Title
Design Of Fault Tolerant Control Systems For Ahs

Authors
Sastry, S.
Horowitz, R.
Hedrick, K.

Publication Date
1998

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

Design of Fault Tolerant Control Systems for AHS

S. Sastry, R. Horowitz, K. Hedrick
University of California, Berkeley

California PATH Research Report
UCB-ITS-PRR-98-16

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.

Report for MOU 288

April 1998

ISSN 1055-1425

Design of Fault Tolerant Control Systems for AHS: Annual Report for MOU 288 [1]

Principal Investigators: Prof. S. Sastry, Prof. R. Horowitz, Prof. K. Hedrick

[1]Research supported by CALTRANS MOU 288

[2]Please direct all correspondence concerning this manuscript to Prof. S. Sastry at EECS Dept., 261M Cory Hall, University of California at Berkeley, Berkeley, Ph: (510) 642 3245, e-mail: sastry@eecs.berkeley.edu.

Abstract

This report describes the research work conducted under CALTRANS MOU 288 on fault detection and handling for automated vehicles. The report describes fault detection and handling designs used in the longitudinal control system of platooned automated vehicles. Results are presented on experimental testing of the designs. During the course of the year the project has extended the past work on fault handling to deal with several special fault classes. We have also designed a consistent interface between the fault detection and handling modules and implemented it in the SHIFT programming language for the specification of hybrid systems. The report also describes part of the modeling formalism required for the formal verification of safety claims.

Keywords: fault management, automated vehicles

Contents

List of Figures

Chapter 1

Executive Summary

This document is an annual report describing the research and development work conducted under Caltrans MOU 288 on the design of fault tolerant control systems for AHS during the period July 1996 to June 1997.

The project is proceeding on schedule and has provided the deliverables as specified in the proposal. The items due are as follows.

1. A working paper on proposal task # im1.3, *Designing a consistent interface between fault detection and fault handling modules*. This working paper is chapter 2 of this report.

2. Software for proposal task # md2.1, *Extending the monitoring scheme*. The required software has been developed by two subgroups. For software on diagnostics see chapter 3 and for that on handling see chapter 2.

In addition to the required deliverables, we have included two chapters describing preliminary results on proposal task # md1.2, *Development of algorithms for special fault classes*. Chapter 3 describes the extensions required to design a complete diagnostic scheme for the longitudinal control system of a platooned vehicle. Chapter 4 describes the inter-vehicle coordination protocol required to diagnose radar range-rate measurement faults in a platoon follower vehicle. Chapter 5 describes preliminary work on proposal task # vpa1, *Formal Verification of Safety Claims*. The following paragraps provide a brief summary of the material in each chapter.

Chapter 2 (*An Interface Between Fault Handling and Detection Modules*) describes a fault handling scheme for the hierarchical PATH architecture. The scheme extends the fault tolerant AHS design framework developed under Caltrans MOU 135. This design proposed two main structures: capability and performance. The first structure, that is the focus of this chapter, deals with *hard* faults that are due to sudden changes in a sensor or actuator performance. The occurrence of hard faults can be detected by appropriate filters, such as those developed by Speyer etal. under MOU 126 and by Hedrick etal. under MOU 101. An efficient representation of the capability structure in SHIFT has been selected after comparing several different approaches. This is also gives us the software corresponding to our design. In the approach selected, faults are coded by a hierarchical structure of sets. The definition of such sets is easy to code in SHIFT and provides a flexible and extensible architecture modelled in a hybrid systems formalism. Since we model in a hybrid systems formalism, the capability structure can be represented by a set of finite states machines. This allows logical verification with the use of verification tools such as COSPAN. In the second part of the

report, a coordination supervisor is designed and tested. This supervisor deals with both the normal and degraded modes of operation of the AHS.

Chapter 3 (*A Complete Fault Diagnostic System for the Longitudinal Control of Automated Vehicles*), describes a complete system for fault diagnostics of the longitudinal controllers in a platoon of automated vehicles is developed. This extends past work to incorporate certain special fault classes (task # md1.2). The diagnostic system is designed to provide automated monitoring and fault identification of all the sensors and actuators used in the longitudinal control system, including radar sensors and inter-vehicle communication. The system uses several reduced- order nonlinear observers constructed from a longitudinal dynamic model of the vehicle. Multiple estimates of signals are obtained by designing each observer to utilize a different sensor measurement. Different combinations of all the available sensor signals and the observer estimates are then processed to construct a bank of 10 different residues. We show analytically that a fault in any of the sensors or actuators creates a unique subset of these residues to grow so as to enable exact identification of the faulty component. Both simulation and experimental results are described to demonstrate the working of the fault diagnostic system in the presence of various faults.

Chapter 4 (*A Diagnostic Protocol for Radar Faults*), describes a inter-vehicle coordination protocol for the diagnosis of failures in the radar range-rate measurement of a platoon follower vehicle. This extends past work to incorporate certain special fault classes (task # md1.2). The radar range-rate measurement is checked by differencing the inertial speed of the car and the car in front. Therefore this difference can turn high if there is a fault in the inertial speed measurement sensors of the car in front. An inter-vehicle coordination protocol is required to eliminate the possibility of a fault in the speed measurement of the car in front before declaring a fault in the radar range-rate measurement.

Chapter 5 (*A Methodology for the Integration of Vehicle Failure Diagnostics*), describes a part of the modeling formalism required for the formal verification of safety claims (proposal task # vpa1). It presents an architecture for the formal integration of diagnostics and degraded mode control at the regulation and coordination layers and the interfaces between them. It also describes a modeling formalism for the synthesis and verification of diagnostic logic and presents some examples that apply the formalism to the verification of diagnostic logic used in the longitudinal control system.

Chapter 2

An Interface Between Fault Handling and Detection Modules

2.1 Introduction

A scheme for the interface between fault detection and fault handling modules in the hierarchical PATH architecture is proposed. The scheme is developed after the fault tolerant AHS design proposed by Lygeros et al in [13] that proposes two main structures in a fault tolerant AHS: capability and performance structures. The capability structure, that is the main focus of this chapter, deals with *hard* faults, those that are due to sudden changes in a sensor or actuator performance. The occurrence of hard faults can be detected by appropriate filters, like those proposed by Speyer's or Hedrick's research groups [3, 4]. An efficient representation of the capability structure in SHIFT have been selected after comparing several different approaches. In the selected approach, system capabilities are decided after the analysis of a hierarchical structure of sets. The input to the lower level in this hierarchy matches the output of the fault detection module. There are two levels of outputs, directed to the regulation and coordination layers of the PATH AHS architecture, respectively. By using these outputs, the system decides on the availability of regulation layer control laws or coordination layer maneuvers. The definition of the hierarchical sets for fault handling is easy to code in SHIFT and provides a flexible architecture, that will accommodate for changes, for example, in the sets of sensor or actuators. Moreover, using the hybrid systems formalism, that forces to model the capability structure as a set of finite states machines, also allows for the logical verification of the design with the use of some other automatic verification software, such as COSPAN [8]. The second part of the chapter presents a coordination layer maneuver supervisor, whose design is base in the same hierarchical structure of sets. This supervisor decides which maneuver a vehicle is able to perform based on its current capabilities, for both the normal and degraded modes of operation of the AHS.

2.2 AHS Fault Tolerant Structure

In this section we follow the ideas presented in [13] for the design of a hierarchical fault tolerant AHS. Lygeros et al [13] divide the problem of supervising the AHS operation into four major structures: sensor, capability, performance and control. The information flow between these structures is depicted in Figure 2.1 The sensor structure encodes all the information that is sensed at the individual vehicles level or at the roadside infrastructure level. The capability structure is designed to determine

discrete changes in the system capability due to faults in the vehicle and roadside hardware. The performance structure objective is to decide on any gradual degradation in system performance due to adverse environmental conditions and gradual wear of AHS components. The control structure, finally, decides on the control actions on the AHS based on the information encoded by the three other structures. In this section we focus on the design of the capability and performance structures.

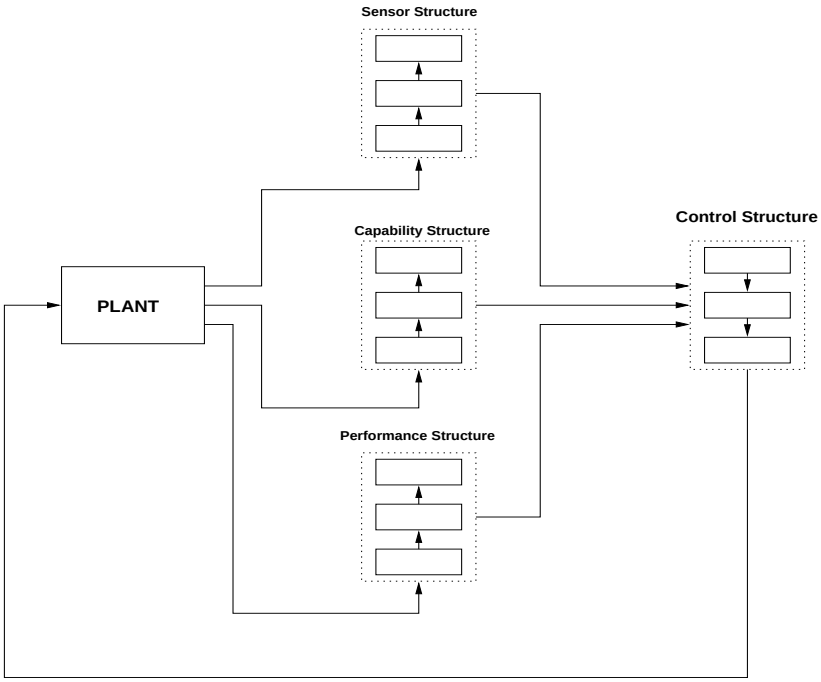


Figure 2.1: Overview of fault tolerant control structure

The design proposed in [13] for the capability structure is in the form of a hierarchy of binary logic predicates, while the performance structure is realized through a set of maps from the causes of gradual performance degradation to the parameters that reflect the performance of the system. These maps can be realized on line. Figure 2.2 shows more details about the capability structure and the performance structure suggested in [13].

The capability structure in Figure 2.2 assumes the existence of a set of signals coming from a fault detection structure already in the appropriate format. The maps for the capability structure in Figure 2.2 are:

F_R - maps capabilities of the physical layer to the regulation layer.
F_I - maps, in the regulation layer, capabilities from its regulator to its supervisor.
F_C - maps capabilities of regulation layer, communication and neighbor vehicles to the coordination layer supervisor.

The maps for the performance structure in Figure 2.2 are:

$f: \quad \mathcal{C} \longrightarrow \mathcal{P}.$
$R_i: \quad \mathcal{P} \longrightarrow \{0, 1\}; \qquad i = 1, \ldots, r.$
$\mathcal{C}$ – causes of degradation and performance parameters.

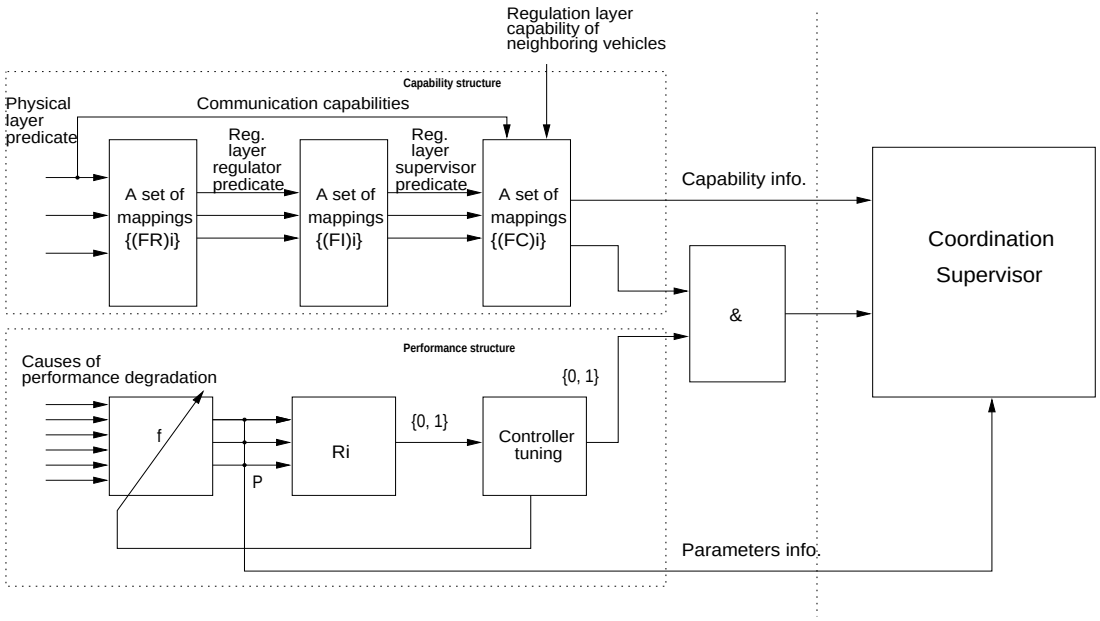


Figure 2.2: Capability and performance structure

$\mathcal{P}$ – set of performance parameters
$\mathcal{P} = P_P \cup P_S \cup P_R \cup P_C \cup P_L \cup P_N$.

where the sub-indexes P, S, R, C, L and N stand for parameters related to the physical layer, sensor structure, regulation, coordination link and network layers, respectively.

2.3 Implementation of the capability structure in SHIFT

In this section we discuss the implementation in SHIFT of the capability structure. We show an example of this implementation for the normal mode of operation of the PATH AHS architecture. This example uses a combination of C and SHIFT codes. We include some some plots that show simulation results. We present a generic design for implementation that is only based on finite states machines coded in SHIFT. Plots illustrating simulation results produced with this code are also included.

2.3.1 An example of the capability structure

We present an example, taken from [13], to illustrate the functioning of the capability structure under the normal mode of operation. The control scheme for normal operating conditions relies on a number of resources: sensors, actuators and communication devices, both on vehicles and on the roadside. From an input-output point of view, the goal of the capability structure is to take information from these resources and to determine the influence of failures in any of them on the ability of a vehicle to perform a given maneuver. To achieve this goal [13] proposed a design based on a hierarchy of logical binary predicates. Figure 2.3 illustrate a logic structure for fault handling in the normal mode. In Figure 2.3 each predicate will monitor a single functional capability and

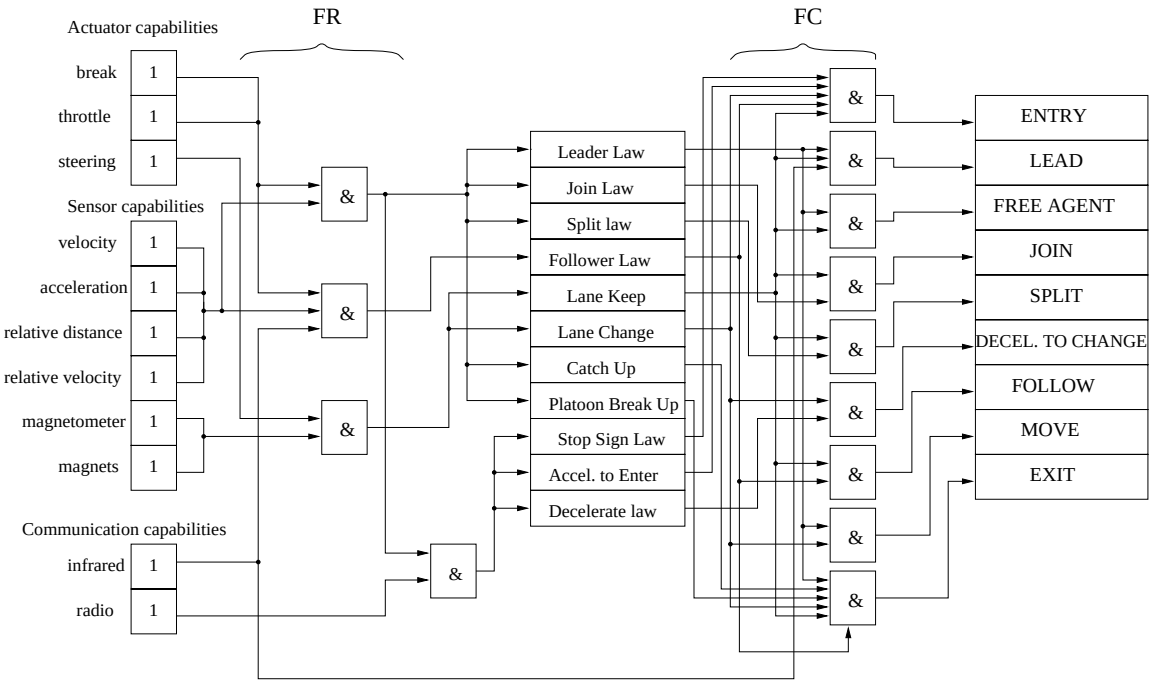


Figure 2.3: Logic structure of fault handling for normal mode AHS

will return a "1" (True) if the system possesses a correct capability or a "0" (False) otherwise[1]. The values returned by the predicates higher in the hierarchy depend, naturally, on the values of predicates at lower levels of the hierarchy. The maps F_R and F_I can be denoted as:

$$F_R: \quad \{0,\ 1\}^{n_{act}+n_{sen}+n_{comm}} \longrightarrow \{0,\ 1\}^{n_{long}+n_{lat}}$$
$$F_I: \quad \{0,\ 1\}^{n_{long}+n_{lat}} \longrightarrow \{0,\ 1\}^{n_{man}}$$

where n_{act}, n_{sen} and n_{comm} indicate the number of actuators, sensors and communication channels, respectively. n_{long} and n_{lat} the number of longitudinal and lateral control laws in the regulation layer, respectively. Finally, n_{man} denotes the number of maneuvers in the coordination layer.

An desired feature in the SHIFT implementation of any capability structure is that the implementation is easy to expand and/or change. This is important when other sensors, actuators, communication devices, or even control laws are added or modified in the structure.

Figure 2.4 illustrates the structure adopted to implement this normal mode capability structure in SHIFT. We define three SHIFT types:

type Fault: Generates faults for simulation purposes.

type Laws: Relates faults and control laws at the regulation layer level.

type Maneuvers: Operates on control laws at the regulation layer level to produce maneuvers capabilities at the coordination layer level.

[1]For this reason the logical predicates are processed by AND operators

Since there is no pointer-like data types in the current SHIFT release, we use `array` and `set` to realize a dynamic data structure. For example, we use following code

```
law_name_cap_arg  :=  [[act_cap[i], sen_cap [j], com_cap[k]]
                      : i in index_act_name,
                      j in index_sen_name,
                      k in index_com_name];
```

to code the availability of the `law_name` regulation layer control law. By changing `index_{act, sen, com}_name`, we can adapt for different combinations and sizes of actuators, sensors and communication channels sets, and by changing `law_name`, for different control laws.

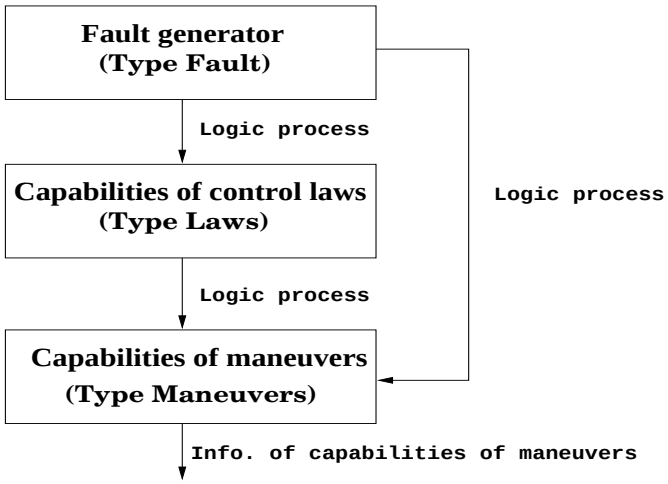


Figure 2.4: Structure of implementation for fault handling in SHIFT

SHIFT does not support functions, although a SHIFT program can refer to external C/C++ functions. In this example, logic and mathematical calculation are programmed using external C functions. The main advantage to code this calculations in C is currently in the debbuging phase. There are also some gains in the use of C libraries.

