COMP 417, Fall 2020

Assignment 2: Path Planning and PID Control (30 points)

Out: Oct 20, 2020

Due: Nov 2, 2020 – 6pm

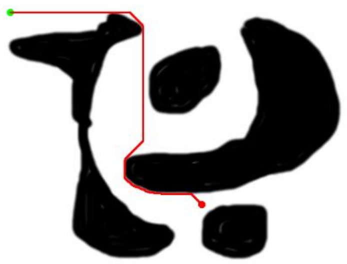
Step 0: Setup a coding environment. We have tested the code with Python 3.7 in miniconda3 and Python 2.7 on the mimi.cs.mcgill.ca server. I think both should work.

A) Obtain the sample code by running:
   $ git clone http://github.com/dmeger/COMP417_Fall2020.git
B) If you're working on your own computer, install these Python packages (pip install hopefully works)
   a. Pillow
   b. Python-opencv
   c. Vispy
   d. Numpy
   e. Matplotlib
C) If you plan to work on a remote server, like mimi, ensure you can open GUI image windows from your terminal:
   a. Mac: XQuartz recommended
   b. Windows: Putty+XMing recommended
   c. Linux: built-in X servers usually work well, nothing to install.
   d. In all cases, the commands "xhost +", connecting with "ssh -XC …" and "export DISPLAY=:0" are typically what's needed. Do some reading for your own system.
   e. Test all of the above by opening the sample map: "eog planning_questions/simple.png" on the . If the map opens as a window, you *should* be set.
   f. In case you cannot make this part work, there are ways to do some of the questions without live feedback, but it won't be quite as good a learning experience.

Question #1: Implement A* Planning (10 points)

We have provided you Python code, called “astar_planner.py” in the “planning_question” folder. That loads a 2D map in the form of a floorplan image. Given a start and a goal, this code now runs Dijkstra’s Algorithm using a simple implementation and finds a plan like the one shown below.

Your job in this question is to modify the algorithm to correctly run the A-star planner. We have left comments “#TODO” in each place where a modification is needed. You should read and understand all of the code (there’s not too much) but will only need to type a very few lines correctly to complete the question (so don’t over-think it!)



Steps to set up:

Run the sample code by running:

```
$ cd COMP417_Fall2020/A2/planning_questions
$ python astar_planner.py simple.png
```

Coding your new planner:

Now modify the code in the places that say #TODO. This is where you’ll need to do most of your thinking, make sure you understood the algorithm presented in class and dig into the code to figure out how to implement your changes. Run it with the same commands as above to test.

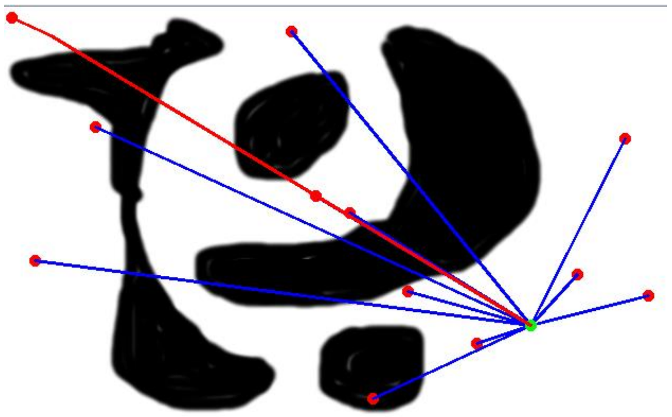
Report the results:

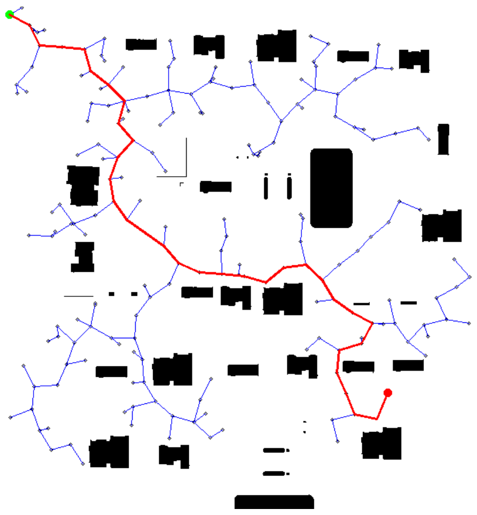
Save 3 images of paths produced by your planner. Keep the starting state the same, but set a different goal for each image (notice the dest_state coded into the main function). Make sure those cover the map well. Save the images as “astar_result[0|1|2]_yourname.png”. You will hand-in these images plus your modified code (just the astart_planner.py file).

Question #2: Implement RRT Planning (10 points)

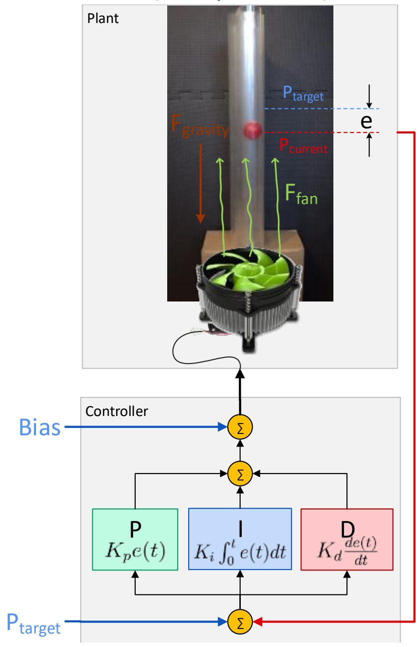
In this case, we have given you the framework for an RRT planner, in "rrt_planner.py" which initially produces the strange star-shaped plan on the left, which is not correct and hits obstacles. You must complete the implementation including steering to new nodes and the overall algorithm structure, to build a correct RRT tree and find the plan. When completed, your code should show an image like the one to the right.

