Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Chad Aeschliman

Entitled Autonomous Vehicle Steering

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

R. A. DeCarlo	
Chair	
D. S. Ebert	
J. Eaton	

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): R. A. DeCarlo

Approved by: M. R. Melloch 12/12/08
Head of the Graduate Program Date

Graduate School Form 20
(Revised 10/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation: Autonomous Vehicle Steering

For the degree of Master of Science in Electrical and Computer Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Chad Aeschliman
Signature of Candidate

12/12/08
Date

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

AUTONOMOUS VEHICLE STEERING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Chad Aeschliman

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2008

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

I would like to thank first my family and friends whose constant encouragement kept me going during the early, difficult years. I am grateful for my advisor, Ray, whose continuous questions forced me to understand what I was doing and in the process greatly improved my research. I want to thank Jim Eaton for introducing me to the power of flow diagrams. I want to thank Eric and Rudy for their courageous help during the first test runs of the system as well as Tyler for his help with the mechanical setup. Finally, I thank my God, to Whom I owe all my ability to think and reason and the safety I've enjoyed throughout this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Aeschliman, Chad M.S.E.C.E., Purdue University, December 2008. Autonomous Vehicle Steering. Major Professor: Raymond DeCarlo.

Controlling the steering of a vehicle using a single forward-looking camera is a difficult problem. This work presents a solution under certain conditions which has proven very robust after many hours of testing and which can be implemented cheaply. The solution is composed of three stages. In the first stage, a novel dual-correlation method is used to identify the lane boundaries and from these the position and orientation of the vehicle. Next, a model of the vehicle as well as some additional vehicle information is used to filter the estimated position and orientation from the first stage. In the final stage, constant state feedback based on the discrete linear quadratic regulator (DLQR) is used to determine the appropriate steering of the vehicle. After each stage is explained, experimental results are given which demonstrate the effectiveness of the proposed solution.

1. INTRODUCTION AND OVERVIEW

Put as simply as possible, the goal is to steer a vehicle to remain in the center of its current lane with a computationally efficient, low cost process. Furthermore the available inputs and outputs are fixed. Figure 1.1 shows the basic input/output of the system. The primary input into the system is a camera mounted in the windshield of the vehicle. Two secondary inputs are the forward speed of the vehicle (obtained using GPS) and the steering wheel angle. The only control output is a 12V motor

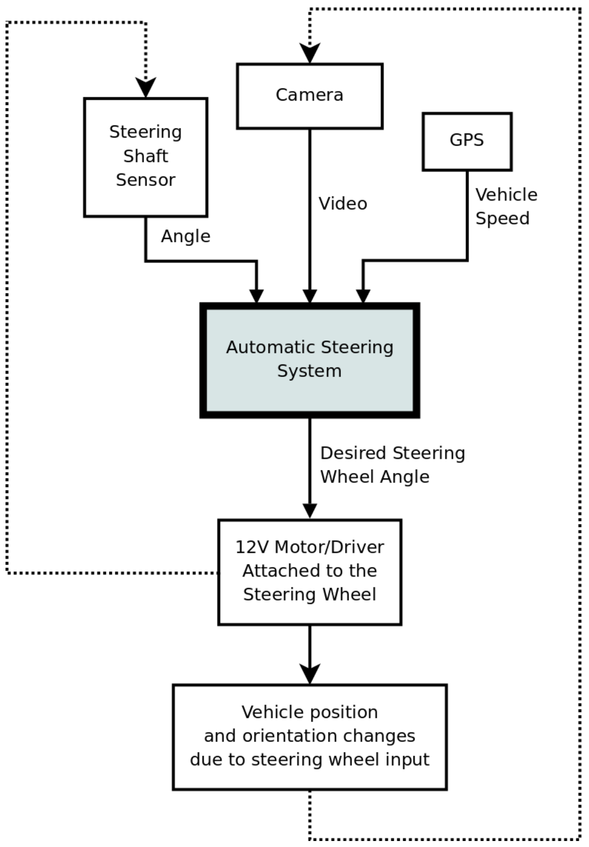
Camera
Steering Shaft Sensor
GPS
Video
Vehicle Speed
Angle
Automatic Steering System
Desired Steering Wheel Angle
12V Motor/Driver Attached to the Steering Wheel
Vehicle position and orientation changes due to steering wheel input

Fig. 1.1. Input/Output Diagram

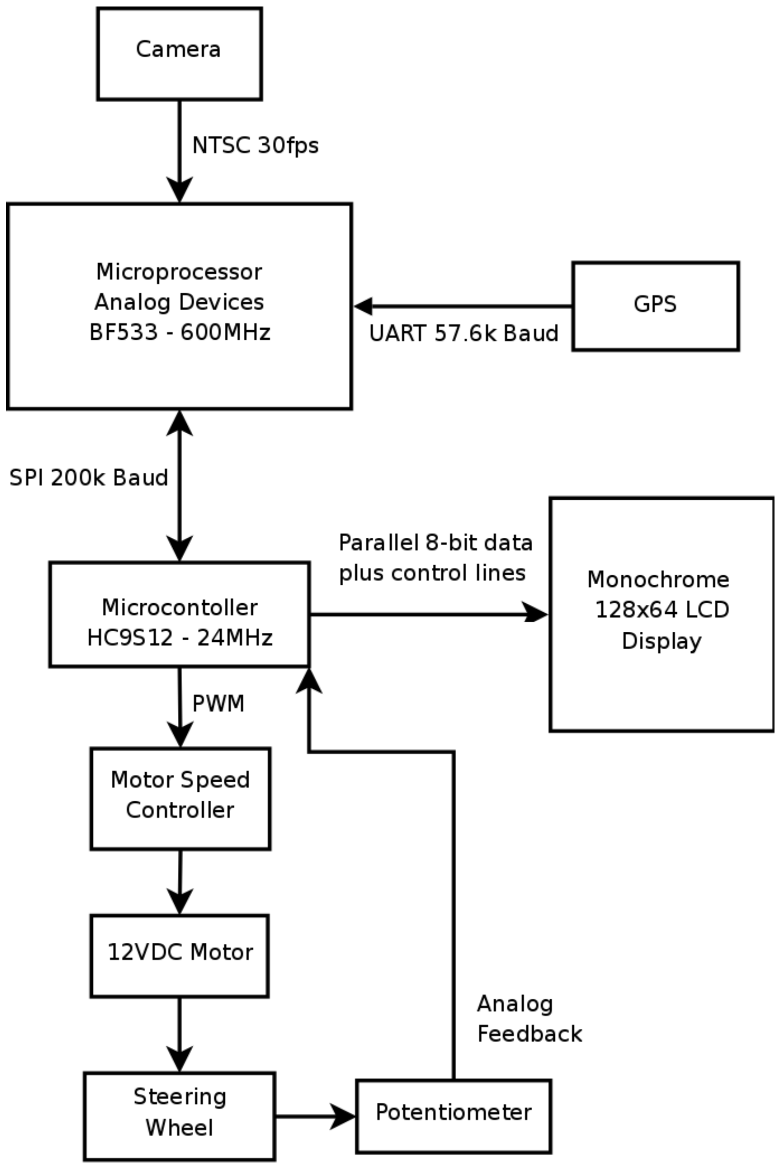

Fig. 1.2. Complete Hardware Diagram

which has a friction fit to the steering wheel and which can easily be overpowered in case of an emergency.

A diagram of the complete hardware setup used is given in figure 1.2. Part numbers for the major components of the system are given in table 1.1. A B/W camera mounted to the windshield captures video in the NTSC format. This video stream is processed image-by-image using a DSP microprocessor to identify the lane boundaries

Table 1.1
Part numbers for the major components used in the system.

Item	Manufacturer	Part Number
Camera	Clover	CM720
Microprocessor	Analog Devices	BF533
Microcontroller	Freescale	HC9S12C32
GPS	Global Sat	EM406A
Motor Speed Controller	Parallax	HB25

and the vehicle position and orientation. From this information an appropriate steering wheel angle can be determined which will drive the vehicle to the center of the lane. The microprocessor transmits the control signals to an external microcontroller which uses a DC motor and a closed loop control to move the steering wheel to the correct angle. This changes the location and orientation of the vehicle and hence the location and orientation of the camera. This in turn changes the subsequent input video stream. Hence the overall process is a closed feedback loop. The external microcontroller also sends current system state information to an LCD display, allowing the driver to monitor the performance of the system. In the event of system failure, the driver can easily regain control of the vehicle.

A three step process is used to determine an appropriate steering wheel angle from the video input as shown in figure 1.3. The first step is determine from the video a noisy measurement of the position and orientation of the vehicle relative to the lane boundaries. This step, which is based on several novel ideas, is the most difficult of the three and is explained in detail in chapter 2. Once the noisy measurements have been made, they are passed through a Kalman filter which uses a model of the vehicle along with the speed of the vehicle and the steering wheel angle to reduce the error in the measurements in an optimal way. This step is explained in chapter 3. Finally, given the filtered state of the system an appropriate control (steering wheel angle) can be computed. The discrete linear quadratic regulator (DLQR) is used to do this as explained in chapter 4. Note that experimental results for each step in the process are given in their respective chapters.

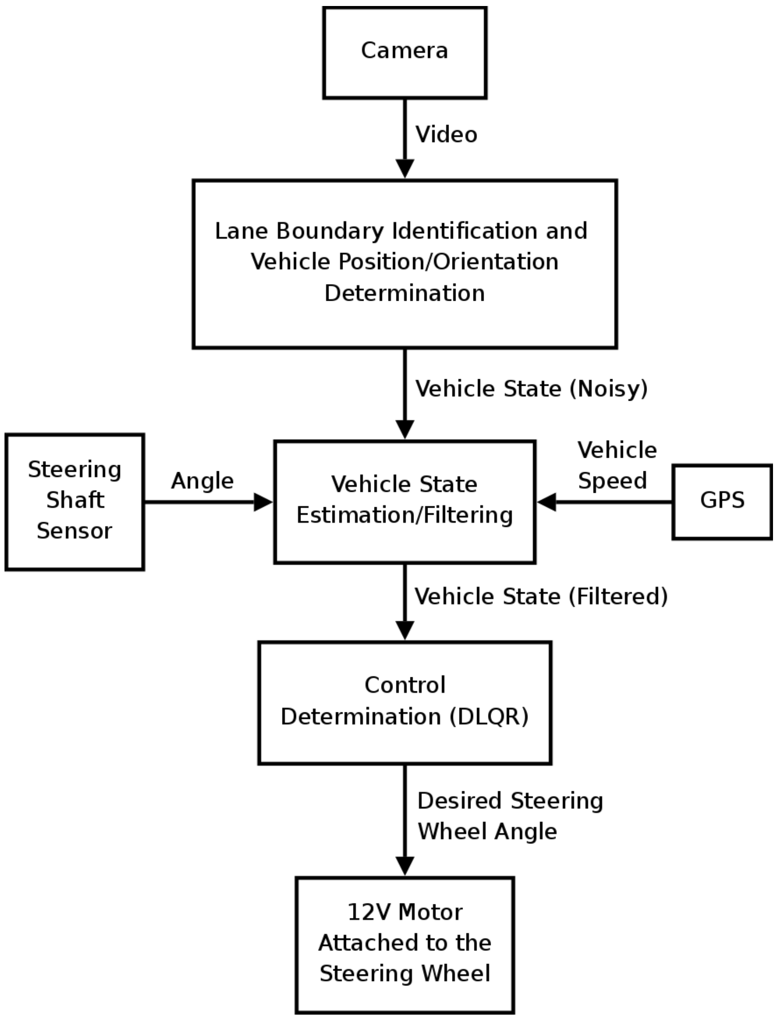
Camera
Video
Lane Boundary Identification and Vehicle Position/Orientation Determination
Vehicle State (Noisy)
Steering Shaft Sensor
Angle
Vehicle State Estimation/Filtering
Vehicle Speed
GPS
Vehicle State (Filtered)
Control Determination (DLQR)
Desired Steering Wheel Angle
12V Motor Attached to the Steering Wheel

Fig. 1.3. Overview of the main processing stages.

2. IMAGE PROCESSING

In order to control the vehicle, its position and orientation relative to the current lane need to be determined. Recall that a B/W camera mounted to the windshield captures video in the NTSC format. This video stream is processed image-by-image using a DSP microprocessor to identify the lane boundaries and the vehicle position and orientation. We first develop the problem statement and necessary assumptions in section 2.1. We then present a solution to the problem by first remapping the input images into a new reference frame, identifying the lane boundaries in the remapped image (section 2.3), and then solving for the position and orientation of the vehicle (section 2.6).

2.1 Problem Statement and Assumptions

As mentioned, a camera attached to the rear view mirror captures NTSC video which has a resolution of 720x480 and a frame-rate of 30fps. To allow for successful control, two questions need to be answered from this sequence of images.

(Q1) For a convenient coordinate system **C**, what is the evolving shape of the lane ahead?

(Q2) Relative to **C**, what is the position and orientation of the vehicle?

The answer to the first question gives a desired trajectory for the vehicle which the control tracks while the answer to the second question gives an error signal relative to this trajectory making control possible. To make the problem practical we also impose limits on processing time and algorithm efficiency by forcing all image processing to be done on a single 600MHz DSP chip.

The problem was simplified by making the following assumptions about the lane and lane boundaries (the white or yellow dashed or solid lines used to delineate the lanes).

(A0) The lane in front of the vehicle is rectangular over a short distance and lies in the plane of motion of the vehicle.

(A1) Lane boundaries are brighter than the nearby road with a contrast of at least 7 on a 0-255 brightness scale.

(A2) There is a well-defined edge with a sharp contrast in brightness between the lane boundary and the road.

(A3) Both edges of a lane boundary can be approximated as straight lines over a series of short segments, where each segment is valid over some short distance.

(A4) The basic geometry of lanes is consistent and well known. Specifically, the width of the lane boundaries varies from 10cm to 15cm and the lane width varies between 2m and 4m.

(A5) At least one edge of the lane is delineated with a lane boundary at all times.

(A6) The camera has a clear view of the lane boundaries ahead for at least 24m.

Assumption A0 greatly simplifies answering the first question (Q1) within a particular coordinate system which will be described later. Also, under this assumption answering both questions reduces to the simpler problem of identifying the lane boundaries. Assumptions A1-A4 are a reasonable set of assumptions that permit lane boundary identification. In fact, they agree closely with how a human driver identifies the lane boundaries. A driver will know the approximate shape and location of the lane boundaries as well as their color/brightness; these assumptions codify this knowledge. Assumptions A5 and A6 are included to ensure that answering the two questions is always possible.

From these assumptions, we present a simple and robust algorithm for determining whether or not a section of the camera image contains a lane boundary. This will make it possible to track the lane boundaries, which will in turn provide the answers to the two questions for each image.

2.2 Related Work and Contributions of this Research

An extensive review of previous work in the area of lane tracking based on image processing is given in [1]. Many use a form of edge detection to identify the lane boundaries or implement a template matching scheme. For example, [1] describes using steerable Gaussian filters to identify the lane boundaries. A histogram method based on color differentiation is described in [2]. More general image identification algorithms, such as scale-invariant keypoints can be used as well [3]. Of the various methods surveyed in [1], only one [4] utilizes cross-correlation which is the closest method to the one presented here.

This research differs from the previous work in three key ways. First, a minimal hardware approach is taken. The image processing and control algorithms both run on a 600MHz DSP chip. A single camera mounted to the windshield is the only sensory input into the system. This minimal approach by necessity limits the complexity of the image processing, resulting in a simpler system with fewer internal failure modes. Despite the simplicity of the approach, the algorithm is quite robust and has been used successfully in an experimental setting with many hours of testing. We believe this approach will be attractive to industry and lead to more widespread use than the generally complex, computationally expensive techniques surveyed in [1].

A second key difference is transforming the image before identifying the lane boundaries from the image coordinate system to a vehicle-centric coordinate system $\mathcal{X}_v$ (see section 2.3). In this coordinate system the lane boundaries are always within a few degrees of vertical, making them simpler to identify. This is a key improvement over methods which identify lane boundaries in the original image where the angle of

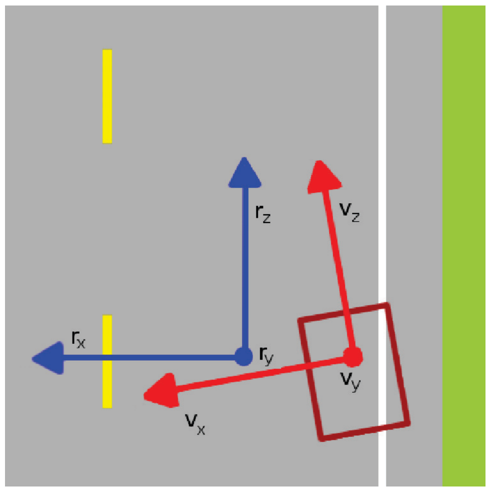

Fig. 2.1. Roadway and Vehicle Coordinate Systems

the lane boundaries can vary from 30 to 90 degrees off the horizontal. Furthermore, in the original image the angle of the lane boundaries is dependent on both the position and angle of the vehicle whereas in $\mathcal{X}_v$ the angle of the lane boundaries is dependent only on the angle of the vehicle, i.e. position and angle are decoupled.

The third key contribution of this paper is the novel dual correlation technique used to identify the lane boundaries. This method provides a significant improvement over traditional normalized/unnormalized cross-correlation. The method is explained in detail in section 2.4. The strength of this method is that it ignores common road deformations such as dark/light patches while having the same computational expense as unnormalized cross-correlation (and hence a significant computational savings over normalized cross-correlation).

2.3 Coordinate Systems

Three coordinate systems need to be described. The roadway coordinate system $\mathcal{X}_r$, which will be used as the system **C** in the questions in section 2.1, is shown in blue in Figure 2.1. The y-axis, r_y, is perpendicular to the road plane and the z-axis, r_x, lies in the road plane parallel to the edges of the lane. The x-axis, r_x, lies in the road plane perpendicular to the z-axis. The origin lies on the road plane midway between the edges of the lane and is chosen in such a way that the z-coordinate of the camera is 0.

The vehicle coordinate system $\mathcal{X}_v$ is shown in red in figure 2.1. This coordinate system is strongly tied to the location of the vehicle, as its name implies. The origin is located in the road plane directly beneath the rear view mirror. This provides a convenient central point for identifying the position of the vehicle. The y-axis, v_y, is again perpendicular to the road plane. The positive z-axis, v_x, points in the direction of the vehicle's forward motion. Hence $\mathcal{X}_v$ differs from $\mathcal{X}_r$ by a rotation about the y-axis followed by a translation along the x-axis of $\mathcal{X}_r$.

2.4 Dual Correlation for Lane Boundary Identification

The camera input image is a projection of 3d-space onto the xy-plane of a third coordinate system, the camera coordinate system. This coordinate system has its origin at the camera with the positive z-axis normal to the center of the camera lens pointing away from the camera. The x and y axes are oriented to correspond with the rows and columns respectively of the captured image.

In this section our approach is to describe the lane boundary identification algorithm in the vehicle coordinate system and then in the next section describe how the actual input images can be converted to this coordinate system. Cross-correlation is used to compare this data to the known geometry of lane boundaries in order to identify whether a small section of the vehicle coordinate system data contains a lane boundary. Once several points containing a lane boundary have been identified they are used to determine the position and orientation of the vehicle relative to the lane boundaries.

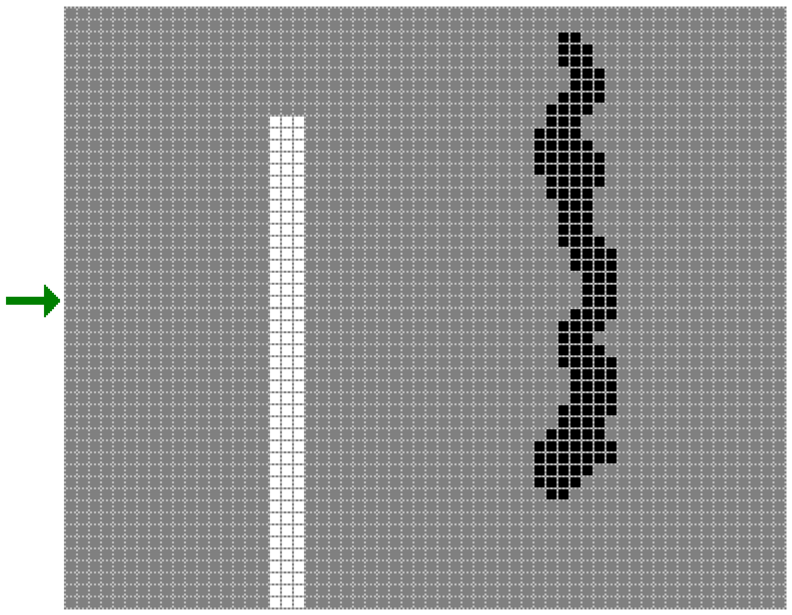

Fig. 2.2. Subsection of an example overhead view.

