Mechanical Engineering 131
Final Project

Alexandre Chong
Travis Buckman
Thomas Dwelley

Special Thanks to:
Jon Gonzales
Pegasus Books
Wenshuo Wong
Prof. Borrelli
Student Machine Shop
Supernode

Spring 2016

1 Abstract

Driver assist systems are becoming standard on more and more consumer vehicles. They have the potential to reduce accidents and save lives by reducing driver fatigue or taking drivers out of the loop during long stretches of highway driving. Using the BARC-project platform, we created a fully autonomous contiguous-lane lane keeping, cruise control vehicle. It utilizes PID for speed control and PD for steering control. For lane detection a computer vision algorithm generates a vertical view, which is then convolved with a lane-detection kernel, thresholded, and the difference between the center of the image and the center of the lane is then used for steering control. This system achieved consistently 1.5 [m/s] lane keeping on multi-turn track with a contiguous lane marker.

2 Introduction

For this study we decided to tackle automated lane keeping with cruise control. A lane keeping system is defined by Wikipedia as "Systems which warn the driver and, if no action is taken, automatically take steps to ensure the vehicle stays in its lane"[1]. Recently, car manufacturers such as Tesla, Infiniti, Mercedes, and Volkswagen have begun selling cars that allow "unassisted driving under limited conditions"[1].

Evidently, lane keeping systems will soon be standard on many consumer automobiles. They have the potential to reduce traffic accidents by removing the driver from the loop or by alerting the driver to potential hazards.

Although automated lane keeping appears to be a solved problem, working with real systems is never trivial and always presents a unique set of challenges. There are still improvements to be made in computer vision, controller robustness, and human robustness.

In lecture we studied a lane keeping controller proposed by Rajamani which consisted of a controller (either proportional or lead compensator) that uses an error based off the sum of the distance between the center lane and the vehicle's center of gravity and the difference between the yaw rate of the road and the yaw rate of the vehicle multiplied the longitudinal distance away from the c.g. of the vehicle where the measurement is made[2]. However, we ended up using a PD controller based purely off lane offset.

Our solution uses a robust, real-time computer vision algorithm for contiguous-lane lane detection. Combined with a PID speed controller and a PD steering controller, it achieved 1.5 [m/s] constant speed on a multi-turn track with tighter turns than one would find on any major highway.

In this report we will explore the theory and implementation of this system, look at the procedure for tuning our controllers, and examine the results and performance of this system.

3 Theory

3.1 System Architecture: Overview

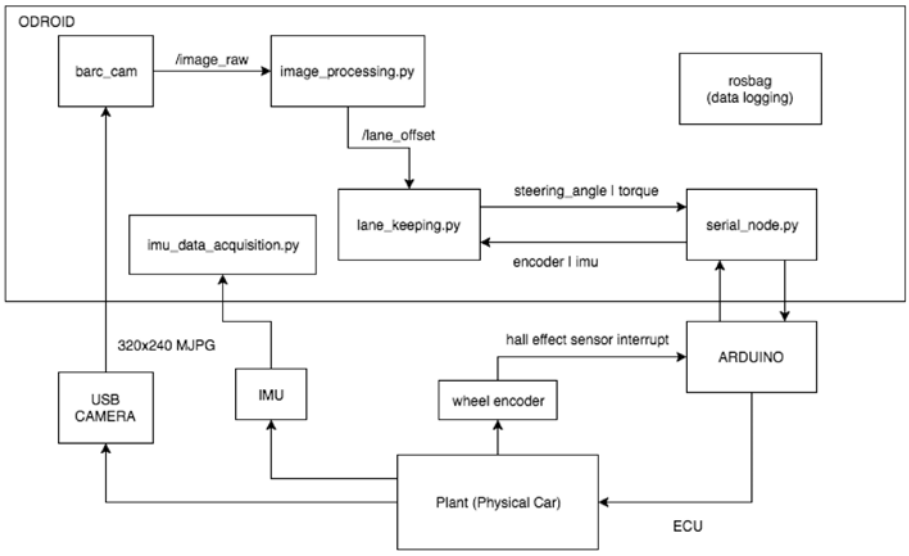


Figure 1: System Architecture Diagram

To perform control, computer vision, and record data all at the same time, we used ROS (robotic operating system) to handle inter-process communication and I/O. Our language of choice was Python. Python allowed us to easily debug our code, avoid complicated build systems, use a variety of pre-compiled modules, and granted us fast iteration time.

3.2 Computer Vision: Introduction

To achieve stable and robust real time lane detection of contiguous lane markers, we implemented a portion of Aly's "Lane Detection Algorithm for Lane markers in Urban Environments"[3].

3.3 Computer Vision: Implementation

In summary, the algorithm has the following pipeline (Figure 2).

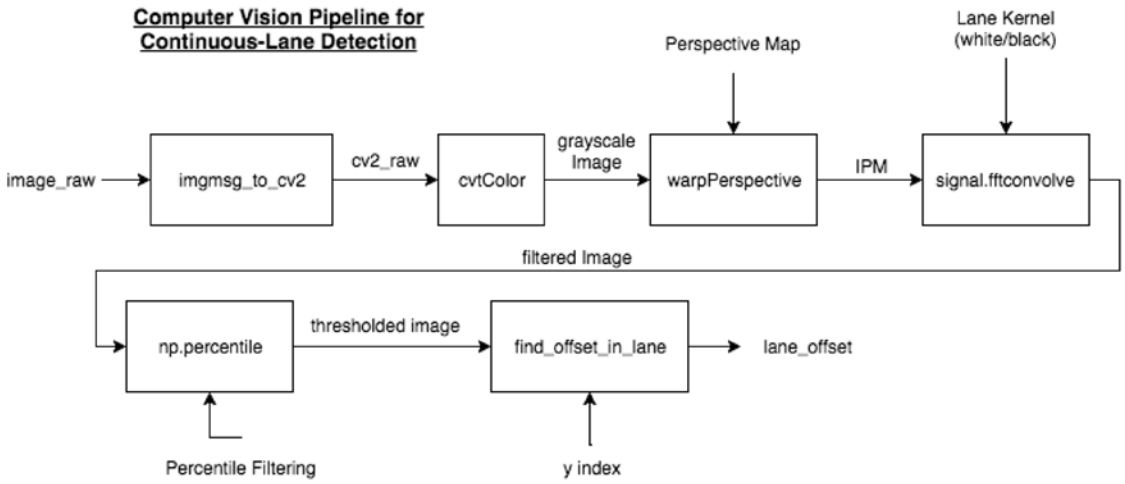


Figure 2: Lane Detection Image Pipeline

Note: To maximize the use of the camera sensor's area we changed the orientation of the camera to portrait and pointed the camera towards below the horizon.

