MAE 4180/5180, ECE 4180/5772, CS 4758/5758

Autonomous Mobile Robots: Lab #3

Mapping

Instructor:
Dr. Hadas Kress-Gazit

Objectives

In this lab, students will generate an occupancy grid map using different types of sensors. This lab is designed to give some understanding of how mapping works and how various factors can contribute to the difficulty of this task. The robot is equipped with a depth sensor that measures the distance to obstacles and bump sensors to detect collisions. The overhead localization system provides truth pose in [x,y,θ].

Required Code

- `freedriveProgram.m`
- `driveArrows.m`
- `driveArrows.fig`
- `limitCmds.m`
- `readStoreSensorData.m`
- `logOddsBump.m`
- `logOddsDepth.m`

Prelab - Creating control code for the lab

Edit the provided file `freedriveProgram.m`. Add your occupancy grid calculations (including map initialization and necessary logging) and plotting. **We highly recommend that you plot the robot trajectory as it is moving.**

Required plots (to be shown to the TAs at the beginning of the lab)

- A plot of the Bump occupancy map from Homework 5.
- A plot of the Depth occupancy map from Homework 5.

---

1 Lab Manual

1.1 Station Set-up

(a) Open Matlab and change the working directory of Matlab to be the folder that contains your files. **Make sure to periodically save data to your online folder.**

(b) Unplug your iRobot Create. Make sure to only pick up the robot from the *bottom* and not the sensor platform. **Be careful not to hit the markers on the top of the robot.** Put it on the floor next to your lab station.

(c) Take note of the name of your robot.

(d) Turn on the robot by pressing the power button. The Raspberry Pi takes about 20-30 seconds to boot.

(e) In your MATLAB Command Window, run `Robot = CreatePiInit('robotName')` where `robotName` is the robot name from step (c). The initialization process creates the variable `Robot`, which contains the port configurations for interfacing with the Create and the added sensors and the robot name; it has five elements:

- `Robot.Name` contains the robot name.
- `Robot.OL_Client` used for getting the robot pose ground truth from the Optitrack system.

- `Robot.CreatePort` used for sending commands and reading sensors from the robot.
- `Robot.DistPort` used for getting depth information from the realsense camera.
- `Robot.TagPort` used for getting tag information from the realsense camera.

(f) Check that you have connected to the robot properly by running `BeepCreate(Robot)`. You should hear your robot beep.

(g) Put your robot on the field. Make sure you can access the robot's localization information by running `[x,y,theta] = OverheadLocalizationCreate(Robot)`.

(h) Put the robot in front of a tag, at least one foot away. Run `RealSenseTag(Robot)` and `RealSenseDist(Robot)`. Make sure you can get tag and depth information.

(i) If any of steps f-h fail, disconnect from the robot (run `CreatePiShutdown(Robot)`), shut it down, close Matlab, restart Matlab, turn the robot on and go back to step e.

(j) Assume that the **Realsense offset** (the location of the sensor in the robot-fixed frame) is (0,8cm), i.e. the x-axis offset is 0 and the y-axis offset is 8 cm.

Important:

- If the control function exits with an error, make sure to stop the robot by typing in the command window: `SetFwdVelAngVelCreate(Robot, 0, 0)`
- When you are done working with the robot, or you wish to restart Matlab or the connection to the robot, first run `CreatePiShutdown(Robot)` to disconnect properly from the robot.

1.2 Driving the Robot Around

(a) The function `driveArrows.m` opens a GUI and allows you to drive the robot around manually. Run your control program (`freedriveProgram.m` from the prelab assignment) and use the up/down/left/right arrow keys (or W/S/A/D keys) to control the robot's forward and angular velocities.**If you are plotting any other figures, make sure to call `figure(h)` immediately after, to keep that GUI in focus. To stop the robot, press the Space bar or Enter.** Practice driving the robot around the field. Notice that the robot's velocity is limited.

(b) After driving the robot around a bit, exit the control program (Ctrl+C) and close the `driveArrows` GUI. Type `global dataStore;` and **make sure all the appropriate sensor data was saved** (`truthPose, bump, rsdepth`). You may want to plot the robot's trajectory and make sure it is the path you manually drove.

1.3 Occupancy Grid with Bump Sensor

Using the code you wrote for homework (`logOddsBump.m`), you will build an occupancy grid map using data from the robot's pose and bump sensors. Use the provided function `freedriveProgram.m` as a starting point. Add your plotting functions according to the comments inside the function. This program will call `driveArrows` which will allow you to control the robot manually.

