Smart Autonomous Vehicle in a Scaled Urban Environment

Devon Bates, Frayne Go, Thomas Joyce and Elyse Vernon,
Student, Bradley University: Department of Electrical and Computer Engineering
James Irwin and Jose Sanchez,
Faculty Advisors, Bradley University: Department of Electrical and Computer Engineering

***Abstract*—Autonomous vehicles are of increasing interest to researchers. However, analysis of full-scale autonomous vehicle technology is costly. The focus of this project is the design of an autonomous control system such that a 1/14 scale vehicle (RC MAN TGX 26.540 6x4 XLX) can navigate a proportionally scaled roadway. The top level behavioral objective is for the vehicle to approach an intersection, halt at the stop line, execute a right turn, and stay within lane lines at all times. A camera module (OV7670) is to be interfaced with two digital signal processors (TMS320C5515) to perform the image processing. The primary controller is implemented using a microcontroller (Stellaris LM4F120) and its output is received by a secondary controller for motor interfacing. The TMS320C5515 communicates with the Stellaris LM4F120 through an inter-integrated circuit bus. Lane detection is implemented on the TMS320C5515 using Canny/Hough estimation of vanishing points to generate a steering angle of correction. Stop sign detection is implemented using histogram oriented gradients with a support vector machine for sign classification, and has an 80% detection rate in simulation. The Stellaris LM4F120 communicates intelligent speed and steering commands to the vehicle. The control loop is closed with a photomicrosensor used for speed sensing. Speed sensor interpretation in software is within 0.4% precision; speed regulation is within 0.658% precision; tracking control is within 2% precision; and deceleration control is within 5% precision. This research has the potential to decrease the costs associated with autonomous vehicle technology thus accelerating technical advancements in the field.**

***Index Terms*—Autonomous, Canny/Hough estimation of vanishing points, Hough transform, kinematics, proportional-integral Control, pulse-width modulation,**

I. Introduction

DEFENSE research has become increasingly interested in autonomous vehicles. The Defense Advanced Research Projects Agency (DARPA) held its first Grand Challenge in 2004. The competition pitted some of the most elite schools against each other to design an autonomous vehicle that would navigate the Mojave Desert. The purpose of the event was to design an autonomous vehicle so that supplies could be moved through a war-torn region without endangering the lives of soldiers. In 2005, the team from Stanford University successfully completed the Grand Challenge along with teams from four other universities. With the successful completion of the Grand Challenge, DAPRA upgraded the event to the Urban Challenge, which took place in an abandoned Air Force base [1]. The objective of the Urban Challenge was to design an autonomous vehicle such that the vehicle could obey all traffic regulations while avoiding other obstacles and merging with traffic. The main difficulty of the Urban Challenge was to design software to make intelligent decisions in real-time, unlike the Grand Challenge, which was more structured [1]. Recently, Google announced its Google Driverless Car project, a fleet of full-scale autonomous vehicles which have logged over 300,000 hours on roadways, navigating them completely accident free [2]. Because of the work done by Google and researchers for the DARPA competitions, autonomous vehicles have been legalized in three states as of 2013: Nevada, Florida, and California [2].

Autonomous vehicles have far reaching benefits, including increasing road safety and allowing those with physical disabilities an independence not otherwise afforded to them. Autonomous vehicles have the potential to increase road safety by reducing the number of motor vehicle related deaths. The U.S. Census Bureau estimates that over 35,000 people died in 2009 alone in motor vehicle related accidents [3]. By reducing the amount of human error, autonomous vehicle technology may dramatically reduce the number of automobile-related deaths once fully integrated into society. The lead engineer of Google's driverless car project, Sebastian Thrun, estimates that autonomous vehicle technology has the potential to reduce the number of automobile accidents by 90% [4]. Autonomous vehicles will also allow those with physical impairments the ability to drive. People who suffer from seizures and those who have full or partial paralysis will now be afforded the independence of driving.

The Smart Autonomous Vehicle in a Scaled Urban Environment project was designed to bring specific research applications of autonomous vehicle analysis to a small scale. The smaller scale allows schools with limited research funding the ability to examine autonomous vehicle technology without the high costs associated with full-scale analysis.

II. Problem Formulation

The primary objective is the design of a vision-based control system such that a vehicle autonomously reaches a constant speed, detects a stop sign, engages brake lights and a right turn signal, decelerates to a halt at the stop line, executes a right-hand turn into the appropriate lane, and navigates within lane boundaries at all times during operation. For the vehicle operate autonomously, the design requires an environment-detecting sensor, means for interpretation of data from the environment-detecting sensor, and motion control capabilities.

Therefore, the system was designed by integrating two subsystems: an image processing subsystem and a vehicle control subsystem.

A. Architecture Design

The high-level system architecture design is shown in Fig. 1. The image processing subsystem consists of a camera to gather data from the environment and a digital signal processor (DSP) to analyze the data. The choice of the Omnivision OV7670 camera was due to its relatively low cost and low power dissipation. The use of the DSP (TMS320C5515) is necessary because these devices have high speed performance as required for real-time image processing. The vehicle control subsystem is comprised of a microcontroller (MCU) (Stellaris LM4F120H5QR), a secondary, commercial-off-the-shelf controller called the Multi-Function Controller (MFC), and other peripheral circuitry. The MCU is used for its ability to output pulse-width modulated (PWM) signals, its low cost, and its robust diagnostic software tools. The PWM signal generation is necessary to communicate with the MFC for control of the vehicle motors used for throttle, steering, and gear shifting. A low-cost speed sensor was implemented with the use of a photomicrosensor to create closed-loop motion control.

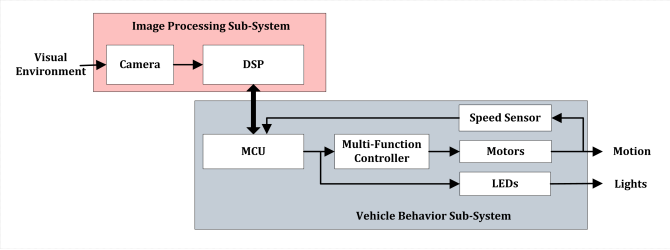


Fig. 1: System Architecture

The vehicle chosen was the Tamiya MAN TGX 26.540 6x4 XLX semi-truck cab, which is a replica at a 1/14-scale. The vehicle is radio-controlled in its original application, which simplified the initial modeling and ultimately provides the means to manually override the vehicle control software for emergency stopping. The bed of the truck also provides space for housing the system.

B. Functional Requirements

Achieving the project objective required elaboration of specific functional requirements. These requirements were established based upon the high level objectives for the vehicle behavior, for the image processing software on the DSP, and for the controller software on the MCU.

1) Vehicle Behavior: To emulate safe driving in an urban environment, the vehicle's behavioral requirements are based on scaling down the rules of the road for a full-size truck operating in the state of Illinois. One critical requirement is that the vehicle shall come to a full and complete stop at or before the stop line before proceeding into the intersection. The vehicle shall wait for a minimum of 2 seconds before proceeding into the intersection and the vehicle shall drive at a speed no more than 3.2 miles per hour. The speed requirement is based on a full-size vehicle speed of 45 mph scaled down to a 1/14 scale. The vehicle shall also be able to stay within the lane lines at all times for the safety of all drivers, passengers, and pedestrians. A radio-controlled safety switch shall be included to stop the vehicle immediately for protection of the equipment. The Illinois rules of the road states that a driver must engage a turn signal at least 100 ft away from the stop sign in non-highway areas [5]. When 100 ft is reduced to the 1/14 scale, this distance is approximately 7 ft. Therefore, the vehicle shall engage a turn signal and brake lights at least 7 ft from the stop line.

2) Image Processing: To integrate the image processing subsystem into the overall system, there are several requirements regarding equipment compatibility, sufficient operational speed, and performance of specific tasks. The camera shall be able to output 1 megapixel of image data. The camera shall be able to send data at a rate of 30 frames per second. The image data from the camera shall be a color image containing 3 layers in order to perform the stop sign detection.

For the image processing algorithms to complete in time for the vehicle control subsystem to adequately respond to visual data, the DSP shall be able to complete the required image processing within 50 ms. The DSP shall send a stop sign detection flag to the MCU after having detected a stop sign. Once the stop sign has been detected, the DSP shall begin to search for the stop line to calculate the distance between the vehicle and the stop line. For the lane line detection functionality, the DSP shall be able to detect and transmit the orientation of the vehicle within the lane lines within 250 ms.

3) Vehicle Control: Functional requirements have been established for the vehicle control subsystem to achieve the necessary control over the vehicle's motion and light-emitting diode (LED) indicators. Analysis of the high-level behavioral objective prompted the requirement that vehicle control software shall contain algorithms for speed regulation, lane tracking, deceleration control, and turning.

The MCU shall be able to communicate to the steering servo a desired steering angle given in degrees. The MCU shall be able to set the transmission to gear 1, 2, or 3. It shall also be able to control vehicle velocity and determine from a speed sensor the actual velocity at which the vehicle is operating. The main loop of the MCU shall focus on vehicle speed regulation and lane tracking. The MCU shall be set up to receive from the DSP a variable that describes the distance between the vehicle and the stop line, as well as a variable that describes the vehicle's required corrective steering angle. The MCU shall repeatedly use kinematic equations to calculate the required deceleration to come to a full and complete stop. The MCU shall engage the brake lights and turn indicator immediately after a stop sign has been detected by the image processing subsystem. One iteration of the deceleration control algorithm shall execute within 50 ms. After being stopped at the stop line for two seconds, the MCU shall instruct the vehicle to make a 90° right turn of smallest possible radius after the vehicle reaches the distance of straight travel necessary to reach the

center of the lane. After completion of the turn, the MCU shall communicate to the DSP that the turn has been completed and the DSP shall resume looking for a stop sign.

III. Methods

The image processing and vehicle control subsystems were independently developed. System development required modular implementation of a camera, image processing software, inter-integrated circuit communication, and a vehicle control subsystem.

In the image processing subsystem, the image data is processed to determine the location of the lane lines and to detect a stop sign. MATLAB (Mathworks, Natick, MA) was used to perform algorithm verification and testing. Algorithms were chosen based on their compatibility with the hardware and functional requirements. Once the algorithms had been selected and completed in simulation, they were implemented in C on a DSP.

The DSP sends a bit to the MCU indicating the detection of a stop sign as well as a corrective steering angle for use by the vehicle control system. The signals are sent across an inter-integrated communication (I^2C) bus. Detection of a stop sign initiates a deceleration control algorithm stored in code memory of the MCU. Communication of a corrective angle initiates a lane tracking algorithm to orient the vehicle straight within the lane.

Modeling and analysis of the vehicle was performed using ground testing data. These procedures produced an overall model of vehicle motion. Motor control has been achieved by reverse engineering the MFC designed by Tamiya Inc. In its intended application, the MFC receives instructions from a radio receiver. Software on the MCU has been implemented to replace the de-modulated radio signals to achieve vehicle motion. These radio signals were analyzed and replicated in the initial stages of the project.

