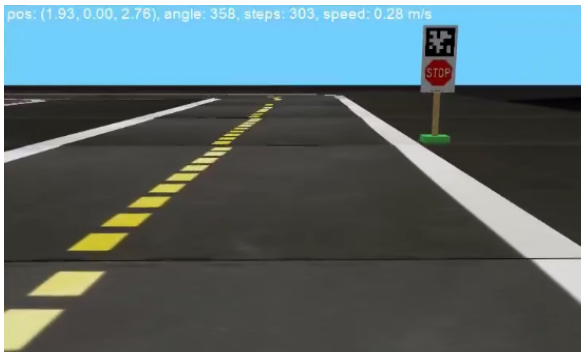
Visual Autonomous Navigation

Complete by **TBA**

1 Overview

In this project, we will develop a vision-based autonomous driving system in a simulated environment, Duckietown. Duckietown is a toy, but comprehensive simulator for autonomous driving. It provides the base platform for AI Driving Olympics, which is hosted at top AI and robotics conferences, including NeurIPS and ICRA. The Duckiebot navigates the Duckietown with *visual inputs* only. It aims to (i) reach a destination as fast as possible (ii) follow traffic rules, and (iii) avoid obstacles and pedestrians along the way. Despite its simplicity, this setup captures the key requirements of autonomous driving.



Here we focus only on the first two objectives, in a simplified environment with no obstacles and pedestrians blocking the way. You will design and implement an autonomous driving system:

- *Input*: images from the Duckiebot's front camera.
- *Output*: actions that drive Duckiebot as fast as possible while obeying the traffic rules.

To obey the traffic rules, there are two key requirements. First, move along the center of the right lane, as Duckietown is located in a country with right-side driving, unlike that in Singapore. The Duckiebot must understand its relative pose with respect to the lane center from camera images. The camera's field of view limits what the Duckiebot can see. On a curved road, it may not see a sufficiently long road segment ahead and must infer the shape of the road and its own pose based on past observations. Second, it must slow down if it is within a certain distance of a stop sign. Again the Duckiebot must recognize the sign from camera images.

We will use the Gym-Duckietown simulator, which replicates the real Duckietown based on the OpenAI Gym environment and Python. Your system will be evaluated in the simulator on a set of given maps. For each map, the algorithm is run for a fixed maximum number of steps and receives a total score according to the , which encourages the Duckiebot to stay

in the center of the right lane, to slow down near a stop sign, and to avoid obstacles. In this project, only the first two requirements are relevant.

You may take any technical approaches for this project. Two examples are briefly described next.

2 Overall System Design

The *classic modular robot system* decomposes the system into components: perception, state estimation, planning, control, etc.. We studied each of these components during the semester. The perception component extracts image features, e.g., lane markers, which allow the robot to estimate the pose and the environment state and to decide on the actions accordingly.

The recent *end-to-end deep learning system* directly maps input images to control actions with a function f_θ represented as a neural network. There are two commonly used learning approaches: imitation learning (IL) and reinforcement learning (RL). IL clones an expert's demonstrations. For training, we first collect a set of expert demonstrations and then minimize the difference between the predicted action using f_θ and the expert demonstration. In reinforcement learning (RL), the robot learns by trial and error. It explores the action space and finds the action sequence that maximizes a reward function in order to achieve the highest score.

These two approaches are different, but not mutually exclusive. For example, you could build a classic feature extraction module and then use the extracted image features as input for RL.

3 Sign Recognition

There are many different approaches to the recognition task. A relatively simple one would be image classification. For this, collect a set of images, label them, and train a classifier to determine whether an input image contains a stop sign or not. This approach, however, does not provide an estimate on the distance to the stop sign. You Duckiebot may slow down unnecessarily.

4 Getting Started

Simulation environment. Clone the GitHub repository, and follow the instructions in README under the directory `gym-duckietown` to install the simulator. It is recommended that you install in a virtual Python environment, rather than the native Python environment on your system. One common way of managing multiple Python environments is Anaconda, which supports all platforms.

Test your installation:

```
$ cd gym-duckietown
$ python manual_control.py --env-name Duckietown-udem1-v0
```

You should be able to control the Duckiebot with the keyboard. Note that you only need to use DuckietownEnv, and specify the map file in the parameters. Gym-Duckietown follows the OpenAI Gym APIs, documented here.

Image feature extraction. One major challenge for classic robot systems is perception. For lane following, you need to detect lanes and estimate the relative pose of the Duckiebot with respect to the lane. You may follow this tutorial for lane detection with OpenCV-Python for "real-world" autonomous driving.

Deep learning. For autonomous driving with deep learning, you may follow this tutorial. Although the observation space in tutorial is different from the Duckietown, the underlying idea remains the same.

Open-source code. You may use open-source code available online. Gym-Duckietown GitHub repository also provides sample IL and RL implementations, though they probably do not perform very well for our task.

5 Submission

You will submit control files along with code, similar to what you have done for the homework assignments.

The GitHub repository currently contains a set of maps, which you may use for development and test. We will release an updated set soon for grading. Do **not** change the environment random seed, as it may affect the starting pose of the robot and thus the robot behavior during the evaluation.