COMP 417 – Assignment 4

Mapping and RL/Control

Out: Nov 22nd, 2020
Due: Dec 3rd, 2020. 6pm.

Complete the following two questions following the instructions below.
Time target: spend no more than 3 hours on Q1 and 4 hours on Q2. If you reach these limits and have no "good" solution, write up a one-pager on what you learned and move on to other deadlines.

Obtain the starter code by running:
$ git clone http://github.com/dmeger/COMP417_Fall2020.git
(or run git pull from the folder you worked in for A2)

Submit all your work in the Assignment 4 folder on My Courses.

Question 1: Occupancy Grid Mapping (20 points)

In this exercise you are going to implement parts of the occupancy grid mapping system discussed in class. In particular, you are going to map an environment based on known odometry estimates and known 2D laser scans. The functionality that you need to implement is marked using to-do comments in the file: **estimation_question/build_occ_map.py**

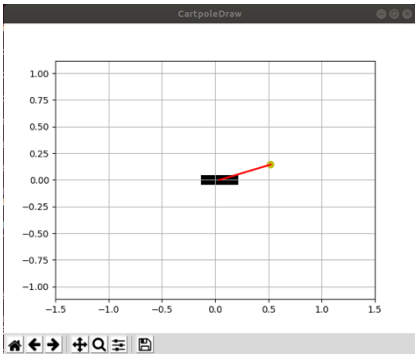
This code reads pre-recorded odometry and lidar scans from a file and processes these one at a time (although currently most functionality is missing). When you complete the TODOs, you should see debugging images of the map saved every 25 lidar scans, so you do not have to wait until the end to know if things are going well. The sequence of images should look as follows:

Hand-in your completed **build_occ_map.py** file and the best final image saved by your code at the end of the best run you achieved. It will be called **final_map.png**.

2 Control of a Cart Pole Robot (30 marks)

For this question, you will choose a control method to balance the famous "cart pole" robot, which has a "cart" that can move in 1D and a freely swinging pendulum (no motor at the link) swinging attached. The ideal controller for this robot can "swing up" the pendulum from any starting configuration and balance it directly upwards without oscillation, while also keeping the cart centred near 0 on the x-axis.

You can do this in several ways and can take on either simple "balancing" or full "swing up and balance". For this question we want you to explore a method and report on your findings – it's not just about succeeding at the task, but you must describe what approach you selected, why, and the details in a short write-up, to be graded along with your solution code.



We've given you the cart-pole simulator in **lqr_question/cartpole_control.py**

You must pick one of these three options to work with the cart pole:

1. Implement and tune PID for balancing. Code the controller code and tune the gains (which are now in several dimensions each, unlike for A2). Because we already saw PID once in class, solutions based only on this are capped at 75% credit for the question (but it can be fast to do – manage your time as you see fit). An example of a good solution is here: [on Youtube].
2. Implement and tune LQR for balancing. We have given you starter code in the folder called "lqr_starter.py".

   The "work" is to solve for the A, B, Q and R matrices for the cartpole. To assist with this, below we have an appendix giving some help on the dynamics equations that you can work with. With these matrices, you'll call the provided helper function, and apply the K that you receive from the LQR solver to the state of the robot within **policyfn**, to compute the control as: $u = K(x - g)$. Run the code to see the cartpole balancing (hopefully… note tuning is often required to make it work well).

3. Find, try-out, and describe the results of code online that implements trajectory optimization, model-based or model-free Reinforcement Learning. Read up on that code (not necessary to understand every detail but do find out how it fits with what we learned in lecture), then explore how to run it on the cartpole. If packages can already run more easily on another cartpole simulation, such as the one in OpenAI's Gym or DeepMind's Control Suite, you can perform you experiments on those targets instead, but you must comment on how their dynamics differ from the provided code. Here's an example of a swing-up solution using DDP/iLQR that a student coded in a previous year [On Youtube].

What to submit for this question?

- A 1-page report in PDF form that describes your investigations and progress:
    - State the key aspects you understood would be required in a solution using the technique chosen. Include things you achieved/discovered, as well as any outstanding items you understood but didn't have time to try.
    - Describe the practical outcomes you achieved. This can include software engineering – what was involved to get the code working? Mostly focus on robotics – what math was done, what did you plot, print or analyze to consider how to make this work?
    - State how much time you invested and break it down into what steps were involved.
- A video of your best cartpole behavior named "**cartpole_reponse_Q2.<ext>**" where <ext> can be ogg, mp4, avi or any other standard movie format. You will hand in your cartpole_learn.py as well as this video result.
- Any code, plots and data files used.

Appendix – Cartpole Dynamics Explained

The state vector of this cart pole is:

$$x = [x \quad \dot{x} \quad \dot{\theta} \quad \theta]^T$$

Where x is the position of the cart along the x axis, in metres, and theta is the angle of the pole (from the zero point: "downwards") in radians. Both velocity components (with a dot above) are in units per second.

The control for this system is simply a force applied to the cart (black rectangle), and otherwise the robot's dynamics are determined by a gravity force that pulls the pole downwards, a drag that opposes cart velocity and the mechanical coupling at the single joint.
We can write these dynamics equations on two lines after integrating the dynamics Lagrangian. You don't need to understand this derivation (we haven't taught you in 417), but feel free to investigate it if interested. For the assignment, you can just trust us that these 2 equations capture the dynamics that the simulator is using to model the cart pole robot:

$$\ddot{x} = \frac{2ml\dot{\theta}^2 \sin\theta + 3mg\sin\theta\cos\theta + 4(u - b\dot{x})}{4(M+m) - 3m\cos^2\theta}$$

$$\ddot{\theta} = \frac{-3(ml\dot{\theta}^2 \sin\theta\cos\theta + 2((M+m)g\sin\theta + (u - b\dot{x})\cos\theta))}{l(4(M+m) - 3m\cos^2\theta)}$$

The relevant constants for our simulation are:

- Pole length, l=0.5
- Pole mass, m=0.5
- Cart mass, M=0.5
- Friction constant, b=0.1
- Force due to gravity, g=9.82

In order to apply LQR, use pen and paper to express this system as a linear approximation, $\dot{x} = f(x, u) = Ax + Bu$ around the goal (the upwards balanced point). In the robot's state space, this goal is $x = [0 \quad 0 \quad 0 \quad \pi]^T$. Intuitively, this describes that the pole is upright ($\theta = \pi$) and the cart is at the centre, x=0, with zero for both velocities.

A linear approximation is achieved by taking a Taylor Expansion to first order. That means taking derivatives of the non-linear expressions with respect to each variable. The resulting A and B are called Jacobians, computed like this:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

For the Q and R matrices for the quadratic cost function, these are obtained by hand tuning in a similar style to the PID gains. There is no one best answer, and tuning values should be a part of your write-up.