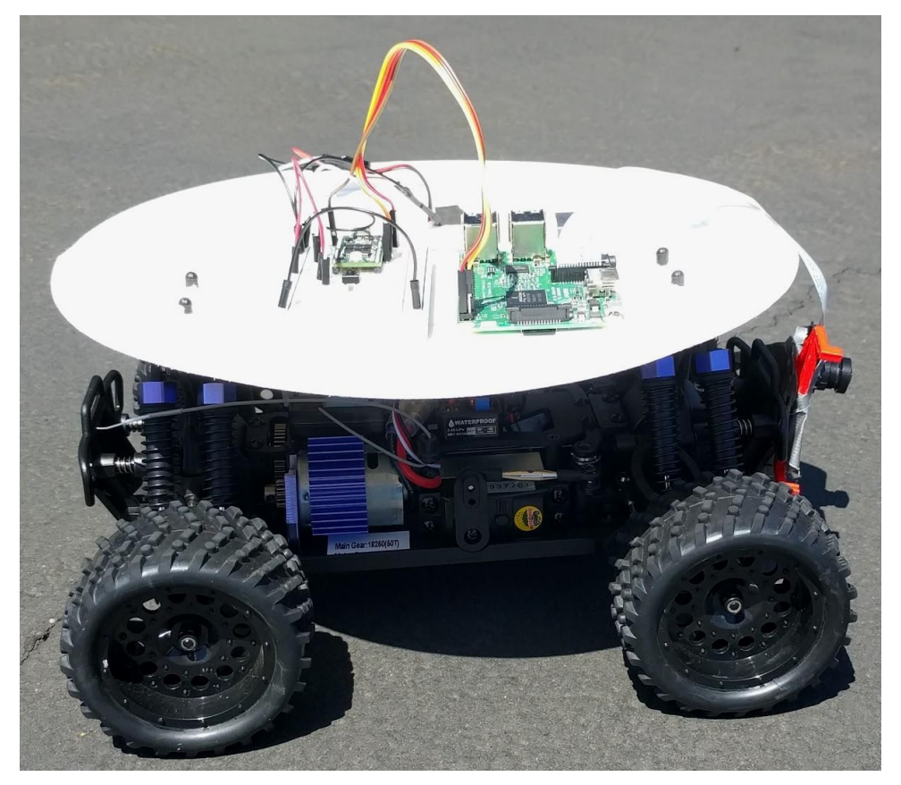
Autonomous Driving - 1/16th Scale Autonomous Racecar

Heather Bradfield, Dylan Hart • Dr. Jason Isaacs • COMP 499 - Capstone

Introduction

- For our capstone, we built an autonomous (self-driving) race car. The car can navigate its way around a race course at high speed while maintaining control. Using vision processing it is able to detect and follow a broken yellow line. Using various sensors and an internal model it can maintain odometry during aggressive driving without the use of positioning systems like GPS.
- Autonomous driving will redefine the automotive world. Drivers are at-fault for most accidents. There are so many factors at play when a driver gets behind the wheel and safety relies on a driver that is purely focused on the road. With autonomous driving, human error would be nullified leading to fewer accidents.
- There are many benefits to autonomous driving, but autonomous vehicles are still undergoing research and can't be trusted to fully operate without error just yet.
- This self-driving race car was based off the cars built specifically for the DIY Robocars competition [1]. This competition is held monthly in Oakland, California. The overall aim of the DIY Robocars group is to make and race pro-level autonomous cars on a budget.



