CS4278 Project Report

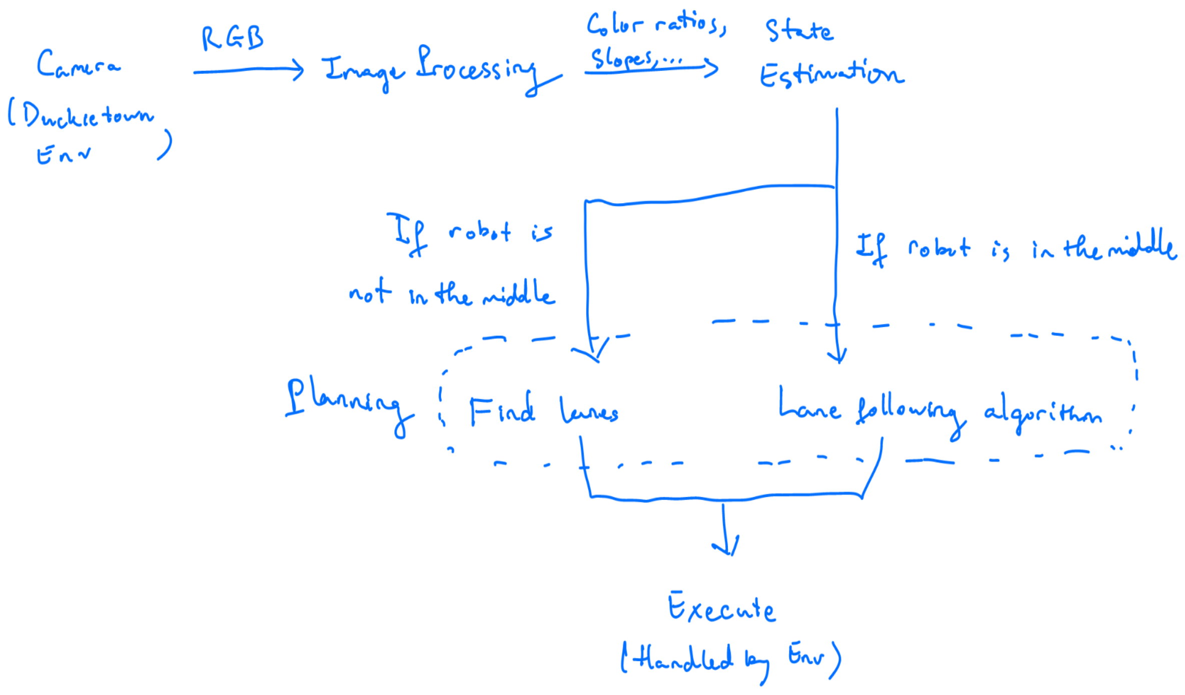
Student name	Matric number
Tran Quang Thanh	A0184577M
Nguyen Dang Phuc Nhat	A0184583U

1. Stop sign recognition
   In the observation, most of the objects do not have red color; therefore, I use red-color detection to classify stop signs. I use a machine learning model to classify if the color is red. Then, I apply the filter to the observation to detect red regions inside the image. However, there are other objects containing red color such as road markers or car lights, and there are also noises from the machine learning model. Therefore, I assign contours to all areas that have red color. After that, based on the area of the contours, I can remove the noises, and based on the shape of the contours, I can remove the road markers. In the end, I will have the stop sign.
2. Classic Modular Robot System
   The end-to-end Reinforcement Learning posed rather daunting questions: "Will the model converge?" and "When will it converge?". Since we're not that well-versed in machine learning, we did not get very lucky with the models tried. The time needed to train them was much larger than we can afford. As a result, we opted to implement a modular system at the last minute. There are no neural networks involved in this system apart from the sign recognition. The system comprises a few components (though the code may not be very clear). Our system focuses on staying 'alive', i.e. being able to run 1500 steps. Many cases are more successful than others.
   - Perception: Duckiebot sees the environment through the camera. We use the RGB value array provided by the Gym-Duckietown environment as perception input. From this, the system will try to determine the lanes: the yellow lane and the white lane. By drawing lines selectively, we can estimate the lanes and calculate the average slopes of the 2 lanes. However, due to the limited experience with image processing, we could not reliably extract the lanes. Interestingly, the environment is full of 'perfect' lines that the Hough algorithm could identify (and the lanes are not in this group). This creates a lot of noise information. Therefore, we had to extract the ratio of yellow pixels and white-gray pixels on the 2 bottom corners of the image. This is how map4 destroyed our happiness. In short, the perception part of our system does the following: find relevant lines and calculates the average positive and negative slopes, find stop signs, find the red bar that marks intersections, find the 'pit' that marks end of roads, and extracts the ratio of yellow and white-gray pixels from the 2 bottom corners and a bottom middle stripe.
   - State estimation: The output of the perception component is very noisy and incomplete, so we could not implement mathematically sound estimation algorithms. Our approach is to use the color ratio and the average slopes mentioned before. A good position of Duckiebot has a certain amount of yellow on the left, and a certain amount of white on the right. Additionally, the absolute values of the slopes should not differ too much. If a slope is very small, it can indicate a turning event coming up later. In reality, the slopes might not always be available (relevant lines not detected) or straight up wrong (perception kept the wrong lines) and the color ratio might be misleading (there's the shiny white bus). Furthermore, the color ratios would also depend on the RGB value constraints. If the constraint is too tight, there's too little information. If the constraint is too laxed, the opposite occurs. When the stop sign comes into play, the system measures its distance with the sign by the number of steps it takes after it last sees the sign. If the system is 200 steps away from the last seen sign, it is probably at least 0.3m apart. In fact, since the speed used is 0.1 (which is 0.14m/s in Duckietown), 200 steps can measure to roughly 0.3m.
   - Planning: Again, the potential is limited by the unreliable input. A pair of speed and steering is the output after the robot has processed the input image. In the ideal case where the color ratios are proper and the slopes exist. The output would be [0.8, abs(negative_slope) - positive_slope]. In other cases,

the output speed and steering varies. When the bot is first initialized, it does not think it's between 2 lanes and will wiggle, move slowly, or turn around to find the lanes. Due to the fact the bot is 'blind' at the beginning and cannot fully trust the visual processing, it is prone to running into illegal poses. Many cases of map3 saw this happen because of the green grass. There are instances where we hard-code a sequence of actions. It relies on the fact that Duckiebot will execute without fail. When the bot is determined to be within 0.3m from a stop sign, the speed will be curbed to 1. There are too many scenarios to be listed here. In fact, we had to handcraft the actions based on each map. Some maps are too different from each other. For example, map4 is a white color nightmare, map1 only has the end-of-road pit to worry about, map3 has some starting positions near the grass tiles that are hard to distinguish from yellow.

- Control: There is no explicit control algorithm implemented. The output from the planning step is sent directly to Duckiebot and it will process the actions with built-in methods.

A summary diagram of the system is below:


Camera (Duckietown Env)
RGB
Image Processing
Color ratios, Slopes, ...
State Estimation
If robot is not in the middle
If robot is in the middle
Planning
Find lanes
Lane following algorithm
Execute (Handled by Env)