The camera node is initialized to publish images at a rate of 30Hz at a resolution of 320x240. Upon receiving an image from the camera the image processing node converts the image to the gray scale color space (Figure 3).

Figure 3: Raw Grayscale Image 320x480

Next an inverse perspective map to a vertical view is performed on the image (Figure 4). This is a linear transformation that must be calibrated using the camera in it's final mounted location and two parallel lines placed in front of the car. These parallel lines are then mapped out to a parallel lines in a

vertical view. Note that the inverse perspective mapping fixes the landscape orientation and increases in noise vertically as pixels stretch out. The noise is approximately proportional to the pixel height squared.

Figure 4: Raw after Inverse Perspective Mapping

After performing the inverse perspective map, the image is then filtered by performing a 2-D convolution with a lane detection kernel. This is also a linear operation. Effectively, it amplifies the regions of the image that look similar to the lane we are trying to detect and attenuates the regions of the image that don't resemble a lane. Figure 5 is a grayscale rendering of the kernel that we use for detecting white lanes on a dark background.

When this kernel is convolved with an image it leaves higher values in places with a white band in the center and dark bands on the sides. This kernel is defined by the function from Aly[3]. It is the gaussian function in the y-direction and the second derivative of the gaussian function in the x-direction.

$$f_v(y) = exp(-\frac{1}{2\sigma_y^2}y^2) \tag{1}$$

$$f_u(x) = \frac{1}{\sigma_x^2}exp(-\frac{x^2}{2\sigma_x^2})(1 - \frac{x^2}{\sigma_x^2}) \tag{2}$$

For best results for the lane markers and camera mount/position we used, we selected a 5 pixels tall by 11 pixels wide kernel where $\sigma_x^2 = 5$ and $\sigma_y^2 = 1$

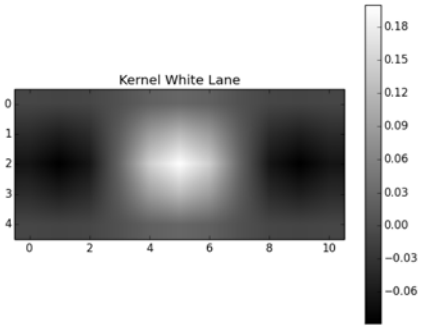


Figure 5: White lane image kernel

We also noticed that we could generate a kernel that would detect dark lanes on a light background simply by flipping the sign of the function of the kernel in the x direction (Figure 6).

$$f_u(x) = -\frac{1}{\sigma_x^2} exp(-\frac{x^2}{2\sigma_x^2})(1 - \frac{x^2}{\sigma_x^2}) \tag{3}$$

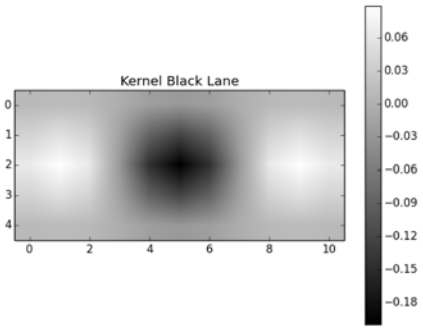


Figure 6: Black lane image kernel

Subsequently, the image goes through a percentile based threshold (Figure 7). This threshold only keeps the highest p% values in the image. Hence, this is a nonlinear operation. Intuitively, this means that this thresholding only keeps the regions of the image that match in the lane detection the kernel the best. We found that selecting the highest 5% of pixels worked well. It removed noise and still kept the lanes distinct. We noted that when lane markers are faded/poorly defined a higher value of p will allow the markers to still be identified. The trade off is that noise is more likely to get through.

Figure 7: Filtering and Thresholding

When ROS is launched we select the midpoint of the image 20 pixels from the bottom and move left and right until we hit white pixels (seen in yellow in Figure 8). The midpoint of these two pixels (in purple Figure 8) represents the center of the lane. The number of pixels this purple point is from the center of the image represents the lane offset. This is the value that is published by the image processing node and subscribed to by the controller node.

To allow us to keep tracking the lane even if it moves past the center of the image we store the previous center of the lane (purple dot). When the function is called back again we use this point as a starting point in our search for the first left and right white pixels along this row.

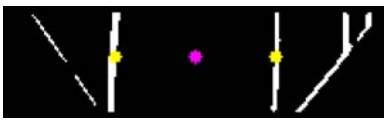


Figure 8: Final Result

3.3.1 Computer Vision: Implementation Notes

The capabilities of the camera and the odroid arm-based computer that are selected for the barc-project specification limits the resolution for sub-second time-delay computer vision based control to 320x240 images. The time delay for transferring image files from the camera to the odroid appears to be $O(N)$ where N is the number of pixels and hence $O(N^2)$ where N is the width of the image.

3.3.2 Computer Vision: What Doesn't Work

We discovered that the commonly used Canny Edge detection algorithm displays poor performance on non binary images and images with motion blur. Motion blur will be introduced whenever the camera is moving. Hence, Canny edge detection on a non-binary image is a poor idea for quickly moving cameras (e.g. BARC project)

We also tried polynomial fitting (5th, 3rd, 1st order) to the lane markers to get an estimate of the lane's yaw rate. In practice this operation yielded a very noisy signal which we found unsuitable for control.

