Pset 4a: The Autonomous OG Mapper

Due Apr 5 by 8am **Points** 10

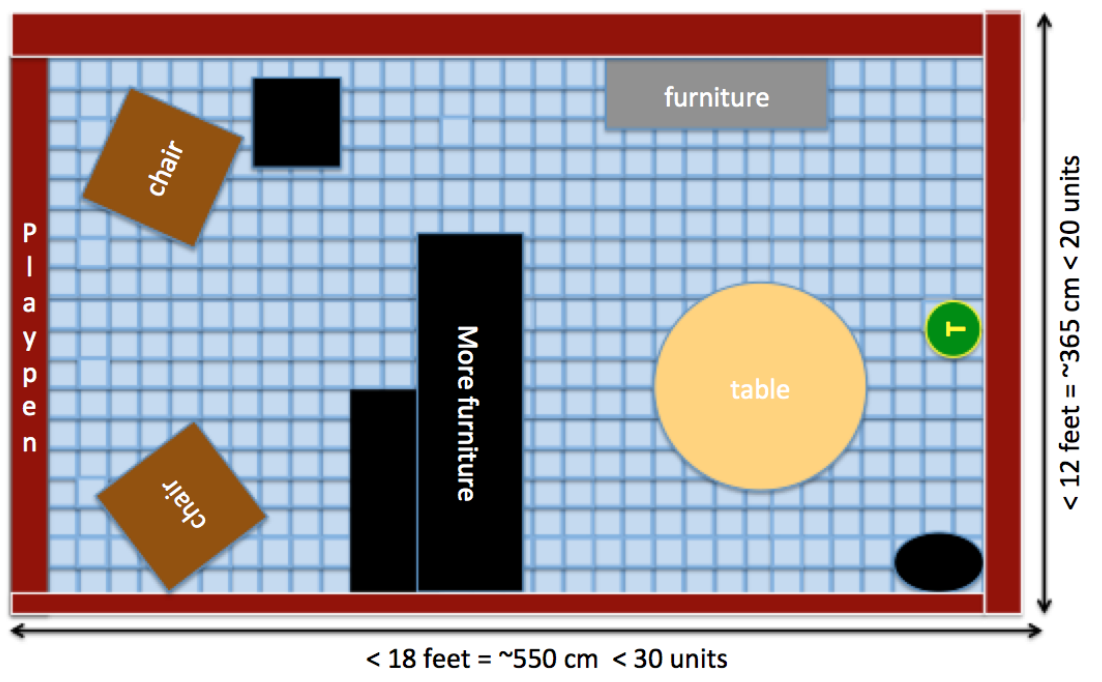
Today's Goal: Create a robot that can autonomously make maps! A map could be helpful for many things: cleaning a room systematically or searching for lost keys, or delivering objects between different offices and hallways on the floor of a building. The first step is for the robot to make a map that allows it to safely and smartly navigate in the environment it is placed.

Create your repository

In this problem set, you will program your robot to... map a fake office!

We will set up a dedicated space in the MD Basement (B127), bounded by a playpen fence and with some moveable furniture, and the goal for your robot is to construct a map of that space. You will use the Occupancy Grid algorithms discussed in lecture, and use the Depth Sensors to determine if a cell should be marked occupied or not. You will also use the EKF package from lab 4 to keep track of your robot's position. This is similar in many ways to the safe wandering problem set, but simultaneously keeping a map of where the robot has been in the past. This problem set is split over two weeks, with 10 points assigned to each week's demonstration.

(PlayPen) Office MAPPING
Caveats: Not drawn to scale. Furniture will move before final test.
Robot will start in the middle front of pen (green T)


furniture
chair
chair
Playpen
More furniture
table
T
< 12 feet = ~365 cm < 20 units
< 18 feet = ~550 cm < 30 units

Step 0: Some Parameters that you will need:

Turtlebot dimensions

- 35.4 cm x 35.4 cm x 42 cm (without laptop)
- Bounding box of 40cm x 40cm

Occupancy Grid Map dimensions

- Grid Unit size = 20 x 20 cm (i.e. robot body = 4 units)
- Playpen Office: Maximum 18 x 12 feet (550 x 365cm)
- Occupancy Grid = At least 20 x 30 units (use some padding for safety)

We will assume that robots are always started against the wall in the front of the playpen, facing into the office as shown in the image (the position will be marked with masking tape on the floor). We will set up a test arena with furniture, but we will rearrange things between the test and final run.

Step 1: Set up your OG Map and basic data structures

(a) *Map:* Pick a map array size that is bigger than the room (e.g. 30 x 40 units for safety). The units in the array are discrete values (empty, occupied, unknown).