A SHIFT simulation results for the normal mode of operation capability structure example is illustrated by Figure 2.5. The first plot, (a) in Fig. 2.5, shows the state of the magnet sensor and the brake actuator. The second plot, (b) in Fig. 2.5, displays the capability of the leader law and change-lane law in the regulation layer. Finally, the third plot, (c) in Fig 2.5, indicates the capability of the Lead and Decel-to-change-lane maneuvers in the coordination layer. A magnet fault takes place at $t = 1.10\ s$, the corresponding sensor changes from "1" to "0". Because of the lack of lateral position sensing, the capability of change-lane law also changes from "1" to "0" at the same time. Notice that the capability of lead control law is still "1". However, since each maneuver consists at least one lateral and one longitudinal control law, the Lead and Decel-to-change-lane maneuvers are both unavailable, as is clearly indicated by the change from "1" to "0" in the third plot (c) in Fig 2.5. A second fault, now a brakes fault, takes place at $t = 1.10s$. The immediate effect of this fault is to make the lead control law unavailable. The brakes fault is then switched off, on and off, respectively at $t = 1.80\ s$, $t = 2.30\ s$ and $t = 2.80\ s$. The effect on the leader control law of the regulation layer is clearly following the capability state of the brakes actuator. The capability of both maneuvers, Lead and Decel-to-change-lane, remains in "0" while the fault on the magnets is present. It is only after

the fault in the magnet is finished, at $t = 3.10\ s$, that the Lead and Decel-to-change-lane maneuvers become again available.

A delay between the time faults happened and the capabilities changed can be noticed in Fig 2.5. This delay has exactly the size of the simulation time step.

2.3.2 A generic capability structure implementation

The example in the previous section is limited, since it is only considering the conditions for the normal mode of operation in AHS. In this section we adopt a more generic structure to implement the fault handling scheme in SHIFT. Figure 2.6 shows a block diagram of a more general architecture for the capability structure. We assume that the sets of sensors, actuators and communication channels are ordered and finite. Suppose, for example, that only some sensors are involved in determining the capability of a regulation layer control law. Ordering the sets will imply that we only need to know the position of those sensors in the ordered set of sensors. More specifically, if every sensor, actuator and communication channel is associated with a power of two corresponding to its order in the set, each combination of capabilities yields an unique number. Therefore, with this convention, in this approach all the information related to the appropriate sensors, actuators and communications involved in a given capability analysis is coded by a set of five numbers:

INDEX_ACT a multiple pointer to the suitable elements from the actuator capability array.

INDEX_SEN a multiple pointer to the suitable element from the sensor capability array.

INDEX_COMM a multiple pointer to the suitable elements from the communication devices capability array.

INDEX_OTHERS a multiple pointer to the suitable elements from other faults capability array, such as "Out of Gas, Low on Gas, etc.".

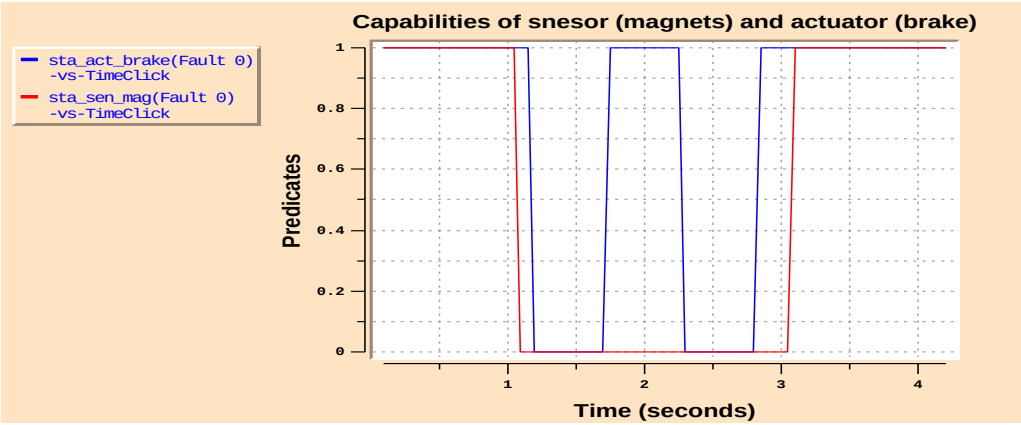
INDEX_SEC a multiple pointer to the suitable elements from the control laws capability array;

The two last pointers in the structure, **INDEX_OTHERS** and **INDEX_SEC** are added to provide more flexibility in the scheme and the possibility of intermediate calculations.

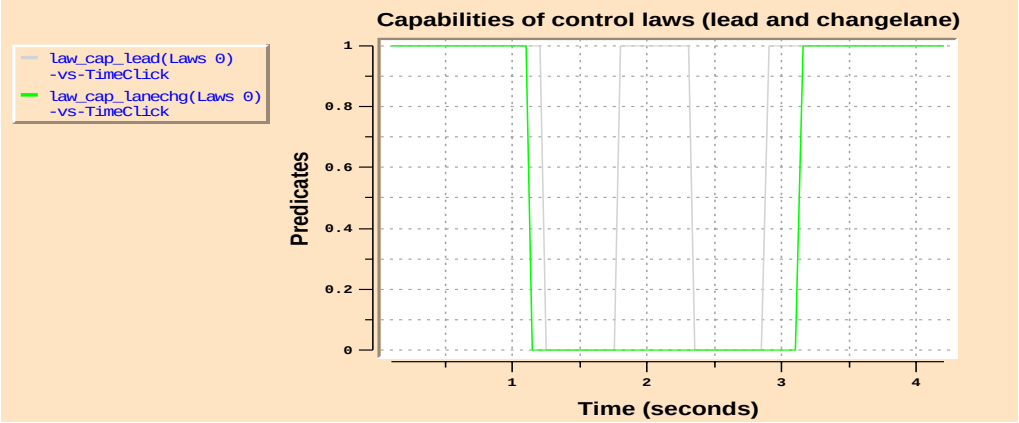
To explain the role of these multiple pointers, consider again the example in section 2.3.1. The set of actuators is composed by the brakes, throttle and steering actuators. For the capability of the lead control law, we require to check the availability of the break and throttle actuators, the first and second elements in the set of actuators, respectively. Therefore we code the use of these two actuators by making INDEX_ACT= 3.

In [13] the process for the capability structure is assumed to be logic. A general structure can consider other types of processes such as probabilistic, fuzzy logic, etc. The use of this kind of post-processing can occur in addition or complementing the filtering that is already applied in the fault detection module [3, 4]. It is not the intention on this section to further discuss this issue of post-processing. We only want propose an structure that will allow such implementation, if desirable. The kind of process that is used to determine a given capability can be coded by

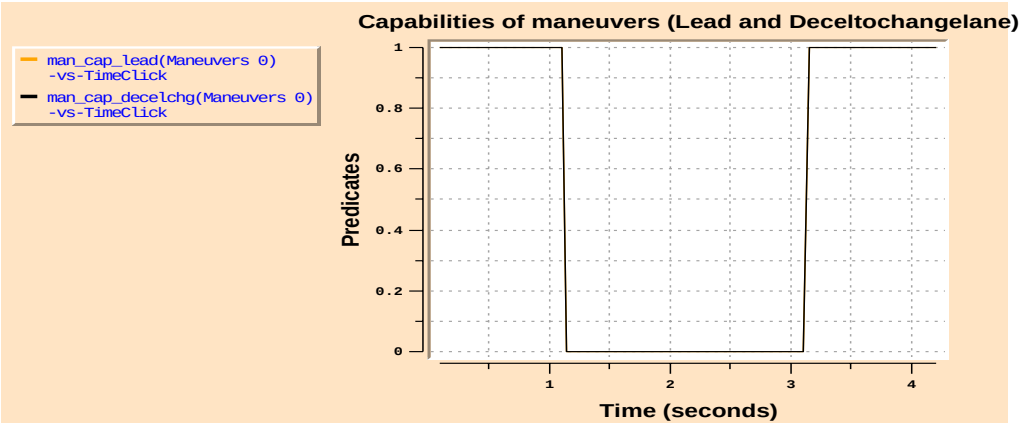
ID # a pointer to the suitable process for a particular capability.



(a) Capabilities for Brake and Magnets



(b) Capabilities for Lead and Change-lane control laws



(c) Capabilities for Lead and Decel-to-Change-Lane maneuvers

Figure 2.5: Simulation results of an example

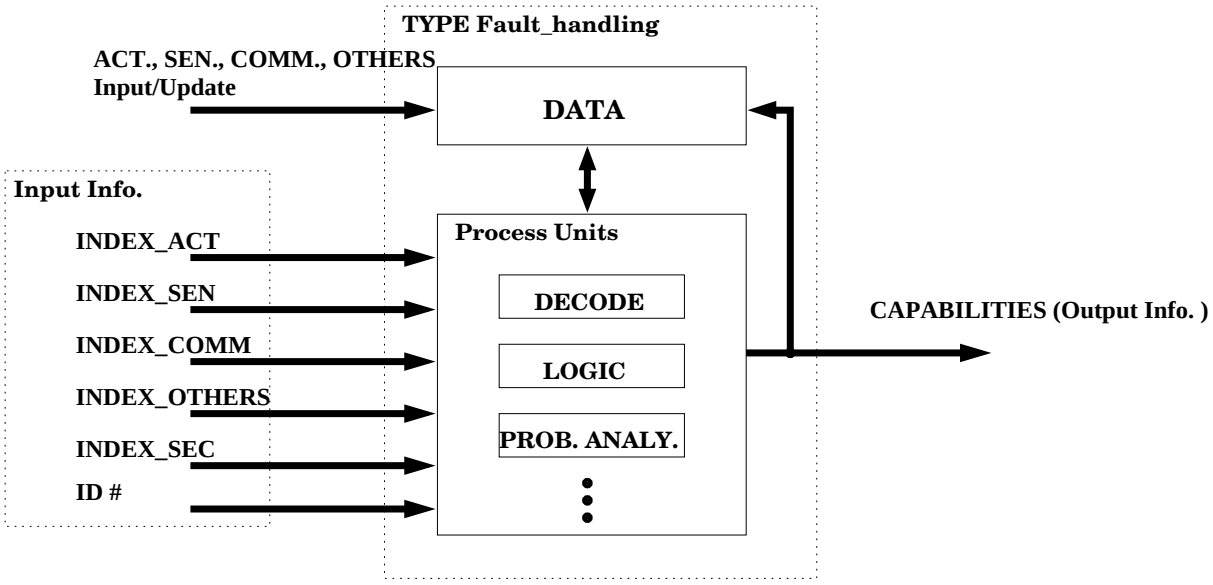


Figure 2.6: A proposed generic structure of fault handling in SHIFT

We can implement this generic structure in SHIFT by creating a class-like `type Fault_handling` with a data storage. By using the data connection, we can input or update the capabilities of actuators, sensors and communication devices. Secondary capabilities, as the control laws capabilities or any other **regulator predicate** in the regulation layer, can be updated by the output of the this `type Fault_handling`. By setting the values of the ID number, the suitable process is chosen. All logic and probability analysis can be realized by C functions.

For the example in the section 2.3.1, assume we want to calculate the lead control law and lead maneuver capabilities. The it is enough to code

```
Fault_handling  lead_law_cap := create(Fault_handling,3,15,0,0,0,0);
Fault_handling  lead_man_cap := create(Fault_handling,0,0,1,0,17,0);
```

where the arguments in the example correspond to sensors, actuator, communications, other inputs[2], regulation layer laws and type of process.

The other maneuver capability processes given in Figure 2.3 can be implemented in a similar fashion.

2.3.3 A capability structure based on hierarchical sets

In the previous sections, we presented two schemes to implement the capability structure that are based on a mixture of C and SHIFT codes. In this section we present a full implementation in SHIFT that also recovers the capability structure proposed in [13] and allows to take full advantage of the hybrid systems formalism under which SHIFT was developed. Although there are clearly some gains in the mixed coded implementations presented in the previous sections, we think that trying to preserve significant portions of the code in SHIFT will add to the robustness and flexibility of the design. In particular we will be able to exploit the verification capabilities that are currently under

[2]This set is empty for the example.

development for SHIFT as well as the automatic generation of real time code, that will be possible in future versions of this language.

This last approach is based on the use of the `Set` formalism of SHIFT. We define two types of sets, one for the capability of the regulation layer control laws and other to verify the capability of the coordination layer maneuvers. When declaring these sets, the user will code a particular instance of the capability structure. They are, however, the only piece of code that will be necessary to modify in the capability structure that we are defining. Two examples of instances of these sets are shown below

```
set(Fault_generate) laws_{law}_set := {act_brake, ..., com_radio};
set(Logic_laws) mans_{maneuver}_set := {lead, ..., decel};
```

Once these sets are defined, the capability structure is composed by a large number of elementary finite state machines (FSM). We designed three types of FSMs

- $Fault\_generators$.
- $Logic\_laws$.
- $Logic\_mans$.

Figure 2.7 show the three types of FSM.

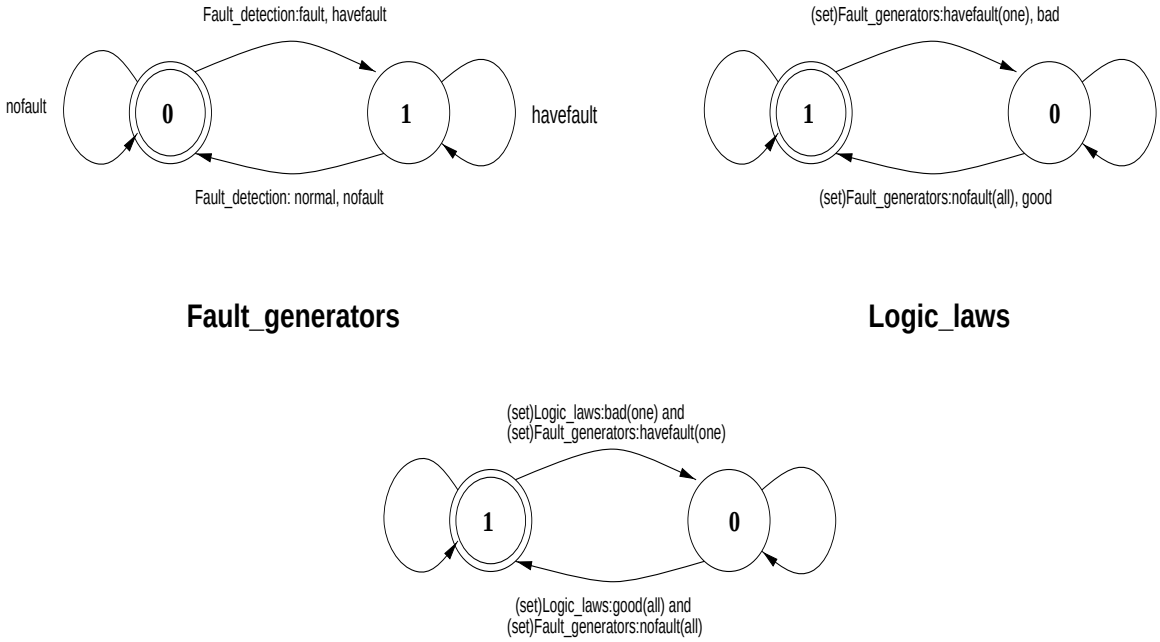


Figure 2.7: basic finite state machines for the capability structure

The $Fault\_generators$ FSMs are the point of connection with the fault detection module. These machines have two states: 0 and 1 that correspond to the $nofault$ and $have\_fault$ conditions,

respectively. Each FSM is associated with only one residue unit in the fault detection module. Initially, the *Fault_generator* FSM will be at the 0 state. Whenever the residue in the fault detection unit changes from *normal* to *fault*, this transition will trigger a transition in the *Fault_generator* to the state 1. It will remain there until the fault detection units goes to a *normal* condition. Notice that when in 0 the FSM will produce the *nofaul*t label and when in the 1 will produce a *have_fault* one.

Each one of the *Logic_law* FSM, the second type in Figure 2.7, analyzes only one set of *Fault_generator* FSMs. The task of this FSM is to decide in the possibility to perform a regulation layer control law. If **any** of the *Fault_generators* that are members of this set produces a label *havefault*, the *Logic_law* FSM will have a transition to the state 0 and will produce the label *bad* while in there. Only when **all** the *Fault_generators* are in *good* condition, will the *Logic_law* FSM have a transition to the 1 state.

The last type of FSM described in Figure 2.7 is the *Logic_man* FSM. Each one of these FSMs is associated with two sets. The first set corresponds to a set of *Faul_generators* FSMs and the second set to a set of *Logic_man* FSMs. The *Login_man* FSM is designed to determine whether a coordination layer maneuver is available. The conditions for the transitions in the *Logic_man* FSM are similar to those of the *Logic_law* FSM: if any element in the two sets produces a *bad* or *have_fault* label the FSM will go to the state 0, indicating the inability to perform the associated maneuver. The *Logic_man* FSM will remain in the 0 state until all the associated FSM produce a *good* or *nofault* label, in which case it will transition to the state 1.

It is very important to remark, that the capability structure can deal with **simultaneous faults**. This is stated with the use of the *one* and *all* options for set-related transitions in SHIFT.

The following is a SHIFT pseudo-code example of the implementation of the sets and FSMs described above.

```
set(Fault_generate) laws_{law}_set := {act_brake, ..., com_radio};
set(Logic_laws) mans_{maneuver}_set := {lead, ..., decel};

type Fault_generate
{
 output symbol indicator_fault;
.
.
.
 transition
     normal -> fault{}
      when ...
      do{
        indicator_fault := $havefault;
.
.
.
},

.
.
.

     fault -> normal{}
      when ...
      do{
        indicator_fault := $nofault;
.
.
```

```
.
},
}

type Logic_laws
{
 output symbol indicator_law;
.
.
.
 transition
     state_1 -> state_0 {}
      when exists Cp in laws_{laws}_set: indicator_fault(Cp) = $havefault
      do{
.
indicator_law := $bad;
.
},

     state_0 -> state_1 {}
      when not(exists Cp in mans_{maneuvers}_set: indicator_fault(Cp)
      = $havefault)
      do{
         .
indicator_law := $good;
.
};
}

type Logic_mans
{ .
.
.
 transition
     state_1 -> state_0 {}
      when exists Cp in mans_{maneuvers}_set: indicator_law(Cp) = $bad
      do{
.
.
.
},

     state_0 -> state_1 {}
      when not(exists Cp in mans_{maneuvers}_set: indicator_law(Cp) = $bad)
      do{
         .
.
.
};
}
```

To illustrate the implementation of the capability structure with hierarchical sets in SHIFT we include results from a simulation in TKSHIFT. Notice that we have to simulate the fault detection module. In particular, we use the following expression to change the value of the predicates P_i of *Actuators*, *Sensors* and *Communication devices*:

$$P_i = \begin{cases} 1 & (11k + 0.9i)s \leq t \leq (11k + 0.9i + 0.5)s \\ 0 & \text{otherwise} \end{cases}$$

$$i = 1, 2, \ldots, 11,\ k \in Z^+$$

Figure 2.8 shows the resultant pattern of induced faults.

Figures 2.9 and 2.10 show the results for the regulation layer control laws and coordination layer maneuvers. The control laws and maneuvers still correspond to the example in Figure 2.3. Notice that the requirements for the coordination layer maneuvers are much stringent than those of the regulation layer control laws. This is clearly indicated by the larger amount of 1 to 0 transitions that one can notice when comparing Figures 2.9 and 2.10.

2.4 Coordination Supervisor

The intention of the extended or degraded mode AHS control architecture is to maintain a safe operation while minimizing the impacts of abnormal conditions on the highway systems. In [13], the extended coordination layer controller is divided into two levels. The first level, called **coordination regulator**, controls the maneuvers protocols. The second level realizes strategic planning and is called **coordination supervisor**. When a fault is detected a specialized monitors classifies it into a unique fault class. Then, the coordination supervisor selects a strategy that depends on three factors:

1. The class/subclass of the fault.
2. The capabilities of the faulty vehicle.
3. The capabilities of the neighbor vehicles.