3.4 Speed Estimation:

The barc-project uses two encoders on the front wheels to estimate speed. They work using edge based interrupt pins on the arduino connected to hall effect sensors triggered by four magnets configured in alternating polarity in the hub of the wheel. Although, there currently exists a state estimation node in the source code it requires parametrization of car by running tests and doing regression fits on the sensor data.

Instead of doing parametrization we decided that a simple digital low-pass filter could smooth out the speed signal and provide us with a more accurate and less noisy speed estimation.

- n_{FL} current encoder interrupt count front tire

- n_{FLprev} previous encoder interrupt count front tire
- dt time between previous and current velocity estimation
- r_{tire} radius of tire
- δ_f steering angle
- $v_{current,estimated}$ value of speed used for control

If $dt > 0.2$ then recalculate current wheel speed:

$$v_{FL} = \frac{n_{FL} - n_{FLprev}}{dt} \frac{2\pi r_{tire}}{4} \tag{4}$$

$$v_{current} = \frac{v_{FL} + v_{FR}}{2} cos(\delta_f) \tag{5}$$

$$v_{current,estimated} = 0.75 \cdot v_{current} + 0.25 \cdot v_{prev,estimated} \tag{6}$$

This is effectively a weighted average filter. It essentially functions as a digital low-pass filter.

4 Procedure

4.1 Vehicle Modifications

In order to implement our controller algorithms, we needed to complete two major modifications to our vehicle in support of the desired functionality. The first was to removed the front wheel drive. Though this could easily be achieved by cutting the belt running longitudinally across the car, we chose to disassemble the front drive train and remove the front axles and universal joints shown in Figure 9. This allowed the front wheels to move independently of the rears, allowing us to take absolute velocity readings of the vehicle for the cruise control, and reduced the number of moving parts for wiring to become tangled in.

Our second modification was to raise the position of the camera with a new camera mount. This served two main purposes: raising the position allowed us to collect a cleaner image of the lane, and a decreased incident angle to the surface aided in image processing by requiring a less dramatic initial transformation (as discussed in section 3.3). In addition, we were able to position the camera in a more protected position without compromising the viewing angle, which proved important during testing to avoid damage to the equipment (see Figure 10).

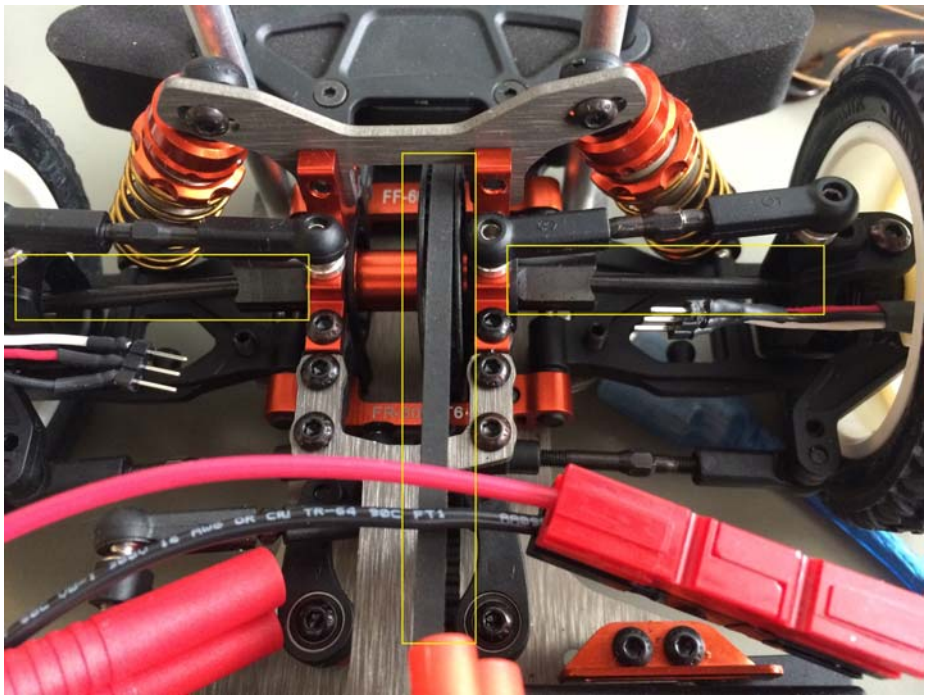


Figure 9: Front drive train, highlighting the removed portions in yellow.

(a) Original mount position (b) New mount position

Figure 10: Camera mount position, before (a) and after (b) the instalation of the new mount. Note that the camera is at risk in the case of a head on collision in the first case, but is protected in the second.

4.2 Longitudinal Control Tuning

Once we had image processing working, it quickly became clear that we needed a longitudinal controller to modulate the vehicle's speed during further testing. We chose to implement a basic PID cruise control algorithm so that we could focus our energy on the lane-keeping goal of the project. In lab, we tested the vehicle at speeds of up to 0.5m/s due to space constraints; we found that at higher speeds our speed estimation algorithms (see Section 3.4) was smoother and allowed for better controller performance. We then brought the vehicle to an outside facility for high speed testing of controller robustness.

4.2.1 Initial Tuning of the PID Controller

Our approach for tuning the cruise control was primarily experimental. This allowed us to gain a better understanding of the impact each term has on the behavior of the vehicle, as well as iron out our data collection and processing procedures. We carried this process out in lab, with a set desired velocity of 0.5m/s for every test. We started by increasing our K_p value for the steering controller until the vehicle became unstable; then, observing the data, selected a value with acceptable performance and settling time as our base K_p value. We then repeated the process for the K_i values and then the K_d values, before settling on a handful of controllers that seemed to optimize performance to the best of our ability without changing the focus of our project.

4.2.2 Testing Cruise Controller Robustness

Once we had settled on controller gains that seemed to yield the desired performance, we tested the car at a variety of desired speeds to ensure reasonable robustness for all our testing conditions. We also used this chance to finalize the K_d values for the speed controller, as there were a couple of diverse values that worked well in the lab environment.

