Team #2 Report, Final Report, Obeying Rules of the Road Report - 12/17/2018

Jane Tangen, Shiyan Yin, Harrison Packer, Adam Rivelli, Vishnu Karpuram

Final (Rules of the Road)			WEEK 1							WEEK 2							WEEK 3						
Start Date: 11/26/18	Time Cost	Assignee(s)	M	T	W	TH	F	SA	SU	M	T	W	TH	F	SA	SU	M	T	W	TH	F	SA	SU
GANTT Design	1 h/Week	All				■	■														■		
Group Meetings	1.5 h/Week	All			■							■							■				
CV Finishing (Curve detection, stop detection, stop line detection)	10 Hours	Harrison/ Shiyan			■	■	■			■	■	■	■	■	■	■	■	■	■	■	■		
Ping Mounting, Camera remounting	8 Hours	Adam		■	■	■																	
State Machine	5 Hours	Jane			■					■		■	■	■			■	■	■	■	■		
Master Program (accept destinations, etc.)	4 Hours	Vishnu				■	■			■	■	■	■	■			■	■	■	■	■		
Report	3 Hours	All										■	■	■							■	■	■
Hours Per Week Per Person		Harrison	3 Hours							18 Hours							28 Hours						
		Jane	2 Hours							17 Hours							29 Hours						
		Adam	2.5 Hours							20 Hours							27 Hours						
		Shiyan	2.5 Hours							20 Hours							30 Hours						
		Vishnu	2.5 Hours							19 Hours							28 Hours						

Figure 1: Our Gantt chart for the final segment of the project.

Robot's Final Working State

In the demo, our robot was teleoperated, not autonomously controlled. We manually told our robot to go the correct way through an intersection, and then we reset our robot to the next intersection. However, this doesn't tell the full story of our robot's functionality. The robot was not demo-ready, but plenty of it worked for non-demo testing. Our final robot is shown in Figure 2. Our code is here.

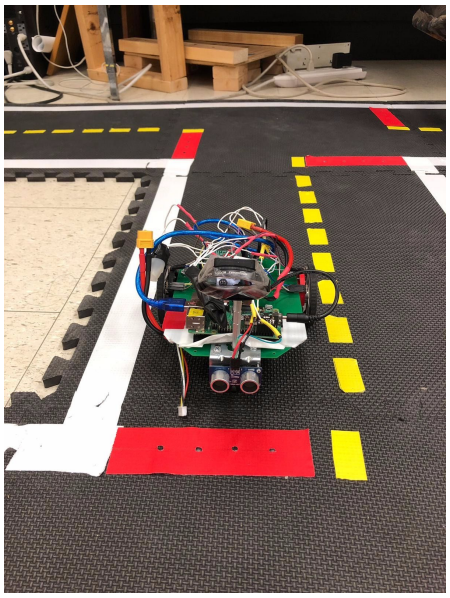

Figure 2: Our robot in its final state.

Visual Lane Following

Our robot could follow lanes through the left and right curves. We used roughly the scheme presented by Team 6 in their best practices talk. We converted our images to HSV, and we set a threshold for what would be considered a white or a yellow pixel. When the robot is in the straight mode, we found all of the yellow pixels and calculated the median of their x-coordinates, and did the same with all of the white pixels. We then averaged these together to find the center of the lane. When the robot is ready to turn left, we get the median of the x-coordinates of white pixels, then we subtract a distance (roughly a constant number from white lane to the center of the lane, calculated when the robot is on straight mode) from it as the center of the lane. We did the similar thing to make the robot turn right, but we increase the calculated distance to the median of the x-coordinates of yellow pixels as the center of the lane. When we calibrated our vision program, we found out what x-coordinate the center of the image should be at. We calibrated several times because the camera shakes during experiments. The error that we fed into our PD controller was the difference between the center that the vision program calculated and the value that we found during calibration.

We also successfully detected red stop lines and the green lights within them using vision. We checked for the number of red pixels in the current frame, and if the proportion of red pixels in the image was greater than a threshold, the robot stopped. The vision program would retain control of the robot until it detected that the green lights went on (once again, by using the proportion of green pixels in the image). Once it detected that the lights had turned green, it signaled this to the master program.

We also thought about checking the position of the color to enhance the performance of the vision. For example, we would have checked that the red and green pixels were mostly at the center of the image. But, we found that color detection was enough to give the correct feedback, especially after we put a polarized lens over the camera to remove the glare, although we did have some position checking for extreme cases. After putting considerable effort into optimizing the vision program, we got it to process between 5 and 8 images per second. We could have reduced the size of the image and tuned the color range to further improve our program. Figures 3 and 4 show some of the outputs from our vision program

