Goal-Directed Visual Navigation (in Simulation)

Instructions

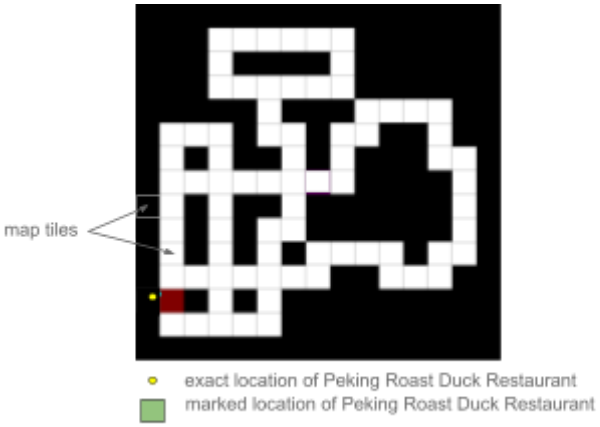
- Completion date *to be confirmed*.
- Each project team consists of 2 students by default.
- After the preamble, this project description consists of 3 main sections.
  - *Understanding the task*. This section helps you think about what you need to do. In real-world engineering, you do not get the exact technical specifications of the task automatically. You have to work them out. We do not ask you to do the actual work here, but hopefully, this section helps you think about the issues likely to be encountered.
  - *Task Specifications*. This section specifies the task formally. You may need to refer to the earlier section in order to digest the details.
  - *System Design*. This section provides some suggestions on the overall system design and the components. You need to follow some of these suggestions closely.
- Submit the code, report, and control files to CANVAS.

Duckietown is a vast city, with a distinguished history. Its greatest secret—the *Peking Roast Duck Restaurant*—is long forgotten. As a young girl, your grandma traded the secret with a Gypsy witch, using three golden coins: "Whoever finds the roast duck will live forever." Being an educated woman, your grandma did not actually believe in immortality, but she always wanted to taste this Peking roast duck as part of her gourmet adventure.

One day, grandma was sleep-talking, and you overheard her exclaiming "Duckie, Peking roast Duckie … I want Duckie". After learning about this legendary Peking Roast Duck Restaurant, you decided to find this treasured place for your beloved grandma. Quickly, you got your gear ready:

- A rugged pickup truck, which you named *duckiebot*.
- A map of Duckietown with the approximate location of the restaurant marked on it.
- Some old photos of Peking Roast Duck Restaurant.
- An inexpensive GPS unit.
- Dr. D's survival guide on motion planning and control. 😁

Duckietown simulator.



Duckietown map.

An image from Duckiebot's forward-looking camera, with the restaurant in view.

A photo of Peking Roast Duck Restaurant.

Understanding the Task

You will build a goal-directed autonomous driving system that controls Duckiebot and enables it to reach its destination. The input to the system consists of (i) Duckiebot's location on the map according to its GPS-based localization unit and (ii) images from the Duckiebot's forward-looking camera. Unfortunately, Duckiebot's GPS unit has poor reception, as the high mountains block the signal. It can only get a rough estimate of the location up to the resolution of map tiles. The driving system outputs the controls so that Duckiebot drives as fast as possible toward the destination while obeying the traffic rules, particularly, following the right lane.

To get started, you call up Dr. D to discuss your plan. Dr. D is very excited about your new system. Below is the conversation between you two.

Get to the Peking Roast Duck Restaurant

Dr. D: This is goal-directed navigation and you have a map, right?

You: Yes, but Peking Roast Duck Restaurant's location is only marked roughly on the map. My GPS position is also inaccurate!
Dr. D: Maybe you want to consider a two-level hierarchical design?
You: Oh, right! Duckiebot uses the Duckietown map to plan a high-level path that goes from one map tile to the next. Then, it uses a motion controller to follow this path. Finally, when Duckiebot is close to Peking Roast Duck Restaurant, it searches the region and uses visual recognition to recognize the famous yellow Duckie mascot and the restaurant!
Dr. D: Sounds excellent! What would your controller do?
You: Hmm … I am not so sure about the details, but it should take in camera images and move the Duckiebot toward the next map tile along the path.
Dr. D: Good.

Reach the Destination as Fast as Possible

You: I need to get back as soon as possible. My grandmother is waiting for me at home.
Dr. D: To reach the goal in the shortest amount of time, we need to optimize both the planning and the motion control levels. For path planning, search for the shortest path on the map, while for path following, we need the controller to optimize for efficiency.
You: Right. For path planning, there are many shortest-paths algorithms: A*, Dijkstra, and Bellman-Ford, …. Which one shall I use?
Dr. D: You want an algorithm that computes the optimal path and does so as efficiently as possible.
You: Ok. How about the path following?
Dr. D: There are various motion controllers, *e.g.,* PID.
You: But we only have camera images as input. We don't have Duckiebot's precise coordinates.
Dr. D: Right. This is indeed a difficult issue. Maybe we can figure out the coordinates from the camera images, or maybe there are other better ways ...