4.3 Lateral Control Tuning

We chose to use an error-based PID method for lateral control. The image processing algorithm previously described yields a convenient estimate for the lateral position error between the car and the center of the lane. The algorithm returns the pixel offset, which is a measure of the lateral distance between the center of the lane and the vehicle's longitudinal axis at the headway point as illustrated in Figure 11. Measurements taken confirm a linear mapping between the pixel offset and physical offset, with 16 pixels $\approx$ 2.5 cm.

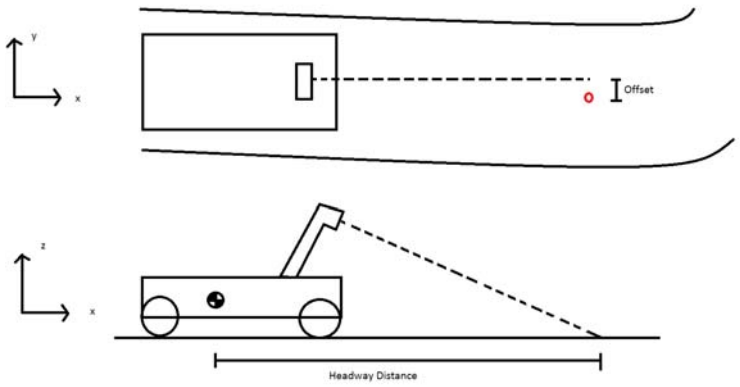


Figure 11: Schematic diagram illustrating geometric meaning of offset. The image processing algorithm returns the offset measured in pixels, which maps linearly into physical units of length

4.3.1 Straight Line Test; PID Control

Our approach to steering control was to employ heuristic PID tuning–such as the Ziegler-Nichols method—to calculate controller gain values within an order of magnitude on a straight track at a desired speed of 1 [m/s], then adjust those values as we increased both the speed and the complexity of the track. The first set of steering tests were run using only proportional (P) control. We tested the vehicle starting with $K_p = 0.05$ and increasing the gain on every run until the vehicle was on the verge of being unstable. At the ultimate K_p value, K_u, the vehicle followed a sinusoidal trajectory oscillating about the center of the lane, but was able to track the lane throughout the test. A representative plot of the pixel offset error and steering command vs. time for this type of testing is presented in Figure 12.

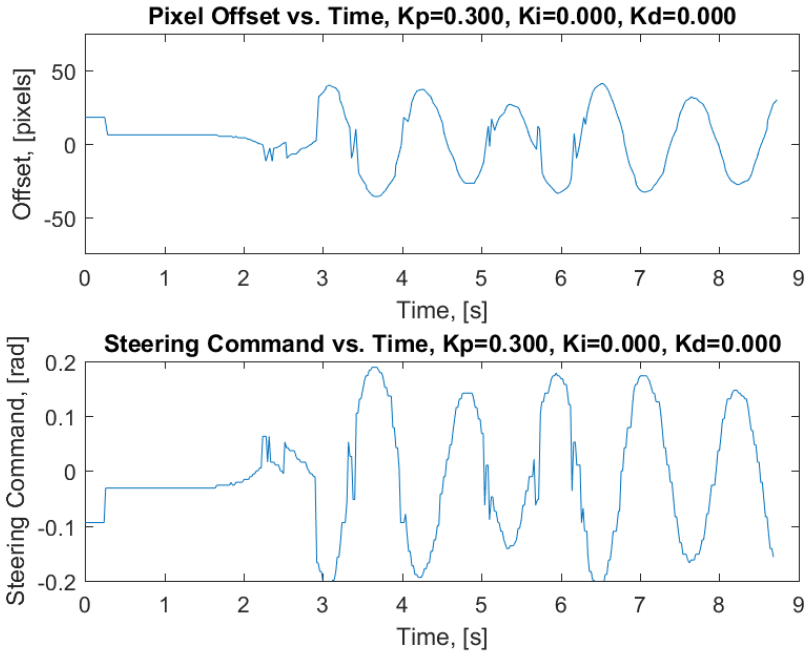


Figure 12: Plot of data used to calculate period of oscillation for $K_p = K_u$

From this type of plot we were able to obtain the period of the oscillation, T_u. K_u and T_u were then used to calculate the integral and derivative controller gains according to the algorithms presented in Table 1[4].

Table 1: Heuristic PID Tuning Schemes

Controller Type	K_p	K_i	K_d
Classic PID	$0.6K_u$	$T_u/2$	$T_u/8$
No Overshoot Method	$0.2K_u$	$T_u/2$	$T_u/3$

Unfortunately, the use of these methods led to the vehicle becoming highly unstable, halting further progress using this approach.
In the process of troubleshooting our tuning methods we hypothesized that we could reduce the oscillatory behavior of the vehicle exhibited under aggressive P control by using D control to damp the oscillations. We designed and ran a test on the vehicle using PD control, and the data agreed with our prediction. Based on this observation we continued experimentation with PD control.

4.3.2 Straight Line Test; PD Control

Using our previous P control testing results as a starting point, we experimented on the straight track again at 1 [m/s] with increasing Kd until we felt we got the

best performance. Using the optimal tuning parameters obtained from straight line testing, we moved on to a more complicated track.

4.3.3 Racetrack; PD Control

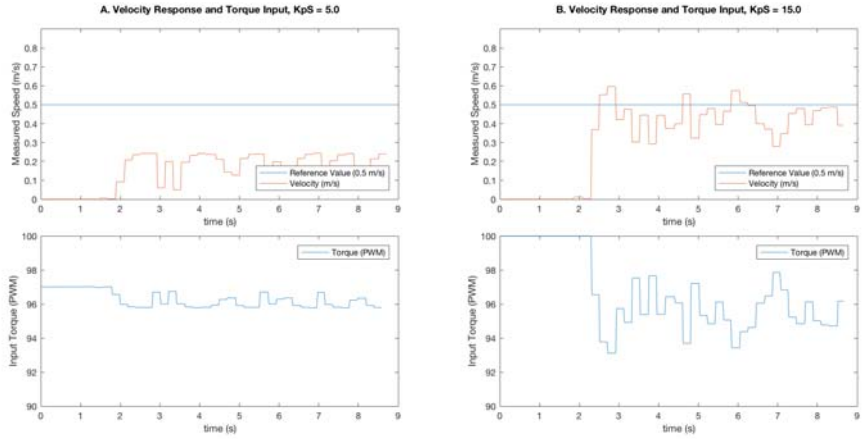
Our racetrack configuration featured common racing elements that a driver might encounter on a full-size course, including a straightaway, a 45deg turn, a constant radius curve and a slight chicane. Our metric for evaluating track performance was simple; we wanted the vehicle to keep track of the lane markings and minimize the offset error at increasing speeds while not crashing.
The controller as tuned on the straight track was unable to keep the racetrack, but minor adjustments led to an acceptable performance range for further tuning. After tuning the controller through trial and error the vehicle exhibited acceptable performance at speeds up to 1.5 [m/s].

