E205 - Lab 5

Path tracking

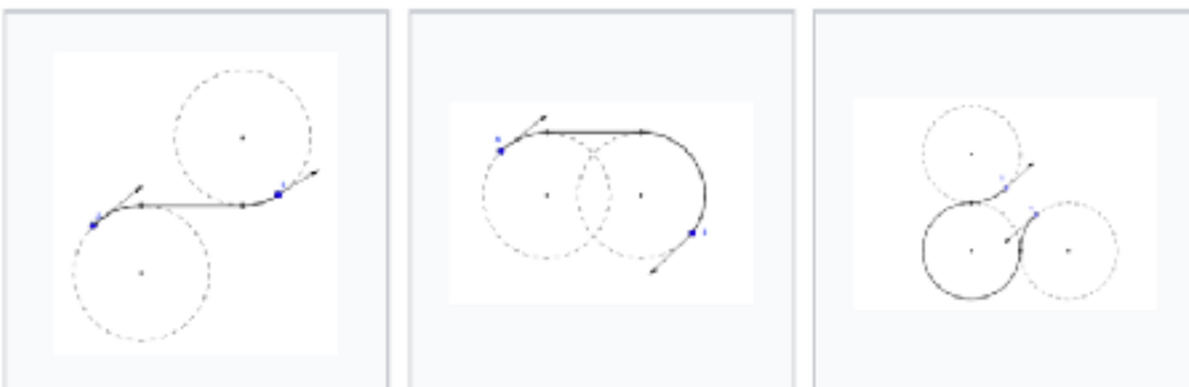
Due: 9:30am Thursday Nov. 3, 2022

---

1. **Dubins Curve**

Install the dubins package (via `pip install -r requirements.txt` or PyCharm). If install fails due to a missing Python.h file, you will need to install python3.8 (it doesn't work on python3.9).

Now, open up e206_lab5_01.py which contains code for this section.
Write a trajectory generator hat uses the dubins library to construct a trajectory from some initial state to some final state. Dubins paths are composed of arc-segment-arc sequences. See the image below.



An RSL Dubins path An RSR Dubins path An LRL Dubins path

Dubins path examples (see https://en.wikipedia.org/wiki/Dubins_path).

Note that the python dubins library only constructs paths, i.e. from some pose [x0, y0, theta0] to another pose [x1, y1, theta1]. You will need to apply time stamps to the poses to create a trajectory - a list of states [time, x, y, theta].

Write your trajectory generator in the function titled construct_dubins_traj(traj_point_0, traj_point_1) that should be near the top of the file traj_planner_utils.py.

You will need to apply time stamps to each pose outputted from the dubins traj. We suggest you assume the trajectory has a constant velocity.
Note that in the main function of the file e206_lab5_01.py calls the construct_dubins_traj() function to create a traj for tracking and plotting.
Add the plots from e206_lab5_01.py in your writeup.

2. **Point Tracking**

We will start by coding up the point tracking controller from lecture 2A. Since the point tracking control law outputs forward velocity *v* and rotational velocity *w*, you will have to convert them to right and left wheel velocities using the equations from slides 48 and 49 of E205 lecture 2A.

Another modification to the point tracker in lecture will be add in a desired theta. You must adapt the calculate of beta to do this:
FORWARD: beta = regular beta calculation + theta_des
BACKWARD: beta = regular beta calculation - theta_des

Your code should be added to the function point_tracking_control() in file traj_tracker_02.py. The inputs to this function should be the robot's state [time, x, y, theta] and the output should be the control signals, i.e. in this case the right and left wheel velocities which we assume are proportional to the right and left wheel torques. Also, you can change k_p, k_a, and k_b based on where the robot is in relation to the desired state. For example, when the robot is far from the desired state, prioritize k_p and k_a and when the robot is near the desired state, change the values.

To enable the code to stop once a point is tracked, you should edit the get_traj_point_to_track() function to set the variable self.traj_tracked to be true if and when the robots state is within tolerance of the point being tracked. Feel free to use the variables MIN_DIST_TO_POINT and MIN_ANG_TO_POINT (defined at the top of the file) as tolerance values. You can change them if you like.

Add the plots from e206_lab5_02.py in your writeup.

3. **Trajectory Tracking**

Write a trajectory tracker that uses the first approach presented in lecture, (trajectory tracking via point tracking). Copy/modify the point_tracking_control from part 2 and implement the tracker in the function get_traj_point_to_track() of file traj_tracker_03.py. Make sure you understand how this function is used to to pick the desired trajectory point on the trajectory to track at every time step, (i.e. see the while loop in the main() function of e206_lab_03.py).

Add the plots from e206_lab5_03.py in your writeup.

3. Deliverables

A lab report should be submitted by 9:30am Tuesday, November 3rd on Gradescope under Lab 5. You NEED to use the IEEE conference paper template, (similar to E80), but the report should be no longer than 3-4 pages. The report should include details of your controller design, plots from the different files, and a discussion on the performance.