2.4.1 1D and 2D cross-correlation and kernel development

The 1D cross-correlation of a kernel $K \in \mathcal{R}^n$ with a vector $X \in \mathcal{R}^m$ (denoted $K \star X$) is a vector $W \in \mathcal{R}^{m-n+1}$ with entries defined as,

$$w_i = \sum_{j=0}^{n-1} k_j x_{i+j} \qquad i = 0, 1, \ldots, m-n \tag{2.1}$$

where w_i, k_i, and x_i denote the i element of W, K, and X respectively. The ith entry in W is a measure of how well the shape of a section of X of length n starting with the ith entry matches the shape of K. A larger value indicates a better match. This is exactly the behavior needed to detect lane boundaries.

The kernel K used will be derived through an example. Figure 2.2 shows the brightness levels of a 60x50 section of typical data. Black indicates smaller values and white indicates larger values. The vertical white strip is a lane boundary and the black area is a patch on the road. The rest of the image is the normal road surface. For simplicity, the brightness of each pixel will be defined on a scale from 0 to 2 with 2 being the brightest. The lane boundary will have brightness 2, the patch brightness

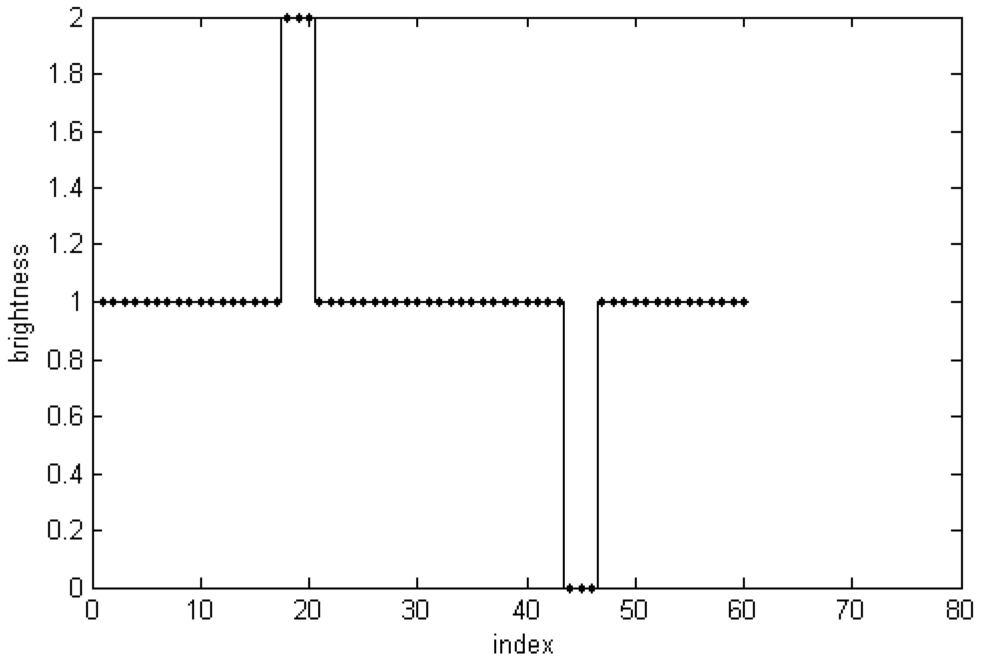

Fig. 2.3. Horizontal cross section of Figure 2.2 with dots indicating the discrete values.

0, and the rest of the image brightness 1. Figure 2.3 shows a horizontal cross section of the image brightness along the row pointed to by the green arrow. This cross section will be denoted as $X = \{1, \ldots, 1, 2, 2, 2, 1, \ldots, 1, 0, 0, 0, 1, \ldots, 1\}$.

Because the width of the lane boundaries is known to be between 10 and 15cm (A4) and the image is sampled at 5cm intervals, a lane boundary will be 2 or 3 samples wide. Hence the kernel K must select a bright section that is 2 or 3 samples wide followed by a dark section. As a first attempt we try $K_0 = \{1, 1, 0, -1, -1\}$. Intuitively, when this kernel is applied to a sequence of five samples from X, it will reward brightness in the first two samples, ignore the third sample, and penalize brightness in the last two samples. $K_0 \star X$ is shown in figure 2.4 and is equal to $\{0, \ldots, 0, -1, -2, -2, 0, 2, 2, 1, 0, \ldots, 0, 1, 2, 2, 0, -2, -2, -1, 0, \ldots, 0\}$. The maximum values are achieved when X matches the general shape of K, i.e. when X is constant at some brightness for a couple samples and then drops to a lower brightness and

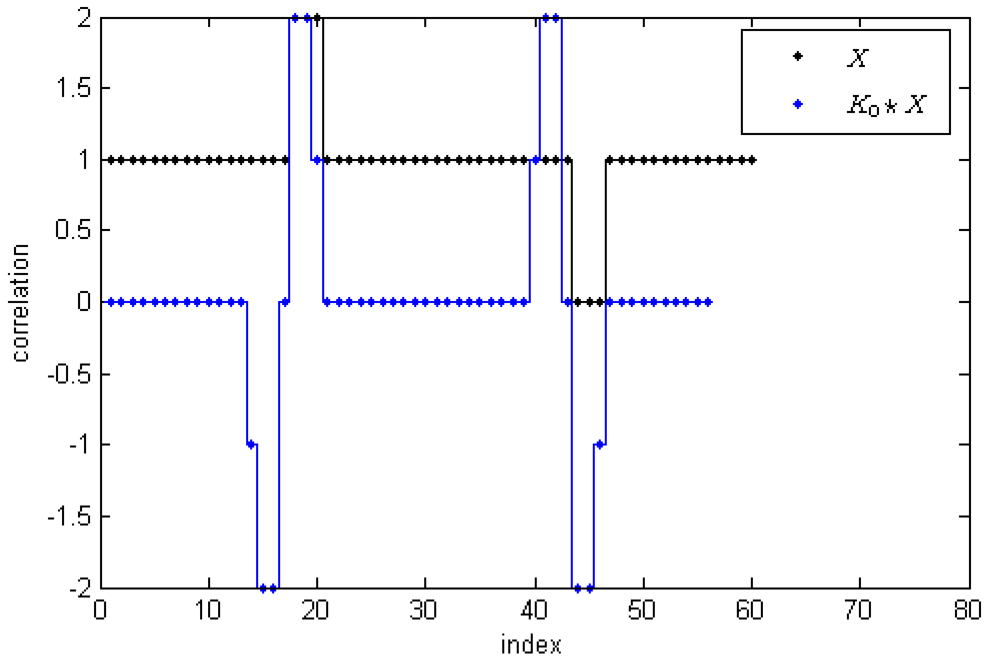

Fig. 2.4. Cross-correlation of first guess K_0 with X.

holds for a couple samples. When X is the opposite of the desired shape (dark first then brighter), a large negative value is given. Most importantly, when X is constant the cross-correlation gives 0.

Although K_0 is clearly not the kernel to use, it does provide a good starting point. It has correctly isolated the areas of major change in the image. Unfortunately, there are two maxima, one near the lane boundary and one near the patch. Notice however that the order of maxima and minima is different near the lane boundary versus the patch. This can be understood by looking at the general shape of the lane boundary and the patch. Moving in order of increasing index (from left to right), for the lane boundary the first edge encountered is a transition from dark to bright. This is the opposite of the general shape of K and hence gives a large negative value for the cross-correlation. The next edge encountered is a transition from bright to dark and hence matches the general shape of K very well giving a large positive value for the cross-correlation. For the patch the same two edges occur but in a reversed order. This provides a key distinction between the lane boundary and the patch and hence it makes sense to start looking for both types of edges with the hope of combining the results in a way which identifies the lane boundary but not the patch. Define K_L to be $\{1, 1, 0, -1 - 1\}$ and K_R to be $\{-1, -1, 0, 1, 1\}$. The cross-correlation of both of these kernels with X is shown in figure 2.5. As expected the order of the maxima between the two cross-correlations is reversed for the patch versus the lane boundary.

The next step is to combine $K_L \star X$ and $K_R \star X$ in a way which exploits the flipped order of maxima and minima to select out only the lane boundary. The trick is to shift the two cross-correlations relative to each other so that the maxima overlap for the lane boundary and the minima overlap for the patch. This can be done by prepending 3 zeros to K_L and appending 3 zeros to K_R. Denote the augmented kernels by $K_{L\text{aug}} = \{0, 0, 0, K_L\}$ and $K_{R\text{aug}} = \{K_R, 0, 0, 0\}$. The cross-correlation of these augmented kernels as well as the intersection of the two outputs is shown in figure 2.6. As can be seen, the intersection has only one maxima and it is near the lane boundary. However, this maxima does not line up with the lane boundary. To

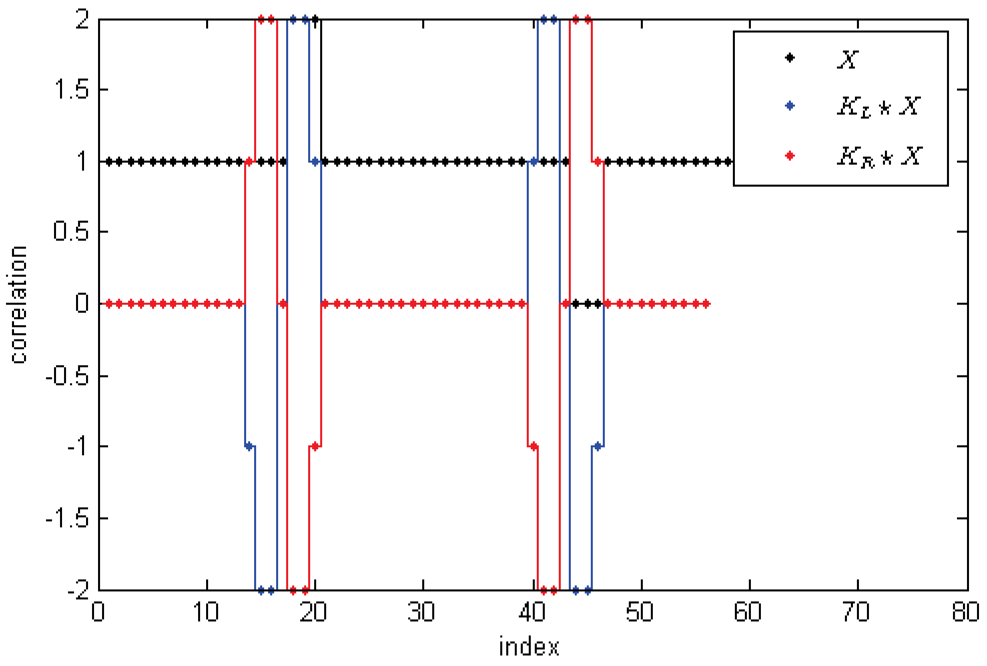

Fig. 2.5. Cross-correlation of K_L and K_R with X.

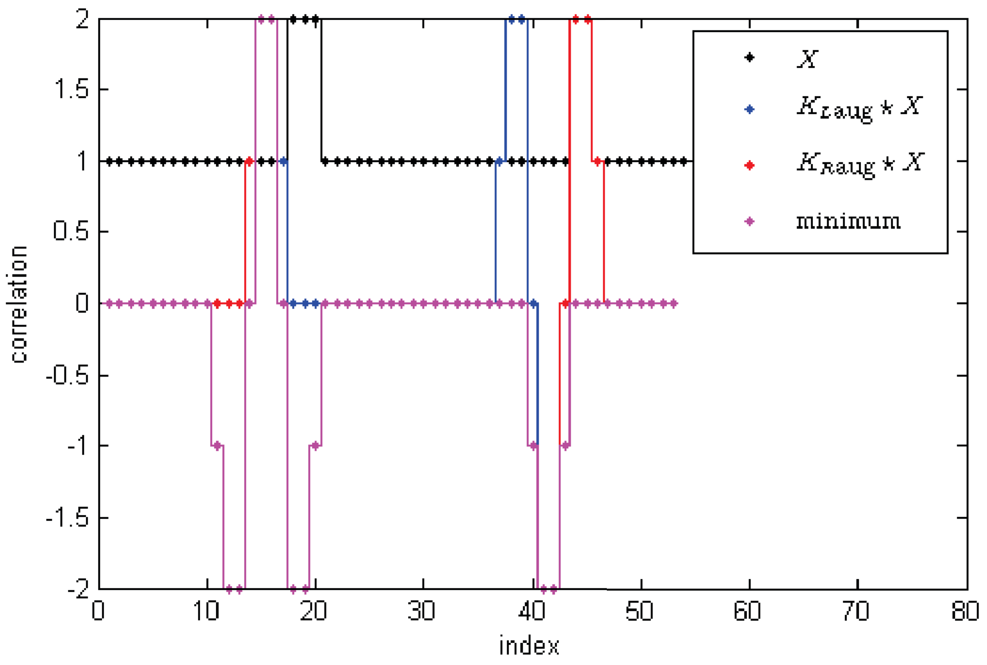

Fig. 2.6. Cross-correlation of the augmented matrices with X.

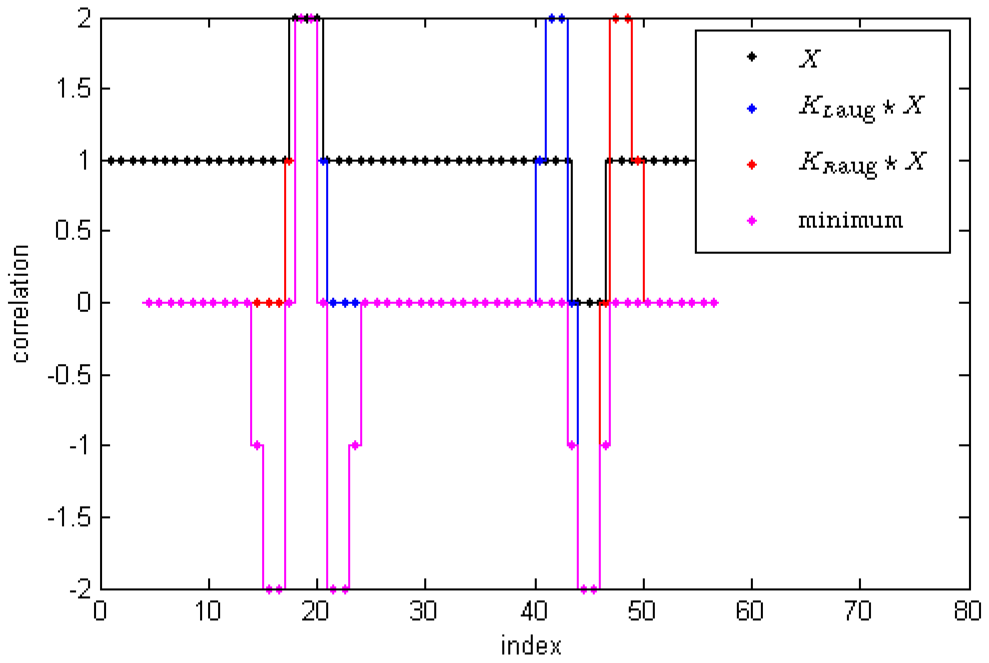

Fig. 2.7. Shifted cross-correlation of the augmented matrices with X.

correct this, the final output will be shifted to the right by 3.5 samples as shown in figure 2.7. This completes the example.

The 2D cross-correlation of a kernel $K \in \mathcal{R}^{m \times n}$ with a matrix $X \in \mathcal{R}^{p \times q}$, denoted $K \star X$, is a matrix $W \in \mathcal{R}^{(m-p+1)\times(n-q+1)}$ with entries defined as,

$$W(i, j) = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} K(a, b)X(i + a, j + b) \tag{2.2}$$

where $W(i, j)$, $K(i, j)$, and $X(i, j)$ denote the entry in the $(i + 1)$ row and $(j + 1)$ column of W, K, and X respectively. Just like for the 1D case, each entry in the 2D cross-correlation indicates how well the shape of K matches the shape of an $m \times n$ subsection of X starting at the ith row and jth column with a larger value indicating a better match. Again, this is exactly the behavior needed to check if a portion of the image contains a lane boundary.

From looking at figure 2.2 it is clear that the desired 2D shape is just the 1D shape (a single row) stacked vertically. Hence the 2D kernels, $K_{L,2D}$ and $K_{R,2D}$ can be constructed by simply stacking several copies of K_L and K_R. The optimal number of rows should be large enough to provide good robustness against non-lane boundary objects but not so large that the detection of lane boundaries fails under poor road conditions. An object which matches the pattern for a length of $\approx$1m (which is five rows) seems likely to be a lane boundary. This gives the final kernels used,

$$K_{L,2D} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \end{bmatrix} \tag{2.3}$$

$$K_{R,2D} = \begin{bmatrix} -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

Finally, to determine whether or not a point of interest p in the xz-plane of the vehicle coordinate system is on a lane boundary, both kernels are applied to an $m \times n$ subsection of the image with p in the center. Note that each cross-correlation will return a scalar value indicating how well the shape of the kernel matched that particular $m \times n$ subsection of the image. When both results exceed a threshold the conclusion is made that p is on a lane boundary.

2.4.2 Comparison with other correlation techniques

A two kernel approach has significant advantages over a single kernel using either normalized or unnormalized cross-correlation. This is most easily shown through an example. Figures 2.8-2.10 The black line represents the brightness of part of an image along a horizontal cross section with a lane boundary at 12 and a shadow on the right side of the image starting at 24. For the two kernel method using a single row of the

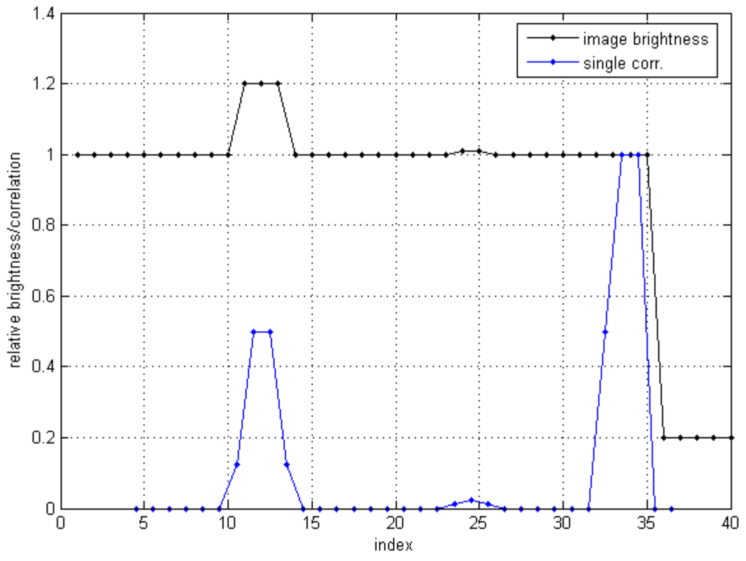

Fig. 2.8. Single unnormalized cross-correlation applied to the brightness levels of an image.

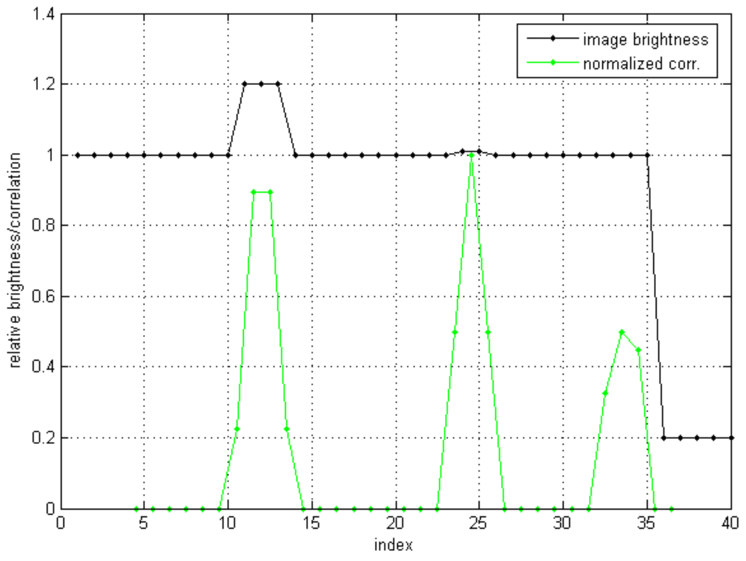

Fig. 2.9. Single normalized cross-correlation applied to the brightness levels of an image.