In [13], it is assumed that each degrade mode maneuver can be decomposed into a sequence of one or more disjoint **atomic maneuvers** in such a way that the strategy also consist of the choice of the **atomic maneuvers** to be executed by the **coordination regulator**. In this report, we denote as strategies the composed maneuvers that are commanded by the coordination supervisor.

2.4.1 Coordination regulator degraded mode maneuvers

The coordination regulator maneuvers includes the following degraded mode *atomic maneuvers*

Forced Split: Similar to the **normal mode** split maneuver. It is used whenever a faulty vehicles requires to split from a platoon to exit the AHS.

Emergency Lane Change: Similar to the **normal mode** lane change maneuver. It is used by a free agent or a platoon in process to exit the AHS.

Front Dock: It is initiated by a platoon leader that has lost he capability to execute the leader maneuver. The maneuver requires the vehicle in front of this leader to decelerate in order to perform a reverse join.

Aided Stop: It is initiated by a follower that has developed a brakes-off failure. The vehicle in front will assist the follower to brake.

Gentle Stop: It is used by a faulty vehicle that is ordered to stop and can do so by using its own brakes. The rate of braking is such that comfort levels are preserved during the execution of the maneuver.

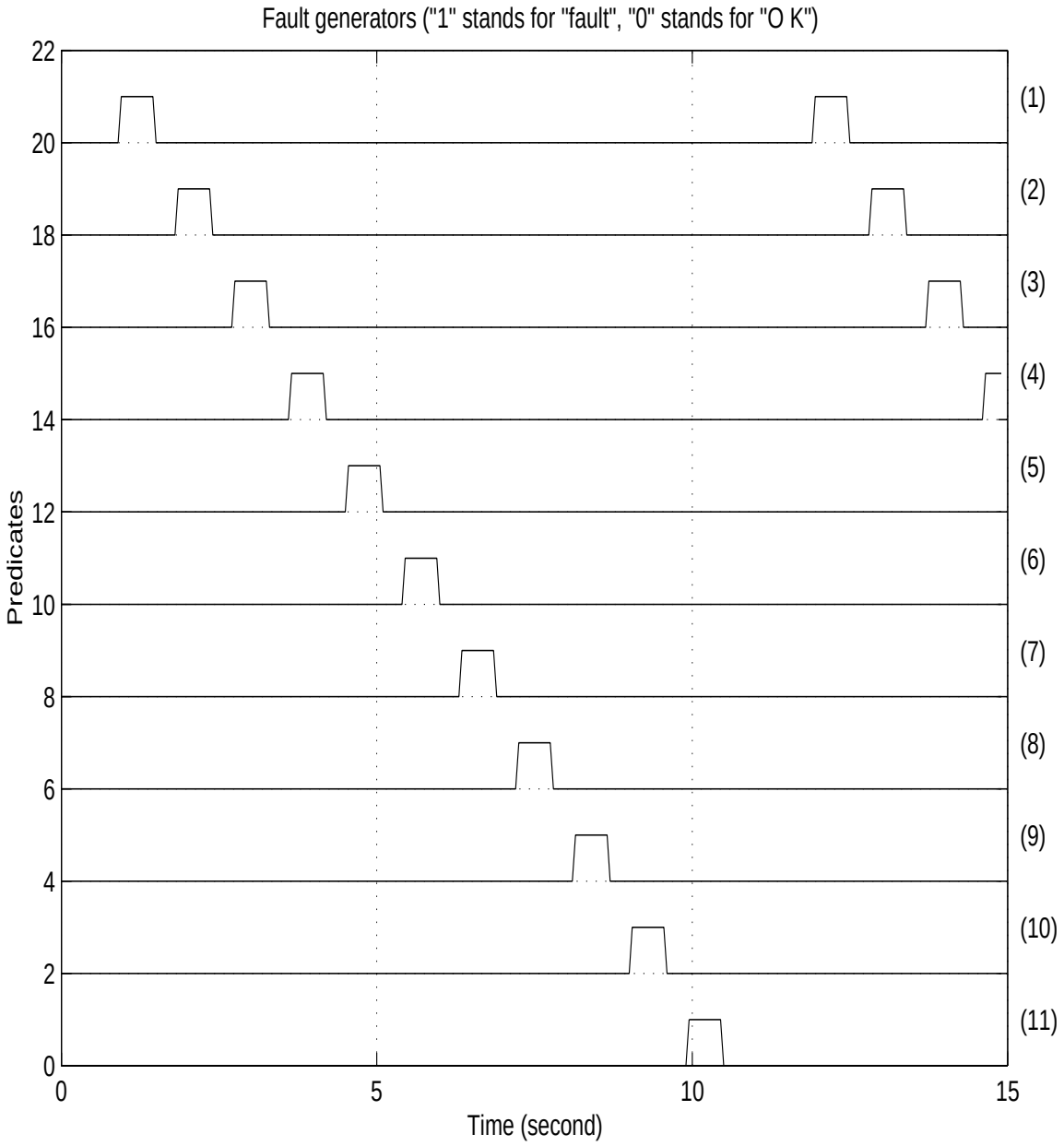


Figure 2.8: Capabilities for physical layer. (1) Brake (actuator) (2) Throttle (actuator) (3) Steering (actuator) (4) Velocity sensor (5) Acceleration sensor (6) Relative distance sensor (7) Relative velocity sensor (8) Magnetometers (9) Magnets sensor (10) Infrared (communication) (11) Radio (communication)

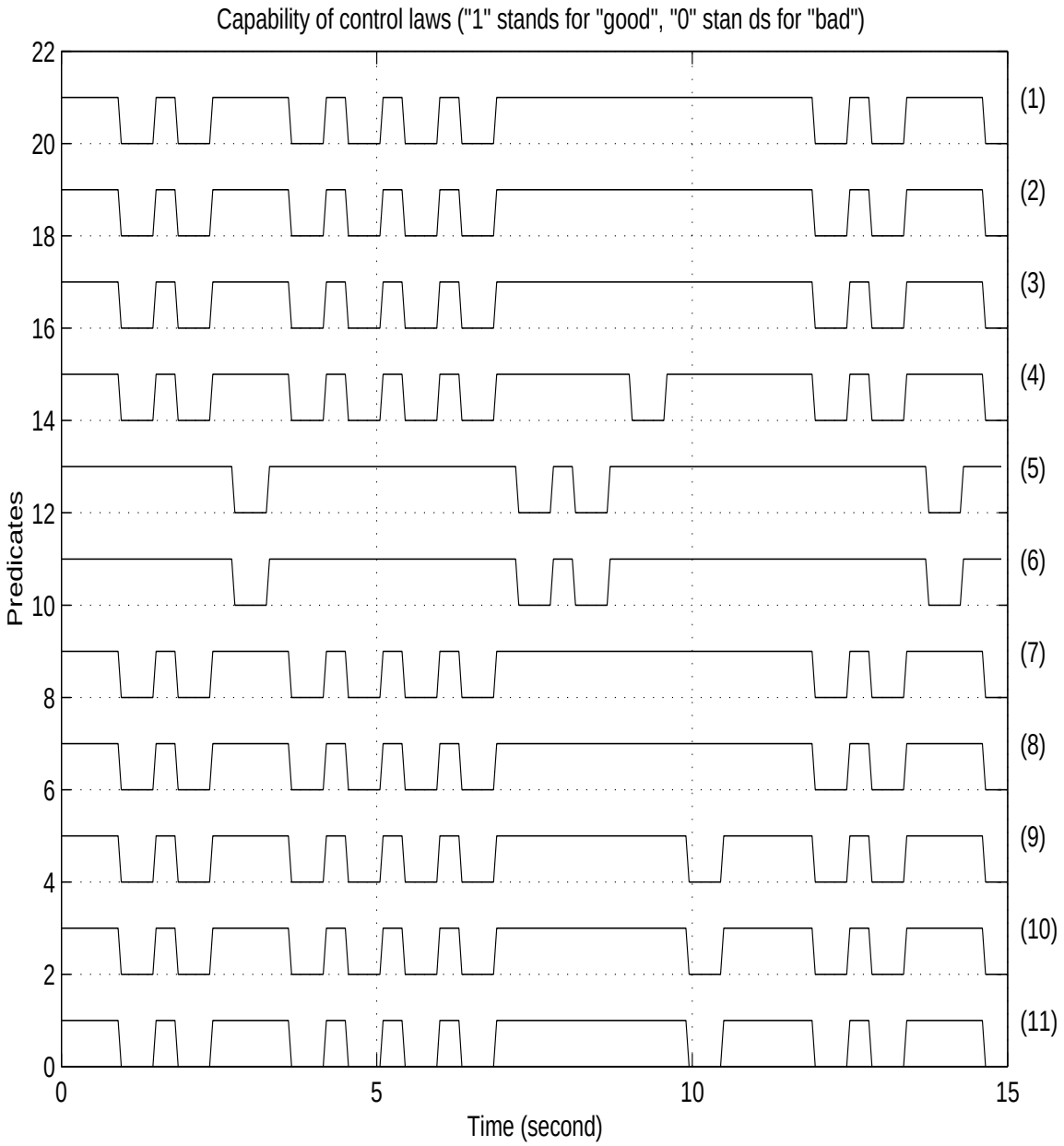


Figure 2.9: Capabilities for control laws (1) Lead law (2) Join law (3) Split law (4) Follow law (5) Lane keep law (6) Lane change law (7) Catch up law (8) Platoon break up law (9) Stop sign law (10) Acceleration to enter law (11) Deceleration law

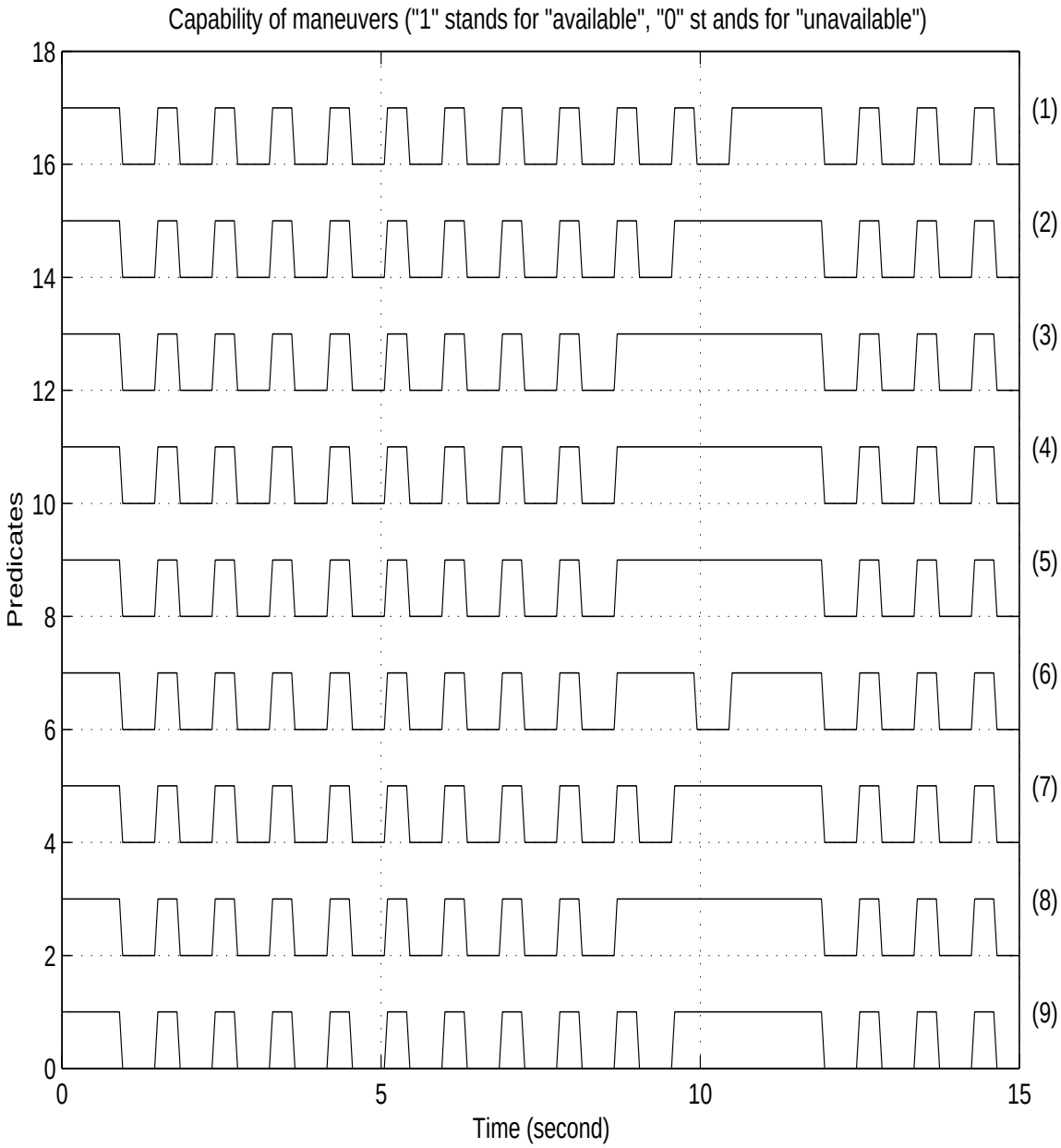


Figure 2.10: Capabilities for maneuvers (1) Entry maneuver (2) Lead maneuver (3) Free agency maneuver (4) Join maneuver (5) Split maneuver (6) Decelerate to change lane maneuver (7) Follow maneuver (8) Move maneuver (9) Exit maneuver

Crash Stop: It is the same as Gentle Stop, although in this case maximum emergency braking is applied.

The normal mode maneuvers: leader, follower, join, split and lane change [11] are also used for fault handling purposes.

2.4.2 Coordination Supervisor Strategies

The following set of strategies are implemented in the coordination supervisor. These strategies correspond to fault class/subclass described in [13] and are executed by using a predefined sequence of basic or atomic normal/degraded maneuvers described in the previous section, depending on the fault and capabilities of the faulty vehicle and its neighbors.

Gentle Stop (GS) The vehicle presents a serious fault and must stop. The vehicle can stop by itself and the fault is not so severe. Comfort braking can be used.

Crash Stop (CS) The faulty vehicle has a serious fault and must stop as soon as possible. The vehicle can stop by itself and does it by applying the maximum brake.

Aided Stop (AS) The fault is most severe, like brakes-off. The vehicle can not stop by itself. The vehicle must stop with the assistance of other vehicles.

Take Immediate Exit (TIE) A fault is detected that does not imply to stop the vehicle in the AHS. The vehicle has lost some capability, although it is still able to exit the AHS by itself. It must therefore exit the AHS as soon as possible.

Take Immediate Exit - Escorted (TIE-E) The fault in the vehicle requires its exit from the AHS. The vehicle has lost capability of exiting by itself. Other vehicles must assist the faulty vehicle to exit.

Take Immediate Exit - Normal (TIE-N) The conditions are almost same as TIE strategy. The severity of the fault is smaller and therefore the normal exit maneuver is used to exit AHS.

Normal The vehicle has no fault, it executes Normal mode maneuvers.

Fig. 2.11 illustrates the combination of the capability structure and the coordination supervisor, for the example in section 2.3.1.

2.4.3 Mutual Exclusion and Priority

In the degraded mode design every platoon is constrained to be engaged in only one maneuver at a time. To achieve a fast response to a degraded condition of operation, it is necessary to interrupt maneuvers whose execution delays a proper fault handling. Thus, a priority on the coordination supervisor strategies has to be imposed to allow higher priority strategies to preempt lower priority ones. The priority of a strategy depends on the severity of the faults, with higher priority assigned to more severe fault. Following [13], the priority assigned to the different strategies is:

$$AS \Rightarrow CS \Rightarrow GS \Rightarrow TIE-E \Rightarrow TIE \Rightarrow TIE-N \Rightarrow Normal$$

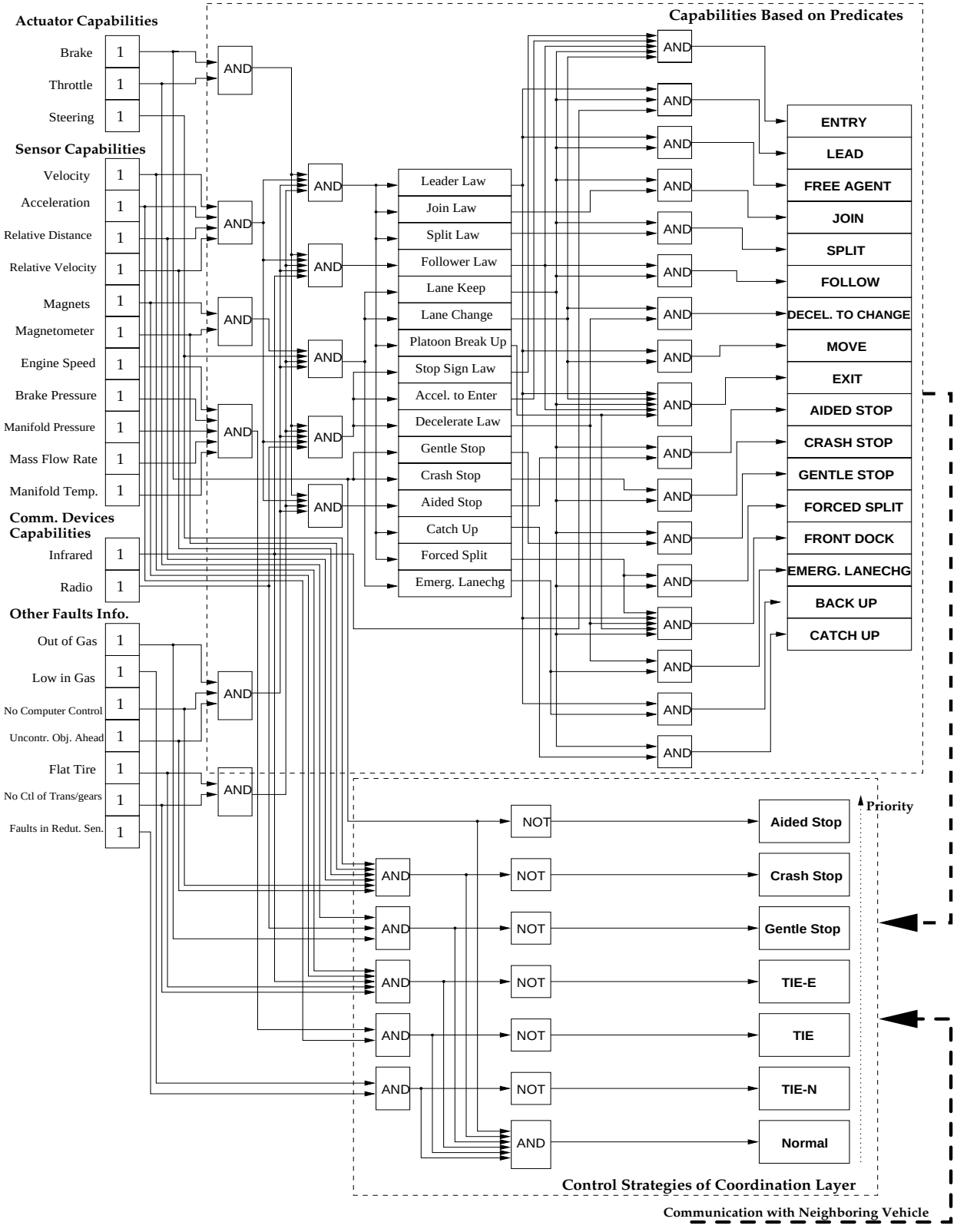


Figure 2.11: Structure of fault handling and coordination supervisor

A lower priority strategy request will not be acknowledged while the vehicle is engaged in a higher priority strategy. However, a higher priority maneuver request can interrupt the current executing lower priority maneuvers.

2.4.4 Implementation

For implementation the coordination supervisor we propose a structure as shown in Figure 2.12.

