Vladimir Roumenov Glavtchev
December 2009
Computer Engineering

EU Speed-limit Sign Detection Using a Graphics Processing Unit

Abstract

In this study we test the idea of using a graphics processing unit (GPU) as an embedded co-processor for real-time speed-limit sign detection. We implement a system that operates in real-time within the computational limits of contemporary embedded general-purpose processors. The input to the system is a set of grayscale videos recorded from a camera mounted in a vehicle. We implement a new technique to realize the radial symmetry transform efficiently using rendering primitives accelerated by graphics hardware to detect the location of speed-limit sign candidates. The system reaches up to 88% detection rate and runs at 33 frames per second on video sequences with VGA (640x480) resolution on an embedded system with an Intel Atom 230 @ 1.67 GHz and a Nvidia GeForce 9200M GS.

Associate Professor John D. Owens
Dissertation Committee Chair

EU Speed-limit Sign Detection Using a Graphics Processing Unit

By

VLADIMIR ROUMENOV GLAVTCHEV
B.S. (University of California at Davis) 2008

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTERS OF SCIENCE

in

Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

_____

_____

Committee in charge

2009

EU Speed-limit Sign Detection Using a Graphics Processing Unit

To my parents, who made this possible. And to my brother for his inspiration.

Acknowledgments

First, I would like to thank my advisor Professor John Owens. Without his excellent guidance and motivation, I might have never entered the field of general-purpose GPU computing research. John has vastly influenced my professional life and has enabled me to pursue my dreams as a better professional and a much stronger individual. A more brilliant example of a teacher would be difficult to find.

I would like to thank the reviewers of my thesis, Kent Wilken and Owen Carmichael, for their helpful feedback and comments along the way.

Next, I would to thank Jeff Ota, who initially proposed this project to our research group and later took me on as his intern for the summer of 2008 at the BMW Technology Office in Palo Alto, CA. For his leadership in two successful GPU conferences in the summers of 2008 and 2009. Also, for his professional and personal guidance ever since.

Many thanks go to Joe Stam and James Fung, who in a large part developed the ideas that led to the highly-efficient implementation of the Radial Symmetry Transform we present here. Without their expertise in computer vision and GPU architecture, most of the hardware acceleration in our implementation would not have been possible.

I would like to thank Ana Lucia Varbanescu for all of her advice on writing a Master's thesis. She was my inspiration for getting the writing process under way. I would like to thank her for the hours of sleep she lost reviewing and encouraging. Her caring ways kept me motivated throughout the editing process.

Finally, my biggest thanks goes out to my family, because without them I would not be where I am today. My parents' dedication for their two sons has continuously opened doors for my brother and I when times were difficult. My parents' work ethic has inspired me to never settle for less than my full potential.

Contents

List of Figures

List of Tables

Abstract

In this study we test the idea of using a graphics processing unit (GPU) as an embedded co-processor for real-time speed-limit sign detection. We implement a system that operates in real-time within the computational limits of contemporary embedded general-purpose processors. The input to the system is a set of grayscale videos recorded from a camera mounted in a vehicle. We implement a new technique to realize the radial symmetry transform efficiently using rendering primitives accelerated by graphics hardware to detect the location of speed-limit sign candidates. The system reaches up to 88% detection rate and runs at 33 frames per second on video sequences with VGA (640x480) resolution on an embedded system with an Intel Atom 230 @ 1.67 GHz and a Nvidia GeForce 9200M GS.

Associate Professor John D. Owens
Dissertation Committee Chair

Chapter 1

Introduction

Over the decades, there have been growing demands and expectations for safety features in vehicles. Systems reacting to the status of a vehicle have already found their way into vehicles and laws of many countries require car manufacturers to include most of these. Examples include seatbelts, airbags, anti-lock brake systems (ABS), dynamic traction control (DTC), and dynamic stability control (DSC). These features take into account the state of the vehicle and take action to either events that have already occurred (airbags inflate after a bumper impact) or try to prevent potential loss of control (anti-lock brakes activate when brakes are depressed rapidly). However, with electronics making their way into cars, more advanced features have also become appealing. Examples include active cruise control, intelligent headlight control, and object detection. These features are "observant" of the environment around the vehicle and react to alert the driver (speed-limit sign detection alerting the driver of the speed limit) or prevent the vehicle from possible collision (active cruise control releasing the gas to avoid a collision with a decelerating vehicle in front).

Observing the environment around a vehicle often requires advanced sensing systems such as cameras, radar, infrared sensors, and others. The amount of data these systems gather is significant and extracting information important to the driver is often computationally expensive. Processing units with high computational capabilities are required for most, if not all, of the features that rely on environmental data collected using

advanced sensing systems. Due to the exponential growth of the semiconductor industry, it is becoming more feasible to process the input data and alert the driver with intelligent information.

1.1 Limitations of Today's In-vehicle Systems

In today's vehicles, we see a large number of features available such as rear and side-view cameras, parking assist, night-vision pedestrian detection, rain sensors, lane departure warning, approaching side vehicle alarm, high beam assist, active cruise control, and speed limit sign detection. Car manufacturers offer these as additional feature packages that can be purchased separately or as all-inclusive "'technology packages."' Each of these driver assist options is usually implemented with a DSP or an application-specific processor. These processors are usually very low-end and meet the minimum performance requirements so as to meet their energy and cost-effectiveness constraints.

The power usage and minimized cost of these features can seem efficient when analyzed independently. However, with these features all present in a vehicle, their cost and power usage can be non-trivial. More importantly, the average use-case must be examined when these systems are active simultaneously. Many of them are used exclusively or only in conjunction with one or two of the others. Even at the worst usage case when most systems are active, there are some that are idle or powered down (i.e. rearview camera and active cruise control would never be used together). This leaves a lot of unused processing capability, which runs directly against the fundamentals of embedded systems where functional unit usage is usually maximized.

1.2 Motivation

Our goal is to present a proof-of-concept: the usage of general-purpose hardware for an embedded automotive application. We demonstrate this using the specific example of speed-limit sign recognition. Our focus is not to only demonstrate the effectiveness of the algorithm chosen, but mainly to show how the computation can be performed using a GPU

as a co-processor. The algorithm we choose (Radial Symmetry Transform), as we show in the Background section, has been widely used and its accuracy and robustness have been proven in the literature. We examine its performance and robustness in the Evaluation Studies section. Our main focus, however, is to demonstrate an implementation which uses hardware acceleration to achieve real-time performance on a programmable general-purpose graphics processing unit (GPGPU). Using a programmable processor is key in our study as it allows for reusability of hardware resources, freeing up host processor load, and allowing task-concurrency.

In this study, we examine several understudied (or at least under represented in the literature) topics related to speed-limit sign detection. Mainly, we implement a system which is able to perform speed-limit sign detection in real-time on embedded-grade hardware. Perhaps more importantly, the system is general-purpose and programmable which allows us to examine its scalability and offers the capability of implementing other automotive tasks such as 3D navigation. Also, we motivate the use of a camera as supplementary to a GPS system through example scenarios where only a camera-based system could provide the driver with up-to-date data. Examples include temporary signs in construction zones and variable digital signs on highways and in tunnels.

Chapter 2

Background

2.1 Overview

Generally, road sign recognition pipelines consist of four main stages:

Preprocessing This step simplifies or otherwise adjusts the input scene to increase the effectiveness of the next stage. Techniques include tresholding, normalization, contrast adjustment, edge detection, and others. Noise and artifacts can be removed in this stage and desired data can be extracted or amplified such as boosting a color channel of interest, extracting edges, or retrieving illumination data.

Detection of Regions of Interest In this step, points and/or regions of interest (ROIs) are selected from the input scene for further examination during the next stage. The selection can be achieved with methods such as the Hough Transform, Pavlidis's algorithm, and the Scale Invariant Feature Transform (SIFT). One or more detected ROIs can be extracted and used as input to the next stage.

Classification Classification can generally be either feature-based or template-based. The former focuses on characteristics of shapes such as size, orientation, edges, and corners to name a few. Feature-based examples include Neural Networks and Haar classifiers. Template-based classification works using predefined templates called classifiers, to which the input is compared to in order to find a match. Template-based classification is usually

done using cross-correlation in either the spatial domain or in the frequency domain. In the spatial domain, a target ROI is compared to a classifier template by convolving the target over the classifier and computing a figure of closeness of the match between the two images. In the frequency domain, the process is similar but the convolution is replaced by a multiplication. This provides significant cost-savings as convolutions are generally computationally expensive.