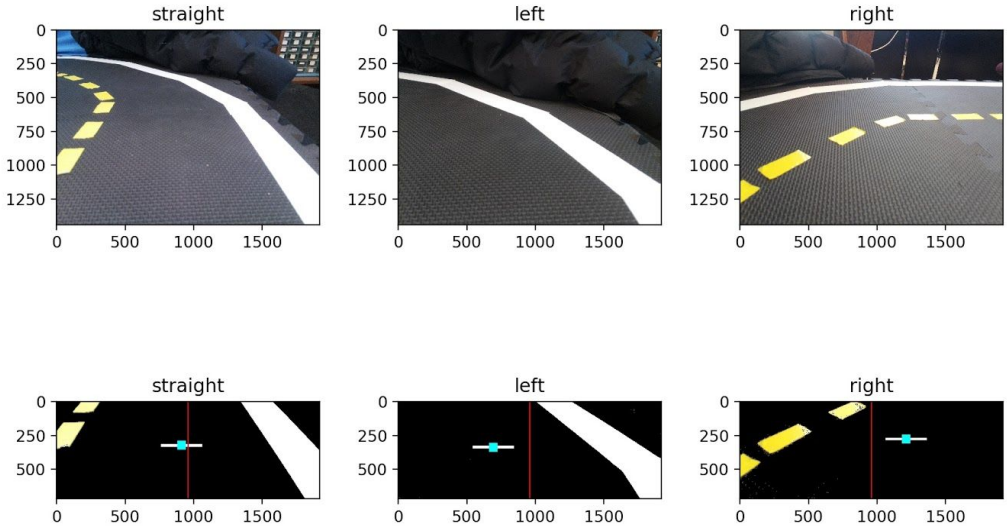

Figure 3: Images showing what the image taken looked like in straightaways and turns, and an image showing the values our program calculated.

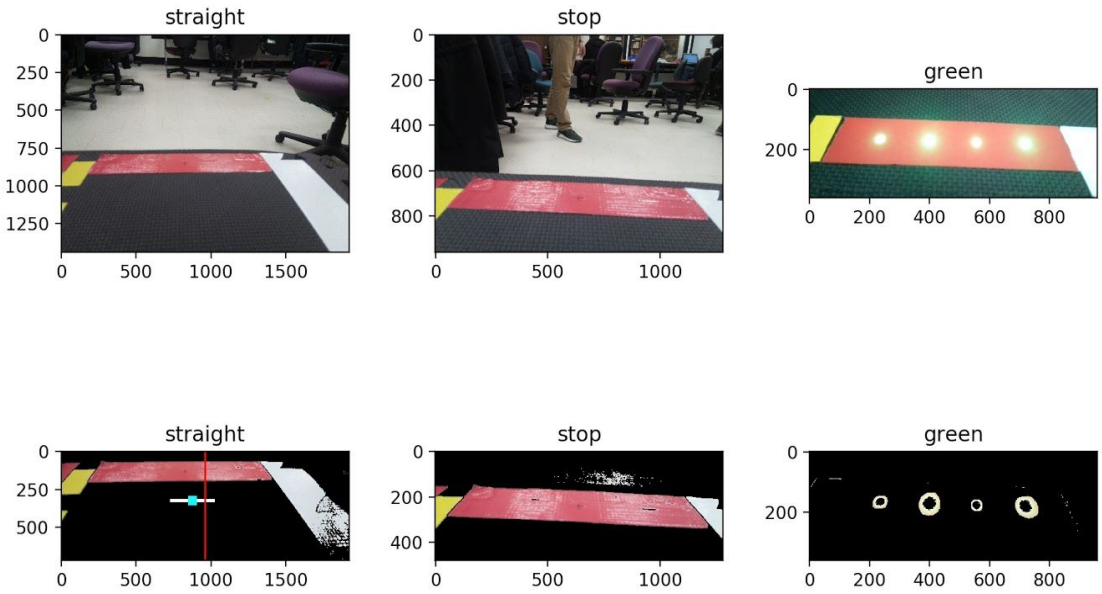

Figure 4: Images showing the images taken by our camera and the windows focused on by our program.

PD Controller

We used the PD controller equation to generate an value that would be integrated onto the PWM commands for each wheel. Initially, the Raspberry Pi would send over raw motor commands, however this was not accurate because both motors did not behave the same. In order to correct this, we generated a PWM-velocity equation by running the robot for a certain amount of time and tracking the number of rotations for each wheel. This gave us an accurate measure of PWM commands that could be used to with the PD controller.

Once we got the PWM equation, we had to tune the constants in the PD controller. We started of tuning by having a value of 0 for B and 0.01 for K. A higher K value means the robot would correct more aggressively and a higher B value adds damping. Once we figured out a reasonable value for K where the robot would oscillate about the correct point, 0.023 in our case, we started tuning B. A value of 0.0123 seemed to be perfect for the damping factor.

An issue we faced was that our midpoint detection was not very accurate. When computed, we found that the midpoint should be 421, however this value seemed to move the robot more towards the white line. Using a value of 530 made it much more centered and allowed us to tune the PD well. Since we had to tune so many values: K, B, midpoint, distance from yellow, and distance from white, we ended up spending a lot of time on tuning visual processing.

Towards the end, we were able to get almost perfect lane following with our tuned values and were impressed by how accurately the robot would follow the lane.