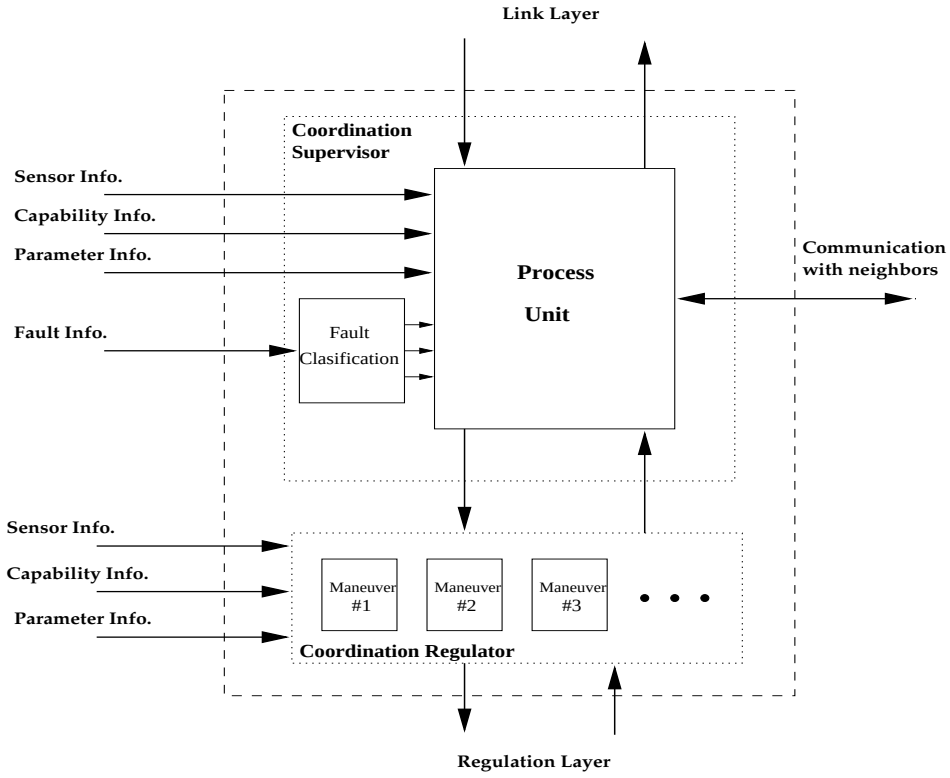


Figure 2.12: Implementation of coordination supervisor

Once a fault has been detected, specialized controllers will be invoked to ensure that the impact of the fault on the system performance is minimized. Since the design complexity increases exponentially with the number of faults, [13] proposed that faults and combinations of faults be grouped into various classes/subclasses. Since each class/subclass of faults affects the system dynamics in a particular fashion, controllers can be designed for the entire class/subclass rather than for individual fault. Thus, we can design only a finite set of controllers. In Figure 2.12 we included a fault classifier module to perform this task. The **process unit** in the coordination supervisor is the central unit. In the current stage we are assuming that there is only one fault in the highway. Therefore the process unit becomes very simple – just invoking the strategy that corresponds to the fault classification. However, in a more general framework, the process unit should:

- Receive the information given by the fault classifier module.
- Accept commands from **Link Layer** to know about the state of the roadside infrastructure.
- Communicate with neighboring vehicles to choose an optimal strategy when the system has lost some capabilities;

- Invoke the proper strategy, depending on the maneuver capabilities of the faulty vehicle and neighboring vehicles.
- Command to the **coordination regulator** the sequence of *atomic maneuvers* that corresponds to the chosen strategy.

In the preliminary design of the coordination supervisor, shown in figure 2.11, it is assumed that there is only one fault at a time, or, that in the event of multiple faults they are isolated by a sufficiently large distance [7]. Notice that the performance structure is not considered.

The normal mode coordination communication protocols were verified in [11] and the degraded mode protocols in [7]. They are both proved to have logic correctness, deadlock free and completeness. It is important to mention that the verification of the finite state machines that implement this coordination layer maneuvers does not guarantee the overall safety of the hybrid system composed by the coordination and regulation layers. Different approaches to guarantee this safety are presented in [12] and [1].

Chapter 3

A Complete Fault Diagnostic System for the Longitudinal Control of Automated Vehicles

3.1 Introduction

The Automated Highway Systems (AHS) Program aims to reduce congestion on highways by closer packing of automatically controlled vehicles into platoons. Studies of automatic control of the longitudinal and lateral motion of cars have been undertaken under the PATH program to establish feasibility of the AHS concept [9, 23, 15]. These experimental studies have demonstrated the viability of automatic driver-less control of cars so as to achieve high traffic throughput on highways.

Studies have shown that over 90% of highway accidents occur due to driver-related errors. The AHS system eliminates these accidents by eliminating the role of the driver. The reliability and safe operation of the hardware is, however, of increased importance. This chapter deals with automated monitoring and diagnostics of the all the sensors and actuators of the longitudinal control system.

Classical results on the design of fault detection filters for linear time-invariant systems are available in [24]. Previous work on fault detection and fault tolerant control related to AHS have been carried out by Speyer, et al [3], Patwardhan and Tomizuka [15] and Garg and Hedrick [5, 6]. The work of Speyer, et al [3] develops fault-detection filters for both longitudinal and lateral sensors using linearized models of vehicle dynamics. The performance of these detection filters is then simulated using the original nonlinear vehicle models. Radar range and range rate and inter-vehicle communication are not included among the sensors considered in the fault detection scheme.

The work by Patwardhan and Tomizuka [15] also uses linear time-invariant dynamic models to design fault detection filters for lateral control sensors. Lateral acceleration, lateral deviation from magnetic markers and yaw rate are measured and used in a fault detection scheme that identifies a fault in any of the three sensors.

The work by Garg and Hedrick [5, 6] on the other hand concentrates on the design of nonlinear filters which can be theoretically proven to identify faults in the original nonlinear system. Application of such nonlinear filters to longitudinal control is also studied and fault detection algorithms are successfully developed for some of the actuators and sensors used in longitudinal control.

The algorithms, however, are not designed to work together to form a coherent fault diagnostic system.

The work presented in this chapter also uses the original nonlinear model directly for the fault detection filter design process. All the sensors and actuators used in the longitudinal controller for platooning are included in the fault monitoring system, including radar range and range rate and inter-vehicle communication. The contribution of the present paper is a complete and automated fault diagnostic system for the entire longitudinal control hardware.

3.2 Simplified Model for Control Design

3.2.1 Simplified Vehicle Model

The reader is referred to Cho, et al [2] and Swaroop, et al [22] for a detailed model of the car's longitudinal dynamics. We present here the simplified model used very effectively for control design in [22].

Under the assumptions that there is no slip between the tire and the road and that the torque converter is locked, the longitudinal velocity of the jth vehicle can be related to the angular velocity of the engine through the gear ratio and tire radius as follows:

$$\dot{x_j} = v_j = (Rh\omega_e)_j \tag{3.1}$$

where

R = gear ratio

h = tire radius

The dynamics relating engine speed ω_e to the pseudo-inputs "net combustion torque" T_{net}, brake torque T_{br}, and aerodynamic losses can be modeled by

$$\dot{\omega}_e = \frac{T_{net} - c_a R^3 h^3 \omega_e^2 - R(hF_f + T_{br})}{J_e} \tag{3.2}$$

where

$J_e = I_e + (Mh^2 + I_w)R^2$ = the effective inertia reflected on the engine side.

The pseudo-input T_{net} is related to the throttle angle α (the actual control or actuator input) by the following dynamics. Steady-state engine maps define T_{net} as a nonlinear function of engine speed ω_e and the mass of air in the intake manifold m_a, $T_{net} = T_{net}(\omega_e, m_a)$. These steady-state maps are available for each car from the manufacturer.

The mass flow rate of air in the manifold is defined by

$$\dot{m}_a = \dot{m}_{ai} + \dot{m}_{ao}(\omega_e, m_a) \tag{3.3}$$

$$\dot{m}_{ai} = MAXTC(\alpha)PRI(m_a) \tag{3.4}$$

where

MAX = a constant dependent on the size of the throttle body

$TC(\alpha)$ = a nonlinear invertible function of the throttle angle

$PRI(m_a)$ = the pressure influence function which describes the choked flow relationship which occurs through the throttle valve

$\dot{m}_{ao}(\omega_e, m_a)$ = a nonlinear function describing the mass flow rate into combustion chamber

3.3 Controller Design

From equations 3.1 and 3.2, it is clear that the acceleration of the jth vehicle can be controlled to any desired positive value by choosing the net combustion torque to be

$$(T_{net})_j = \frac{J_e}{Rh} u_j + [c_a R^3 h^3 w_e^2 - R(hF_f + T_{br})]_j \tag{3.5}$$

By choosing the combustion torque to be the function described above in equations 3.3 and 3.4, the acceleration of the jth vehicle becomes

$$\ddot{x}_j = u_j \tag{3.6}$$

It is important to ensure "string stability" of the platoon [9, 22]. String stability is a term which means that any error in spacing between the first and second cars of the platoon does not result in increasing spacing errors between the other cars further down the platoon. The desired acceleration for each car has to be determined so that a desired constant spacing is maintained between the cars of the platoon and string stability of the platoon is also ensured. Swaroop, et al [22] have shown that both string stability and robustness can be achieved even with very small inter-car spacing if feedback information from all of the following information is used in determining the desired acceleration of each vehicle:

- the car's own velocity
- distance to preceding car
- velocity of preceding car
- acceleration of preceding car
- acceleration of lead car of the platoon
- velocity of lead car of the platoon

Once the desired combustion torque has been determined from equation 3.5, the desired mass of air in the intake manifold and consequently the throttle angle can be determined by using a "multi-surface" sliding mode controller, as described in [22]. If the brake actuator needs to be used for providing the desired synthetic acceleration, the desired brake torque can be calculated from Equation 3.5 by setting the combustion torque to zero. Here we assume that the brake torque is an actuator input to the system and can be directly specified by the user.

3.4 Sensors and Actuators

Having reviewed the vehicle dynamics model and the controller, we now find that the following sensors are needed for longitudinal control:

Table 3.1: Sensor and Actuator Characteristics

Sensor Fault	Information Lost	Noise Level
Radio	Lead and previous vehicle's velocity and acceleration	
Radar	Distance and relative velocity from preceding car	1 inch
Accelerometer	Acceleration	0.5 m/s^2 bias 0.1 m/s^2 noise
Wheel Speed Sensor	Velocity	0.25 m/s
Throttle Angle Sensor	Throttle angle	0.25^o
Mass Flow Rate Sensor	Mass air flow rate into intake manifold	
Brake Pressure Sensor	Brake-line pressure	10 psi
Manifold Temperature Sensor	Manifold temperature	0.25 C^o
Manifold Pressure Sensor	Intake manifold pressure	
Engine RPM Sensor	Engine speed	1 rpm

The information lost due to a fault in any of these sensors is also indicated in the table. The throttle actuator and the brake actuator are the two actuators used by the control system. Throttle angle and brake torque are the actuator inputs used in the diagnostic system design.

3.5 Analytical and Observer-based Redundant Signals

If three measurements of the same variable are available, then the three residues obtained by the three different mutual pairs among these measurements can be used to determine exactly which of the three sensors is at fault. This is demonstrated in section 3.5.1 and used to determine if the wheel speed, engine speed, or range rate sensors are at fault. Once it is ensured that all three of these sensors are operational, several observers are designed using the engine speed and wheel speed measurements to detect faults in the actuators, the engine manifold mass flow rate sensor, and the radar range sensor. Schemes for detecting faults in the accelerometer and the communications system are also described.

3.5.1 Speed Sensor Redundancy

The longitudinal speed of the vehicle can be obtained by three different methods, as described in [5].

1. Wheel speed sensor

   Multiply the angular wheel speed by the tire radius to obtain longitudinal velocity. It is assumed that there is no slip between the tire and the road.

2. Engine speed sensor

   At speeds above 30 mph, the torque converter is locked. The engine speed is then directly related to the wheel speed by the gear ratio.

3. Range rate sensor

   The closing rate with the preceding vehicle (relative velocity) can be obtained using the radar sensor. The speed of the preceding vehicle is obtained through radio communication. The two variables can be algebraically summed to obtain longitudinal velocity.

The following three residues are then calculated by using different combinations of the above three longitudinal velocity signals.

R_1 = wheel speed/engine speed residual

R_2 = wheel speed/radar range rate residual

R_3 = engine speed/radar range rate residual

Table 3.2 is provided as an illustration of how three similar signals can be used to determine a fault in any of the three.

Table 3.2: Truth Table for Speed Sensor Fault Detection

Faulty Component	Residual R_1	Residual R_2	Residual R_3
Radar closing - rate sensor	Low	High	High
Engine speed sensor	High	Low	High
Wheel speed sensor	High	High	Low

3.5.2 Inter-car spacing

We propose the following two methods to obtain inter-car spacing information.

1. Currently a radar sensor is used to measure the distance between the vehicle (i) and the preceding vehicle (i-1)
$$\delta_i = x_i - x_{i-1} \tag{3.7}$$

2. The following observer is proposed in this paper to obtain one more estimate of inter-car spacing. This observer for the ith vehicle uses a magnetometer measurement to count the number of magnetic markers passed by the ith and i-1th vehicles
$$\dot{\hat{\delta}}_i = v_i - v_{i-1} + k_s[(n_{i-1} - n_i)L + \delta_o - \hat{\delta}_i] \tag{3.8}$$

   where

   $(n_{i-1} - n_i)$ = the difference in the number of markers passed by the previous and current vehicle

   L = the inter-marker spacing.

   The estimation error using the given observer is
$$\dot{\tilde{\delta}}_i = -k_s[(n_{i-1} - n_i)L + \delta_o - \tilde{\delta}_i] \tag{3.9}$$

The variable $[(n_{i-1}-n_i)L+\delta_o-\hat{\delta}_i]$ is equal to $\tilde{\delta}_i$ to within a resolution of L meters. The use of this variable ensures that any drift associated with integrating the velocities v_i-v_{i-1} is eliminated. If the signal $v_i - v_{i-1}$ were perfect with no dc offsets, the use of the signal $[(n_{i-1} - n_i)L + \delta_o - \hat{\delta}_i]$ would be unnecessary.

3.5.3 Throttle Actuator or Throttle Angle Sensor and Manifold Mass Flow Rate Sensor Faults

In [5], two different nonlinear detection filters are proposed for the throttle actuator and the manifold mass flow rate sensor fault detection. A first order detection filter is constructed so as to estimate engine speed asymptotically in the absence of throttle actuator fault. A fourth order detection filter that estimates manifold mass flow rate along with several other variables is also proposed for fault diagnosis of the mass flow rate sensor.

We propose using one second order nonlinear observer to estimate both the engine speed and manifold mass flow rate from engine speed measurements.

$$\dot{\hat{\omega}}_e = \frac{T_{net}(\hat{\omega_e}, m_{a,des}) - c_a R^3 h^3 \hat{\omega_e^2} - RhF_f}{J_e} + l_1(\omega_e - \hat{\omega_e}) \tag{3.10}$$

$$\dot{\hat{m}}_a = MAXTC(\alpha_{des})PRI(\hat{m}_a) - \dot{m}_{ao}(\hat{\omega}_e, \hat{m}_a) + l_2(\omega_e - \hat{\omega}_e) \tag{3.11}$$

This observer can be designed to be asymptotically stable in the absence of throttle actuator faults by proper choice of gains l_1 and l_2. The gains can be calculated using observer design results from [17] assuming the nonlinear functions to be Lipschitz.

A throttle actuator fault will cause the residue between estimated and measured engine speeds to grow. Assuming no fault in the engine speed measurement sensor, the growth in this residue can then be used to diagnose a throttle actuator fault. If a throttle actuator fault has **not** occurred, the residue between measured and estimated manifold mass flow rate can be used for diagnostics of the mass flow rate sensor.

3.5.4 Brake Actuator and Sensor Fault

Under the action of brakes, the throttle is set to the idle position. Due to internal friction in the engine and pumping losses, a negative T_{net} is created which assists in slowing the vehicle. This "engine braking" is typically one to two orders of magnitude smaller than the applied brake torque T_{br}, and can therefore be neglected in Equation 3.2. The following observer can then be used to estimate the engine speed under the action of the braking actuator

$$\dot{\hat{\omega}}_e = \frac{-c_a R^3 h^3 \hat{\omega}_e^2 - R(hF_f + T_{br,des})}{J_e} + l_1(\omega_e - \hat{\omega}_e) \tag{3.12}$$

The dynamics of the estimation error $\dot{\tilde{\omega}}_e = \omega_e - \tilde{\omega}_e$ are then given by

$$\dot{\tilde{\omega}}_e = -a\tilde{\omega}_e^2 - l\tilde{\omega}_e - c\mu(t) \tag{3.13}$$

where

$a = \frac{C_a R^3 h^3}{J_e}$

$c = \frac{R}{J_e}$

$\mu(t) =$ nonzero only when there is a fault in the brake actuator.

Since the engine speed can never physically exceed 5000 rpm, the nonlinearity ω_e^2 can be regarded as locally Lipschitz. Since the observer has access to a measurement of ω_e, the gain l can

be chosen larger than the Lipschitz constant of ω_e^2 in order to ensure stability of the estimation error dynamics in the absence of faults.

3.5.5 Vehicle Speed Estimation using Accelerometers and Markers

The following observer using the accelerometer on the car and a magnetometer measurement to count the number of magnetic markers passed by the cars can be used to estimate car velocity

$$\dot{\hat{v}} = a + k_v(\frac{\Delta nL}{T} - \hat{v}) \tag{3.14}$$

where

a = the vehicle acceleration

Δn = the marker count of the vehicle over the time interval T

L = the inter-marker spacing

The estimation error using the given observer is

$$\dot{\tilde{v}} = -k_v(\frac{\Delta nL}{T} - \tilde{v}) \tag{3.15}$$

The variable $(\frac{\Delta nL}{T} - \tilde{v})$ is equal to $\tilde{v}$ to within a resolution of L meters. The use of this variable ensures that any drift associated with integrating the acceleration a is eliminated. If the signal a were perfect with no dc offsets, the use of the signal $(\frac{\Delta nL}{T} - \tilde{v})$ would be unnecessary.

3.5.6 Communication fault

Faults in the vehicle's communications system are detected as follows:

1. The car that communicates ensures that its sensors are not faulty.
2. If no packet is received, the information from the last packet is frozen till the next packet arrives.
3. If no packet is received for more than 3 consecutive cycles, a communication fault is declared.

3.6 A System for Automated Fault Diagnosis

Table 3.3 summarizes 18 different signals to be used in the fault detection and identification scheme. Some of the signals are directly measured while others are estimates obtained from the observers discussed in the previous section.

Table 3.4 summarizes 10 different residues calculated using combinations of the signals from the previous table.

By processing the 10 residues, it is possible to identify a fault in any of the sensors or actuators. Table 3.5 shows how a fault in any of the sensors or actuators causes a **unique** combination of residues to grow.

Table 3.3: Bank of signals for fault diagnostics

Signal	Description	Sensor Observer
$z_1 = h\omega_w$	vehicle speed	wheel speed sensor
$z_2 = Rh\omega_e$	engine speed	engine speed and gear ratio sensors
$z_3 = x_{i-1} - x_i$	distance to preceding car (range)	radar range signal
$z_4 = v_{i-1} - v_i$	relative velocity of preceding car	radar range rate signal
$z_5 = v_{i-1}$	velocity of preceding car	communication
$z_6 = \omega_e$	engine speed	engine speed sensor
$z_7 = \dot{m}_a$	mass flow rate of air in manifold	mass flow rate sensor
$z_8 = a_i$	acceleration	accelerometer
$z_9 = a_{i-1}$	acceleration of preceding car	communication
$z_{10} = \hat{\omega}_e$	estimated engine speed	observer of Eqns. 3.10 & 3.11
$z_{11} = \hat{m}_a$	estimated flow rate in manifold	observer of Eqns. 3.10 & 3.11
$z_{12} = \alpha_{des}$	commanded throttle angle	calculated by controller
$z_{13} = \alpha$	throttle angle	throttle angle sensor
$z_{14} = T_{br,des}$	commanded brake torque	calculated by controller
$z_{15} = \hat{\delta}_i$	estimated distance to preceding car	observer of Eqn. 3.8
$z_{16} = \hat{v}_i$	estimated velocity	observer of Eqn. 3.14
$z_{17} = u_i$	synthetic acceleration	calculated by controller
$z_{18} = \hat{\omega}_e$	estimated engine speed during braking	observer of Eqn. 3.12

To detect and identify faults, the algorithm depicted in Figure 3.1 can be used. The algorithm has been obtained from Table 3.5 and is a systematic method of using Table 3.5 to successively check for faults in each of the sensors and actuators.

