Learning end-to-end control for duckiebot driving

Group: Dropouts

June 2021

1 Group Information

Members:

- Keyan Pishdadian (keyanp@cs.washington.edu)
- Jakub Filipek (balbok@cs.washington.edu)
- Kyle Deeds (kdeeds@cs.washington.edu)

Source code for behavior cloning approach:
`https://github.com/keyan/duckiebot_behavior_cloning`
Source code for reinforcement learning approach:
`https://github.com/balbok0/cse571-sp21-project-2-dropouts`

2 Task Overview

In this project we sought to explore methods for learning an end-to-end control policy that would allow the duckiebot to navigate tracks autonomously. Our goal was to enable the duckiebot and have it navigate around an entire track without any major driving infractions (defined as exiting the lane or going off the road), using only monocular camera data. Ideally this system would be robust enough to generalize to new tracks and work on the real robot as well as in simulation.

Inspired by previous efforts in learning end-to-end control for driving using deep reinforcement learning [**?**] our original plan was to evaluate a selection of reinforcement learning algorithms to solve this task. Knowing that reproducing reinforcement results is difficult and often unsuccessful [**?**], we planned to also experiment with a behavior cloning approach inspired by prior work from NVIDIA where a convolutional neural network was trained to steer a car using only image data [**?**].

3 Environment Setup

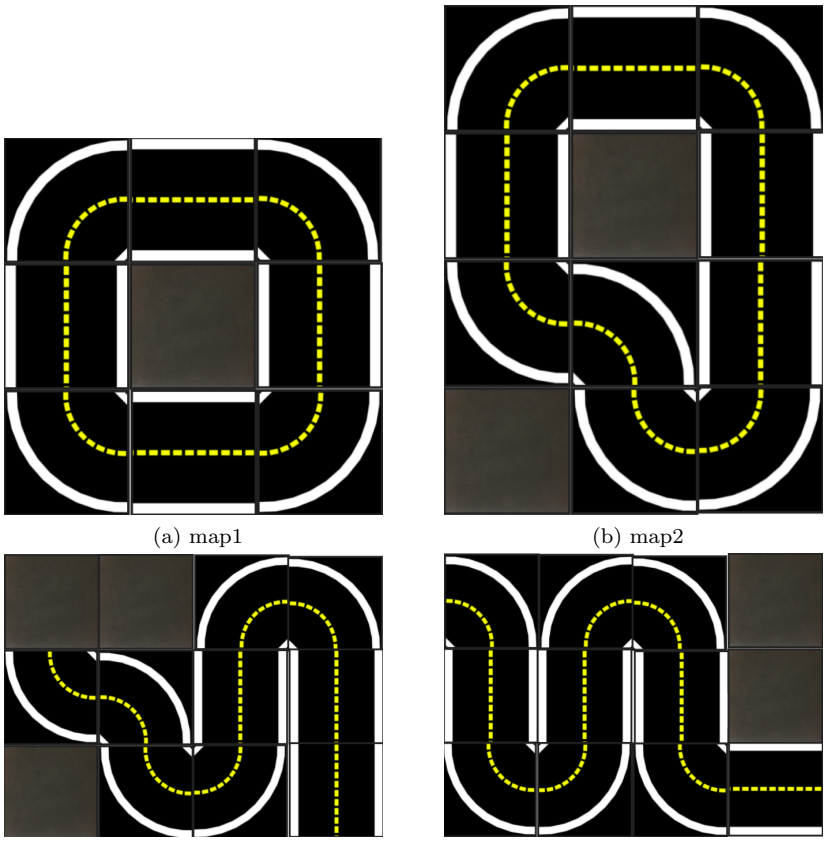
We performed experimentation and evaluation of our agents in simulation and on the real robot primarily relying on the "AI Driving Olympics" infrastructure [**?**]. Our agent was structured into the AIDO submission format and then executed either in simulation or on the real duckiebot. We used the default episode length configuration, which runs agents for 60 seconds. The AIDO submission format requires structuring an agent that obeys the AIDO agent interface and accepts image observations and outputs control commands. We referenced the baseline implementations for both reinforcement learning and behavior cloning submission types, `https://github.com/duckietown/challenge-aido_LF-baseline-RL-sim-pytorch` and `https://github.com/duckietown/challenge-aido_LF-baseline-behavior-cloning`, respectively.

