Road-following Robots
CompRobo
Paul Ruvolo
Shane Skikne and Mac-I Crowell

For our code, please see StopSignFinder.py and Driver.py in the scripts folder.

What was the goal of your project?
The goal of our project was two-fold. On the one hand, we wanted to make a robot that could travel along a line while watching out for stop signs, just using the camera and odometry. This gave us the challenge of navigating a line perfectly while ensuring we don't pass the stop sign on the course. We could, of course, optimize our robot to do that perfectly by moving slowly and carefully analyzing every picture. To make it more interesting, we decided to challenge another team to also complete the same course and see who could complete the course faster, without sacrificing accuracy. So overall, our goal was to follow the lines and stop at the stop sign as fast as possible,

How did you solve the problem? (Note: this doesn't have to be super-detailed, you should try to explain what you did at a high-level so that others in the class could reasonably understand what you did).

In order to drive the robot along a line we looked for the mean location of the line in the frame of the camera. This allowed us to feed that one parameter into a PID control loop which would then pilot the neato. We also used feature matching followed by homography to find the stop sign. Once the stop sign is found, we calculate the distance to the sign based on its approximate height in the image and then use odometry to determine when we have reached it.

Describe a design decision you had to make when working on your project and what you ultimately did (and why)? These design decisions could be particular choices for how you implemented some part of an algorithm or perhaps a decision regarding which of two external packages to use in your project.

One of the more interesting decisions we made was to put our driving code and our stop sign detection code in separate nodes. This was done because we needed to run them on different threads and the functions require very little communication. We needed to run them on different threads because the stop sign detection could not run fast enough

to not interfere with the mission critical driving function. ROS made it very easy to compartmentalize code but allowed communication as needed. The publisher/subscriber architecture is great for asynchronous communication between threads.

How did you structure your code?

For both simplicity and better performance, we chose to break our code into two main classes that only interact when it is necessary. We have the Driver class, which steers the robot around the track and handles all motion. Then, we have the StopSignFinder, which is constantly watching out for the stop sign. When the StopSignFinder thinks it has found the sign, it publishes to the sign_found topic the probable distance from the stop sign and the odometry of the robot at the time the picture was taken. This is the only interaction between the two classes.

Both classes have a callback function to handle received images. They both then use their images as necessary. Driver has another callback for when information about the stop sign is published by StopSignFinder. Both classes also have a variety of helper functions to take care of some specific tasks. The Driver class also has a helper class for doing PID control.

What if any challenges did you face along the way?

One of the major challenges we faced was the trade-off between performance and accuracy. For example, each time we search for a the stop sign took .2 seconds. While this is much better than what we started with, it is still much too long to be running every time we received an image. The first step we took to handle this issue was break the code into the two classes described above, StopSignFinder and Driver. This allowed the code to run on separate nodes and not hinder the other code. While this helped, we still found that our threshold for detecting the stop sign was too high because it required too much computing time. We lowered the threshold required to detect the stop sign and this allowed us to run at more acceptable speed. We then focused on filtering out more false positives from the StopSignFinder. While this did not reduce computation time, it did make our guesses much more accurate, which allowed us to have more faith in the system and have less safe guards in place. At one point, the robot had to locate the stop sign 8 times to be sure it was there, but, after these changes, the robot now only has to find the stop sign twice before it is sure.

What would you do to improve your project if you had more time?

One thing that we would do if we had more time is detect the line using a different metric than color. While color allowed us to find the track relatively easily it is very susceptible to changing lighting conditions. It may be possible to use complex edge detection to pick out the road or some other technique. It would also be very interesting if we could have the robot build a map of the track on the first run and then use the map for navigation.

Did you learn any interesting lessons for future robotic programming projects? These could relate to working on robotics projects in teams, working on more open-ended (and longer term) problems, or any other relevant topic.

Mac-I:
One thing that I learned was that ROS works very nicely for compartmentalizing code. It is a great framework so that code communicates only when needed and the communication can happen asynchronously. Additionally this project reaffirmed my feeling that cameras can be useful in a carefully controlled environment but beyond that their usefulness is greatly reduced.

Shane:
This project was an excellent first experience in working with code where time is an actual constraint. I have not worked on projects where computation was so expensive that we had to lower our accuracy to accommodate for it. In the past, I've worked to optimize my code, but I always did so, because I knew it was the correct thing to do. In this project, we had to find a way to make searching for the stop sign less computationally intensive or else the project wouldn't run effectively. Knowing how to make code more efficient is not always enough. We also had to find ways to be creative about how the code ran (breaking the code into multiple scripts, not checking every image, lowering the size of query image so it is easy to search for). This wasn't really a lesson, as much an experience. The lesson here would be that writing code that is easy to break up and move around is so important. It allowed us to move the code around and try many different strategies with just a few minutes of work so finding the best way to run our code was as painless as possible.

Sources:
http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html
http://code.activestate.com/recipes/577231-discrete-pid-controller/