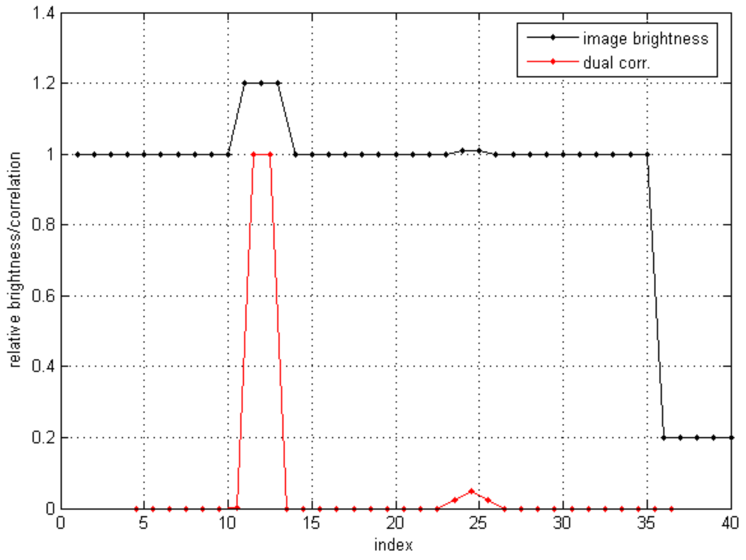

Fig. 2.10. Dual correlation applied to the brightness levels of an image.

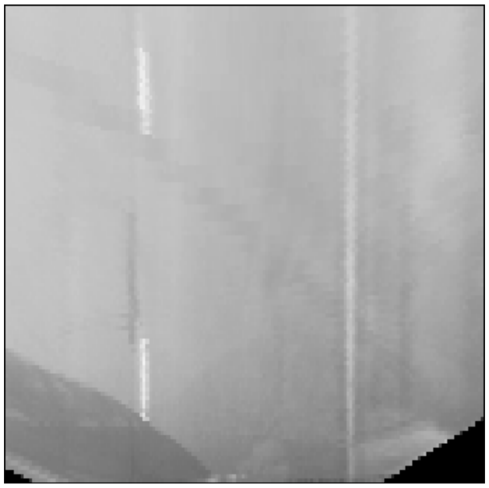

Fig. 2.11. An example image which has been remapped according to the method described in section 2.5.

appropriate kernels, $K_{1L} = \{-1, -1, 0, 1, 1, 0, 0, 0\}$ and $K_{1R} = \{0, 0, 0, 1, 1, 0, -1, -1\}$ were applied separately to the the brightness data and then the minimum value was chosen element by element resulting in the red line on the figure. These two kernels were then added to give a single kernel $K_{1C} = \{-1, -1, 0, 2, 2, 0, -1, 1\}$ which when applied to the brightness data gave the result shown. All results were normalized to a maximum value of 1 to make comparison easy and negative values were ignored.

Notice that only the dual correlation method identifies the lane boundary better than the other disturbances in the image. Specifically, unnormalized cross-correlation is susceptible to the dark area in the image as discussed in the preceding section. Normalized cross-correlation has improved robustness against the dark area in the image but is susceptible to small bright spots. Because of the normalization, the absolute magnitude information in the image is destroyed. This makes it impossible to distinguish low-contrast blips in the image from the higher contrast lane boundaries.

Figures 2.11-2.14 show a real-life example image and the relative correlations of each of the three methods that have been considered. Again, the dual correlation method shows the best performance.

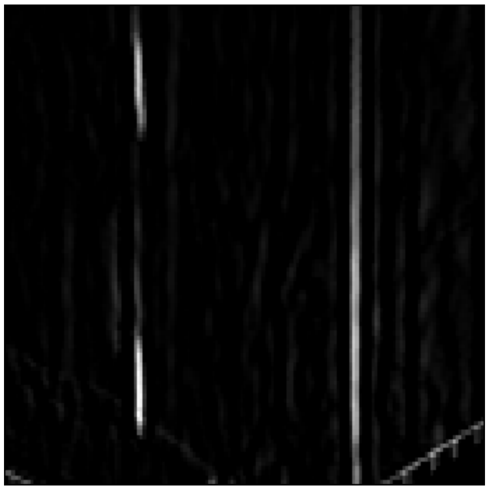

Fig. 2.12. Single unnormalized cross-correlation applied to the example image.

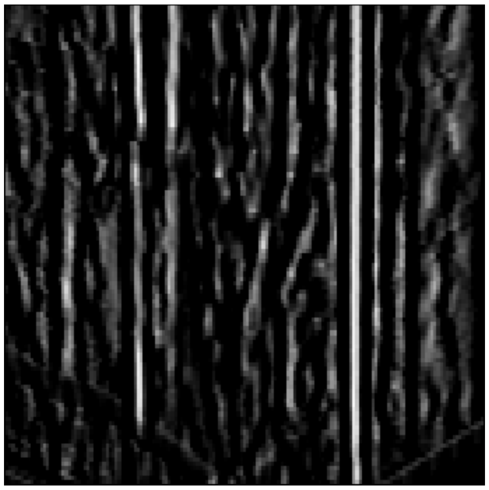

Fig. 2.13. Single normalized cross-correlation applied to the example image.

Fig. 2.14. Dual correlation applied to the example image.

2.4.3 Tracking lane boundaries

With the capability in place to determine whether or not a point is on a lane boundary the actual tracking of the lane boundaries is straightforward. Rather than track the entire lane boundary, which is computationally prohibitive, six points grouped as three pairs are tracked on each side of the image. The points are grouped into pairs to ensure maximum coverage of the line and also to reduce the computational requirements as will be explained later. The image is segmented vertically into three equal sections and each pair is constrained to operate only within its section. Three sections are used because this guarantees complete coverage of up to two dashed lane boundaries.

The first step is to check each point to see if the lane boundary is near where it was in the last image (for the first image, reasonable initial values are set). This is done by checking a small rectangle around the point using the lane boundary identification algorithm described earlier.

For each point that is not still near a lane boundary, one of two things occurs. If the point's associated buddy is on a lane boundary then the points are overlapped, otherwise the entire region is searched. After this process any points which are still not on a lane boundary are ignored.

In order to maximize the coverage of the lane boundaries, each point is now moved in small increments along the lane boundary, with the two points in a pair moving in opposite directions until the lane boundary is no longer detected or the edge of the section is reached. The last successful detection is taken to be the final location for that point.

With the general area containing the lane boundary identified, it is desirable to obtain a reasonable estimate of the center of the lane boundary. Note that if the lane boundary is not exactly 2 pixels wide, e.g. figure 2.10, the maximum value of the dual correlation may not correspond to the center of the lane boundary. Hence, while the maximum value in a small region could be used, better methods probably exist.

For example, the average location of all dual correlations exceeding a threshold could be used. For this paper, the method used is the center of mass of the positive dual correlation in a small region around the maximum point.

As a final verification, for each buddy pair the slope of the line connecting the points is checked to make sure it is reasonable. If the angle away from vertical is greater than 3 degrees, whatever was detected is not a lane boundary so both points are thrown out. Based on experimental testing of typical angles during highway driving the angle of the lane boundaries is not likely to exceed 2 degrees so 3 degrees represents a safe upper bound.

2.5 Conversion of the Image Plane to the Vehicle Reference Frame

The image processing algorithm of the preceding section identifies the coordinates of several points in the xz-plane of the vehicle coordinate system $\mathcal{X}_v = \mathcal{R}^3$ which are on the lane boundaries. In this section the preparation of the input data for this algorithm will be presented by first identifying the necessary mappings and then developing the tools to efficiently implement these mappings. In the following section, the output of the image processing algorithm will be used to identify the relationship between the roadway and vehicle coordinate systems which in turn gives the position and orientation of the vehicle in the roadway coordinate system as required by the control.

The input to the system comes through a camera which can be modelled as a projection of the brightness of all points within its field of view to the image plane $\mathcal{X}_i = \mathcal{R}^2$, which is the xy-plane of the camera coordinate system $\mathcal{X}_c = \mathcal{R}^3$. In practice the camera combines this projection with a uniform sampling of the brightness of the projected data; however, we will ignore this sampling and assume continuous data until section 2.5.2.

The spatial point which each coordinate in the image plane corresponds to cannot be determined as the projection is not invertible. However, by assuming that the camera has a clear view of the roadway, i.e. that all of the brightness data has been collected from the road plane, a 1-1 mapping can be established between the coordinates in $\mathcal{X}_i$ with the coordinates of the represented spatial point in any 3D coordinate system. Because the image processing algorithm operates on coordinates in $\mathcal{X}_v$, of particular interest will be the mapping between coordinates in $\mathcal{X}_v$ and coordinates in $\mathcal{X}_i$

$$f_M : \mathcal{X}_v \rightarrow \mathcal{X}_i \tag{2.5}$$

f_M is derived in section 2.5.1.

Because the image processing requires sampled brightness information on a grid of uniformly spaced coordinates in the xz-plane of $\mathcal{X}_v$ we set up this grid of coordinates and then map them using f_M to a grid of coordinates in $\mathcal{X}_i$. By sampling the brightness of the image at these coordinates in $\mathcal{X}_i$ we are directly measuring the brightness at the desired coordinates in $\mathcal{X}_v$. Unfortunately f_M is a nonlinear mapping which means the uniform grid of coordinates in $\mathcal{X}_v$ maps to a non-uniform grid of coordinates in $\mathcal{X}_i$. This conflicts with the way in which the camera collects data. This disparity will be resolved in section 2.5.2.

The specific linear grid of coordinates in $\mathcal{X}_v$ used is chosen to balance the need for high resolution data in order to distinguish lane boundaries from other objects and the limitations of realtime operation. Because the lane boundaries are 10 - 15 cm wide an x interval of 5 cm guarantees at least two samples per lane boundary. A z interval of 0.2m guarantees at least fifteen samples on each 3.4m long dashed line. These intervals have been experimentally successful.

The bounds for the grid also need to be determined. The hood limits the closest z-coordinate to about 4m in front of the vehicle. The resolution of the camera limits the farthest z-coordinate to around 24m from the camera. The x-coordinates are constrained to lie within a little more than one lanewidth to either side of the vehicle (approximately $\pm$ 4m).

2.5.1 Coordinate transformations

Throughout the remainder of this section homogeneous coordinates will be used. Homogeneous coordinates express a coordinate in $\mathcal{R}^3$ as a coordinate in $\mathcal{R}^4$ using the mapping

$$f_H : (x, y, z) \rightarrow (xw, yw, zw, w) \tag{2.6}$$

This mapping is not unique since the choice of w is arbitrary, although w is often set to 1. f_H^{-1} always exists and is obtained by dividing each coordinate by the last coordinate and then dropping the last coordinate. Note that f_H generalizes to express a coordinate in $\mathcal{R}^n$ as a coordinate in $\mathcal{R}^{n+1}$. A superscript H is used to denote coordinate systems composed of homogeneous coordinates.

The advantage of homogeneous coordinates is that they allow a translation of the coordinate system to be expressed as an invertible linear operator, i.e. there always exists an invertible linear mapping $f_T : X_A^H \rightarrow X_B^H$ if X_B can be obtained from X_A through a translation. In matrix form, if X_T is obtained from X_A through translations by x_T, y_T, and z_T along the x, y, and z axes of X_A respectively then the coordinate of a point p in X_T^H is given by

$$p_T^H = \begin{bmatrix} 1 & 0 & 0 & x_T \\ 0 & 1 & 0 & y_T \\ 0 & 0 & 1 & z_T \\ 0 & 0 & 0 & 1 \end{bmatrix} p_A^H \tag{2.7}$$

where p_A^H is the coordinate of p in X_A^H. Note that this matrix is always invertible with the inverse given by changing the sign on x_T, y_T, and z_T.

Invertible linear mappings also exist for rotating a coordinate system around the x, y, and z axes. If X_B is obtained from X_A through a rotation then there exists a mapping

$$f_R : X_A^H \rightarrow X_B^H, \tag{2.8}$$

which will map the coordinate of a point from X_A^H to X_B^H. In matrix form, if p_A^H is the coordinate of a point in X_A^H then the coordinate of the point in coordinate systems $X^H_{R_X}$, $X^H_{R_Y}$, and $X^H_{R_Z}$ obtained through a rotation around the x, y, and z axes respectively is given by

$$p^H_{R_X} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p^H_A, \tag{2.9}$$

$$p^H_{R_Y} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p^H_A, \tag{2.10}$$

$$p^H_{R_Z} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p^H_A \tag{2.11}$$

where θ is the angle of rotation. Note that all three matrices are invertible with the inverse given by changing the sign on θ.

With this framework in place, the relation between the various coordinate systems can be precisely specified by giving the translation and rotation mappings between the coordinates for a common point in the two coordinate systems. The roadway coordinate system maps to the vehicle coordinate system through a rotation around the y-axis followed by a translation along the x-axis. Hence a point p with coordinate p^H_r in X^H_r has the coordinate p^H_v in X^H_v given by

$$p^H_v = f_{T_X}(f_{R_Y}(p^H_r)) \tag{2.12}$$

where f_{T_X} is used to indicate that the translation is only along the x axis. The mappings f_{T_X} and f_{R_Y} are unknown and are estimated in section 2.6 in order to determine the position and orientation of the vehicle in the roadway coordinate system.

The vehicle coordinate system maps to the camera coordinate system in much the same way, except in this case the rotation is about all three axes and the translation is along the y-axis. A point p with coordinate p_v^H in the vehicle coordinate system has the coordinate p_c^H in the camera coordinate system given by

$$p_c^H = (f_T \circ f_{R_Z} \circ f_{R_Y} \circ f_{R_X})(p_v^H) \tag{2.13}$$

Because the camera is rigidly attached to the vehicle these mappings are constant and can be determined from either the geometry of the camera setup or through a calibration procedure. Since the mappings remain constant it is helpful to combine them into a single map

$$f_{cv} = f_{T_Y} \circ f_{R_Z} \circ f_{R_Y} \circ f_{R_X}. \tag{2.14}$$

As mentioned above, the function of a camera is to perform a projection from X_c^H to X_i^H and is described by the non-invertible mapping $f_P : X_c^H \rightarrow X_i^H$. In matrix form, a point with coordinate p_c^H maps to a coordinate in X_i^H given by

$$p_i^H = \begin{bmatrix} \frac{1}{\tan(u)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(v)} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} p_c^H \tag{2.15}$$

where u and v are fixed camera parameters. Note that unlike the translation and rotation matrices, the projection matrix is not invertible. This is because f_P maps a four dimensional system to a three dimensional system.

The mapping $f_M : \mathcal{X}_v \rightarrow \mathcal{X}_i$ can now be specified as a chain of mappings:

$$f_M = f_H^{-1} \circ f_P \circ f_{cv} \circ f_H. \tag{2.16}$$

2.5.2 Resampling

The B/W camera uniformly samples the brightness of the projection onto X_i resulting in both a spatial and magnitude quantization of the brightness giving an 8-bit array of brightness data denoted Q_i. This process is denoted by the mapping $f_Q : (X_i \times \mathcal{R}) \rightarrow Q_i$.

Recall from the beginning of this section that we require the brightness on a nonuniform grid on X_i. To approximate the required data an interpolation is used between the required coordinates in X_i and those available in Q_i as expressed by the mapping $f_I : (X_i, Q_i) \rightarrow \mathcal{B}$ where X_i can be any coordinate of interest in the image plane and $\mathcal{B}$ is the approximate brightness at that coordinate with values in the interval $[0, 255]$. A convenient choice for f_i is nearest neighbor interpolation in which the coordinate in Q_i closest to the coordinate X_i is selected and the associated 8-bit brightness value is used. This mapping method gives good results and is computationally the fastest interpolation method available.

2.6 Resolution of the Vehicle and Roadway Reference Frames

The algorithm of section 2.3 is used to identify and track the coordinates of several points on a lane boundary in the vehicle coordinate system $\mathcal{X}_v$. Once estimated, the goal is to use these coordinates to identify the position and orientation of the vehicle in the roadway coordinate system $\mathcal{X}_r$. Recall that the position and orientation of the vehicle is represented by the positive z-axis in $\mathcal{X}_v$ (c.f. figure 2.1). Therefore, if we can estimate the linear mapping between coordinates in $\mathcal{X}_v$ and $\mathcal{X}_r$ we can use this mapping to obtain an estimate for the position and orientation of the vehicle in $\mathcal{X}_r$. Specifically, an estimate is found for the linear mapping of coordinates from $\mathcal{X}_v$ to $\mathcal{X}_r$ using homogeneous coordinates, i.e. $f_{rv}^H : \mathcal{X}_v^H \rightarrow \mathcal{X}_r^H$. Then f_{rv}^H is applied to a vector along the z-axis of $\mathcal{X}_v$ which generates a vector in $\mathcal{X}_r$ representing the position and orientation of the vehicle. As will be seen, f_{rv}^H is represented by a 4x4 nonsingular matrix and hence is well defined and linear.

Estimating f_{rv}^H is greatly simplified by considering the geometry of the problem. f_{rv} is constrained to be a rotation about the y-axis and a translation along the x-axis of $\mathcal{X}_r$. Because of this, any coordinate (or vector) in the xz-plane of $\mathcal{X}_r$ maps to a coordinate (or vector) in the xz-plane of $\mathcal{X}_v$. Furthermore, because the linear mapping f_{rv}^H has only two degrees of freedom, finding the mapping between a single vector in

$\mathcal{X}_r$ and the corresponding vector in $\mathcal{X}_v$ is sufficient to fully specify f_{rv}^H. Recall that in $\mathcal{X}_r$ the center of the lane is along the z-axis. Also, a vector representing the center of the lane in $\mathcal{X}_v$ can be estimated using the lane boundaries. Hence the center of the lane in each coordinate system is used to estimate the mapping f_{rv}^H.

All that remains is to find a robust method for estimating the center of the lane in $\mathcal{X}_v$. The center of the lane will be estimated in slope-intercept form. First the x-intercept and the slope relative to the z-axis for both lane boundaries is determined using a least squares fit through the data points identified to be on each lane boundary. To determine the slope of the center of the lane the slopes for each lane boundary will be combined using a weighted average, i.e. the slope of the center of the lane is estimated by

$$\hat{m}_C = a_m \hat{m}_L + (1 - a_m)\hat{m}_R. \tag{2.17}$$

where $\hat{m}_L$ and $\hat{m}_R$ are the estimated slopes of the left and right lane boundaries respectively and $0 \leq a_m \leq 1$ is the weighting factor given by

$$a_m = \frac{\sigma_{m_R}^2}{\sigma_{m_L}^2 + \sigma_{m_R}^2}. \tag{2.18}$$

The terms $\sigma_{m_R}^2$ and ${}_{m_L}^2$ are explained in the derivation of this equation given in Appendix A.1.

Similarly, the x-intercept of the two lane boundaries can be used to estimate the x-intercept of the center of the lane. To do this the estimated x-intercept of each lane boundary is first shifted towards the center of the lane by half of the estimated lanewidth $\hat{w}$. This generates two independent estimates for the x-intercept of the center of the lane. The shifted intercepts are then combined using a weighted average, i.e. the x-intercept of the center of the lane is estimated by

$$\hat{b}_C = a_b \left(\hat{b}_L - \frac{\hat{w}}{2} \right) + (1 - a_b) \left(\hat{b}_R + \frac{\hat{w}}{2} \right) \tag{2.19}$$

where $\hat{b}_L$ and $\hat{b}_R$ are the x-intercepts of the left and right lane boundaries respectively and $0 \leq a_b \leq 1$ is the weighting factor given by

$$a_b = \frac{1}{2} \left(\frac{2\sigma_{b_R}^2 + \sigma_w^2}{\sigma_{b_L}^2 + \sigma_{b_R}^2 + \sigma_w^2} \right). \tag{2.20}$$

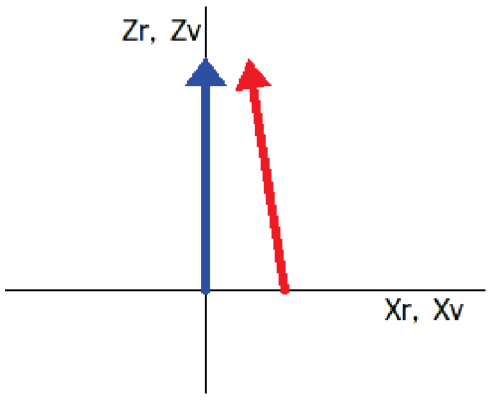

Fig. 2.15. Center of the lane in the roadway (blue) and vehicle (red) coordinate systems.

The terms $\sigma^2_{b_R}$, $\sigma^2_{b_L}$, and σ^2_w are explained in the derivation of this equation given in Appendix A.2.