A. Capturing Images from Camera

The camera was chosen because of its high signal-to-noise ratio, low operating voltage, low power consumption, and high frame rate. Figure 2 shows the interface between the camera and DSP. The interface consists of four different signals for communication grouped into three buses. The external clock is the operation bus which allows for synchronized operation between the DSP and the camera. The Serial Camera Communication Bus (SCCB) is used to configure the parameters of the camera. The parallel data line and the sync line form the image bus which is responsible for the transmission of image data.

1) Operation Bus: In order for the communication between the camera and the DSP to be established, the camera requires both a power source and a clock source. The DSP is responsible for controlling how fast the camera outputs image data by transmiting a clock signal. An external clock source between 10 and 54 MHz is ideal for image output [6]. 12.5MHz was used in the design.

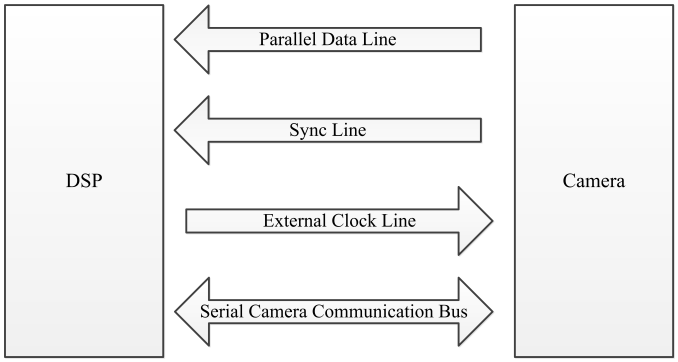


Fig. 2: Camera interfacing with I/O signals

2) Configuration Bus: For the communication between the camera and the DSP to be established, the camera requires both a power source and a clock source. The DSP is responsible for controlling the image data rate by transmitting a clock signal to the camera. An external clock source between 10 and 54 MHz is ideal for image output [6]. A frequency of 12.5 MHz was selected to allow the DSP to be able to synchronize and read the image bus based on the limited sampling frequencies on its IO ports.

$\mathbf{D_6 \ldots D_0}$	R/W	X

Fig. 3: Phase Information

At idle state, SIO_C and SIO_D are high. To start a transmission, SIO_C will pull down from idle state to activate SCCB protocol between the devices. This allows the transmission of phases through SIO_D line, which consists of a set of 9-bit data as shown in Fig. 3. A phase is transmitted from most significant bit to least significant bit. The first 7-bits, D_6 through D_0 are data used for register addresses or commands, while the 8th bit, *R/W*, is the read or write bit. The last bit is the acknowledge bit, *X*, which is ignored by Omnivision SCCB standards [6]. The 9-bit phases are read at active high when SIO_C generates pulses. Once transmission is complete, SIO_C and SIO_D will pull up to idle state to halt SCCB protocol.

The camera performs two functions: reading from the DSP and writing to the DSP. The write transmission shown in Fig. 5 can be divided into three phases. The DSP first transmits the IP address of the camera to notify the camera to begin SCCB communication. In the second phase, the camera receives a 7-bit address which is associated with one of its configuration registers and allows the DSP to transmit data to the associated configuration register in the last phase. The read transmission illustrated in Fig. 5 is divided into two 2-phase transmissions. The master first broadcasts a pointer to a camera's configuration register, which then transmits what information is within the register.

The OV7670 has the ability to adjust the images transmitted to the DSP. By default, the camera is setup to output a luminance, red-luminance, and blue-luminance (YUV), 640x480

IP Address	Sub Address	Write Data

Fig. 4: 3-Phase Write Transmission

IP Address	Sub Address	Pause	IP Address	Read Data

Fig. 5: 2-Phase Point and Read Transmission

VGA resolution image. Instead of VGA resolution the output is adjusted to be a 320x240 quarter video graphic array (QVGA) resolution image. This choice was made because the DSP has a limited amount of space to process the image and a QVGA image takes up less memory than a VGA resolution image. Also, a red, green, and blue (RGB) image is preferred to avoid color transformation computation of an YUV image to a RGB image.

3) Image Bus: In total, there are 11 signals on the image bus including three sync lines and eight parallel data lines. The parallel data lines operate at the same frequency of the external clock signal. The sync lines are used for timings to read information pixel by pixel, read the next row of pixels, and read the next image. The 12.5 MHz clock provided by the DSP is also the same frequency for the data lines. Sync timings for the image bus indicate that a transmitted image will be captured within 30-32 ms [6]. For the stop detection, DSP will have 18-20 ms remaining after receiving the transmitted image to process an image and send data to the MCU. For lane line detection, the processing window for the image processing and data transfer is 218-220 ms.

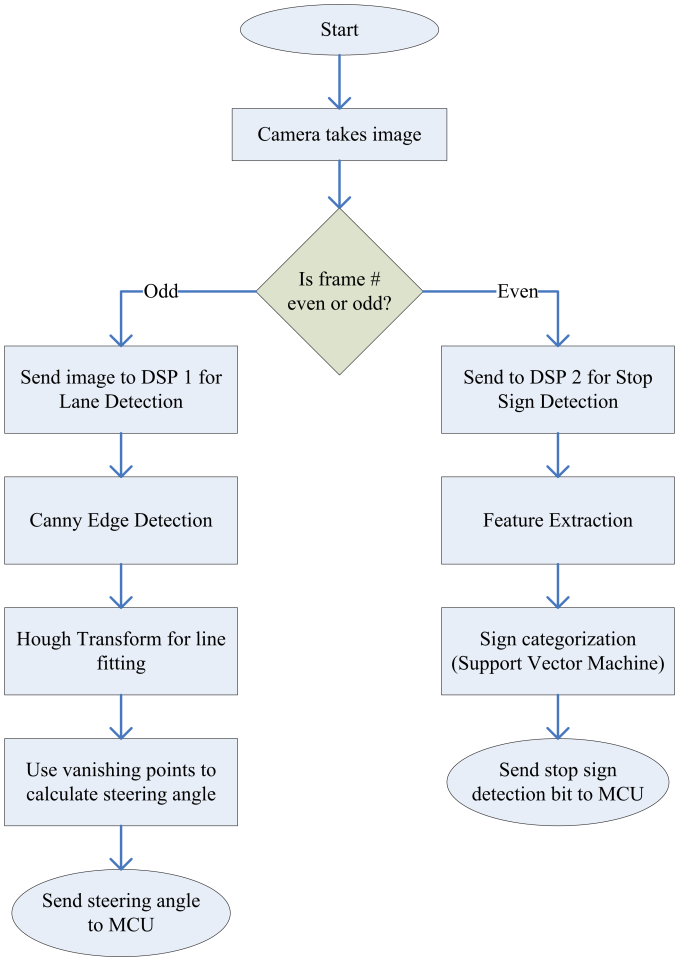


Fig. 6: Image Processing High Level Software Flowchart

B. Image Processing

Two algorithms are operating on the DSP that comprise the image processing subsystem. One of the algorithms is locating the lane lines on the road and the other is detecting the stop sign. The two independent algorithms run concurrently to analyze each frame imported from the camera. Within each frame, a 320x240 image is acquired from the camera. Once the stop line and stop sign detection calculations are performed, the DSP will transmit the stop-bit and lane correction signals to the MCU. The stop sign detection function transmits a bit to the MCU when a stop sign is positively classified. The lane line detection transmits the calculated value for the corrective steering angle to the MCU every 250 ms. The steering angle (in degrees) is transmitted to the MCU so that lane tracking can be performed in the vehicle control subsystem. Once a stop sign has been detected, the lane line detection also transmits the distance between the vehicle and the stop line to the MCU.

The intention for the project is to process the visual environment by utilizing two separate DSPs which are both connected to the camera. Using two DSPs allows each function to utilize more memory space and processing time. Also, using two DSPs allows the lane detection and stop sign detection functions to be performed simultaneously.

1) Stop Sign Detection Algorithm: Stop sign detection is performed in two main steps: feature description and image classification. The feature description is performed using the histogram of oriented gradients (HOG). The feature data is then classified as being a stop sign or not by a support vector machine (SVM). The SVM has two main processes, the teaching process, which is performed externally in MATLAB and the testing process, which classifies the image and is performed on the DSP in real-time.

a) Histogram of oriented gradients:

The first step in detecting a stop sign is generating the feature data from the image. The feature values are calculated for each i^{th} sub-block, j^{th} template and k^{th} orientation. The sub-block is a 3x3 block in the image; the template is a 3x3 masking template; and the orientation is the discrete representation of the edge direction. The feature values are calculated using gradient magnitudes and edge direction. A gradient magnitude is the difference between pixel intensity when comparing a pixel and its adjacent pixels. To calculate the gradient intensity, the derivatives are calculated in the x- and y-direction [7]

$$G_{rx}(x,y) = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * I_r(x,y) \tag{1a}$$

$$G_{ry}(x,y) = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T * I_r(x,y), \tag{1b}$$

which are then used to calculate the magnitude and direction of the gradient [7]

$$G_r(x,y) = \sqrt{G_{rx}(x,y)^2 + G_{ry}(x,y)^2} \tag{2a}$$

$$\Theta_r(x,y) = \tan^{-1} \frac{G_{ry}(x,y)}{G_{rx}(x,y)}. \tag{2b}$$

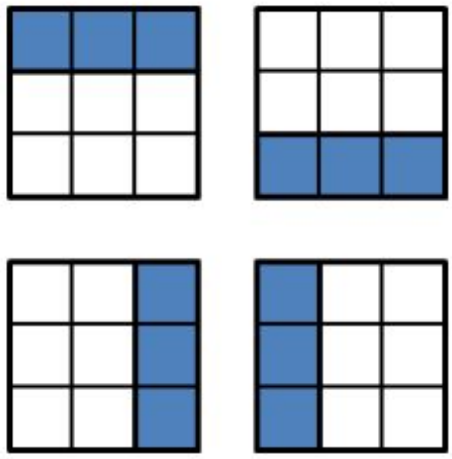
The direction is originally calculated over the range from -180° to 180°; however, for calculation purposes the angles are sorted into 8 bins. For bin sorting, which is shown in Fig. 7(b), any angles contained in the yellow triangle are sorted into Bin 1. Each of the pixels in the *k* bin arrays, Ψ_{rk}, contains the magnitude of the pixel if its corresponding edge orientation is sorted into that particular bin or a 0 if the edge orientation is sorted into a different bin (3). The gradients are calculated on each layer of the RGB image. Although the main layer is the red layer, the green and blue layers are used to weight the results which accounts for changes in lighting. Once the gradient magnitudes and edge directions have been calculated for each pixel, templates are used to organize the data into feature values. A template is a 3x3 pixel sub-block that masks out, or ignores, irrelevant pixel data. There is a set of four templates that are used which are shown in Fig. 7(a). Blue shaded pixels would be masked out in each template.

$$\Psi_{rk}(x,y) = \begin{cases} G_r(x,y) & \text{if } \Theta_r(\text{x,y}) \in \text{bin}_k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

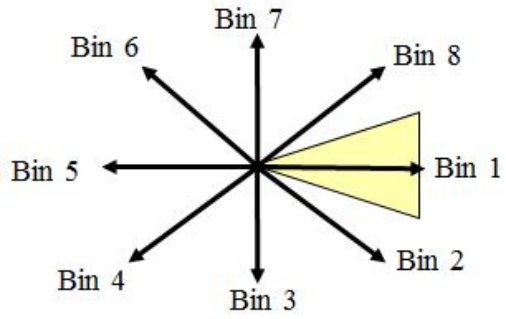
The gradient calculations for the red layer are divided by the summation of the calculations for all three layers and combined using masked sub-blocks, where C_j is the set of non-masked pixels in the j^{th} template and B_i denotes the i^{th} sub-block.

$$f_r(i,j,k) = \frac{\sum\limits_{(x,y)\in C_j} \psi_{rk}(x,y)}{\sum\limits_{(x,y)\in B_i} G_r(x,y) + G_g(x,y) + G_b(x,y)} \tag{4}$$

The weighting of the gradients is important for the robustness of the algorithm. Weighting the red layer with the summation of the three layers allows the HOG algorithm to be an effective image descriptor regardless of variations in levels of ambient light. In addition, HOG uses the RGB color scheme, which cuts out the need for color space transformations lower the overhead computational time. The number of feature values in the resulting feature value vector depends on the number of sub-blocks, templates, and bins.