Adaptive Cruise Control

Our adaptive cruise control system was rudimentary. The ultrasonic sensor fired off every second, and if it detected any obstacles closer than 20 centimeters away, it completely stopped the robot until the obstruction was removed, and then it continued at its previous pace.

Path Planning and State Transitions

Our path planning system used the states given in an email, plus a few extra states that we used. These extra states are shown in Figure 3. These extra states were used because they are where we changed behaviors, allowing for only 4 types of transitions for the whole map- 3 ways to get through an intersection, and path following until a red line. For example, to traverse the edge 8-10, we first did a blind turn using our odometry. Once the turn completed, the robot would begin visual lane following. So, state 10* is a logical addition to make - it is where our robot needed to switch to lane following.

Each edge between the nodes contained a piece of metadata describing how the robot should transition between the two states. In the previous example, 8-10* had the metadata 'turn left,' and 10*-10 had the metadata 'follow lane.'

First, our path planning program calculated the shortest path between each of the given states. Then it iterated through the edges on this path. If the metadata for an edge was to go through an intersection, it sent a command to the Arduino to complete the turn. The Arduino would complete the turn, and then indicate to the path planner that it had completed the turn. Then, the path planner would relinquish control to the lane-following program until it reached a red line. The path planning paused until the green line was seen by the camera.

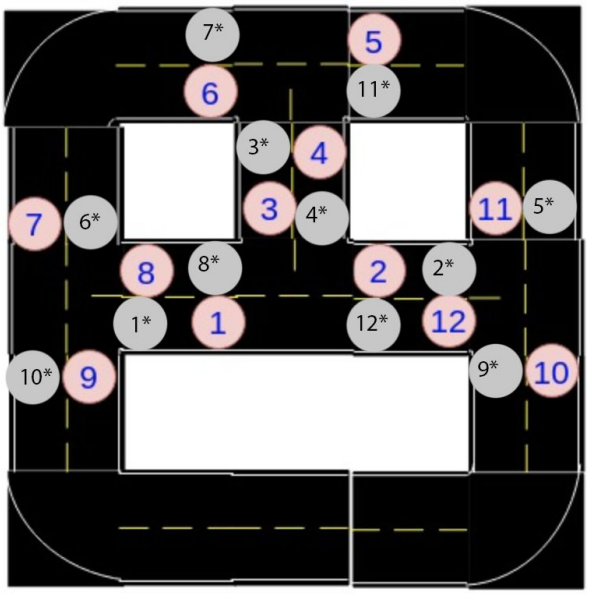

Figure 3. Annotated map showing the additional states that we planned our paths using.

Tying it All Together

Tying the program together was difficult because we gained quite a bit of code to test in a short amount of time. It took a lot of debate to agree how states would work. We initially decided to have the Pi control the robot through the blind turns, but we eventually decided that they would be programmed into routines on the Arduino. The ultrasonic sensor and encoders ended up being controlled entirely by the Arduino, but this was decided fairly last minute. The state machine depended on these routines, and once the Arduino could take commands, state transitions were programmed on the Pi.

While we could test the path planning on a computer, we couldn't test everything together - which was a much larger task. It takes a lot of time and testing on the Pi to get it all right, and we used literally no time to test it all together. The largest unresolved issue was communication with the Arduino. At some point with everyone working on the code, the delicate code on the Arduino suddenly could only take in several commands before over-filling the buffer. The Pi requested the location from the Arduino every 200 milliseconds, which overpowered the Arduino almost immediately. This problem and some other small problems made our project fail spectacularly.

Future Revisions

The biggest problems we faced in the final segment of this project weren't technical challenges, but were planning challenges. For the lane following project, we remarked that we wished that had put more work into the project earlier. We did better on the final segment of the project - we put in considerable amounts of work in week 2 of the project, which we still believe is early enough to begin work. However, we didn't use our time as effectively as we should have.

At several points, we had a vision system that worked well enough, even through turns. But, each time we had it at an acceptable state, we found small issues that we wanted to improve upon. So we made those changes, and then had to debug those changes, and then had to re-tune parameters for the PD controller. This revision loop continued into demo day - until probably about 7:00 pm. We should have accepted an earlier version that was 90% as good as the final version.

Because our vision code wasn't finalized, we couldn't integrate it into the overall program, and we couldn't fix the issues that came up during the integration, which were described in "Tying it All Together". In addition to these issues, our Pi began to run out of memory, and eduroam and CICS had their issues - the combination of these two problems made it so when we ran into issues in our integration, we couldn't debug them before either network or memory problems made it impossible to use the Pi.

These problems wouldn't have been so frustrating and insurmountable if we had planned ahead and began our final integration earlier. We could have just walked away from the lab for an afternoon if we ran into these problems earlier, but because everything was coming down to the wire, we had to stick it through and be hampered by these problems.

Acknowledgements

Thank you Rod and Khoshrav for putting together an excellent class. Overall, we really enjoyed this class, and we think it has given each us valuable experiences. Not only did we gain valuable insight into the challenges faced by physical systems, but we also learned more about project management in this class than in any other college class.