# CS5478: Intelligent Robots Project Report - Group 5

**Aishik Pyne**
A0250592E
e0945774@u.nus.edu

**Niharika Shrivastava**
A0254355A
e0954756@u.nus.edu

## 1 System Design

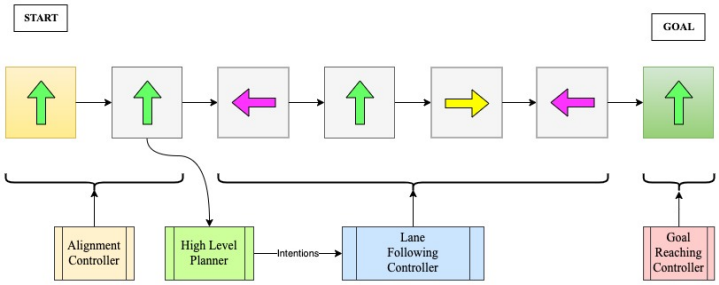


Figure 1: Duckietown System Architecture. The arrows are intentions F, L, R and the x-axis is the tiles visited during the journey

The system design, as seen in Figure 1 consists of 3 controllers - Alignment Controller, Lane Following Controller, Localization module (Goal-Reaching Controller), and 1 high-level planner. For each milestone, for the first 2 tiles, the aligner makes sure the agent reaches the right lane, after which the high-level planner generates a plan for the Lane Following Controller to reach the goal tile. Finally, the Localization module guides the agent toward the restaurant with the duck. In the following sections, we describe and justify our design choices.

## 2 State Encoding

At test time the agent has access to:

1. **Observation**: RBG image from front camera of size $(160 \times 120 \times 3)$.
2. **Intention**: Instruction from High-Level controller of the form **Forward, Left, Right** to guide the Lane following controller.
3. **Intention Time-step**: An artificially created state which counts the time spent by the agent in one tile. At the start of each tile $t = 0$ and at each timestep, $t$ grows by 0.01 till $t = 1$. This acts as a historical information encoding variable.

## 3 Aligner

### 3.1 Analysis of Start tile

After multiple manual simulations, empirical observations were made about the starting position of the agent.

1. The agent can start in either the right lane or the left lane. It might or might not be facing a lane. In extreme cases, it might not even have any information like yellow lanes, white lanes or red lines, such as looking at grass or at a junction.
2. There is a huge penalty when the agent is driving in the left lane. The objective is to move over to the right lane as quickly as possible without going off the road.

### 3.2 Alignment Controller

Since the objective is to merge into the right lane, we need no intention. We use a mix of the provided basic lane follower (AdaComp, ) and our trained LaneFollower (described later). For the first few $t <= 0.05$ timesteps we use the given lane follower because when it is too close to the edge or it is facing the grass, it does tank turns. After this, our trained lane follower is capable to recover into a lane.

## 4 High Level Planner

The High-Level Planner performs global planning to find the shortest path from the 3rd tile to the goal tile using the **A* algorithm**. As the heuristic, the **Manhattan distance** is used, and the cost to come was set to 1 for each neighbour. Note that multiple shortest paths can exist for a particular milestone because of loops in the map.

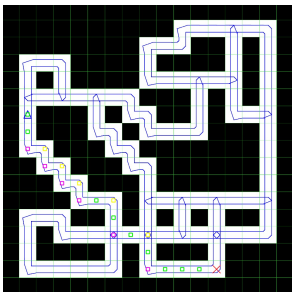

Figure 2: Sample High Level Plan. Triangle = Start. Cross = Goal. Squares are the paths where colours signify intentions

However, we can play to the Lane Following controller's strengths to choose from the set of shortest paths. We have observed that the LaneFollowing controller can move faster and collect more rewards when moving forward compared to turns. Moreover, left turns consistently incur less penalty than taking right turns. Thus while expanding the node we add additional cost based on *Intention* used to expand the node: 1 for **Forward**, 2 for **Left**, and 4 for **Right**

## 5 Lane Follower

The lane follower module in Figure 3 is a data-driven controller. We use Behaviour Cloning to learn a Neural Network that maps Observations and Intentions directly to Actions by mimicking an expert.

### 5.1 Designing an Expert

During training time we have access to the ground truth information from the environment. This can be used to develop a

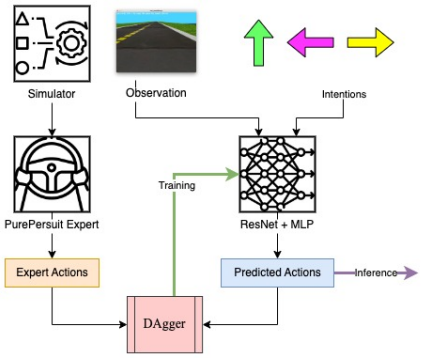


Figure 3: Lane Following Controller using Imitation Learning

model-based controller and use that as an expert. This is convenient because this would not require collecting data manually. In our case, we use a PurePursuit Controller as the expert.

