Team name: What the duck?
Team members:
Kushal Jhunjhunwalla, kushaljh@cs.uw.edu
Andrew Wei, nowei@cs.uw.edu
Kyle Zhang, guohaz@cs.uw.edu

Problem of interest

People don't always use crosswalks which can lead to accidents. A person driving a car always has their eyes on the road and can stop for said pedestrian in a split second. As we go towards a self-driving future, we must have autonomous cars be cognizant of this scenario and be as good as and even better at reacting than humans to avoid all accidents. Hence, we try to incorporate this as our robot is roaming around duckietown! :)

Objective

The goal of our project was to avoid hitting duckies. Specifically, we wanted to make our Duckiebot traverse a map and stop when it sees a duckie in the path to our goal.

Setup

The physical setup involved building the maps with lanes, a duckiebot, targeting a stop-sign AprilTAG as the goal and another AprilTAG as a duckie. Both the duckie and the AprilTAG were stuck to a white box. The goal would be placed on the right side of the lane since it is Right Hand Drive. The duckie would be placed in the middle of the lane. We also controlled the lighting to be consistent and avoided natural lighting.

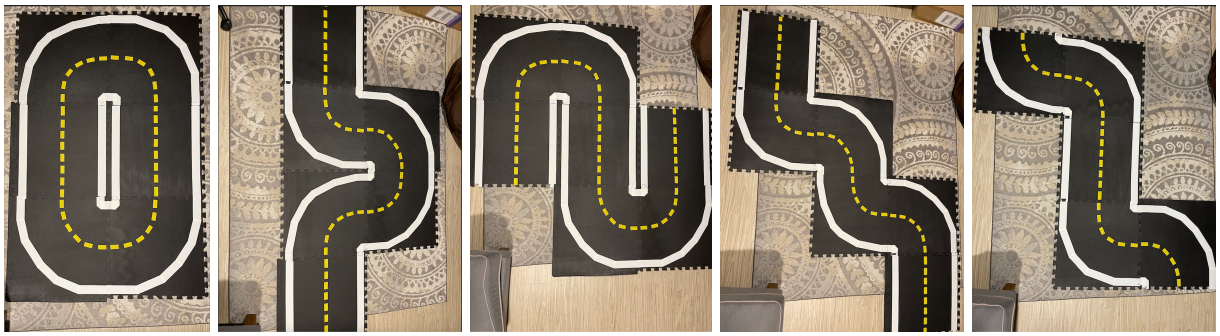

Figure 1. 5 maps built for testing

Approach

Duckie detection

The initial idea we had for duckie detection involved training a neural network model and doing object detection to detect the duckie. To do this, we followed the guide in [Duckie detection] to collect 2000 images of a robot moving around a map (with duckies) in simulation.

We first took an existing resnet 50 model off of PyTorch and used that as a basis for training. We trained locally with a GPU, but the model was too large and we could only fit one image on top of our model to train at a time. This means that we were essentially training with stochastic

gradient descent, so the model would take more time to train to converge and it would take more time to obtain a decent model. Unfortunately, it took 4-6 hours to train each epoch and after 4 epochs, the training accuracy was only 55%.

We then looked into training the model on Google Colab, which had a GPU that had more memory, which allowed us to train faster. Unfortunately, the session timed out within the first epoch, so we couldn't properly train there. In general, we were set too far back by setup issues to obtain a decent model and we didn't have the necessary resources to catch up.

There were also issues with using PyTorch directly on the Duckiebot. Many of the docker images were out of date and the Duckietown reference pages and StackOverflow did not have relevant information/working solutions for the installation. To our knowledge, the Jetson Nano is PyTorch-capable, but we could not find resources to make it Duckiebot-compatible.

Thus, we developed a Subscriber/Publisher setup where our desktop or laptop could subscribe to the node on the Duckiebot that publishes the image. After we get the image, we can run the detection model using the message. Finally, we could publish the detection results to a topic that the Duckiebot subscribes to, see Figure 2. We have this setup implemented in our code, but we were still missing a proper detection model. As a backup, we used an AprilTag as an honorary duckie for the purposes of our project, see Figure 3.