3.1 Simulation

We used the `gym-duckietown` simulator [**?**] for training and evaluation of agents when using a reinforcement learning approach, as well as for evaluation when using behavior cloning (through the AIDO submission infrastructure). For reinforcement learning we only ran evaluation on the most simple map type, "loop_empty", which is an square map with no obstacles. Evaluations for behavior cloning used a specific "challenge", `aido-LF-sim-validation` which runs the agent on a static selection of maps and ranks submissions against other users.

3.2 Real World

Evaluation on the real duckiebot was done on four different maps (1), with a 60 second maximum episode duration. Only map2 was used for real robot data collection.



(a) map1 (b) map2

(c) map3 (d) map4

Figure 1: Digital overview of the four maps used for real-world evaluation.

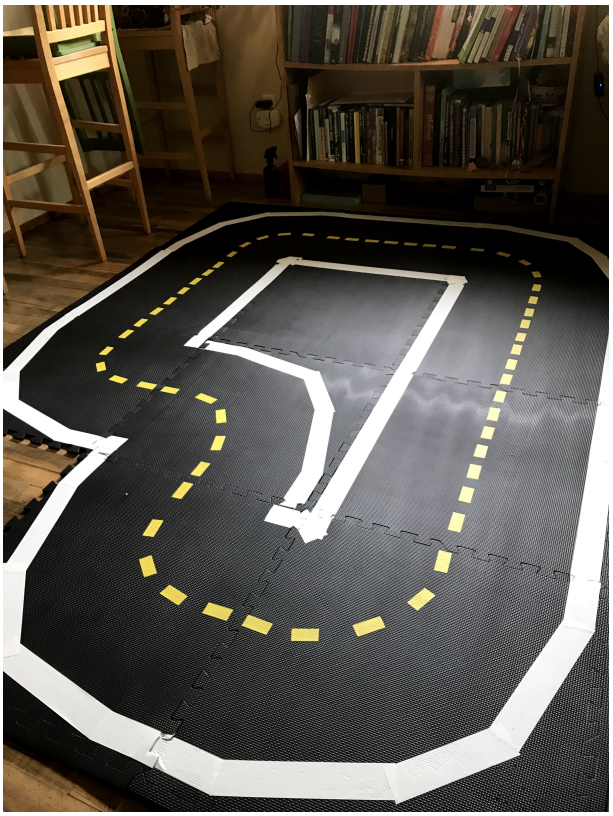


Figure 2: Photo of physical layout of map2.

4 Reinforcement Learning Approach

Our initial solution was to use the `gym-duckietown` environment to train an agent using a model-free off-policy Q-learning algorithm, for which we explored both DQN [**?**] and DDPG [**?**] algorithms. Our training code leveraged pre-written implementations of both algorithms provided through the RLLib package [**?**]. We leveraged the existing default reward function provided by the gym environment (`https://git.io/JGnsg`) which is a linear function of the robot speed, position within the lane relative to the right line, and proximity to obstacles.

[TODO:] Some brief amount of discussion about DQN and DDPG that shows we at least understand it

Unfortunately we were unsuccessful in achieving reasonable training results with this method, with our agent consistently showing an inability to increase average reward obtained despite experimenting with domain randomization in the simulator, training across multiple maps, and performing hyperparameter search (Figure 3). Our initial impression was that training an agent to perform well in simulation would be straightforward and the main challenge would be in transferring the agent to the real environment. When our agents failed to show any promising results during training and subsequently failed to perform in simulation at all, we decided to abandon this approach. Videos of evaluation of these agents in simulation can be found at `https://youtu.be/SDZ59_2zhGg` (DQN agent) and `https://youtu.be/1xL8b2MK6N4` (DDPG agent).

