queue

Queue a buffer at the start of a list. This function takes no locks and you must therefore hold required locks before calling it.

A buffer cannot be placed on two lists at the same time.

4.15.2.6 void __skb_queue_purge (struct sk_buff_head * list)

`__skb_purge` - empty a list : list to empty

Delete all buffers on an `&sk_buff` list. Each buffer is removed from the list and one reference dropped. This function does not take the list lock and the caller must hold the relevant locks to use it.

4.15.2.7 void __skb_queue_tail (struct sk_buff_head * list, struct sk_buff * newsk)

`__skb_queue_tail` - queue a buffer at the list tail : list to use : buffer to queue

Queue a buffer at the end of a list. This function takes no locks and you must therefore hold required locks before calling it.

A buffer cannot be placed on two lists at the same time.

4.15.2.8 struct sk_buff* alloc_skb (unsigned int size, int priority)

Allocate a new skbuff.

We do this ourselves so we can fill in a few 'private' fields and also do memory statistics to find all the [BEEP] leaks.

QNX 4: removed Linux-dependent interrupt stacking check, fields not in current version of skbuff.h

4.15.2.9 struct sk_buff* dev_alloc_skb (unsigned int *length*)

dev_alloc_skb - allocate an skbuff for sending : length to allocate

Allocate a new &sk_buff and assign it a usage count of one. The buffer has unspecified headroom built in. Users should allocate the headroom they think they need without accounting for the built in space. The built in space is used for optimisations.

NULL is returned in there is no free memory. Although this function allocates memory it can be called from an interrupt.

4.15.2.10 void kfree_skb (struct sk_buff * *skb*)

kfree_skb - free an sk_buff : buffer to free

Drop a reference to the buffer and free it if the usage count has hit zero.

4.15.2.11 void skb_append (struct sk_buff * *old*, struct sk_buff * *newsk*)

skb_append - append a buffer : buffer to insert after : buffer to insert

Place a packet after a given packet in a list. The list locks are taken and this function is atomic with respect to other list locked calls. A buffer cannot be placed on two lists at the same time.

4.15.2.12 int skb_cloned (struct sk_buff * *skb*)

skb_cloned - is the buffer a clone : buffer to check

Returns true if the buffer was generated with [skb_clone\(\)](#) and is one of multiple shared copies of the buffer. Cloned buffers are shared data so must not be written to under normal circumstances.

4.15.2.13 struct sk_buff* skb_copy (struct sk_buff * *skb*, int *priority*)

This is slower, and copies the whole data area.

4.15.2.14 struct sk_buff* skb_dequeue (struct sk_buff_head * *list*)

skb_dequeue - remove from the head of the queue : list to dequeue from

Remove the head of the list. The list lock is taken so the function may be used safely with other locking list functions. The head item is returned or NULL if the list is empty.

4.15.2.15 struct sk_buff* skb_dequeue_tail (struct sk_buff_head * list)

skb_dequeue - remove from the head of the queue : list to dequeue from

Remove the head of the list. The list lock is taken so the function may be used safely with other locking list functions. The tail item is returned or NULL if the list is empty.

4.15.2.16 struct sk_buff* skb_get (struct sk_buff * skb)

skb_get - reference buffer : buffer to reference

Makes another reference to a socket buffer and returns a pointer to the buffer.

4.15.2.17 int skb_headroom (const struct sk_buff * skb)

skb_headroom - bytes at buffer head : buffer to check

Return the number of bytes of free space at the head of an &sk_buff.

4.15.2.18 void skb_insert (struct sk_buff * old, struct sk_buff * newsk)

skb_insert - insert a buffer : buffer to insert before : buffer to insert

Place a packet before a given packet in a list. The list locks are taken and this function is atomic with respect to other list locked calls. A buffer cannot be placed on two lists at the same time.

4.15.2.19 void skb_orphan (struct sk_buff * skb)

skb_orphan - orphan a buffer : buffer to orphan

If a buffer currently has an owner then we call the owner's destructor function and make the unowned. The buffer continues to exist but is no longer charged to its former owner.

4.15.2.20 struct sk_buff* skb_peek (struct sk_buff_head * list_)

skb_peek : list to peek at

Peek an &sk_buff. Unlike most other operations you MUST be careful with this one. A peek leaves the buffer on the list and someone else may run off with it. You must hold the appropriate locks or have a private queue to do this.

Returns NULL for an empty list or a pointer to the head element. The reference count is not incremented and the reference is therefore volatile. Use with caution.

4.15.2.21 struct sk_buff* skb_peek_tail (struct sk_buff_head * list_)

`skb_peek_tail` : list to peek at

Peek an `&sk_buff`. Unlike most other operations you **MUST** be careful with this one. A peek leaves the buffer on the list and someone else may run off with it. You must hold the appropriate locks or have a private queue to do this.

Returns NULL for an empty list or a pointer to the tail element. The reference count is not incremented and the reference is therefore volatile. Use with caution.

4.15.2.22 `unsigned char* skb_pull (struct sk_buff * skb, unsigned int len)`

`skb_pull` - remove data from the start of a buffer : buffer to use : amount of data to remove

This function removes data from the start of a buffer, returning the memory to the headroom. A pointer to the next data in the buffer is returned. Once the data has been pulled future pushes will overwrite the old data.

4.15.2.23 `unsigned char* skb_push (struct sk_buff * skb, unsigned int len)`

`skb_push` - add data to the start of a buffer : buffer to use : amount of data to add

This function extends the used data area of the buffer at the buffer start. If this would exceed the total buffer headroom the kernel will panic. A pointer to the first byte of the extra data is returned.

4.15.2.24 `unsigned char* skb_put (struct sk_buff * skb, unsigned int len)`

`skb_put` - add data to a buffer : buffer to use : amount of data to add

This function extends the used data area of the buffer. If this would exceed the total buffer size the kernel will panic. A pointer to the first byte of the extra data is returned.

4.15.2.25 `int skb_queue_empty (struct sk_buff_head * list)`

`skb_queue_empty` - check if a queue is empty : queue head

Returns true if the queue is empty, false otherwise.

4.15.2.26 `void skb_queue_head (struct sk_buff_head * list, struct sk_buff * newsk)`

`skb_queue_head` - queue a buffer at the list head : list to use : buffer to queue

Queue a buffer at the start of the list. This function takes the list lock and can be used safely with other locking `&sk_buff` functions safely.

A buffer cannot be placed on two lists at the same time.

4.15.2.27 `__u32 skb_queue_len (struct sk_buff_head * list_)`

`skb_queue_len` - get queue length : list to measure

Return the length of an `&sk_buff` queue.

4.15.2.28 `void skb_queue_purge (struct sk_buff_head * list)`

`skb_purge` - empty a list : list to empty

Delete all buffers on an `&sk_buff` list. Each buffer is removed from the list and one reference dropped. This function takes the list lock and is atomic with respect to other list locking functions.

4.15.2.29 `void skb_queue_tail (struct sk_buff_head * list, struct sk_buff * newsk)`

`skb_queue_tail` - queue a buffer at the list tail : list to use : buffer to queue

Queue a buffer at the tail of the list. This function takes the list lock and can be used safely with other locking `&sk_buff` functions safely.

A buffer cannot be placed on two lists at the same time.

4.15.2.30 `void skb_reserve (struct sk_buff * skb, unsigned int len)`

`skb_reserve` - adjust headroom : buffer to alter : bytes to move

Increase the headroom of an empty `&sk_buff` by reducing the tail room. This is only allowed for an empty buffer.

4.15.2.31 `int skb_shared (struct sk_buff * skb)`

`skb_shared` - is the buffer shared : buffer to check

Returns true if more than one person has a reference to this buffer.

4.15.2.32 `int skb_tailroom (const struct sk_buff * skb)`

`skb_tailroom` - bytes at buffer end : buffer to check

Return the number of bytes of free space at the tail of an `sk_buff`

4.15.2.33 `void skb_trim (struct sk_buff * skb, unsigned int len)`

`skb_trim` - remove end from a buffer : buffer to alter : new length

Cut the length of a buffer down by removing data from the tail. If the buffer is already under the length specified it is not modified.

4.15.2.34 void skb_unlink (struct sk_buff * skb)

skb_unlink - remove a buffer from a list : buffer to remove

Place a packet after a given packet in a list. The list locks are taken and this function is atomic with respect to other list locked calls

Works even without knowing the list it is sitting on, which can be handy at times. It also means that THE LIST MUST EXIST when you unlink. Thus a list must have its contents unlinked before it is destroyed.

4.16 timer.c File Reference

Replacement version of linux timer.c using QNX timers.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include <time.h>
#include <linux_compat.h>
#include <linux/timer.h>
```

Functions

- void **init_timer** (timer_list_t *timer)
- void **add_timer** (timer_list_t *timer)
- void **del_timer** (timer_list_t *timer)
- int **mod_timer** (timer_list_t *timer, unsigned long expires)

4.16.1 Detailed Description

Replacement version of linux timer.c using QNX timers.

4.17 trace_init.c File Reference

Standalone program to do card reset and print firmware settings.

```
#include <unistd.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <sys/inline.h>
#include <sys/types.h>
#include <errno.h>
#include <linux_compat.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <linux/skbuff.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
#include <linux/etherdevice.h>
#include <linux/wireless.h>
#include "hermes.h"
#include "hermes_rid.h"
#include "orinoco.h"
```

Defines

- #define **USER_BAP** 0
- #define **DREG**(name) fprintf(stdout,"%-16s: %04x\n", #name, inpw(base + (HERMES_##name)))

Functions

- int **main** (int argc, char *argv[], char *envp[])

4.17.1 Detailed Description

Standalone program to do card reset and print firmware settings.

4.18 tracehermes.c File Reference

Trace routines for Hermes Records ID (RID) tables, calls to the routines can be added as needed during debugging to hermes.c.