With the computed estimated slope $\hat{m}_C$ and intercept $\hat{b}_C$ of the center of the lane in $\mathcal{X}_v$, f^H_{rv} can be determined. Recall that f^H_{rv} consists of a rotation about the y-axis and a translation along the x-axis of $\mathcal{X}_r$. Also, the center of the lane in $\mathcal{X}_r$ is along the z-axis. Figure 2.15 shows the center of the lane in both coordinate systems on a common set of axes. From the figure it can be seen that the center of the lane in $\mathcal{X}_v$ can be made to overlap the center of the lane in $\mathcal{X}_r$ by translating it along the x-axis by $-\hat{b}_C$ and then rotating it about the y-axis by $\theta_C = -\tan^{-1}(\hat{m}_C)$. The matrix forms for translation and rotation from the previous section accomplish

$$f^H_{rv} = \begin{bmatrix} \cos(\theta_C) & 0 & \sin(\theta_C) & -\hat{b}_C\cos(\theta_C) \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_C) & 0 & \cos(\theta_C) & \hat{b}_C\sin(\theta_C) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.21}$$

Because the position and orientation of the vehicle in $\mathcal{X}_v$ is given by the positive z-axis, the position and orientation of the vehicle can be found in $\mathcal{X}_r$ by mapping just two points: the origin of $\mathcal{X}_v$ and one additional point along the positive z-axis. Specifically the coordinates $[0,0,0,1]^T$ and $[0,0,1,1]^T$ in $\mathcal{X}_v^H$ are mapped to $\mathcal{X}_r^H$:

$$f_{rv}^H \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\hat{b}_C \cos(\theta_C) \\ 0 \\ \hat{b}_C \sin(\theta_C) \\ 1 \end{bmatrix} \tag{2.22}$$

$$f_{rv}^H \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -\hat{b}_C \cos(\theta_C) + \sin(\theta_C) \\ 0 \\ \hat{b}_C \sin(\theta_C) + \cos(\theta_C) \\ 1 \end{bmatrix} \tag{2.23}$$

Hence the angle of the vehicle relative to the z-axis of $\mathcal{X}_r$ is

$$\theta = tan^{-1}\left(\frac{\sin(\theta_C)}{\cos(\theta_C)}\right) = \theta_C. \tag{2.24}$$

The x-coordinate of the position of the vehicle in $\mathcal{X}_r$ is

$$x = -\hat{b}_C \cos(\theta_C) \approx -\hat{b}_C \tag{2.25}$$

where the approximation holds within 1cm. even under the extreme scenario when θ_C is 4 degrees and $\hat{b}_C$ is 3.5m suggesting reasonable robustness.

2.7 Experimental Results

The experimental setup described in the introduction was used to evaluate the performance of the methods described in this paper. Unfortunately, the nature of the problem we are solving makes it difficult to determine the error for a particular measurement. Furthermore, the broad range of conditions encountered on roads makes it difficult to even estimate limits of performance. Nonetheless, we present results which speak to the reasonableness of the proposed solution.

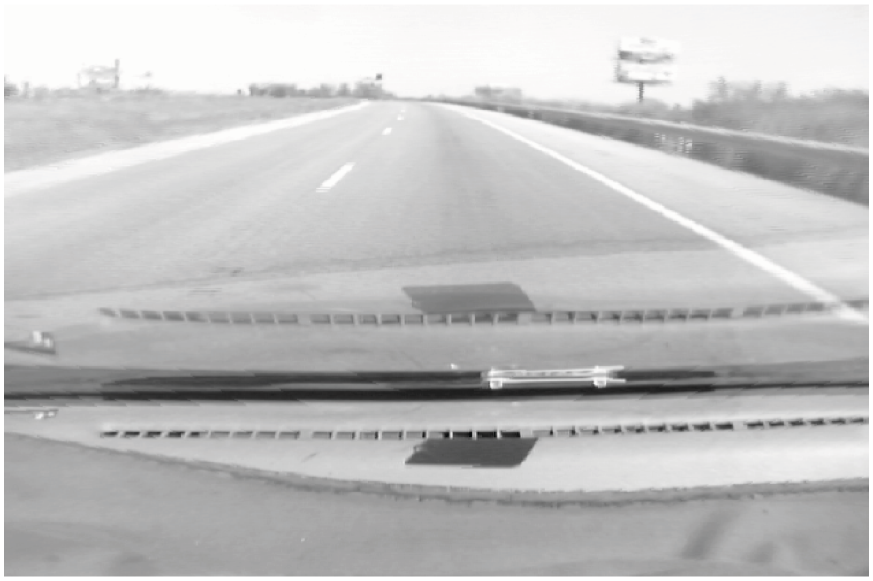

Fig. 2.16. Image of a gentle curve as captured by the camera.

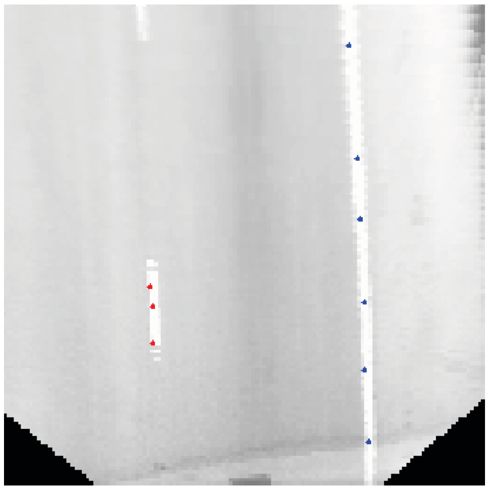

Fig. 2.17. Remapped image of a gentle curve with the tracked points on each lane boundary marked.

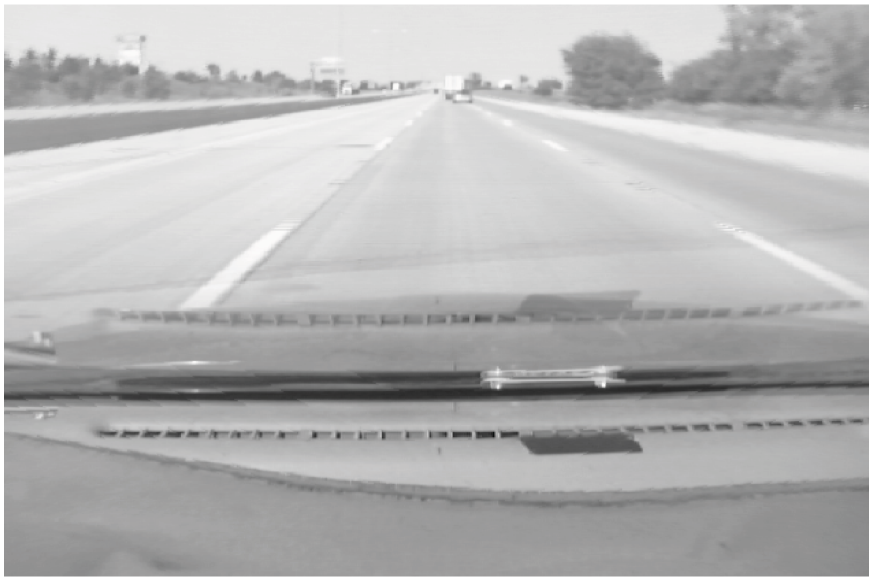

Fig. 2.18. Image of a multilane highway as captured by the camera.

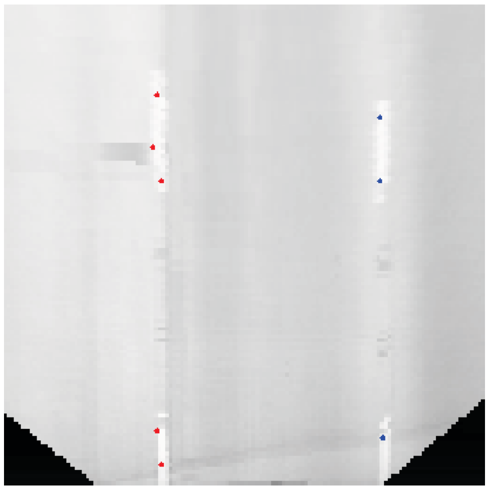

Fig. 2.19. Remapped image of a multilane highway with the tracked points on each lane boundary marked.

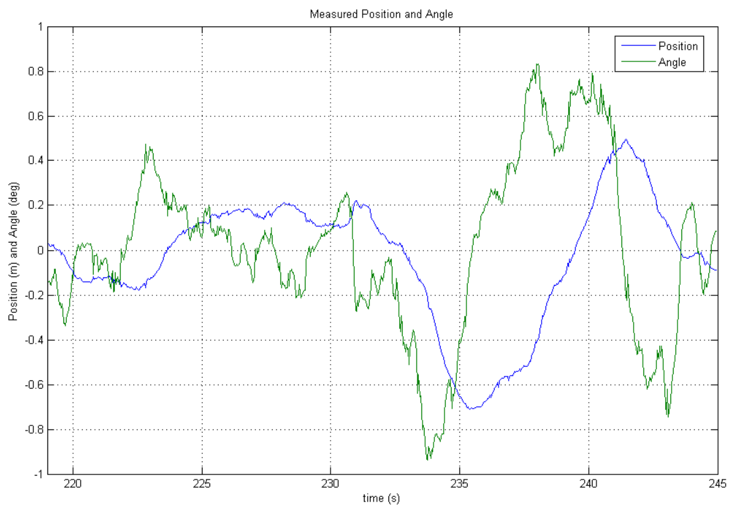

Fig. 2.20. Position (blue) and angle (green) of the vehicle in the roadway coordinate system during an aggressive driving maneuver.

Figures 2.16 and 2.18 show two images as captured by the camera. Figures 2.17 and 2.19 show the corresponding representations of these images in the vehicle coordinate system as well as the points identified to be on a lane boundary by the tracking algorithm. These figures demonstrate both the reasonableness of doing image processing in the vehicle coordinate system as well as the ability of the tracking algorithm to correctly identify the lane boundaries. Figure 2.20 shows the measured position and angle of the vehicle during an aggressive driving sequence. This data gives an indication of the relative smoothness of the two measurements and also of the robustness of the tracking algorithm. Note that these are direct measurements, before any filtering is applied.

Figures 2.21 and 2.22 show measurements for the same diving maneuver as figure 2.20 with only the dashed or solid line available. These figures also show the differences between the single lane boundary (dashed or solid) and two lane boundaries cases. These differences may be taken as representative of the error in the measurement when one of the lane boundaries is missing. As long as one solid lane boundary is available the performance is nearly the same as the two lane boundaries case. When

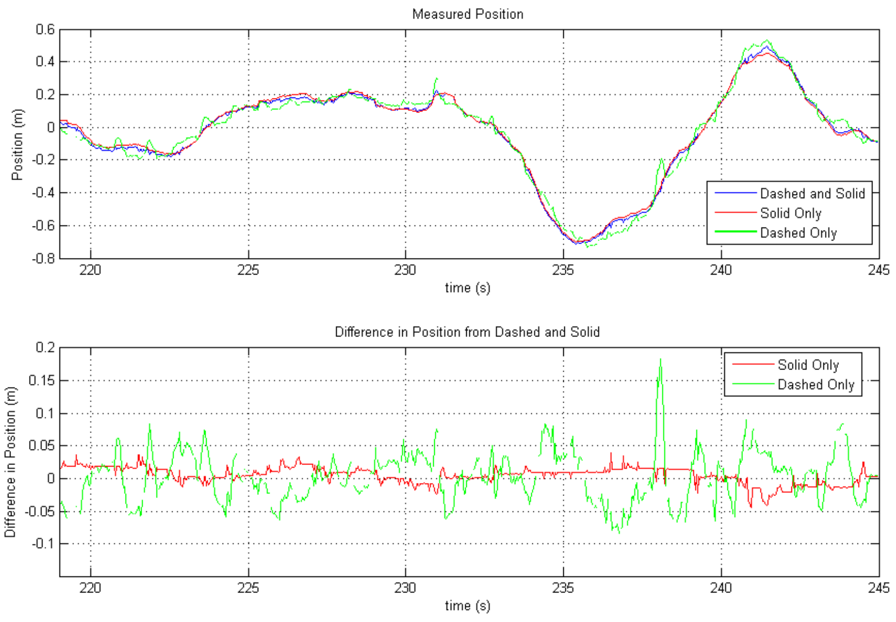

Fig. 2.21. Position of the vehicle during the above driving maneuver with limited lane boundaries.

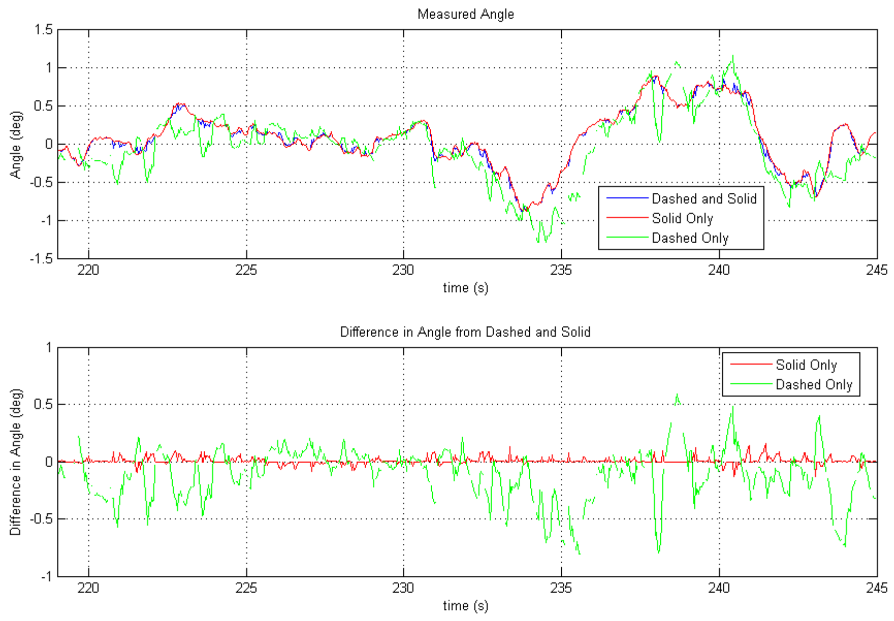

Fig. 2.22. Angle of the vehicle during the above driving maneuver with limited lane boundaries.

only a dashed lane boundary is available there is a significant increase in the error on both measurements however with the filtering techniques described in the next chapter the measurements are still sufficiently accurate for control.

3. MODELING AND STATE ESTIMATION

Controlling the steering of the vehicle using the output of the image processing algorithm of the preceding chapter is a two step process. First, recall that the output of the image processing algorithm is a noisy measurement of the position and orientation of the vehicle. The image processing algorithm also outputs an estimated variance on each of these measurements. Let us definte the state of the system $x \in \mathcal{R}^2$ to be $x = [p,\ \theta]^T$ where p is the horizontal position of the vehicle and θ is the angle of the vehicle relative to the center of the lane. A Kalman-Bucy filter can be used to combine the measured state with a model of the vehicle in a way which produces a more accurate (filtered) estimate of the vehicle state. By expanding the state vector to account for measurement bias in the steering wheel angle due to environmental/vehicle factors, the Kalman-Bucy filter can also be used to estimate this bias state.

Second, with these refined estimates of position, angle, and bias, LQR control is used to compute an appropriate steering wheel angle which is then commanded. The focus of this chapter is to estimate the expanded state of the vehicle, the first step of the process. The next chapter explains and gives experimental results for the LQR control.

A vehicle model incorporating the states appropriate to this project is developed in the first section. This model is augmented with a bias on the steering wheel angle in 3.2 and the resulting model is used in the Kalman-Bucy filter described in section 3.3. Finally, some experimental results are given in the last section.

3.1 Modeling the Vehicle

A simple kinematic model for the vehicle is [5]:

$$\begin{bmatrix} \dot{p} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v(t)\sin(\theta) \\ \frac{v(t)}{L}\tan(\delta) \end{bmatrix} \tag{3.1}$$

where p is the horizontal position and θ is the angle of the vehicle both relative to the center of the lane, $v(t)$ is the speed of the vehicle, L is the wheelbase of the vehicle, and δ is the angle of the front wheels.

In practice θ and δ are both limited to a few degrees so the approximations $\sin(\theta) \approx \theta$ and $\tan(\delta) \approx \delta$ can be made. These approximations yield the linearized state model

$$\begin{bmatrix} \dot{p} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & v(t) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{v(t)}{L} \end{bmatrix} \delta \tag{3.2}$$

δ can be approximated for a particular vehicle as $\alpha\phi$ where α is a constant dependent on the vehicle and ϕ is the steering wheel angle. If we let $\hat{\alpha} = \frac{\alpha}{L}$ then the linearized time-varying state model becomes

$$\begin{bmatrix} \dot{p} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & v(t) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ \hat{\alpha}v(t) \end{bmatrix} \phi \tag{3.3}$$

Discretizing the system using a zero-order hold with a sample time of T and assuming that $v(t) = v_k,\ kT \leq t \leq (k+1)T$ gives the discrete (non-stationary) linear state model

$$\begin{bmatrix} p_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & v_kT \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\hat{\alpha}v_k^2T^2 \\ \hat{\alpha}v_kT \end{bmatrix} \phi_k = A_k \begin{bmatrix} p_k \\ \theta_k \end{bmatrix} + B_k\phi_k \tag{3.4}$$

In this description T is dictated by the frame-rate of the images produced by the camera, which in this case is 30fps giving $T \approx 33ms$.

3.2 Adding a Bias State

A large number of environmental factors influence the vehicle system, violating the model of 3.4. The biggest such factor is the curvature of the road. Other factors which exhibit similar effects on the system include the slope of the road, poor vehicle alignment, and steady crosswinds. Unfortunately, these environmental factors in general cannot be known by the system (some can be estimated but with insufficient accuracy in the current implementation to be useful). However, if these outside disturbances can be characterized in a simple way which can then be added to the model, a Kalman-Bucy filter can be applied to the augmented model to produce an estimate of the combined influence of these environmental factors.

It should be noted that the goal is not to obtain an accurate estimate of any of the environmental factors in particular or a complete characterization of their influence on the system. Rather, the goal is to find a simple addition to the model which characterizes the dominant thrust of their combined influence and which can be easily corrected for in the control. Based on this goal, one characterization which has proven effective in practice is to model these environmental factors as a bias in the steering wheel which is nearly constant compared to the sampling rate of the system. This approach was chosen for two reasons. First, this approach is consistent with how a human driver responds to the environmental factors listed above. For example, a flat curve in the road can typically be followed using a constant steering wheel bias (at a constant speed, $\phi = \text{constant} \Rightarrow \dot{\theta} = \text{constant} \Rightarrow$ circular path.) Second, the estimated steering wheel bias is easily compensated for in the control with a simple change. Hence, as a crude model, these influences are combined into a bias term b on the steering wheel which affects the system as follows

$$\begin{bmatrix} p_{k+1} \\ \theta_{k+1} \end{bmatrix} = A_k \begin{bmatrix} p_k \\ \theta_k \end{bmatrix} + B_k(\phi_k + b_k) \tag{3.5}$$

$$= A_k \begin{bmatrix} p_k \\ \theta_k \end{bmatrix} + [B_k \; B_k] \begin{bmatrix} \phi_k \\ b_k \end{bmatrix} \tag{3.6}$$

Equation 3.6 now incorporates the additional bias input b into the model. However, this formulation suggests that b is a controllable input and known which is not true.

Rather, the goal is to use a Kalman-Bucy filter to obtain an estimate of b. Hence it is necessary to shift b from an input to a state giving the discrete state dynamics

$$\begin{bmatrix} p_{k+1} \\ \theta_{k+1} \\ b_{k+1} \end{bmatrix} = \begin{bmatrix} A_k & B_k \\ 0 & \beta \end{bmatrix} \begin{bmatrix} p_k \\ \theta_k \\ b_k \end{bmatrix} + \begin{bmatrix} B_k \\ 0 \end{bmatrix} \phi_k \tag{3.7}$$

Ideally speaking, at least for small sections of the roadway, the bias term b_k is relatively constant (consistent with the biases described above, all of which are relatively constant compared to the sampling rate of the system) and thus we can presume that from discrete sample to discrete sample, $b_{k+1} = b_k$ and hence β in equation 3.7 may be taken as 1. This gives the fully expanded state model

$$\begin{bmatrix} p_{k+1} \\ \theta_{k+1} \\ b_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & v_k T & \frac{1}{2}\hat{\alpha} v_k^2 T^2 \\ 0 & 1 & \hat{\alpha} v_k T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_k \\ \theta_k \\ b_k \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\hat{\alpha} v_k^2 T^2 \\ \hat{\alpha} v_k T \\ 0 \end{bmatrix} \phi_k = \hat{A}_k \begin{bmatrix} p_k \\ \theta_k \\ b_k \end{bmatrix} + \hat{B}_k \phi_k \tag{3.8}$$

Although under this model the bias term is constant and cannot change, the Kalman-Bucy filter assumes that an additional unknown Gaussian noise input is driving every state. This additional input is shown in equation 3.11. This model can now be used in the Kalman-Bucy filter to give an estimate of b_k. It should be noted that the bias term is not strictly driven by Gaussian noise as the Kalman-Bucy filter assumes. However, because the system is observable, when the bias term is constant (and hence the model of 3.8 is accurate) and in the absence of noise, the Kalman-Bucy filter produces a discrete state observer which is guaranteed to converge to the true state. In practice, these conditions are not met (there is noise and the bias term is not strictly constant); however, experimentation has shown that under normal driving conditions the bias term converges quickly after step changes to the true bias and is resilient to noise.