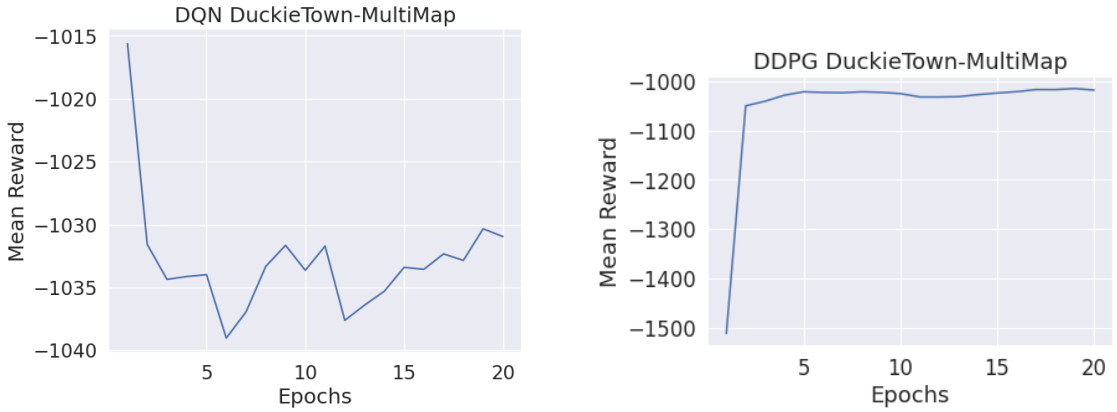


Figure 3: Mean reward obtained per epoch during training of the DQN (a) and DDPG (b) solutions. Both solutions failed to improve mean reward. Here only 20 epochs are shown, but the plateau in reward obtained was observed even when trained for many more epochs.

5 Behavior Cloning Approach

Our focus then shifted towards a strategy where we would train a convolutional neural network (CNN) that would accept a single camera image from the robot at each timestep and then output a prediction for the linear and angular velocity that the robot should apply. This approach was influenced by work from NVIDIA where this technique was applied to predicting a single value representing the steering of the vehicle [**?**]. We designed several CNN models (discussed in Section 5.2) and trained them on a combination of simulation and real world data, then evaluated these models using the AIDO leaderboard and on the real duckiebot. An high-level overview of this system is shown in Figure 4.

5.1 Data Preparation

Data collection from simulation in the `gym-duckietown` environment is easier, faster, and less noisy than real world data, so the overall data strategy was to focus on collecting a large volume of simulated data and augmenting its applicability to real robot evaluation by including a smaller amount of real robot data extracted from ROS bags.

To collect simulated data, a built-in demo was executed in the simulator that uses a pure pursuit controller [**?**] to geometrically determine the optimal robot controls, the resulting image observations and joystick controls were aggregated. Image observations from simulation were downsampled to 150x200 and domain randomization was applied during the simulation runs to help prevent overfitting to the simulation environment.

Real robot data was collected through a combination of accessing publicly available ROS bags [**?**] and ROS bags recorded from demonstrations with one of our own robots on map2 (Figure 1b). When selecting public robot logs, we attempted to find executions which were successfully navigating the map, were free of obstacles, and contained turns. For the custom data robot controls were given by a human driver.

Ultimately we used two final datatsets. The largest (henceforth "dataset A") was used for training the initial model and consisted of 58,414 total observation frames with associated controls, this was a mix of simulated data collected from `gym-duckietown` (50,337 frames) and public real robot data (8,077 frames). The second dataset (henceforth "dataset B") was used for experimenting with fine tuning our models and consisted of 15,560 total frames, this only contained our custom real robot data.

5.2 Model Architectures

Our experiments were focused around two model architectures, the first was an attempt to reproduce results from NVIDIA [**?**] which we named the "nvidia" model (Figure 6a). In order to predict two values (linear and angular velocity) two separate instantiations of this model were created with disjoint weights. The only

