CS5478: Project Report on Visual Autonomous Navigation

Group Members: Lim Sze Chi (A0093985J), Tran Ba Phong (A0228533E)

Model Architecture For our final submission, we used an end-to-end neural network learning approach, specifically a Deep Q-learning agent (DQN). We trained our DQN agent by experience replay memory and used a Convolutional Neural Network (CNN) as a function approximator that maps input observations by Duckiebot to output actions. To enable the agent to learn from experience, we allow it to explore at a high exploration rate at the beginning of training. Hence, the agent takes a random action and stores its transition to a memory buffer for multiple episodes most of the time (an episode is a complete run until *done==True*). This memory buffer stores all the most recent transitions that an agent took up to a maximum length, and contains (state, action) pairs and their corresponding (next_state, reward) result. As the number of episodes increases, the exploration rate decays exponentially as designed and the agent will slowly switch to exploitation and execute actions from the CNN model output. After multiple rounds of experimentation, our final CNN model architecture is illustrated in Figure 1.

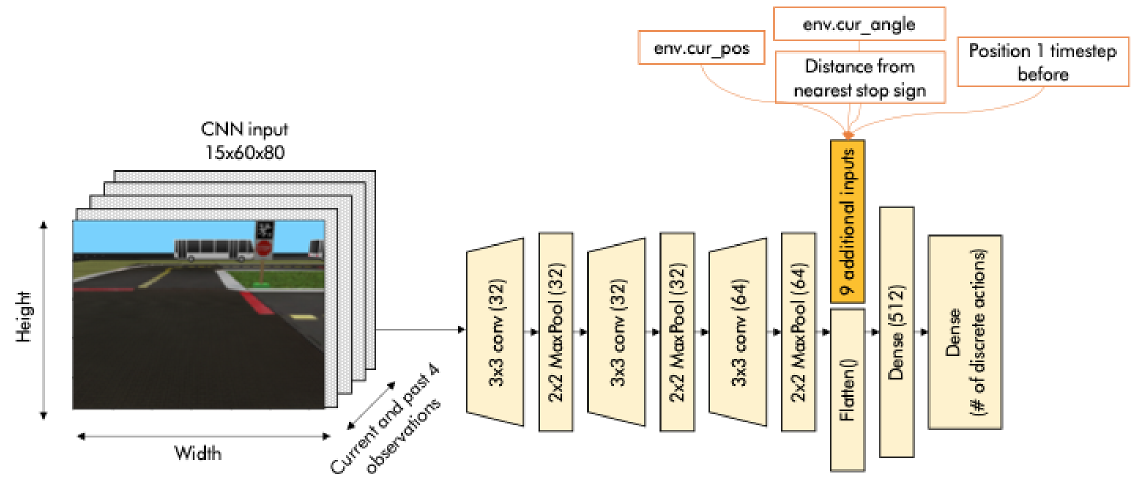
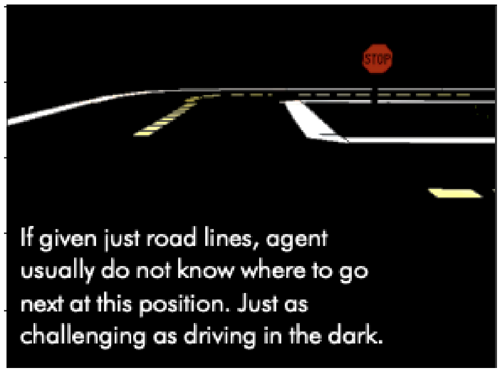


Figure 1: Our DQN agent architecture.

Choice of Input and CNN Taking reference from Almási et. al. [1], we resized our input to 12.5% of the original DuckietownEnv observations, normalized pixel values from 0 to 1 and used a similar CNN architecture. We also used their proposed idea of concatenating past observations to specially guide the agent especially at turns (with the intuition that it may allow the agent to gauge the road curvature and use it to infer its next action), which worked better than a single observation. However, we made a few modifications to fit our problem better.


STOP
If given just road lines, agent usually do not know where to go next at this position. Just as challenging as driving in the dark.

1. The authors have proposed using only feature maps of road lines and that of the past 4 time steps as input features. We attempted the same using OpenCV but also captured stop signs in red (Figure 2). However, the results were not as good as original RGB inputs. Specifically for map3 in the middle of road intersections, the input observations are just a few road lines in the horizon and the agent seems "lost".
2. The authors also proposed cropping away the horizon (usually top ⅓ of the image) but we did not as we would like to capture the stop sign.
3. Lastly, we concatenate 9 additional inputs to the first fully connected layer (a 6-vector of current and previous positions, 2 values from current and previous angle, and a distance to nearest stop sign value). This has allowed the agent to gain full information of its position in the map, and to slow down successfully at stop signs, especially at areas near the stop sign but is a blindspot to the input observations e.g. agent directly below the stop sign.

Figure 2: Road lines and stop sign extracted features using simple thresholding with OpenCV.

Choice of Discrete Actions In DQN, it learns a Q-function and gives a policy that takes an action with maximum Q-value. The max function requires actions to be discrete. Hence, we need to discretize the action space. Discretizing the action space is tricky and we have handled this carefully by giving a large range of reasonable actions to ensure that transition to every best state is possible for the agent i.e. lane following during high and low speeds (near stop signs), and during road bends.