Obey the Traffic Rules

Dr. D: Do you know the traffic rules in Duckietown?
You: Yes. To obey the traffic rules, Duckiebot must move near the center of the right lane, as Duckietown is located in a country with right-side driving, unlike Singapore.
Dr. D: It seems that the Duckiebot must understand its relative pose with respect to the lane center from camera images. The motion controller has lots of work to do.

Localize and Recognize Peking Roast Duck Restaurant

You: Wait a minute. This system does *not* work!
Dr. D: Why so?
You: How does Duckiebot know whether it has reached the goal and should stop?
Dr. D: You have the restaurant location marked on your map?
You: Yes, but that's a rough location. Also, Duckiebot's localization unit doesn't work that well. It can only make out the map tile of the current vehicle location and not the precise coordinates.
Dr. D: Well, getting to that rough location is a great first step, isn't it? But you are right. It looks like you've got another difficulty on hand now. What other information do you have?
You: Uhh… Oh, some images of Peking Roast Duck Restaurant.

Dr. D: Great! We have two sources of information: the map tile containing the coarse location of the restaurant and some images of it. First, we can find a path to reach the designated map tile. Then we search locally in the region and, hopefully, spot the restaurant visually.

Other Things ...

Dr. D: It seems you are in shape. Any other issues?
You: Navigation seems hard, much harder than I had thought. Even simple things, such as following the lane and stopping at the goal, which human takes for granted, seem hard. To do either, Duckiebot needs to know its location. However, the images are complex. More than that, the Duckitbot's GPS unit is not that accurate.
Dr. D: Yes, now you know the difficulty of real-world robotics: *uncertainty*.
You: One more thing. I heard that machine learning is taking over robotics like a storm. Can I try it?
Dr. D: Sure, you should try. I don't know about the "storm" part. Machine learning is a powerful tool. Like all other tools, think carefully about when and where you want to use them.
You: Where do I get data for learning?
Dr. D: Yeah, great question! Lucky you, we've got a Duckietown simulator.
You: I can't wait to get started.
Dr. D: Good luck and do us proud!

Task Specifications

In this project, we will develop a goal-directed autonomous driving system *Duckiebot* in the *Gym-Duckietown* simulator. Gym-Duckietown is an integrated simulator for autonomous driving, based on the OpenAI Gym environment and Python. While simplified and somewhat toyish, Duckietown in fact captures the key requirements of autonomous driving. The simulator is the foundational platform for AI Driving Olympics, hosted at top AI and robotics conferences, including NeurIPS and ICRA.

The specific requirements are listed below:

- Duckiebot navigates in Duckietown and aims to reach the destination as fast as possible.
- Duckiebot has a map of the town with the start location and approximate goal location marked on it.
- Duckiebot must obey the traffic rules, particularly, staying near the center of the right lane. Traffic violations incur penalties. For simplicity, there are no pedestrians or other obstacles along the way.
- At any time, Duckiebot's localization unit reports the map tile in which it is currently located. Its camera provides images showing the local environment in front.

Your autonomous driving system is evaluated in the simulator on a set of test maps. For each map, the system is run for a fixed maximum number of steps and receives a total score accounting for the following:

- The total time taken for Duckiebot's journey.
- Simulation score summarizing Duckiebot's driving performance, *i.e.,* observing the traffic rules.
- The distance between Duckiebot's final location and the ground-truth location of Peking Roast Duck Restaurant.

System Design

Here are some suggestions that you may consider while designing your own system:

- Shortest path planning on the map.
- Lane following for Duckiebot motion control.
- Visual recognition to search for Peking Roast Duck Restaurant.

Shortest Path Planning

You may use any of the shortest-path algorithms covered in class or elsewhere, *e.g.*, Dijkstra, A*, or Hybrid A*. You have already implemented some of them in Lab 1. You may need to discretize and represent the map carefully so that the solution path integrates well with your low-level lane-following motion controller.

Lane Following

Given a path on the map, you need to design or learn a low-level motion controller to follow this path while obeying traffic rules. The controller takes as input the path, Duckiebot's front camera images, and possibly, readings of Duckiebot's localization unit. It outputs motion control commands for Duckiebot.

There are two common approaches. The classic modular robot system decomposes the system into modules: perception, state estimation, planning, control,..., which have all been covered in the class. The perception module extracts image features, *e.g.*, lane markers. The motion control module uses this information to estimate Duckiebot's pose with respect to the lane and choose the actions accordingly.

Alternatively, the recent *end-to-end deep learning system* directly maps input images to control actions with a function 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. First, it collects a set of expert demonstrations and then trains the neural network predictor to minimize the difference between the predicted actions and the expert demonstrations. RL learns by trial and error. It explores the action space and finds actions that maximize a reward function in order to achieve the highest score.

While the two approaches are different, they are not mutually exclusive. For example, you could build a feature extraction module using classic image processing techniques and then use the extracted image features as input for IL or RL.