5 Results

5.1 Cruise Control

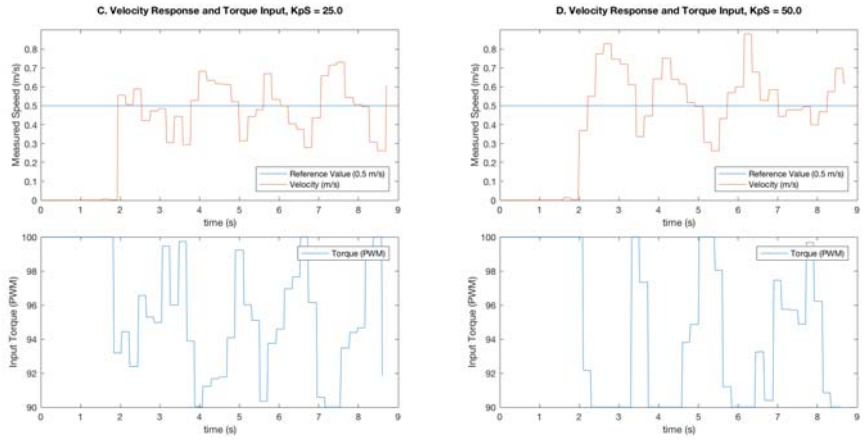
5.1.1 PI Controller Tuning: Proportional Term

Figure 13 demonstrates the impact of the K_p term on the longitudinal controller (here labeled KpS to differentiate it from the lateral controller). As the proportional gain increases, the response of the system increases but the settling time also increases. Increasing the proportional gain too much can also lead to instability: as seen in 13 when $K_{pS} = 50$. Using these results, we settled on a base K_{pS} value of 15.



(a) $K_{pS} = 5$

(b) $K_{pS} = 15$



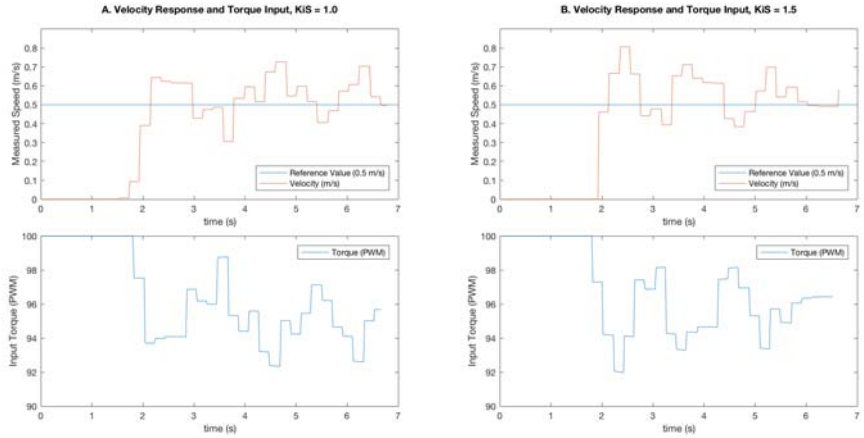
(c) $K_{pS} = 25$

(d) $K_{pS} = 50$

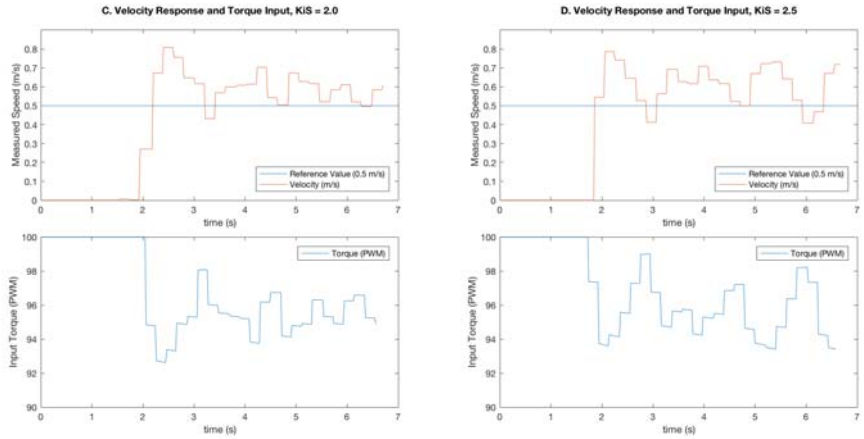
Figure 13: Vehicle Input (Torque, in PWM) and Response (Velocity, in m/s) for a set desired velocity of 0.5 m/s for a selection of proportional gain values. Integral term held constant at $K_{iS} = 0.01$ and Derivative term held constant at $K_{dS} = 0$.

5.1.2 PI Controller Tuning: Integral Term

Figure 14 demonstrates the impact of the integral controller on the system. Increasing the integral term was a trade off: it decreased the settle time of the system, but also increased the initial overshoot of the vehicle and can drive the system unstable. Here, we see that a K_{iS} value of 2.0 effectively reduces the settling time, while a K_{iS} value of 2.5 begins to cause instability in the system.