3.7 Simulation Results

The fault detection system designed in the previous sections was simulated to test its performance with a more realistic vehicle model incorporating a torque converter, wheel slip, tire radius variation, sensor noise, etc. Details of the full vehicle simulation model are available in [9]. The **noise levels assumed for the sensor measurements** are shown in Table 3.1 and are realistic estimates based on experimental measurement. The marker spacing was assumed to be 1 m.

For the simulation, a 3 car platoon was assumed to be traveling with a spacing of 1 m at a speed of 65 mph. At time t = 3 secs, the lead car begins the following velocity maneuver

$$v_{des} = 3.5[1 - cos20\pi(t-3)]$$

Figures 3.2 to 3.4 show the performance of the nonlinear observers designed in Section 3.5 in the absence of a fault. Figure 3.2 shows the convergence of the inter-car spacing observer and its ability to track the actual radar measurement in the presence of noise and 1 m marker spacing.

Figures 3.3 and 3.4 show the performance of the 2nd order nonlinear observer of Equations 3.10 and 3.11. The engine speed and mass flow rate of air in the manifold are estimated well

Table 3.4: Calculation of residues

Residues	Sensors & Acuators Involved
$R_1 = z_1 - z_2$	wheel speed sensor engine speed sensor
$R_2 = z_5 - z_4 - z_1$	radar range rate sensor wheel speed sensor communication
$R_3 = z_5 - z_4 - z_2$	radar range rate sensor engine speed sensor communication
$R_4 = z_3 - z_{15}$	radar range sensor wheel speed sensor magnetometer communication
$R_5 = z_{17} - z_8$	accelerometer
$R_6 = z_{16} - z_1$	accelerometer markers wheel speed sensor
$R_7 = z_{10} - z_2$	throttle actuator engine speed
$R_8 = z_{12} - z_{13}$	throttle angle sensor throttle actuator
$R_9 = z_{11} - z_7$	mass flow rate sensor throttle actuator
$R_{10} = z_{18} - z_2$	brake actuator engine speed

Table 3.5: Behavior of Residues under Sensor/Actuator Faults

Faulty Sensor / Actuator	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}
wheel speed sensor	H	H	L	H	L	L	L	L	L	L
engine speed sensor	H	L	H	L	L	L	L	L	L	L
radar range rate sensor	L	H	H	L	L	L	L	L	L	L
radar range sensor	L	L	L	H	L	L	L	L	L	L
accelerometer	L	L	L	L	H	H	L	L	L	L
magnetometer	L	L	L	H	L	H	L	L	L	L
throttle actuator	L	L	L	L	L	L	H	H	H	L
throttle angle sensor	L	L	L	L	L	L	L	H	L	L
mass flow rate sensor	L	L	L	L	L	L	L	L	H	L
brake actuator	L	L	L	L	L	L	L	L	L	H

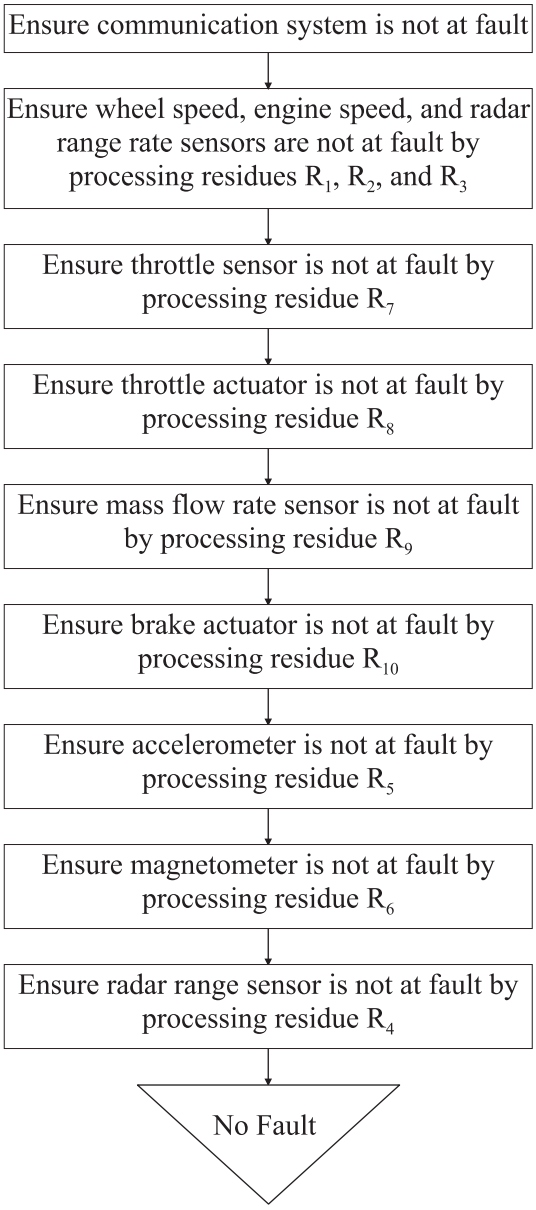


Figure 3.1: Fault Diagnosis Algorithm

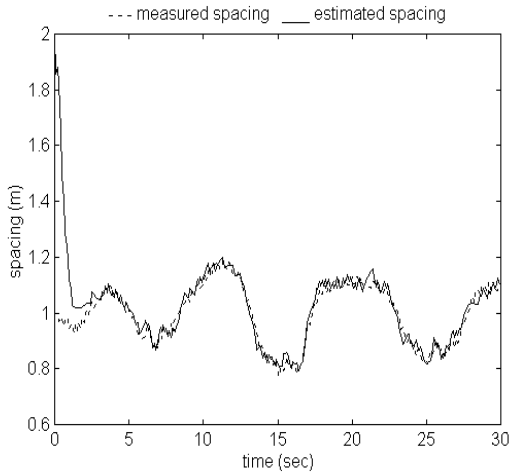


Figure 3.2: Inter-car spacing observer in the absence of faults

by the observer. The jumps in engine speed occur due to down-shifting from the 4th to the 3rd gear, as shown in Figure 3.5.

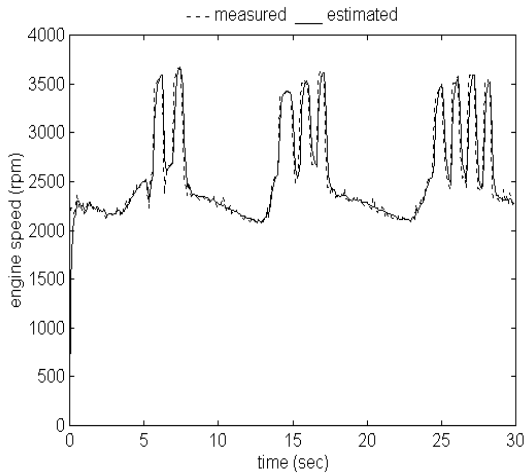


Figure 3.3: Engine speed observer of Equations 3.10 and 3.11 in the absence of faults

In the presence of a radar fault, the inter-car spacing observer diverges distinctly from the measured value, as shown in Figure 3.6. Here a fault in the radar was assumed to cause it to read a constant value of 6 m. The observer estimate converges to a value of -4 m, thus ensuring that the residue from the radar measurement is sufficiently big to identify the fault. Figure 3.7 shows the values of all 10 residues of Table 3.5 during the radar fault. We see that only residue 4 is high which clearly indicates from Table 3.4 that the radar sensor is the fault one.

3.8 Experimental Results

This section presents experimental results on the use of the magnetic observer of equations 3.8 and 3.9. The magnetic observer was implemented on the automated cars in the August 1997 NAHSC

demonstration. The observer was used both to detect faults in the radar range sensor and also to replace the radar in the closed-loop controller in the event of a radar fault.

In the August 1997 NAHSC demonstration, passengers were allowed to ride in a platoon of eight fully automated cars operating at close spacing. During the demonstration, the automated vehicles performed a series of maneuvers, such as lane changes, splitting of the platoon, and merge maneuvers to demonstrate the capabilities of the system. To guarantee the safety of the passengers, a fault detection and management scheme was implemented to monitor the throttle and brake actuators, the communications system, and the radar range sensor.

The magnetic observer played a very important role in ensuring safe automated operation during the demonstration. Radar range sensor faults were detected and automatically replaced by the magnetic observer on several occasions during the platoon runs.

Figure 3.8 is a good illustration of the ability of the magnetic observer to replace the radar range sensor. To allow range measurement, a rectangular opening had been designed into the front grill of each car. The radar was located behind this grill and below the hood of each car. In the following test run, a mis-orientation of the grill mounting caused the radar to fail repeatedly on the fifth car in the platoon. The readings of the radar jump from zero to the correct spacing value many times during the run.

The magnetic observer worked well throughout this run and provided a fairly accurate estimate of inter-car spacing. The fault detection system was triggered due to the 6 meter difference in the actual and estimated values of range. In response to the radar fault, the spacing between cars 5 and 4 was increased to 15 meters by the fault management system. The remainder of the run continued at this larger spacing. The closed-loop controller used the magnetic observer estimate to replace the radar range measurement in the calculation of synthetic acceleration. The reconfigured controller was able to provide an excellent ride with a spacing variation of less than 1.3 meters. The maximum errors in spacing occurred in the presence of uphill and downhill grades.

An error of one or two magnet spacing occured occasionally in the spacing estimate of the magnetic observer due to the magnetometer system failing to detect the passing of a magnet on the highway. To prevent a permanent bias in the observer estimate due to a missed magnet, the following algorithm was used to detect if a magnet had been missed. If the time taken to detect the

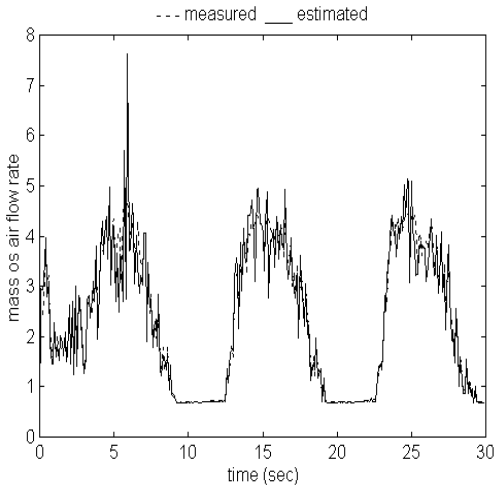


Figure 3.4: Mass flow rate observer of Equations 3.10 and 3.11 in the absence of faults

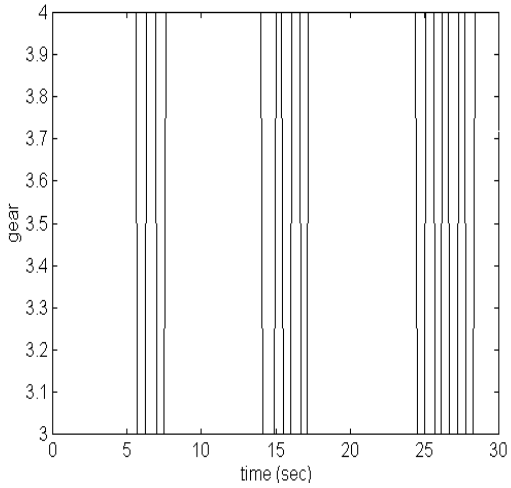


Figure 3.5: Gear shifting during the maneuver

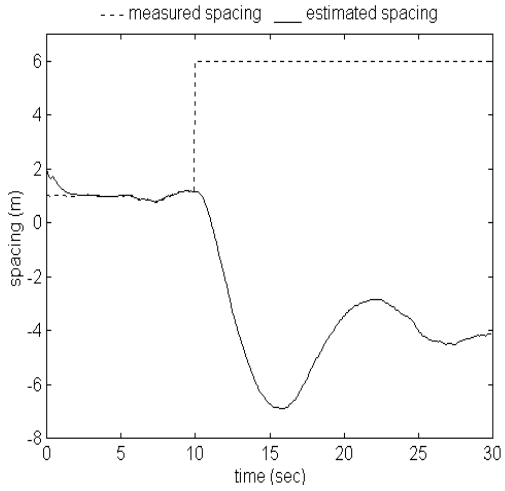


Figure 3.6: Radar fault causes the inter-car spacing observer to diverge

next magnet exceeded a threshold, it was assumed that the magnetometer had missed the detection of a magnet. The threshold was determined as a function of the average velocity during that time

$$t_{nextmagnet} > \frac{1.5L}{v_{average}} \tag{3.16}$$

This means that the magnetic observer could have errors of the order of 1 - 2 meters for brief periods of time in the spacing estimate.

3.9 Conclusions

The fault diagnostic system developed in this chapter was shown to work well when simulated with a detailed vehicle model incorporating realistic unmodeled dynamics. Experimental results using the magnetic observer to detect radar faults and replace the radar sensor were shown to work extremely

effectively. The diagnostic system developed provides a methodology to continuously monitor all the sensors and actuators of the longitudinal control system so as to ensure their health.

Related analytical results of interest include the development of sufficient and necessary conditions as well as a systematic methodology to enable detection filter design for nonlinear systems.

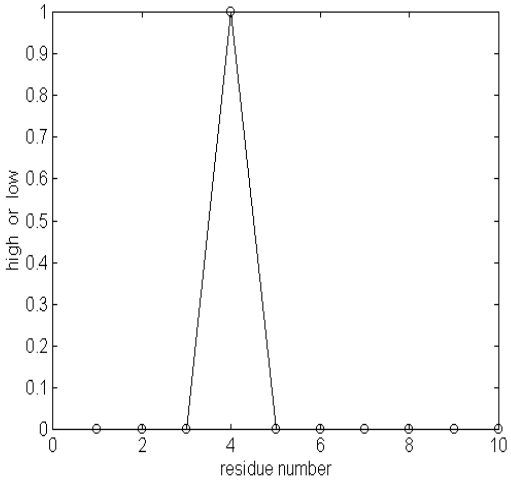


Figure 3.7: Values of the different residues of Table 3.4 during a radar sensor fault

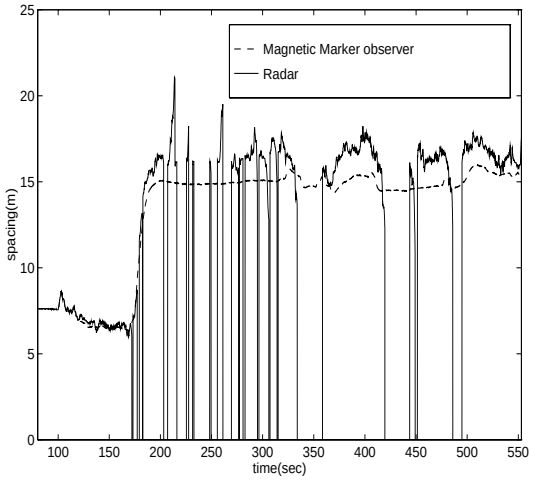


Figure 3.8: Experimental Results on the use of the magnetic observer for radar faults

Chapter 4

A Diagnostic Protocol for Radar Faults

We are interested in the diagnosis of faults in the components of the longitudinal control system of a platoon follower vehicle. It turns out that system level diagnostics for radar range rate measurements requires inter-vehicle coordination. The following table describes a residue scheme for this purpose. A residue is the difference of the signals from the information sources indicated in the table. The information sources generating a residue are used to generate two independent estimates of the same quantity. If the information sources are all normal then the two estimates should agree and the residue should be zero. On the other hand if there is a fault in one of the information sources then the estimates will disagree and the residue will be high.

Residue no.	Information Sources	Assumptions	Quantity Estimated
R1	Wheel Speed Sensor Engine Speed Sensor	Torque Converter Locked	Wheel Speed
R2	Observer using magnetometers, accelerometer Wheel Speed Sensor	No slip	Wheel Speed
R3	Radar Range-rate Sensor Wheel Speed Sensor Wheel Speed Sensor of the vehicle in front	No slip	Wheel Speed

Table 4.1: Residue Table

The observer equation is similar to chapter 3 and is as follows:

$$\dot{\hat{v}} \;=\; a_m + k(v_m - \hat{v}),$$

where,

$\hat{v}$ = inertial velocity estimate from the observer,
a_m = longitudinal acclerometer measurement,
k = observer gain,
v_m = is the velocity estimate from magnetometer measurements.

The residue truth table is as follows.

Fault	R1	R2	R3
Normal	L	L	L
WSS	H	H	H
ESS	H	L	L
ACC	L	H	L
RRS	L	L	H
WSS of vif	L	L	H
ACC & WSS of vif	L	H	H
ESS & WSS of vif	H	L	H

Table 4.2: Residue Truth Table

In the subsequent development we assume that there is some independent means of validating the low slip assumption. Then the methods developed in [16] may be used to diagnose magnetometer failures. If the magnetometers have failed then the residues presented in table 4.2 should be ignored. Otherwise the residue values may be analyzed.

The torque converter locked assumption may be checked by checking the gear the car is in. In third and fourth gears the torque convertor is locked. Therefore at lower gears the residues should be ignored.

Assuming that there is no slip, the torque converter is locked, there is not more than one fault per vehicle, there are no faults in the magnetometers, the truth table and the properties of the associated observer indicate that a vehicle can detect failures in its wheel speed sensor(WSS), engine speed sensor (ESS), and longitudinal accelerometer (ACC). The radar range rate sensor failure is not separable from the wheel speed sensor failure of the vehicle in front (vif) without a diagnostic protocol. We describe the protocol modeling formalism briefly.

We assume that all protocol entities are modelled by finite state machines. The following definition of a FSM is adapted from [19]. A FSM is a 4-tuple

$$M = \langle \Sigma, Q, 0, \delta \rangle,$$

where Σ is the alphabet or finite set of events associated with the state transitions of the system, Q is the set of states, 0 is the initial state, and $\delta : \Sigma \times Q \rightarrow Q$ is a partial function known as the transition function. If $\delta(\sigma, x) = x'$ it signifies that if the system is in state x then the occurrence of event σ will cause the system to transition to the state x'. Note that since δ is a partial function transitions do not necessarily exist for all triples $(\sigma, x, x') \in \Sigma \times Q \times Q$. This defintion is almost identical to that in [19]. The only difference is that M has no marking function.

We will need a composition operator on finite state machines. The following definition of the *synchronous composition* operator is adopted from [19]. Let $M_1 = \langle \Sigma_1, Q_1, 0_1, \delta_1 \rangle$ and $M_2 = \langle \Sigma_2, Q_2, 0_2, \delta_2 \rangle$ be two FSM's. We also define a binary relation $R \subseteq \Sigma_1 \times \Sigma_2$. This relation is symmetric and transitive and means that if two events in two machines are related then they must be executed together. We let R_{Σ_1} and R_{Σ_2} represent the projections of R on the sets Σ_1 and Σ_2 respectively. Formally the synchronous composition of M_1 and M_2 is a FSM $M = \langle \Sigma, Q, 0, \delta \rangle$ defined as follows.

$$\Sigma \quad = \quad (\Sigma_1 - R_{\Sigma_1}) \cup (\Sigma_2 - R_{\Sigma_2}) \cup R$$

$$\begin{aligned} Q &= Q_1 \times Q_2 \\ 0 &= (0_1, 0_2) \end{aligned}$$

The transtion function of M is defined by

$$\begin{aligned} \delta(\sigma,(q_1,q_2)) &= (\delta_1(\sigma,q_1),q_2) && \text{if } \sigma \in \Sigma_1 - R_{\Sigma_1}, \\ \delta(\sigma,(q_1,q_2)) &= (q_1,\delta_2(\sigma,q_2)) && \text{if } \sigma \in \Sigma_2 - R_{\Sigma_2}, \\ \delta((\sigma,\sigma'),(q_1,q_2)) &= (\delta_1(\sigma,q_1),\delta_2(\sigma',q_2) && \text{if } (\sigma,\sigma') \in R, \end{aligned}$$

and undefined otherwise.