Temporal Integration Frame-by-frame detection and classification can lead to unstable results that vary significantly over an interval of several frames. An object of interest that matches a predefined classifier might return several false results over a span of frames in a moving-camera or moving-object scene. Temporal integration accumulates the results of classification over a specified length of time and increases the confidence with which an object is recognized or rejected. The integration can be intelligent using a complex algorithm such as Markov Chains or a Kalman Filter.

2.2 Detection Using Color Input

Traffic signs often include high-contrast color schemes such as red on white, yellow on black, black on white and others. An intuitive detection technique then is searching for a desired color scheme. Isolating a certain color channel or band of colors can simplify detection. Priese et al. [23] apply what they call Color Structure Code to detect traffic signs using a tree structure which separates different signs by color. Targeting EU speed limit signs specifically, Zheng [26] uses a color-segmenting technique developed by Ritter [25]. In addition, color information has been used during both detection and classification. Hatzidimos [11] uses color thresholding to find ROIs and also performs individual cross-correlations using each color channel (red, green, blue) in the classification stage. In general, using color for preprocessing or as an initial step to detection reduces the processing required in any following steps. However, normal color (RGB) information is highly sensitive to illumination and detection rates can vary depending on the time of day, weather, or headlight illumination. An alternative is to utilize HSI (hue, saturation, intensity) color information instead,

which according to Gajdamowicz [8] is more invariant to lighting. However, using color increases system cost as a more expensive camera, higher bandwidth, and larger memories are required in order to capture, transport, and store all channels of the color space.

2.3 Detection Using Grayscale Input

Detection using grayscale images inherently relies solely on the characteristics of the objects present in the scene. Characteristics such as edges, corners, and gradient angles can be used either individually or in groups to detect target shapes. Algorithms can search for these features individually across the input scene and piece together larger shapes (i.e. polygons can be detected using extracted corners and/or edges). Alternatively, the characteristics can be used in groups or as a whole in algorithms performing variations of matching ROIs with target templates.

Grayscale Over Color Not having color information restricts a given recognition technique to focus only on the characteristics that define the detection targets. However, that is in fact the main advantage of using grayscale images, argue Gavrila [10] and Barnes [1]. These characteristics are invariant to lighting conditions and fading of sign colors over time [20]. Garcia [9] claims that color-based detection is less precise due to the fact that the blue and red color spaces can overlap in signs which are mostly white (as is the case with speed-limit signs). Garcia is able to back that statement with detection rates over 97% using a shape-based approach on grayscale input. Using these findings, we pursue an approach based on grayscale input.

2.3.1 Template-based Detection

Detection using templated-based techniques is performed by "sliding" (usually a convolution in the spatial domain) a target template over the entire input scene and computing a value indicative of the similarity between the input and the target. Piccioli [22], for example, uses cross-correlation, which is the most commonly used technique. This method has the disadvantages of performing expensive convolutions and perhaps requiring

multiple passes. The convolutions can be avoided when working in the frequency domain, as proposed by Javidi [12]. However, the technique relies on the ability to quickly perform FFTs. Even then, multiple passes are needed to detect signs of different sizes, orientations, or different traffic signs in general (individual passes are needed to detect the speed limit signs "10," "20," "30," etc). A major disadvantage is that objects which are of shape or orientation not present in the target template set can be missed altogether. Therefore, target template sets need to be very comprehensive of variations.

2.3.2 Feature-based Detection

Algorithms in this category utilize prominent features in the input scene such as corners, line segments, and edges. Techniques of different complexities can be used to extract these features such as a simple Sobel filter, a more involved Canny edge detector, or an intelligent edge-chaining algorithm such as Pavlidis's [21] used by Piccioli [22]. These filters are commonly used as preprocessing steps to algorithms which then seek to perform polygonal approximation from the extracted features.

Hough and Reisfeld Transforms Traffic sign detection algorithms have naturally sought to take advantage of the geometry of the target signs. For rectangular and otherwise polygonal (diamond, octagon, etc) signs, algorithms targeting straight edges which form line segments have been used widely. Perhaps the most widely used algorithm has been the general Hough Transform (HT). Examples include detection of lane markers and diamond curve signs [13], as well as octagonal stop-signs [14]. For circular signs, the circular HT is the preferred algorithm and is an adapted version of the general transform. Both forms of the HT require the use of sine and cosine computations for each input pixel and the algorithms are considered computationally expensive and memory demanding [16]. In a more holistic approach, the General Symmetry Transform (GST) [24] measures the contribution of edge pixel pairs to a central point (a one-to-many approach), thus searching for edges of potential geometric shapes which exhibit horizontal or vertical symmetry. However, the computational cost of the GST is also high as each pixel searches a neighborhood of a certain width around it for edges contributing to a symmetry. Both HT and GST have inspired

simpler formulations which have replaced them in the recent literature and have produced implementations able to execute in real-time on mid-range machines such as desktops and laptops.

Radial Symmetry Loy and Zelinsky [18] introduced the Radial Symmetry Transform (RST) which inverts the symmetry-seeking technique of the GST to a many-to-one approach. Individual pixels vote for a common center of symmetry with their results accumulating in a common voting map. In order to motivate the technique, the authors show that the transform's computational complexity is much smaller than that of the GST, circular HT, and that of the lesser-used Symmetry from Local Phase algorithm. A notable characteristic of the radial symmetry transform is its parameterization. Varying the parameters of the algorithm allows detection of other shapes such as diamonds, squares, octagons, and in fact all n-sided symmetric polygons. In the first known application of the RST, Loy and Barnes [17] demonstrate the algorithm's robustness in an implementation of a driver-assistance system capable of recognizing octagonal stop signs, triangular warning signs, and diamond indicator signs.

A recent adaptation of the RST demonstrates a real-time implementation which specifically highlights the major benefit of the method: it is a pixel-based shape detector. This attribute makes the algorithm more robust under different lighting conditions than color-based approaches [2]. Another point which the authors show is that using a simple 3x3 Sobel filter in the preprocessing stage is sufficient to allow RST to handle up to 30% noise. Important to note are that both the preprocessing and detection stage are, in fact, per-pixel. This allows parallel architectures to accelerate the RST.

The implementations of the RST in the literature have shown that the algorithm reduces the computational requirements of the overall system in several ways. First, preprocessing consists simply of edge-extraction using a 3x3 Sobel filter with thresholding. Other methods require stronger edges and use more complex edge-chaining algorithms such as the Canny edge detector. Second, the ability of the transform to produce reliable results even with missing feature pixels eliminates the need for using highly-serial connected-component or flood-fill algorithms. Third, the radial symmetry transform is invariant to in-plane rota-

tions of the signs [17].

2.3.3 Focus of Recent Work

The goal of all previous work mentioned above and the goal of traffic sign recognition in general, has been the same: achieve high recognition rates in real-time. However, the large amounts of data that need to be processed and often complex algorithms require significant amounts of computational power. Even though automotive recognition systems are meant to run on embedded platforms in an automobile, recent literature has not paid sufficient attention to providing techniques which can run on such platforms. Previous studies which perform real-time speed-limit sign recognition mostly use dual-core desktop or laptop architectures [15, 20] as their running platforms. Even in one of the most recent studies, Barnes and Zelinsky [2] demonstrate a system which runs in real-time, but the implementation only does so on a server-grade Intel Xeon 3.40 GHz machine.

There have been solutions that more closely resemble embedded platforms. Examples include the use of specialized processors to accelerate certain stages. Escalera [5], for example uses a pipelined image processor to preprocess images in conjunction with a desktop PC for the main processing stages. Escalera's study would be a better match for the embedded space if it utilized a mobile or embedded host processor. However, those experiments were carried out in 1997 when embedded processors were simply not powerful enough. Coersmeier et al. [3] propose a real-time speed-limit sign recognition system which targets an embedded environment. In their study, the authors utilize four parallel ARM processors to test the speedup of a traffic sign recognition system for future mobile devices. Another hybrid approach is an existing commercial platform by MobileEye [19]. The system uses the EyeQ [6] processor which consists of an ARM core and four programmable ASIC cores called *vision computing engines* (VCEs). The VCEs offer hardware acceleration for common image-processing primitives such as image operations (scaling, warping), edge detection, convolutions, and histogram building.