```
#include <linux_compat.h>
#include <errno.h>
#include "hermes.h"
#include "hermes_rid.h"
```

Data Structures

- struct **record_info**

Defines

- #define **NUM_STATIC_RIDS** (sizeof(static_cfg_rid_table) / sizeof(record_info_t))
- #define **NUM_DYNAMIC_RIDS** (sizeof(dynamic_cfg_rid_table) / sizeof(record_info_t))
- #define **NUM_NIC_INFO_RIDS** (sizeof(nic_info_rid_table) / sizeof(record_info_t))
- #define **NUM_MAC_INFO_RIDS** (sizeof(mac_info_rid_table) / sizeof(record_info_t))
- #define **NUM_MODEM_INFO_RIDS** (sizeof(modem_info_rid_table) / sizeof(record_info_t))
- #define **HERMES_TRACE_MAX_FIELD_SIZE** 34

Typedefs

- typedef record_info **record_info_t**

Functions

- void **hermes_trace_show_rid_table** (hermes_t *hw, record_info_t *rid_table, int num_rids)
- void **hermes_trace_static_cfg** (hermes_t *hw)
- void **hermes_trace_dynamic_cfg** (hermes_t *hw)
- void **hermes_trace_nic_info** (hermes_t *hw)
- void **hermes_trace_mac_info** (hermes_t *hw)
- void **hermes_trace_modem_info** (hermes_t *hw)
- **EXPORT_SYMBOL** (hermes_trace_static_cfg)

Variables

- record_info_t **static_cfg_rid_table** []
- record_info_t **dynamic_cfg_rid_table** []
- record_info_t **nic_info_rid_table** []
- record_info_t **mac_info_rid_table** []
- record_info_t **modem_info_rid_table** []

4.18.1 Detailed Description

Trace routines for Hermes Records ID (RID) tables, calls to the routines can be added as needed during debugging to hermes.c.

4.18.2 Variable Documentation

4.18.2.1 record_info_t dynamic_cfg_rid_table []

Initial value:

```
{
    {HERMES_RID_CNFGROUPADDRESSES, 0, "CNFGROUPADDRESSES"},
    {HERMES_RID_CNFCREATEIBSS, 0, "CNFCREATEIBSS"},
    {HERMES_RID_CNFFRAGMENTATIONTHRESHOLD, 0, "CNFFRAGMENTATIONTHRESHOLD"},
    {HERMES_RID_CNFRTSTHRESHOLD, 0, "CNFRTSTHRESHOLD"},
    {HERMES_RID_CNFTXRATECONTROL, 0, "CNFTXRATECONTROL"},
    {HERMES_RID_CNFPROMISCUOUSMODE, 0, "CNFPROMISCUOUSMODE"},
    {HERMES_RID_CNFBASICRATES_SYMBOL, 0, "CNFBASICRATES_SYMBOL"},
    {HERMES_RID_CNFPREAMBLE_SYMBOL, 0, "CNFPREAMBLE_SYMBOL"},
    {HERMES_RID_CNFFRAGMENTATIONTHRESHOLD0, 0, "CNFFRAGTHRESHOLD0"},
    {HERMES_RID_CNFRTSTHRESHOLD0, 0, "CNFRTSTHRESHOLD0"},
    {HERMES_RID_CNFSHORTPREAMBLE, 0, "CNFSHORTPREAMBLE"},
    {HERMES_RID_CNFWEPKEYS_AGERE, 0, "CNFWEPKEYS_AGERE"},
    {HERMES_RID_CNFECLUDELONGPREAMBLE, 0, "CNFECLUDELONGPREAMBLE"},
    {HERMES_RID_CNFTXKEY_AGERE, 0, "CNFTXKEY_AGERE"},
    {HERMES_RID_CNFAUTHENTICATIONRSPTO, 0, "CNFAUTHENTICATIONRSPTO"},
    {HERMES_RID_CNFBASICRATES, 0, "CNFBASICRATES"},
    {HERMES_RID_CNFSUPPORTEDRATES, 0, "CNFSUPPORTEDRATES"},
    {HERMES_RID_CNFTICKTIME, 0, "CNFTICKTIME"},
    {HERMES_RID_CNFSKANREQUEST, 0, "CNFSKANREQUEST"},
    {HERMES_RID_CNFJOINREQUEST, 0, "CNFJOINREQUEST"},
    {HERMES_RID_CNFAUTHENTICATESTATION, 0, "CNFAUTHENTICATESTATION"},
    {HERMES_RID_CNFCHANNELINFOREQUEST, 0, "CNFCHANNELINFOREQUEST"},
    {HERMES_RID_MAXLOADTIME, 0, "MAXLOADTIME"},
    {HERMES_RID_DOWNLOADBUFFER, 0, "DOWNLOADBUFFER"}
}
```


4.18.2.2 record_info_t mac_info_rid_table[]**Initial value:**

```
{
    {HERMES_RID_PORTSTATUS, 0, "PORTSTATUS"},
    {HERMES_RID_CURRENTSSID, 0, "CURRENTSSID"},
    {HERMES_RID_CURRENTBSSID, 0, "CURRENTBSSID"},
    {HERMES_RID_COMMSQUALITY, 0, "COMMSQUALITY"},
    {HERMES_RID_CURRENTTXRATE, 0, "CURRENTTXRATE"},
    {HERMES_RID_CURRENTBEACONINTERVAL, 0, "CURRENTBEACONINTERVAL"},
    {HERMES_RID_CURRENTSCALETHRESHOLDS, 0, "CURRENTSCALETHRESHOLDS"},
    {HERMES_RID_PROTOCOLRSPTIME, 0, "PROTOCOLRSPTIME"},
    {HERMES_RID_SHORTRETRYLIMIT, 0, "SHORTRETRYLIMIT"},
    {HERMES_RID_LONGRETRYLIMIT, 0, "LONGRETRYLIMIT"},
    {HERMES_RID_MAXTRANSMITLIFETIME, 0, "MAXTRANSMITLIFETIME"},
    {HERMES_RID_MAXRECEIVELIFETIME, 0, "MAXRECEIVELIFETIME"},
    {HERMES_RID_CFPOLLABLE, 0, "CFPOLLABLE"},
    {HERMES_RID_AUTHENTICATIONALGORITHMS, 0, "AUTHENTICATIONALGORITHMS"},
    {HERMES_RID_PRIVACYOPTIONIMPLEMENTED, 0, "PRIVACYOPTIONIMPLEMENTED"},
    {HERMES_RID_CURRENTTXRATE6, 0, "CURRENTTXRATE6"},
    {HERMES_RID_OWNNMACADDR, 0, "OWNNMACADDR"},
    {HERMES_RID_SCANRESULTSTABLE, 0, "SCANRESULTSTABLE"},
}
```

4.18.2.3 record_info_t modem_info_rid_table[]**Initial value:**

```
{
    {HERMES_RID_PHYTYPE, 0, "PHYTYPE"},
    {HERMES_RID_CURRENTCHANNEL, 0, "CURRENTCHANNEL"},
    {HERMES_RID_CURRENTPOWERSTATE, 0, "CURRENTPOWERSTATE"},
    {HERMES_RID_CCAMODE, 0, "CCAMODE"},
    {HERMES_RID_SUPPORTEDDATARATES, 0, "SUPPORTEDDATARATES"},
    {HERMES_RID_BUILDSEQ, 0, "BUILDSEQ"},
    {HERMES_RID_FWID, 0, "FWID"},
}
```

4.18.2.4 record_info_t nic_info_rid_table[]**Initial value:**