Peking Roast Duck Restaurant Recognition

Once Duckiebot reaches the map tile containing the goal marked on the map, it may still be some distance away from the actual location of Peking Roast Duck Restaurant. Duckiebot then explores the local region. It tries to spot the restaurant visually and gets as close to the restaurant as possible. There are many different methods for visual recognition. If the visual appearance of the restaurant has easily identifiable features, *e.g.*, a very special color, we may build an image classifier manually through template matching. We could also learn an image classifier from labeled data. For this, you need to collect a set of images of Peking Roast Duck Restaurant in the simulator, label them, and train a classifier to determine whether an input camera image contains Peking Roast Duck Restaurant or not. Neither method, however, provides an estimate of the distance to the restaurant. Duckiebot may still stop at a place too far away from the exact restaurant location.

System Architecture

While many different system architectures are possible, most of them follow a two-level hierarchical design. At the high level, a planning module reasons over a map and plans an abstract path over the map tiles. It does not process sensor data, *e.g.*, visual observations, directly. At the low level, a motion control module takes as input sensor data and the path computed at the high level; it outputs a motion command, *e.g.,* steering angle and speed, for robot execution.

The paper below shows a modern instantiation of this design, as an example. Of course, there are many other possibilities.

W. Gao, D. Hsu, W.S. Lee, S. Shen, and K. Subramanian. Intention-Net: Integrating planning and deep learning for goal-directed autonomous navigation. In *Proc. Conference on Robot Learning*, 2017.[1]

[1] If you are keen on research, you may find additional information about this project here.

Getting Started

Simulation Environment

Clone the GitHub repository and follow the instructions in README to install the simulator. Gym-Duckietown follows the OpenAI Gym APIs, documented here. We recommend that you install in a virtual Python environment, rather than the native Python environment on your computer. One common way of managing multiple Python environments is Anaconda, which supports all common systems.

Lane Detection

Your system design may or may not require explicit lane detection. If it does, this tutorial explains lane detection with OpenCV-Python for "real-world" autonomous driving.

Open-Source Code

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

Submission

To help you manage the project, we divide the project into two milestones, each with its own submission.

Milestone 1 (due date 24 March)

This milestone focuses on Duckiebot's low-level controller, which follows a given high-level path (go-forward, turn-left, turn-right, …) to navigate from a start tile to a goal tile. Duckiebot aims to reach the goal tile containing Peking Roast Duck Restaurant, not necessarily the restaurant exactly.

The submission consists of two parts:

- Duckiebot control files
- Code

Organize your submission folder strictly according to the following structure:

```
student_id.zip
|-- code
|-- control_files
    |-- map4_0_seed2_start_1,13_goal_3,3.txt
    |-- …
```

Milestone 2 (due date TBA)

In this submission, complete the high-level planning and other modules. Connect them with the controller to achieve fully autonomous navigation. You will **not** be given the high-level paths as in the previous submission.

We will provide a very basic low-level controller in case your own does not work at all. If you choose to use the provided low-level controller, you must declare so in your report.

The final submission consists of three parts:

- Duckiebot control files,
- code,
- a short report.

Organize your submission folder strictly according to the following structure:

```
student_id.zip
|-- code
|-- control_files
    |-- map4_0_seed2_start_1,13_goal_3,3.txt
    |-- ...
```

Control Files

In each submission, there are 25 test cases in the Github repository. Generate one control file for each test case. Make sure that you do **not** change the random seeds specified in the test data files or reset simulator states during navigation, as this would affect the robot's behavior during evaluation.

See `example.py` for the control file format. For each test case, name your control file according to the following format:

```
{map_idx}_seed{seed}_start_{start_tile[0]},{start_tile[1]}_goal_{goal[0]},{goal[1]}.txt
```

for example, `map4_0_seed2_start_1,13_goal_3,3.txt`.

Code

Submit the exact source code that you use to generate the control files. It may be checked for consistency.

Report

Provide a short report (maximum 2 pages, Times Roman 10 points) that

- describes your overall approach and
- highlights interesting ideas and techniques that you have used in the project.

If you follow the classic modular system design approach, provide a system diagram and describe

- how the system processes the visual input,
- how it determines the position with respect to the lane,
- how it controls the vehicle to follow the lane,
- how it performs global navigate
- how it integrates global navigate with local control

If you use an end-to-end neural network learning approach, provide the network architecture diagram. Describe your architecture choices, algorithms, modified reward functions, and modules, if any.

If you use external code in your implementation, make sure to cite the sources in the report. Otherwise, this will be taken as plagiarism.

Grading Criteria

Your project is graded primarily based on the system performance on the test maps:

- *V*: Follow the lane. *V* indicates driving performance per unit step. It is Duckietown simulator's score divided by the number of steps executed.
- *D*: Navigate to the goal tile as fast as possible. *D* is the success-weighted path length (SPL)
- *T*: Approach the exact location of Peking Roast Duck Restaurant as closely as possible.

The report helps us to understand your approach and when needed, moderate unreasonably raw performance scores. In other words, moderation, if applied, will increase your final marks.

Only *V* and *T* are used for milestone 1. All are used for milestone 2.