(a)



(b)

Fig. 7: (a)Masking Templates and (b) Gradient Calculations for HOG

b) Support Vector Machine:

The second step in recognizing the stop sign is classification, which is implemented using a Support Vector Machine (SVM). A SVM is a supervised learning model that is used to analyze data and recognize patterns, usually used for classification and regression analysis. The SVM is based on the margin maximization principle. To train the SVM, feature vectors are extracted from multiple images of the target classification. Each set of feature values is classified based on whether it contains a stop sign or not. The types of classifications can be diversified to include full stop signs and partially obscured stop signs, distant stop signs, etc. so the detection algorithm would be robust in every circumstance. For the scaled urban environment, the algorithm will focus solely on full stop sign recognition. The SVM maps input vectors into a high dimensional feature space through non-linear mapping [8]. A hyperplane is a plane present in the high dimensional space that separates the two classifications. Equation (5) outlines the representation of the linear optimal hyperplane, *u*, which is defined by the normal vector, *w*, and the threshold vector, *b* [9].

$$u = \overrightarrow{w} \cdot \overrightarrow{x} - b \tag{5}$$

The hyperplane separates the two classifications of stop signs at *u* equals to 0. An optimal hyperplane is part of the linear decision function and is the hyperplane with maximal margin between the vectors of the two classes. The SVM computes the solution for the optimal hyperplane using the example support vectors in the training stage. The optimal hyperplane is made of the specific solution of *w0* and *b0* that separates the training data into two groups with a maximal margin. In other words, the SVM determines the hyperplane so that the distance between the projections of the two different classes is maximal. Equation (6) shows the formula for determining the maximum margins between the hyperplane.

$$m = \frac{1}{\|w\|_2} \tag{6}$$

The normal vector, *w*, and the threshold, *b*, are the components used to calculated the maximal margins which can be calculated using the Lagrangian multipliers. Typically, calculating *w* and *b* is a quadratic programming problem. However, without access to any quadratic programming libraries, the hyperplane was calculated using sequential minimal optimization (SMO). Instead of solving the problem for the entire set of Lagrangian multipliers, SMO solves the problem for two individual Lagrangian multipliers and then uses several testing states to determine when the solution is optimized for the whole system. The two components of the optimal hyperplane are calculated using (7) once the Lagrangian multipliers are calculated [8]. The optimal solution for the normal vector, *w0*, is the summation of the products of the Lagrangian multipliers αi and the support vectors z_i.

$$\boldsymbol{w_0} = \Sigma \alpha_i \boldsymbol{z_i} \tag{7}$$

Once the learning step has been performed, the optimal hyperplane is loaded into memory for the testing stage. The testing stage begins with the calculated feature data as the input into the SVM as a vector *z*. The decision function for an image to be classified as a stop sign is [8]

$$I(\boldsymbol{z}) = sign(\boldsymbol{w_0} \cdot \boldsymbol{z} + b_0). \tag{8}$$

If I(z) is greater than 0, then the vector is classified as a stop sign. If I(z) is less than 0, then the data is classified as not a stop sign.

2) Lane Detection Algorithm: The lane detection function detects and calculates the lane lines for navigation purposes. The algorithm being used is the Canny/Hough edge detection with vanishing points. There are three main stages of the lane detection algorithm. First, Canny edge detection analyzes the image, producing binary edge data that describes all edges in the image. Second, the Hough transform uses the binary edge data and performs line fitting to calculate the line data. Lastly, the line generated by the Hough transform is used to calculate the vanishing points which can be used to determine the navigation angle for the car to adjust the orientation.

a) Canny Edge Detection:

Canny edge detection is performed on a black and white image. The first step in Canny edge detection is to perform Gaussian smoothing to eliminate any noise present in the image. To perform the Gaussian smoothing, a 1D Gaussian kernel is multiplied across the image in the x- and y-directions. The second step is to perform gradient calculations on the smoothed image. The gradient magnitude and edge direction are determined in the same way as the gradient calculations in the histogram of oriented gradients. Similarly to the HOG algorithm, the angles are sorted into a discrete number of bins.

After all of the gradient values are calculated, the function performs non-maximum suppression on each pixel in the image. Non-maximum suppression takes the calculated bin values for each pixel and compares the gradient magnitudes from the pixels in the indicated bin direction.

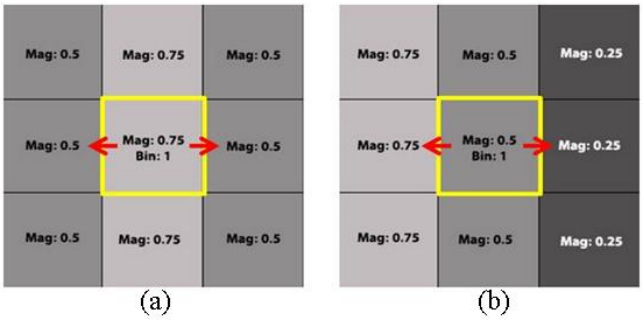


Fig. 8: Non-maximum Suppression for Canny Edge Detection where (a) is a maximum and (b) is a non-maximum

Fig. 8 shows two instances of non-maximum suppression. The bin of the center pixel in fig. 8(a) is 1, so the center pixel is compared to the pixel in the Bin 1 direction and the reverse direction. In the first example, the magnitude is a maximum, so the gradient magnitude would be kept. In fig. 8(b), the magnitude is not a maximum, so the center pixel would be masked out in the results. The last step is to perform thresholding on the remaining gradient magnitudes. Thresholding involves comparing the magnitude of each pixel with a threshold level of 10. If the pixel is greater than the threshold, that pixel is set to 1. If not, then it is set to 0. After thresholding, the image contains the binary edge data of the lane line that is used for the Hough transform.

b) Hough transform:

The Hough transform is a technique that can be used to detect lines or other parametric forms. In this study, the linear version of the Hough transform is used. The line is described by ρ and θ variables where θ is the angle of the line and ρ is the distance from the line to the origin. The Hough transform examines each pixel that is an edge. For that pixel, it cycles through all of the available θ values and calculates the corresponding value and then increments the accumulator at the indices that are equal to ρ and θ.

A simplified version of the Hough transform is shown in

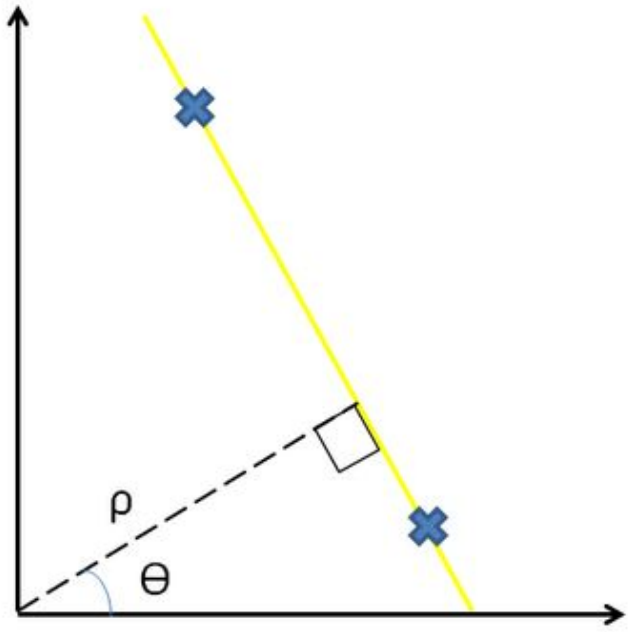


Fig. 9: Hough Transform line calculation

Fig. 9. The two points marked with an X highlight two edge pixels in the image data. The yellow line would be the linear fit between the two edge pixels. The highest value stored in the two-dimensional accumulator is representative of the line that is fit. One index represent the θ value and the other index represents the ρ value. The Hough transform is performed twice, once for the left lane and once for the right lane.

c) Vanishing Points:

The Vanishing points calculation projects the resulting lines from the Hough transform into the distance. The intersection is determined by converting the lines into slope-intercept representation. Equations (8a) and (8b) are used to determine the slope, *m*, and intercept, *b*.

$$m = \tan(\theta) \tag{8a}$$

$$b = \frac{\rho}{\theta} \tag{8b}$$

The intersection point between the left and right lane lines is used to determine the angle of orientation that needs to be transmitted to the MCU for navigation purposes. The mapping of the lines into a navigation angle requires the location and orientation of the camera on the car.

C. Inter-Integrated Circuit Communication

To facilitate communication between the MCU and the DSPs, an inter-integrated circuit (I^2C) bus was implemented. The I^2C bus allows bi-directional data transfer between two or more devices. Because the DSP performs the computationally intensive operations, the DSP serves as the master. The MCU receives and responds to the information obtained from calculations performed on the DSP through the I^2C bus. The MCU is the slave device on the bus. To communicate between devices, I^2C protocol requires two lines: a clock line (SCL) and a data line (SDA). The SCL line is used to synchronize data transfer while the SDA line contains the information to be communicated between devices. The process for data transfer between the master and the slave is as follows: first, the master selects which slave it shall communicate with by transmitting a slave address; second, the slave with the corresponding slave address responds by transmitting an acknowledge bit to the master, which verifies that the master is ready to receive data; third, the master transmits the 8-bits serial data across the SDA line; finally, the slave transmits another acknowledge bit to confirm that the information has been received.

D. Vehicle Control

The vehicle control subsystem accepts information about the vehicle's visual environment and responds with motion and LED indicator control. The subsystem responds by executing software which has been instantiated on the MCU. The software controls PWM signals for motor control to the MFC. There are three PWM signal outputs for control of the following motors: throttle (direct current motor), steering (servo motor), and gear shift (servo motor). The vehicle control subsystem also consists of peripheral circuitry including a speed sensor, a brake light amplifier, and a radio-controlled shutoff signal.