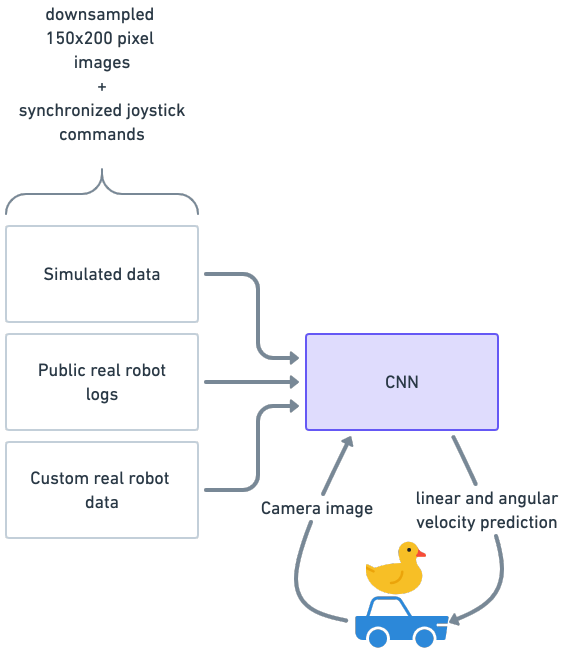
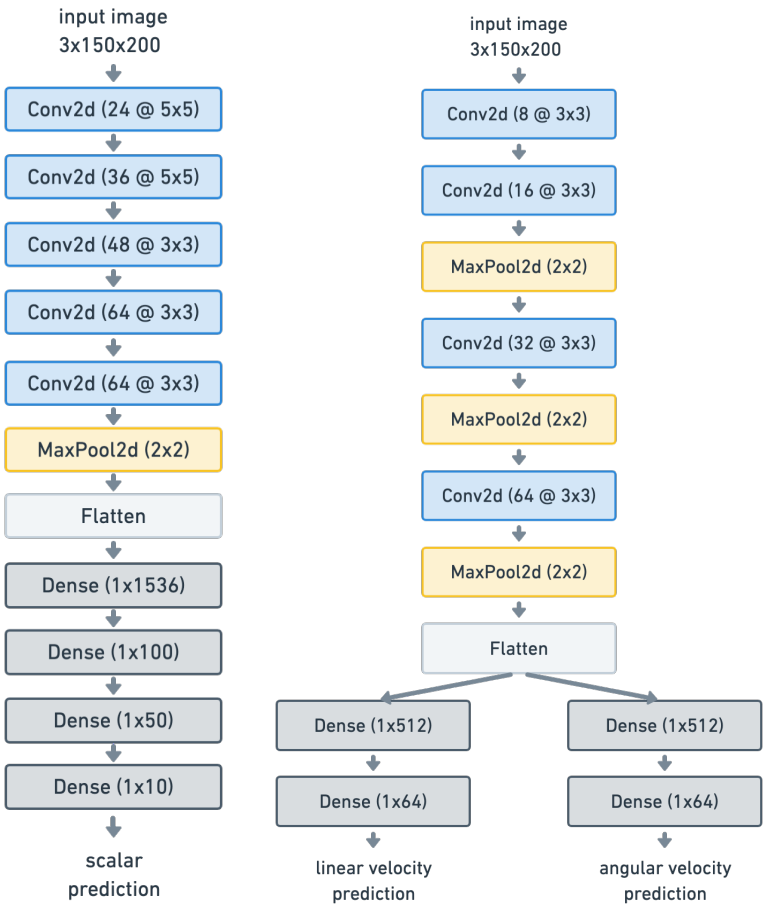


Figure 4: Overview of input training data and data flow during inference for the behavior cloning approach to end-to-end control. At runtime the robot passes each observation frame to the CNN model to produce predicted linear and angular velocities. These velocities are converted to left and right wheel velocity commands which are issued to the robot.

(a) Simulated data (b) Public log data (c) Custom data

Figure 5: Example frames from the three different data sources used for training. Observations are downsampled 150x200 pixel images.



(a) (b)

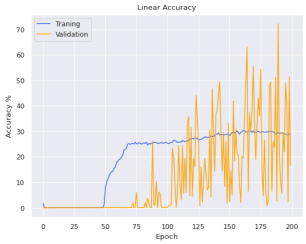
Figure 6: Overview of the two primary model architectures evaluated in this work. (a) features a larger number of total parameters and no shared weights, each of the two prediction values retain their own copy of the model (b) uses shared convolutional layers and small linear layers preceding each of the two outputs.

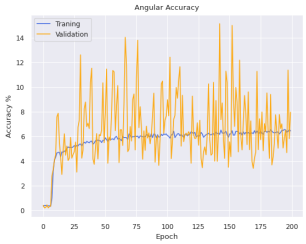
major deviations from the originally published model were the use of leaky rectified linear unit (LeakyReLU) activations and dropout in-between linear layers ($p = 0.5$). The second architecture used a smaller number of total parameters, shared weights across the convolutional layers, and two outputs preceded by small disjoint linear layers (Figure 6b). We named the initial version of this model "modelv0", subsequent experimentation produced a second version "modelv1" which substituted ReLU activations for LeakyReLU, and added dropout to the final linear layers ($p = 0.5$) and the shared convolutional layers ($p = 0.1$).

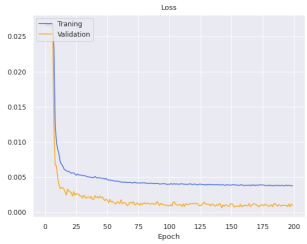
5.3 Training

All training was done using stochastic gradient descent and mean squared error loss. The models were trained initially on the aggregated dataset ("dataset A") for 200 epochs, results in Figure 7. As the "nvidia" model did not show promising results during training, it was excluded from further experimentation. Then "modelv0" and "modelv1" were fine tuned for 50 epochs using the custom real robot dataset ("dataset B"), results in Figure 8.