3.3 Estimating the States

A Kalman-Bucy filter [6] is used to improve the estimates on x and θ as well as to estimate the disturbance state d. For ease of notation, let $\mathbf{z}_k = [x_k, \theta_k, b_k]^T$ be the states of the system and $\mathbf{y}_k = [\hat{x}_k, \hat{\theta}_k]^T$ be the output of the image processing algorithm.

The image processing algorithm provides estimates $\hat{p}_k$ and $\hat{\theta}_k$ according to

$$\mathbf{y}_k = \begin{bmatrix} \hat{p}_k \\ \hat{\theta}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{z}_k + \psi_k = C\mathbf{z}_k + \psi_k \tag{3.9}$$

where $\psi_k \in \mathcal{R}^2$ is a noise term modelled as a zero mean Gaussian random variable with covariance matrix

$$\Psi_k = \begin{bmatrix} \sigma^2_{x,k} & 0 \\ 0 & \sigma^2_{\theta,k} \end{bmatrix} \tag{3.10}$$

$\sigma^2_{x,k}$ and $\sigma^2_{\theta,k}$ are the estimated variance of the vehicle position and orientation respectively as determined by the image processing algorithm of the preceding section. Note that these measurements are uncorrelated and hence only the diagonal entries of Ψ_k are nonzero. Out objective is to recover z_k.

Furthermore, the state dynamics are augmented with a noise term $\xi \in \mathcal{R}^3$ which models the unknowns of the state dynamics, including unknown inputs to the system. Thus we arrive at a more complete state description of the process as

$$\mathbf{z}_{k+1} = \hat{A}_k \mathbf{z}_k + \hat{B}_k \phi_k + \xi_k \tag{3.11}$$

where ξ is modelled as a zero mean Gaussian random variable with covariance matrix Ξ. For the corresponding continuous time Gaussian random variable the covariance matrix $\Xi_c(\tau_1, \tau_2)$ is of the form

$$\Xi_c(\tau_1, \tau_2) = \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix} \delta(\tau_1 - \tau_2) \tag{3.12}$$

where q_1, q_2, and q_3 are vehicle dependent. The corresponding discrete time covariance matrix for a system with state transition matrix $\Phi(t_1, t_2)$ is given by [7]

$$\Xi_{d,k} = \int_{kT}^{(k+1)T} \int_{kT}^{(k+1)T} \Phi_k(\tau_1, (k+1)T) \Xi_c(\tau_1, \tau_2) \Phi_k^T(\tau_2, (k+1)T) \mathrm{d}\tau_2 \mathrm{d}\tau_1 \tag{3.13}$$

$$= \begin{bmatrix} q_1 T + \frac{1}{3} q_2 v_k^2 T^3 + \frac{1}{20} q_3 \hat{\alpha}^2 v_k^4 T^5 & \frac{1}{2} q_2 v_k T^2 + \frac{1}{8} q_3 \hat{\alpha}^2 v_k^3 T^4 & \frac{1}{6} q_3 \hat{\alpha} v_k^2 T^3 \\ \frac{1}{2} q_2 v_k T^2 + \frac{1}{8} q_3 \hat{\alpha}^2 v_k^3 T^4 & q_2 T + \frac{1}{3} q_3 \hat{\alpha}^2 v_k^2 T^3 & \frac{1}{2} q_3 \hat{\alpha} v_k T^2 \\ \frac{1}{6} q_3 \hat{\alpha} v_k^2 T^3 & \frac{1}{2} q_3 \hat{\alpha} v_k T^2 & q_3 T \end{bmatrix} \tag{3.14}$$

A general form for a state estimator which observes the state z_k of the system giving an estimated state $\bar{\mathbf{z}}_k$ is

$$\hat{\mathbf{z}}_k = \hat{A}_{k-1} \bar{\mathbf{z}}_{k-1} + \hat{B}_{k-1} \phi_{k-1} \tag{3.15}$$

$$\bar{\mathbf{z}}_k = \hat{\mathbf{z}}_k + L_k (\mathbf{y}_k - C \hat{\mathbf{z}}_k) \tag{3.16}$$

where L_k is the filter gain matrix. The Kalman-Bucy filter defines the optimal choice of L_k to be

$$L_{k,opt} = \left(S_k C^T\right) \left(\Psi_k + C S_k C^T\right)^{-1} \tag{3.17}$$

where S_k is the solution to the discrete algebraic Ricatti equation (DARE)

$$S_k = \left(\Xi_{d,k} + \hat{A}_k S_k \hat{A}_k^T\right) - \left(\hat{A}_k S_k C^T\right) \left(\Psi_k + C S_k C^T\right)^{-1} \left(C S_k \hat{A}_k^T\right) \tag{3.18}$$

Using $L_{k,opt}$ in the estimation equation 3.16 gives the desired optimal state estimates.

3.4 Experimental Results

Figure 3.1 shows the measured and estimated position of the vehicle during an aggressive driving sequence. Similarly, figure 3.2 shows the measured and estimated vehicle angle during the same sequence. The estimates for position track closely with the measurements because the variance on the position measurements is very low. There is relatively more variance in the angle measurements resulting in a stronger filtering effect in the estimated angle. Note that there is almost no phase shift between the measured and estimated states. This is critical for smooth control

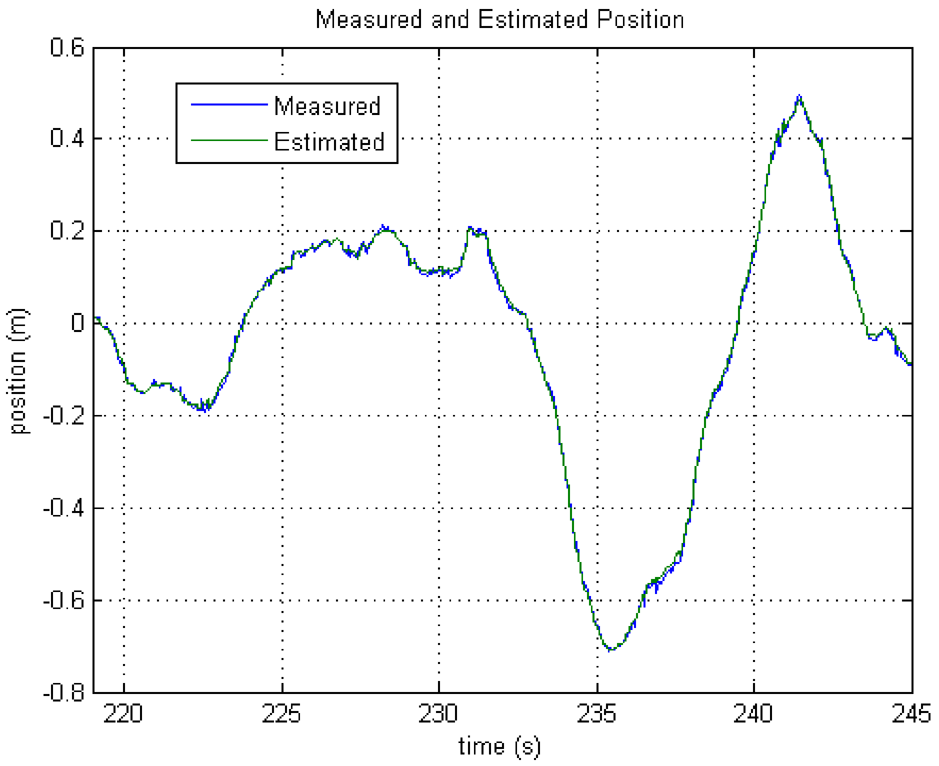

Fig. 3.1. Position of the vehicle during an aggressive driving maneuver.

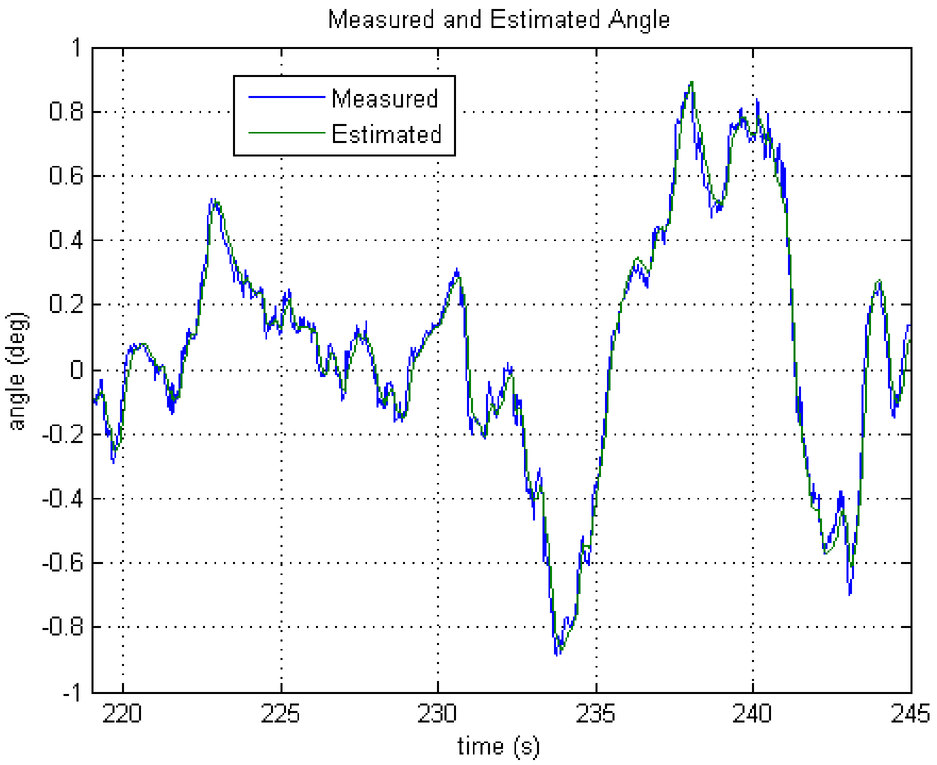

Fig. 3.2. Angle of the vehicle during an aggressive driving maneuver.

Table 3.1
Standard deviation of the error between the true and estimated vehicle position under various lane boundary conditions.

	Solid & Dashed	Solid Only	Dashed Only
Cloudy Day	2.57cm	2.68cm	5.21cm

as feedback delays often lead to oscillation and is one of the primary advantages of the state estimation technique used over traditional digital filtering.

In order to objectively evaluate the performance of the system, a second camera was attached to the vehicle on the passenger side pointing down at the road. The video output of this camera was recorded while the system data was recorded on a laptop. The position of the vehicle was then manually determined from the video at 150 points in time and the error between the true position and the system determined position was computed. The standard deviation of the error under three lane boundary conditions is given in table 3.1.

Finally, because the state estimation technique of this chapter and the control technique of the next chapter rely heavily on the state model of the system, it is important to verify the model by comparing the modeled response to the actual response. To evaluate the model, the control inputs to the model were measured over a 480 time step window along with the measured position and angle of the vehicle as determined by the image processing algorithm. After initializing the model to the first measured position and angle, the control inputs were fed into the model to get the feedforward response purely based on the model (no corrections from the true measurements). Figures 3.3 and 3.4 compare the response of the model and the measured position and angle of the vehicle. The vehicle acts as a double integrator so some large bias errors are expected, particularly in position. However, the modeled and measured responses match remarkably well indicating that the model is a very good representation of the actual system.

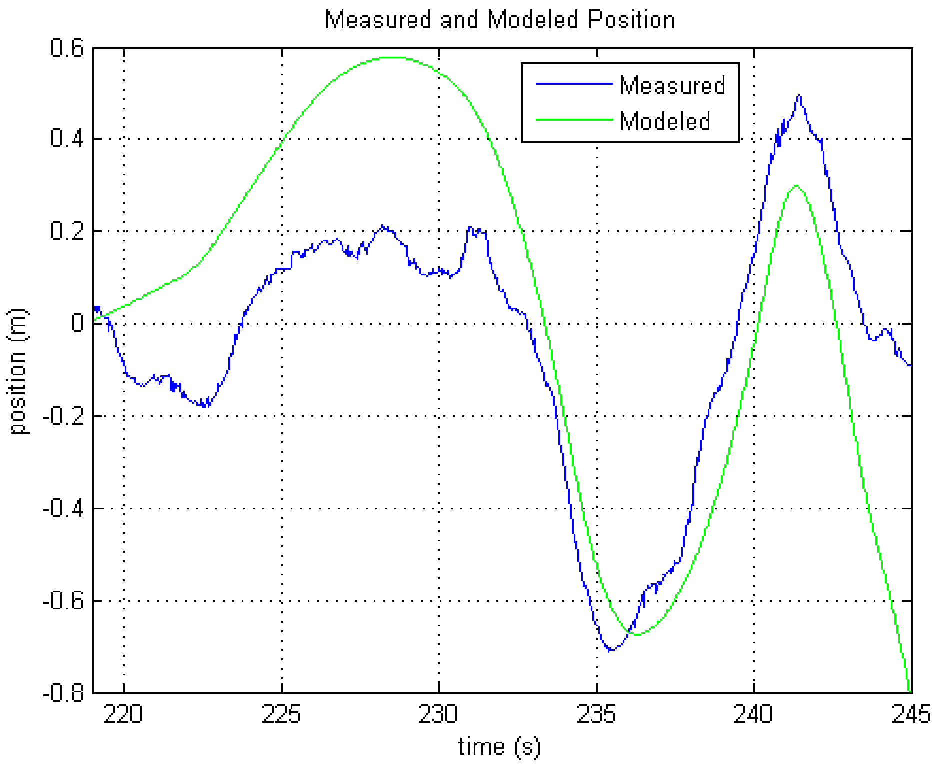

Fig. 3.3. Comparison of the modeled and actual position of the vehicle during an aggressive driving maneuver.

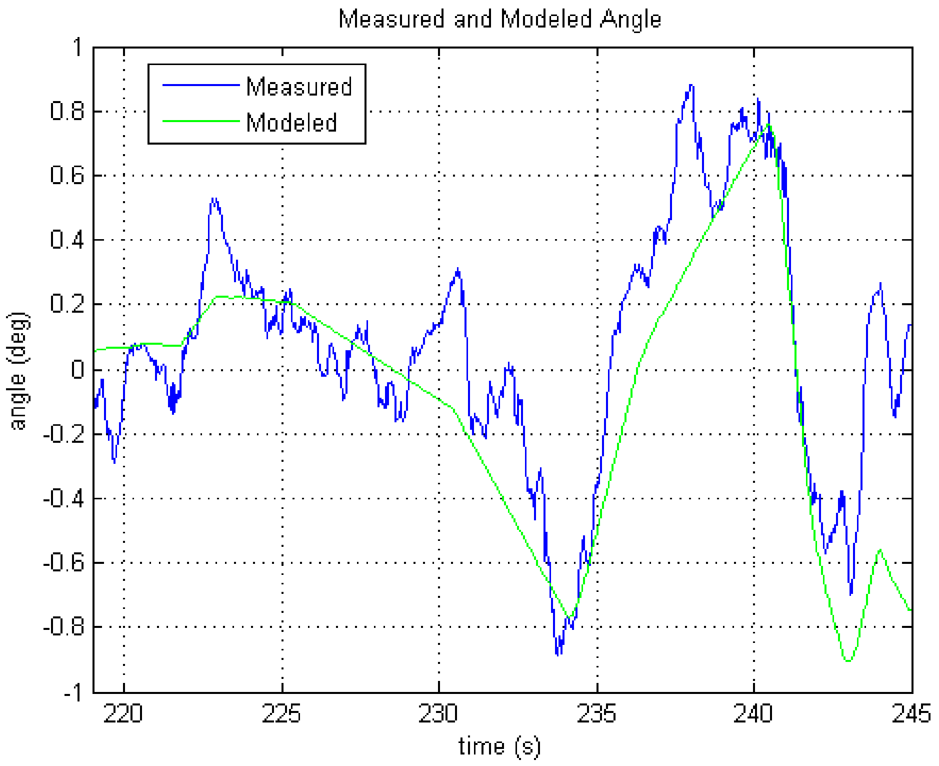

Fig. 3.4. Comparison of the modeled and actual angle of the vehicle during an aggressive driving maneuver.

4. CONTROL

With the state estimates of the preceding section, it is possible to stabilize the vehicle and track the **0** state of the system. Specifically the goal is to select a control to drive the vehicle to the center of the lane (0 position) and hold it there (which implies 0 angle). This is done using standard state feedback with the discrete linear quadratic regulator (DLQR) method used to compute the gain matrix K [6]. Sections 4.1-4.3 describe how this gain matrix is selected based on the discrete model of the vehicle developed in the preceding chapter. In section 4.4 a compensation for the estimated steering wheel bias is applied. Finally, in section 4.5 some experimental results are given.

The output of the Kalman-Bucy filter of the preceeding chapter is a reasonable estimate of the current state of the system. Because of this, state-feedback control can be applied to the estimates with a high degree of confidence. As mentioned above, the control strategy consists of two steps. First, an appropriate feedback control is selected using DLQR based only on p_k and θ_k. Second, the bias term in the model b_k represents a steering wheel bias and hence is directly compensated for by subtracting it from the computed control in the first step.

4.1 Discrete Model and Control

Recall that the discretized linear state model of the vehicle system is given by

$$\begin{bmatrix} x_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & v_k T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\hat{\alpha} v_k^2 T^2 \\ \hat{\alpha} v_k T \end{bmatrix} \phi_k = A_k \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} + B_k \phi_k \qquad (4.1)$$

where x_k and θ_k are the horizontal position and orientation respectively of the vehicle at time kT. ϕ_k is the steering wheel input at time kT. For convenience we define a state vector $z_k = [x_k\ \theta_k]^T$ which is the state of the system at time kT. The goal of discrete control is to choose a feedback matrix K_k s.t. by using a commanded control $\bar{\phi}_k = K_k z_k$ the system is stabilized and has the desired performance.

4.2 DLQR Feedback Selection

The discrete linear quadratic regulator (DLQR) control [6] seeks to choose K_k so that the system is stabilized and the cost function J_k is minimized for the system of 4.1 where

$$J_k = \sum_{i=k}^{\infty} z_i^T Q z_i + R\phi_i^2 \tag{4.2}$$

Q is a matrix representing the cost of the state vector deviating from 0 and is set experimentally. R is a scalar representing the cost of using the steering wheel and is also set experimentally.

Once Q and R are chosen, the K_k which stabilizes the system while minimizing 4.2 is given by

$$K_k = P_k B_k \left(R + B_k^T P_k B_k\right)^{-1} \tag{4.3}$$

where P_k is the solution to the discrete algebraic Ricatti equation (DARE)

$$P_k = \left(Q + A_k^T P_k A_k\right) - \left(B_k^T P_k A_k\right)^T \left(R + B_k^T P_k B_k\right)^{-1} \left(B_k^T P_k A_k\right) \tag{4.4}$$

4.3 Using a Time-Invariant K

Although K_k can be computed using the above method, this is computationally expensive. Equation 4.4 must be solved iteratively. In the case of solving the DARE in the preceding chapter 3.18, S_k is fairly constant over changes in A_k and hence a good initial guess can be provided resulting in very fast convergence. However, P_k in 4.4 changes significantly with changes in A_k and B_k resulting in poor convergence from a fixed initial guess. Experimentation shows that these changes in P_k are largely cancelled by corresponding changes in B_k in 4.3 resulting in a K_k which is nearly constant over wide variations in v_k. It has been experimentally determined that an offset on the position (in meters) should be weighted about the same as an offset of the angle (in degrees) in order to obtain good tracking of the center of the lane. For example, let Q and R be chosen as

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & \frac{\pi}{180} \end{bmatrix} \tag{4.5}$$

$$R = 6 \tag{4.6}$$

where R was experimentally chosen to yield a reasonably fast control response (a smaller value would give a faster response but at the expense of user comfort). Then the following results are obtained for K_k when $v_k = 10\text{MPH}$ and $v_k = 120\text{MPH}$

$$K_{k,10} = [0.406\ 14.1] \tag{4.7}$$

$$K_{k,120} = [0.388\ 13.8] \tag{4.8}$$

where the proportional constant $\hat{\alpha}$ is estimated to be 0.0041. 10MPH and 120MPH are reasonable bounds on the velocities for which the steering system could be used. Because K_k is nearly constant across this range it is reasonable to use a fixed K computed at an intermediate velocity. The nominal value of K is taken at $v = 60\text{MPH}$ which yields

$$K = [0.398\ 13.9]. \tag{4.9}$$

4.4 Bias Correction

Recall that in the preceding chapter an additional nearly constant bias state b was estimated. b acts on the vehicle system as a bias in the steering wheel position. Hence, by adding $-b$ to the commanded steering wheel position, the effect of the bias is cancelled. The final commanded steering wheel angle $\hat{\phi}_k$ is given by

$$\hat{\phi}_k = Kz_k - b_k \tag{4.10}$$

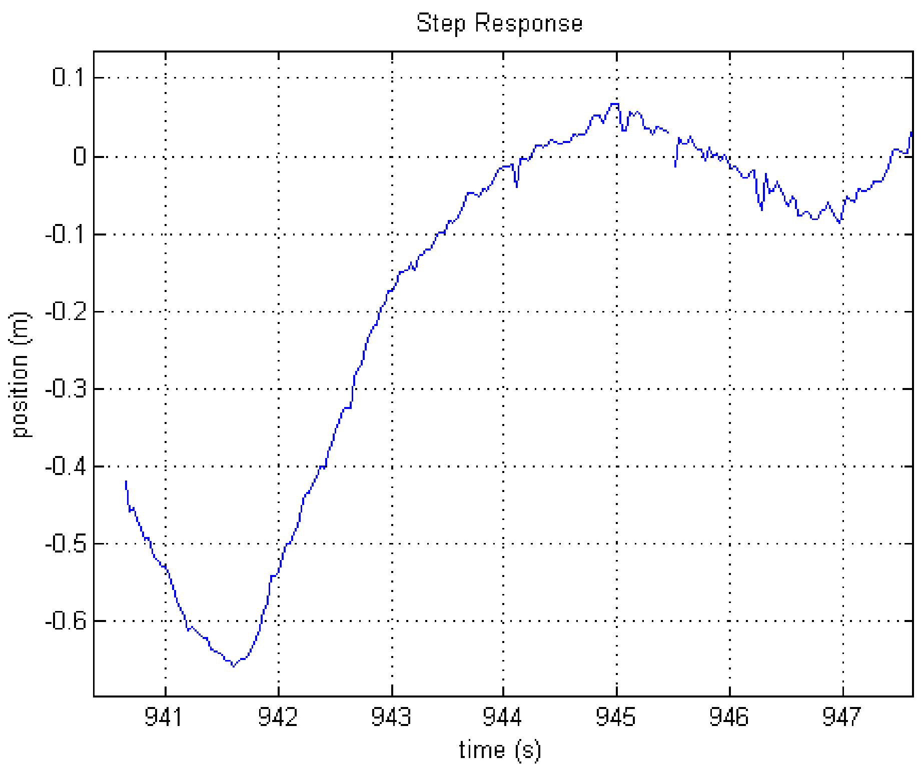

Fig. 4.1. Position of the vehicle after a large disturbance.