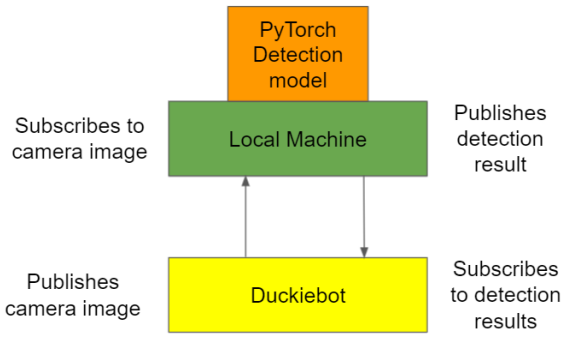

Figure 2. The Publisher/Subscriber detection workflow.

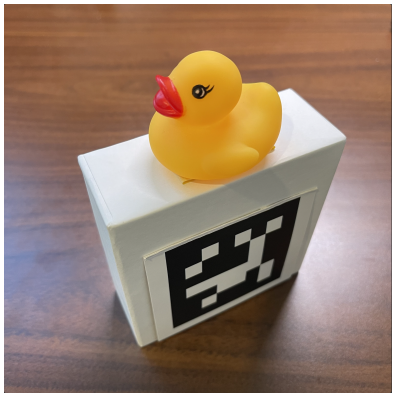

Figure 3. The honorary duckie (AprilTAG)

<u>Lane following</u>

For lane-following, we used a feature implemented in dt-core. The tutorial we followed can be found here: [Lane following (duckietown.org)]

The lane-following feature subscribes to the camera images and looks for lane edges, as can be seen in Figure 4. When an edge is detected, it calculates the angular and linear velocities the bot should maintain to stay in the lane and publishes them.

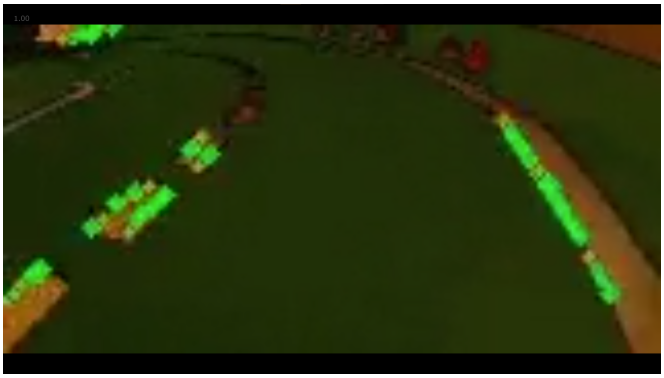

Figure 4. Lane following view

Path planning

Due to difficulties with our setup, we were unable to implement path planning completely, but we did have some ideas for how it could be implemented in the duckie setting if we had more time. Essentially, our ideas for implementing path planning involve taking time to plan, performing the plan, and then reevaluating our plan if we come across an obstacle in our path.

The first idea we had was to use A* and creating a vertex and edge representation of the map, in which the edges are directional and indicate the proper direction for moving along the road. There would also be backwards edges for when we need to back up, and vertices and edges along the middle of the roads for when we need to do a U-turn. To accomplish this, we were mostly missing a way to translate movement along an edge with movement in the physical map, which would've required integrating the localization and motion model from Project 1 to this new setting. Then to find a path to the goal, we would have to first plan the path and then follow it. If we see an obstacle along the way (using object detection), we could wait to see if the obstacle moves. If it does, we may continue on the path. If not, we can take a few seconds to update our idea of the map and block off the paths passing through the obstacle and recompute the best path to the goal.

An alternative idea was to use RRT* for non-holonomic cars to try to find the goal, where we interpret the middle of the road and the boundaries as obstacles we couldn't cross. Then we could sample random points along the map until we find a path going towards the goal. If on the path, we find that there's an obstacle in the way (using object detection), we can update our idea of the map, replan, and then take the next viable path we find. This would also require localization and a motion model.