Observe that though diagnosing the range rate sensor fault requires information on the status of the wheel speed sensor of the vehicle in front, diagnosis of wheel speed sensor failures does not require information from any other vehicle. Therefore if residue $R3$ turns high then the vehicle in front should check that its wheel speed sensor is normal. If this is so then the follower vehicle knows that its range-rate sensor has failed. This is reflected in the following protocol. Figure 4.1 is the diagnostic protocol finite state machine of a vehicle. The events $F\_WSS, F\_ESS, F\_ACC, F\_RRS$ represent messages given to the control system indicating failures of the wheel speed sensor, engine speed sensor, accelerometer, and radar range-rate sensor respectively. The event $F : F\_WSS$ is a message from the car in front indicating that its wheel speed sensor has failed. It is executed in synchrony with the event F_WSS in the finite state machine of the car in front. The synchronous composition of the finite state machines indicates that if a radar range rate sensor fault occurs then the appropriate message is generated. Moreover, the message is never generated without the fault provided the assumptions stated earlier are valid. The residue events are represented by a triple of symbols.

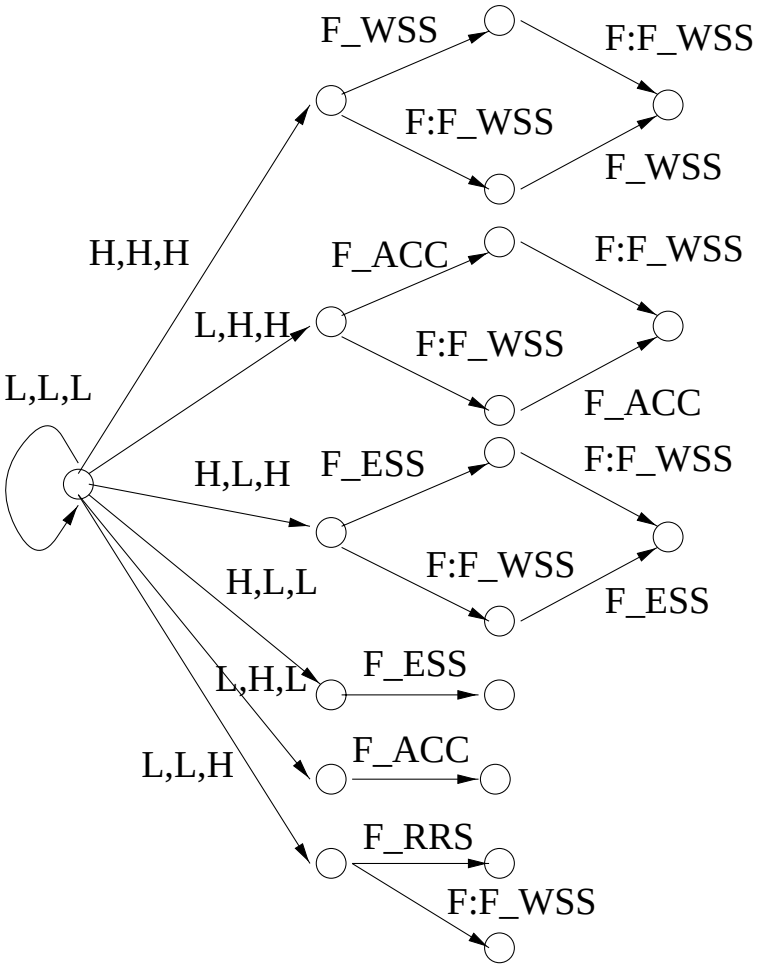


Figure 4.1: Diagnostic Protocol Finite State Machine

Chapter 5

A Methodology for the Integration of Vehicle Failure Diagnostics

5.1 Introduction

We are interested in the diagnosis of failures that occur in the sensing, control, acutation, and communication components of vehicles. Present day cars, trucks and buses are large-scale systems equipped with many components and layers of control intelligence. Since failure diagnosis requires either analytical or physical redundancy, diagnostic schemes cannot be designed component by component, but must be designed for the system as a whole. This makes diagnostic design a complex and large-scale problem that needs a systematic design procedure. We propose a model based design method and demonstrate it by application to the design of diagnostics for the longitudinal control system of a fully automated vehicle capable of operating in a platoon on an Automated Highway System. However, we believe that the methods are generic enough to apply to partially automated vehicles as well.

We will assume that the vehicle, hereafter referred to as the plant, is modelled in two domains, i.e., a continuous model domain and a discrete event model (DEM) domain. Figure 5.1 illustrates the architectural organization associated with this modelling decomposition. The redundancy necessary for diagnosis is created by including sensors and observers in the continuous domain. The outputs of the sensors and observers are compared by generating residues. A residue is defined in the continuous domain and is an *algebraic expression in two or more variables, where each variable is a real valued function of time and represents a sensor, control or observer output.* We assume that if the components generating the residue are normal then they will produce the same outputs, and consequently the residue will have a low value. On the other hand if one of the components has failed, then they will not produce the same outputs and the residue will have a high value. Formally, let $\mathcal{R}$ be the set of residues. For each residue we assume the existence of a finite set of symbols Σ_r and a map $\gamma_r : I\!R \to \Sigma_r$. The function γ_r maps the residue into finitely many symbolic values. It is assumed that γ_r is onto and that the sets $\Sigma_r, r \in \mathcal{R}$ are pairwise disjoint. The set of symbols $\Sigma_R = \uplus_{r\in\mathcal{R}}\Sigma_r$ are called *residue events.* These symbols are incorporated into discrete event plant models. We do not discuss the design of the interface between the continuous models and DEM's further. This paper focusses on other aspects of the diagnostic problem. Note that in our architecture a discrete event plant model is not a model of an uncontrolled plant. It is actually a model of a system with sensors, observers and other continuous controllers, i.e., it is a model of a system that already has

intelligence instilled with control and possibly diagnostic requirements in mind. The DEM plant is a model of the causal relationships between failures, residue values and control.

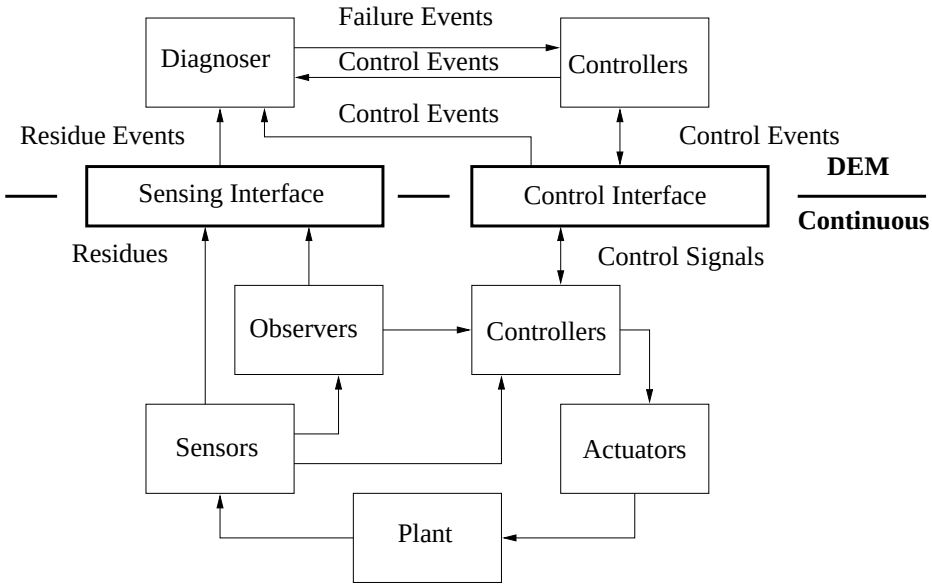


Figure 5.1: Fault Management Architecture

Since designing diagnostics requires analysis of a large scale system, it is desirable that the design process have certain properties. First, it is desirable that there be a means of *verifying* that a diagnostic scheme is correct. We believe, that it is non-trivial to ascertain whether a proposed diagnostic scheme indeed diagnoses all the failures it was designed to diagnose as the vehicle passes through multiple operating modes and environmental conditions. We are interested in ways to formulate the failure diagnosis problem in the context of discrete event models. One of the advantages of such an approach, is that given a formal specification of a diagnostic requirement, a formal specification of a diagnostic design, and a DEM for the system, it is possible to develop algorithms for certain types of formalisms that automatically verify whether a specified scheme meets the specified requirement. Secondly, if a diagnostic scheme is verified to be correct it is desirable to obtain a specification of the software or hardware needed to perform the diagnosis. Finally, for model based methods, the modelling process should be modular. This ensures that if the components or control schemes of the system changes, plant models and diagnostic designs may be re-constructed with relative ease.

The diagnostic design methods proposed in this paper are derived from those presented in [20, 21]. The proposed methods are intended to offer partial solutions to the problems of modeling, verification, and the problem of specifying the software or hardware required to perform the diagnosis. We assume that all our DEM plant models are finite state machines (FSM's) [19]. This class of DEM's appears adequate for the vehicle daignostic problems we have investigated so far. We introduce a formalism for specifying a diagnostic requirement and a system property we call *FH-diagnosability*. This property is based on the *diagnosability* property defined in [20]. The FH-diagnosability notion is consistent with a more flexible way of specifying diagnostic requirements. Though this is the case, we have been able to define the property in such a way that it is possible to map the verification of FH-diagnosability into the verification of diagnosability by using the synchronous composition operator [19]. The entity performing the diagnosis, hereafter called the diagnoser, is specified in the same way as [20].

The layout of the paper is as follows. Section 2 presents mathematical definition of the required DEM's and compostion operators, the formal specification of a diagnostic requirement and the definition of FH-diagnosability. Section 3 presents our plant modeling method in detail and is particularly concerned with the modular modelling. Section 4 describes the diagnoser construction and the verification methods. The material in this section is a summary of that in [20]. It is included to enhance readability. Section 5 descibes the application of our methods to the design of diagnostics for the longitudinal control system of a fully automated vehicle capable of operating in a platoon. The continuous domain sensor, observer and residue design for the application is from [18]. Section 6 is a summary of the paper and a discussion of future work.

5.2 Problem Formulation

This section formalises the DEM's used to model the plant, the specification of a diagnostic requirement, and the definition of FH-diagnosability.

We assume all DEM's modeling the behavior of the vehicle are deterministic finite state machines (FSM's). The following definition of a FSM is adapted from [19]. A FSM is a 4-tuple

$$M = \langle \Sigma, Q, 0, \delta \rangle,$$

where Σ is the alphabet or finite set of events associated with the state transitions of the system, Q is the set of states, 0 is the initial state, and $\delta : \Sigma \times Q \to Q$ is a partial function known as the transition function. If $\delta(\sigma, x) = x'$ it signifies that if the system is in state x then the occurrence of event σ will cause the system to transition to the state x'. Note that since δ is a partial function transitions do not necessarily exist for all triples $(\sigma, x, x') \in \Sigma \times Q \times Q$. This defintion is almost identical to that in [19]. The only difference is that M has no marking function.

$L(M)$ denotes the set of all sequences of transitions generated by the FSM starting at the initial state. $L(M)$ is called the language generated by the FSM M. If Σ is the alphabet of M then $L(M)$ is a subset of Σ^* (the kleene closure of Σ [10]). To define the language generated by a FSM formally, we introduce the extended transition function $\delta^* : \Sigma^* \times Q \to Q$ defined by

$$\begin{array}{lcll} \delta^*(\varepsilon, x) & = & x, & \text{for } x \in Q, \\ \delta^*(\sigma, x) & = & \delta(\sigma, x), & \text{for } \sigma \in \Sigma, \text{ and } \delta(\sigma, x) \text{ is exists}, \\ \delta^*(s\sigma, x) & = & \delta(\sigma\delta^*(s, x)), & \text{for } s \in \Sigma^*, \sigma \in \Sigma, \text{ and } \delta^*(s, q) \text{ exists}, \end{array}$$

and undefined otherwise. The symbol $\varepsilon \in \Sigma^*$ denotes the empty string. Then $L(M) = \{s \in \Sigma^* : \delta^*(s, 0) \text{ is defined}\}$. For $s \in \Sigma^*$ if there exists $u, v \in \Sigma^*$ such that $uv = s$ then we say that u is a prefix of v and denote it by $u \leq s$. Note that the language generated by a FSM is always prefix-closed as per the definition of $L(.)$. All languages in the subsequent development are assumed to be prefix-closed. It is also assumed that all the FSM's are accessible w.r.t. the initial state, i.e., for all $x \in Q$, there exists $s \in \Sigma^*$ such that $\delta^*(s, 0)$ is defined and $x = \delta^*(s, 0)$.We will often use the unit FSM $M_\sigma = \langle \{\sigma\}, \{0_\sigma, 1_\sigma\}, 0_\sigma, \delta_\sigma \rangle$, where $\delta_\sigma(\sigma, 0_\sigma) = 1_\sigma$, and δ_σ is undefined otherwise, as a building blocks for larger FSM's.

It is assumed that the set of events Σ is the union of three disjoint sets, i.e., $\Sigma = \Sigma_F \uplus \Sigma_R \uplus \Sigma_{con}$. Σ_F denotes the set of failure events and should be selected to represent all the failures of interest. $\Sigma_R = \uplus_{r \in \mathcal{R}} \Sigma_r$ is the set of residue events, as discusssed in the introduction. Σ_{con} is the set of all control events and is meant to represent control commands or messages. The examples presented in section 5.5 demonstrate the use of these sets.

The set of events is also partitioned into observable and unobservable events, i.e., $\Sigma = \Sigma_o \uplus \Sigma_{uo}$. We assume $\Sigma_o = \Sigma_{con} \uplus \Sigma_R$, i.e., all control and residue events are observable, and $\Sigma_{uo} = \Sigma_F$, i.e., all failure events are unobservable.

We will need a composition operator on finite state machines. The following definition of the *synchronous composition* operator is adopted from [19]. Let $M_1 = \langle \Sigma_1, Q_1, 0_1, \delta_1 \rangle$ and $M_2 = \langle \Sigma_2, Q_2, 0_2, \delta_2 \rangle$ be two FSM's. The synchronous composition of M_1 and M_2 is a FSM $M = \langle \Sigma, Q, 0, \delta \rangle$ defined as follows.

$$\begin{aligned} \Sigma &= \Sigma_1 \cup \Sigma_2 \\ Q &= Q_1 \times Q_2 \\ 0 &= (0_1, 0_2) \end{aligned}$$

The transtion function of M is defined by

$$\begin{aligned} \delta(\sigma, (q_1, q_2)) &= (\delta_1(\sigma, q_1), q_2) && \text{if } \sigma \in \Sigma_1 - \Sigma_2, \\ \delta(\sigma, (q_1, q_2)) &= (q_1, \delta_2(\sigma, q_2)) && \text{if } \sigma \in \Sigma_2 - \Sigma_1, \\ \delta(\sigma, (q_1, q_2)) &= (\delta_1(\sigma, q_1), \delta_2(\sigma, q_2) && \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \end{aligned}$$

and udefined otherwise.

We use the symbol M_P to denote the plant model in the DEM domain. The examples in section 5.5 illustrate the construction of M_P. M_P models

- all sequences of failures that can happen in the system under consideration,
- the causal relationships between the failures, control commands and residue trajectories.

We assume that once a component fails it stays failed. Therefore it is meaningless for a component to fail twice. It is assumed that in any sequence of events $s \in L(M_P)$ has a particular failure event in it at most once.

A diagnostic requirement is specified using two formalisms. The first is a partition Π_F

$$\Sigma_F = \uplus_{i \in \mathcal{I}(\Pi_F)} \Sigma_{Fi},$$

on the set of failure events. $\mathcal{I}(\Pi_F) \subseteq I\!N$ is an index set associated with the partition Π_F. The partition represents the fineness with which the diagnostic scheme is required to pinpoint a failure. For example, if a failure event $\sigma \in \Sigma_{Fi}$ occurs then instead of being required to diagnose σ, it is deemed adequate for the diagnoser to diagnose that some event in the set Σ_{Fi} has occurred, i.e., a failure of type i has occurred. Such a relaxation of diagnostic requirements may be useful because

- inadequate instrumentation may render it impossible to diagnose uniquely every possible failure,
- failure management control strategies may be the same for several failures. Thus it may not be important to distinguish them for the purposes of control.

The finest partition, $\Pi_F^{\uparrow} = \uplus_{\sigma \in \Sigma_F} \{\sigma\}$, represents the most stringent diagnostic requirement.

The second formalism is also a way of relaxing diagnostic requirements. For example, it may be acceptable to do the diagnosis assuming that the system has no more than one failure

at any given time. Such an assumption may be justified on the premise that the occurrence of two simultaneous failures or the occurrence of a second failure before repair of the first are sufficiently rare events. However, the plant model being non-probabilistic, does not distinguish sequences of failure events that are likely from those that are not. It generates all the failures that can occur in various possible orders. Nevertheless, the designer may consider it acceptable to exclude certain sufficiently rare but troublesome conbinations of failures from the diagnoser synthesis and verification exercise. This is accomplised by defining the *failure hypothesis FSM* $M_F = \langle \Sigma_F, Q_F, 0_F, \delta_F \rangle$, and putting it in synchronous composition with M_P. Once again, it is assumed that any sequence of events $s \in L(M_F)$ has a particular failure event in it at most once. Note that there is a maximal $M_F^{\uparrow} = \otimes_{\sigma \in \Sigma_F} M_\sigma$. $M_F^{\uparrow}$ has the property $L(M_F \otimes M_P) \subseteq L(M_F^{\uparrow} \otimes M_P) = L(M_P)$ for all M_F. It is assumed that M_F and M_P are so defined that $M_D = M_F \otimes M_P$ is *live* [20], i.e., the transition function is defined for at least one event out of every state. The FSM M_F is a way of separating hypotheses on the occurrence of failures from the plant models. This allows the same set of plant models to be re-used and verified under different fault hypotheses for a given failure partition.

Before stating the defintion of diagnosability and FH-diagnosability we introduce some more notation. Let $\overline{s}$ denote the prefix-closure of any trace $s \in \Sigma^*$ and L/s denote the postlanguage of L after s, i.e.,$L/s = \{t \in \Sigma^* \mid st \in L\}$. We also define the *projection* function $P : \Sigma^* \rightarrow \Sigma_o^*$ in the usual manner [19]:

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(\sigma) &= \sigma \quad \text{if } \sigma \in \Sigma_o \\ P(\sigma) &= \epsilon \quad \text{if } \sigma \in \Sigma_{uo} \\ P(s\sigma) &= P(s)\ P(\sigma) \quad s \in \Sigma^*, \quad \sigma \in \Sigma. \end{aligned} \tag{5.1}$$

P is the projection onto the set of observable events and simply 'erases' the unobservable events in a trace. The inverse projection operator P^{-1} is defined as usual to be

$$P^{-1}(y) = \{s \in \Sigma^* : P(s) = y\}. \tag{5.2}$$

For all $\Sigma' \subseteq \Sigma$ we define

$$\Psi_L(\Sigma') = \{s\sigma \in L : \sigma \in \Sigma'\}, \tag{5.3}$$

i.e., $\Psi_L(\Sigma')$ denotes the set of all traces of L that end in an event belonging to the set Σ'. The notation $\sigma \in s$ denotes that σ is an event in the trace s.

We present next the definition of FH-diagnosability. It is based on the definition of diagnosability in [20] which is stated next for reference.