(a) $K_{iS} = 1.0$ (b) $K_{iS} = 1.5$

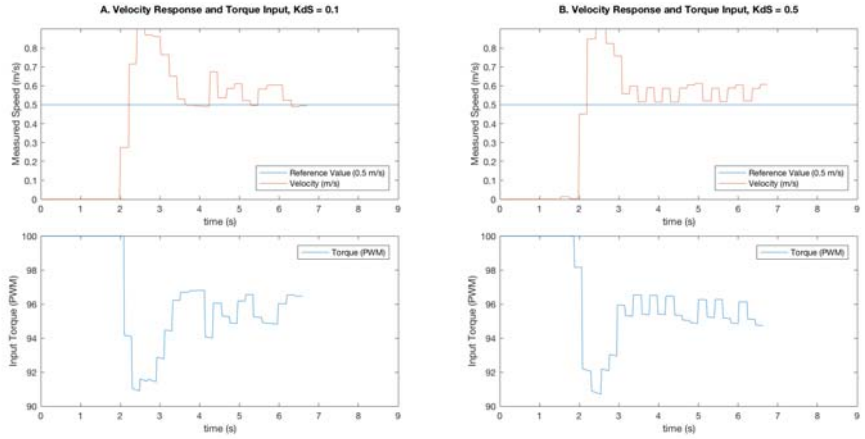


(c) $K_{iS} = 2.0$ (d) $K_{iS} = 2.5$

Figure 14: Vehicle Input (Torque, in PWM) and Response (Velocity, in [m/s]) for a set desired velocity of 0.5 [m/s] for a selection of integral gain values. Proportional term held constant at $K_{pS} = 15$, and Derivative term held constant at $K_{dS} = 0$.

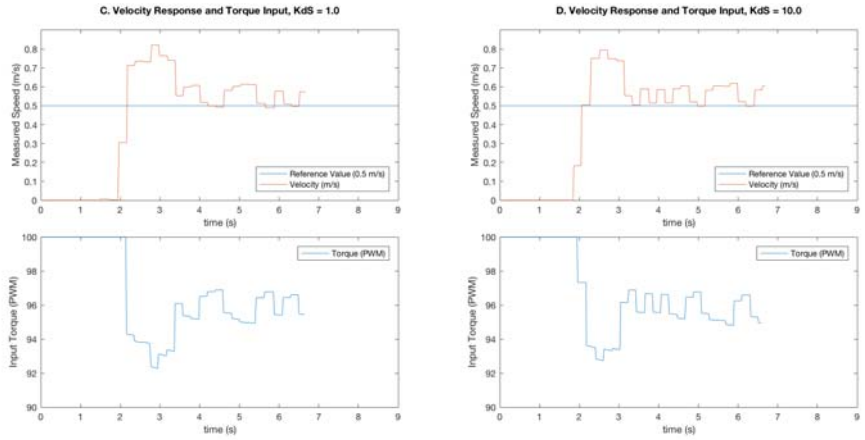
5.1.3 PID Controller Tuning: Derivative Term

In Figure 15, we see that the derivative term dampens the initial response of the system, decreasing overshoot but potentially decreasing long term stability. In this case, we see that a K_{dS} value of 0.5 is sufficient to dampen the response of the system after the initial overshoot; a K_{dS} value of 10.0 does a better job of dampening this initial response but also increases the settle time of the system.



(a) $K_{dS} = 0.1$

(b) $K_{dS} = 0.5$



(c) $K_{dS} = 1.0$

(d) $K_{dS} = 10.0$

Figure 15: Vehicle Input (Torque, in PWM) and Response (Velocity, in [m/s]) for a set desired velocity of 0.5 [m/s] for a selection of Derivative gain values. Proportional term held constant at $K_{pS} = 15$, and Integral term held constant at $K_{iS} = 2.0$.

5.1.4 PID Controller Tuning: Robustness

The final values obtained for the controller gains are presented in Table 2. We examined the robustness of this tuning by performing straight line tests with increasing speed. Figure 16 illustrates the ability of the controller to track the reference speed up to 1 [m/s]. The controller treats the reference speed as a minimum speed, which is suitable to our purposes.

Table 2: Final Cruise Control Tuning Parameters

Controller Gain	Value
K_{pS}	15.0
K_{iS}	2.00
K_{dS}	0.50

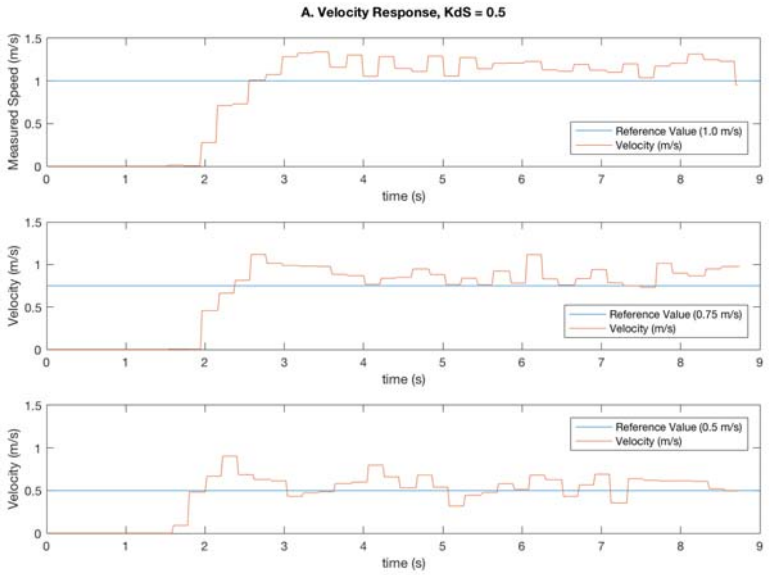
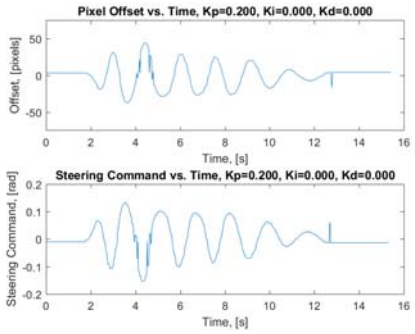


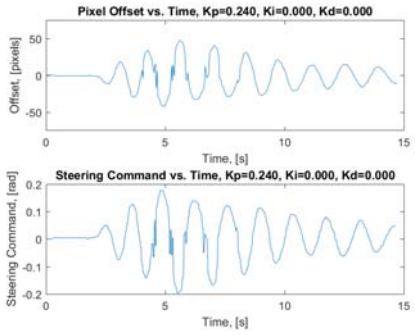
Figure 16: Vehicle Response (Velocity, in [m/s]) for a set $K_{pS} = 15.0$, $K_{iS} = 2.0$, and $K_{dS} = 0.50$. Desired velocity (top to bottom): 1 m/s, 0.75m/s, and 0.5 m/s. Note that the velocity is stable for all reference values.