Desktop and laptop platforms are expensive in terms of power, cost, efficiency and are unrealistic for an embedded system. Using ASIC solutions means long design cycles and very specific functionality: extension or reuse of hardware is very low in that case. We

believe that embedded designs should instead focus on using general purpose embedded-grade hardware that is widely available and can easily be used for other funcionalities.

2.3.4 Our Contribution

In order to meet real-time performance goals using the given constraints of architectures in the embedded space, a process which can be mapped efficiently onto the given hardware must be chosen. With this in mind, we focus on efficiency in several dimensions. First, algorithms which are highly-parallel should be used. Second, on-board and off-board memory transfers should be minimized - with data remaining onto a processing unit until all necessary processing per frame is completed by that unit. Third, the ratio of serial to parallel operations in the overall process should be such that it reflects the given hardware and, therefore, maximizes functional-unit throughput. Fourth, considering the smaller memory sizes available in the embedded environment as compared to desktop or laptop platforms, memory requirements of the algorithms must be kept within range. Fifth, due to the streaming nature of in-vehicle recognition, processing time per frame should not vary significantly with varying input. For example, processing an input frame with a cluttered city environment should take as much time to process as a country-road setting. We describe a system that tries to maximize performance using these metrics of efficiency.

The GPU is an obvious choice for a co-processor in many computer-vision detection, recognition, and augmented reality schemes. This is due in part to the exposure of the unified shaders through C-like languages. Before general purpose languages such as CUDA, Brook, and OpenCL became widely available, many experts in the computer-vision field were already harnessing the power of GPUs through shader languages like Cg and GLSL. Fung and Mann [7] demonstrated several algorithms ported to the GPU using Cg that are accelerated when compared to their CPU counter-parts or made possible in real-time due to the processing power available in GPUs. Canny filter, Hough and Chirplet transforms, and feature tracking are a few examples. Projects such as OpenVIDIA and GPU4VISION have developed a steadily growing community of programmers. Inspired from the success of GPU-based computer-vision projects, we choose the GPU as our detection co-processor of choice due to its highly-parallel architecture, large memories, and its programmability.

We implement the radial symmetry transform as presented by Loy and Barnes [17] using the Compute Unified Device Architecture (CUDA) in conjunction with OpenGL to achieve real-time performance on a graphics processor. We present a new technique for implementing the radial symmetry transform which takes advantage of the native rendering features of the graphics hardware to accelerate detection. Several stages of the detection process are optimized using techniques that enable hardware acceleration such as fast texture look-ups and polygon rendering. By targeting an embedded CPU-GPU platform, an understudied environment for speed-limit sign recognition, our study contributes to the field and helps filling the gap.

Chapter 3

Methodology

The input to our system is a grayscale VGA (640x480 pixels) image with a depth of eight bits per pixel.

3.1 Preprocessing

The radial symmetry transform operates using gradients of the edges present in the input scene. To obtain the edges, we apply a 3x3 Sobel edge detector to the entire input scene. Edges with gradient magnitudes under a specified treshold (explained in the Evaluation Studies section) are set to zero. Then, only the gradient angles of the non-zero gradient pixels are passed on to the next stage.

3.2 Shape Detection

Each element with a non-zero gradient angle casts a vote (increments an element on a cumulative voting image V) for a potential circle center a distance r away (where r is the radius of the circle which we are targeting for detection) in the direction of the gradient angle. Since we do not know the direction in which the centroid of the targeted circle is *a priori*, voting is performed both in the direction of the gradient and 180 degrees across from it. In order to take into account skewing due to viewing perspective or out-of-plane sign rotation, the vote is extended to a line of pixels (of width w) which are oriented

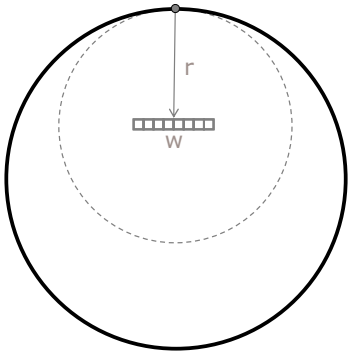


Figure 3.1: Votes cast by one edge pixel in the direction of its gradient at a distance of the targeted circle radius. Omitted in this diagram are the votes cast 180 degrees opposite of the gradient angle.

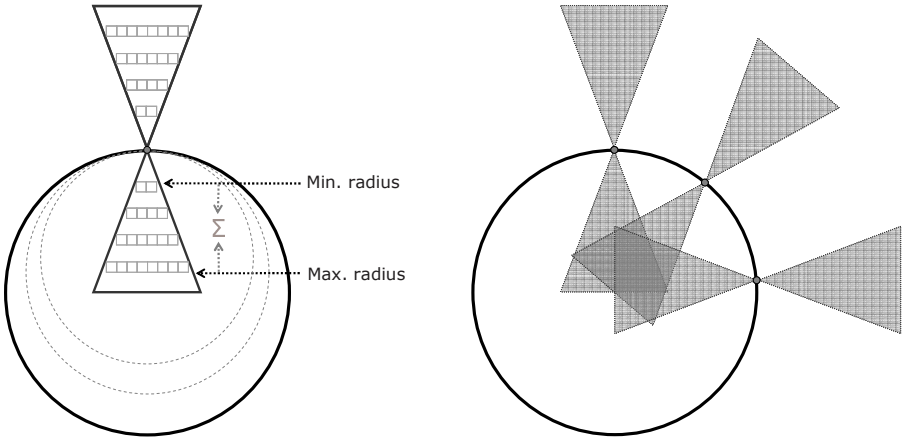
perpendicularly to the voting pixel's gradient angle. Figure 3.1 shows line-voting for one specific radius. The voting width is defined by Equation 3.1,

$$w = \text{round}(r\tan\frac{\pi}{n}) \tag{3.1}$$

where n represents the number of sides in the n-sided polygon. In this case, we target octagonal shapes ($n = 8$) as an approximation to circles. Since a speed limit sign will vary in radius depending on its distance away from the camera, we target a range of possible radii from R_{min} to R_{max}. Noticing that w increases with increasing r, the overall voting pattern for each pixel becomes triangular if $Rmin = 0$. This assumption simplifies the shape of the voting pattern and has performance implications as discussed in the Implementation section. Thus each pixel casts its votes in a triangular pattern on image V, as shown in Figure 3.2(a). The overall voting image is formed by summing the votes of all edge pixels, as Figure 3.2(b) shows. The points voted for are called affected pixels and are defined as:

$$p_{aff}(p) = p \pm \text{round}(r\text{g(p)}) \tag{3.2}$$

where g(p) is the unit gradient for pixel p. Figure 3.3 illustrates the steps of the detection algorithm and the possible speed-limit sign candidates extracted from the input image.



(a) Voting pattern for one edge pixel. (b) Overlapping voting triangles.

Figure 3.2: Triangular estimation of each edge pixel's voting pattern. Votes accumulate where the triangles overlap to create a combined voting image.

(a) Input image. (b) Edges detected in image.

(c) Radial symmetry voting image. (d) Detected speed-limit sign candidates.

Figure 3.3: Detection of circular speed-limit signs. (a) The original input image. (b) Edges found after Sobel with thresholding. (c) Voting intensity map with bright areas representing circular features. (d) Corresponding speed-limit sign candidates found using the detection algorithm.

3.3 Classification and Tracking

The classification method we utilize returns a confidence value for the best match for each candidate along with an estimated size and in-plane rotation. Temporal integration is implemented as a running tracking table of sign candidates across the last ten (the average number of frames in which a sign appears in our database) frames. A circular buffer is used for each sign classifier ("10", "20", "30", etc km/h) and is incremented with the classification's confidence of that sign along with scaling and orientation factors. Confidence values of signs appearing larger in size than their last appearance are multiplied by a scaling factor. This reduces the possibility of artifacts obtaining a high confidence level for several consecutive frames but with a decreasing size to contribute to the tracking results. The orientation factor is used similarly and it also prevents artifacts with changing rotations to affect tracking of real target signs.

3.4 Extension: Finding the Size

Due to the fact that we span the entire range of targeted radii using the triangular voting patterns, the size of the detected signs are not recovered, only the location. If needed, the size of the detected sign could be recovered using an incremental voting process using a set of voting images, where each targets a specific range of radii. The size is then recovered in two steps: a.) finding the maxima in a accumulated image S of all the results and b.) finding the voting image V_i (where V_i represents a voting image for a specific range of radii) which has a maximum with a sufficient magnitude and also appears in S.