Calibration

- When using a fisheye (>160 degree field-of-view) lens, OpenCV's fisheye calibration removes lens distortion [2].
- We needed to calibrate the position of the ground for the Bird's Eye transform using OpenCV's getPerspectiveTransform:

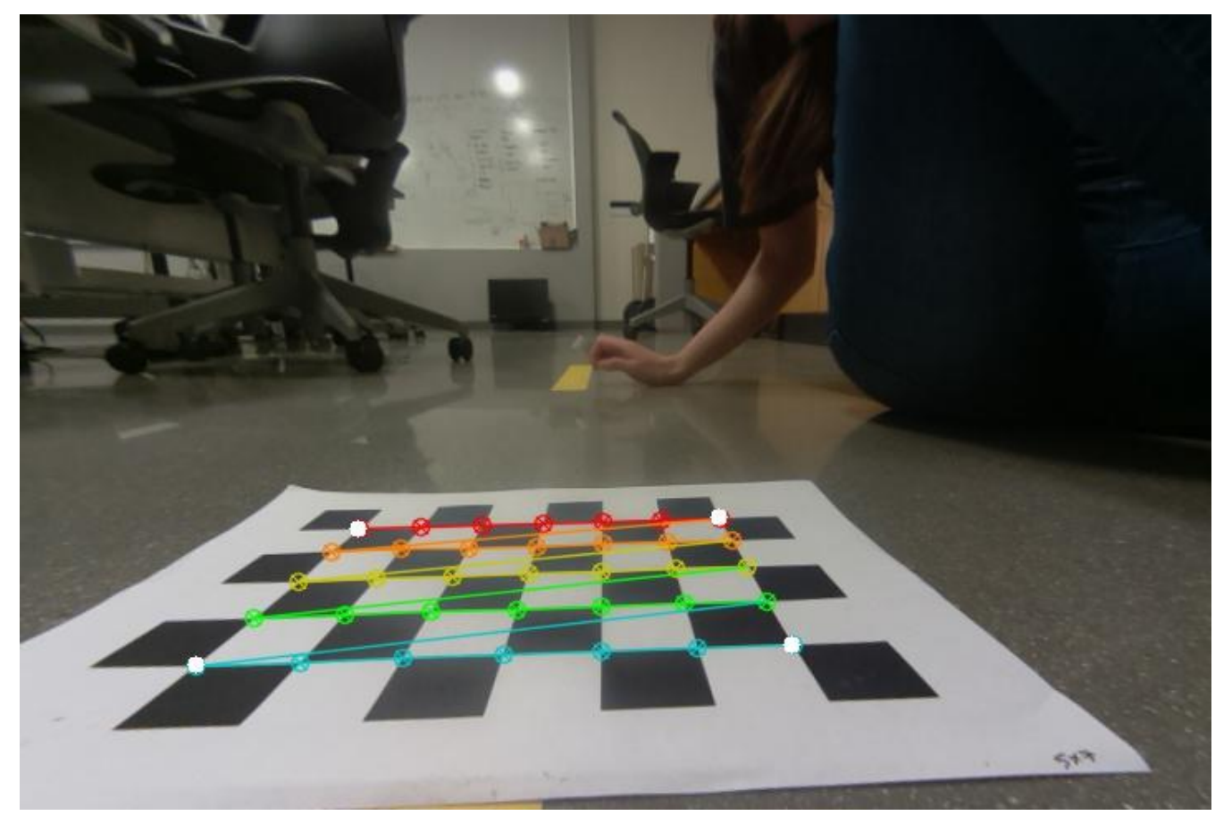
1. Compute undistortion and rectification transformation maps
2. Apply a generic geometrical transformation to the image
3. Since the camera was mounted upside down on the race car, rotate image 180 degrees
4. Find the positions of internal corners of the chessboard
5. Find outer four corners
6. Calculate a perspective transform from four pairs of the corresponding points:

$$\begin{bmatrix} t_i x_i' \\ t_i y_i' \\ t_i \end{bmatrix} = map\_matrix \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