5.2 Lateral Control

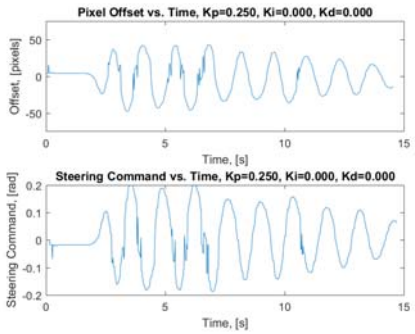
5.2.1 Straight Line Test



(a)



(b)



(c)

Figure 17: The plots above show that settling time increases as K_p increases from (a) 0.200; to (b) 0.240; to (c) 0.250

Figure 17 demonstrates the trend of settling time increasing with K_p. Figure 18 shows that when $K_p = 0.3$ the offset does not converge to zero, so this was taken as K_u in regards to heuristic tuning.

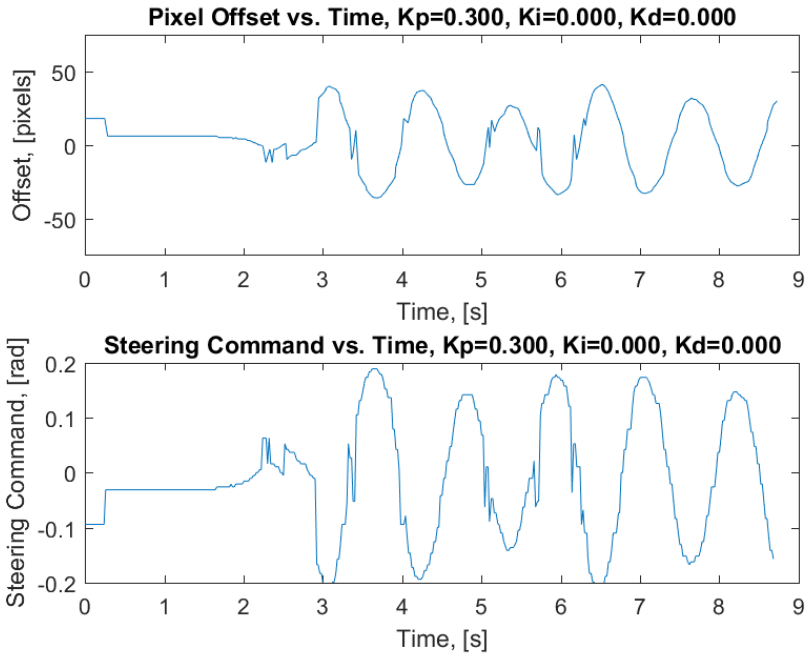


Figure 18: Plot of data used to calculate period of oscillation for $K_p = K_u$

As previously reported however, both sets of tuning parameters obtained from heuristic methods yielded wildly unstable performance, nearly crashing on every run after immediately careening off the track. Switching to PD control as outlined in the previous sections immediately led to better overall performance.

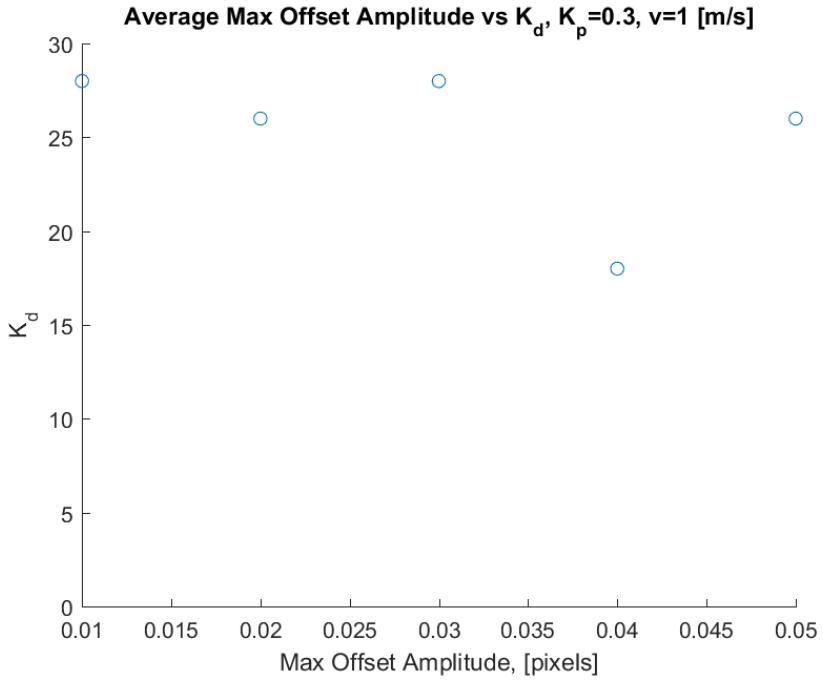


Figure 19: Plot of data used to calculate period of oscillation for $K_p = K_u$

Figure 19 shows the effect of Kd on the average maximum offset amplitude over 5 runs with K_p held constant at 0.3. $K_d = 0.04$ corresponds to the minimum average offset for the straight line test at 1 m/s. Plots of offset and steering command vs. time for $K_d = 0.04$ are presented in Figure 20.