1) Software Design: The high level flowchart for the MCU software is shown in Fig. 10. The software is designed to iterate indefinitely between a speed regulation algorithm and tracking algorithm. The main loop is interrupted only when the DSP engages a stop sign detection flag. Along with stop detection information, the distance between the vehicle and the stop line is communicated by the DSP such that the vehicle can decelerate from its current speed to zero velocity upon reaching the stop line. Following the deceleration control loop, the vehicle pauses for two seconds at the stop line before finally executing a turning algorithm so that it moves into the center of the new lane.

a) Speed Regulation Algorithm:

The speed regulation model is given by the block diagram in Fig. 11. The symbolic digital multiplexer shown in Fig. 11 selects the top path only upon the first iteration of the speed control algorithm. The top path initializes the throttle PWM signal based on vehicle modeling data. The signal initialization is given by the linear model

$$\textit{PWM Timer Value} = k_p * \textit{Command} + c, \tag{9}$$

where *Command* is a command signal of the desired steady-state vehicle velocity in $\frac{\text{m}}{\text{s}}$, k_p is a constant defining the slope of the linear relationship, and c is a constant defining the zero crossing. To determine the parameters k_p and c, vehicle modeling data was measured for the vehicle with free-spinning wheels (unloaded) and on the ground (loaded). The vehicle modeling data is shown in Fig. 12. Measurements performed on the loaded system suggests that k_p = 8,321.5 and c =

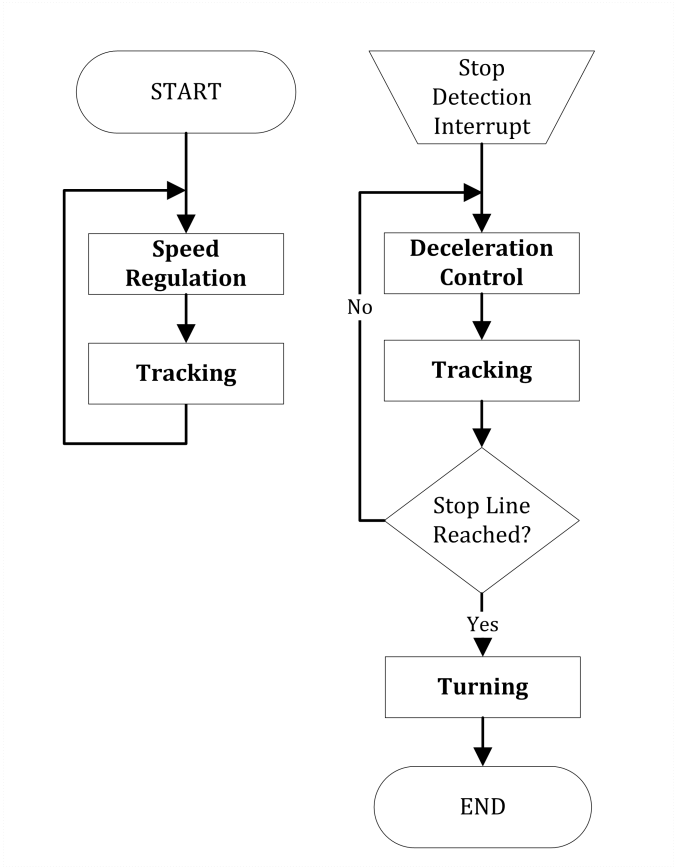


Fig. 10: MCU main loop

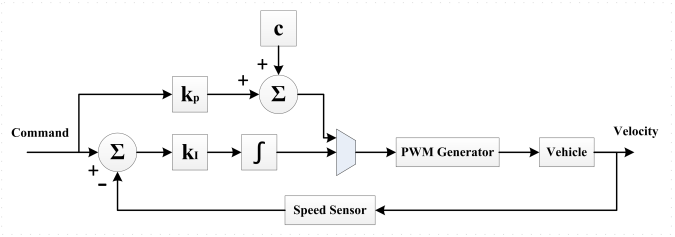


Fig. 11: Speed Control Block Diagram

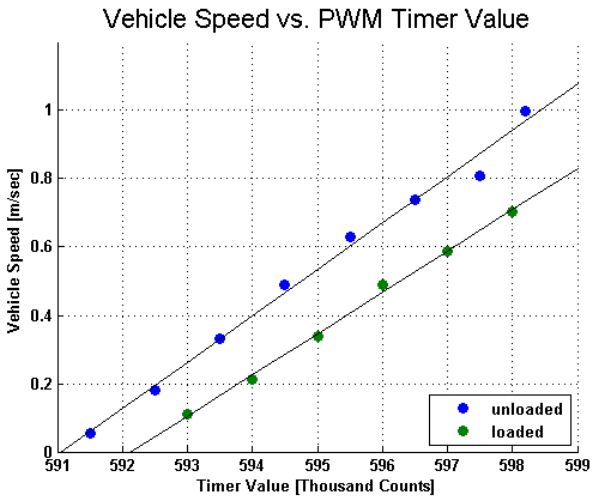


Fig. 12: Vehicle Modeling Results

592,150. The unloaded system measurements suggest a linear relationship with a lower zero crossing and a larger slope than the loaded system, which further suggests that the vehicle's weight has an effect on the relationship between vehicle steady-state speed and signal pulse width. The data in Fig. 12 could be used to determine a relationship between the weight of the vehicle and the constants k_p and c if the vehicle weight was to vary significantly. However, small variations in system weight were anticipated, suggesting that such a relationship was unnecessary for the loaded system design.

After initialization of the PWM signal, the multiplexer in Fig. 11 selects the bottom path for all remaining iterations. The bottom path is a digital proportional-integral control structure. Proportional-integral control operates by generating an error signal as the difference between the command signal (intended vehicle velocity) and system output (actual vehicle velocity). The error signal is multiplied by gain k_I and integrated. k_I can be analytically designed or tuned to optimize the output response. Integration of the error signal is performed to decrease steady-state error. the proportional-integral control structure was therefore chosen for its low steady-state error characteristic. k_I was tuned to be 1000 based on observation of vehicle behavior. The choice for k_I yields satisfactory overshoot and steady-state error for the loaded system. In ground testing, no measurable overshoot was detected; this is important for the urban operating conditions of the system, in which speeding is not acceptable. Steady-state error was measured to be within a mean value of 0.658% error, which was determined to be tolerable based on analysis of full-scale operating conditions. The command signal of 0.64 m/s being used for testing corresponds to a command signal of 20 MPH in a full-scale system. The 0.658% error suggests that for an intended command signal of 20 MPH, the system would respond with a steady-state speed between 19.86 MPH and 20.14 MPH. The variability of human drivers' cruising speed is much greater than the error for a given sample of drivers on the road, which justifies the k_I choice.

b) Tracking Algorithm:

The vehicle's control tracking algorithm uses the corrective angle transmitted by the DSP to correct the vehicle's orientation. The geometric analysis in Fig. 13 was used for initial corrective angle modeling. The geometric model assumes that for a given vehicle's steering angle, the turn radius, *R*, is given by [10]

$$R = \frac{Wheelbase}{\sin(\theta_{steer})} + \frac{Track}{2}, \tag{10}$$

where *Wheelbase* is the distance between the front and rear axle of the vehicle (0.2885 m), *Track* is the distance between wheels along the axle (0.171 m), and θ_{steer} is the steering angle on the front axle. Given *R* and a corrective steering angle, θ_{corr}, in degrees, the distance of travel along the perimeter of the circle is given by

$$D = 2\pi R * \frac{\theta^{\circ}_{corr}}{360^{\circ}}. \tag{11}$$

Using (11) and assuming constant vehicle velocity, the steering time interval, T_{steer}, can be determined by

$$T_{steer} = \frac{D}{V}. \tag{12}$$

After correcting the vehicle's orientation, the steering servo returns to a 0° steering angle for every iteration of the algorithm.

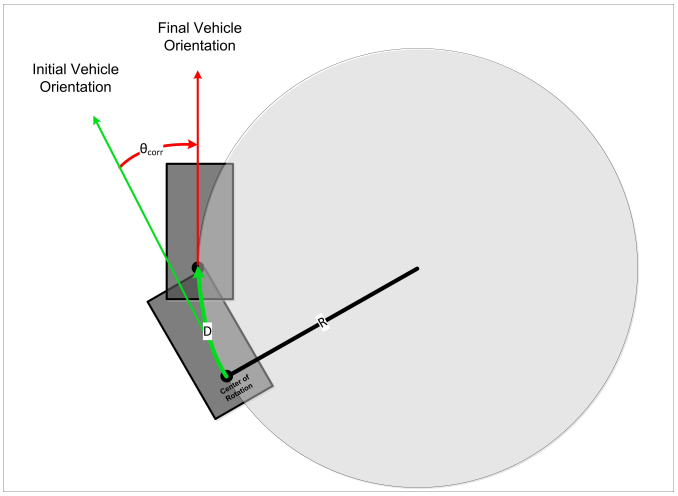


Fig. 13: Tracking Algorithm Geometric Analysis

After geometric analysis was performed, a new model for the turn radius was determined based on empirical data specific to the vehicle by measuring turn radii for a set of software-controlled steering angles. By testing three trials each of 6°, 18°, 21°, 24°, and 25° steering angles (see appendix A), a relationship was interpolated as given by

$$R = \frac{x}{\sin(\theta_{steer})}, \tag{13}$$

where x is a constant of proportionality for all steering angles. From test data, x was determined to be 0.340 with 2.59% standard deviation. Using the model obtained from empirical data, the steering time interval is represented by

$$T_{steer} = 0.00593 \left[\frac{\theta^{\circ}_{corr}}{v * \sin(\theta_{steer})} \right]. \tag{14}$$

After tracking control testing resulted in consistent error of 15% below desired orientation, the time interval in (14) was adjusted accordingly to account for this error correction by multiplying the constant *0.00593* by 115% to yield Equation (15).

$$T_{steer} = 0.00682 \left[\frac{\theta^{\circ}_{corr}}{v * \sin(\theta_{steer})} \right]. \tag{15}$$

Observation of vehicle's behavior during tracking testing revealed that for sequential corrective angles in opposing directions, the vehicle's steering performed with mechanical undershoot in the new direction. To compensate for the undershoot, a value with a magnitude of 1000 was added to the to the PWM timer value corresponding to a 0° angle after completion of each tracking algorithm iteration. In effect, a small left turn command follows a corrective angle in the right direction, and a small right turn command follows a corrective angle in the left direction. The value of 1000 was found using a tuning method and has improved vehicle orientation performance based on observation of vehicle behavior during testing.

c) Deceleration Control Algorithm:

Before the first iteration of the deceleration control algorithm, the software determines the constant deceleration, *Acc*, required to bring the vehicle from its initial velocity, v_i, to zero velocity over the distance between the vehicle's initial position and the stop line, *D*. Equation (16) shows the computation of *Acc* [11].