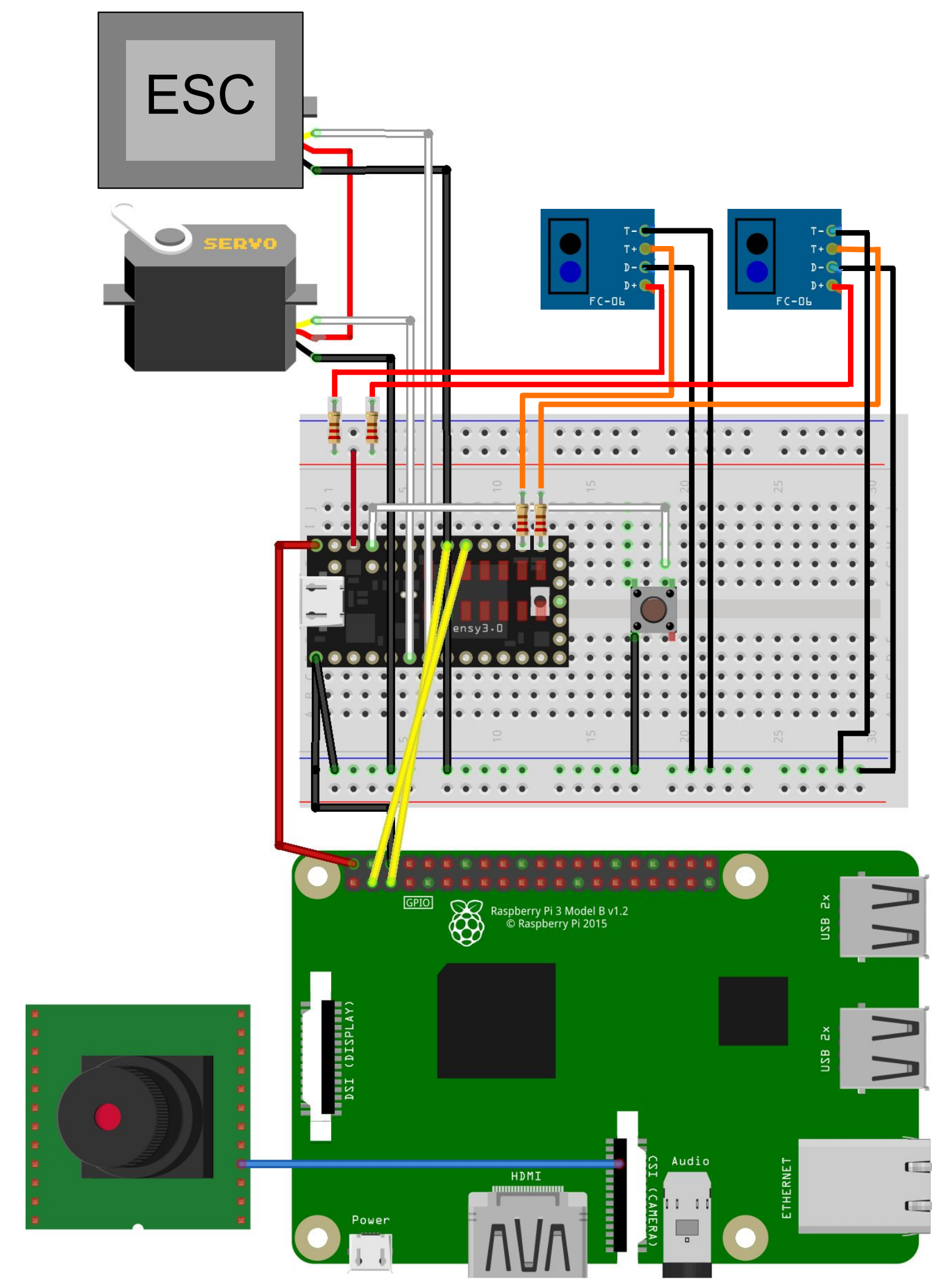
where

$$dst(i) = (x_i', y_i'), src(i) = (x_i, y_i), i = 0,1,2,3$$

Raw Calibration Image



Chessboard Detection Visuals (Undistorted)