```
{
    {HERMES_RID_PRIID, 0, "PRIID"},
    {HERMES_RID_PRISUPRANGE, 0, "PRISUPRANGE"},
}
```

```
{HERMES_RID_CFIACRANGES, 0, "CFIACRANGES"},
{HERMES_RID_NICSERNUM, 0, "NICSERNUM"},
{HERMES_RID_NICID, 0, "NICID"},
{HERMES_RID_MFISUPRANGE, 0, "MFISUPRANGE"},
{HERMES_RID_CFISUPRANGE, 0, "CFISUPRANGE"},
{HERMES_RID_CHANNELLIST, 0, "CHANNELLIST"},
{HERMES_RID_REGULATORYDOMAINS, 0, "REGULATORYDOMAINS"},
{HERMES_RID_TEMPTYPE, 0, "TEMPTYPE"},
{HERMES_RID_CIS, 0, "CIS"},
{HERMES_RID_STAID, 0, "STAID"},
{HERMES_RID_STASUPRANGE, 0, "STASUPRANGE"},
{HERMES_RID_MFIACRANGES, 0, "MFIACRANGES"},
{HERMES_RID_CFIACRANGES2, 0, "CFIACRANGES2"},
{HERMES_RID_SECONDARYVERSION_SYMBOL, 0, "SECONDARYVERSION_SYMBOL"},
}
```

Index

- `__attribute__`
 - `linux_compat.h`, 20
 - `__cacheline_aligned`
 - `linux_compat.h`, 20
 - `netdev.c`, 24
 - `__cli`
 - `linux_compat.h`, 20
 - `__dev_alloc_skb`
 - `skbuff.c`, 37
 - `__init`
 - `linux_compat.h`, 20
 - `__kernel_size_t`
 - `linux_compat.h`, 21
 - `__kfree_skb`
 - `skbuff.c`, 36
 - `__kfree_skbmem`
 - `skbuff.c`, 37
 - `__s32`
 - `linux_compat.h`, 21
 - `__setup`
 - `linux_compat.h`, 20
 - `__skb_append`
 - `skbuff.c`, 36
 - `__skb_dequeue`
 - `skbuff.c`, 37
 - `__skb_dequeue_tail`
 - `skbuff.c`, 38
 - `__skb_insert`
 - `skbuff.c`, 36
 - `__skb_pull`
 - `skbuff.c`, 36
 - `__skb_push`
 - `skbuff.c`, 36
 - `__skb_put`
 - `skbuff.c`, 36
 - `__skb_queue_head`
 - `skbuff.c`, 38
 - `__skb_queue_purge`
 - `skbuff.c`, 38
 - `__skb_queue_tail`
 - `skbuff.c`, 38
 - `__skb_trim`
 - `skbuff.c`, 36
 - `__skb_unlink`
 - `skbuff.c`, 36
 - `__sti`
 - `linux_compat.h`, 20
 - `__u16`
 - `linux_compat.h`, 21
 - `__u32`
 - `linux_compat.h`, 21
 - `__u8`
 - `linux_compat.h`, 21
- `access_ok`
 - `linux_compat.h`, 20
 - `add_timer`
 - `timer.c`, 43
 - `ALLOC_COMPL_TIMEOUT`
 - `hermes.c`, 5
 - `alloc_etherdev`
 - `net_init.c`, 23
 - `alloc_orinocodev`
 - `orinoco.c`, 32
 - `orinoco.h`, 34
 - `alloc_skb`
 - `skbuff.c`, 38
 - `allow_ibss`
 - `orinoco_private`, 4
 - `ap_density`
 - `orinoco_private`, 4
 - `atomic_dec_and_test`
 - `linux_compat.h`, 21
 - `atomic_inc`
 - `linux_compat.h`, 20
 - `atomic_read`
 - `linux_compat.h`, 21
 - `atomic_t`
 - `linux_compat.h`, 21
 - `attach_orinoco_dev`
 - `ori.c`, 27
 - `bitrate_table`
 - `orinoco.c`, 32
 - `BITRATE_TABLE_SIZE`
-

- orinoco.c, 31
- bitratemode
 - orinoco_private, 4
- br_write_lock_bh
 - linux_compat.h, 20
- br_write_unlock_bh
 - linux_compat.h, 20
- broken_cor_reset
 - orinoco_private, 3
- BUG
 - linux_compat.h, 20
- capable
 - linux_compat.h, 20
- card
 - orinoco_private, 3
- channel
 - orinoco_private, 4
- channel_frequency
 - orinoco.c, 33
- channel_mask
 - orinoco_private, 3
- clear_bit
 - linux_compat.c, 18
 - linux_compat.h, 21
- cli
 - linux_compat.h, 20
- CMD_BUSY_TIMEOUT
 - hermes.c, 5
- CMD_COMPL_TIMEOUT
 - hermes.c, 5
- CMD_INIT_TIMEOUT
 - hermes.c, 5
- config_option_reset
 - ori.c, 27
- copy_from_user
 - linux_compat.h, 20
- copy_to_user
 - linux_compat.h, 20
- cpu_to_le16
 - linux_compat.h, 19
- create_proc_entry
 - linux_compat.h, 20
- create_proc_read_entry
 - linux_compat.h, 20
- DEBUG
 - hermes.c, 5
 - orinoco.h, 34
- del_timer
 - timer.c, 43
- desired_essid
 - orinoco_private, 4
- dev_alloc
 - netdev.c, 25
- dev_alloc_name
 - netdev.c, 25
- dev_alloc_skb
 - skbuff.c, 39
- devno
 - ori.c, 26
- dir_dev
 - orinoco_private, 4
- DISPLAY_BYTES
 - orinoco.c, 31
- DISPLAY_STRING
 - orinoco.c, 31
- DISPLAY_WORDS
 - orinoco.c, 31
- DISPLAY_XSTRING
 - orinoco.c, 31
- do_gettimeofday
 - linux_compat.c, 18
- DO_TRACE_HERMES
 - hermes.h, 10
- DREG
 - show_hermes.c, 35
 - trace_init.c, 44
- DUMMY_FID
 - orinoco.c, 31
- dynamic_cfg_rid_table
 - tracehermes.c, 46
- encaps_hdr
 - orinoco.c, 32
- ENCAPS_OVERHEAD
 - orinoco.c, 31
- env
 - ori.c, 26
- ETH_ALEN
 - linux_compat.h, 19
- ether_setup

- net_init.c, 22
- EXPORT_SYMBOL
 - linux_compat.h, 20
 - net_init.c, 22, 23
 - tracehermes.c, 45
- firmware_type
 - orinoco_private, 3
- FIRMWARE_TYPE_AGERE
 - orinoco.h, 34
- FIRMWARE_TYPE_INTERSIL
 - orinoco.h, 34
- FIRMWARE_TYPE_SYMBOL
 - orinoco.h, 34
- frag_thresh
 - orinoco_private, 4
- GFP_ATOMIC
 - linux_compat.h, 21
- handler
 - ori.c, 26
- hard_reset
 - orinoco_private, 3
- has_big_wep
 - orinoco_private, 3
- has_ibss
 - orinoco_private, 3
- has_ibss_any
 - orinoco_private, 3
- has_mwo
 - orinoco_private, 3
- has_pm
 - orinoco_private, 3
- has_port3
 - orinoco_private, 3
- has_preamble
 - orinoco_private, 3
- has_sensitivity
 - orinoco_private, 3
- has_wep
 - orinoco_private, 3
- hermes.c, 5
 - ALLOC_COMPL_TIMEOUT, 5
 - CMD_BUSY_TIMEOUT, 5
 - CMD_COMPL_TIMEOUT, 5
 - CMD_INIT_TIMEOUT, 5
 - DEBUG, 5
 - hermes_allocate, 6
 - hermes_bap_pread, 6
 - hermes_bap_pwrite, 6
 - hermes_docmd_trace, 6
 - hermes_docmd_wait, 6
 - hermes_read_ltv, 6
 - hermes_reset, 6
 - hermes_struct_init, 6
 - hermes_write_ltv, 6
 - IO_TYPE, 5
- hermes.h, 7
 - DO_TRACE_HERMES, 10
 - HERMES_16BIT_-REGSPACING, 10
 - HERMES_32BIT_-REGSPACING, 10
 - HERMES_802_11_OFFSET, 9
 - HERMES_802_2_OFFSET, 9
 - HERMES_802_3_OFFSET, 9
 - HERMES_ALLOC_LEN_MAX, 7
 - HERMES_ALLOC_LEN_MIN, 7
 - hermes_allocate, 10
 - HERMES_ALLOCFID, 8
 - HERMES_AUXDATA, 8
 - HERMES_AUXOFFSET, 8
 - HERMES_AUXPAGE, 8
 - HERMES_BAP_BUSY_-TIMEOUT, 9
 - HERMES_BAP_DATALEN_-MAX, 7
 - HERMES_BAP_OFFSET_MAX, 7
 - hermes_bap_pread, 11
 - hermes_bap_pwrite, 11
 - HERMES_BYTES_TO_-RECLen, 10
 - HERMES_CHINFORESULT_-MAX, 7
 - HERMES_CMD, 7
 - HERMES_CMD_ACCESS, 9
 - HERMES_CMD_AINFO, 8
 - HERMES_CMD_ALLOC, 9
 - HERMES_CMD_BUSY, 8

- HERMES_CMD_CLRPRST, 9
- HERMES_CMD_CMDCODE, 8
- HERMES_CMD_DIAG, 9
- HERMES_CMD_DISABLE, 9
- HERMES_CMD_DOWNLD, 9
- HERMES_CMD_ENABLE, 9
- HERMES_CMD_INIT, 9
- HERMES_CMD_INQUIRE, 9
- HERMES_CMD_MACPORT, 8
- HERMES_CMD_MONITOR, 9
- HERMES_CMD_NOTIFY, 9
- HERMES_CMD_PROGMODE, 8
- HERMES_CMD_RECL, 8
- HERMES_CMD_TX, 9
- HERMES_CMD_WRITE, 8
- HERMES_CONTROL, 8
- HERMES_DATA0, 8
- HERMES_DATA1, 8
- HERMES_DEBUG_BUFSIZE, 9
- HERMES_DESCRIPTOR_-
OFFSET, 9
- hermes_disable_port, 10
- hermes_docmd_wait, 10
- hermes_enable_interrupt, 11
- hermes_enable_port, 10
- HERMES_EV_ALLOC, 9
- HERMES_EV_CMD, 8
- HERMES_EV_DTIM, 8
- HERMES_EV_INFDROP, 8
- HERMES_EV_INFO, 8
- HERMES_EV_RX, 9
- HERMES_EV_TICK, 8
- HERMES_EV_TX, 9
- HERMES_EV_TXEXC, 9
- HERMES_EV_WTERR, 8
- HERMES_EVACK, 8
- HERMES_EVSTAT, 8
- HERMES_INFOFID, 8
- HERMES_INQ_LINKSTATUS, 9
- HERMES_INQ_SCAN, 9
- HERMES_INQ_TALLIES, 9
- hermes_inquire, 10
- HERMES_INTEN, 8
- HERMES_IO, 10
- HERMES_LTV_LEN_MAX, 7
- HERMES_MAGIC, 7
- HERMES_MAX_MULTICAST, 7
- HERMES_MEM, 10
- HERMES_MONITOR_-
DISABLE, 9
- HERMES_MONITOR_-
ENABLE, 9
- HERMES_NUMPORTS_MAX, 7
- HERMES_OFFSET0, 8
- HERMES_OFFSET1, 8
- HERMES_OFFSET_BUSY, 8
- HERMES_OFFSET_DATAOFF, 8
- HERMES_OFFSET_ERR, 8
- HERMES_PARAM0, 7