$$Acc = -\frac{v_i^2}{2D} \tag{16}$$

The initial velocity, v_i, in (16) is determined using speed sensor estimates. The distance *D* between the vehicle and the stop line is calculated by the image processing subsystem and is communicated to the MCU over the I^2C bus. The total stopping time, T_{stop}, is given by [11]

$$T_{stop} = \frac{2D}{v_i}. \tag{17}$$

The update rate was set at 250 ms to allow for deceleration accuracy while allowing sufficient iteration time for each tracking algorithm. Thus, the number of deceleration control iterations is calculated as

$$N = \frac{T_{stop}}{0.250}. \tag{18}$$

Fig. 14 shows a flowchart of the deceleration control algorithm, which functions to update the new velocity command each iteration using the deceleration model. To make the deceleration control update interval 250 ms, the algorithm delays for an amount of time equal to the difference between the tracking algorithm update time T_{steer} and 250 ms. For the case in which the steering time is less than 250 ms, the new velocity is set as shown in (19). When T_{steer} is larger than 250 ms, there is no additional delay and the new speed is immediately updated by multiplying the deceleration by T_{steer} as in (20).

$$v_n = v_{n-1} + Acc * T_steer \tag{19}$$

$$v_n = v_{n-1} + Acc * 0.250 \tag{20}$$

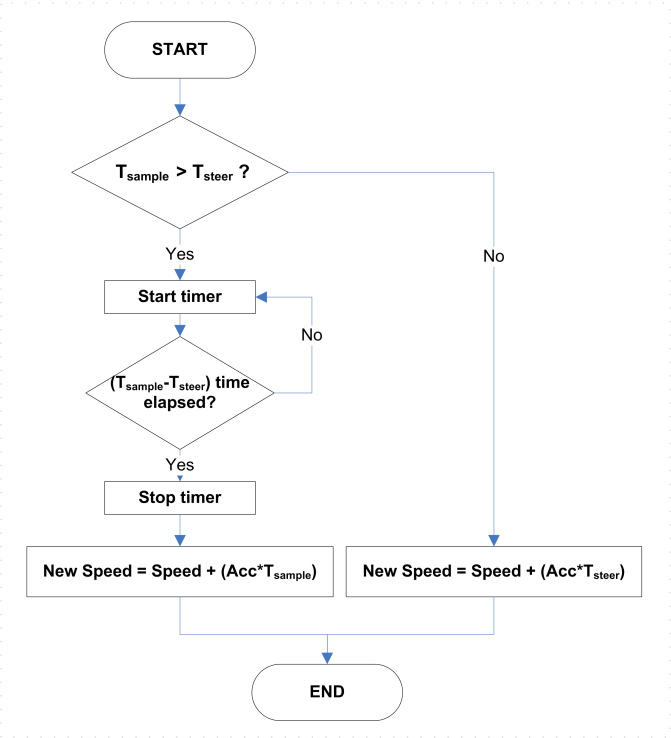


Fig. 14: Deceleration Control Algorithm Iteration

d) Turning Algorithm:

The turning algorithm executes in two stages. The first stage is the distance of straight travel, in which the vehicle drives forward at a straight steering angle for a finite distance beyond the stop line. The second stage is the 90° turn. The purpose of the first stage is to line up the vehicle's center of rotation appropriately to ensure that the vehicle ends up in the center of the lane upon completion of the second stage. The distance of straight travel, $D_{straight}$, is a function of the vehicle's turn radius, R_{veh_trn}, the intersection turn radius, R_{int}, and the distance between the front of the vehicle and its center of rotation, L. Based on measurements of the vehicle's 90° turn using a maximum 25° steering angle, the vehicle's turn radius is 0.80 m. The distance L was measured as 0.60 m. Therefore, to determine the distance of straight travel, Equation (20a)was derived using vector addition. Using the measured constants for R_{veh_trn} and L, Equation (20a) was reduced to (20c), which is only a function of the intersection radius.

$$D_{straight} = (R_{int} - R_{veh\_trn}) + L \tag{20a}$$
$$D_{straight} = (R_{int} - 0.80) + 0.60 \tag{20b}$$
$$D_{straight} = R_{int} - 0.20 \tag{20c}$$

Completion of the first stage is determined by the speed sensor and peripheral timers within the MCU that are used to update a cumulative summation distance variable compared with the constant Dstraight. This is accomplished by solving for $D_{straight}$ in (16) for an update time of 125 ms and sampling the speed sensor upon timer overflow.

In the second stage of the turning algorithm, the vehicle's steering servo is engaged to 25° until the vehicle has made a 90° turn. Based on the measured R_{veh_trn}, the 90° turn is accomplished in a distance of 1.25 m based on (15) for θ_{corr} equal to 90°. Upon completion of the second stage, the steering column is returned to 0°. The MCU software finally returns the program counter to the main loop to resume speed regulation and tracking.

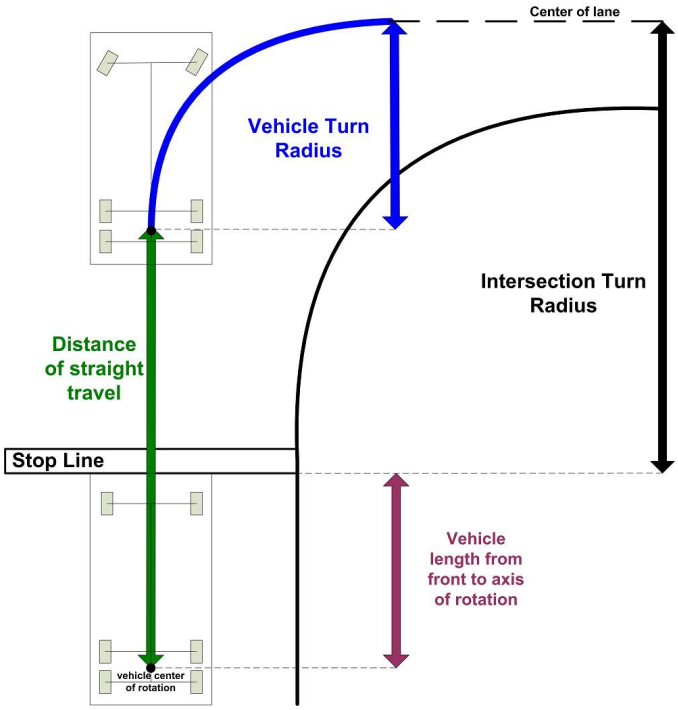


Fig. 15: Turning Algorithm

2) Interfacing with Vehicle Peripherals: To achieve control of the vehicle's motion, the design requires communication between the MCU and MFC. In addition, communication is required between the MCU and analog circuitry including a speed sensor, a brake light signal amplifier, and a radio-controlled shutoff signal.

a) PWM Signal Generation:

Interfacing the MCU with the motors on the Tamiya RC MAN TGX 26.540 6x4 XLX vehicle required specific communication between the MCU and the MFC. Communication required observation of the radio receiver output signals so that they could be emulated by the MCU. Fig. 16 shows recorded signals transmitted by the radio receiver corresponding to gear 1 (top) and gear 2 (bottom). These signals were observed and recorded using a TDS 2024B oscilloscope (Tektronix, Portland, OR). Command signals for each motor (DC throttle motor, steering servo motor, and gear shift servo motor) are identical in frequency and positive pulse width range. Each is a 56.15 Hz PWM signal with a positive pulse width in the range of 1 ms to 2 ms.

Forward vehicle velocity is achieved by setting the positive pulse width of the throttle PWM signal to a value in the range of 1 ms to 1.5 ms. 1.5 ms corresponds to zero velocity and 1 ms corresponds to maximum speed for a given gear. For the steering PWM signal, the extreme values of positive pulse

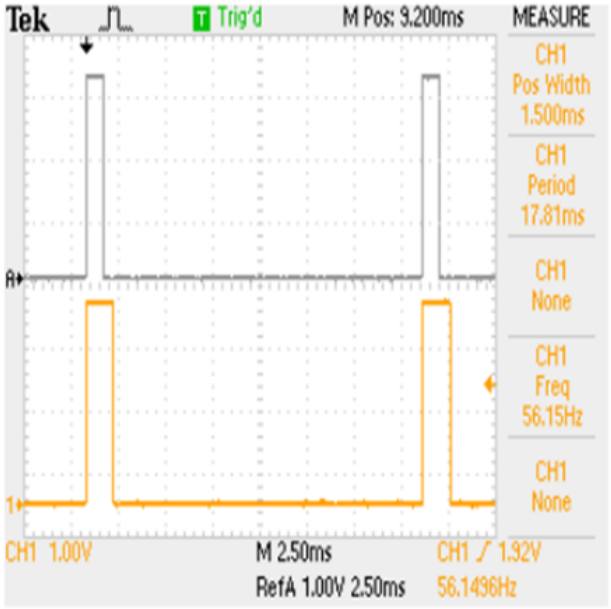


Fig. 16: PWM Signals Generated by the Radio Receiver

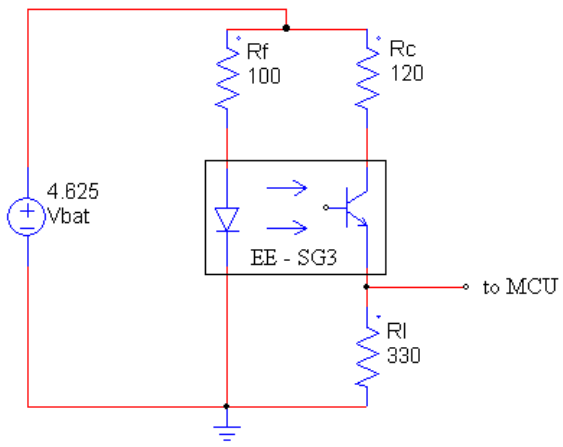


Fig. 17: Omron EE-SG8 Photomicrosensor External Circuit

widths are limited in software to the range of 1.19 ms to 1.88 ms. The limitation accounts for mechanical considerations to prevent contact between the front axle and the undercarriage of the vehicle. The range corresponds to steering control in the range of approximately -25° to 25°. Because the vehicle has three gears in which it can operate, there are three discrete values of gear shift PWM signal positive pulse widths: 1 ms for first gear, 1.5 ms for second gear, and 2 ms for third gear.

b) Speed Sensor:

The method of speed sensing chosen for the project was optical encoder of a transparent disk. The optical encoder is the Omron EE-SG3 photomicrosensor. The circuit designed for the EE-SG3 is shown in Fig. 17. The choice of R_f as 100 Ω allows for complete transistor gating and the choice of R_L as 330 Ω and R_c as 120 Ω allows for an output of 3.3 V given a battery voltage of 4.625 V, which is the voltage of the AAA battery pack mounted on the vehicle. Ensuring transistor-transistor logic (TTL) output voltage levels is critical for proper signal interpretation and for safe interfacing with internal MCU circuitry. Note that as battery voltage declines over time, the logic level high output voltage will remain above the minimum TTL level of 2 V until the battery voltage drops below 3 V.

The resistor design choices for the photomicrosensor external circuit are based on typical operating conditions in the EE-SG3 data sheet [12]. Specifically, the diode current is 34.25 mA with a forward voltage of 1.2 V across the diode (30 mA diode current is typical). The voltage drop across R_c is approximately 1.2 V, the collector-emitter saturation voltage is approximately 0.1 V, and the output voltage across R_L is approximately 3.3 V for an emitter current of 10 mA. The design allows for collector current at approximately half of maximum ratings and for typical collector-emitter saturation voltage.