4.5 Experimental Results

We present two types of experimental results to show the effectiveness of the above described control scheme. First, the response of the system to a single large disturbance (a sort of step response) is shown in figure 4.1. This data was captured by covering the camera for a short period of time until the vehicle began to drift out of the lane and then suddenly uncovering the camera (at 940.6 seconds). The position of the vehicle at the beginning of the response is -0.42 meters and the angle is -1.3 degrees. The slight overshoot and minor oscillation at the end of the response is probably due to delays in the system.

Figure 4.2 is a plot of the position of the vehicle as the control attempts to track zero. This data was captured at night while it was raining which represents a worst case scenario for the image processing. Furthermore there were puddles on the road

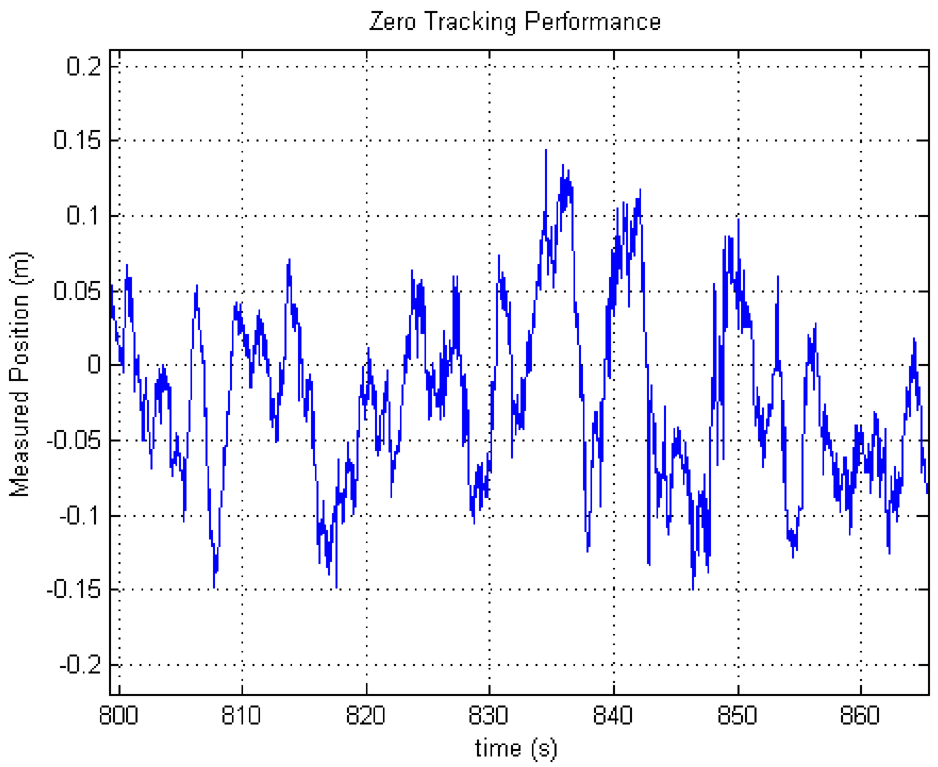

Fig. 4.2. Zero tracking during very poor conditions.

which were constantly pulling the vehicle to one side or the other due to increased drag. Hence figure 4.2 shows the ability of the system to track close to zero even under very poor conditions. Even under these conditions, the standard deviation of the position error is only 5.85cm.

5. FUTURE DIRECTION

Future research on autonomous vehicle steering will focus on the first and third steps in the process: image processing and the control algorithm. The image processing will be expanded to include an estimate of the curvature of the road. Also, some benefit in the smoothness of the measurements of the image processing algorithm may be achieved with greater attention to post-processing the data before curve fitting. One example is to modify the least squares fit to penalize large changes in the fit parameters. Another example is to search for the lane boundaries on a line connecting the buddy pairs, expanding the number of points in the least squares fit. Modifications will also be made to handle a greater range of road scenarios than the current set of assumptions allow, e.g. turn lanes, hills, changing lanes, etc. Part of this may be achieved by utilizing better integration of the GPS module. For example, the GPS module could provide information about where turn lanes are located, temporarily switching the image processing algorithm to a mode which specifically rejects turn lanes. Alternatively, a second camera may be used to improve some scenarios, though this would damatically increase the processing requirements. Further research will explore vehicle state estimation when no lane boundaries are present using alternative image features and lane estimation techniques. The motion of any static object between frames provides some information about the motion of the vehicle and hence by tracking several static objects it should be possible to obtain estimates of vehicle motion.

In the future, the control algorithm will progress beyond basic linear control strategies such as DLQR. There are two primary goals for the control, tracking the center of the lane and user comfort. The DLQR strategy is very effective at accomplishing the first goal but the second goal is not directly a part of the control. Future control algorithms will need to explicitly include both goals. To do this, human driving patterns and preferences need to be analyzed in order to define what user comfort entails. This will then be incorporated into the control, perhaps as a modified cost function or by using path planning/tracking. Model predictive control (MPC) will be used to implement the control strategy because of its flexibility in dealing with nonlinear models and changing environments.

LIST OF REFERENCES

LIST OF REFERENCES

[1] J. C. McCall and M. M. Trivedi, “Video-based lane estimation and tracking for driver assistance: Survey, system, and evaluation,” in *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, pp. 20–37, 2006.

[2] K. C. Kluge, “Performance evaluation of vision-based lane sensing: Some preliminary tools, metrics, and results,” in *IEEE Intelligent Transportation Systems Conference*, pp. 723–728, 1997.

[3] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” in *International Journal of Computer Vision*, 2004.

[4] B. Southall and C. J. Taylor, “Stochastic road shape estimation,” in *IEEE Proceedings of the International Conference on Computer Vision*, vol. 1, pp. 205–212, 2001.

[5] R. Pepy, A. Lambert, and H. Mounier, “Reducing navigation errors by planning with realistic vehicle model,” in *IEEE Intelligent Vehicles Symposium*, pp. 300–307, 2006.

[6] P. Dorato, C. T. Abdallah, and V. Cerone, *Linear Quadratic Control, an Introduction.* Malabar, Florida: Krieger Publishing Company, 2000.

[7] R. G. Brown, *Introduction to Random Signal Analysis and Kalman Filtering.* New York, New York: John Wiley & Sons, Inc., 1983.

APPENDIX

A. APPENDIX TO IMAGE PROCESSING

A.1 Slope weight determination

Simple linear regression is used to estimate the slope of each lane boundary

$$\hat{m} = \frac{n\sum x_i z_i - \sum x_i \sum z_i}{n\sum z_i^2 - (\sum z_i)^2} \tag{A.1}$$

where (x_i, z_i) for $i = 1, ..., n$ are the n coordinates in $\mathcal{X}_v$ identified to be on a particular lane boundary. The error in the slope estimate is given by $e_m = m - \hat{m}$ and the variance of e_m is given by

$$\sigma_m^2 = \frac{\sigma_x^2}{\sum(z_i - \bar{z})^2} \tag{A.2}$$

where $\bar{z}$ is the mean z coordinate and it is assumed that the error for each coordinate pair is independent of the coordinate pairs.

Recall that the estimated slope of the center of the lane m_C is a weighted average of the slopes of each lane boundary, i.e.

$$m_C = a_m m_L + (1 - a_m) m_R. \tag{A.3}$$

The error in this estimate is simply the weighted average of the error for each lane boundary, i.e.

$$e_{m_C} = a_m e_{m_L} + (1 - a_m) e_{m_R}. \tag{A.4}$$

The goal is to choose a_m so that the variance of e_{m_C} is minimized. The variance of e_{m_C} is given by

$$\sigma_{m_C}^2 = a_m^2 \sigma_{m_L}^2 + (1 - a_m)^2 \sigma_{m_R}^2. \tag{A.5}$$

where it has been assumed that the error of the two lane boundaries is independent. To minimize the variance we take the derivative of equation A.5 with respect to a_m, set it equal to zero, then solve for a_m to get

$$a_m = \frac{\sigma_{m_R}^2}{\sigma_{m_L}^2 + \sigma_{m_R}^2}. \tag{A.6}$$

A.2 Intercept weight determination

Simple linear regression is again used to estimate the x-intercept of each lane boundary

$$\hat{b} = \bar{x} - \hat{m}\bar{z} \tag{A.7}$$

where $\bar{x}$ and $\bar{z}$ are the mean x and z coordinates respectively for all of the coordinates identified to be on a particular lane boundary. The error in the x-intercept estimate is given by $e_b = b - \hat{b}$ and the variance of e_b is given by

$$\sigma_b^2 = \sigma_m^2 \bar{z}^2. \tag{A.8}$$

Recall that the estimated x-intercept for the center of the lane is obtained by first shifting the x-intercept for each lane boundary by half of the estimated lanewidth $\hat{w}$ towards the center of the lane and then taking a weighted average of the shifted intercepts, i.e.

$$\hat{b}_C = a_b \left(\hat{b}_L - \frac{\hat{w}}{2} \right) + (1 - a_b) \left(\hat{b}_R + \frac{\hat{w}}{2} \right) \tag{A.9}$$

The error in the estimate of $\hat{b}_C$ is given by

$$e_{b_C} = a_b e_{b_L} + (1 - a_b) e_{b_R} + \left(\frac{1}{2} - a_b \right) e_w \tag{A.10}$$

where e_w is the error in the lanewidth estimate. The variance of e_{b_C} is given by,

$$\sigma_{b_C}^2 = a_b^2 \sigma_{b_L}^2 + (1 - a_b)^2 \sigma_{b_R}^2 + \left(\frac{1}{2} - a_b \right)^2 \sigma_w^2 \tag{A.11}$$

where σ_w^2 is the variance of the error in the lanewidth estimate and the errors for each lane boundary and the lanewidth have been assumed to be independent of each other. The goal is to choose a_b in order to minimize $\sigma_{b_C}^2$. This can be done by taking the derivative of equation A.11 with respect to a_b, setting it equal to zero, and solving for a_b to get

$$a_b = \frac{1}{2} \left(\frac{2\sigma_{b_R}^2 + \sigma_w^2}{\sigma_{b_L}^2 + \sigma_{b_R}^2 + \sigma_w^2} \right). \tag{A.12}$$

B. DIAGRAMS

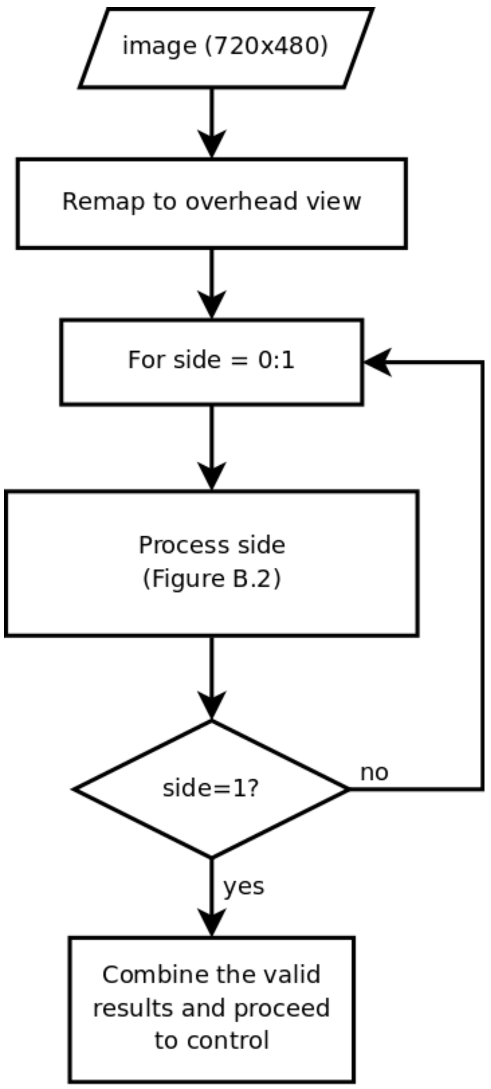
image (720x480)
Remap to overhead view
For side = 0:1
Process side (Figure B.2)
side=1?
no
yes
Combine the valid results and proceed to control

Fig. B.1. Diagram of the image processing algorithm

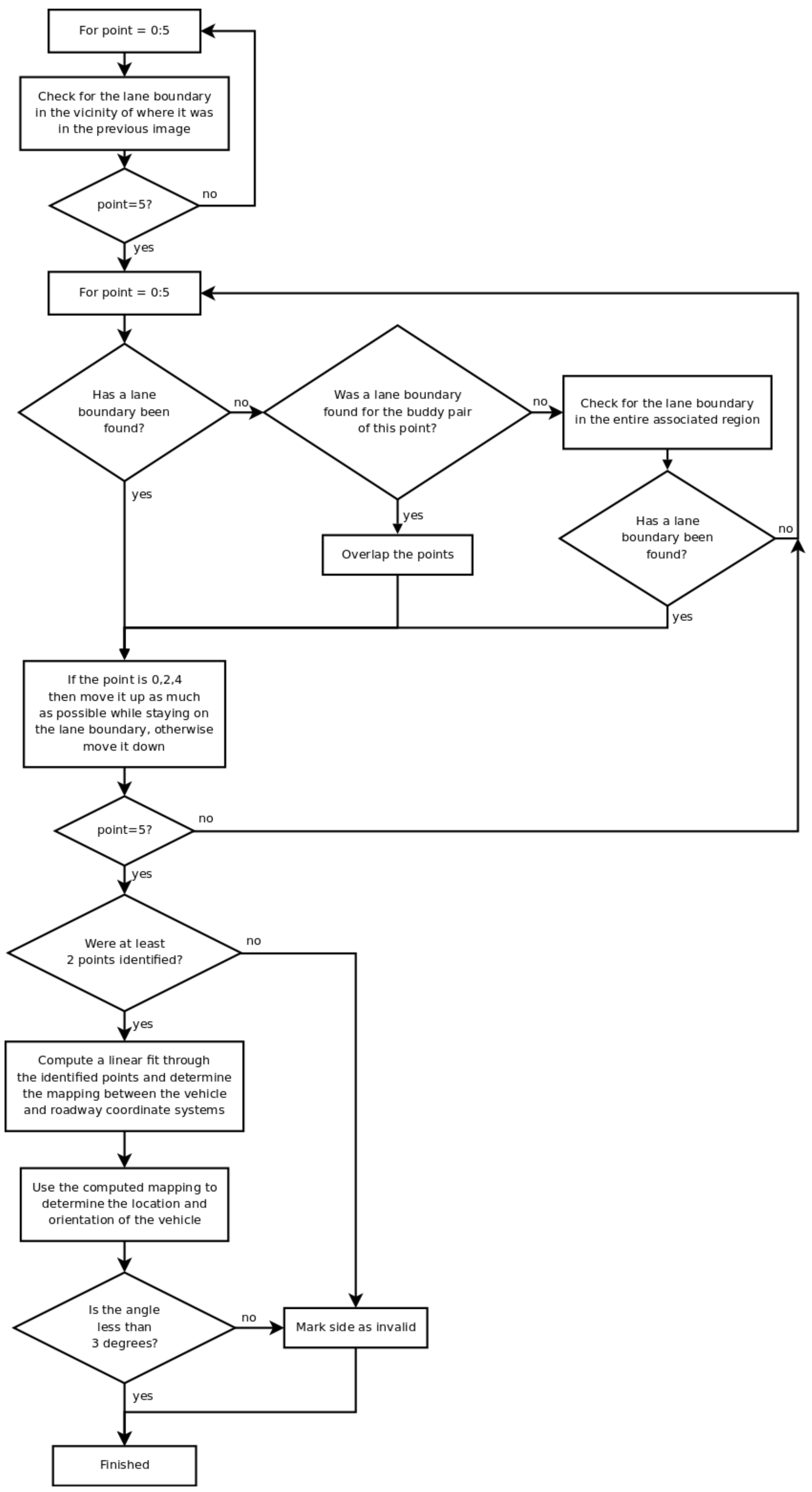
For point = 0:5
Check for the lane boundary in the vicinity of where it was in the previous image
point=5?
no
yes
For point = 0:5
Has a lane boundary been found?
no
Was a lane boundary found for the buddy pair of this point?
no
Check for the lane boundary in the entire associated region
Has a lane boundary been found?
no
yes
Overlap the points
yes
yes
If the point is 0,2,4 then move it up as much as possible while staying on the lane boundary, otherwise move it down
point=5?
no
yes
Were at least 2 points identified?
no
yes
Compute a linear fit through the identified points and determine the mapping between the vehicle and roadway coordinate systems
Use the computed mapping to determine the location and orientation of the vehicle
Is the angle less than 3 degrees?
no
Mark side as invalid
yes
Finished

Fig. B.2. Diagram of the update process for one side of the image

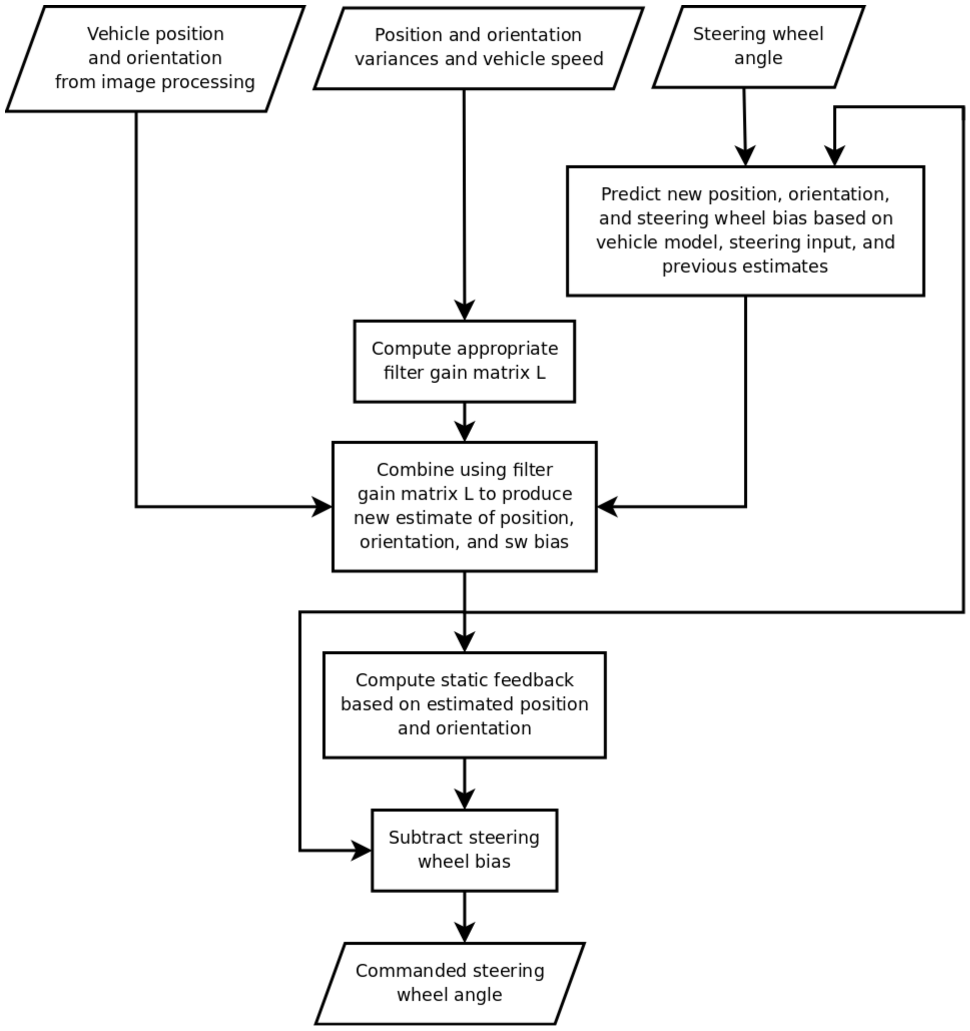
Vehicle position and orientation from image processing
Position and orientation variances and vehicle speed
Steering wheel angle
Predict new position, orientation, and steering wheel bias based on vehicle model, steering input, and previous estimates
Compute appropriate filter gain matrix L
Combine using filter gain matrix L to produce new estimate of position, orientation, and sw bias
Compute static feedback based on estimated position and orientation
Subtract steering wheel bias
Commanded steering wheel angle

