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

Autonomous Mobile Robots

Homework 2 - due Fri. Feb 26 by 11:59PM

A document containing the answers and figures should be uploaded to **Gradescope** as `HW2writeup.pdf`. All the code and the files needed to run the code are to be uploaded to **Canvas** as a zip file named `HW2code.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
- **Reusing code:** We encourage you to reuse your prior functions in the new functions. For the autograded assignments, if you want to use a previous function you have written (for example robot2global), simply copy-paste it below the function you are writing.

Dead Reckoning (45 Points)

The iRobot Create encoder information can be read from the distance and angle packets (using functions `DistanceSensorRoomba` and `AngleSensorRoomba`, respectively). These packets provide the distance the robot traveled and the angle it rotated since the sensor was last read.

1. Given a known initial configuration (x, y, θ) of the robot within a global frame, the distance traveled d and the angle the robot turned ϕ, compute the new configuration of the robot. Explain your calculation. Note that you **cannot** assume a "turn then move" scheme. (Hint: Assume that the robot's wheels turn at a constant rate between sensor readings.)
2. Edit the function `integrateOdom.m` to integrate the robot odometry as calculated in part 1 (this is known as "dead reckoning"). Submit this function in the autograded assignment **Homework 2 integrateOdom.m** on Canvas
3. Generate a trajectory and gather data: **No need to submit the control code or the simulator map and config files**.
   (a) Define a map for the simulator. You may use the map from Homework 1 or define a new one.
   (b) Write a control program that will drive the robot in the simulator. The robot motion should include simultaneous non-zero forward and angular velocity (meaning the robot motion should include arcs). The program should read the odometry, either by periodically calling `DistanceSensorRoomba` and `AngleSensorRoomba` and storing the data or by periodically calling the function `readStoreSensorData`. Make sure the robot runs into a wall at least once. The program should be deterministic, that is the trajectory should be the same in repeated runs.
   (c) Using ConfigMakerGUI, create a config file that defines errors on the odometry.
   (d) Run the simulation without a config file (no noise is added to the sensors). Save the data.
   (e) From the same initial pose as the previous run, run the simulation with a config file. Save the data.

4. Plot in the same figure the true trajectory as captured by the overhead localization, the trajectory for the integrated noiseless odometry (using the function `integrateOdom.m` with data from the first run) and the trajectory for the integrated odometry which contains errors (using the function `integrateOdom.m` with data from the second run). Specify what error parameters were used.

5. Did the integrated trajectory from the first run match the overhead localization? Did you expect it to? Explain.

6. How does sensor noise affect the quality of the localization?

7. What might cause errors in odometry measurements?

Expected depth measurement (30 points)

In this section you will write a function that given the robot pose and the map, predicts the depth measurements. Note that the realsense sensor provides a depth image that corresponds to the depth of obstacles along the sensor-fixed x-axis, as shown in Fig 1 .

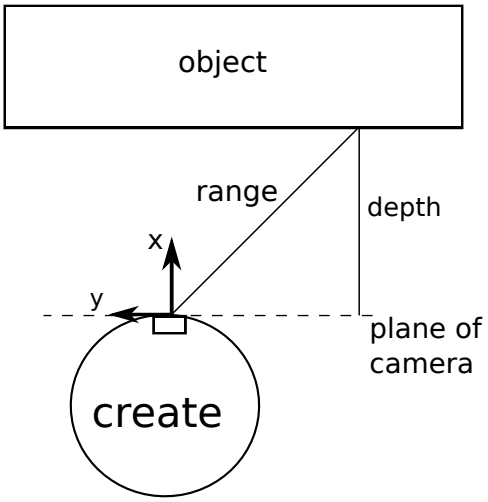

Figure 1: Depth information

1. Edit the file `depthPredict.m` to calculate the expected depth measurements (based on the pose) for a robot operating in a known map. These depth measurement correspond to the distance between the sensor and the closest obstacle. Assume the sensor reference frame is not rotated with respect to the robot-fixed reference frame. You may want to use the provided function, `intersectPoint.m`, which calculates the intersection point of two line segments. Hint: why do we need to provide the sensor-fixed frame origin?

   Note that the Field of View (FOV) of the depth image is centered around the sensor-fixed x-axis, the first element in the vector is the leftmost point and the last element in the vector is the rightmost point. We recommend looking at the documentation of the `RealSenseDist` function regarding the format of the depth measurements. Submit this function in the autograded assignment **Homework 2 depthPredict.m** on Canvas.

2. You may continue to work with the same control program and config file from the previous section, or create new ones. Your config file should have non-zero values for the realsense noise (in the "St. Dev." column). Make sure you are collecting and saving the following data (in `dataStore`):

   - `truthPose` (measurements from overhead localization)
   - `rsdepth`

3. In the simulator, load a map (you may use `box_map`), load the config file and run your control function. Save the data.

4. Based on the map and `dataStore.truthPose`, and given that the sensor (in the simulator) is positioned 0.16m along the X-axis of the robot-fixed frame, calculate the expected depth measurements. Plot the actual (simulated) depth measurement and the expected depth measurements on the same graph. Are they identical? Do you expect them to be? Explain.