3.5 Classification

In our study, we use a FFT template-matching method for verification of our detected. This classification scheme operates on a region of interest (ROI) in a scene. In order to achieve best performance, we perform detection that does not return the size of a detected sign but only an estimate of the center of the sign. The coordinates of the center are then used to segment the ROI from the image and apply the classification algorithm.

Chapter 4

Implementation

From our discussion thus far, it can be seen that both preprocessing and detection are highly parallel processes. Each algorithm operates on a per-pixel basis with no data or control dependencies. Thus we map both of these algorithms entirely on the GPU in order to take advantage of its many processing cores.

One choice for implementing these algorithms would have been to dedicate one GPU thread per pixel. However, there is an associated overhead with creating and scheduling a thread. To amortize the overhead cost of scheduling threads we dedicate one GPU thread for every e pixels. Thus each thread processes its pixels in serial but concurrently with all other threads that can be scheduled.

The incoming image is copied over to the GPU's video memory and is mapped as a texture. The first stage, Sobel filter, is a CUDA kernel which runs per pixel. Each pixel samples its immediate neighbors (3x3 pixel border) using the texture sampling tex2D() method. The input image remains as a texture and is unmodified, as it might be needed for classification after successful detection. The result of the Sobel filter (an edge map containing gradient angles) is saved to global video memory. Then, a per-pixel radial symmetry kernel runs using the gradient angle image as its input. Each non-zero element uses the gradient angle stored in the input location to calculate its voting areas. The values calculated at this stage are (x,y) coordinate pairs for the vertices of the voting triangles. Each pixel stores its result in an OpenGL Vertex Buffer Object (VBO). Once the radial

symmetry kernel finishes, OpenGL uses this VBO to draw triangles defined by the (x,y) pairs. Using the CUDA-OpenGL interoperability, there are no memory transfers between these two stages.

OpenGL then binds a pixel buffer object to a texture. The pixel buffer is chosen as the rendering target. With blending enabled, each triangle is then rendered with a high transperency (lowest non-zero alpha) value onto the pixelbuffer. The graphics hardware blends together all overlapping triangles, which causes these areas to appear brighter. The result here is an intensity map with the accumulated votes of the radial symmetry stage. A large gain here comes from the drawing and blending hardware of the GPU. A high number of overlapping incremental votes causes contention and serialization in most architectures, but the GPU hardware is optimized for this operation. In this stage, our technique takes full advantage of the present resources in a highly-efficient voting strategy.

For the final stage (maxima detection), we use a block maximum reduction where the cumulative voting image V is divided into b blocks of $e \times e$ pixels each. The reduction is performed per block and returns the index and value of the element with the highest vote total within each block. This process is also parallelized where possible, but becomes serial in its final stages. First, each thread finds the maximum element among its e elements in serial. Then, thread 0 of each $e \times e$ tile locates the maximum within that tile. This process occurs concurrently among the b blocks covering the entire voting image V. Maximum-value candidate sign centroids (in x,y pairs) are then copied back to the host processor.

On the host, a $\frac{640 \times 480}{b}$ reduction is performed to locate the block with the maximum value. That block is then marked as used and is further invalid. The maximum reduction is performed c number of passes, where c is the number of candidates, with each time invalidating the latest maximum block. In our implementation, we chose $e = 8$ where each thread processes that number of pixels. This value was used as it was determined that it provides a good balance between amortizing thread overhead costs and offers a high level of thread parallelism. The maximum-reduction which is performed on the CPU is only a 80×60 and executes fast enough on the host processor such that it does not affect the overall runtime of the detection process.

Extension In order to find the size of a speed limit sign, the iterative size technique described earlier can be used. For optimal performance, up to three voting ranges can be used due to the fact that a GPU can render to three different color buffers: red, green, and blue buffer. For each range of radii, all the voting primitives for that range are drawn in one of these three colors. The result is three color buffers containing the intensity maps of each range of radii. Intensity maxima in these individual maps that correspond in location to a maximum in the overall sum voting image indicate the radius range that the sign belongs in.

4.1 Memory

As previously mentioned, one of the goals for efficiency is minimizing expensive data transfers. The overhead delay of data transfers between devices is largely independent of the data transfer size. The overhead cost can be amortized if the data transfer is relatively large. In our implementation the incoming video image is copied to the GPU's video memory at arrival. It remains there for the duration of the detection process. Data buffers used in each step of the detection process are also allocated from the video memory during initialization and remain there for the entire execution of the program. Data buffers are allocated and mapped such that they can be accessed and modified as required without any need for duplication or transfer to main system memory. This point is important to keep in mind because memory allocated in CUDA that is later used by OpenGL must be formatted such that no duplication or transferring (for type-casting or otherwise formatting) must occur for each API to inter-changeably use the data. This is crucial for achieving high peformance on a system with limited processing and storage resources.

In total, there are three main data buffers here. A VGA (640×480 pixels) texture containing the input image, a VGA×10 (there are five (x,y) coordinate pairs per voting pixel) VBO containing the vertices necessary for voting, and a second VGA texture containing the results of radial symmetry voting. All three of these buffers are allocated in video memory and remain there for the duration of program execution. No per-frame allocations or memory transfers are necessary except for loading the video input data onto

video memory. Savings in memory transfers, however, are not the only gains here - hardware acceleration also plays an important part.

4.2 Hardware acceleration

Our implementation takes advantage of hardware acceleration in several key ways. First, the preprocessing stage uses texture sampling for accessing its input. Memory accesses through texture look-ups have several benefits when compared to global or shared memory reads (as per the CUDA Programming Guide [4]). Textures are cached, which allows for higher bandwidth provided that there is locality in the fetching patterns. In the case of the Sobel kernel, the texture is accessed in 3x3 blocks which exhibits good spatial locality. Another advantage is that data cached as a texture does not exhibit the same loss in performance as does global and shared memory due to uncoalesced accesses and bank conflicts. Lastly, texture fetches have automatic boundary case handling. This eliminates the need for auxiliary functions or extra branch statements that handle the image boundaries.

Hardware acceleration is also key to the performance of radial symmetry voting. First, the Arithmetic Logic Units (ALUs) in the hardware shaders of the GPU are used to calculate triangle vertex coordinates in parallel. Since RST is pixel-based without any data dependencies between pixels, as many individual pixels can be processed in parallel as there are ALUs available on the GPU, if scheduling permits. Then, in order to construct the voting image V, OpenGL uses the calculated vertices to render the results to a texture. OpenGL then assigns each voting triangle a low alpha (opaqueness) value and issues a highly-efficient glDrawElements() call to draw and blend the triangles. Blending accumulates the overall opaqueness of the voting triangles and the brightness at any point on the voting image directly depends on the number of overlapping triangles at that pixel. Areas with many overlapping triangles appear brighter than those with few overlapping triangles. Both drawing and blending are accelerated using the native graphics pipeline of the GPU. The last step is also hardware-accelerated as a CUDA maximum-reduction kernel uses fast texture-lookups to access the voting image texture.

In the Discussion section we explain how the different aspects of the implementation meet the metrics of efficiency established earlier.

Chapter 5

Results and Analysis

5.1 Evaluation Conditions

5.1.1 Environment

The video footage used for testing was chosen to cover as many different environments as present in our database. Environments we consider to be the type of road the vehicle is driving on including its surroundings. We define five different environments: inter-city highway, urban, rural, construction zone, and tunnel. Each of these categories exercises the robustness of the system in different ways. An inter-city highway, for example, usually has signs which are on either side of the highway and may appear relatively small when seen from the middle lane(s) of a multi-lane road. Highways exhibit an increased level of relevant features present on the road such as circular taillights, round tires, and round exit signs. An urban setting often exhibits the highest degree of relevant and irrelevant features and complicates detection as it often contains a large number of other traffic signs, circular features in billboard advertisements, round tires of vehicles perpendicular to the road, etc. Rural settings can offer simple detection where a sign is clearly juxtaposed against the sky and therefore has high-contrast, but trees can add a significant amount of edges due to their branches. Tunnels usually feature round over-head lamps, digital speed-limit signs, and vehicle head and tail lights which often appear round (and with high contrast) to a camera. Construction zones tend to contain highly-visible signs, but also many additional

Environment	No. Videos
City	12
Country road	27
Highway	33
Construction	6
Tunnel	2
Total	80

Table 5.1: Breakdown of driving environments in the input scenes.

warning signs and other precautionary measures which increase clutter. The distribution of the test set across these categories is shown in Table 5.1.