Fig. B.3. Diagram of the control algorithm

C. CODE

C.1 image_ processing.h

```
#ifndef _IMAGE_PROCESSING_H_
#define _IMAGE_PROCESSING_H_

// image parameters
#define Line_Length 720

// line finding params
#define MIN_DETECTIONS 4
#define MIN_DIFF_Z 0.5f
#define MIN_DIFF_Z_ANGLE 5.0f
#define MIN_DIFF_Z_POS 1.0f
#define MAX_ANGLE 0.052360f
#define MAX_DIFF_ANGLE 0.0052360f
#define MAX_DIFF_ANGLE_SQ 0.0000274156f
#define MAX_ANGLE_CHECK 0.0610865f
#define MIN_LANEWIDTH 2.2f
#define SIGMA2_X 0.0025f
#define SIGMA2_W 0.0025f
#define NUMBER_OF_POINTS 6
#define FORCE_SPACING 3
#define VERT_SEARCHWIDTH 2
#define MIN_SEARCHWIDTH 6
#define DEFAULT_SEARCHWIDTH 40
#define MAX_SEARCHWIDTH 40
#define SLIDE_INC 5
#define BRIGHTINC 0.01f
#define BRIGHTMAX 35.0f
#define BRIGHTMIN 7.0f
```

```
#define KERNEL_BUFF_SIDE 3
#define KERNEL_BUFF_WIDTH 8
#define KERNEL_BUFF_TOP 2
#define KERNEL_BUFF_HEIGHT 5
#define KERNEL_SCALE 10

// observer parameters
#define EDT 0.03337f
#define CONTROL_SCALE 0.0041f

#endif
```