- HERMES_PARAM1, 7
- HERMES_PARAM2, 8
- HERMES_PDA_LEN_MAX, 7
- HERMES_PDA_RECS_MAX, 7
- HERMES_PDR_LEN_MAX, 7
- HERMES_PORTID_MAX, 7
- hermes_present, 10
- hermes_read_ltv, 11
- HERMES_READ_RECORD, 10
- hermes_read_reg, 10
- hermes_read_regn, 10
- hermes_read_wordrec, 11
- hermes_read_words, 11
- HERMES_RECLEN_TO_-
BYTES, 10
- hermes_reset, 10
- HERMES_RESP0, 8
- HERMES_RESP1, 8
- HERMES_RESP2, 8
- hermes_response_t, 10
- HERMES_RXFID, 8
- HERMES_RXSTAT_1042, 9
- HERMES_RXSTAT_BADCRC, 9
- HERMES_RXSTAT_ERR, 9
- HERMES_RXSTAT_MACPORT, 9
- HERMES_RXSTAT_MSGTYPE, 9
- HERMES_RXSTAT_TUNNEL, 9

- HERMES_RXSTAT_-
UNDECRYPTABLE,
9
- HERMES_RXSTAT_WMP, 9
- HERMES_SCANRESULT_-
MAX, 7
- HERMES_SELECT0, 8
- HERMES_SELECT1, 8
- hermes_set_irqmask, 11
- HERMES_STATUS, 8
- HERMES_STATUS_-
CMDCODE, 8
- HERMES_STATUS_RESULT, 8
- hermes_struct_init, 10
- HERMES_SWSUPPORT0, 8
- HERMES_SWSUPPORT1, 8
- HERMES_SWSUPPORT2, 8
- hermes_t, 10
- hermes_trace_dynamic_cfg, 11
- hermes_trace_mac_info, 11
- hermes_trace_modem_info, 11
- hermes_trace_nic_info, 11
- hermes_trace_static_cfg, 11
- HERMES_TXCOMPLFID, 8
- HERMES_TXCTRL_802_11, 9
- HERMES_TXCTRL_ALT_-
RTRY, 9
- HERMES_TXCTRL_TX_EX, 9
- HERMES_TXCTRL_TX_OK, 9
- HERMES_TXSTAT_AGEDERR,
9
- HERMES_TXSTAT_DISCON, 9
- HERMES_TXSTAT_FORMERR,
9
- HERMES_TXSTAT_-
RETRYERR, 9
- hermes_write_ltv, 11
- HERMES_WRITE_RECORD, 10
- hermes_write_reg, 10
- hermes_write_regn, 10
- hermes_write_wordrec, 11
- hermes_write_words, 11
- packed, 11
- HERMES_16BIT_REGSPACING
hermes.h, 10
- HERMES_32BIT_REGSPACING
hermes.h, 10
- HERMES_802_11_OFFSET
hermes.h, 9
- HERMES_802_2_OFFSET
hermes.h, 9
- HERMES_802_3_OFFSET
hermes.h, 9
- HERMES_ALLOC_LEN_MAX
hermes.h, 7
- HERMES_ALLOC_LEN_MIN
hermes.h, 7
- hermes_allocate
hermes.c, 6
hermes.h, 10
- HERMES_ALLOCFID
hermes.h, 8
- HERMES_AUXDATA
hermes.h, 8
- HERMES_AUXOFFSET
hermes.h, 8
- HERMES_AUXPAGE
hermes.h, 8
- HERMES_BAP_BUSY_TIMEOUT
hermes.h, 9
- HERMES_BAP_DATALEN_MAX
hermes.h, 7
- HERMES_BAP_OFFSET_MAX
hermes.h, 7
- hermes_bap_pread
hermes.c, 6
hermes.h, 11
- hermes_bap_pwrite
hermes.c, 6
hermes.h, 11
- HERMES_BYTES_TO_RECLN
hermes.h, 10
- HERMES_CHINFORESULT_MAX
hermes.h, 7
- HERMES_CMD
hermes.h, 7
- HERMES_CMD_ACCESS
hermes.h, 9
- HERMES_CMD_AINFO
hermes.h, 8
- HERMES_CMD_ALLOC
hermes.h, 9

- HERMES_CMD_BUSY
 - hermes.h, 8
- HERMES_CMD_CLRPRST
 - hermes.h, 9
- HERMES_CMD_CMDCODE
 - hermes.h, 8
- HERMES_CMD_DIAG
 - hermes.h, 9
- HERMES_CMD_DISABLE
 - hermes.h, 9
- HERMES_CMD_DOWNLD
 - hermes.h, 9
- HERMES_CMD_ENABLE
 - hermes.h, 9
- HERMES_CMD_INIT
 - hermes.h, 9
- HERMES_CMD_INQUIRE
 - hermes.h, 9
- HERMES_CMD_MACPORT
 - hermes.h, 8
- HERMES_CMD_MONITOR
 - hermes.h, 9
- HERMES_CMD_NOTIFY
 - hermes.h, 9
- HERMES_CMD_PROGMODE
 - hermes.h, 8
- HERMES_CMD_RECL
 - hermes.h, 8
- HERMES_CMD_TX
 - hermes.h, 9
- HERMES_CMD_WRITE
 - hermes.h, 8
- HERMES_CONTROL
 - hermes.h, 8
- HERMES_DATA0
 - hermes.h, 8
- HERMES_DATA1
 - hermes.h, 8
- HERMES_DEBUG_BUFSIZE
 - hermes.h, 9
- HERMES_DESCRIPTOR_OFFSET
 - hermes.h, 9
- hermes_disable_port
 - hermes.h, 10
- hermes_docmd_trace
 - hermes.c, 6
- hermes_docmd_wait
 - hermes.c, 6
 - hermes.h, 10
- hermes_enable_interrupt
 - hermes.h, 11
 - hermes_qnx.c, 12
- hermes_enable_port
 - hermes.h, 10
- HERMES_EV_ALLOC
 - hermes.h, 9
- HERMES_EV_CMD
 - hermes.h, 8
- HERMES_EV_DTIM
 - hermes.h, 8
- HERMES_EV_INFDDROP
 - hermes.h, 8
- HERMES_EV_INFO
 - hermes.h, 8
- HERMES_EV_RX
 - hermes.h, 9
- HERMES_EV_TICK
 - hermes.h, 8
- HERMES_EV_TX
 - hermes.h, 9
- HERMES_EV_TXEXC
 - hermes.h, 9
- HERMES_EV_WTERR
 - hermes.h, 8
- HERMES_EVACK
 - hermes.h, 8
- HERMES_EVSTAT
 - hermes.h, 8
- HERMES_INFOFID
 - hermes.h, 8
- HERMES_INQ_LINKSTATUS
 - hermes.h, 9
- HERMES_INQ_SCAN
 - hermes.h, 9
- HERMES_INQ_TALLIES
 - hermes.h, 9
- hermes_inquire
 - hermes.h, 10
- HERMES_INTEN
 - hermes.h, 8
- HERMES_IO
 - hermes.h, 10

-
- HERMES_LTV_LEN_MAX
 - hermes.h, 7
 - HERMES_MAGIC
 - hermes.h, 7
 - HERMES_MAX_MULTICAST
 - hermes.h, 7
 - HERMES_MEM
 - hermes.h, 10
 - HERMES_MONITOR_DISABLE
 - hermes.h, 9
 - HERMES_MONITOR_ENABLE
 - hermes.h, 9
 - hermes_multicast_t
 - hermes_rid.h, 16
 - HERMES_NUMPORTS_MAX
 - hermes.h, 7
 - HERMES_OFFSET0
 - hermes.h, 8
 - HERMES_OFFSET1
 - hermes.h, 8
 - HERMES_OFFSET_BUSY
 - hermes.h, 8
 - HERMES_OFFSET_DATAOFF
 - hermes.h, 8
 - HERMES_OFFSET_ERR
 - hermes.h, 8
 - HERMES_PARAM0
 - hermes.h, 7
 - HERMES_PARAM1
 - hermes.h, 7
 - HERMES_PARAM2
 - hermes.h, 8
 - HERMES_PDA_LEN_MAX
 - hermes.h, 7
 - HERMES_PDA_RECS_MAX
 - hermes.h, 7
 - HERMES_PDR_LEN_MAX
 - hermes.h, 7
 - HERMES_PORTID_MAX
 - hermes.h, 7
 - hermes_present
 - hermes.h, 10
 - hermes_qnx.c, 12
 - hermes_enable_interrupt, 12
 - hermes_read_wordrec, 12
 - hermes_read_words, 12
 - hermes_set_irqmask, 12
 - hermes_write_wordrec, 12
 - hermes_write_words, 12
 - hermes_read_ltv
 - hermes.c, 6
 - hermes.h, 11
 - HERMES_READ_RECORD
 - hermes.h, 10
 - hermes_read_reg
 - hermes.h, 10
 - hermes_read_regn
 - hermes.h, 10
 - hermes_read_wordrec
 - hermes.h, 11
 - hermes_qnx.c, 12
 - hermes_read_words
 - hermes.h, 11
 - hermes_qnx.c, 12
 - HERMES_RECLEN_TO_BYTES
 - hermes.h, 10
 - hermes_reset
 - hermes.c, 6
 - hermes.h, 10
 - HERMES_RESP0
 - hermes.h, 8
 - HERMES_RESP1
 - hermes.h, 8
 - HERMES_RESP2
 - hermes.h, 8
 - hermes_response_t
 - hermes.h, 10
 - hermes_rid.h, 12
 - hermes_multicast_t, 16
 - HERMES_RID_-
 - AUTHENTICATIONALGORITHMS, 15
 - HERMES_RID_BUILDSEQ, 16
 - HERMES_RID_CCAMODE, 16
 - HERMES_RID_-
 - CFIACRANGES, 15
 - HERMES_RID_-
 - CFIACRANGES2, 15
 - HERMES_RID_-
 - CFISUPRANGE, 15
 - HERMES_RID_CFPOLLABLE, 15
-

- HERMES_RID.-
 CHANNELLIST, 15
- HERMES_RID_CIS, 15
- HERMES_RID.-
 CNFALTRETRYCOUNT,
 14
- HERMES_RID.-
 CNFAPPCFINFO, 14
- HERMES_RID.-
 CNFAUTHENTICATESTATION,
 14
- HERMES_RID.-
 CNFAUTHENTICATION,
 13
- HERMES_RID.-
 CNFAUTHENTICATIONRSPTO,
 14
- HERMES_RID.-
 CNFBASICRATES, 14
- HERMES_RID.-
 CNFBASICRATES_-
 SYMBOL, 14
- HERMES_RID.-
 CNFBEACONINT, 14
- HERMES_RID.-
 CNFCHANNELINFOREQUEST,
 14
- HERMES_RID.-
 CNFCREATEIBSS, 14
- HERMES_RID.-
 CNFDEFAULTKEY0,
 13
- HERMES_RID.-
 CNFDEFAULTKEY1,
 13
- HERMES_RID.-
 CNFDEFAULTKEY2,
 13
- HERMES_RID.-
 CNFDEFAULTKEY3,
 13
- HERMES_RID.-
 CNFDESIREDDSSID,
 13
- HERMES_RID.-
 CNFENHSECURITY,
 14
- HERMES_RID.-
 CNFEXCLUDELONGPREAMBLE,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD0,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD1,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD2,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD3,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD4,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD5,
 14
- HERMES_RID.-
 CNFFRAGMENTATIONTHRESHOLD6,
 14
- HERMES_RID.-
 CNFGROUPADDRESSES,
 14
- HERMES_RID.-
 CNFHSTAUTHENTICATION,
 13
- HERMES_RID.-