As we do not want the agent to move backwards, velocity is [0,1] and steering angles can span from negative to positive values and usually in terms of **π**, as we realized by experimentation it creates smoother turns at road bends. At first, we experimented with large discretization bins of steering angles. It had no issues when travelling at high speeds and had good lane-following rewards but did not work well near stop signs, where the agent tends to travel in a zig-zag fashion. To give us an idea of how to discretize steering angles at low speeds, we observe the steering angle outputs from a pure-pursuit controller [3] fixed at low speeds. At low speeds, the pure-pursuit controller outputs a huge range of steering angles from 10^{-3} to >1. Learning from that, we discretized steering angles to capture a similar range and observed a much smoother lane-following paths near stop signs. In total, we have given the agent **263** discrete actions for this project. As the number of discrete actions increases, the DQN training becomes more unstable and can diverge (i.e. accumulated rewards decreases and loss increases) but we have coped with this challenge by letting the agent and target DQN networks update at lower frequency (every 500 episodes).

Reward Shaping Our reward shaping is illustrated in Figure 3. The weights in the weighted sum of different components are to scale the components properly so that not one component dominates the other. Our key modifications are: (1) Adding additional penalty when agent's speed is >0.15 when distance from nearest stop sign is <0.35. (2) As we are doing a one step look-back during training, we used distance threshold 0.35 instead of 0.3 so that the agent can slow down on time. (3) Reward function for *`lane following and minimize speed`* is scaled in the form of $\frac{2.4}{1.2+Speed} - 1$. Hence when speed is minimum at 0, this value is 1 and when speed is maximum at 1.2, this value is 0, which achieves the purpose of greater reward at lower speed. Having $\frac{2.4}{1.2+Speed} - 1$ also limits this value from [0, 1], which is a comparable scale to the Speed component in *`reward for lane following and maximize speed`*.

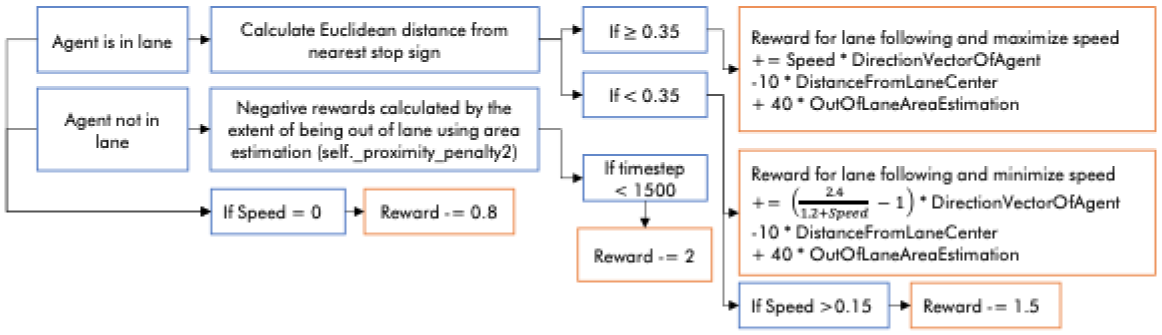

Agent is in lane
Calculate Euclidean distance from nearest stop sign
If ≥ 0.35
If < 0.35
Reward for lane following and maximize speed
+= Speed * DirectionVectorOfAgent
-10 * DistanceFromLaneCenter
+ 40 * OutOfLaneAreaEstimation
Agent not in lane
Negative rewards calculated by the extent of being out of lane using area estimation (self._proximity_penalty2)
If timestep < 1500
Reward -= 2
Reward for lane following and minimize speed
+= (2.4/(1.2+Speed) − 1) * DirectionVectorOfAgent
-10 * DistanceFromLaneCenter
+ 40 * OutOfLaneAreaEstimation
If Speed = 0
Reward -= 0.8
If Speed >0.15
Reward -= 1.5

Figure 3: Our reward shaping.

Why DQN? With a good reward function, DQN can easily converge to find a reasonable policy, and less likely to converge to a local minima. We have tried several networks that can output continuous actions e.g. DDPG and give the same level of exploration as our DQN but the agent ends up in an absorbing state like rotation. Also, DQN explores many states in the map to enable it to learn a good Q-function that can generalize across all start states, so there is no need for one model per seed. Lastly, we leave the DQN to find good policies which surprisingly can make **U-turns** in several maps like in maps 1 and 3.

Experimenting with Continuous Action Space In the beginning stages of the project, we tried algorithms capable of handling continuous actions such as PPO and SAC. This is implemented with stable-baselines3 framework. The reward function and input processing is similar to our use in DQN. In map 1, for action, we used a tanh function to limit velocity [0 to 1] so the agent can go forward. For other maps, we experimented with different action mappings: positive angle, positive velocity, braking (1 - velocity), and wheel velocity [4]. We found out that the default reward function does not incentivize the agent to move forward. We then crafted the new reward based on orientation and lateral error from desired trajectory. In this way the agent can drive in the right lane, however this took long training steps even using GPU - 10 millions steps to see minimal progress. DQN can give us a reasonable policy with fewer training steps, hence the choice to use DQN thereafter.

References

[1] Almási, P., Moni, R., & Gyires-Tóth, B. (2020). Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World. *Proceedings of the International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN48605.2020.9207497
[2] Sazanovich, M., Chaika, K., Krinkin, K., & Shpilman, A. (2020). Imitation Learning Approach for AI Driving Olympics Trained on Real-world and Simulation Data Simultaneously. *ArXiv*.
[3] Pure-pursuit Controller Script. https://github.com/duckietown/challenge-aido_LF-baseline-behavior-cloning

[4] Kalapos, A., Gór, C., Moni, R., & Harmati, I. (2020). Sim-to-real reinforcement learning applied to end-to-end vehicle control. *ArXiv*. https://doi.org/10.1109/ISMCR51255.2020.9263751