5.1.2 Lighting and Weather

In order to evaluate the system's robustness, it was tested in various lighting and weather conditions. The data set was divided among the following: daylight, nighttime, dawn or dusk, tunnel, and snow.. Cloudy and rainy weather are not separated due to insignificant variation in observed detection rates. Brightness and contrast changes due to these conditions offers much less of a difference than nighttime or dawn/dusk conditions. Snowy weather, however, is separated as heavy snow tends to partially or fully occlude signs. Tunnels are included in both lighting and environment due to features which are very characteristic of the environment within them and also man-made lighting in an enclosed space. In general, there has been a lack of coverage of lighting conditions in the literature. However, we believe the robustness of the RST algorithm is highlighted best through its ability to detect signs with high accuracy in conditions where the driver could have a difficult time seeing traffic signs such as at night. The distribution of the videos across the various lighting and weather conditions is shown in Table 5.2.

5.1.3 Digital Signs

Digital signs offer traffic controllers the option of having a variable speed-limit depending on the time of day, weather conditions, traffic status, upcoming zones with an accident, or unlimited speed if the signs are turned off. The frequency with which the

Lighting	No. Videos
Daylight	47
Night	19
Dusk / dawn	8
Tunnel	2
Snow	4
Total	80

Table 5.2: Breakdown of lighting conditions in the input scenes. *Videos containing digital signs are distributed among the other lighting conditions and are not counted separately towards the scene total.

Environment	No. Videos	Signs	Detected
Digital	9	18	18
Snow	4	11	11
Total	13	29	29

Table 5.3: Statistics for special conditions encountered in input scenes.

signs vary make it hard or impossible for either standalone or built-in GPS devices to keep up-to-date information in such zones. In addition, digital speed-limit signs are often used in tunnels where a dynamic GPS system without any pre-stored data could lose satellite connection and be unable to provide information to the driver. Therefore, the ability of a camera-based system to detect and identify digital signs becomes beneficial to driver-awareness. Results of digital sign recognition rates are presented in Table 5.3.

5.1.4 Testing Constants

In order to collect performance statistics and make valid comparisons, a few variables were held constant. The maximum target sign radius size and the number of detected candidates were determined through empirical testing. A maximum target sign radius determines the size of the voting triangular regions. Spanning the range of all possible radii is impractical as the size of a sign which can appear in the camera view is limited to the proximity the vehicle is to the sign. Using a maximum radius which is too small means that signs of a larger radius might not be detected. However, a maximum radius that is too large inherently adds a lot of noise to the image as non-sign edges can create false areas

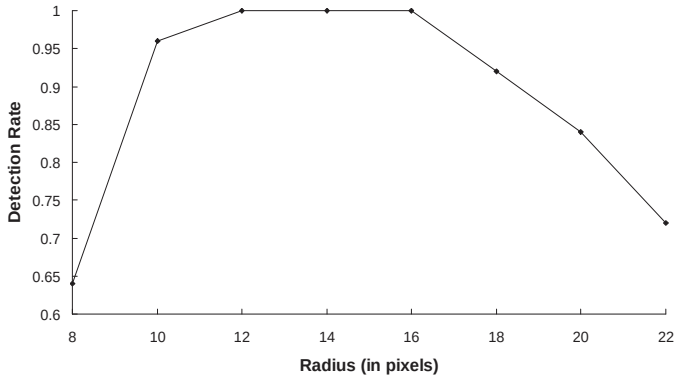


Figure 5.1: Detection rate with increasing target radius (in pixels) on a subset of nine videos.

of interest. Therefore, it is necessary to select a target maximum radius that offers a high detection rate.

Using a subset of the test set of video footage, detection rates for varying target radius sizes were recorded and are shown in Figure 5.1. As the graph show, the radii of 12 to 16 pixels achieve the highest detection rates of 100%. In order to determine which radius provides the best results, a further test was performed. We used two scenes containing initially small speed-limit signs which potentially become large in the camera's view to determine which radius size returns the highest number of frames in which the signs are detected. Figure 5.2 shows the results of this test and concludes that a target radius of 12 is most effective. In a total of 45 frames, detection rates of 52.3%, 33.3%, and 14.3% are observed respectively for target radii of 12, 14, and 16. This result is important as it shows that the target radius with the highest detection rate would allow better tracking and provide a candidate containing the present speed-limit sign to the recognition algorithm for the highest number of frames. Therefore, for all remaining tests, we use a target radius of 12 pixels.

5.2 System Performance

The system achieved 88% detection rate while executing at 33 frames/s using the host CPU for only issuing commands and the GPU for the main computation. The detection

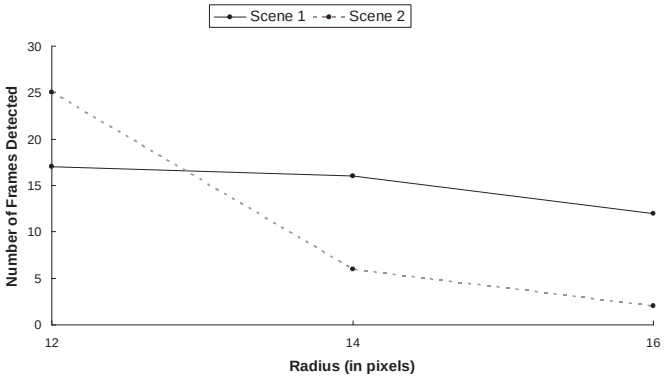


Figure 5.2: A closer look at system accuracy at different radius values. The graph indicates the number of frames in which a target sign was detected in two different scenes during the sign's interval of appearance in the camera's view.

runtime is invariant to the number of candidates detected, we cover this in greater detail in the Discussion section. Our base testing platform is an Intel Atom 230 @ 1.67 GHz and a Nvidia GeForce 9400M GS GPU. Both of these processors are found in low-end laptops and due to their ultra-low power requirements, are an ideal match for an embedded automotive system. The detection rate was achieved on 80 video scenes totaling 42 minutes and 43 seconds filmed at 16.7 Hz on European roads.

5.2.1 Accuracy

Detection rate evaluation was performed using the $\tilde{4}3$ minutes of video footage of roads in the European Union (ex: Germany, UK, France, etc) under various daytime and nighttime conditions as listed in Table 5.1. As Table 5.4 shows, the footage contains a total of 164 signs, of which 144 were detected correctly for a detection rate of 88%. In order to maximize the detection rate, a maximum of seven candidates per frame are used. We explain the selection of the number of candidates per frame in the Discussion section. Candidates which return single-frame confidence values above a certain instant threshold are then tracked. Each sign is tracked throughout the interval of its first appearance in the scene until the last frame before its disappearance (i.e. temporal integration). A successful detection is recorded when the temporal integration process returns an aggregate confidence value higher than a certain threshold. The classifier returned 0 false positives for the test set

No. Videos	No. Frames	Frame Rate	Time [mm:ss]	Detected	Present	Rate
80	42,778	16.7	42m : 43s	144	164	87.8%

Table 5.4: Test set information including detection rate figures.

video footage. Note, however, that the classifier is used only to validate our detection stage and is not our focus here to report *classification* statistics (true positives, false positives, true negatives, false negatives); we therefore leave that discussion for a separate publication.[1]

5.3 Examples

5.3.1 Robustness Evaluation

As previously mentioned, the detection system's robustness is demonstrated through its tolerance of changes in lighting and in-plane rotations. Figures 5.3(a) and 5.3(b) show successful detection of signs in the same scene city location both at daylight and at night. Figures 5.3(c) and 5.3(d) show signs which have been detected successfully despite an in-plane rotation of up to 6 degrees.

An important case for many European roads is detection of variable digital speed-limit signs. This is perhaps the most obvious case where a camera-based system is needed. Our detection results are shown in Table 5.3 and have a 100% detection rate. The digital signs used for testing were only those that appear visually legible to a human viewer in the video footage. Note that there is a special case in which these signs appear ineligible due to synchronization issues which we discuss in the Special Cases section.