 CNFJOINREQUEST,
 14
- HERMES_RID.-
 CNFKEYLENGTH_-
 SYMBOL, 13
- HERMES_RID.-
 CNFMANDATORYBSSID_-
 SYMBOL, 13
- HERMES_RID.-
 CNFMAXASSOCSTA,
 13

- HERMES_RID_-
CNFMAXDATALEN,
13
- HERMES_RID_-
CNFMAXSLEEPPDURATION,
13
- HERMES_RID_-CNFMMLIFE,
13
- HERMES_RID_-
CNFMULTICASTPMBUFFERING,
13
- HERMES_RID_-
CNFMULTICASTRECEIVE,
13
- HERMES_RID_-
CNFMWOROBUST_-
AGERE, 13
- HERMES_RID_-
CNFOWNATIMWINDOW,
13
- HERMES_RID_-
CNFOWNCHANNEL,
13
- HERMES_RID_-
CNFOWNDTIMPERIOD,
13
- HERMES_RID_-
CNFOWNMACADDR,
13
- HERMES_RID_-
CNFOWNNAME, 13
- HERMES_RID_-CNFOWNSSID,
13
- HERMES_RID_-
CNFPMENABLED, 13
- HERMES_RID_-CNFPMEPS, 13
- HERMES_RID_-
CNFPMHOLDOVERDURATION,
13
- HERMES_RID_-
CNFPORTTYPE, 13
- HERMES_RID_-
CNFPREAMBLE_-
SYMBOL, 14
- HERMES_RID_-
CNFPRIORITYQUSAGE,
14
- HERMES_RID_-
CNFPROMISCUOUSMODE,
14
- HERMES_RID_-
CNFRVCRCERROR,
13
- HERMES_RID_-
CNFROAMINGMODE,
13
- HERMES_RID_-
CNFRTSTHRESHOLD,
14
- HERMES_RID_-
CNFRTSTHRESHOLD0,
14
- HERMES_RID_-
CNFRTSTHRESHOLD1,
14
- HERMES_RID_-
CNFRTSTHRESHOLD2,
14
- HERMES_RID_-
CNFRTSTHRESHOLD3,
14
- HERMES_RID_-
CNFRTSTHRESHOLD4,
14
- HERMES_RID_-
CNFRTSTHRESHOLD5,
14
- HERMES_RID_-
CNFRTSTHRESHOLD6,
14
- HERMES_RID_-
CNFSCANREQUEST,
14
- HERMES_RID_-
CNFSHORTPREAMBLE,
14
- HERMES_RID_-
CNFSTAPCFINFO, 14
- HERMES_RID_-
CNFSUPPORTEDRATES,
14

- HERMES_RID_-
CNFSYSTEMSCALE,
13
- HERMES_RID_-
CNFTHIRTY2TALLY,
14
- HERMES_RID_-
CNFTICKTIME, 14
- HERMES_RID_-CNFTIMCTRL,
14
- HERMES_RID_-
CNFTXCONTROL, 13
- HERMES_RID_-CNFTXKEY_-
AGERE, 14
- HERMES_RID_-
CNFTXRATECONTROL,
14
- HERMES_RID_-
CNFWDSADDRESS,
13
- HERMES_RID_-
CNFWDSADDRESS1,
13
- HERMES_RID_-
CNFWDSADDRESS2,
13
- HERMES_RID_-
CNFWDSADDRESS3,
13
- HERMES_RID_-
CNFWDSADDRESS4,
13
- HERMES_RID_-
CNFWDSADDRESS5,
13
- HERMES_RID_-
CNFWDSADDRESS6,
13
- HERMES_RID_-
CNFWEPDEFAULTKEYID,
13
- HERMES_RID_-
CNFWEPENABLED_-
AGERE, 13
- HERMES_RID_-
CNFWEPFLAGS_-
INTERSIL, 13
- HERMES_RID_-
CNFWEPKEYMAPPINGTABLE,
13
- HERMES_RID_-
CNFWEPKEYS_AGERE,
14
- HERMES_RID_-
COMMSQUALITY, 15
- HERMES_RID_-
CURRENTBEACONINTERVAL,
15
- HERMES_RID_-
CURRENTBSSID, 15
- HERMES_RID_-
CURRENTCHANNEL,
15
- HERMES_RID_-
CURRENTPOWERSTATE,
15
- HERMES_RID_-
CURRENTSCALETHRESHOLDS,
15
- HERMES_RID_-
CURRENTSSID, 15
- HERMES_RID_-
CURRENTTXRATE,
15
- HERMES_RID_-
CURRENTTXRATE1,
15
- HERMES_RID_-
CURRENTTXRATE2,
15
- HERMES_RID_-
CURRENTTXRATE3,
15
- HERMES_RID_-
CURRENTTXRATE4,
15
- HERMES_RID_-
CURRENTTXRATE5,
15
- HERMES_RID_-
CURRENTTXRATE6,
15

- HERMES_RID_-
 DOWNLOADBUFFER,
 15
- HERMES_RID_FWID, 16
- HERMES_RID_-
 LONGRETRYLIMIT,
 15
- HERMES_RID_-
 MAXLOADTIME, 14
- HERMES_RID_-
 MAXRECEIVELIFETIME,
 15
- HERMES_RID_-
 MAXTRANSMITLIFETIME,
 15
- HERMES_RID_-
 MFIACRANGES, 15
- HERMES_RID_-
 MFISUPRANGE, 15
- HERMES_RID_NICID, 15
- HERMES_RID_NICSERNUM,
 15
- HERMES_RID_-
 OWNMACADDR, 15
- HERMES_RID_PHYTYPE, 15
- HERMES_RID_PORTSTATUS,
 15
- HERMES_RID_PRIID, 15
- HERMES_RID_-
 PRISUPRANGE, 15
- HERMES_RID_-
 PRIVACYOPTIONIMPLEMENTED,
 15
- HERMES_RID_-
 PROTOCOLRSPTIME,
 15
- HERMES_RID_-
 REGULATORYDOMAINS,
 15
- HERMES_RID_-
 SCANRESULTSTABLE,
 15
- HERMES_RID_-
 SECONDARYVERSION_-
 SYMBOL, 15
- HERMES_RID_-
 SHORTRETRYLIMIT,
 15
- HERMES_RID_STAID, 15
- HERMES_RID_-
 STASUPRANGE, 15
- HERMES_RID_-
 SUPPORTEDDATARATES,
 16
- HERMES_RID_TEMPTYPE, 15
- packed, 16
- HERMES_RID_-
 AUTHENTICATIONALGORITHMS
 hermes_rid.h, 15
- HERMES_RID_BUILDSEQ
 hermes_rid.h, 16
- HERMES_RID_CCAMODE
 hermes_rid.h, 16
- HERMES_RID_CFIACRANGES
 hermes_rid.h, 15
- HERMES_RID_CFIACRANGES2
 hermes_rid.h, 15
- HERMES_RID_CFISUPRANGE
 hermes_rid.h, 15
- HERMES_RID_CFPOLLABLE
 hermes_rid.h, 15
- HERMES_RID_CHANNELLIST
 hermes_rid.h, 15
- HERMES_RID_CIS
 hermes_rid.h, 15
- HERMES_RID_-
 CNFALTRETRYCOUNT
 hermes_rid.h, 14
- HERMES_RID_CNFAPPCFINFO
 hermes_rid.h, 14
- HERMES_RID_-
 CNFAUTHENTICATESTATION
 hermes_rid.h, 14
- HERMES_RID_-
 CNFAUTHENTICATION
 hermes_rid.h, 13
- HERMES_RID_-
 CNFAUTHENTICATIONRSPTO
 hermes_rid.h, 14
- HERMES_RID_CNFBASICRATES
 hermes_rid.h, 14

- hermes_rid.h, 13
- HERMES_RID_-
 - CNFOWNMACADDR
 - hermes_rid.h, 13
- HERMES_RID_CNFOWNNAME
 - hermes_rid.h, 13
- HERMES_RID_CNFOWNSSID
 - hermes_rid.h, 13
- HERMES_RID_CNFPMENABLED
 - hermes_rid.h, 13
- HERMES_RID_CNFPMEPS
 - hermes_rid.h, 13
- HERMES_RID_-
 - CNFPMHOLDOVERDURATION
 - hermes_rid.h, 13
- HERMES_RID_CNFPORTTYPE
 - hermes_rid.h, 13
- HERMES_RID_CNFPREAMBLE_-
 - SYMBOL
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFPRIORITYQUSAGE
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFPROMISCUOUSMODE
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRVCRCERROR
 - hermes_rid.h, 13
- HERMES_RID_-
 - CNFROAMINGMODE
 - hermes_rid.h, 13
- HERMES_RID_-
 - CNFRTSTHRESHOLD
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD0
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD1
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD2
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD3
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD4
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD5
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFRTSTHRESHOLD6
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFSCANREQUEST
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFSHORTPREAMBLE
 - hermes_rid.h, 14
- HERMES_RID_CNFSTAPCFINFO
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFSUPPORTEDRATES
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFSYSTEMSCALE
 - hermes_rid.h, 13
- HERMES_RID_-
 - CNFTHIRTY2TALLY
 - hermes_rid.h, 14
- HERMES_RID_CNFTICKTIME
 - hermes_rid.h, 14
- HERMES_RID_CNFTIMCTRL
 - hermes_rid.h, 14
- HERMES_RID_CNFTXCONTROL
 - hermes_rid.h, 13
- HERMES_RID_CNFTXKEY_AGERE
 - hermes_rid.h, 14
- HERMES_RID_-
 - CNFTXRATECONTROL
 - hermes_rid.h, 14
- HERMES_RID_CNFWDSADDRESS
 - hermes_rid.h, 13
- HERMES_RID_-
 - CNFWDSADDRESS1
 - hermes_rid.h, 13
- HERMES_RID_-
 - CNFWDSADDRESS2
 - hermes_rid.h, 13

-
- HERMES_RID_-
 - CNFWDSADDRESS3
 - hermes_rid.h, 13
 - HERMES_RID_-
 - CNFWDSADDRESS4
 - hermes_rid.h, 13
 - HERMES_RID_-
 - CNFWDSADDRESS5
 - hermes_rid.h, 13
 - HERMES_RID_-
 - CNFWDSADDRESS6
 - hermes_rid.h, 13
 - HERMES_RID_-
 - CNFWEPDEFAULTKEYID
 - hermes_rid.h, 13
 - HERMES_RID_-
 - CNFWEPENABLED_-
 - AGERE
 - hermes_rid.h, 13
 - HERMES_RID_CNFWEPFLAGS_-
 - INTERSIL
 - hermes_rid.h, 13
 - HERMES_RID_-
 - CNFWEPKEYMAPPINGTABLE
 - hermes_rid.h, 13
 - HERMES_RID_CNFWEPKEYS_-
 - AGERE
 - hermes_rid.h, 14
 - HERMES_RID_COMMSQUALITY
 - hermes_rid.h, 15
 - HERMES_RID_-
 - CURRENTBEACONINTERVAL
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTBSSID
 - hermes_rid.h, 15
 - HERMES_RID_-
 - CURRENTCHANNEL
 - hermes_rid.h, 15
 - HERMES_RID_-
 - CURRENTPOWERSTATE
 - hermes_rid.h, 15
 - HERMES_RID_-
 - CURRENTSCALETHRESHOLDS
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTSSID
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE1
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE2
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE3
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE4
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE5
 - hermes_rid.h, 15
 - HERMES_RID_CURRENTTXRATE6
 - hermes_rid.h, 15
 - HERMES_RID_-
 - DOWNLOADBUFFER
 - hermes_rid.h, 15
 - HERMES_RID_FWID
 - hermes_rid.h, 16
 - HERMES_RID_LONGRETRYLIMIT
 - hermes_rid.h, 15
 - HERMES_RID_MAXLOADTIME
 - hermes_rid.h, 14
 - HERMES_RID_-
 - MAXRECEIVELIFETIME
 - hermes_rid.h, 15
 - HERMES_RID_-
 - MAXTRANSMITLIFETIME
 - hermes_rid.h, 15
 - HERMES_RID_MFIACRANGES
 - hermes_rid.h, 15
 - HERMES_RID_MFISUPRANGE
 - hermes_rid.h, 15
 - HERMES_RID_NICID
 - hermes_rid.h, 15
 - HERMES_RID_NICSERNUM
 - hermes_rid.h, 15
 - HERMES_RID_OWNMACADDR
 - hermes_rid.h, 15
 - HERMES_RID_PHYTYPE
 - hermes_rid.h, 15
 - HERMES_RID_PORTSTATUS
 - hermes_rid.h, 15
 - HERMES_RID_PRIID
 - hermes_rid.h, 15
 - HERMES_RID_PRISUPRANGE
-

- hermes_rid.h, 15
- HERMES_RID_-
 - PRIVACYOPTIONIMPLEMENTEDHERMES_SCANRESULT_MAX
 - hermes.h, 9
 - hermes_rid.h, 15
- HERMES_RID_-
 - PROTOCOLRSPTIME
 - hermes_rid.h, 15
- HERMES_RID_-
 - REGULATORYDOMAINS
 - hermes_rid.h, 15
- HERMES_RID_-
 - SCANRESULTSTABLE
 - hermes_rid.h, 15
- HERMES_RID_-
 - SECONDARYVERSION_-
 - SYMBOL
 - hermes_rid.h, 15
- HERMES_RID_-
 - SHORTRETRYLIMIT
 - hermes_rid.h, 15
- HERMES_RID_STAID
 - hermes_rid.h, 15
- HERMES_RID_STASUPRANGE
 - hermes_rid.h, 15
- HERMES_RID_-
 - SUPPORTEDDATARATES
 - hermes_rid.h, 16
- HERMES_RID_TEMPTYPE
 - hermes_rid.h, 15
- HERMES_RXFID
 - hermes.h, 8
- HERMES_RXSTAT_1042
 - hermes.h, 9
- HERMES_RXSTAT_BADCRC
 - hermes.h, 9
- HERMES_RXSTAT_ERR
 - hermes.h, 9
- HERMES_RXSTAT_MACPORT
 - hermes.h, 9
- HERMES_RXSTAT_MSGTYPE
 - hermes.h, 9
- HERMES_RXSTAT_TUNNEL
 - hermes.h, 9
- HERMES_RXSTAT_-
 - UNDECRYPTABLE
 - hermes.h, 9
- HERMES_RXSTAT_WMP
 - hermes.h, 9
- HERMES_SCANRESULT_MAX
 - hermes.h, 7
- HERMES_SELECT0
 - hermes.h, 8
- HERMES_SELECT1
 - hermes.h, 8
- hermes_set_irqmask
 - hermes.h, 11
 - hermes_qnx.c, 12
- HERMES_STATUS
 - hermes.h, 8
- HERMES_STATUS_CMDCODE
 - hermes.h, 8
- HERMES_STATUS_RESULT
 - hermes.h, 8
- hermes_struct_init
 - hermes.c, 6
 - hermes.h, 10
- HERMES_SWSUPPORT0
 - hermes.h, 8
- HERMES_SWSUPPORT1
 - hermes.h, 8
- HERMES_SWSUPPORT2
 - hermes.h, 8
- hermes_t
 - hermes.h, 10
- hermes_trace_dynamic_cfg
 - hermes.h, 11
 - tracehermes.c, 45
- hermes_trace_mac_info
 - hermes.h, 11
 - tracehermes.c, 45
- HERMES_TRACE_MAX_FIELD_-
 - SIZE
 - tracehermes.c, 45
- hermes_trace_modem_info
 - hermes.h, 11
 - tracehermes.c, 45
- hermes_trace_nic_info
 - hermes.h, 11
 - tracehermes.c, 45
- hermes_trace_show_rid_table
 - tracehermes.c, 45
- hermes_trace_static_cfg

- hermes.h, 11
- tracehermes.c, 45
- HERMES_TXCOMPLFID
 - hermes.h, 8
- HERMES_TXCTRL_802_11
 - hermes.h, 9
- HERMES_TXCTRL_ALT_RETRY
 - hermes.h, 9
- HERMES_TXCTRL_TX_EX
 - hermes.h, 9
- HERMES_TXCTRL_TX_OK
 - hermes.h, 9
- HERMES_TXSTAT_AGEDERR
 - hermes.h, 9
- HERMES_TXSTAT_DISCON
 - hermes.h, 9
- HERMES_TXSTAT_FORMERR
 - hermes.h, 9
- HERMES_TXSTAT_RETRYERR
 - hermes.h, 9
- hermes_write_ltv
 - hermes.c, 6
 - hermes.h, 11
- HERMES_WRITE_RECORD
 - hermes.h, 10
- hermes_write_reg
 - hermes.h, 10
- hermes_write_regn
 - hermes.h, 10
- hermes_write_wordrec
 - hermes.h, 11
 - hermes_qnx.c, 12
- hermes_write_words
 - hermes.h, 11
 - hermes_qnx.c, 12
- host_string
 - orinoco_private, 4
- htons
 - linux_compat.h, 20
- hw
 - orinoco_private, 3
- HZ
 - linux_compat.h, 20
- ibss_port
 - orinoco_private, 3
- ieee802_11.h, 16
 - IEEE802_11_DATA_LEN, 16
 - IEEE802_11_FCTL_FROMDS, 16
 - IEEE802_11_FCTL_FTYPE, 16
 - IEEE802_11_FCTL_-
 - MOREDATA, 17
 - IEEE802_11_FCTL_-
 - MOREFRAGS, 16
 - IEEE802_11_FCTL_ORDER, 17
 - IEEE802_11_FCTL_PM, 16
 - IEEE802_11_FCTL_RETRY, 16
 - IEEE802_11_FCTL_STYPE, 16
 - IEEE802_11_FCTL_TODS, 16
 - IEEE802_11_FCTL_VERS, 16
 - IEEE802_11_FCTL_WEP, 17
 - IEEE802_11_FRAME_LEN, 16
 - IEEE802_11_FTYPE_CTL, 17
 - IEEE802_11_FTYPE_DATA, 17
 - IEEE802_11_FTYPE_MGMT, 17
 - IEEE802_11_HLEN, 16
 - IEEE802_11_SCTL_FRAG, 17
 - IEEE802_11_SCTL_SEQ, 17
 - IEEE802_11_STYPE_ACK, 17
 - IEEE802_11_STYPE_ASSOC_-
 - REQ, 17
 - IEEE802_11_STYPE_ASSOC_-
 - RESP, 17
 - IEEE802_11_STYPE_ATIM, 17
 - IEEE802_11_STYPE_AUTH, 17
 - IEEE802_11_STYPE_BEACON, 17
 - IEEE802_11_STYPE_CFACK, 17
 - IEEE802_11_STYPE_-
 - CFACKPOLL, 17
 - IEEE802_11_STYPE_CFEND, 17
 - IEEE802_11_STYPE_-
 - CFENDACK, 17
 - IEEE802_11_STYPE_CFPOLL, 17
 - IEEE802_11_STYPE_CTS, 17
 - IEEE802_11_STYPE_DATA, 17
 - IEEE802_11_STYPE_DATA_-
 - CFACK, 17

- IEEE802.11_STYPE_DATA_-
CFACKPOLL, 17
- IEEE802.11_STYPE_DATA_-
CFPOLL, 17
- IEEE802.11_STYPE_DEAUTH,
17
- IEEE802.11_STYPE_-
DISASSOC, 17
- IEEE802.11_STYPE_-
NULLFUNC, 17
- IEEE802.11_STYPE_PROBE_-
REQ, 17
- IEEE802.11_STYPE_PROBE_-
RESP, 17
- IEEE802.11_STYPE_PSPOLL,
17
- IEEE802.11_STYPE_-
REASSOC_REQ, 17
- IEEE802.11_STYPE_-
REASSOC_RESP, 17
- IEEE802.11_STYPE_RTS, 17
packed, 17
- IEEE802.11_DATA_LEN
ieee802.11.h, 16
- IEEE802.11_FCTL_FROMDS
ieee802.11.h, 16
- IEEE802.11_FCTL_FTYPE
ieee802.11.h, 16
- IEEE802.11_FCTL_MOREDATA
ieee802.11.h, 17
- IEEE802.11_FCTL_MOREFRAGS
ieee802.11.h, 16
- IEEE802.11_FCTL_ORDER
ieee802.11.h, 17
- IEEE802.11_FCTL_PM
ieee802.11.h, 16
- IEEE802.11_FCTL_RETRY
ieee802.11.h, 16
- IEEE802.11_FCTL_STYPE
ieee802.11.h, 16
- IEEE802.11_FCTL_TODS
ieee802.11.h, 16
- IEEE802.11_FCTL_VERS
ieee802.11.h, 16
- IEEE802.11_FCTL_WEP
ieee802.11.h, 17
- IEEE802.11_FRAME_LEN
ieee802.11.h, 16
- IEEE802.11_FTYPE_CTL
ieee802.11.h, 17
- IEEE802.11_FTYPE_DATA
ieee802.11.h, 17
- IEEE802.11_FTYPE_MGMT
ieee802.11.h, 17
- IEEE802.11_HLEN
ieee802.11.h, 16
- IEEE802.11_SCTL_FRAG
ieee802.11.h, 17
- IEEE802.11_SCTL_SEQ
ieee802.11.h, 17
- IEEE802.11_STYPE_ACK
ieee802.11.h, 17
- IEEE802.11_STYPE_ASSOC_REQ
ieee802.11.h, 17
- IEEE802.11_STYPE_ASSOC_RESP
ieee802.11.h, 17
- IEEE802.11_STYPE_ATIM
ieee802.11.h, 17
- IEEE802.11_STYPE_AUTH
ieee802.11.h, 17
- IEEE802.11_STYPE_BEACON
ieee802.11.h, 17
- IEEE802.11_STYPE_CFACK
ieee802.11.h, 17
- IEEE802.11_STYPE_CFACKPOLL
ieee802.11.h, 17
- IEEE802.11_STYPE_CFEND
ieee802.11.h, 17
- IEEE802.11_STYPE_CFENDACK
ieee802.11.h, 17
- IEEE802.11_STYPE_CFPOLL
ieee802.11.h, 17
- IEEE802.11_STYPE_CTS
ieee802.11.h, 17
- IEEE802.11_STYPE_DATA
ieee802.11.h, 17
- IEEE802.11_STYPE_DATA_CFACK
ieee802.11.h, 17
- IEEE802.11_STYPE_DATA_-
CFACKPOLL
ieee802.11.h, 17
- IEEE802.11_STYPE_DATA_CFPOLL

- ieee802_11.h, 17
- IEEE802_11_STYPE_DEAUTH
 - ieee802_11.h, 17
- IEEE802_11_STYPE_DISASSOC
 - ieee802_11.h, 17
- IEEE802_11_STYPE_NULLFUNC
 - ieee802_11.h, 17
- IEEE802_11_STYPE_PROBE_REQ
 - ieee802_11.h, 17
- IEEE802_11_STYPE_PROBE_RESP
 - ieee802_11.h, 17