The transparency disk for the optical encoder is mounted on the vehicle's propellor shaft outside of the transmission. The ratio of propellor shaft speed to vehicle wheel speed is 2.666:1 $\left[\frac{\text{rev}}{\text{rev}}\right]$. The measured wheel diameter is 0.08334 m, indicating that wheel circumference is 0.2618 m. Using the previous constants, the vehicle speed can be extracted from disk speed by (21a). Equation (21a) is reduced to (21b) in code memory to reduce calculation time and memory space.

$$VehicleSpeed\left[\frac{\text{m}}{\text{s}}\right] = DiskSpeed\left[\frac{\text{m}}{\text{s}}\right] * 0.2618[\text{m}] * \frac{1}{2.666}\left[\frac{\text{rev}}{\text{rev}}\right] \quad (21a)$$

$$VehicleSpeed\left[\frac{\text{m}}{\text{s}}\right] = DiskSpeed * 0.09821 \quad (21b)$$

Refer to Fig. 18 for speed sensor interpretation by MCU software. The speed sensor speed is estimated in MCU software by sampling the encoder input pin every 1 ms, based on a timer overflow interrupt. The count variable *Count 1* increases from 1 until a rising or falling edge is detected, at which time the count value is added to a cumulative summation variable before resetting to 1. A second count variable *Count 2* is used to count the total number of edges detected on the signal. Once 16 edges have been counted, a revolution has occurred on the disk. The number stored in the cumulative summation variable is multiplied by 0.001 to produce the total time in one revolution. Therefore, the inverse of this number is the disk speed in revolutions per second.

The sampling time of 1 ms was chosen to ensure reliable accuracy while allowing for sufficient processing time. For the MCU clock speed of 40 MHz used in the design, 40,000 clock cycles occur between samples. Because the method of speed sensing is less precise for higher vehicle velocity (in effect, less sampled data per revolution), the largest speed sensing imprecision will occur during the highest operational vehicle velocity. The steady-state vehicle velocity has been chosen to be 0.64 m/s, which is 20 MPH on a 1/14 scale. At 0.64 m/s

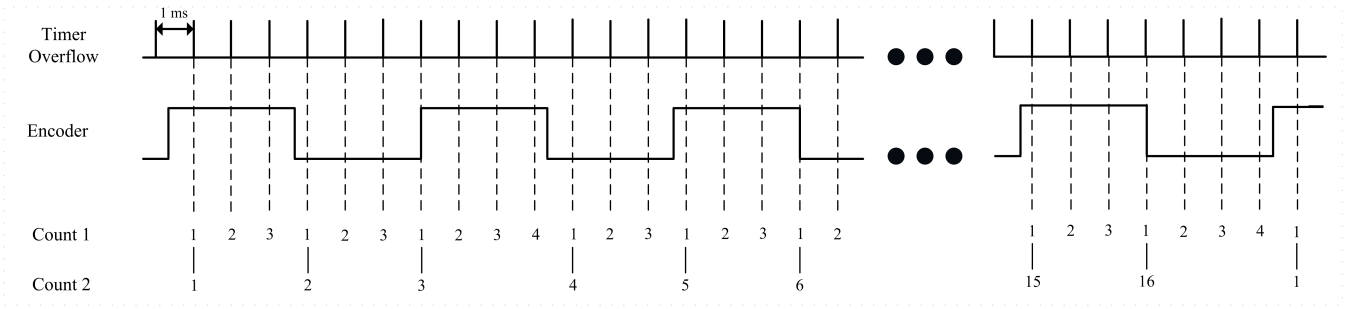


Fig. 18: Speed Sensor Software Interpretation

and at the sampling period of 1 ms, either 153 or 154 samples occur per revolution, depending on when the first edge is detected. These detected samples correspond to 0.297% error and 0.354% error in vehicle speed software interpretation, respectively.

c) Brake Light Amplifier:

For the system to meet the objectives, turn signals and brake lights were required. The vehicle's brake light LED requires greater voltage than the 3.3 V at MCU output to fully forward bias. Thus, it was necessary to design a brake light amplifier circuit to achieve LED functionality. Functionality is accomplished through the use of a PNP switching transistor circuit shown in Fig. 19. The circuit uses the 4.625 V battery pack to pull the output signal high. The 22 Ω gating resistor was chosen as a standard TTL gating value, and the 300 Ω was designed using a tuning method until the brake light LED was at optimal brightness. Due to the PNP transistor circuit configuration, the brake light gating signal from the MCU operates with active-low logic.

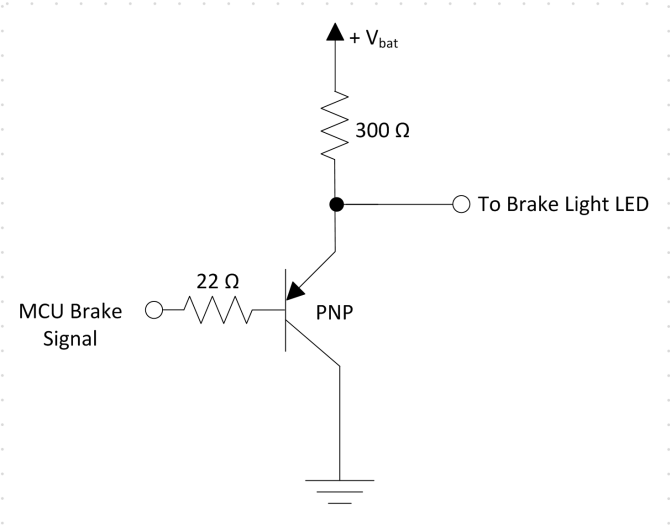


Fig. 19: PNP Transistor Circuit for Brake Signal Amplification

d) Radio-Controlled Safety Shutoff Signal:

To maintain the integrity of the equipment during ground testing, a safety shutoff signal was designed. The safety signal is transmitted by the radio receiver and is input to the MCU. Once the human-operated radio transmitter is turned on, a PWM signal is transmitted by the radio receiver which triggers an interrupt in code memory to halt output to the throttle. Upon safety signal trigger by the human operator, the vehicle comes to a stop in less than 1 ft when at operating speed of 0.64 m/s.

IV. Results

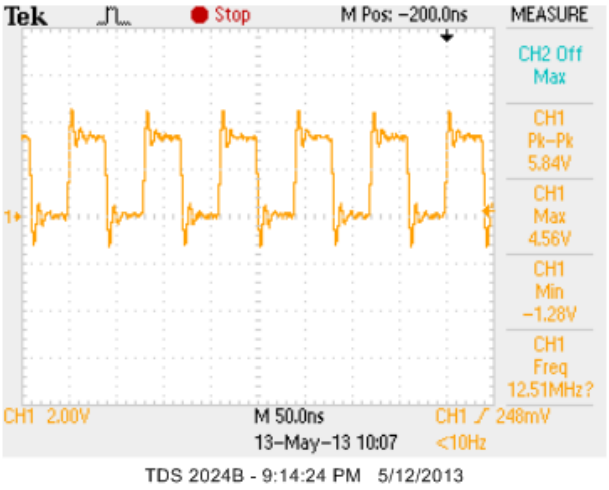
Results of the subsystems were measured using physical and analytical methods. Testing was performed for the camera, the image processing software, the inter-integrated circuit bus, and the vehicle control subsystem. The image processing subsystem was tested analytically. The lane detection has been fully implemented on the DSP and the stop sign detection has been successfully implemented in simulation. The vehicle control subsystem has been fully implemented and the physical results has been tested with the vehicle on the ground. While the subsystems were never integrated, the individual results demonstrate the potential for full system integration.

A. Camera

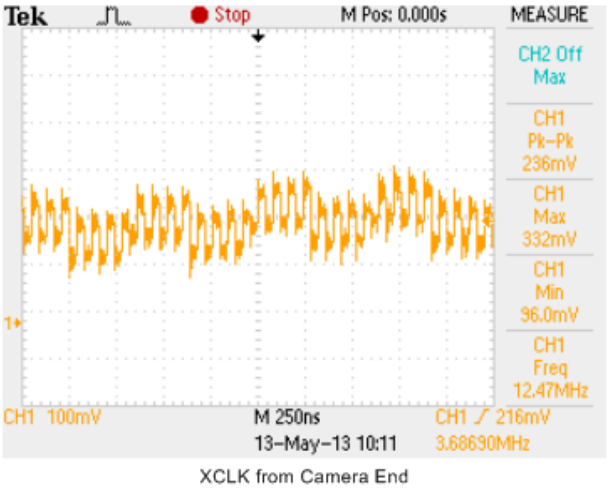
Results required direct implementation of test code onto a DSP using Texas Instrument's Code Composer Studio V4 and Texas Instrument's Chip Support Library. A TDS 2024B oscilloscope was used to measure the signal outputs while a HP E3630A triple output power supply and an Agilent 33220A waveform generator was used for isolation testing to check specific functionalities of the camera.

1) Operation Bus/Image Bus: Initially, a general purpose I/O (GPIO) port was to be used to provide the clock signal. Because the DSP was using all eight GPIO ports for the parallel data line to send image data, the inter-integrated sound (I^2S) peripheral was used instead to provide the external clock. I^2S has a sourceable clock line where the 100 MHz system clock from the DSP is processed through a clock divider. Using the clock divider, the I^2S clock source can output a 12.5 MHz clock, which is used with the camera. Fig. 20(a) shows the result of the I^2S clock signal after it has been run through the clock divider, which was transmitted through an I/O port from the DSP. Fig. 20(b) shows the signal provided to the camera's external clock port, which has the correct frequency of 12.5 MHz but due to transmission line effects, the signal is degraded. The I^2S maximum amplitude is expected to be around 4 V. However, at the camera end of line, the signal attenuates to 178 mV as shown in Fig. 20(b), which is below

activate high state according to [6]. It is hypothesized that the cause of the attenuation is due to non-matching impedance.