Stopping for a goal

We were trying two approaches to stop for a goal:

- The bot has complete knowledge of the map and we make a path from a start coordinate to a goal coordinate in a map as we start. Then when we encounter unseen obstacles we reroute.
- We look for a goal AprilTAG and the bot moves on the map until it detects the AprilTAG

As we make our way to this goal AprilTAG, we may encounter some pedestrians (duckies). To avoid accidents, we should stop for the duckies but stay ready to continue our search for the goal when the duckie is out of the way as that is our final destination we want to reach.

Stopping for duckies

As discussed above, we were training a vision model to detect duckies but we were not very successful at it and hence as a contingency we used AprilTAGs to denote duckies (Figure 3).

We then ran the bot around until we reached a threshold distance of 5 cm away from a duckie. We would stop and then follow the duckie if it moved along the lane but maintain the same distance. When we reach our overall goal of the STOP sign, we would no longer follow the duckie to maintain the fixed distance as we had achieved our higher level objective.

Evaluation results

First of all, we tested the performance of the lane following package in dt-core. The robot can follow the lane in most cases. However, it fails to stay in the middle of the lane while the lane has too many turns continuously. One possible reason is that the layouts of the markings are not perfect, another reason can be the model is only trained for simple road layouts, or that mechanical errors get compounded with multiple, consecutive turns.

In addition, we tested the performance of reaching the goal. On the five maps we have, we randomly select goal locations, which could be on the straight road or any place during a curve. Table 1 below only shows the success rate in general.

Result	# of times
Stop	47
Went Over	8
Total	55
	Success rate: 85.46%

Table 1. Goal Navigation Result

Result	# of times
Stop	18
Hit Duckie	4
Lane Following Fails	3
Total	25
	Success rate: 72.0%

Table 2. Duckie Avoidance Result

Apparently, it is harder for the robot to detect the goal and calculate the distance when the goal is placed in a curve.

Finally, we tested the performance of stopping in front of duckies. On the same five maps, we placed the duckie randomly on the robot's path to the goal. Most cases of hitting the duckie is

caused by incorrect estimation of distances. It would happen when the AprilTAG publishing frequency is low. However, once the robot can stop for the duckie, it can always keep a fixed distance to the duckie while we move the duckie forward. Sometimes, the lane following would fail once we placed the duckie or removed it. Our assumption is that the box for holding the duckie is too big and too white, which might confuse the lane following model to behave differently.

The experiments we performed were not comprehensive, since we should have done more trials and recorded specific results for different cases. However, as we mentioned earlier, technical set-up issues were a huge problem for the group, so we ran as many trials as possible.

Qualitative Results here: https://youtu.be/Fcb-TL2SdWM

Conclusion

In conclusion, while we didn't make as much progress as we had hoped, we believe that we made good strides towards the goal despite the numerous setbacks encountered when working with the Duckiebot environment. In the end, we learned that working with and starting to work with robots is non-trivial and requires a lot of patience and resources in order to get everything up and running.

We were able to avoid hitting an object representing a duckie and stop at a goal and we hope that real systems have the safety features we demonstrated in this project.

References

- Object Detection: https://docs.duckietown.org/daffy/duckietown-learning-robotics/out/lra_object_detection.html
- Lane Following: https://docs.duckietown.org/daffy/opmanual_duckiebot/out/demo_lane_following.html
- Qualitative Results: https://youtu.be/Fcb-TL2SdWM

Individual Contribution

Kyle

- Create map
- Tested all physical maps and run evaluation
- Work on AprilTAG for detection of duckie and goal
- Maintaining fixed distance with duckies
- Helped with lane following
- Video editing

Andrew

- Create map yaml and png
- Duckie detection training on laptop and Colab
- Set up initial Pub/Sub model
- Help debug docker issues
- Video editing
- Donated robot (named sadge) to Kyle (later met some connectivity issues)

Kushal

- Set up and test lane following
- Test duckie detection on laptop
- Contribute to Pub/sub model for training
- Cried at VM for days
- Helped create a physical map (and hence 2 tiles that belong to me will be returned with Kyle's items)