(b) *Robot Position:* A grid unit is a quarter of a robot, so a robot position spans multiple grid cells. Use the lab 4 EKF package to determine the robot's position and pose on the map at all times.

(c) *Display:* At all times, you should display a simple window with the current map and robot position. The map should have every cell marked one of three colors: white (empty) black (occupied) and blue (unknown). You should also show starting position in green and current robot position in red. This will help greatly with debugging. To make the display, you can use OpenCV and draw boxes for each cell your robot discovers (incrementally update a map image for efficiency), and periodically save the image to a file. It is also helpful, but not necessary, if you display the robot's trajectory (e.g. a line linking boxes).

Step 2: Simple Exploration Strategy

For exploration, start with a simple random walk (e.g. similar to pset2) using the depth sensor to avoid running into objects. Past experience has shown that favoring straight line movement in the random walk is better than lots of turning. You can also try other movement strategies, such a favoring movement along grid lines. *As a first milestone, you should get your robot moving safely through the fake office space, without bumping into furniture, and while using EKF to report its position on the map.* Note that over time you will see drift/inaccuracy in where the robot thinks it is and where it really is on the map, in spite of the EKF corrections. In general we will not run the robot for longer than 2-3 minutes.

Step 3: Populating the Map

As your robot moves around, it can mark visible cells as Occupied or Empty.

(1) Wherever your robot is physically standing should be marked Empty

(2) If your robot detects an obstacle within 20cm in front of it (e.g. two grid cells in front of you) mark those cells Occupied.

Note that this is a very conservative model --- To build the full map, your robot must visit all empty spots, ignoring information about empty spaces from the depth sensor. In addition, a random walk is fairly inefficient for exploration. You will improve this is part (b) of the pset. *For part (a), the main goal is to demonstrate correct behavior by sticking with a very simple strategy that is easy to visually debug.*

Step 4: Testing

Start your robot in the start location and have it do mapping until you send it a pause signal (usually 2 minutes or so). During the process, the robot should continuously display its current map, and at the end it should leave the final map up on the screen.

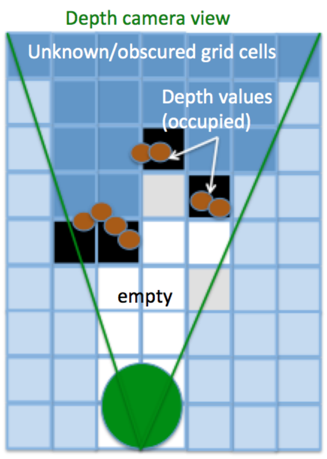
Part (a) Demonstration:

Grading Scheme (10 points): We will run the same test as explained in Step 4. **8 Points for correct behavior:** safely moving about the area, displaying a visually helpful real-time map, marking both empty and occupied locations, reasonable match between map and reality. **2 points for code/strategy.** Make sure that your README file clearly explains the details for Step1 (map parameters and visualization), Step 2 (your exploration strategy), and Step 3 (how you marked occupied, and how this interacts with collision avoidance).

Step 5: A Better Mapper

Your mapper can be improved in many ways (some of which might be inspired from your experience during part a). For part (b) you will extend your mapper in the following two ways:

(1) Use a Depth Sensor Model: Your robot can see open areas around it using the depth camera. Pick a fixed distance/angle range over which your robot can reliably determine "empty" and "occupied" spots; mark those spots so that your robot does not need to travel there in order . Explain your depth sensor model in your writeup.



(2) Use a Smart exploration strategy: Use the occupancy grid map to guide the robot towards unknown areas to construct a map more efficiently -- combine a movement strategy with frontier search. Use the frontier node criteria to determine if you have finished the map, and save the final map image as a png file.

The combination of (1) and (2) means that your robot can much more quickly construct a map of the whole area.

Part (b) Testing and Grading Scheme

Test: Your robot will explore the space for 3 minutes, or until it declares that it has finished the map. At the end, it should save the final map as a png file. Upload this map as part of your final . (Note: Your map should also show the starting and final position of the robot.)

Note: However hard you try, your map will be a distorted version of the office layout, and that is OK! Both furniture and walls won't fall perfectly along grid boundaries even if there was no error. The longer you explore, the more you can overwrite the original map because of location drift. This is why SLAM is an exciting topic, though computationally and practically much more complex to get working.

Grading Scheme (10 points): 5 points for final map quality, 2 points for < 3 minutes time, and 3 points for code implementation. Make sure that your readme file explains (1) how you came up with and implemented your depth sensor model and (2) what your exploration strategy is and how you incorporated frontier search.