Another common environment for digital signs is inside tunnels. Tunnels can have overhead lamps which appear as high-contrast circular features in a scene as Figure 5.5 shows. However, this setting usually contains very few features, most of which are high-contrast and allow for easier detection.

[1]This part of the work is largely complete and is lead by Pınar Muyan-Özçelik.

(a) (b)

(c) (d)

Figure 5.3: Robustness of RST: lighting and in-plane rotation invariance. (a) City scene at daylight. (b) The same city scene at night with street lighting, taillights, and oncoming headlights. (c) Scene with a sign which appears with an in-plane rotation of 5 degrees CCW. (d) Scene with a scene which appears with an in-plane rotation of 6 degrees CW.

(a) (b)

Figure 5.4: Digital speed-limit sign detection. (a) Highway scene with a successfully detected overhead digital sign. (b) Highway scene at night with all three displays detected.

Figure 5.5: Successfully detected speed-limit sign in a tunnel.

5.3.2 Missed Signs

Though invariant to lighting, our system is largely dependent on contrast. This is to be expected as it operates on the main features in a scene, which are extracted using a gradient operator (Sobel filter). Features with low contrast might not be extracted during the preprocessing stage and therefore make it impossible for RST to detect them. Currently, our system's effectiveness is reduced in low-contrast scenes such as fog and at dawn or dusk, which are shown in Figure 5.6. Bright features like oncoming headlights and direct sunlight at the horizon usually cause auto-gain cameras to lower the overall contrast and can make detection more difficult.

(a) (b)

Figure 5.6: Missed speed-limit signs due to low overall contrast in the scene. Note the bright oncoming headlights and saturated white highlights from the sun at the horizon.

5.3.3 Special Cases

There are some corner cases which can reduce the effectiveness of the recognition system. One example is a mismatch between the refresh frequency of digital signs (LED displays are not always on, but instead pulse on and off at an assigned frequency) and the

capture rate of the on-board camera. A situation can happen when the two devices are out of sync (i.e. digital sign is off when the camera captures a frame) and the digital signs appear turned off to the camera. In this case, from a sequence of consecutive frames there can be only a part of the frames in which the signs are visible and therefore recognition and tracking could be affected significantly. Figure 5.7 demonstrates this occurrence. Results for this situation will unquestionably vary on a case-by-case basis depending on the number of visible frames which the classifier can use and track in a sequence.

(a) (b)

Figure 5.7: Digital sign frequency with camera-capture rate synchronization occurence can be seen in the two consecutive frames above. (a) All four digital displays are lit and allow for detection. (b) The two displays on the left appear completely off while the right ones are in the process of flashing off and appear dim.

Signs which are largely occluded also present a detection problem. Snow, mud, debris, or other vehicles can occlude a sign partially and reduce its visibility. Figure 5.8 shows such a case in which our system was not able to successfully track and detect the present sign. This example perhaps could have been detected and recognized using a character-detecting algorithm, as the numbers are highly legible.

Figure 5.8: Speed-limit sign in a snowy setting that has been significantly occluded. The main characteristic (the red border around the sign) has been covered with ice and presents a difficult detection and recognition case.

Chapter 6

Discussion

6.1 Overall Performance

We measure the performance of only the detection stage using a runtime average per frame over a testing subset of the entire video footage. Due to the parallel implementation of the maximum-reduction stage, the achieved detection runtime of 30.3ms per frame is effectively invariant to the number of candidates detected. The much simpler CPU maximum reduction of 80×60 elements per additional candidate adds no significant increase in runtime. Important, however, is the increase in execution rate due to the classification stage with increasing number of candidates which need to be validated. We turn to this point through an examination of the tradeoffs in the system.

6.2 Accuracy-Runtime Tradeoff

Important to examine is how the system can be tuned to optimize runtime and accuracy. As is typical of this type of system, a gain in accuracy usually comes with an increase in runtime. However, we can gain some insides of how to choose a configuration which can offer optimal performance on a given platform of availability. Examining the tradeoffs between accuracy and runtime help us see what detection rate can be achieved on a resource-constrained platform, for example. Vice-versa, given unlimited resources, we can get an idea of the maximum accuracy obtainable. The factor which provides a

No. Candidates	1	2	3	4	5	6	7
Detected	9	14	15	15	16	16	17
Detection Rate	53%	82%	88%	88%	94%	94%	100%
Runtime (ms)	86	145	203	262	324	380	440
Overhead (ms)	56	59	58	59	62	56	60

Table 6.1: Runtime and accuracy scaling with increasing number of validated candidates.

direct correlation between accuracy and runtime in our system is the number of detected candidates which we choose to classify per frame. We evaluate the effects of this parameter using a subset of nine videos from out test set which gives us a representative variation in system performance.

6.2.1 Accuracy and Runtime Scaling

The effect which using increasing number of candidates has on accuracy and runtime is shown in Table 6.1 and Figure 6.1. These results show a non-linear increase in accuracy with increasing number of candidates. Accuracy eventually reaches 100% for the subset testing set. Important to note is that the achieved accuracy here is only on the subset testing set which was chosen at random to represent the overall testing conditions. The 100% accuracy result in this subset does not imply that this result can be obtained on the entire data set. In fact, we use a maximum of seven candidates because any further increase does not offer a higher detection rate. This is due to the fact that our system is currently contrast-limited and therefore has an upper bound of 88%. A preprocessing method which increases the contrast could increase the detection rates.

Table 6.1 and Figure 6.2 show a linear increase in runtime per frame with increasing number of candidates. The detection runtime alone is 28 ms per frame or 36 frames/s. While runtimes can be a useful benchmark, it is usually performance in terms of frames/s that gives us a better insight for system design, as camera capture rates and and driver alert rates can be considered in these terms.

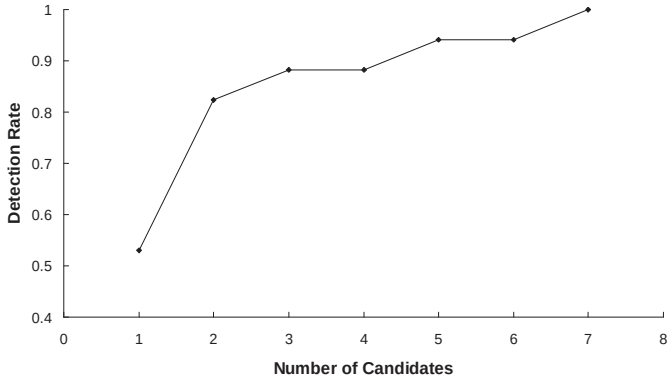


Figure 6.1: Detection rate with increasing number of candidates. Tested on a subset of nine videos from the full test set containing 17 signs.

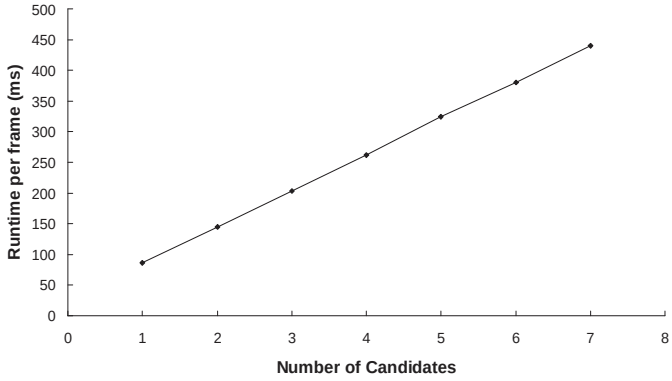


Figure 6.2: Runtime per frame with increasing number of detected candidates. Tested on a subset of nine videos from the full test set containing 17 signs.

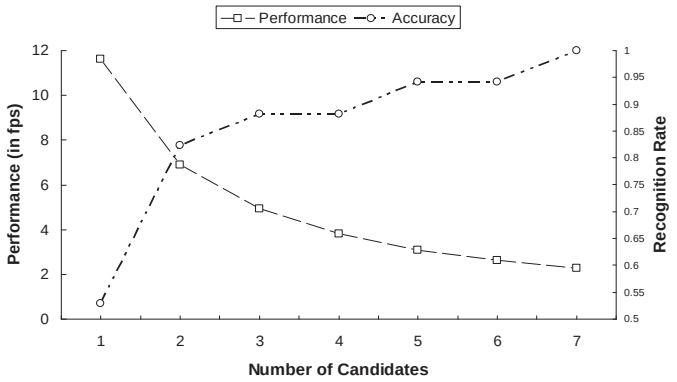


Figure 6.3: Performance (in frames/s) and recognition rates with increasing number of candidates. The curves indicate the relationship between the recognition rates possible with a desired execution rate.

