University of Vermont
Autonomy
—
HW 6

Assigned: Thursday Oct 24, 2024
Due: Tuesday Nov 5, 2024 at 11:59pm.

This HW assignment will be graded out of 12 points. START EARLY!

Grading: Each problem will be graded out of 6 points. For the first problem, 3 points for the correct code and 3 points for the figures/discussion. For the second problem, 3 points for the correct code and 3 points for the answers.

1. **Trajectory following with spline interpolation.** In this problem, you will implement spline interpolation for trajectory following and compare it against linear interpolation. Recall that for trajectory following, we need to evaluate the interpolated path at every timestep, which we will use as input to the GTG logic, but we also need to find the rate of change of the path, which we will use as a feedforward term in the GTG logic.

   Perform the following steps:

   (a) Download the Python files on trajectory following from Brightspace (under Source Code -> Differential drive trajectory following Oct 15). Set up your Python environment by downloading and installing the correct packages, as indicated in `requirements.txt`. In this assignment, you will modify the `main.py` file. Do not modify any other files.

   (b) Run `main.py` as is and hit the spacebar to start the sim. Once the robot reaches the end of the trajectory, stop the sim by hitting the spacebar again. Take a screenshot of the game window and save it to the PDF file of your solution. Click on Save to CSV, and change the filename from `data.csv` to `data_linear.csv`.

   (c) Now open `main.py` again. Below the definitions of `t_waypoints`, `x_waypoints`, and `y_waypoints`, create two cubic spline objects for x-waypoints and y-waypoints using `scipy.interpolate.CubicSpline`. Also create two cubic spline derivative objects for the x-waypoints and y-waypoints using the `derivative` function of the aforementioned objects. Look up the Python documentation for details.

   (d) Inside the function `trajectory(t)`, you will see an if/else statement. Modify the else block to replace linear interpolation with cubic spline interpolation. The goal is to evaluate the cubic spline and its derivative to obtain `x`, `y`, `ux`, and `y`.

   (e) Now run `main.py` again and hit the spacebar to start the sim. The path should be smoother. Once the robot reaches the end of the trajectory, stop the sim by hitting the spacebar. Take a screenshot of the game window and save it to the PDF file of your solution. Click on Save to CSV, and change the filename from `data.csv` to `data_spline.csv`.

   (f) Open `data_linear.csv` and `data_spline.csv` using Matlab, Python, or Excel. Create two figures: the first figure should plot the left motor commands over time, with two curves, one for linear interpolation and one for cubic spline interpolation. The second figure should plot the right motor commands in the same way. Based on these plots, determine which interpolation method causes more abrupt motor commands, leading to jerkier movements of the robot.

   The PDF file of your solution should contain the two screenshots, the two figures, and a discussion on which interpolation leads to jerkier response. Please also include your `main.py`.

2. **Pure Pursuit algorithm for path following with the differential drive ground robot.** In this problem, you will implement the pure pursuit algorithm for path following, as introduced in class. The goal is to get the differential drive robot is to track a path comprised of predefined waypoints (or nodes) at maximum speed. The pure pursuit algorithm works as follows: at each run of the control loop, the algorithm finds the point on the path that is closest to the robot's current location. It then

advances this point forward along the path by a fixed distance and executes a standard "Go-to-Goal" maneuver to go towards this point.

A path with N waypoints can be compactly represented using a $2 \times N$ matrix, where each column represents the x, y coordinates of a waypoint. Let the matrix be stored in a variable named `path`. The starting point of the path is then `path[:,0]` (Python notation), and the endpoint of the path is `path[:, N-1]`. We assume that the path is directed, is linear in between the waypoints (linear interpolation), and there are no self-intersections or loops.

Follow the following steps:

(a) Download the Python files attached to this homework assignment from Brightspace and save them to the same folder. Set up your Python environment by downloading and installing the correct packages, as indicated in `requirements.txt`. In this assignment, you will modify the `update()` function of your controller located linside `controller.py`. Do not modify any other files. As before, `main.py` is the main file that you will run eventually to test our your code. Notice that, as compared to the previous versions of the differential drive code, the controller's `update()` function now accepts the variable `path` as an argument.

(b) Inside the `update()` function of the controller, you will find an empty function called `getNormalPoint`. This function is intended to take as input a point P and a line segment with endpoints A and B, and return as output a point on the line segment that is closest to P. Specifically, the function should return the "normal point" (i.e., perpendicular projection) if it is located on the line segment or return one of the endpoint (the one closest to P) otherwise. Using the above requirements, complete this function.