Setup, coding and reporting follow the same pattern as described for Q1 (only work with the rrt_planner.py code this time and name your files rrt_result[0|1|2]_yourname.png).





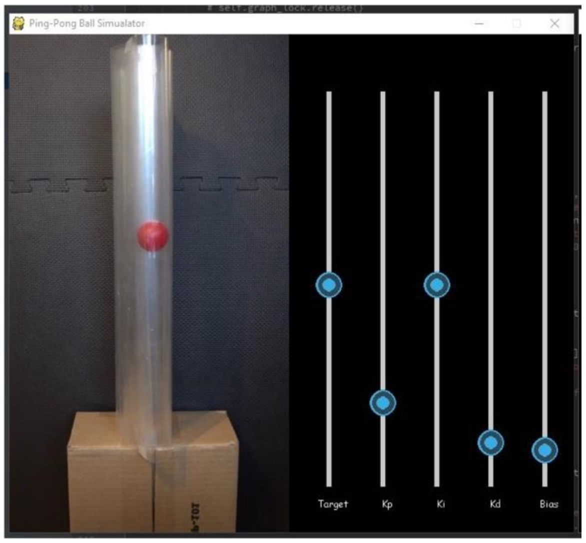
Question #3: PID for a Ball Controlled by Spring and Fan (10 points)



The goal of this question is to understand the PID controller in the context of balancing a ping-pong ball using a fan as shown to the right. The main tasks will be to code the PID control logic, and then perform tuning of your controller for several cases. Inspect the Python code in the **pid_question** folder.

The simulator is run using the command: **$ python main.py**. Afterwards, you can type 'e' to run the simulator in experimental mode (normal mode of operation), or 'v' [target height] (ie 'v 0.5') for evaluation mode where the program will run a step response for several seconds and then show a plot with the results. The validation mode will be used to mark your assignment.

A window should appear as shown. The window is the state of a simulator that computes the red ball's reaction to a fan blowing upwards. The four sliders adjust the target height, Kp ,Ki, Kd gains and bias (constant offset) parameters of the pid controller, but note only the bias term will do anything now (until you code the controller). You can manually control the output of the fan by pressing the left mouse button inside the image area and slide the cursor up and down to adjust the fan rpm.



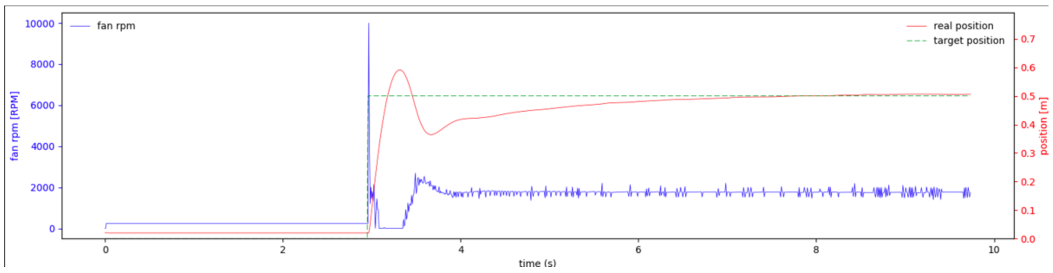
Some additional commands are:
•'r' - reset the simulation, the target height is set to zero and the ball is reinitialized to the starting position.
•'g' - show a plot of the target height, fan output, detected ball position, and real output.
•numbers '0' through '9' - sets the target height of the ball. This is useful to simulate the step response of the PID controller.

The graph that appears with the "g" command is a real-time graph of the state of the system (Note that currently, this live graph does not appear on macOS due to a threading limitation by the OS, however it is not necessary for the assignment. The 'v' mode of the simulator still works to pull up a snapshot graph for marking, even on Mac). You can manipulate the graph using the mouse buttons and scroll wheel and the keyboard commands are:
•'r' - reset the graph to show whole plot.
•'s' - auto-scroll

Step 1) Code the PID controller

In the file **pid_question/pid.py** you will find the implementation of a PID control class. Note that the "self." variables in that class for the bias and PID gains are set by our code automatically as you interact with the sliders on the GUI. The function **get_fan_rpm** is supposed to implement the controller, but in the starter code, it only outputs the setting of the "bias" slider – as in, it doesn't really do any control. You must fill in the right logic to get the ball moving automatically to the target. Implement the proportional, integral and derivative contributions as discussed in class. You can add new global or member variables or change that file in any way that you like, except do not change the names of the existing member variables of the class, as that would break the GUI interface. When you're finished, run the code and move the sliders to see that things work as you expect. You will hand in your **pid.py** file.

Step 2) Tune the PID gains and report performance. It is possible to find PID gains that make the ball respond quickly, but without overshoot or oscillation to target changes such as 0->0.5. Using methods in class, or those you discover yourself, experiment with gains until you have a responsive and smooth control performance without excessive oscillation or overshoot. Can you get a better control curve than this? (remember to see this type of graph by pressing 'g' on the simulator or running initially in 'v' mode.)



Save a screenshot of your graphs resulting from the best gains you can find and save as "**PID_response_Q3.png"**. Hand in the image plus the code in your pid.py file.