6.2.2 System Tradeoffs

Individually, each of the previous results show us how each performance aspect (either accuracy or runtime) vary with a changing parameter. These results can be useful individually for system optimization, however, independently they help us little on selecting or designing a recognition platform. Thus, putting the results together present an accuracy-performance tradeoff which a system architect must consider when choosing the right platform for a specific application. Figure 6.3 shows the relationship between accuracy and performance of the system including the classification stage. Depending on the classification technique used, the performance curve could be shifted vertically. However, its general characteristic is unlikely to vary significantly as linearly increasing the number of examined candidates would most likely require the classification process to be executed for each candidate. In terms of accuracy, assuming a perfect classification technique would not affect the recognition curve as the whole process is limited to the accuracy of the detection stage.

Stage	Runtime [ms]	Percentage
Load Frame	3.2	10.6%
Sobel Filter	2.0	6.7%
Radial Symmetry	15.5	51.2%
Max. Reduction	9.6	31.5%
Total	30.3	–

Table 6.2: Breakdown of runtimes per stage and their corresponding percent of the entire process on a GPU. Runtimes are averages taken over a span of 1,000 frames.

Stage	Runtime [ms]	Percentage
Load Frame	–	–
Sobel Filter	28.4	10.8%
Radial Symmetry	229.0	87.0%
Max. Reduction	6.0	2.2%
Total	263.4	–

Table 6.3: Breakdown of runtimes per stage and their corresponding percent of the entire process on a CPU. Runtimes are averages taken over a span of 1,000 frames. Note that the frame-loading stage on the CPU does not incur a memory transfer since the video frame data already resides on the host's memory or cache and is readily accessible.

6.3 Hardware

6.3.1 Benefits of Hardware Acceleration

In order to analyze the effects of hardware acceleration, we compare the results to a CPU-only implementation. This implementation does not render RST results, instead it accumulates votes in a voting map. Each edge pixel increments the positions in the voting map corresponding to where it is casting its votes. The results of the detection exactly match those of the GPU implementation. Table 6.3 and Table 6.3 show a stage-by-stage breakdown of the detection process for a CPU and GPU, respectively. CPU results are measured on an Intel Atom 230 @ 1.6 GHz processor.

Interesting to compare are the runtimes of the per-pixel stages of the process (Sobel filter and RST). The CPU implementation clearly suffers in these stages as the CPU cannot exploit the available amount of parallelism in the algorithms. As Table 6.4 shows, both of these stages are nearly 15x slower on the CPU. Overall, the GPU implementation is up to 10x faster than its counterpart. The performance gain can be accredited to the

Stage	CPU [ms]	GPU [ms]	Gain
Sobel Filter	28.4	2.0	14.2x
Radial Symmetry	229.0	15.5	14.8x
Max. Reduction	6.0	9.6	0.63x
Total	263.4	27.1	9.72x

Table 6.4: Comparison of runtimes per stage and amount of GPU acceleration over the CPU implementation. The frame-loading stage is not included here due to a lack of this stage on the CPU.

large number of ALUs available in the GPU which can perform the necessary operations for many pixels simultaneously.

6.3.2 Hardware Scaling

An important factor for production is choosing a platform that is cost-effective. Therefore, it is helpful to understand how performance scales as increasing amounts of hardware resources are added. These results will then yield a cost per unit performance figure for different hardware options. Table 6.5 shows the performance of the system on several devices, with a percentage increase in performance listed for each.

Performance does not scale ideally; a 4x increase in the number of processing cores does not offer a 4x increase in the number of frames/s. This is to be expected, however, as there are several overheads and bottlenecks introduced in a larger device. First, there is a larger overhead for scheduling threads across four SMs as opposed to just a single SM. Second, distributing operands to the threads awaiting them involves a more complicated on-chip network. Third, with more threads executing at once there is higher memory contention and certain reads or writes could end up being serialized if the memory controller cannot service all requests simultaneously.

This insight into performance scaling can be useful to a system designer for balancing the expected performance gain with the associated increase in system cost. These results can be useful in determining which hardware is suitable for a recognition system if a certain classification is to be included. If a target rate of 30 frames/s is desired, for example, a designer might choose to work with a four-core solution as it leaves some headroom for the additional computation required for classification.

GPU	SMs	Runtime [ms]	Frames/s	Speedup
9200M GS	1	54.5	18.3	–
9400M G	2	30.3	33	1.8x
9600M GS	4	17.0	58.7	3.2x

Table 6.5: GPU scalability across a variety of CUDA-enabled Nvidia devices from the GeForce family. Speedup figures are shown with respect to the device with one streaming multiprocessor (GeForce 9200M GS).

6.4 Efficiency

Our implementation follows the five metrics of efficiency (highly-parallel algorithms, minimized memory transfers, algorithms should reflect the architecture, low memory sizes, near-constant frame rate) we described in the Background section. First, all three stages of our detection (Sobel preprocessing, RST, parallel maximum reduction) operate on a per-pixel basis and are therefore highly parallel. Second, once the input video frame reaches the GPU's memory, it resides there for the duration of the detection process. A minimal 80×60 element copy is performed at the end to locate the top candidates. CUDA to OpenGL communication, where vertices are used for rendering and the resulting voting image is uses for the maximum reduction, involve no memory transfers since these operations consist of simply passing pointers. Third, the amount of parallelism present in the GPU architecture is reflected very closely in the algorithms chosen for the detection process. This allows high functional unit utilization and reduces the effect of system bottlenecks. Fourth, the total memory size used for the main data structures is around 3,600 kB or 3.5 MB (input 640×480 image + 640×480×10 vertex buffer object + 640×480 voting image). Fifth, since each step of the process is pixel-based and there is a constant number of pixels in the input image, the runtime should not vary significantly. More importantly, no algorithms are used which have an unbounded possible execution time. Processing per frame will not be entirely constant, however, as variations in the number of total edge pixels extracted in the preprocessing stage will cause the number of voting triangles to be rendered to vary, as well, and can affect runtimes. But again, this is bounded by the constant number of pixels and in the worst case as many as all 640×480 pixels will require a voting triangle to be rendered.

6.5 Limitations in our System

6.5.1 Frame Data Memory Transfer

Currently, the input frame data is loaded into host memory then copied to the GPU's texture memory. This step occurs for each frame and it adds some overhead to the system. Table 6.2 shows the distribution of runtime for each stage of the detection process. The overall effect of paying the cost for the video data transfer is nearly 11%. Theoretically, in a zero-copy implementation, the overall detection process could be accelerated to 27.1 ms per frame or 36.9 frames/s. This could be accomplished if the frame data transfer is bypassed in a dedicated embedded system using a Direct Memory Access (DMA) transfer which would place the frame data into GPU memory. Another way to reduce or completely eliminate the frame data copy overhead is to use a double buffer approach where the GPU is processing the last input frame while the new frame is copied into a second buffer. When processing of the first frame is done, the two buffer pointers would be swapped and the GPU can process the next frame.

6.5.2 Negative Votes

In our implementation, only positive votes are cast using the rendered triangles. Negative voting of the RST, described by Loy and Barnes [17], which is used to attenuate the effect of straight lines too long to correspond to any relevant shape, was not implemented. Its implementation could consist of rendering a second positive vote map consisting of only the areas affected by negative voting and simply subtracting this image from the positive voting image. Its usage could reduce a lot of the noise and clutter encountered in our voting images due to edges not a part of any geometric shapes present.

6.5.3 Tracking

Using a Hidden Markov Model (HMM) or a Kalman filter might improve the results of our tracking. Currently in our system, size and in-plane rotation factors are used to increase the confidence with which candidates are tracked and reduce the possibility of artifacts appearing as false positives. An additional tracking factor of position could

increase tracking performance and perhaps return results faster and with higher confidence.