At each time step, the controller calculates the closest point to the right lane and looks ahead on that lane to find a point to pursue. Then it calculates a vector towards that point and the angle of that vector w.r.t the orientation of the agent is used as $\omega$. The velocity $v$ is chosen such that it is inversely correlated to $\omega$. This means when the deviation is high, it should mostly turn instead of moving forward.

During turns, however, the Pure pursuit controller needs to be modified. After empirically trying out multiple magic values, we designed intention-conditioned generic control values which swerve the agent into the intended path. The adaptive control uses the Intention timestep to control the $\omega$.

### 5.2 Neural Net Architecture

The Neural-net first uses a **ResNet-50 pre-trained on ImageNet** data to extract features from the *Observation*. This produced a 2048 long feature vector, which is passed through a **Linear+LeakyReLU layer** to out a 128 sized vector. The *Intention* is one-hot encoded and concatenated with the *Intention Timestep* to form the *Intention tuple* e.g. $[0, 1, 0, 0.140]$. Since the *Intention tuple* is length 4, it is repeated 32 times to make it size 128. Finally, these two feature vectors are concatenated and passed through a final **Linear layer** which produces the regression output $[v, \omega]$. $v$ is passed though **Sigmoid** and $\omega$ through **TanH**.

### 5.3 Training using DAgger

Classical Behavioural cloning sufferers from distribution shift. Due to any errors, like prediction errors, if the agent takes a wrong step and makes an error, it will compound the errors in the subsequent steps. To overcome this we use DAgger (Ross et al., 2011) which during simulation uses a mix of expert and agent prediction to generate trajectories, however when it deviates from the right lane, the control goes to the expert which generates corrective actions.

### 5.4 Inference

Although the learned network should have been able to take turns, it at times gets confused and crashes. However, since it learned how to follow a lane and recover into a lane very well, during inference, if the initial few timesteps during a turn is injected with some additional $\delta\omega$ it can consistently make turns.

## 6 Localization Module

Our goal-reaching controller gets activated once we have reached the high-level goal tile. The duckie building is expected to be somewhere in this tile. In most cases, once we reach the tile, the building is out of direct sight. Through observation of the given test cases, there are 2 main cases:

- **Building directly in front of the agent:** In this case, in order to reach close to the goal, the agent has to move forward towards the building.
- **Building needs to be found after exploring the goal tile:** In this case, the agent explores the tile by rotating in the clockwise direction, detects the building, and moves forward towards it.

In order to generalize the localization module, we make the agent rotate in the clockwise direction till it finds an observation where the duck building can be seen and then moves as close as possible towards it. The duck building is detected using a simple classification model based on Structural Similarity Scores (Nilsson, 2020) which detects the similarity between the current observation and a ground truth dataset containing images of the duck-building in the goal tile.

We collect this dataset by storing the 360° observations of the goal tile using all the maps. The dataset is then manually labelled as (i) containing the duck building, and (ii) not containing the duck building. During test time, once the agent has reached the discrete goal tile, the agent (1) rotates and gets the observation at each turn, (2) checks the SSIM score of the observation with the ground-truth images (containing duck-building) and detects the building if the score is above a certain confidence threshold $\lambda$ (3) moves towards it as close as possible using forward actions.

Since the dataset is small, we achieve a high score of 0.98-1, however, the computation time is slightly high. We can further augment this classifier by replacing it with a more complex function approximation such as a neural network, and perform inference on this model to detect duck buildings.

## 7 Conclusions

The design of the Pure Pursuit model parameters was challenging. This is because we wanted to optimize the speed and the reward which is dependent 10 times more on the distance from the right lane than the angle the bot makes with the track. We also tried goal-conditioned RL using DDPG for lane following however, the reward engineering was difficult. We observed that the agent preferred to turn in one place and collect negative rewards rather than risk going close to the edge which had a massive penalty.

## References

[AdaComp, ] AdaComp. Adacompnus-cs4278-5478-project-basic-lane-follower.

[Nilsson, 2020] Nilsson, J. (2020). Understanding ssim.

[Ross et al., 2011] Ross, S., Gordon, G. J., and Bagnell, J. A. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. (arXiv:1011.0686). arXiv:1011.0686 [cs, stat].