- IEEE802_11_STYPE_PSPOLL
 - ieee802_11.h, 17
- IEEE802_11_STYPE_REASSOC_-REQ
 - ieee802_11.h, 17
- IEEE802_11_STYPE_REASSOC_-RESP
 - ieee802_11.h, 17
- IEEE802_11_STYPE_RTS
 - ieee802_11.h, 17
- init_timer
 - timer.c, 43
- inline
 - linux_compat.h, 19
- install_orinoco_dev
 - ori.c, 27
- interrupt_id
 - ori.c, 26
- intr_proxy
 - ori.c, 26
- inw
 - linux_compat.h, 19
- IO_TYPE
 - hermes.c, 5
- ip_frag_mem
 - skbuff.c, 37
- IRQ_BAP
 - orinoco.c, 31
- iw_mode
 - orinoco_private, 3
- jiffies
 - linux_compat.h, 21
- KERN_CRIT
 - linux_compat.h, 20
- KERN_DEBUG
 - linux_compat.h, 20
- KERN_ERR
 - linux_compat.h, 20
- KERN_INFO
 - linux_compat.h, 20
- KERN_NOTICE
 - linux_compat.h, 20
- KERN_WARNING
 - linux_compat.h, 20
- keys
 - orinoco_private, 4
- kfree
 - linux_compat.h, 20
- kfree_skb
 - skbuff.c, 39
- kfree_skbmem
 - skbuff.c, 36
- KM_SKB_DATA_SOFTIRQ
 - linux_compat.h, 21
- kmalloc
 - linux_compat.h, 20
- LARGE_KEY_SIZE
 - orinoco.c, 31
- le16_to_cpu
 - linux_compat.h, 19
- le16_to_cpus
 - linux_compat.h, 19
- linux_compat.c, 18
 - clear_bit, 18
 - do_gettimeofday, 18
 - set_bit, 19
 - simulate_jiffies, 19
 - test_bit, 18
 - udelay, 19
- linux_compat.h, 19
 - _attribute_, 20
 - _cacheline_aligned, 20
 - _cli, 20
 - _init, 20
 - _kernel_size_t, 21
 - _s32, 21
 - _setup, 20
 - _sti, 20

__u16, 21
 __u32, 21
 __u8, 21
 access_ok, 20
 atomic_dec_and_test, 21
 atomic_inc, 20
 atomic_read, 21
 atomic_t, 21
 br_write_lock_bh, 20
 br_write_unlock_bh, 20
 BUG, 20
 capable, 20
 clear_bit, 21
 cli, 20
 copy_from_user, 20
 copy_to_user, 20
 cpu_to_le16, 19
 create_proc_entry, 20
 create_proc_read_entry, 20
 ETH_ALEN, 19
 EXPORT_SYMBOL, 20
 GFP_ATOMIC, 21
 htons, 20
 HZ, 20
 inline, 19
 inw, 19
 jiffies, 21
 KERN_CRIT, 20
 KERN_DEBUG, 20
 KERN_ERR, 20
 KERN_INFO, 20
 KERN_NOTICE, 20
 KERN_WARNING, 20
 kfree, 20
 KM_SKB_DATA_SOFTIRQ, 21
 kmalloc, 20
 le16_to_cpu, 19
 le16_to_cpus, 19
 local_irq_restore, 21
 local_irq_save, 21
 max_t, 20
 NR_CPUS, 20
 ntohs, 19
 out_of_line_bug, 20
 outw_p, 19
 printk, 20
 remove_proc_entry, 20
 rtnl_lock, 20
 rtnl_unlock, 20
 rwlock_t, 21
 s32, 21
 set_bit, 21
 simulate_jiffies, 22
 smp_processor_id, 20
 spin_lock_bh, 20
 spin_lock_init, 20
 spin_lock_irqsave, 20
 spin_unlock_bh, 20
 spin_unlock_irqrestore, 20
 spinlock_t, 21
 sti, 20
 test_and_clear_bit, 21
 test_and_set_bit, 21
 test_bit, 21
 u16, 21
 u32, 21
 u8, 21
 udelay, 22
 ulong, 21
 verify_area, 20
 lo_cong
 netdev.c, 24
 local_irq_restore
 linux_compat.h, 21
 local_irq_save
 linux_compat.h, 21
 lock
 orinoco_private, 3
 mac_info_rid_table
 tracehermes.c, 46
 main
 ori.c, 26
 show_hermes.c, 35
 trace_init.c, 44
 MAX_DEV_NAME_LENGTH
 ori.h, 30
 MAX_IRQLOOPS_PER_IRQ
 orinoco.c, 31
 MAX_IRQLOOPS_PER_JIFFY
 orinoco.c, 31
 MAX_MULTICAST

- orinoco.c, 31
- max_t
 - linux_compat.h, 20
- mc_count
 - orinoco_private, 4
- mod_cong
 - netdev.c, 24
- mod_timer
 - timer.c, 43
- modem_info_rid_table
 - tracehermes.c, 47
- mwo_robust
 - orinoco_private, 4
- ndev
 - orinoco_private, 3
- net_allocs
 - skbuff.c, 37
- net_fails
 - skbuff.c, 37
- net_free_locked
 - skbuff.c, 37
- net_init.c, 22
 - alloc_etherdev, 23
 - ether_setup, 22
 - EXPORT_SYMBOL, 22, 23
- net_locked
 - skbuff.c, 37
- net_skbcount
 - skbuff.c, 37
- netdev.c, 24
 - __cacheline_aligned, 24
 - dev_alloc, 25
 - dev_alloc_name, 25
 - lo_cong, 24
 - mod_cong, 24
 - netdev_max_backlog, 24
 - netdev_rx_stat, 24
 - netif_device_attach, 24
 - netif_device_detach, 24
 - netif_device_present, 24
 - netif_qnx_close, 24
 - netif_qnx_open, 24
 - netif_queue_stopped, 24
 - netif_running, 24
 - netif_rx, 25
- netif_start_queue, 24
- netif_stop_queue, 24
- netif_wake_queue, 24
- no_cong, 24
 - no_cong_thresh, 24
 - weight_p, 24
- netdev_max_backlog
 - netdev.c, 24
- netdev_rx_stat
 - netdev.c, 24
- netif_device_attach
 - netdev.c, 24
- netif_device_detach
 - netdev.c, 24
- netif_device_present
 - netdev.c, 24
- netif_qnx_close
 - netdev.c, 24
- netif_qnx_open
 - netdev.c, 24
- netif_queue_stopped
 - netdev.c, 24
- netif_running
 - netdev.c, 24
- netif_rx
 - netdev.c, 25
- netif_start_queue
 - netdev.c, 24
- netif_stop_queue
 - netdev.c, 24
- netif_wake_queue
 - netdev.c, 24
- nic_info_rid_table
 - tracehermes.c, 47
- nicbuf_size
 - orinoco_private, 3
- nick
 - orinoco_private, 4
- no_cong
 - netdev.c, 24
- no_cong_thresh
 - netdev.c, 24
- NR_CPUS
 - linux_compat.h, 20
- ntohs
 - linux_compat.h, 19

- NUM_CHANNELS
 - orinoco.c, 31
- NUM_DYNAMIC_RIDS
 - tracehermes.c, 45
- NUM_MAC_INFO_RIDS
 - tracehermes.c, 45
- NUM_MODEM_INFO_RIDS
 - tracehermes.c, 45
- NUM_NIC_INFO_RIDS
 - tracehermes.c, 45
- NUM_RIDS
 - orinoco.c, 31
- NUM_STATIC_RIDS
 - tracehermes.c, 45
- ori.c, 25
 - attach_orinoco_dev, 27
 - config_option_reset, 27
 - devno, 26
 - env, 26
 - handler, 26
 - install_orinoco_dev, 27
 - interrupt_id, 26
 - intr_proxy, 26
 - main, 26
 - process_ori_close, 27
 - process_ori_open, 27
 - process_wtrp_config, 28
 - process_wtrp_rx, 28
 - process_wtrp_tx, 28
 - wtrp_setup, 28
- ori.h, 28
 - MAX_DEV_NAME_LENGTH, 30
- orinoco.c, 30
 - alloc_orinocodev, 32
 - bitrate_table, 32
 - BITRATE_TABLE_SIZE, 31
 - channel_frequency, 33
 - DISPLAY_BYTES, 31
 - DISPLAY_STRING, 31
 - DISPLAY_WORDS, 31
 - DISPLAY_XSTRING, 31
 - DUMMY_FID, 31
 - encaps_hdr, 32
 - ENCAPS_OVERHEAD, 31
 - IRQ_BAP, 31
 - LARGE_KEY_SIZE, 31
 - MAX_IRQLOOPS_PER_IRQ, 31
 - MAX_IRQLOOPS_PER_JIFFY, 31
 - MAX_MULTICAST, 31
 - NUM_CHANNELS, 31
 - NUM_RIDS, 31
 - orinoco_interrupt, 32
 - ORINOCO_MAX_MTU, 31
 - ORINOCO_MIN_MTU, 31
 - orinoco_proc_dev_cleanup, 32
 - orinoco_proc_dev_init, 32
 - orinoco_reset, 32
 - orinoco_shutdown, 32
 - orinoco_standalone_ioctls, 32
 - orinoco_standalone_multicast, 32
 - packed, 32
 - PROC_BUFFER_SIZE, 31
 - PROC_LTV_SIZE, 31
 - PROC_REC, 31
 - PROC_SAFE_SIZE, 31
 - record_table, 32
 - SIOCIWFIRSTPRIV, 31
 - SMALL_KEY_SIZE, 31
 - SPY_NUMBER, 31
 - SYMBOL_MAX_VER_LEN, 31
 - TX_NICBUF_SIZE_BUG, 31
 - USER_BAP, 31
- orinoco.h, 33
 - alloc_orinocodev, 34
 - DEBUG, 34
 - FIRMWARE_TYPE_AGERE, 34
 - FIRMWARE_TYPE_INTERSIL, 34
 - FIRMWARE_TYPE_SYMBOL, 34
 - orinoco_instances, 34
 - orinoco_interrupt, 34
 - orinoco_key_t, 34
 - orinoco_keys_t, 34
 - ORINOCO_MAX_KEY_SIZE, 34
 - ORINOCO_MAX_KEYS, 34
 - orinoco_proc_dev_cleanup, 34
 - orinoco_proc_dev_init, 34

- orinoco_reset, 34
 - orinoco_shutdown, 34
 - orinoco_standalone_ioctl, 34
 - ORINOCO_STATE_DOIRQ, 34
 - ORINOCO_STATE_INIRQ, 34
 - RUP_EVEN, 34
 - TRACE_ENTER, 34
 - TRACE_EXIT, 34
 - WIRELESS_SPY, 34
- orinoco_instances
 - orinoco.h, 34
- orinoco_interrupt
 - orinoco.c, 32
 - orinoco.h, 34
- orinoco_key_t
 - orinoco.h, 34
- orinoco_keys_t
 - orinoco.h, 34
- ORINOCO_MAX_KEY_SIZE
 - orinoco.h, 34
- ORINOCO_MAX_KEYS
 - orinoco.h, 34
- ORINOCO_MAX_MTU
 - orinoco.c, 31
- ORINOCO_MIN_MTU