Definition 1 *Let M_F be a fault hypothesis FSM, Π_F a partition on Σ_F, P the projection function on the set of observable events, and M_P the plant FSM. M_P is FH-diagnosable w.r.t fault hypothesis M_F, projection P, and partition Π_F, iff $L(M_P \otimes M_F)$ is diagnosable w.r.t. the projection P and the partition Π_F on Σ_F.*

Definition 2 *A prefix-closed and live language L is* **diagnosable** *with respect to the projection P and the partition Π_F on Σ_F if the following holds:*

$$(\forall i \in \Pi_F)\ (\exists n_i \in I\!N)\ (\forall s \in \Psi(\Sigma_{Fi}))\ (\forall t \in L/s) \qquad [\ ||t|| \geq n_i \Rightarrow D\ ]$$

where the diagnosability condition D is:

$$\omega \in P^{-1}[P(st)] \cap L \Rightarrow \Sigma_{Fi} \in \omega\ .$$

The definition of diagnosability means the following. A system is diagnosable if it cannot have two behaviors of arbitrarily long length that generate the same sequence of observations, though one behavior contains a failure of type i and the other does not contain a failure of type i. $FH-$diagnosability simply requires the same condition to be true for the system restricted by the fault hypothesis M_F. More specifically, let s be any trace generated by the system that ends in a failure event from the set Σ_{Fi} and let t be any sufficiently long continuation of s. Condition D then requires that every trace belonging to the language that produces the same record of observable events as the trace st should contain in it a failure event from the set Σ_{Fi}. This implies that along every continuation t of s one can detect the occurrence of a failure of the type F_i with a finite delay; specifically in at most n_i transitions of the system after s. Alternately speaking, diagnosability requires that every failure event leads to observations distinct enough to enable unique identification of the failure type within finitely many transitions.

5.3 Plant Modelling

Modeling the causal relationships between faults, control modes, and residues for a system with multiple components and control modes can result in models with thousands of states. Such models are manageable only if they can be constructed in a modular fashion, i.e., by composing small models. A compositional method for construction of the plant model also makes it relatively easier to regenerate a new plant model if there are some changes in the constituent components. This section describes our plant modeling method for the automated vehicle diagnostics application. We start with a discussion of the modeling in a fixed control mode and then discuss the case where the residue values may be dependent on the control mode.

We assume that the modeling process starts with a set of residues R, set of fault event Σ_F and a relation π on $R \times \Sigma_F$. Thus if (r, α) is an element of the relation it signifies that the fault event α affects the value of the residue. We set $\Sigma_{fr} = \{\sigma \in \Sigma_F : (r, \sigma) \in \pi\}$.

Recall the map γ_r, defined in the introduction, that maps the real values of residue r into a residue event set Σ_r. It is assumed that $\gamma_r(\mathbb{R}) = \{(r, L), (r, H)\}$, i.e., the residue is either high or low. Furthermore we use the following qualitative guidelines on the causal relationship between residues and the related faults.

1. If none of the fault events in Σ_r have occurred then only the (r, L) event will be observed.
2. If one and only one of the fault events in Σ_r have occurred then only the (r, H) event will be observed.
3. If two or more of the fault events related to the residue r have occurred then either the (r, L) or (r, H) events may be observed.

The last assumption represents the fact that the sensors and observers are such that the value of the residue r in the presence of multiple faults is uncertain.

Using these guidelines we can derive a finite state machine for a single residue r. For all $\sigma \in \Sigma_{fr}$ consider the FSM M_σ defined previously. Define the FSM $M'_r = \otimes_{\sigma \in \Sigma_{fr}} M_\sigma = \langle \Sigma_{fr}, Q'_r, 0'_r, \delta'_r \rangle$. The FSM $M_r = \langle \Sigma_{fr} \uplus \Sigma_r, Q_r, 0_r, \delta_r \rangle$, is defined by $Q_r = Q'_r, 0_r = 0'_r$ and the transition function is

as follows.

$$\begin{array}{lcl}
\delta_r(\sigma,.) & = & \delta'_r(\sigma,.), \text{ for } \sigma \in \Sigma_{fr}, \\
\delta_r((r,L),0_r) & = & 0_r, \\
\delta_r((r,H),\delta_r(\sigma,0_r)) & = & \delta_r(\sigma,0_r), \text{ for } \sigma \in \Sigma_{fr}, \\
\delta_r(\sigma',x) & = & x, \text{ for } \sigma' \in \Sigma_r, \text{ and } \{x \in Q_r : x \neq 0_r, x \neq \delta_r(\sigma,0_r), \sigma \in \Sigma_{fr}\},
\end{array}$$

and undefined otherwise. The set of all fault and residue event traces that may be generated by the plant is the language $L(\otimes_{r\in\mathcal{R}} M_r)$. Let this FSM be denoted by M'_P.

The FSM M'_P allows failure events to occur one after the other without any observable events being generated in between. This can make diagnosis very difficult. We introduce one more FSM that represents the assumption that after each failure event the relevant residue events will be observed before the next failure event occurs. This FSM is denoted by M_s and constructed as follows. The construction is complicated but modular.

Define the FSM $M_L = \otimes_{r\in\mathcal{R}} M_{(r,L)}$. Define

$$M_{rs} = \langle \Sigma_r, \{0_{rs}, 1_{rs}\}, 0_{rs}, \delta_{rs}\rangle, M'_{rs} = \langle \Sigma_r, \{0'_{rs}, 1'_{rs}\}, 0'_{rs}, \delta'_{rs}\rangle,$$

where

$$\begin{array}{rcl}
\delta_{rs}(\sigma,0_{rs}) & = & 1_{rs}, \sigma \in \Sigma_r, \\
\delta'_{rs}(\sigma 0'_{rs}) & = & 1'_{rs}, \sigma \in \Sigma_r,
\end{array}$$

and undefined otherwise. Next define $M_{Rs} = \otimes_{r\in\mathcal{R}} M_{rs}$ and $M'_{Rs} = \otimes_{r\in\mathcal{R}} M'_{rs}$. Finally define the FSM's M_{fs} and M'_{fs}. The definitions are

$$M_{fs} = \langle \Sigma_F, \{0_{fs}, 1_{fs}\}, 0_{fs}, \delta_{fs}\rangle, M'_{fs} = \langle \Sigma_F, \{0'_{fs}, 1'_{fs}\}, 0'_{fs}, \delta'_{fs}\rangle$$

where $\delta_{fs}(\sigma,0_{fs}) = 1_{fs}$ for $\sigma \in \Sigma_F$, undefined otherwise, and δ'_{fs} is defined similarly. Note that the FSM's $M_L, M_{Rs}, M'_{Rs}, M_{fs}$ and M'_{fs} are acyclic. Moreover every path in each of these FSM's ends at one unique state with respect to which the entire FSM is *co-accessible* ([19]). We denote these unique final states by $x_L, x_{Rs}, x'_{Rs}, x_{fs}, x'_{fs}$ respectively. Note also that the state sets of these FSM's are all disjoint.

The machine M_s will be constructed by first building a non-deterministic FSM A with ϵ transitions in the manner described in section 2.4 of [10]. We let

$$\Sigma_A = \Sigma_R \cup \Sigma_F \cup \Sigma_{con}, Q_A = Q_L \cup Q_{Rs} \cup Q'_{Rs} \cup Q_{fs} \cup Q'_{fs}, 0_A = 0_L.$$

The transition function is defined to be equal to $\delta_L, \delta_{Rs}, \delta'_{Rs}, \delta_{fs}, \delta'_{fs}$ on the respective state sets and in addition is augmented with the following $\epsilon-$transitions.

$$\begin{array}{rcl}
\delta_A(\epsilon,x_L) & = & 0_L \\
\delta_A(\epsilon,0_L) & = & 0_{fs} \\
\delta_A(\epsilon,x_{fs}) & = & 0_{Rs} \\
\delta_A(\epsilon,x_{Rs}) & = & 0'_{Rs} \\
\delta_A(\epsilon,x'_{Rs}) & = & 0'_{Rs} \\
\delta_A(\epsilon,0'_{Rs}) & = & 0'_{fs} \\
\delta_A(\epsilon,x'_{fs}) & = & 0_{Rs}
\end{array}$$

M_s is defined to be the canonical ([10]) deterministic FSM generating the same language as A.

We may choose $M_P = M'_P$ or $M_P = M'_P \otimes M_s$.

5.4 Diagnoser Construction and Diagnosability Verification

This section defines the diagnoser and presents the necessary and sufficient conditions for diagnosability. Verification is accomplished by confirming that the FSM $M_P \otimes M_F$ satisfies the sufficient conditions for partition Π_F and the partition $\Sigma = \Sigma_o \uplus \Sigma_{uo}$. The development is similar to [20].

We let $M_P = \langle \Sigma_F \cup \Sigma_{con} \cup \Sigma_R, Q_P, 0_P, \delta_P \rangle$. The following additional notation is used in this section. Let

$$X_o = \{0_P\} \cup \{x \in Q_P : x \text{ has an observable event into it}\}.$$

$L(G,x)$ denotes the set of all traces in FSM G that originate from state x. We define

$$\begin{aligned} L_o(M,x) &= \{s \in L(M,x) : s = u\sigma, \quad u \in \Sigma_{uo}^* \quad , \sigma \in \Sigma_o\} \\ L_\sigma(M,x) &= \{s \in L_o(M,x) : s = u\sigma, u \in L_o(M,x)\}. \end{aligned}$$

to be the set all traces that originate from state x and end at the first observable event, and the set of all traces in $L_o(M,x)$ that end with the particular observable event σ, respectively. The set of *failure labels* is denoted $\Delta_f = \{F_1, F_2, ...F_m\}$ where $|\Pi_F| = m$ and the complete set of possible labels is

$$\Delta = \{N\} \ \cup \ 2^{\{\Delta_f\}}. \tag{5.4}$$

Here N is to be interpreted as meaning "normal", and $F_i, \ i \in \{1, m\}$ as meaning that a failure of the type F_i has occurred. We also define the state set

$$Q_o = 2^{X_o \times \Delta}.$$

The diagnoser for M_P is the FSM

$$\langle M_d = (\Sigma_o, Q_d, 0_d, \delta_d \rangle \tag{5.5}$$

where Q_d, Σ_o, δ_d and 0_d have the usual interpretation. The initial state of the diagnoser 0_d is defined to be $\{(0_P, \{N\})\}$. The transition function δ_d of the diagnoser is constructed as explained below. The state space Q_d is the resulting subset of Q_o composed of the states of the diagnoser that are reachable from 0_d under δ_d. Since the state space Q_d of the diagnoser is a subset of Q_o, a state q_d of G_d is of the form

$$q_d = \{(x_1, \ell_1), \ldots, (x_n, \ell_n)\}$$

where $x_i \in X_o$ and $\ell_i \in \Delta$, i.e., ℓ_i is of the form $\ell_i = \{N\}$, $\ell_i = \{F_{i_1} F_{i_2}, \ldots, F_{i_k}\}$, or $\ell_i = \{F_{i_1} F_{i_2}, \ldots, F_{i_k}\}$ where in the last two cases $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$.

An *observer* for M_P (see [14]) gives estimates of the current state of the system after the occurrence of every observable event. The diagnoser M_d can be thought of as an *extended observer* where we append to every state estimate a label of the form mentioned above. The labels attached to the state estimates carry failure information and failures are diagnosed by checking these labels. We assume the plant M_P is normal to start with, hence we define $q_0 = \{0_P, \{N\})\}$.

Before defining the transition function δ_d of the diagnoser, we define the two functions called the Label Propagation function LP, and the Range function R.

Definition 3 *The* **Label Propagation Function** $LP : X_o \times \Delta \times \Sigma^* \to \Delta$.
Given $x \in X_o$, $\ell \in \Delta$, and $s \in L_o(M_P, x)$, LP propagates the label ℓ over s, starting from x and following the dynamics of M_P, i.e., according to $L(M_P, x)$. It is defined as follows:

$$LP(x, \ell, s) = \begin{cases} \{N\} \text{ if } \ell = \{N\} \ \wedge \ \forall i, \overline{s} \cap \Psi(\Sigma_{Fi}) = \emptyset \\ \{F_i : F_i \in \ell \ \vee \ \overline{s} \cap \Psi(\Sigma_{Fi}) \neq \emptyset\} \text{ otherwise.} \end{cases}$$

Definition 4 *The* **Range Function** $R : Q_o \times \Sigma_o \rightarrow Q_o$ *is defined as follows:*

$$R(q,\sigma) = \bigcup_{(x,\ell)\in q} \quad \bigcup_{s\in L_\sigma(M_P,x)} (\delta(x,s), LP(x,\ell,s)).$$

The transition function of the diagnoser $\delta_d : Q_o \times \Sigma_o \rightarrow Q_o$ is defined by

$$\delta_d(\sigma,q) \quad = \quad \begin{cases} R(q,\sigma), \text{ if } R(q,\sigma) \neq \emptyset \\ \text{undefined otherwise} \end{cases}$$

The verification of diagnosability is done by checking for a specific type of cycle in the diagnoser and plant model. We introduce the concepts of an F_i-*uncertain state* and a F_i-*indeterminate cycle.* F_i-uncertain states are defined as follows.

Definition 5 *1. A state $q \in Q_d$ is said to be F_i-***certain** *if $\forall (x,\ell) \in q, \quad F_i \in \ell$.*

*2. A state $q \in Q_d$ is said to be F_i-***uncertain** *if $\exists (x,\ell),(y,\ell') \in q$, such that $F_i \in \ell$ and $F_i \notin \ell'$.*

The defintion of an F_i-indeterminate cycle is as follows. A cycle in an FSM M is a sequence of states $x_1, x_2, \ldots, x_n \in X$ such that $\exists s \in L(M,x_1)$ such that $s = \sigma_1\sigma_2\ldots\sigma_n$ and $\delta(x_l,\sigma_l) = x_{(l+1) \ mod \ n}, \ l = 1,2,\ldots,n$.

Before stating the next defintion we introduce the FSM $M'_P = \langle \Sigma, X_o, 0_P, \delta'_P \rangle$. M'_P is in general a non-deterministic FSM that generates the observable projection of $L(M_P)$. The transition function δ'_P is defined as follows.

$$\delta'_P(\sigma,x) = x' \text{ for } \delta_P(s,x) = x', s \in L_\sigma(M_P,x)$$

Definition 6 *A set of F_i-uncertain states $q_1, q_2, \ldots, q_n \in Q_d$ is said to form an F_i-***indeterminate** *cycle if*

1. *$q_1, q_2, \ldots, q_n$ form a cycle in M_d with $\delta_d(q_l,\sigma_l) = q_{l+1}, \ l = 1,\ldots,n-1, \ \delta_d(q_n,\sigma_n) = q_1$, where $\sigma_l \in \Sigma_o, \ l = 1,\ldots,n$; and*
2. *$\exists (x_l^k, \ell_l^k), (y_l^r, \tilde{\ell}_l^r) \in q_l, \ l = 1,\ldots,n, \ k = 1,\ldots,m$, and $r = 1,\ldots,m'$ such that*
   (a) *$F_i \in \ell_l^k, \ F_i \notin \tilde{\ell}_l^r$ for all l,k, and r;*
   (b) *The sequences of states $\{x_l^k\}, \ l = 1,\ldots,n, \ k = 1,\ldots,m$ and $\{y_l^r\}, \ l = 1,\ldots,n, \ r = 1,\ldots,m'$ form cycles in M'_P with*

$$\begin{aligned} (x_l^k, \sigma_l, x_{(l+1)}^k) &\in \delta_{M'_P}, \ l = 1,\ldots,n-1, \ k = 1,\ldots,m, \\ (x_n^k, \sigma_n, x_1^{k+1}) &\in \delta_{M'_P}, \ k = 1,\ldots,m-1, \text{ and} \\ (x_n^m, \sigma_n, x_1^1) &\in \delta_{M'_P}, \end{aligned}$$

and

$$\begin{aligned} (y_l^r, \sigma_l, y_{(l+1)}^r) &\in \delta_{M'_P}, \ l = 1,\ldots,n-1, \ r = 1,\ldots,m,' \\ (y_n^r, \sigma_n, y_1^{r+1}) &\in \delta_{M'_P}, \ r = 1,\ldots,m'-1, \text{ and} \\ (y_n^{m'}, \sigma_n, y_1^1) &\in \delta_{M'_P}. \end{aligned}$$

In other words, an F_i-indeterminate cycle in G_d is a cycle composed exclusively of F_i-uncertain states for which there exist:

1. a corresponding cycle (of observable events) in M'_P involving only states that carry F_i in their labels in the cycle in M_d (this is the sequence $\{x_l^k\}$) and
2. a corresponding cycle (of observable events) in M'_P involving only states that do *not* carry F_i in their labels in the cycle in G_d (this is the sequence $\{y_l^r\}$).

Observe that in the above definition, m and m' denote the number of times the cycle $q_1, q_2, \ldots, q_n$ in G_d is completed before the cycle in M'_P is completed, i.e, nm and nm' are the cycle lengths in M'_P for $\{x_l^k\}$ and $\{y_l^r\}$ respectively.

An Fi-indeterminate cycle in M_d indicates the presence in L of two traces s_1 and s_2 of arbitrarily long length, such that they both have the same observable projection, and s_1 contains a failure event from the set Σ_{fi} while s_2 does not. The notion of an F_i-indeterminate cycle is the most crucial element in the development of necessary and sufficient conditions for diagnosability. The theorem, quoted from [20], is as follows.

Theorem 1 *A language $L(M_P \otimes M_F)$ is diagnosable if and only if there are no F_i-indeterminate cycles in its diagnoser G_d, for all failure types in the partition Π_F.*

Thus diagnosability and FH-diagnosability are verified by checking for the existence of such F_i-indeterminate cycles. We have constructed algorithms to check these conditions. Their use is demonstrated in section 5.5.

5.5 Failure Diagnosis of an Automated Vehicle

In this section we illustrate the applicability of the previously developed theory. The continuous domain, sensor, observer, and residue design is summarized in Tables 5.1 and 5.2 included from [18]. Table 5.1 summarizes 18 different signals to be used in the fault detection and identification scheme. Some of the signals are directly measured while others are estimates obtained from the observers discussed in [18]. Table 5.2 summarizes 10 different residues calculated using combinations of the signals from the previous table.

5.5.1 The Two Residue Illustration

In this section we illustrate the methods by considering residue 1 and 7 of table 5.2. The residues are associated with failures in the wheel speed sensor, engine speed sensor and throttle actuator components. We verify diagnosability under the single failure hypothesis and the partition $\{F\_WSS\}, \{F\_ESS\}, \{F\_TA\}$. The diagnoser is also constructed. The component models are illustrated in figure 5.2. The third model in the figure is the fault hypothesis FSM M_F representing the single failure assumption.

The synchronous composition of $M_1 \otimes M_7 \otimes M_f$ is shown in figure 5.4. The machine M_s representing the assumption that all residue values are read after a failure is shown in figure 5.3. The final system to be verified for diagnosability ($M_1 \bigotimes M_7 \bigotimes M_f \bigotimes M_s$) is shown in figure 5.5. The diagnoser M_d is shown in figure 5.6. The diagnostic state for the engine speed sensor failure

is $(2,2,2,F1)$ (residue sequence R1H, R7H), that for the wheel speed sensor failure is $(3,1,2,F1)$ (residue sequence R1H, R7L), and that for the the throttle actuator failure is $(1,3,2,F1)$ (residue sequence R1L, R7H).

5.5.2 The Dependence of Residue on Control

Residue 3 is generated by using intra-platoon communication. Such communication is available when the vehicle is in platoon follower mode but not in the platoon leader mode. Thus the value of the residue in platoon leader mode is indeterminate. Therefore in all states in which the vehicle is in leader mode the residue values are either R3H or R3L, while in the follower mode the residue values are always one or the other. The residue model for single faults is shown in figure 5.7. The system is not diagnosable by looking at a single residue.

5.5.3 The Composite System

We next applied the design methods to all sensors, actuators and their associated residues with and without communications faults. Hence a machine, M_i, was constructed for each of the ten residues, a failure events machine, M_f, consisting of two states and ten transitions and finally the machine M_s consisting of 2047 states. M_s was constructed algorithmically by the methods described in section 5.3. In the case when it was assumed that there are no faults in the communication system, the system is determined to be diagnosable, i.e. no indeterminate cycles were found. The full set of residue model files is attached.