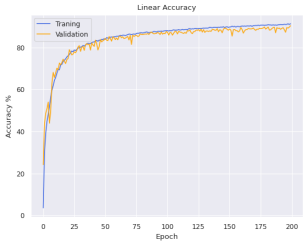
nvidia

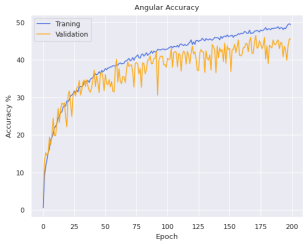


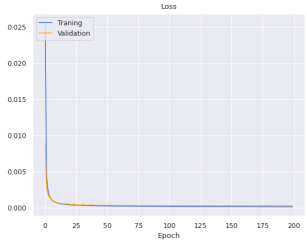




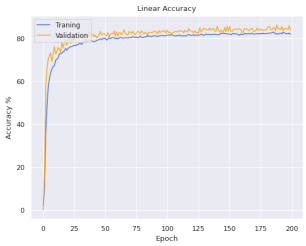
modelv0

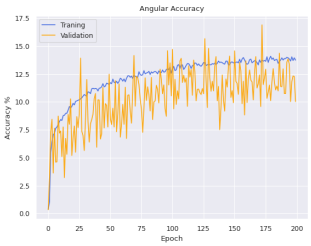






modelv1





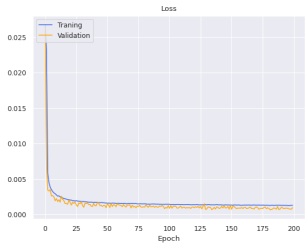
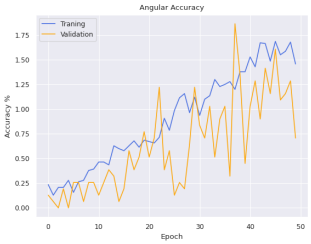
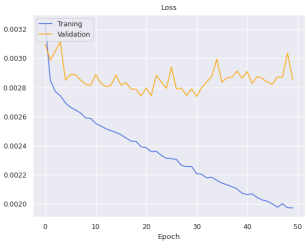


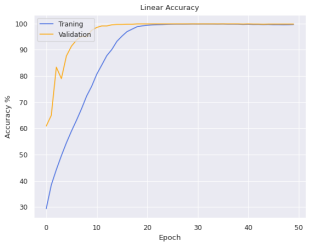
Figure 7: Results from training the three different model architectures on dataset A, showing linear accuracy, angular accuracy, and loss achieved. A prediction was considered correct if the continuous velocity value was within 5% of the label value.

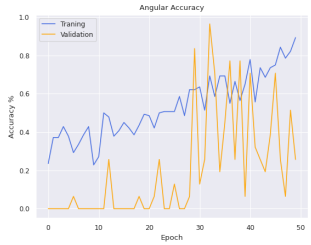
modelv0





modelv1





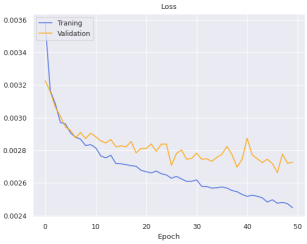


Figure 8: Results from fine tuning the shared weights models on dataset B, showing linear accuracy, angular accuracy, and loss achieved. A prediction was considered correct if the continuous velocity value was within 5% of the label value.

6 Performance Evaluations

6.1 Simulator and AIDO

[**TODO:**] 3x3 figure showing overview map trajectories from AIDO for initial 3 models on 3 maps

[**TODO:**] 2x3 figure showing overview map trajectories from AIDO for modelv0+v1 with fine tuning on 3 maps

[**TODO:**] Share links to videos from simulator and AIDO submissions

[**TODO:**] Discuss quantative performance for each model including baseline

6.2 Real World

[**TODO:**] Discuss quantative performance for each model including baseline

[**TODO:**] Share links to videos for each map/model evaluated

Baselines map2: `https://youtu.be/oNrlfTRxNy0`

Model v1 map2: `https://youtu.be/NMiHz7VIw_4`

7 Potential Future Improvements

[**TODO:**] Discuss algorithmic enhancements

[**TODO:**] Discuss robot limitations