 - orinoco.c, 31
- orinoco_private, 3
 - allow_ibss, 4
 - ap_density, 4
 - bitratemode, 4
 - broken_cor_reset, 3
 - card, 3
 - channel, 4
 - channel_mask, 3
 - desired_essid, 4
 - dir_dev, 4
 - firmware_type, 3
 - frag_thresh, 4
 - hard_reset, 3
 - has_big_wep, 3
 - has_ibss, 3
 - has_ibss_any, 3
 - has_mwo, 3
 - has_pm, 3
 - has_port3, 3
 - has_preamble, 3
 - has_sensitivity, 3
 - has_wep, 3
 - host_string, 4
 - hw, 3
 - ibss_port, 3
 - iw_mode, 3
 - keys, 4
 - lock, 3
 - mc_count, 4
 - mwo_robust, 4
 - ndev, 3
 - nicbuf_size, 3
 - nick, 4
 - pm_mcast, 4
 - pm_on, 4
 - pm_period, 4
 - pm_timeout, 4
 - port_type, 4
 - preamble, 4
 - prefer_port3, 3
 - promiscuous, 4
 - rts_thresh, 4
 - rx_proxy, 4
 - rx_queue, 5
 - spy_address, 4
 - spy_number, 4
 - spy_stat, 4
 - state, 3
 - stats, 3
 - tx_key, 3
 - tx_proxy, 5
 - txfid, 3
 - wep_on, 3
 - wep_restrict, 3
 - wstats, 3
- orinoco_proc_dev_cleanup
 - orinoco.c, 32
 - orinoco.h, 34
- orinoco_proc_dev_init
 - orinoco.c, 32
 - orinoco.h, 34
- orinoco_reset
 - orinoco.c, 32
 - orinoco.h, 34
- orinoco_shutdown
 - orinoco.c, 32

- orinoco.h, 34
- orinoco_standalone_ioctl
 - orinoco.c, 32
 - orinoco.h, 34
- orinoco_standalone_multicast
 - orinoco.c, 32
- ORINOCO_STATE_DOIRQ
 - orinoco.h, 34
- ORINOCO_STATE_INIRQ
 - orinoco.h, 34
- out_of_line_bug
 - linux_compat.h, 20
- outw_p
 - linux_compat.h, 19
- packed
 - hermes.h, 11
 - hermes_rid.h, 16
 - ieee802_11.h, 17
 - orinoco.c, 32
- pm_mcast
 - orinoco_private, 4
- pm_on
 - orinoco_private, 4
- pm_period
 - orinoco_private, 4
- pm_timeout
 - orinoco_private, 4
- port_type
 - orinoco_private, 4
- preamble
 - orinoco_private, 4
- prefer_port3
 - orinoco_private, 3
- printk
 - linux_compat.h, 20
- PROC_BUFFER_SIZE
 - orinoco.c, 31
- PROC_LTV_SIZE
 - orinoco.c, 31
- PROC_REC
 - orinoco.c, 31
- PROC_SAFE_SIZE
 - orinoco.c, 31
- process_ori_close
 - ori.c, 27
- process_ori_open
 - ori.c, 27
- process_wtrp_config
 - ori.c, 28
- process_wtrp_rx
 - ori.c, 28
- process_wtrp_tx
 - ori.c, 28
- promiscuous
 - orinoco_private, 4
- record_info_t
 - tracehermes.c, 45
- record_table
 - orinoco.c, 32
- remove_proc_entry
 - linux_compat.h, 20
- rtnl_lock
 - linux_compat.h, 20
- rtnl_unlock
 - linux_compat.h, 20
- rts_thresh
 - orinoco_private, 4
- RUP_EVEN
 - orinoco.h, 34
- rwlock_t
 - linux_compat.h, 21
- rx_proxy
 - orinoco_private, 4
- rx_queue
 - orinoco_private, 5
- s32
 - linux_compat.h, 21
- set_bit
 - linux_compat.c, 19
 - linux_compat.h, 21
- show_hermes.c, 35
 - DREG, 35
 - main, 35
 - USER_BAP, 35
- show_net_buffers
 - skbuff.c, 36
- simulate_jiffies
 - linux_compat.c, 19
 - linux_compat.h, 22

- SIOCIWFIRSTPRIV
 - orinoco.c, 31
- skb_append
 - skbuff.c, 39
- skb_clone
 - skbuff.c, 37
- skb_cloned
 - skbuff.c, 39
- skb_copy
 - skbuff.c, 39
- skb_dequeue
 - skbuff.c, 39
- skb_dequeue_tail
 - skbuff.c, 39
- skb_get
 - skbuff.c, 40
- skb_headlen
 - skbuff.c, 36
- skb_headroom
 - skbuff.c, 40
- skb_insert
 - skbuff.c, 40
- skb_is_nonlinear
 - skbuff.c, 36
- skb_orphan
 - skbuff.c, 40
- skb_peek
 - skbuff.c, 40
- skb_peek_tail
 - skbuff.c, 40
- skb_pull
 - skbuff.c, 41
- skb_push
 - skbuff.c, 41
- skb_put
 - skbuff.c, 41
- skb_queue_empty
 - skbuff.c, 41
- skb_queue_head
 - skbuff.c, 41
- skb_queue_head_init
 - skbuff.c, 36
- skb_queue_len
 - skbuff.c, 41
- skb_queue_purge
 - skbuff.c, 42
- skb_queue_tail
 - skbuff.c, 42
- skb_reserve
 - skbuff.c, 42
- skb_shared
 - skbuff.c, 42
- skb_tailroom
 - skbuff.c, 42
- skb_trim
 - skbuff.c, 42
- skb_unlink
 - skbuff.c, 42
- skbuff.c, 35
 - _dev_alloc_skb, 37
 - _kfree_skb, 36
 - _kfree_skbmem, 37
 - _skb_append, 36
 - _skb_dequeue, 37
 - _skb_dequeue_tail, 38
 - _skb_insert, 36
 - _skb_pull, 36
 - _skb_push, 36
 - _skb_put, 36
 - _skb_queue_head, 38
 - _skb_queue_purge, 38
 - _skb_queue_tail, 38
 - _skb_trim, 36
 - _skb_unlink, 36
- alloc_skb, 38
- dev_alloc_skb, 39
- ip_frag_mem, 37
- kfree_skb, 39
- kfree_skbmem, 36
- net_allocs, 37
- net_fails, 37
- net_free_locked, 37
- net_locked, 37
- net_skbcount, 37
- show_net_buffers, 36
- skb_append, 39
- skb_clone, 37
- skb_cloned, 39
- skb_copy, 39
- skb_dequeue, 39
- skb_dequeue_tail, 39
- skb_get, 40

- skb_headlen, 36
- skb_headroom, 40
- skb_insert, 40
- skb_is_nonlinear, 36
- skb_orphan, 40
- skb_peek, 40
- skb_peek_tail, 40
- skb_pull, 41
- skb_push, 41
- skb_put, 41
- skb_queue_empty, 41
- skb_queue_head, 41
- skb_queue_head_init, 36
- skb_queue_len, 41
- skb_queue_purge, 42
- skb_queue_tail, 42
- skb_reserve, 42
- skb_shared, 42
- skb_tailroom, 42
- skb_trim, 42
- skb_unlink, 42
- SMALL_KEY_SIZE
 - orinoco.c, 31
- smp_processor_id
 - linux_compat.h, 20
- spin_lock_bh
 - linux_compat.h, 20
- spin_lock_init
 - linux_compat.h, 20
- spin_lock_irqsave
 - linux_compat.h, 20
- spin_unlock_bh
 - linux_compat.h, 20
- spin_unlock_irqrestore
 - linux_compat.h, 20
- spinlock_t
 - linux_compat.h, 21
- spy_address
 - orinoco_private, 4
- SPY_NUMBER
 - orinoco.c, 31
- spy_number
 - orinoco_private, 4
- spy_stat
 - orinoco_private, 4
- state
 - orinoco_private, 3
- static_cfg_rid_table
 - tracehermes.c, 46
- stats
 - orinoco_private, 3
- sti
 - linux_compat.h, 20
- SYMBOL_MAX_VER_LEN
 - orinoco.c, 31
- test_and_clear_bit
 - linux_compat.h, 21
- test_and_set_bit
 - linux_compat.h, 21
- test_bit
 - linux_compat.c, 18
 - linux_compat.h, 21
- timer.c, 43
 - add_timer, 43
 - del_timer, 43
 - init_timer, 43
 - mod_timer, 43
- TRACE_ENTER
 - orinoco.h, 34
- TRACE_EXIT
 - orinoco.h, 34
- trace_init.c, 43
 - DREG, 44
 - main, 44
 - USER_BAP, 44
- tracehermes.c, 44
 - dynamic_cfg_rid_table, 46
 - EXPORT_SYMBOL, 45
 - hermes_trace_dynamic_cfg, 45
 - hermes_trace_mac_info, 45
 - HERMES_TRACE_MAX_-
FIELD_SIZE, 45
 - hermes_trace_modem_info, 45
 - hermes_trace_nic_info, 45
 - hermes_trace_show_rid_table, 45
 - hermes_trace_static_cfg, 45
 - mac_info_rid_table, 46
 - modem_info_rid_table, 47
 - nic_info_rid_table, 47
 - NUM_DYNAMIC_RIDS, 45
 - NUM_MAC_INFO_RIDS, 45

- NUM_MODEM_INFO_RIDS, [45](#)
- NUM_NIC_INFO_RIDS, [45](#)
- NUM_STATIC_RIDS, [45](#)
- record_info_t, [45](#)
- static_cfg_rid_table, [46](#)
- tx_key
 - orinoco_private, [3](#)
- TX_NICBUF_SIZE_BUG
 - orinoco.c, [31](#)
- tx_proxy
 - orinoco_private, [5](#)
- txfid
 - orinoco_private, [3](#)
- u16
 - linux_compat.h, [21](#)
- u32
 - linux_compat.h, [21](#)
- u8
 - linux_compat.h, [21](#)
- udelay
 - linux_compat.c, [19](#)
 - linux_compat.h, [22](#)
- ulong
 - linux_compat.h, [21](#)
- USER_BAP
 - orinoco.c, [31](#)
 - show_hermes.c, [35](#)
 - trace_init.c, [44](#)
- verify_area
 - linux_compat.h, [20](#)
- weight_p
 - netdev.c, [24](#)
- wep_on
 - orinoco_private, [3](#)
- wep_restrict
 - orinoco_private, [3](#)
- WIRELESS_SPY
 - orinoco.h, [34](#)
- wstats
 - orinoco_private, [3](#)
- wtrp_setup
 - ori.c, [28](#)