Race Car Circuit Diagram

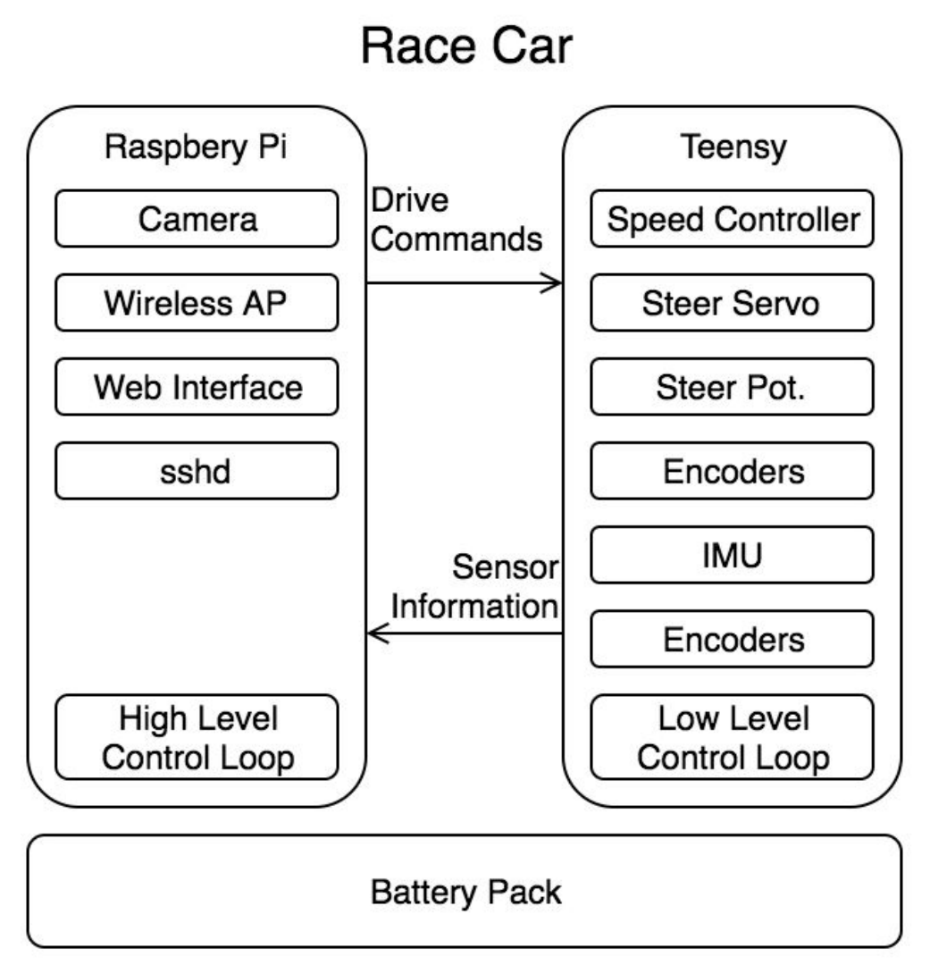
Measuring Speed

- Measuring the speed of the car is done using encoders on the rear wheels. We created encoders using IR optical reflection sensors which consist of an IR photodiode and an IR phototransistor.
- White markings were then made on the inside of the rear wheels using paint markers. These markings are more reflective than the unmarked black portions of the tire.
- Speed and distance are measured by recording the pulse width (w) of the signal from the reflection sensor. The pulse width can be converted to RPM of the drive wheels:

$$\frac{1}{w}(ticks/sec) * \frac{1}{12}(rev/tick) = \frac{1}{12w}(rev/sec)$$

$$\frac{1}{12w}(rev/sec) * D_{wheel} * \pi = current\ speed$$

Methodology

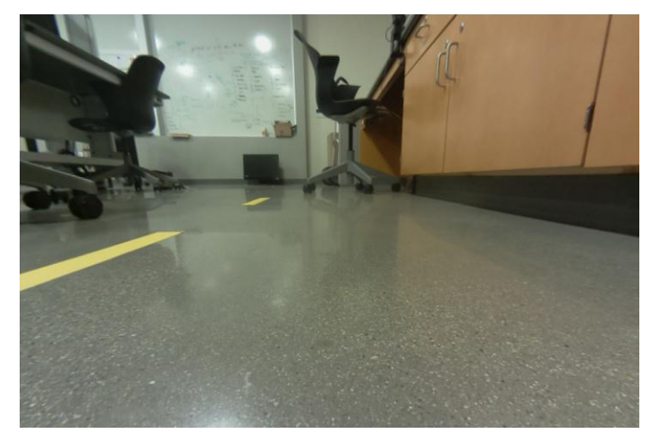


- Our race car is an off-the-shelf RC car with mounted encoders on the wheels and a potentiometer on the steering servo. We can perform feedback control with these existing systems.
- The first controller is the Teensy 3.2. This controller is responsible for interfacing with the various hardware components with the exception of the camera. It controls the actuators and aggregates the sensor data.
- The second controller is the Raspberry Pi 3. This controller runs a full Linux stack and is responsible for the vision processing and high-level control. The Teensy focuses on how to drive by controlling the hardware while the Pi focuses on where to drive by running our line-follower algorithm.
- The racecar interacts with two main external entities – the user and the environment. A user is anyone who wishes to connect to the system in order to develop, perform maintenance, test, observe, or even "play" with the racecar. A user will be able to connect to the race car through SSH.

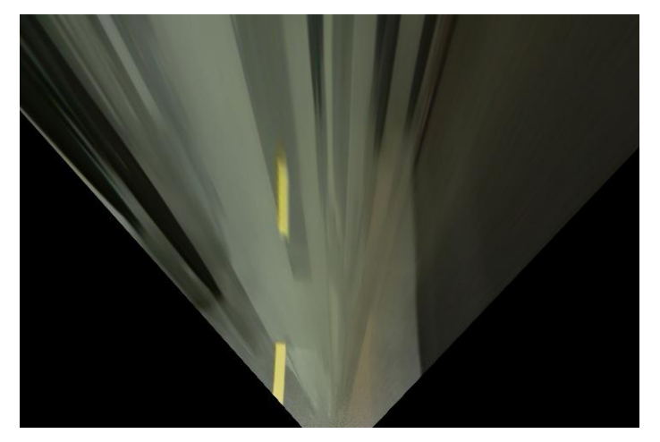
Detecting Lines

- To mock the yellow broken lines found on roads, we used yellow tape 35mm – 55mm (1.5-2 inch) wide. In our test courses, the tape was dashed with ~20cm of tape and a ~20cm gap.
- In order for the racecar to see each yellow line, each captured image was filtered:

1. Convert image to the YUV color space
   a. YUV represents the human perception of color more closely than the standard RGB model
   b. Y is the luminance (brightness) component while U and V are the chrominance (color) components [3].
2. Apply Sobel Filter to U channel using custom kernel
   a. U = B' – Y' (blue – luma)
   b. Kernel = $\begin{bmatrix} -1 & -1 & 2 & 2 & -1 & -1 \\ -1 & -1 & 2 & 2 & -1 & -1 \\ -1 & -1 & 2 & 2 & -1 & -1 \end{bmatrix}$
3. To differentiate the pixels we are interested in from the rest, perform a comparison of each pixel intensity value with respect to a threshold
4. Apply the Hough Line Transform to detect straight lines
5. Fit a polynomial curve (degree 2) to the detected lines