C.2 image_ processing.c

```
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <services\services.h>
#include "ezkitutilities.h"

// image_reconstruction.h must be first
#include "image_reconstruction.h"
#include "image_processing.h"
#include "matrix_support.h"

// globals
u16 volatile *imageData;
static float brightval = 20.0;
static float lanewidth = 3.0;
s16 kernel_thresh;
static float xspacing;
static float zspacing;
static float x_lqe[3] = {0,0,0};
static float mean_z[2];

// tracked points structure
```

```
static unsigned char tracked_points_i[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_j[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_if[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_i_orig[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_j_orig[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_searchwidth[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_j_min[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_j_max[2][NUMBER_OF_POINTS];
static char tracked_points_dir[2][NUMBER_OF_POINTS];
static bool tracked_points_extra_check[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_num_detected[2][NUMBER_OF_POINTS];
static unsigned char tracked_points_buddy[2][NUMBER_OF_POINTS];
static bool tracked_points_detected[2][NUMBER_OF_POINTS];
static bool tracked_points_true_detected[2][NUMBER_OF_POINTS];
static float tracked_points_x[2][NUMBER_OF_POINTS];
static float tracked_points_z[2][NUMBER_OF_POINTS];
static float ls_pos[2];
static bool ls_valid[2];
static float ls_angle[2];
static bool ls_angle_valid[2];
static float ls_b[2] = {0,0};
static float ls_c[2] = {-1.5f,1.5f};
static const float A_lqe_ns[9] = {1,EDT,0.5f*EDT*EDT*CONTROL_SCALE        ,
        0,1,EDT*CONTROL_SCALE        ,        0,0,1};
static const float B_lqe_ns[3] = {0.5f*EDT*EDT*CONTROL_SCALE,EDT*
    CONTROL_SCALE,0};

static u8 overhead_view_u8[GRID_HEIGHT][GRID_WIDTH];
static s16 overhead_view_computed[GRID_WIDTH][GRID_HEIGHT];

static float angle=0;
static float xout=0;
static bool valid=false;
static float currentdist=0;
static float currenttheta=0;
static float currentpos=0;
static float lwfilt=0.1f;
static u16 deadtime=0;
static float control_delay = 0;
static float control_output = 0;

static float worst_sigma = 0;
static float worst_sigma_theta = 0;

extern float speed_ms;

float sigma_e[2] = {0,0};
float sigma_e_theta[2] = {0,0};

// needed headers
s16 determine_control(float actualpos, float currenttheta, float
    currentdist, float currentsw, char control_mode, bool valid, float
    *);

void image_processing_init()
{
  unsigned char side,point;
  xspacing = xvals[1]-xvals[0];
  zspacing = zvals[1]-zvals[0];
  for (side=0;side<2;side++)
  {
    for (point=0;point < NUMBER_OF_POINTS; point++)
    {
      tracked_points_i_orig[side][point] = side==0?((GRID_WIDTH*3))/4:(
          GRID_WIDTH/4);
      tracked_points_i[side][point] = tracked_points_i_orig[side][point
          ];
      tracked_points_if[side][point] = 0;
      tracked_points_x[side][point] = xvals[tracked_points_i[side][point
          ]]+0.05f*xspacing;
```

```
        tracked_points_detected[side][point] = 0;
        tracked_points_extra_check[side][point] = 0;
        tracked_points_num_detected[side][point] = 0;
        tracked_points_searchwidth[side][point] = DEFAULT_SEARCHWIDTH;
        tracked_points_j_min[side][point] = 0 + FORCE_SPACING + (point/2*(
            GRID_HEIGHT))*2.0f/(float)NUMBER_OF_POINTS;
        tracked_points_j_max[side][point] = 0 - FORCE_SPACING + ((point
            /2+1)*(GRID_HEIGHT))*2.0f/(float)NUMBER_OF_POINTS;
        if (0 == point%2)
        {
          tracked_points_j[side][point] = tracked_points_j_max[side][point
              ];
          tracked_points_j_orig[side][point] = tracked_points_j[side][
              point];
          tracked_points_z[side][point] = zvals[tracked_points_j[side][
              point]];
          tracked_points_buddy[side][point] = point + 1;
          tracked_points_dir[side][point] = -1;
        }
        else
        {
          tracked_points_j[side][point] = tracked_points_j_min[side][point
              ];
          tracked_points_j_orig[side][point] = tracked_points_j[side][
              point];
          tracked_points_z[side][point] = zvals[tracked_points_j[side][
              point]];
          tracked_points_buddy[side][point] = point - 1;
          tracked_points_dir[side][point] = 1;
        }
      }
    }
  }
```

```
void compute_overhead_view_partial(u16 i_start, u16 i_stop, u16 j_start,
     u16 j_stop)
{
  unsigned char i,j;
  for (i=i_start;i<i_stop;i++)
  {
    for (j=j_start;j<j_stop;j++)
    {
      overhead_view_u8[i][j] = imageData[Line_Length*grid_y[j][i]+grid_x
          [j][i]]/256;
    }
  }
}

void check_for_line_in_region(short col_start, short col_stop, short
    row_start, short row_stop, bool store_match)
{
  u8 i,j,j2,iL,iR,iC;
  s16 sumL,sumR,sumC,zeroC=0,zeroR=0,oneC=0,oneR=0,twoC=0,twoR=0,row,col
     ,old_row,new_row;

  // first slide the box down, then move one slot to the right and slide
       again, ...
  for (col=col_start;col<col_stop;col++)
  {
    sumL = 0;
    sumR = 0;
    sumC = 0;
    iL = col-KERNEL_BUFF_SIDE;
    iC = col;
    iR = col+KERNEL_BUFF_SIDE;
    row = row_start;
    j2 = row-KERNEL_BUFF_TOP;

    if (col-col_start<3)
```

```
{
  for (j=0;j<KERNEL_BUFF_HEIGHT;j++,j2++)
  {
    sumL += overhead_view_u8[j2][iL];
    sumL += overhead_view_u8[j2][iL+1];
    sumC += overhead_view_u8[j2][iC];
    sumC += overhead_view_u8[j2][iC+1];
    sumR += overhead_view_u8[j2][iR];
    sumR += overhead_view_u8[j2][iR+1];
  }
}
else if (0==col%3)
{
  sumL = zeroC;
  sumC = zeroR;
  for (j=0;j<KERNEL_BUFF_HEIGHT;j++,j2++)
  {
    sumR += overhead_view_u8[j2][iR];
    sumR += overhead_view_u8[j2][iR+1];
  }
}
else if (1==col%3)
{
  sumL = oneC;
  sumC = oneR;
  for (j=0;j<KERNEL_BUFF_HEIGHT;j++,j2++)
  {
    sumR += overhead_view_u8[j2][iR];
    sumR += overhead_view_u8[j2][iR+1];
  }
}
else
{
  sumL = twoC;
  sumC = twoR;
  for (j=0;j<KERNEL_BUFF_HEIGHT;j++,j2++)
  {
    sumR += overhead_view_u8[j2][iR];
    sumR += overhead_view_u8[j2][iR+1];
  }
}

if (0==col%3)
{
  zeroC = sumC;
  zeroR = sumR;
}
else if (1==col%3)
{
  oneC = sumC;
  oneR = sumR;
}
else
{
  twoC = sumC;
  twoR = sumR;
}

if (((sumC-sumL) > kernel_thresh) && ((sumC-sumR) > kernel_thresh))
{
  overhead_view_computed[col][row] = 1;
}
else
{
  overhead_view_computed[col][row] = -1;
}

old_row = (row_start)-KERNEL_BUFF_TOP;
new_row = (row_start)-KERNEL_BUFF_TOP+KERNEL_BUFF_HEIGHT;
for (row=row_start+1;row<row_stop;row++,old_row++,new_row++)
```

```
      {
        sumL -= overhead_view_u8[old_row][iL];
        sumL -= overhead_view_u8[old_row][iL+1];
        sumC -= overhead_view_u8[old_row][iC];
        sumC -= overhead_view_u8[old_row][iC+1];
        sumR -= overhead_view_u8[old_row][iR];
        sumR -= overhead_view_u8[old_row][iR+1];
        sumL += overhead_view_u8[new_row][iL];
        sumL += overhead_view_u8[new_row][iL+1];
        sumC += overhead_view_u8[new_row][iC];
        sumC += overhead_view_u8[new_row][iC+1];
        sumR += overhead_view_u8[new_row][iR];
        sumR += overhead_view_u8[new_row][iR+1];
        if (store_match)
        {
            s16 matchL = sumC-sumL;
            s16 matchR = sumC-sumR;
            overhead_view_computed[col][row] = min(matchL,matchR);
        }
        else
        {
            if (((sumC-sumL) > kernel_thresh) && ((sumC-sumR) >
                kernel_thresh))
            {
              overhead_view_computed[col][row] = 1;
            }
            else
            {
              overhead_view_computed[col][row] = -1;
            }
        }
      }
    }
}
```

```
void invalidate_side(u8 side)
{
  u8 point;
  for (point=0;point < NUMBER_OF_POINTS; point++)
  {
    tracked_points_detected[side][point] = false;
  }
}

void shift_lane(float distance)
{
  u8 side,point;
  u8 offset = (u8)(GRID_WIDTH*distance/X_WIDTH);
  for (side=0;side<2;side++)
  {
    for (point=0;point<NUMBER_OF_POINTS; point++)
    {
      tracked_points_i[side][point] += offset;
    }
  }
}

void line_find_new(void)
{
  unsigned char i_center;
  unsigned char j_center;
  unsigned char i_width;
  short i_start;
  short i_stop;
  short j_start;
  short j_stop;
  int i_sum;
  int j_sum;
  unsigned char detected_count;
  unsigned short total_detected = 0;
```

```
char ls_count[2];
float ls_spread[2];
unsigned char side,point,i,j;

kernel_thresh = (s16)(brightval*KERNEL_SCALE);

// first loop through both sides
for (side = 0; side<2; side++)
{
  // first try to confirm points with only minor movements
  for (point=0;point<NUMBER_OF_POINTS;point++)
  {
    tracked_points_detected[side][point] = false;
    tracked_points_extra_check[side][point] = false;
    tracked_points_if[side][point] = 0;
    i_center = tracked_points_i[side][point];
    j_center = tracked_points_j[side][point];
    i_width = tracked_points_searchwidth[side][point]/2;
    i_start = i_center-i_width-KERNEL_BUFF_SIDE;
    i_start = i_start<0?0:i_start;
    i_start += KERNEL_BUFF_SIDE;
    i_stop = i_center+i_width-KERNEL_BUFF_SIDE+KERNEL_BUFF_WIDTH;
    i_stop = i_stop>=GRID_WIDTH?GRID_WIDTH-1:i_stop;
    i_stop += KERNEL_BUFF_SIDE - KERNEL_BUFF_WIDTH + 1;
    j_start = j_center-VERT_SEARCHWIDTH-KERNEL_BUFF_TOP;
    j_start = j_start<0?0:j_start;
    j_start += KERNEL_BUFF_TOP;
    j_stop = j_center+VERT_SEARCHWIDTH-KERNEL_BUFF_TOP+
        KERNEL_BUFF_HEIGHT;
    j_stop = j_stop>=GRID_HEIGHT?GRID_HEIGHT-1:j_stop;
    j_stop += KERNEL_BUFF_TOP - KERNEL_BUFF_HEIGHT + 1;
    i_sum = 0;
    j_sum = 0;
    detected_count = 0;
  check_for_line_in_region(i_start,i_stop,j_start,j_stop,false);
  for ( i=i_start;i<=i_stop;i++)
  {
    for ( j=j_start;j<=j_stop;j++)
    {
      if (overhead_view_computed[i][j]>0)
      {
        i_sum += i;
        j_sum += j;
        detected_count++;
      }
    }
  }
  if (detected_count >= MIN_DETECTIONS)
  {
    tracked_points_detected[side][point] = true;
    tracked_points_i_orig[side][point] = i_sum/detected_count;
    tracked_points_j_orig[side][point] = j_sum/detected_count;
    tracked_points_searchwidth[side][point] = MIN_SEARCHWIDTH;
  }
}

// second use major movements to detect points
for ( point=0;point<NUMBER_OF_POINTS;point++)
{
  if (!tracked_points_detected[side][point])
  {
    unsigned char buddy = tracked_points_buddy[side][point];
    if (tracked_points_detected[side][buddy])
    {
      tracked_points_detected[side][point] = true;
      tracked_points_i_orig[side][point] = tracked_points_i_orig[
          side][buddy];
      tracked_points_j_orig[side][point] = tracked_points_j_orig[
          side][buddy];
  tracked_points_searchwidth[side][point] = MIN_SEARCHWIDTH;
}
else
{
  if (!tracked_points_extra_check[side][buddy])
  {
    tracked_points_extra_check[side][point] = true;
    if (tracked_points_searchwidth[side][point]<
        DEFAULT_SEARCHWIDTH)
    {
      tracked_points_searchwidth[side][point] =
          DEFAULT_SEARCHWIDTH;
    }
    else if (tracked_points_searchwidth[side][point]>
        MAX_SEARCHWIDTH-2)
    {
      tracked_points_searchwidth[side][point] = MAX_SEARCHWIDTH;
    }
    else
    {
      tracked_points_searchwidth[side][point] += 2;
    }
    char dir = tracked_points_dir[side][point];
    unsigned char j_min = tracked_points_j_min[side][point];
    unsigned char j_max = tracked_points_j_max[side][point];
    i_center = tracked_points_i[side][point];
    i_width = tracked_points_searchwidth[side][point]/2;
    i_start = i_center-i_width-KERNEL_BUFF_SIDE;
    i_start = i_start<0?0:i_start;
    i_start += KERNEL_BUFF_SIDE;
    i_stop = i_center+i_width-KERNEL_BUFF_SIDE+KERNEL_BUFF_WIDTH
        ;
    i_stop = i_stop>=GRID_WIDTH?GRID_WIDTH-1:i_stop;
    i_stop += KERNEL_BUFF_SIDE - KERNEL_BUFF_WIDTH + 1;
    j_start = j_min-VERT_SEARCHWIDTH-KERNEL_BUFF_TOP;
```

```
            j_start = j_start <0?0:j_start;
            j_start += KERNEL_BUFF_TOP;
            j_stop = j_max+VERT_SEARCHWIDTH-KERNEL_BUFF_TOP+
                KERNEL_BUFF_HEIGHT;
            j_stop = j_stop>=GRID_HEIGHT?GRID_HEIGHT-1:j_stop;
            j_stop += KERNEL_BUFF_TOP - KERNEL_BUFF_HEIGHT + 1;
            i_sum = 0;
            j_sum = 0;
            detected_count = 0;
            check_for_line_in_region(i_start,i_stop,j_start,j_stop,false
                );
            for ( i=i_start;i<=i_stop;i++)
            {
              for ( j=j_start;j<=j_stop;j++)
              {
                if (overhead_view_computed[i][j]>0)
                {
                  i_sum += i;
                  j_sum += j;
                  detected_count++;
                }
              }
            }
            if (detected_count >= MIN_DETECTIONS)
            {
              tracked_points_detected[side][point] = true;
              tracked_points_i_orig[side][point] = i_sum/detected_count;
              tracked_points_j_orig[side][point] = j_sum/detected_count;
              tracked_points_searchwidth[side][point] = MIN_SEARCHWIDTH;
            }
          }
        }
      }
    }
```

```
// slide
for ( point=0;point<NUMBER_OF_POINTS; point++)
{
  if (tracked_points_detected[side][point])
  {
    s8 dir = tracked_points_dir[side][point];
    u8 j_max = tracked_points_j_max[side][point];
    u8 j_min = tracked_points_j_min[side][point];
    u8 i_next = tracked_points_i_orig[side][point];
    u8 j_next = tracked_points_j_orig[side][point];
    u8 i_frac = 0;
    u8 j_frac = 0;
    i_width = tracked_points_searchwidth[side][point]/2;
    if (j_next < j_min)
    {
        j_next = j_min+FORCE_SPACING;
    }
    if (j_next > j_max)
    {
        j_next = j_max-FORCE_SPACING;
    }
    u8 i_prev = i_next;
    u8 j_prev = j_next;
    while ((j_next + dir*SLIDE_INC <= j_max) && (j_next + dir*
       SLIDE_INC >= j_min))
    {
      i_center = i_next;
      j_center = j_next+dir*SLIDE_INC;
      i_start = i_center-i_width-KERNEL_BUFF_SIDE;
      i_start = i_start<0?0:i_start;
      i_start += KERNEL_BUFF_SIDE;
      i_stop = i_center+i_width-KERNEL_BUFF_SIDE+KERNEL_BUFF_WIDTH;
      i_stop = i_stop>=GRID_WIDTH?GRID_WIDTH-1:i_stop;
      i_stop += KERNEL_BUFF_SIDE - KERNEL_BUFF_WIDTH + 1;
      j_start = j_center-VERT_SEARCHWIDTH-KERNEL_BUFF_TOP;
j_start = j_start<0?0:j_start;
j_start += KERNEL_BUFF_TOP;
j_stop = j_center+VERT_SEARCHWIDTH-KERNEL_BUFF_TOP+
    KERNEL_BUFF_HEIGHT;
j_stop = j_stop>=GRID_HEIGHT?GRID_HEIGHT-1:j_stop;
j_stop += KERNEL_BUFF_TOP - KERNEL_BUFF_HEIGHT + 1;
i_sum = 0;
j_sum = 0;
u8 temp_detected_count = 0;
check_for_line_in_region(i_start,i_stop,j_start,j_stop,false);
for ( i=i_start;i<=i_stop;i++)
{
  for ( j=j_start;j<=j_stop;j++)
  {
    if (overhead_view_computed[i][j]>0)
    {
      i_sum += i;
      j_sum += j;
      temp_detected_count++;
    }
  }
}
if (temp_detected_count >= MIN_DETECTIONS)
{
  detected_count = temp_detected_count;
  i_next = i_sum/detected_count;
  j_next = j_sum/detected_count;
  if ((j_next+dir*SLIDE_INC > j_max)||(j_next+dir*SLIDE_INC <
      j_min))
  {
      break;
  }
  i_prev = i_next;
  j_prev = j_next;
}
```

```
        else
        {
          break;
        }
      }
      tracked_points_num_detected[side][point] = detected_count;
      total_detected += detected_count;
      tracked_points_i[side][point] = i_prev;
      tracked_points_j[side][point] = j_prev;
      tracked_points_z[side][point] = zvals[j_prev];
    }
  }

  // finally refine the position estimate
  for (point=0;point<NUMBER_OF_POINTS;point++)
  {
    tracked_points_true_detected[side][point] =
        tracked_points_detected[side][point];
    if (tracked_points_detected[side][point])
    {
          float i_mean;
          s32 match_sum;
          u8 mean_count;
          u16 i_temp;
        i_center = tracked_points_i[side][point];
        j_center = tracked_points_j[side][point];
        i_width = MIN_SEARCHWIDTH/2;
        i_start = i_center-i_width-KERNEL_BUFF_SIDE;
        i_start = i_start<0?0:i_start;
        i_start += KERNEL_BUFF_SIDE;
        i_stop = i_center+i_width-KERNEL_BUFF_SIDE+KERNEL_BUFF_WIDTH;
        i_stop = i_stop>=GRID_WIDTH?GRID_WIDTH-1:i_stop;
        i_stop += KERNEL_BUFF_SIDE - KERNEL_BUFF_WIDTH + 1;
        j_start = j_center-VERT_SEARCHWIDTH-KERNEL_BUFF_TOP;
        j_start = j_start<0?0:j_start;
```

```
j_start += KERNEL_BUFF_TOP;
j_stop = j_center+VERT_SEARCHWIDTH-KERNEL_BUFF_TOP+
   KERNEL_BUFF_HEIGHT;
j_stop = j_stop>=GRID_HEIGHT?GRID_HEIGHT-1:j_stop;
j_stop += KERNEL_BUFF_TOP - KERNEL_BUFF_HEIGHT + 1;

check_for_line_in_region(i_start,i_stop,j_start,j_stop,true);

i_mean = 0;
mean_count = 0;
for ( j=j_start;j<=j_stop;j++)
{
    i_sum = 0;
    match_sum = 0;
    for ( i=i_start;i<=i_stop;i++)
    {
    if (overhead_view_computed[i][j]>0)
    {
        match_sum += overhead_view_computed[i][j];
      i_sum += i*overhead_view_computed[i][j];
    }
  }
  if (match_sum > 0)
  {
      i_mean += (float)i_sum/(float)match_sum;
      mean_count++;
  }
}
i_mean /= (float)mean_count;
  i_temp = (u16)(i_mean*16);
  tracked_points_i[side][point] = i_temp/16;
  tracked_points_if[side][point] = i_temp & 0x000F;
  tracked_points_x[side][point] = xvals[tracked_points_i[side
      ][point]] + 0.5*xspacing + 0.0625*xspacing*
      tracked_points_if[side][point];
    }
}

// peform validation
for (point=0;point<NUMBER_OF_POINTS; point++)
{
  if (tracked_points_detected[side][point])
  {
    u8 buddy = tracked_points_buddy[side][point];
    if (tracked_points_detected[side][buddy])
    {
      float dy = fabsf(tracked_points_x[side][point] -
          tracked_points_x[side][buddy]);
      float dx = fabsf(tracked_points_z[side][point] -
          tracked_points_z[side][buddy]);
      if ((dy>=dx*MAX_ANGLE_CHECK)||(dx<MIN_DIFF_Z))
      {
        tracked_points_detected[side][point] = false;
        tracked_points_detected[side][buddy] = false;
        continue;
      }
    }
  }
}

// compute a least squares position and angle
float sy=0, sxy=0, sx2y=0;
float sx=0, sx2=0, sx3=0, sx4=0;
u8 n = 0;
float z_max = Z_START;
float z_min = Z_STOP;
ls_angle[side] = 0;
for (point=0;point<NUMBER_OF_POINTS; point++)
{
  if (tracked_points_detected[side][point])
  {
    n++;
    float x2 = tracked_points_z[side][point]*tracked_points_z[side][
        point];
    sy += tracked_points_x[side][point];
    sxy += tracked_points_x[side][point]*tracked_points_z[side][
        point];
    sx += tracked_points_z[side][point];
    sx2 += x2;
    z_max = fmaxf(z_max,tracked_points_z[side][point]);
    z_min = fminf(z_min,tracked_points_z[side][point]);
  }
}
ls_count[side] = n;
ls_spread[side] = z_max-z_min;
mean_z[side] = sx/(float)n;
if ((n > 1)&&ls_spread[side]>MIN_DIFF_Z_POS)
{
  ls_angle[side] = (float)(((float)n*sxy - sx*sy)/((float)n*sx2 - sx
      *sx));
  if (fabsf(ls_angle[side]) > MAX_ANGLE)
  {
    ls_valid[side] = false;
    ls_angle_valid[side] = false;
    invalidate_side(side);
    ls_angle[side] = 0;
    ls_pos[side] = 0;
  }
  else
  {
    ls_valid[side] = true;
    ls_pos[side] = (float)((sy-ls_angle[side]*sx)/(float)n);
    ls_angle_valid[side] = true;
  }
}
```

```
  else
  {
    ls_valid[side] = false;
    ls_angle_valid[side] = false;
  }
  float zbar = mean_z[side];
  sigma_e_theta[side] = SIGMA2_X/(sx2-sx*zbar);
  sigma_e[side] = sigma_e_theta[side]*zbar*zbar;
  if (ls_spread[side] < MIN_DIFF_Z_ANGLE)
  {
    ls_angle_valid[side] = false;
  }


}

// determine vehicle position and angle and possibly lanewidth from ls
     data
u8 x_count = 0;
float x_sum = 0;
float angle_sum = 0;
bool use_optimal_pos = ls_valid[0]&&ls_valid[1];
if (use_optimal_pos)
{
  if ((ls_pos[1] - ls_pos[0])<MIN_LANEWIDTH)
  {
    use_optimal_pos = false;
    ls_valid[0] = false;
    ls_angle_valid[0] = false;
    invalidate_side(0);
    ls_valid[1] = false;
    ls_angle_valid[1] = false;
    invalidate_side(1);
  }
  else
```

```
    {
      float worst_sigma_theta = (sigma_e_theta[1] + sigma_e_theta[0]);
      float max_angle_diff = fmaxf(MAX_DIFF_ANGLE_SQ, worst_sigma_theta);
      float angle_diff = fabsf(ls_angle[0] - ls_angle[1]);
      if (angle_diff*angle_diff>max_angle_diff)
      {
        use_optimal_pos = false;
        if (fabsf(ls_angle[0] + angle) > fabsf(ls_angle[1] + angle))
        {
          ls_valid[0] = false;
          ls_angle_valid[0] = false;
          invalidate_side(0);
        }
        else
        {
          ls_valid[1] = false;
          ls_angle_valid[1] = false;
          invalidate_side(1);
        }
      }
    }
  }

  float a = 1;
  float b = 1;
  if (use_optimal_pos)
  {
    a = (sigma_e[0] + .5f*SIGMA2_W)/(sigma_e[0] + sigma_e[1] + SIGMA2_W)
        ;
    b = (sigma_e_theta[0])/(sigma_e_theta[0] + sigma_e_theta[1]);
  }
  for (side = 0; side<2; side++)
  {
    if (ls_valid[side])
    {
      x_count++;
      if (use_optimal_pos)
      {
        if (side==0)
        {
          x_sum += (1-a)*(ls_pos[side] + 0.5f*lanewidth);
          angle_sum += (1-b)*ls_angle[side];
        }
        else
        {
          x_sum += (a)*(ls_pos[side] - 0.5f*lanewidth);
          angle_sum += (b)*ls_angle[side];
        }
      }
      else
      {
        x_sum += ls_pos[side] - (side - 0.5f)*lanewidth;
        angle_sum += ls_angle[side];
      }
    }
  }
  bool pos_valid = (x_count > 0);
  bool angle_valid = pos_valid;
  if (pos_valid)
  {
    xout = -x_sum;
  }
  else
  {
    xout = 0;
  }
  if (xout > 0.5f*lanewidth)
  {
    shift_lane(lanewidth);
    xout -= lanewidth;
    x_lqe[0] = xout;
  }
  else if (xout < -0.5f*lanewidth)
  {
    shift_lane(-lanewidth);
    xout += lanewidth;
    x_lqe[0] = xout;
  }
  if (angle_valid)
  {
    angle = -angle_sum;
  }
  else
  {
    angle = 0;
  }
  if (x_count == 1)
  {
    worst_sigma = ls_valid[0]?sigma_e[0]:sigma_e[1];
    worst_sigma_theta = ls_valid[0]?sigma_e_theta[0]:sigma_e_theta[1];
    worst_sigma += 0.5f*SIGMA2_W;
  }
  else if (x_count == 2)
  {
    float temp_lanewidth = fabsf(ls_pos[1]-ls_pos[0]);
    worst_sigma = a*a*sigma_e[1] + (1-a)*(1-a)*sigma_e[0] + (0.5f-a)
        *(0.5f-a)*SIGMA2_W;
    worst_sigma_theta = b*b*sigma_e_theta[1] + (1-b)*(1-b)*sigma_e_theta
        [0];
  }
  if (2==x_count && sigma_e[0] < 4*SIGMA2_X && sigma_e[1] < 4*SIGMA2_X)
  {
    lanewidth = (1-lwfilt)*lanewidth + lwfilt*(ls_pos[1]-ls_pos[0]);
  }
```

```
// move points which weren't detected to the most logical position for
    the next time step
for (side = 0; side<2; side++)
{
  for (point=0;point<NUMBER_OF_POINTS;point++)
  {
    if (!tracked_points_detected[side][point])
    {
      tracked_points_searchwidth[side][point] = (u8)min(
          MAX_SEARCHWIDTH,max(MIN_SEARCHWIDTH,
          tracked_points_searchwidth[side][point] + 2));
      // reset j to original position
      if (0 == point%2)
      {
        tracked_points_j[side][point] = 0 + FORCE_SPACING + (point/2*(
            GRID_HEIGHT))*2.0f/(float)NUMBER_OF_POINTS;
      }
      else
      {
        tracked_points_j[side][point] = 0 - FORCE_SPACING + ((point
            /2+1)*(GRID_HEIGHT))*2.0f/(float)NUMBER_OF_POINTS;
      }
      tracked_points_z[side][point] = zvals[tracked_points_j[side][
          point]];

      // now try to intelligently set i
      if (pos_valid && angle_valid)
      {
          float est_pos = xout + tracked_points_z[side][point]*(angle
              - ls_a*tracked_points_z[side][point]);
        tracked_points_i[side][point] = (u8)(GRID_WIDTH/2-GRID_WIDTH
            *((-est_pos + (side-0.5f)*lanewidth)/X_WIDTH));
      }
      else if (pos_valid)
      {
        // assume angle was 0
        tracked_points_i[side][point] = (u8)(GRID_WIDTH/2-GRID_WIDTH
            *((-xout + (side-0.5f)*lanewidth)/X_WIDTH));
      }
      else
      {
        // reset to original position
        tracked_points_i[side][point] = side==0?((GRID_WIDTH*3))/4:(
            GRID_WIDTH/4);
      }
      tracked_points_x[side][point] = xvals[tracked_points_i[side][
          point]]+0.5f*xspacing;
    }
  }
 }
 // adjust brightness based on detection counters
 brightval -= BRIGHTINC * ((NUMBER_OF_POINTS*4*(2*VERT_SEARCHWIDTH+1)
     *2)/3 - total_detected);
 brightval = brightval<BRIGHTMIN?BRIGHTMIN:brightval;
 brightval = brightval>BRIGHTMAX?BRIGHTMAX:brightval;
 valid = pos_valid;
}

void prepare_for_processing(void)
{
 u16 i;
 // clear out computed array
 for (i=0;i<GRID_WIDTH*GRID_HEIGHT/4;i++)
 {
   *(((u32 *)&overhead_view_computed[0])+i)=0;
 }
}

void Main_Process_Image(u16 *image)
{
```

```
imageData = image;

// initialize the image processing algorithm
prepare_for_processing();

// compute the "overhead view" (resample the image into X_v)
compute_overhead_view_partial(0,GRID_HEIGHT,0,GRID_WIDTH);

// identify the lane boundaries and compute position and orientation
    estimates
line_find_new();

// check if line_find_new() was successful
if (!valid)
{
  // clear orientation
  currenttheta = 0;
  deadtime++;

  // clear estimator
  x_lqe[0] = 0;
  x_lqe[1] = 0;

  // apply control
  outputint = determine_control(actualpos,
                                currenttheta,
                                currentdist,
                                currentsw,
                                control_mode,
                                valid,
                                &control_output);

}
else
{
```

```
if (deadtime>0)
{
  x_lqe[0] = xout;
  x_lqe[1] = angle;
}

// update lqe
float x_pred[4];
float L_lqe[3][2];
float A_lqe[3][3];

    // set a reasonable initial guess for P
float P[9] = {0.00052,0.000005,0.000024,0.000005,0.000002,0.000010,
   0.000024,0.000010,0.0005};
float R[4] = {0,0,0,0};
float Q_lqe[9];
float A[9];
float Atr[9];
float B[6] = {1,0      ,      0,1      ,      0,0};
float BT[6] = {1,0,0      ,      0,1,0};

float temp1_3x3[9];
float temp2_3x3[9];
float BTP[6];
float PB[6];
float temp1_2x2[4];
float temp2_2x2[4];
float BTPB[4];
float BTPA[6];
float temp1_2x3[6];
float temp2_2x3[6];
float temp1_3x2[6];
float L[6];

unsigned char i,j;
```

```
// fill in R matrix
R[0] = worst_sigma;
R[3] = worst_sigma_theta;

// create Q_lqe matrix
Q_lqe[0] = Q11*EDT   +   Q22*EDT*EDT*EDT/3.0*speed_ms*speed_ms   +
       Q33*CONTROL_SCALE*CONTROL_SCALE*EDT*EDT*EDT*EDT*EDT*0.05*
    speed_ms*speed_ms*speed_ms*speed_ms;
     Q_lqe[1] = Q22*EDT*EDT*0.5*speed_ms   +   Q33*CONTROL_SCALE*
         CONTROL_SCALE*EDT*EDT*EDT/3.0*speed_ms*speed_ms*speed_ms;
     Q_lqe[2] = Q33*CONTROL_SCALE*EDT*EDT*EDT/6.0*speed_ms*speed_ms;
     Q_lqe[3] = Q_lqe[1];
     Q_lqe[4] = Q22*EDT   +   Q33*CONTROL_SCALE*CONTROL_SCALE*EDT*EDT
         *EDT/3.0*speed_ms*speed_ms;
     Q_lqe[5] = Q33*CONTROL_SCALE*EDT*EDT*0.5*speed_ms;
     Q_lqe[6] = Q_lqe[2];
     Q_lqe[7] = Q_lqe[5];
     Q_lqe[8] = Q33*EDT;

// create A' matrix
mat_copyf(A_lqe_ns,9,Atr);
Atr[1] *= speed_ms;
Atr[2] *= speed_ms*speed_ms;
Atr[5] *= speed_ms;
for (i=0;i<9;i++)
{
  j = i/3;
  A_lqe[j][i-3*j] = Atr[i];
}

// compute A'
mat_transf(Atr,3,3,A);
```

```
// iterate DARE a few times  P = (Q + A'PA) - (B'PA)' (R + B'PB)
   ^(-1) (B'PA)
for (i=0;i<8;i++)
{
  mat_multf_3x3_3x3(Atr,P,temp1_3x3);
  mat_multf_3x3_3x3(temp1_3x3,A,temp2_3x3);
  mat_sumf(temp2_3x3,Q_lqe,9,temp1_3x3);
  mat_multf_2x3_3x3(BT,P,BTP);
  mat_multf_2x3_3x3(BTP,A,BTPA);
  mat_multf_2x3_3x2(BTP,B,BTPB);
  mat_sumf(BTPB,R,4,temp1_2x2);
  mat_invf_2x2(temp1_2x2,temp2_2x2);
  mat_multf_2x2_2x3(temp2_2x2,BTPA,temp1_2x3);
  mat_transf(BTPA,2,3,temp1_3x2);
  mat_multf_3x2_2x3(temp1_3x2,temp1_2x3,temp2_3x3);
  mat_subf(temp1_3x3,temp2_3x3,9,P);
}

// now compute L = PB (R + B'PB)^(-1)
mat_multf_3x3_3x2(P,B,PB);
mat_multf_2x3_3x2(BT,PB,BTPB);
mat_sumf(BTPB,R,4,temp1_2x2);
mat_invf_2x2(temp1_2x2,temp2_2x2);
mat_multf_3x2_2x2(PB,temp2_2x2,L);

// fill in L_lqe
for (i=0;i<6;i++)
{
  j = i/2;
  L_lqe[j][i-2*j] = L[i];
  //L_lqe[j][i-2*j] = L_lqe_60[j][i-2*j];
}

x_pred[0] = A_lqe[0][0]*x_lqe[0]+(A_lqe[0][1])*x_lqe[1]+(A_lqe
    [0][2])*x_lqe[2]+control_delay*(B_lqe_ns[0]*speed_ms*speed_ms);
    x_pred[1] = A_lqe[1][0]*x_lqe[0]+A_lqe[1][1]*x_lqe[1]+A_lqe[1][2]*
        x_lqe[2]+control_delay*(B_lqe_ns[1]*speed_ms);
    x_pred[2] = A_lqe[2][0]*x_lqe[0]+A_lqe[2][1]*x_lqe[1]+A_lqe[2][2]*
        x_lqe[2]+control_delay*B_lqe_ns[2];
    x_lqe[0] = x_pred[0] + L_lqe[0][0]*(xout-x_pred[0])+L_lqe[0][1]*(
        angle-x_pred[1]);
    x_lqe[1] = x_pred[1] + L_lqe[1][0]*(xout-x_pred[0])+L_lqe[1][1]*(
        angle-x_pred[1]);
    x_lqe[2] = x_pred[2] + L_lqe[2][0]*(xout-x_pred[0])+L_lqe[2][1]*(
        angle-x_pred[1]);

    // pick off the state variables
    currentpos = x_lqe[0];
    currenttheta = x_lqe[1];
    currentdist = x_lqe[2];

    // apply control
    outputint = determine_control(actualpos, currenttheta, currentdist,
        currentsw, control_mode, valid, &control_output);

    // clear deadtime
    deadtime = 0;
  }
}
```

C.3 control.h

```
#define EDT 0.03337f
#define CONTROL_FILT 0.075f
#define MAX_CONTROL_CHANGE 30
#define CONTROL_MAX 90
#define CONTROL_MIN -90
#define CONTROL_OFFSET 512
```

C.4 control.c

```
#include <cdefBF533.h>
#include <ccblkfn.h>
#include <math.h>
#include "control.h"
#include "image_processing.h"
#include "matrix_support.h"

static const float K_lqr[] = {0.389878f,13.790752f};
static float controller_output=0;
static float filtered_control = 0;
static short old_outputint = CONTROL_OFFSET;
extern float speed_ms;

short determine_servo_output(float control, short old_outputint)
{
  short output = control*125.0f;
  short last_output = old_outputint-CONTROL_OFFSET;

  if (output - last_output > MAX_CONTROL_CHANGE)
  {
    output = (last_output + MAX_CONTROL_CHANGE);
  }
  if (output - last_output < -MAX_CONTROL_CHANGE)
  {
    output = (last_output - MAX_CONTROL_CHANGE);
  }
  output = output>CONTROL_MAX?CONTROL_MAX:output;
  output = output<CONTROL_MIN?CONTROL_MIN:output;

  return output+CONTROL_OFFSET;
}

short determine_control(float actualpos, float currenttheta, float
    currentdist, float currentsw, char control_mode, bool valid, float *
    control_output)
```

```
{
  short outputint=0;

  if (valid)
  {
    // LQR controller
    float last_control = controller_output;
    controller_output = 0;
    controller_output -= K_lqr[0]*actualpos;
    controller_output -= K_lqr[1]*currenttheta;
        controller_output -= currentdist;
    filtered_control = CONTROL_FILT*controller_output + (1.0f-
       CONTROL_FILT)*filtered_control;
  }
  else
  {
    controller_output = filtered_control;
  }
  outputint = determine_servo_output(controller_output, old_outputint);
  *control_output = (float)(outputint-CONTROL_OFFSET)*0.008f;
  old_outputint = outputint;

  return outputint;
}
```