5.6 Summary

We have presented a systematic procedure for creating DEM's based on residues generated in the continuous domain. The method is modular in the sense that the model of the integrated system is built by composing unit models using mathematically well-defined operators. We provide a formal way of defining a diagnostic requirement and thereafter proving whether a given diagnostic scheme satisfies the requirement. We have also provided a means of generating a FSM specification of the residue processing software.

We have provided two examples to intuitively explain the working of the methods. We have also established how the methods may be applied to the design of diagnostics for the longitudinal control system of a fully automated vehicle.

We are interested in a more intensive investigation on the detection of communication faults. We are also interested in the process of translating the FSM specification of the diagnoser to real-time diagnostic code.

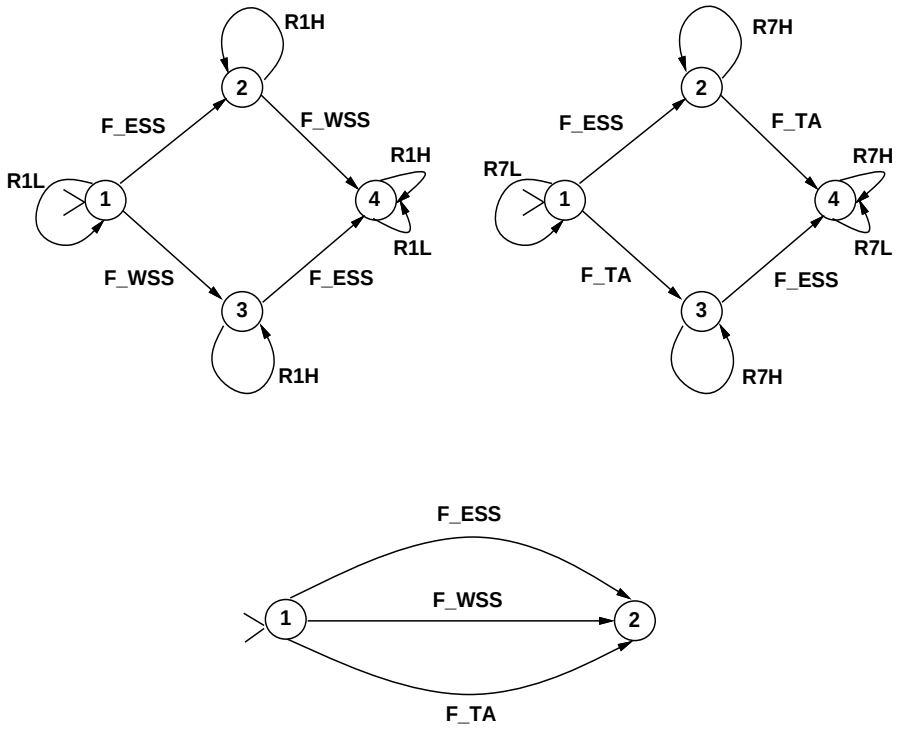


Figure 5.2: Component models M_1, M_7 and M_f

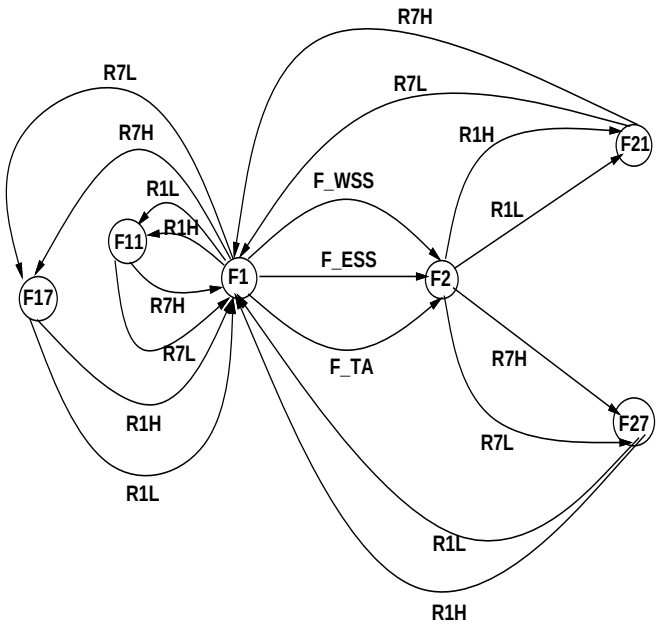


Figure 5.3: M_s for residues 1 and 7 only

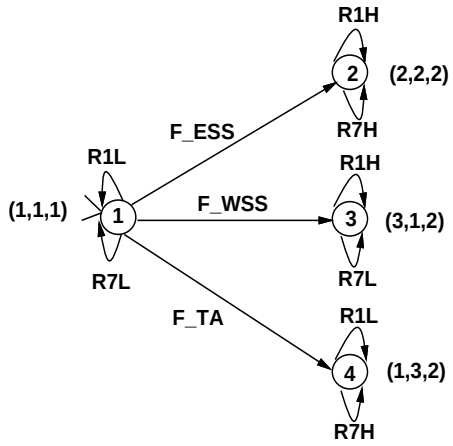


Figure 5.4: Synchronous composition $M_1 \bigotimes M_7 \bigotimes M_f$

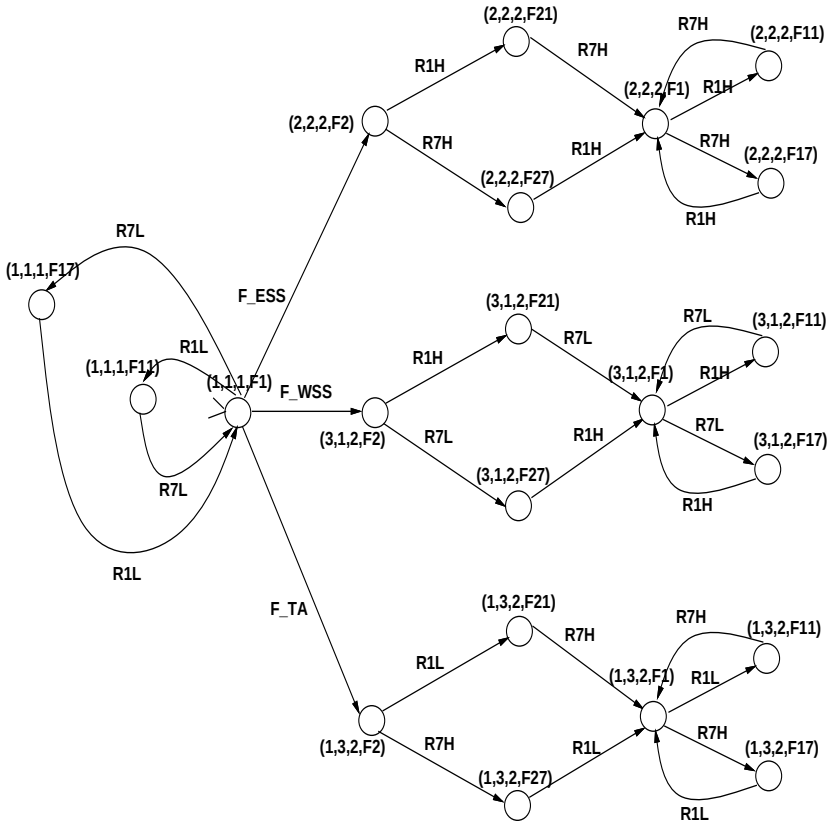


Figure 5.5: Synchronous composition $M_1 \bigotimes M_7 \bigotimes M_f \bigotimes M_s$

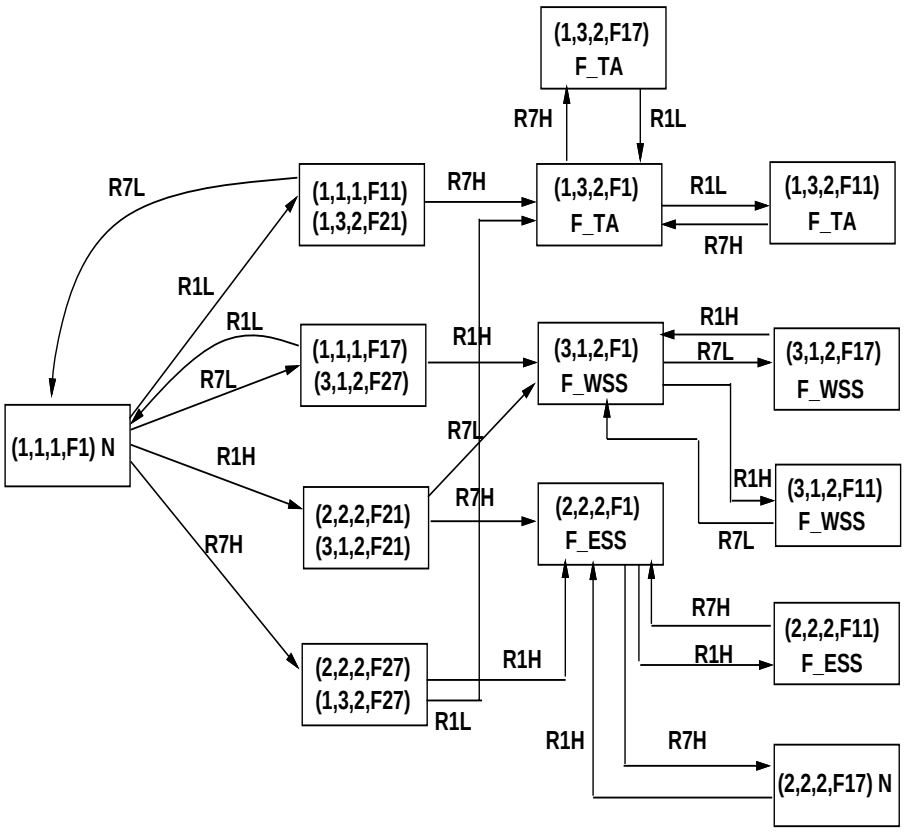


Figure 5.6: Diagnoser M_d for residues 1 and 7 only

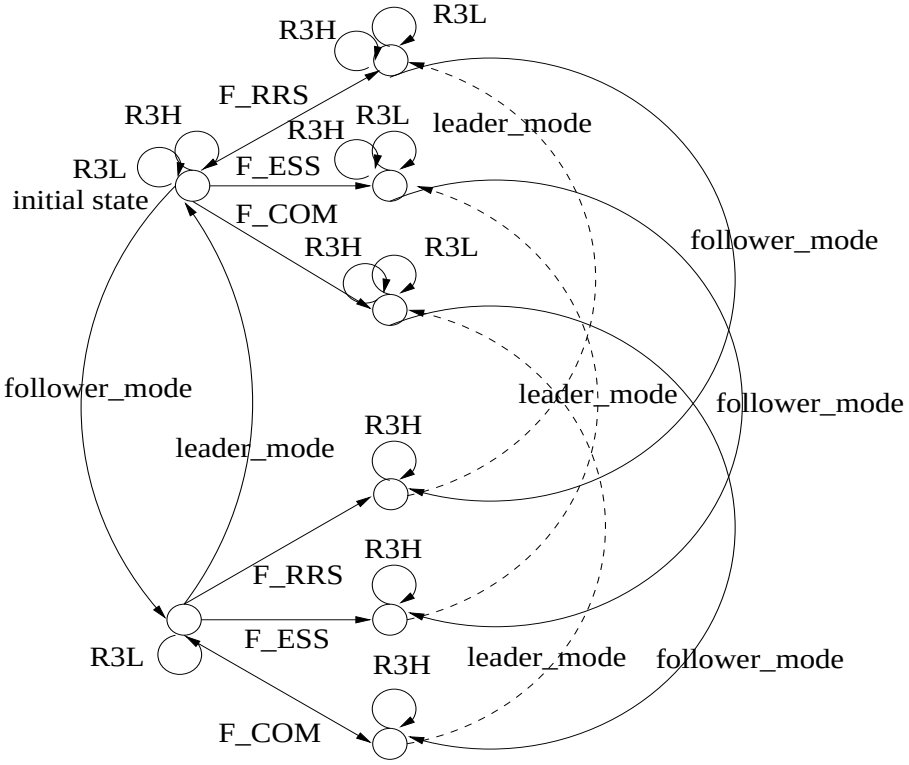


Figure 5.7: The dependence of residue 3 on control

SIGNAL	DESCRIPTION	SENSOR/OBSERVER
$z_1 = h\omega_w$	vehicle speed	wheel speed sensor
$z_2 = Rh\omega_e$	engine speed	engine speed and gear ratio sensors
$z_3 = x_{i-1} - x_i$	distance to preceding car (range)	radar range signal
$z_4 = v_{i-1} - v_i$	relative velocity of preceding	radar range rate signal
$z_5 = v_{i-1}$	velocity of preceding car	communication
$z_6 = \omega_e$	engine speed	engine speed sensor
$z_7 = \dot{m}_a$	mass flow rate of air in manifold	mass flow rate sensor
$z_8 = a_i$	acceleration	accelerometer
$z_9 = a_{i-1}$	acceleration of preceding car	communication
$z_{10} = \hat{\omega}_e$	estimated engine speed	observer of eqns. (11)-(12) speed sensor
$z_{11} = \hat{\dot{m}}_a$	estimated flow rate in manifold	observer of eqns. (11)-(12) speed sensor
$z_{12} = \alpha_c$	commanded throttle angle	calculated by controller using eqn. (6)
$z_{13} = \alpha$	throttle angle	throttle angle sensor
$z_{14} = T_{br\_com}$	commanded brake torque	calculated by controller using eqn.(6)
$z_{15} = \hat{\delta}_i$	estimated distance to preceding car	observer of eqn. (9)
$z_{16} = \hat{v}_i$	estimated velocity	observer of eqn. (15)
$z_{17} = a_{syn}$	synthetic acceleration	calculated by controller using eqn. (6)
$z_{18} = \omega_{e\_br}$	estimated engine speed during braking	observer of eqn. (13)

Table 5.1: Signals for automated vehicle longitudinal control system diagnostics

RESIDUES	SENSORS/ACTUATORS INVOLVED
$R_1 = z_1 - z_2$	wheel speed sensor engine speed sensor
$R_2 = z_5 - z_4 - z_1$	radar range rate sensor wheel speed sensor communication
$R_3 = z_5 - z_4 - z_1$	radar range rate sensor engine speed sensor communication
$R_4 = z_3 - z_{15}$	radar range sensor wheel speed sensor magnetometer communication
$R_5 = z_{17} - z_8$	accelerometer
$R_6 = z_{16} - z_1$	accelerometer markers wheel speed sensor
$R_7 = z_{10} - z_2$	throttle actuator engine speed
$R_8 = z_{12} - z_{13}$	throttle angle sensor throttle actuator
$R_9 = z_{11} - z_7$	mass flow rate sensor throttle actuator
$R_{10} = z_{18} - z_2$	brake actuator engine speed

Table 5.2: Residues and the associated components

Chapter 6

Future Work

During the following year we will focus on the formalisation of the architecture integrating diagnostics and control at the regulation and coordination layers of the vehicle. The formalism should support a deterministic and probabilistic theory of verification. This formalization will serve to precisely identify the assumptions under which the diagnostic designs developed by the project are expected to function satisfactorily.

The fault diagnostic system has been proven through simulation and limited experimental tests to be extremely effective at detecting faults in the longitudinal control system. However, further experimental testing will be performed to validate the entire fault diagnostic system on the platooning vehicles at PATH.

One limitation of the current fault diagnostic system is the neglection of the lateral control sensors and steering actuator. For the development of a complete fault diagnostic system for automated vehicles, this limiation needs to be addressed, and is therefore the subject of further research. Furthermore, the identification logic developed in chapter 3 relies on the decision of a residual signal being "high" or "low". These subjective ratings will need to be quantified for each residual signal, based on knowledge of sensor noise characteristics and modeling errors. In addition, an algorithm will be required for experimental implementation of the decision logic and the complete fault diagnostic system.

The project will continue to develop the interfaces between the fault detection and handling and its implementation in the SHIFT hybrid systems programming language. In particular we will concentrate on the following items.

1. Complete the implementation in SHIFT of the coordination layer supervisor.
2. Extend the design of the coordination layer supervisor to allow it to synchronize with more than one capability structure.
3. Complete the design for the performance structure in the fault tolerant AHS.
4. Implement a performance structure design in SHIFT.
5. Perform extensive tests of the capability and performance structures.
6. Integrate the coordination layer supervisor with the capability and performance structures.
7. Perform simulations of the complete fault handling systems.

Bibliography

[1] L. Alvarez and R. Horowitz. Safe platooning in automated highway systems. Submitted as a PATH research report, 1997.

[2] D. Cho and J.K. Hedrick. Automotive powertrain modeling for control. In *ASME Transactions on Dynamic Systems, Measurement and Control*, volume 111, December 1989.

[3] R.K. Douglas and D.L. et al Speyer. Fault detection and identification with application to advanced vehicle control systems. Ucb-its-prr-95-26, California PATH Research Report, 1995.

[4] V. Garg. *Fault Detection in Nonlinear Systems.* PhD thesis, Department of Mechanical Engineering, University of California at Berkeley, 1995.

[5] V. Garg. *Fault Detection in Nonlinear Systems : An Application to Automated Highway Systems.* PhD thesis, Department of Mechanical Engineering, University of California at Berkeley, 1995.

[6] V. Garg and J.K. Hedrick. Fault detection filters for a class of nonlinear systems. In *Proceedings of the 1995 American Control Conference*, pages 1647–1651, June 1995.

[7] D. Godbole, J. Lygeros, E. Singh, A. Deshpande, and A.E. Lindsey. Towards a Fault Tolerant AHS Design Part II: Design and Verification of Communication Protocols. PATH Technical Report UCB-ITS-PRR-96-15, Institute of Transportation Studies, University of California, Berkeley, 1996.

[8] Z. Har'El and R. P. Kurshan. *COSPAN User's Guide.* AT&T Bell Laboratories, Murray Hill, NJ, 1987.

[9] J.K. Hedrick, D. McMahon, V.K. Narendran, and D. Swaroop. Longitudinal vehicle controller design for ivhs systems. In *Proceedings of the 1991 American Control Conference*, volume 3, pages 3107–3112, June 1991.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[11] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya. Protocol Design for an Automated Highway System. *Discrete Event Dynamic Systems*, 2(1):4–16, 1994.

[12] John Lygeros. *Hierarchical, Hybrid Control of Large Scale Systems.* PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1996.

[13] John Lygeros, Datta Godbole, and Mireille Broucke. Towards a Fault Tolerant AHS Design Part I: Extended Architecture. PATH Technical Report UCB-ITS-PRR-96-14, Institute of Transportation Studies, University of California, Berkeley, 1996.

[14] C. M. Özveren and A. S. Willsky. Observability of discrete event dynamic systems. *IEEE Trans. Automatic Control*, 35(7):797–806, July 1990.

[15] S. Patwardhan and M. Tomizuka. Robust failure detection in lateral control for ivhs. In *Proceedings of the 1992 American Control Conference*, June 1992.

[16] Satyajit N. Patwardhan. Fault detection and tolerant control for lateral guidance of vehicles in automated highways. Tech. Rep. UCB-ITS-PRR-94-17, California PATH, University of California, Berkeley, 1994.

[17] R. Rajamani. Observer design for lipschitz nonlinear systems. to appear in IEEE Transactions on Automatic Control, 1997.

[18] R. Rajamani, J.K. Hedrick, and A. Howell. A complete fault diagnostic system for longitudinal control of automated vehicles. In *Proc. of Symposium of Advanced Automotive Technologies, 1997 ASME International Congress*, 1997.

[19] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, January 1987.

[20] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Automatic Control*, 40, September 1995.

[21] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Control Systems Technology*, 4, March 1996.

[22] D. Swaroop and et al Hedrick, J.K. A comparison of spacing and headway control laws for automatically controlled vehicles. *Vehicle System Dynamics Journal*, 23, November 1994.

[23] M. Tomizuka and J.K. Hedrick. Automated vehicle control for ivhs systems. In *Proceedings of the IFAC Conference*, Sydney, 1993.

[24] J.E. White and J.L. Speyer. Detection filter design :spectral theory and algorithms. In *IEEE Transactions on Automatic Control*, volume 32, pages 593–603, 1987.