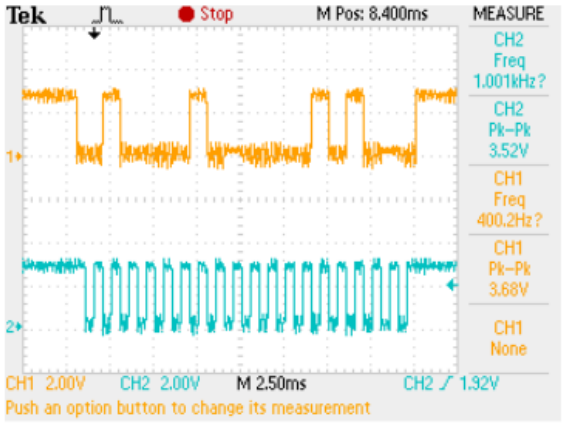
(a)



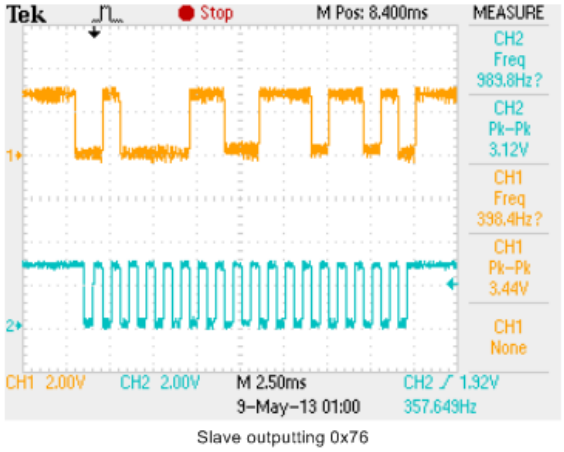
(b)

Fig. 20: Measuring the external clock (a) is measured at the DSP's I^2S clock and (b) is measured at the camera's external clock port

A solution to this problem is to create a PCB. The PCB functions as an interconnection interface with the DSP, camera, and the MCU allowing any signal including the parallel data lines to minimize transmission line effects. By creating a PCB, the impedance can be matched on the lines and reduce transmission line effects on the data lines to increase cleaner digital signals. Due to time constraints, a PCB could not be designed. In lieu of a PCB, the DSP was isolated from the camera. Since the DSP's external clock was being attenuated, a wave generator providing a 12.5 MHz signal was used. Also an external power supply providing 2.4V was sourced for the safety of the camera where the DSP voltage source is 3.3V, which is 0.3V above the tolerated input voltage [6]. Both a clock and voltage source was needed for testing of the configuration bus.



(a)



(b)

Fig. 21: SCCB (a) Point and (b) Read Transmission

2) Configuration Bus: Using test code, the DSP repeatedly used SCCB protocol to read from the product ID control register (0x0A) of the camera, which holds the value 0x76. The TDS 2024B was used to validate the transmission between the devices. As shown in Fig. 21, Fig. 21(a) represents the point transmission while Fig. 21(b) represent the read transmission. Channel 1 is the data line, while channel 2 is the clock line. Inspecting the signals closely, the master transmitted a pointer value to control register 0x0A and a camera output value 0x76 was registered on the data line.

B. Image Processing

The image processing results were acquired using both simulation in MATLAB and the implementation on the DSP. The stop sign detection results were gathered from simulations and show that the algorithm can detect the stop sign. The results of the lane line detection show that the lane line can be detected by the algorithm.

1) Stop Sign Detection Results: The HOG algorithm is implemented as a simulation in MATLAB. The gradient magnitude and edge detection correctly describe the test image.

(a)

(b)

Fig. 22: (a) original image and (b) feature data

The result of the gradient magnitude calculations is shown in Fig. 22. In Fig. 22(b), larger gradient magnitudes result in brighter pixels. The simulation program then aggregates the gradients of the image into the relevant feature data vector to feed into the SVM. The simulation of the SVM was built around a teaching set of 30 images, of which 15 are stop signs and 15 are not. The test set of contains 15 images. The image size was 250x250 pixels. The stop sign detection does not analyze the entire image that is captured by the camera; only the top right corner is used.. The test set for the simulation results is shown in Fig. 23. Each image in the test set is a subsection of a larger image.

Fig. 23: Support Vector Machine Test Set

The SVM showed a success rate of 80% when simulated with the set of 15 test images in Fig. 23. The only failures were three false negatives. The SVM categorized those three images that contained a stop sign as not being a positive classification. Some potential causes of failure in those three images are heavy shadows across the stop sign, extra sign edges in the image, and extreme angles. Those characteristics would be expected in a real world environment, but not in the scaled urban environment. The SVM would be able to adapt to those variations in stop sign images by using a broader teaching set, which is unnecessary for the environment and objectives.

2) Lane Line Detection Results: The CHEVP algorithm is implemented on the DSP in C, and was tested using the subsections of a larger image to test the left and right lane. The algorithm is designed to work on images that are 160x20 pixels.

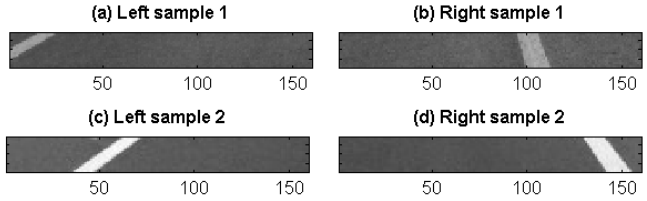


Fig. 24: Sample lane line images

The output from the DSP is illustrated in Fig. 25 in each of the corresponding steps in the Canny edge detection algorithm. The input image which is not shown was taken from the left lane side of a test image. The program also performs the transform on the right side of the lane. The Hough transform is then performed upon the output of the Canny edge detection, which is shown in the thresholding image of Fig. 25(a). The DSP outputs the contents of the accumulator.

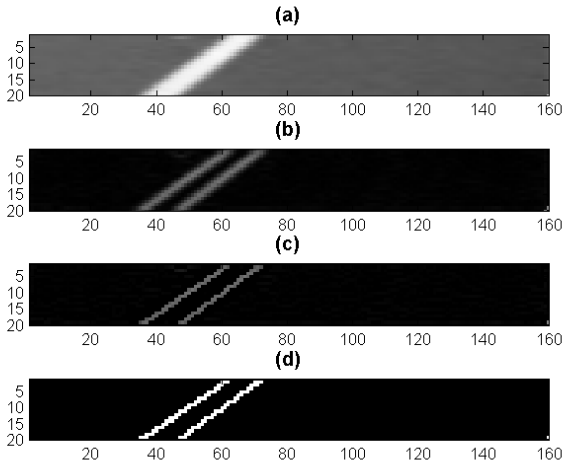


Fig. 25: Results of Canny edge on DSP

The results of the Canny edge detection were determined from the lane image shown in Fig. 24(a). The results of the Canny edge detection are shown in Fig. 25(a-b). Each

section corresponds to a step in the canny edge detection algorithm. The results of Gaussian smoothing are shown in Fig. 25(a). The smoothing reduces the noise in the image, which could potentially create extraneous edges that would affect the outcome of the Hough transform. The gradient magnitudes in Fig. 25(b) designate the two lines that represent the edges of the lane line. The input image, shown in Fig. 24(c), was taken from the left side of a full test image. The canny edge detection is performed for the left and right lane lines. The Hough transform is then performed on the outputs of the canny edge detection, which is shown in the thresholding results shown in Fig. 25(d). Fig. 26 shows the DSP output of the contents of the accumulator calculated from the edge data found in Fig. 25.

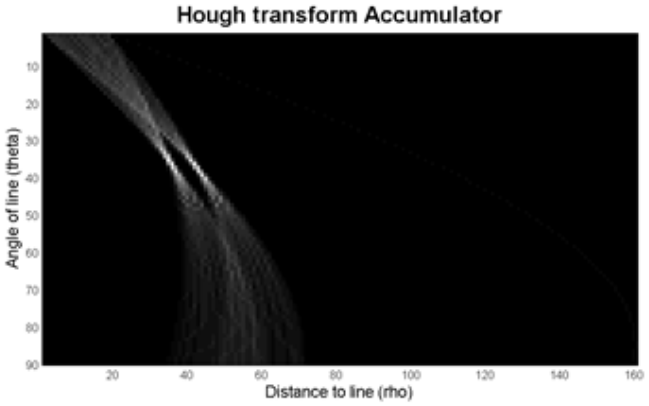


Fig. 26: DSP Accumulator Results

The contents of the accumulator gathered from the binary output in Fig. 25 is displayed in Fig. 26. There are two main focal points in the accumulator. They represent the two different edges of the lane line. The largest magnitude in the accumulator corresponds to the correct ρ and θ values of a lane line.

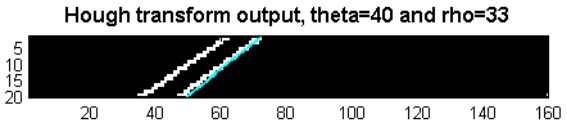


Fig. 27: DSP Accumulator Results with Calculated Line

C. Inter-Integrated Circuit Communication

The DSP has been configured to output the first transmission of data to the MCU as shown in Fig. 28. Channel 1 corresponds to the SDA line and channel 2 corresponds to the SCL line. Because the MCU's I^{2}C program is not functional, the DSP forced the acknowledge bits high so the master protocol could be tested. Testing was done by setting the MCU slave address to a value of 0x36 and setting the data to be transmitted as 0x01. Fig. 28 shows the representation of the transmission. The DSP first transmits the value of the MCU slave address (0x36) followed by a forced acknowledge bit, the data (0x01), and another forced acknowledge bit.

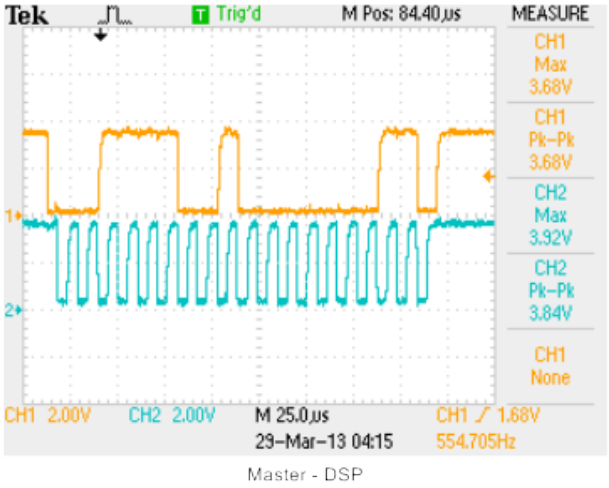


Fig. 28: DSP I^{2}C Communication

The data for the MCU is configured correctly such that the SDA line is at a high voltage level in the idle state. However, the SCL line remains in the low state after configuration. Due to this discrepancy, the MCU and DSPs were never able to be connected along the I^{2}C bus to validate communication.

D. Vehicle Control

Vehicle control results can be described by the performance of software algorithm design and by the functionality of peripherals in the vehicle. Software algorithms were tested by observing vehicle behavior using both bench testing and ground testing. Refer to appendix A for the testing data.

1) Software Algorithm Results: Testing results indicate that the speed regulation algorithm achieves the desired steady-state vehicle velocity to a mean value of 0.658 % error. Data was taken by performing bench testing of the unloaded vehicle for a command of 0.6386 m/s and comparing actual steady-state vehicle velocity with the command signal. The vehicle velocity was measured by reading TDS 2024B oscilloscope data from the Omron EE-SG8 speed sensor. Because testing conditions were performed for an unloaded model, testing reaffirms that the proportional-integral control structure has low steady-state error regardless of initial error magnitude.

The tracking algorithm produces vehicle orientation precise to within 2% error. The error was found by testing a 90° corrective corrective angle command after allowing for a calibration period for vehicle speed regulation. The testing result suggests that for smaller corrective angle commands, which are realistic for the intended operating conditions, lane tracking precision is well within 1°.

Deceleration control stops the vehicle at a specified distance within 5% error. The error measurement was obtained by performing ground testing with a command signal stopping distance of 3 m and measuring the distance from the point of initial deceleration to the point of zero velocity. Note that testing used a single distance command signal throughout a given deceleration control trial. In the fully integrated system,

the DSP will communicate updates of the distance variable throughout deceleration control to improve the system's precision.