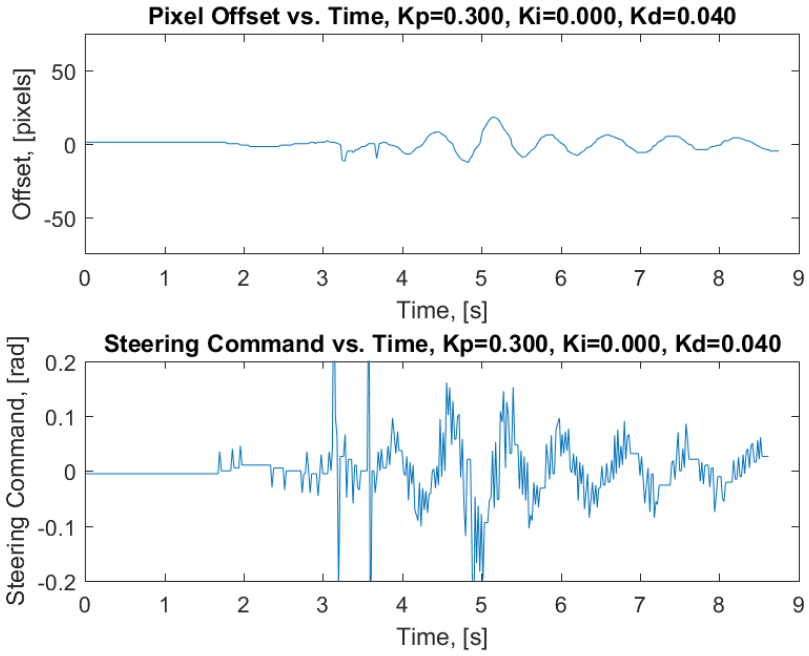


Figure 20: With $K_d = 0.04$ and $K_p = 0.3$, tests performed on the straight line track exhibited the smallest maximum offset error compared to other tested tuning parameters

5.2.2 Racetrack

The highest reference speed we were able to achieve around the racetrack while keeping the lane was 1.5 [m/s]. The final tuning parameters used to achieve lane tracking at this speed are presented in Table 3.

Table 3: Final Tuning Parameters; Lateral PD Control

Controller Gain	Value
K_p	0.14
K_d	0.05

Plots of offset and steering command vs. time using these parameters at a reference speed of 1.5 [m/s] around our racetrack are presented in Figure 21. The largest absolute offset error was about 50 pixels, which is equivalent to approximately 8 [cm] between the center of the track and the longitudinal axis of the vehicle at the headway point (see Figure 11).

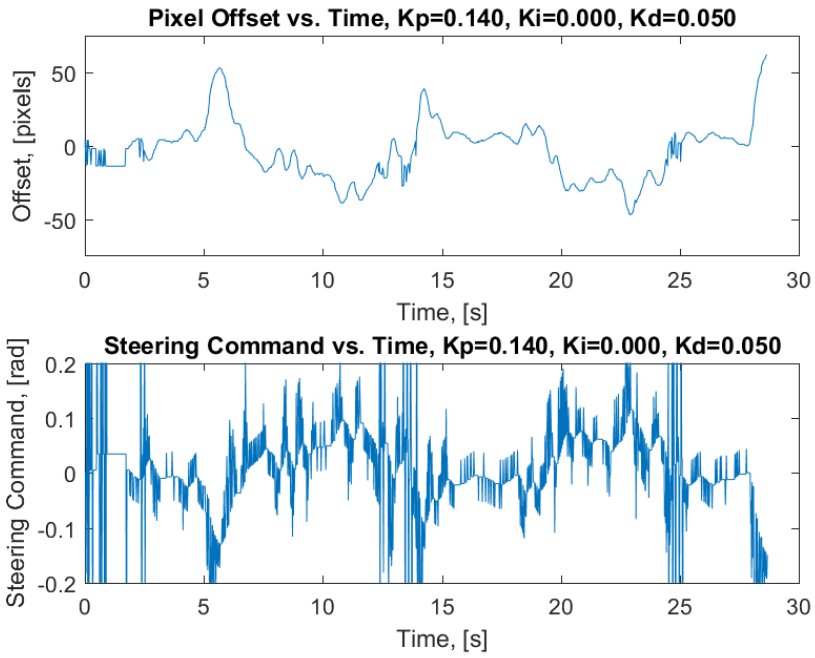


Figure 21: After tuning to optimal PD controller gains, test performed on the racetrack at 1.5 [m/s] yielded the above plot of offset error and steering command vs. time

video of the final run: https://youtu.be/1BEeBSESXIY

6 Conclusion

We developed an autonomous contiguous-lane lane keeping vehicle that navigated a multi-turn track using a PD steering controller, PID cruise control, and a robust computer vision system. We learned that PD control produces strong results for steering, but a PID controller is needed for an effective cruise control. Linear operations also produce strong results for computer vision. We were exposed to various controls, programming, computer vision, manufacturing, and assembly problems for which we found and researched solutions. This project was immensely rewarding, and we are grateful to have had the opportunity.

Topics for Further Exploration:

- Improving the vision system to allow it to track non-contiguous lane markers.
- Improve steering control to stay in the center of the lane during long turns

- Reduce the jerkiness of the steering input.
- Vary speed based on lane curvature. Essentially, we could modify the controller so it goes full throttle for straight-aways and reduces speed for turns.
- Add adaptive cruise control using the ultra-sound sensors.

We believe that most future work in this domain revolves around localization and object recognition. In a controlled, steady-state environment, well-within the limits of traction, automobiles are very controllable systems. The true challenge is giving the controller accurate, precise data about its environment with low latency.

7 Appendix

References

[1] Wikipedia. Lane departure warning system — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 08-May-2016].

[2] Rajamani Rajesh. *Vehicle Dynamics and Control.* 2012.

[3] Mohamed Aly. Real Time Detection of Lane Markers in Urban Streets. 2008.

[4] Wikipedia. Ziegler–Nichols method — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 05-May-2016].

8 Individual Contributions

Thomas - Mechanical Assembly, Longitudinal Controller
Travis - Mechanical Assembly, Lateral Controller
Alex - Computer Vision Implementation, Script Kiddie, Unix Guy