Hints: you may use the dot product from geometry. See Fig. 1. The vectors a and b in the figure correspond to $P - A$ and $B - A$ in our problem. The location of the normal point is therefore $A + \frac{a \cdot b}{\|b\|^2} b$, where $\|.\|$ is the vector magnitude and the dot product is: $a \cdot b = a^T b$.

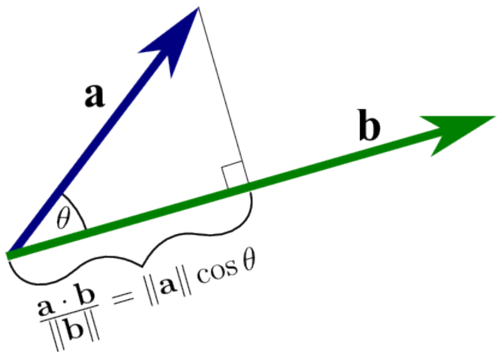
a
b
θ
$\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{b}\|} = \|\mathbf{a}\| \cos \theta$

Figure 1: Problem 3a.

(c) Inside the `update()` function of the controller, you will find another function called `findTarget`. This function takes as arguments a matrix (`path`) and a point P, and returns as output the target look-ahead point on the path. Here, `path` is a given $2 \times N$ matrix representing the path and P is a given 2×1 point representing the location of the robot. Complete this function using the following pseudocode: first use the function in part (b) to find the points closest to P on every line segment comprising the path. Then, among the $N - 1$ points that are computed, choose the one closest to P (use the "norm" function to calculate distances). Store this point in a variable named Q. Finally, compute the `target` (i.e., the output of the function) to be the point on the path that is 2.5m ahead of Q (towards the endpoint of the path). If `target` is beyond the final endpoint of the path, then the target must be the endpoint. Note that this requires some thinking so grab a pen and paper! Note also that the 2.5m advancement will ensure that the robot will move with maximum velocity (clipped at 2m/s) when executing the Go-to-Goal maneuver with gain $K_e = 1$. For an illustration of these ideas, see Fig. 2, where the red line is the path, the red circles are the waypoints, the blue square is the point Q as computed by the algorithm, and the red dot is the target to seek.

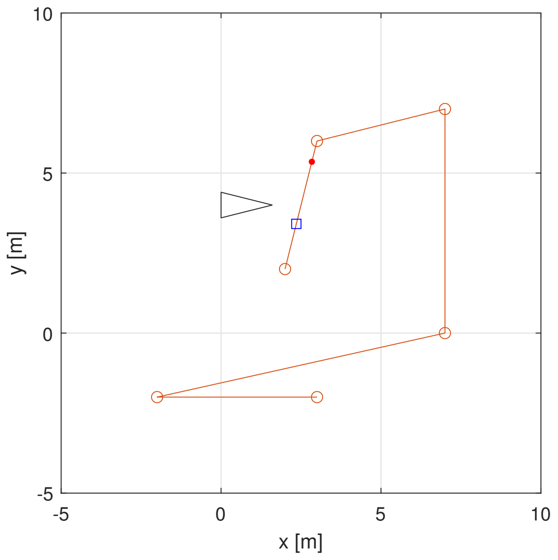

Figure 2: Problem 3b.

(d) Now modify the GTG control logic to calculate `x_des` and `y_des` by calling the `find_target()` function you just wrote. To do so, insert the following lines right before the line `U_GTG = self.K_e*np.array([x_des - x, y_des - y])`:

```
self.target = find_target(path, np.array([x, y]))
x_des, y_des = self.target[0], self.target[1]
```

(e) Run `main.py`. The robot should follow the path closely. When the robot gets to the final waypoint and stops, download the CSV file and plot the speed of the robot vs. time. Make sure the speed is close to the maximum speed throughout the maneuver.

(f) Can you suggest ways of improving the path following performance, especially around tight corners (i.e., making the robot track the path more closely without slowing down too much)? Implement your suggestion and include your code/plots. Include rationale for your design. Don't be afraid to be creative. There is not one right answer.

In the PDF file of your solution, include your plot from part (e) and your answer to part (f). Please also include your `controller.py` file(s). Make sure the code is thoroughly commented for full grade.