(a) Initialize a 16×28 occupancy grid of the environment with $p_0(\text{occ}) = 0.5$ as the prior occupancy probability for all cells (i.e. $\ell_0(\text{occ}) = 0$). The boundaries of the environment in the x direction are $[-3.5m, 3.5m]$ and y direction are $[-2.5m, 2.5m]$.

(b) Within your control program, call `logOddsBump.m` to update the occupancy grid at each time step with your new pose and bump measurements. You may want to add a new field to `dataStore` for the occupancy grid, so you can plot it later.

(c) Have your program plot the updated occupancy grid in real time so you can watch how your map evolves. On the same figure, plot the robot's full trajectory.

(d) Place your robot on the field and start your control program. Drive around the environment, frequently bumping into walls. You may even (politely) bump into other robots. Save your data to file. **Make sure all the sensor data, including the depth data, is saved**.

(e) Repeat for each member of your group, using a different grid size. Keep in mind that a finer grid may take longer to run.

1.4 Occupancy Grid with Depth

Using the code you wrote for homework (`logOddsDepth.m`), you will build an occupancy grid map using data from the robot's pose and depth sensors.

(a) Initialize a 22×12 occupancy grid of the same environment with $p_0(\text{occ}) = 0.5$ as the prior occupancy probability for all cells (i.e. $\ell_0(\text{occ}) = 0$).

(b) Within your control program, call `logOddsDepth.m` to update the occupancy grid at each time step with your new pose and depth measurements.

(c) Have your program plot the updated occupancy grid in real time so you can watch how your map evolves. If your program runs particularly slowly, you may prefer to calculate your occupancy grid update offline after you've collected all the data, but at least try to run it in real time. On the same figure, plot the robot's full trajectory.

(d) Place your robot on the field and start your control program. Drive around the environment, and try to "see" as much of the map as possible. Save your data to file.

(e) Repeat for each member of your group, using a different grid size. Keep in mind that a finer grid may take longer to run.

---

2 Post-Lab Assignment

Remember to submit your assignment as a group on Gradescope. To form a group:

1. **One** individual from the group submits the PDF on Gradescope.
2. When you click on your submitted assignment, there will be a section denoted by "Group" below the assignment name. Click on "View or edit group."
3. Add student(s) by selecting their name(s) from the drop-down menu under "Add Student."

2.1 Occupancy Grid with Bump Sensor (25 Points)

(a) Using the data you collected from Section 1.3, plot the final occupancy grid using the bump sensors (this should be the same as you saw in lab). On the same figure, plot the robot's trajectory. The file `lab3Map.mat` contains the coordinates of the walls in the environment. Plot the walls on the occupancy grid.

(b) How "good" was the map generated by the bump sensor? Comment on how complete or incomplete the map is (compared to truth).

(c) How did the grid resolution affect the map? What are the pros and cons of using a finer grid with the bump sensor?

(d) Describe how each group member modeled the bump sensor within `logOddsBump.m` (e.g. what values were used for $p(\text{occ}|\text{bump})$ and $p(\text{occ}|\neg\text{bump})$)? Did anyone's algorithm seem to work "better"? Why or why not?

(e) What happened when you bumped into another robot? (If you didn't bump another robot, what would you expect to see in your data?) Why?

2.2 Occupancy Grid with Depth (30 Points)

(a) Using the data you collected from Section 1.4, plot the final occupancy grid using the depth sensors (this should be the same as you saw in lab). On the same figure, plot the robot's trajectory and the environment walls.

(b) How "good" was the map generated by the depth sensors? Comment on how complete or incomplete the map is (compared to truth).

(c) How did the grid resolution affect the map? What are the pros and cons of using a finer grid with the depth sensor?

(d) Describe how each group member modeled the depth sensors within `logOddsDepth.m` (e.g. what was $p(\text{occ}|z_{depth})$)? Whose algorithm seemed to work "best"? Why or why not?

(e) What happened when you "saw" another robot with your sensor? (If you didn't see another robot, what would you expect to see in your data?) Why?

(f) What would you expect to happen to your map if there were lots of moving objects in the environment? How might you adjust your sensor model to account for this?

2.3 Lab preparation (mandatory, not graded)

(a) Which robot did you use?

(b) For the code you wrote as part of the homework, were there any coding errors that were not discovered when using the simulator?

(c) How much time did you spend debugging your code in the lab?

(d) Did you observe any behaviors in the lab that you did not expect, or that did not match the simulated behaviors?

(e) What was the contribution of each group member to the lab and the report?