MAE 4180/5180, CS 3758, ECE 4772/5772

Autonomous Mobile Robots

Homework 1 - due Fri. Feb 19 by 11:59 PM

A document containing the answers and figures should be uploaded to **Gradescope** as `HW1writeup.pdf`. All the code and the files needed to run the code are to be uploaded to **Canvas** as a zip file named `HW1code.zip`. Specific functions, as indicated below, should be uploaded to their corresponding assignments on **Canvas**.

Notes about autograded assignments:

- We **highly** recommend you develop and debug the autograded functions in Matlab. The error messages that the autograder provides are not as explicit as in Matlab.
- You may submit as many times as you like until the deadline
- Make sure to also include the functions in the zip file you upload to Canvas

Changing Reference Frames (30 Points)

Nearly every robot task requires operating in multiple reference frames. Download and open `lidarScan.mat`, containing a sample lidar scan (as returned by `LidarSensorCreate.m`). The 681 data points correspond to the measured ranges (in meters) equally spaced over the angles $[-120°, 120°]$.

1. Edit the file `lidar_range2xy.m` to convert the raw lidar scan into 2D local robot coordinates. Assume the lidar sensor is located at the front of the robot pointing forward (i.e. the sensor pose in local coordinates is $[x_l, y_l, \theta_l] = [r, 0, 0]$, where r is the robot radius). The robot's x-axis points forward and y-axis points left. Assume the robot radius is 0.2 meters. Submit this function in the autograded assignment **Homework 1 lidar_range2xy** on Canvas.
2. Edit the file `robot2global.m` to transform any 2D point from the local robot coordinates into global coordinates, given the robot's pose. Submit this function in the autograded assignment **Homework 1 robot2global** on Canvas.
3. Edit the file `global2robot.m` to transform any 2D point from global coordinates into local robot coordinates, given the robot's pose. Submit this function in the autograded assignment **Homework 1 global2robot** on Canvas.
4. In the same figure, plot using `lidar_range2xy.m` the lidar scan (points in 2D) in the global reference frame assuming the robot pose is
   - $[3, -4, 2\pi/3]$
   - $[0, 3, \pi/2]$

   Plot each set of points (corresponding to the scan as viewed by the robot) in a different color. Make sure to label your axis and include a legend.

Getting to Know the Simulator and Running a Simple Control Program (20 Points)

Download the MATLAB CreateSim toolbox (download as zip)
`https://github.com/autonomousmobilerobots/iRobotCreateSimulatorToolbox`
download the latest version of the Matlab Toolbox for the Create from

`https://github.coecis.cornell.edu/AMR/MatlabiRobotCreateRaspPi`
and add both to your MATLAB path. Review the iRobot Create Simulator User Guide pages 4–10, and the MATLAB Toolbox for the iRobot Create (available on Canvas) pages 11–12.

The file `turnInPlace.m` is a simple control program that reads the robot's sensor data and makes the robot turn in place. You can use this function as a template for later functions you will write.

Make sure the CreateSim toolbox is in your MATLAB path and run `SimulatorGUI.m`. This will open the simulator interface. Click the Load Map button and select `box_map` from the files provided for this homework. Place the simulated robot anywhere on the map by clicking the Set Position button (click once to set the center location, click again to set the heading direction). Run the control program by clicking the Start button in the Autonomous group (bottom right side of the interface) and selecting `turnInPlace.m`.

1. What does the command `SetFwdVelAngVelCreate` (line 73 in `turnInPlace.m`) do? What are the inputs to this function?

2. The left and right wheel velocities are limited to ± 0.5m/s (observe what happens when you increase `cmdV` and `cmdW` on lines 66–67 in `turnInPlace.m`). Edit the program `limitCmds.m` to appropriately scale the forward and angular velocity commands so the wheel motors don't saturate (the shape of the trajectory should be maintained, i.e. do not simply use a max function without making sure the trajectory shape remains the same). Submit this function in the autograded assignment **Homework 1 limitCmds** on Canvas.

3. Edit `turnInPlace.m` to use `limitCmds.m` in the appropriate place, with wheel2Center=0.13 and maxV a value that is less than 0.5. From now on, you should always call `limitCmds` before sending commands to the robot via `SetFwdVelAngVelCreate.m`.

4. Write a function `backupBump.m` to have the robot drive forward at constant velocity until it bumps into something. If a bump sensor is triggered, command the robot to back up 0.25m and turn clockwise 30°, before continuing to drive forward again. Run the simulator and plot the robot trajectory.

   Note: To access the sensor data collected during simulation, type `global dataStore` in the MATLAB Command Window. The ground truth pose of the robot is available in `dataStore.truthPose`. The sensor data is collected by `readStoreSensorData.m`; we recommend you take a look at the function to see what sensor information is stored.

5. Write a control function, properly named, to make the robot do something new. What does the function do? Run the simulator using your new control function and plot the robot trajectory.

Feedback Linearization (30 Points)

The iRobot Create is a non-holonomic vehicle, so we can't send arbitrary V_x and V_y commands to the robot. Instead we must send V and ω commands, as with `SetFwdVelAngVelCreate.m`. Feedback linearization approximately converts $[V_x, V_y] \rightarrow [V, \omega]$ via a linear transformation.

1. Write a function `feedbackLin.m` to transform V_x and V_y commands into corresponding V and ω commands using the feedback linearization technique presented in class for a differential drive robot. (In other words, do not use a turn-then-move control strategy.). Assume the V_x and V_y commands are given with respect to the inertial frame. Submit this function in the autograded assignment **Homework 1 feedbackLin** on Canvas.

2. In the SimulatorGUI, place the robot near $[0, 0]$ by clicking the Set Position button. For a desired **local** (body-fixed) control input $[V_x, V_y]_B = [1, 1]$, plot the robot trajectory for different values of ϵ.

3. Write a function `visitWaypoints.m` to calculate the desired $[V, \omega]$ controls to drive the robot along a series of waypoints (using `feedbackLin.m` appropriately). The waypoints are given as a nx2 matrix where each row is the (x,y) coordinate of a waypoint. Keep track of which waypoint is being driven

toward using the index `gotopt`, and consider the waypoint reached if the robot is within `closeEnough` of the waypoint location.

4. Using `visitWaypoints.m` as the control function with $\epsilon = r$ (the robot radius, 0.2m) and `closeEnough`= 0.1, plot the robot trajectories for the following sets of waypoints: [-3 0; 0 -3; 3 0; 0 3] and [-1 0; 1 0].