The turning algorithm performs an accurate 90° right turn within 5° of error. The algorithm uses the measured turning distance of 1.25 m as the command distance of travel, and consistently undershoots the 90° turn by approximately 5°. Since this is an error of 5.9%, the distance command of 1.25 m can be multiplied by a factor of 1.059 to obtain a command of 1.32 m. This hypothetical software change is expected to yield a large amount of error reduction.

2) Peripheral Interfacing Results: The speed sensor precision and accuracy was measured by comparing the speed sensor variable in the Code Composer Studio software debugger to TDS 2024B oscilloscope measurements. These measurements were interpreted by project team members to be precise to three significant digits based on observation of TDS 2024B oscilloscope measurement variability. Results from these measurements produced data indicating that speed sensor interpretation was accurate to a mean value of 0.382% error. The error range was expected and is very comparable to calculations anticipated in the methods section.

During vehicle operational testing, unexpected system shutdowns were occurring. The cause of these shutdowns was theorized to be voltage spikes on the safety shutoff signal, generated on a ground node that was common between the vehicle's DC motor and controller electronics. The shutdowns were alleviated by adding a 1 μF capacitor in parallel with a 750 Ω resistor in order to resist sudden voltage changes and provide a channel for current to flow to ground. Because the real-time voltage spikes were theorized but not observed, an optimized analog design was not attainable. Instead, a tuning method was used until the design caused the cessation of the shutdown problem.

V. Discussion & Conclusion

The results of the image processing subsystem and the completion of the vehicle control subsystem verifies the value of small scale analysis of autonomous vehicles. In the image processing subsystem, successful implementation of the CHEVP algorithm on the DSP and positive simulation results for the HOG algorithm provide a basis for full implementation of image processing software. All vehicle control algorithms were instantiated on the MCU and were integrated with electronics and vehicle peripherals to achieve motion and indicator control. All high-level system objectives could be completed with further development. With the addition of a PCB, communication between the DSP and the camera could be achieved. Doing so would allow environmental image data to be processed in real time. Serial communication between the subsystems can be achieved by rectifying the I^2C initialization error. With continued research, small scale autonomous vehicles will allow for more schools to explore autonomous vehicle technology by eliminating extraordinary expenses.

APPENDIX A
VEHICLE CONTROL TEST DATA

TABLE I: Speed Controls Test

* operation speed of 0.5 m/s is 15.625 MPH scaled by 1/14

Trial	Command $\left[\frac{m}{s}\right]$	PI Structure	Zero Point	Steady State Velocity (Unloaded) $\left[\frac{m}{s}\right]$	Steady State Velocity (Loaded) $\left[\frac{m}{s}\right]$	Average Steady State Velocity (Loaded) $\left[\frac{m}{s}\right]$	Sensor Reading $\left[\frac{m}{s}\right]$	Sensor Error (Unloaded) [%]	Speed Control Error (Loaded) [%]
1	0.5	No PI; k = 0.1	592150	0.738	0.638	0.782	5.976	47.685	27.776
2	0.5	No PI; k = 0.1	592150	0.738	0.638	0.773	4.724	47.685	27.776
3	0.5	No PI; k = 0.1	592150	0.727	0.638	0.779	7.148	45.489	27.776
						Avg	5.949	46.953	27.776
1	0.5	No PI; k = 1.0	591000	0.542	0.469	0.566	4.320	8.523	6.034
2	0.5	No PI; k=1.0	591000	0.530	0.467	0.554	4.527	6.166	6.455
3	0.5	No PI; k=1.0	591000	0.530	0.467	0.551	3.940	6.166	6.455
						Avg	4.262	6.951	6.315
1	0.5	No PI; k=0.707	591000	0.533	0.471	0.558	4.541	6.755	5.610
2	0.5	No PI; k=0.707	591000	0.545	0.463	0.567	4.0379	9.132	7.285
3	0.5	No PI; k=0.707	591000	0.533	0.459	0.559	4.839	6.755	8.100
						Avg	0.333	8.335	3.657

* operation speed of 0.5 m/s is 15.625 MPH scaled by 1/14

1	0.638	Err = (Err + newErr)/2	591000	0.687	0.612	0.689	0.291	7.579	4.061
2	0.638	Err = (Err + newErr)/2	591000	0.673	0.609	0.674	0.295	5.386	4.620
3	0.638	Err = (Err + newErr)/2	591000	0.692	0.616	0.696	0.654	8.362	3.495
						Avg	0.413	7.109	4.059
1	0.638	No PI; k=1.0	591000	0.687	0.616	0.689	0.320	7.579	3.495
2	0.638	No PI; k=1.0	591000	0.701	0.602	0.704	0.359	9.84	5.719
3	0.638	No PI; k=1.0	591000	0.687	0.627	0.689	0.320	7.579	1.757
						Avg	0.333	8.335	3.657
1	0.638	var = var+1000*Err	591100	0.642		0.641	0.015	0.532	
2	0.638	var = var+1000*Err	591100	0.638		0.639	0.283	0.093	
3	0.638	var = var+1000*Err	591100	0.630		0.635	0.899	1.346	
						Avg	0.399	0.657	
						Cumulative Average	0.382		

TABLE II: Stopping

Trial	Velocity $\left[\frac{m}{s}\right]$	Distance command [m]	Distance Traveled [m]	Error [m]	Error [%]
1	0.64	3	2.834	0.165	5.512
2	0.64	3	2.926	0.073	2.464
3	0.64	3	2.804	0.195	6.528
			Avg	0.145	4.834

TABLE III: Turning

*90° Turn Test (25° steering angle)

Trial	Error [°]
1	5
2	6
3	2
4	5
Avg	4.5

* Operating speed at 0.4 m/s

TABLE IV: Tracking: 6° Turn Test

Trial	Radius [in]	Radius [m]
1	124	3.149
2	123	3.124
3	124	3.149
Avg		3.141

TABLE V: Tracking: 9° Turn Test

Trial	y Radius [in]	y Radius [m]	x Radius [in]	x Radius [m]
1	108.312	2.751	113	2.870
2	113.312	2.878	111	2.819
3	113.312	2.878	110	2.79
Avg		2.835		2.827

TABLE VI: Tracking: 12° Turn Test

Trial	y Radius [in]	y Radius [m]	x Radius [in]	x Radius [m]
1	79.312	2.014	77	1.955
2	79.312	2.014	79	2.006
3	79.312	2.014	78	1.981
Avg		2.014		1.981

TABLE VII: Tracking: 15° Turn Test

Trial	y Radius [in]	y Radius [m]	x Radius [in]	x Radius [m]
1	61.3125	1.557	63	1.600
2	63.3125	1.608	63	1.600
3	61.3125	1.557	63	1.600
Avg		1.574		1.600

TABLE VIII: Tracking: 18° Turn Test

Trial	Radius [in]	Radius [m]
1	45.5	1.155
2	44.5	1.130
3	44.5	1.130
Avg		1.138

TABLE IX: Tracking: 21° Turn Test

Trial	Radius [in]	Radius [m]
1	38	0.965
2	37.25	0.946
3	37.5	0.9525
Avg		0.954

TABLE X: Tracking: 24° Turn Test

Trial	Radius [in]	Radius [m]
1	32.5	0.825
2	33.5	0.850
3	33	0.838
Avg		0.838

TABLE XI: Tracking: 25° Turn Test

Trial	Radius [in]	Radius [m]
1	31.5	0.800
2	31.5	0.800
3	31	0.7874
Avg		0.795

TABLE XII: Turn Radius Extrapolation

θ [°]	x = R*sin(θ)
6	0.328
18	0.352
21	0.342
24	0.341
25	0.336
Average	0.339
Std Dev [%]	2.585

TABLE XIII: Turn Radius Extrapolation

x = R*sin(θ)	Command [°]	Error [°]	Error [%]
0.34	90	14	15.555
0.391	90	1	1.111

ACKNOWLEDGMENT

The authors would like to thank Dr. Jose Sanchez for his guidance and help throughout the duration of the project, Dr. James Irwin for his assistance in assembling the Tamiya RC MAN TGX 26.540 6x4 XLX, and Dr. Yufeng Lu for advice on how to interface the Omnivision OV7670 with the DSP.

REFERENCES

[1] *Urban Grand Challence - Resources* [Online] Available: http://archive.darpa.mil/grandchallenge/resources.asp.

[2] O. Thomas. (2010, September 7). *Google's Self-Driving Cars May Cost More Than A Ferrari* [Online]. Available: http://www.businessinsider.com/google-self-driving-car-sensor-cost-2012-9.

[3] *Motor Vehicle AccidentsNumber and Deaths: 1990 to 2009*, U.S. Census Bureau, Chicago, IL, 2010.

[4] C. Mui. (2013, January 22). *Fasten Your Seatbelts: Google's Driverless Car Is Worth Trillions (Part 1)* [Online]. Available: http://www.forbes.com/sites/chunkamui/2013/01/22/fasten-your-seatbelts-googles-driverless-car-is-worth-trillions/

[5] J. White. (2011, December). *2012 Illinois Rules of the Road* [Online]. Availavle: http://www.cyberdriveillinois.com/publications/pdf_publications/dsd_a112.pdf

[6] OmniVision. (2006). *OV7670/OV7171 CMOS VGA (640x480) $CameraChip^{TM}$ Sensor with OmniPixel Technology* [Online]. Available: http://www.eleparts.co.kr/data/design/product_file/Board/OV7670_CMOS.pdf

[7] C.H. Chen et. al. *Detection and Recognition of Alert Traffic Signs*, Stanford University, Stanford, CA. 2003.

[8] C. Cortes. *Support Vector Networks* [Online]. Available: http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf

[9] J. C. Platt. (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines* [Online]. Available: https://research.microsoft.com/pubs/69644/tr-98-14.pdf

[10] J. Kroll. (2010, September 24). *How to Calculate a Turning Circle* [Online]. Available: http://www.ehow.com/how_7225784_calculate-turning-circle.html.

[11] *The Kinematic Equations* [Online]. Available: http://www.physicsclassroom.com/Class/ n1DKin/U1L6a.cfm

[12] Omron. *Photomicrosensor (Transmissive) EE-SG3 / EE-SG3-B* [Online]. Available: http://www.omron.com/ecb/products/pdf/en-ee_sg3_sg3_b.pdf

Devon Bates is a senior Electrical Engineering student at Bradley University. She will be attending graduate studies at UC Santa Cruz after graduation.

Frayne Go is currently a senior attending Bradley University, pursuing a Bachelor of Science in Electrical Engineering with Computer Option. He received his Associate of Science in Engineering at College of Lake County (Grayslake, Il) in August 2010. Frayne is pursuing a career in embedded system design specializing in image processing.

Tom Joyce is a senior Electrical Engineering student at Bradley University. He will be working for UTC Aerospace Systems in Rockford, IL as a circuit design engineer after graduation.

Elyse C. Vernon is currently undergraduate student at Bradley University where she is working towads her Bachelor's degree in Electrical Engineering with an expected graduation date of May 2013. Upon graduation she will be working for Ameren IL in Peoria IL.