6.6 Future Work

6.6.1 Clutter Reduction: Minimum Radius

Currently, the voting distance R_{min} is set to 0 in order to approximate a triangular voting area. However, this adds unnecessary clutter to the voting image as it is impractical to detect target signs smaller than a few pixels in radius. Using a voting radius of $R_{min} = 0$ increases the voting effect that irrelevant features such as tree branches add to a given scene. Changing R_{min} to be greater than 0 implies modifying the voting shape to a quadrilateral. This would increase memory storage, number of vertex calculations, and rendering complexity. Quadrilaterals would have to be rendered instead of triangles, which would require two triangles to be rendered for each voting area (a quadrilateral is divided into two triangles before it is rendered). It would be worthwhile to examine the effects on detection and execution rate such a change would have.

6.6.2 US Speed-Limit Signs and Other Traffic Signs

As Loy and Barnes [17] demonstrate, RST can be used to detect other polygonal signs such as octagonal stop signs, triangular and diamond-shaped warning signs, and rectangular traffic information signs. The complexity of the algorithm does not change with the different detection target shapes. The modification is simply through adjusting the algorithm's parameters - changing the voting width w according to the number of sides n in the targeted polygon. It would be worthwhile examining the robustness of the algorithm for non-circular detection using our hardware-accelerated implementation.

Keller et al. [15] extends the use of the RST to US speed-limit signs using a few simple modifications. Since US speed-limit signs are typically rectangular with a height larger than the width, the voting distances of pixels with a vertical gradient (either 90 or 270 degrees) should correspondingly be larger than those with a horizontal gradient. Also, since the border edges of US signs are either approximately horizontal or vertical (with slight perspective skewing), voting can be limited to pixels with gradients within a few

degrees of the horizontal and vertical axes. This could also reduce runtimes as less voting areas would have to be rendered.

Chapter 7

Conclusion

Contribution We have introduced a novel implementation of the feature-based Radial Symmetry Transform which we applied to speed-limit sign detection. There was no general loss of robustness compared to the original formulation of the technique, as testing in various environments and lighting conditions showed detection rates as high as 88%. Our comprehensive testing exceeds what has been detailed in the literature to this date as a much broader range of driving conditions is covered in our study compared to previous work. Using various techniques of hardware acceleration, our implementation allows for real-time speed-limit sign detection on resource-constrained embedded processors. Our focus on hardware acceleration and efficiency metrics for embedded hardware also fills a gap in the current literature of in-vehicle computer vision.

Efficiency Throughout our search for algorithms which could map well onto a parallel processor, we kept in mind several key principles of embedded system design. Our implementation reflects that, as well, and we adhered to these ideas throughout the development process. Any system aiming to perform in-vehicle detection or recognition must focus on the hardware constraints present in the target platform. We believe that not enough attention has been paid to this topic in the literature. There needs to be a shift of focus in the field of in-vehicle computer vision if any part of the vast number of driver and pedestrian safety schemes in the research are to be integrated in production vehicles. We hope the techniques and ideas presented here help future researchers and developers target their computer vision

algorithms towards real-time execution on embedded-grade hardware.

Motivating GPGPU We also presented the idea of using a GPU as an embedded general-purpose processor highly capable of computer-vision tasks. The programmability and performance scalability of the GPU make it a processor worthy of examination for embedded platforms. Its programmability can cut down design cycle lengths and overall system costs. It can also allow other automotive tasks to use its resources when available. The native graphics capability is an added advantage and can be used to accelerate rendering of a graphical user interface or navigation information. Its scalability and wide availability offer system designers the option of increasing system resources without major system redesigns and extra application-specific hardware. As a co-processor, the GPU can free up the main processor and enable it to handle other concurrent tasks.

Bibliography

[1] N. Barnes and A. Zelinsky. Real-time radial symmetry for speed sign detection. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 566–571, 2004.

[2] N. Barnes, A. Zelinsky, and L. S. Fletcher. Real-time speed sign detection using the radial symmetry detector. *IEEE Transactions on Intelligent Transportation Systems*, 9(2):322–332, 2008.

[3] Edmund Coersmeier, Sven Jaborek, Patrick Paul, Martin Bucker, Marc Hoffmann, Lukas Pustina, Simon Schwarzer, Felix Leder, and Peter Martini. Multicore processing for object recognition in mobile devices. In *Proceedings of the embedded world Conference*, 2008.

[4] NVIDIA Corp. NVIDIA CUDA Compute Unified Device Architecture. `http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf`, 2007.

[5] A. de la Escalera, L.E. Moreno, M.A. Salichs, and J.M. Armingol. Road traffic sign detection and classification. *IEEE Transactions on Industrial Electronics*, 44(6):848–859, Dec 1997.

[6] EyeQ. `http://www.mobileye.com/manufacturer-products/processing-platforms/EyeQ`.

[7] James Fung and Steve Mann. OpenVIDIA: Parallel GPU computer vision. In *2005 Proceedings of the 13th Annual ACM International Conference on Multimedia*, pages 849–852, New York, NY, USA, 2005. ACM.

[8] Krzysztof Gajdamowicz. *Automated processing of georeferenced colour stereo images for road inventory.* PhD thesis, KTH, Department of Geodesy and Photogrammetry, 1998.

[9] M.A. Garcia-Garrido, M.A. Sotelo, and E. Martm-Gorostiza. Fast traffic sign detection and recognition under changing lighting conditions. In *Intelligent Transportation Systems Conference, 2006 IEEE*, pages 811–816, Sept. 2006.

[10] Dariu Gavrila. Traffic sign recognition revisited. In *Mustererkennung 1999, 21. DAGM-Symposium*, pages 86–93, London, UK, 1999. Springer-Verlag.

[11] J. Hatzidimos. Automated traffic sign recognition in digital images. In *International Conference on Theory and Applications of Mathematics and Informatics*, pages 174–184, 2004.

[12] Bahram Javidi, Maria-Albertina Castro, Sherif Kishk, and Elisabet Perez. Automated detection and analysis of speed limit signs. Technical Report JHR 02-285, University of Connecticut, February 2002.

[13] V. Kamat and S. Ganesan. A robust Hough transform technique for description of multiple line segments in an image. In *1998 International Conference on Image Processing*, volume 1, pages 216–220 vol.1, Oct 1998.

[14] N. Kehtarnavaz, N.C. Griswold, and D.S. Kang. Stop-sign recognition based on color-shape processing. *MVA*, 6:206–208, 1993.

[15] Christoph Gustav Keller, Christoph Sprunk, Claus Bahlmann, Jan Giebel, and Gregory Baratoff. Real-time recognition of U.S. speed signs. In *IEEE Intelligent Vehicles Symposium*, pages 518–523, June 2008.

[16] M. Lalonde and Y. Li. Road sign recognition - survey of the state of art. `http://nannetta.ce.unipr.it/argo/theysay/rs2/#publications`, August 1995.

[17] G. Loy and N. Barnes. Fast shape-based road sign detection for a driver assistance system. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 70–75 vol.1, 2004.

[18] G. Loy and A. Zelinsky. Fast radial symmetry for detecting points of interest. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):959–973, 2003.

[19] Consolidated Solutions. `http://www.mobileye.com/manufacturer-products/applications/consolidated-solutions`.

[20] F. Moutarde, A. Bargeton, A. Herbin, and L. Chanussot. Robust on-vehicle real-time visual detection of American and European speed limit signs, with a modular traffic signs recognition system. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 1122–1126, June 2007.

[21] T. Pavlidis. *Algorithms for Graphics and Image*. Computer Science Press, 1982.

[22] G. Piccioli, E. De Micheli, P. Parodi, and M. Campani. Robust road sign detection and recognition from image sequences. In *Proceedings of the 1994 Symposium on Intelligent Vehicles*, pages 278–283, Oct. 1994.

[23] L. Priese, J. Klieber, R. Lakmann, V. Rehrmann, and R. Schian. New results on traffic sign recognition. In *Proceedings of the 1994 Symposium on Intelligent Vehicles*, pages 249–254, Oct. 1994.

[24] Daniel Reisfeld, Haim Wolfson, and Yehezkel Yeshurun. Context-free attentional operators: the generalized symmetry transform. *Int. J. Comput. Vision*, 14(2):119–130, 1995.

[25] W. Ritter, F. Stein, and R. Janssen. Traffic sign recognition using colour information. *MathMod*, 22:149–161, 1995.

[26] Yong-Jian Zheng, W. Ritter, and R. Janssen. An adaptive system for traffic sign recognition. In *Proceedings of the 1994 Symposium on Intelligent Vehicles*, pages 165–170, Oct. 1994.