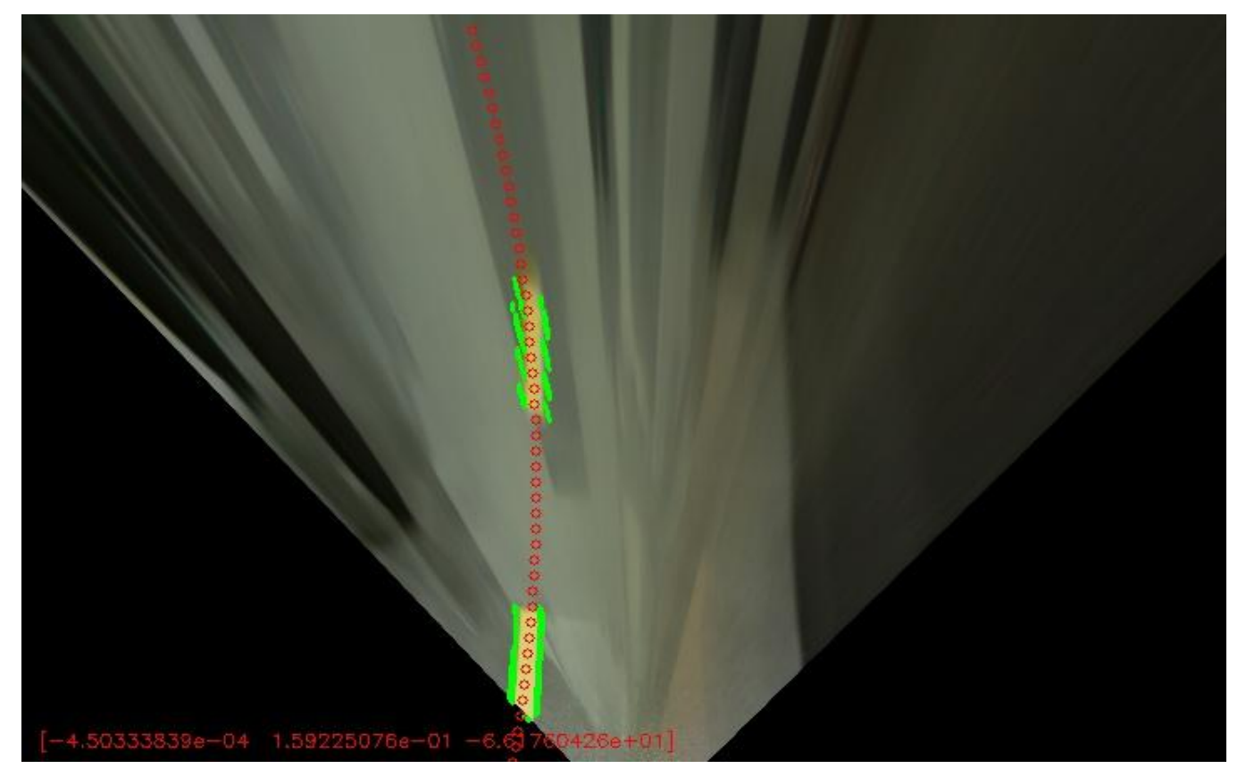
Undistorted Image



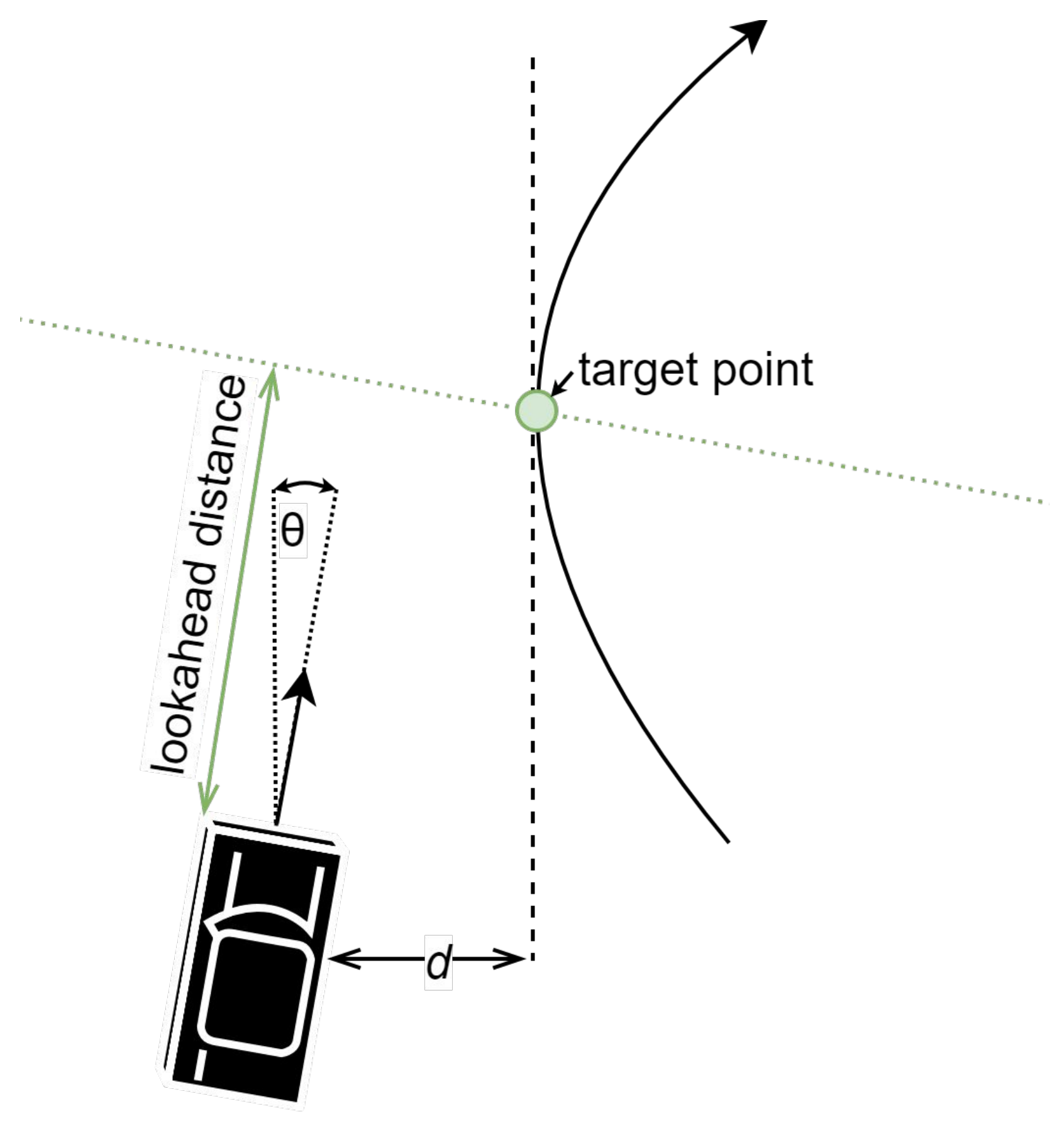
Birds-Eye View

U-Channel

Thresholded Sobel Image



Polynomial Plotted with Detected Lines



Steering Calculation

Line Following

- The car uses the following algorithm in order to drive along the curve that was created earlier:

1. A target point on the curve is chosen by selecting the point that is a constant "lookahead distance" in front of the car.
2. The tangent line at the target point is calculated.
3. The car then tries to drive along this tangent line using PID control on it's steering while travelling at a constant speed.
4. Since the position of the target point changes as the car progresses along the course, the car will always be trying to realign itself with the course.

- The steering calculation takes into account two variables:
  - The distance from the line (d)
  - The relative angle to the line (θ)
- The distance from the line is used to determine a target angle that the car should be driving at to either stay on or approach the line.
- Then the relative angle can be compared to this value in order to determine the current course error as part of the PID calculation.

Results

- We were able to build an autonomous racecar capable of navigating along a given course using our line follower algorithm.
- We found images that the race car was unable to process properly.
  - This was due to a combination of poor lighting, reflections on the floor, other yellow objects in the environment, and other factors.

Future Research

- Perform mapping of the course being navigated by the car
  - Find a raceline that optimizes vehicle maneuvers for speed
  - Can stitch Bird's Eye projections together as a panorama to form an aerial view.
- Look into different ways to detect lane lines in order to compare effectiveness of alternate algorithms.

Literature Cited

1. "Autonomous Cars For the Rest of Us." DIY Robocars, diyrobocars.com/.
2. "OpenCV Documentation." OpenCV 2.4.13.6 Documentation, docs.opencv.org/2.4/index.html.
3. Wright, Christopher. "YUV Colorspace." Softpixel, softpixel.com/~cwright/programming/colorspace/yuv/.

Acknowledgements

- We would like to acknowledge Dr. Jason Isaacs for being a supportive mentor, helping us purchase materials, and letting us use the Embedded Lab for testing.
- Thank you all CSUCI Computer Science faculty for your insight, encouragement, and guidance over the past 4 years.