Lab 4: Mapping with LIDAR

CSCI 3302: Introduction to Robotics

Due 03/01/22 @ 11:59pm

Note that this is a 1-week long lab

2 points/day extra credit for early turn-in (up to 10pts)

The goals of this lab are to:

- Implement a basic discrete map representation
- Use coordinate transforms to map sensor readings into world coordinates
- Understand world representations and algorithmic aspects of mapping

You need:

- A working Webots Installation
- The Lab 4 base code from Canvas

Overview

Mapping and navigation are standard primitives in autonomous robots. Although the different map representations and planning algorithms vary drastically in complexity, the problems remain the same: localization and sensing are uncertain, maps are quickly corrupted due to changes in the environment, and both mapping and navigation have to deal with limited onboard resources. In this exercise, you will use ePuck's new LIDAR sensor to map objects in the environment. You will initially display the objects' coordinates in the console and then implement data structures and helper functions that (1) allow you to store a map in ePuck's memory and (2) are suitable for path planning.

Instructions

Each group must develop their own software implementation, turned in by one team member with the lab report. Thus, each team should turn in a zip file including:

- The group's code (1 .py file)
- A single lab report (1 .pdf file)

- The Webots environment folder containing your world and code file.

If your group does not finish the implementation by the end of the class, you may continue this lab on your own time as a complete group.

Part 0: Preliminaries

The coordinate system for the ePuck assumes that point (0,0) is at the top-left of the arena, with x increasing to the right (your right and ePuck's left to forward) and y increasing to the bottom (your bottom direction and ePuck's forward direction). The robot's odometry equations have been modified to accommodate this change, and the start position of the robot has been pre-specified in the starter code.

Part 1: Setting up your LIDAR

1. Initialize an array/list data structure to hold your LIDAR's incoming readings.
   You will be accessing the LIDAR's sensor readings with lidar.getRangeImage(). This will return an array containing a distance measurement from each angular 'bin'.

2. Before your controller's while loop, using the provided constants LIDAR_ANGLE_RANGE and LIDAR_ANGLE_BINS, programmatically compute the angle offset ϕ for each of your LIDAR's readings, using the knowledge that the middle bin (e.g., array index 10) is at ϕ=0 since it's pointing directly in front of the robot, and that the total horizontal spread is LIDAR_ANGLE_RANGE radians. Store the result in an array such that:
   lidar_offsets[beam_index] = angle_offset_in_radians.

3. We have already added code to get sensor readings from the robot's LIDAR to the beginning of your WHILE loop. The sensor object is named *lidar* in the code, and this can be done with *lidar.getRangeImage()*, which returns an array of distance measurements corresponding to each laser being emitted from your robot.

Part 2: Map Construction

1. This world is equipped with a `display` node that is added to the robot's `turretSlot`. You can write to the display by first setting a color (`display.setColor(0x0000FF)`) and then draw individual pixels (`display.drawPixel(int x, int y)`). The color is provided as a three byte hexadecimal number in the order Red, Green and Blue. For example, 0xFF0000 is bright red. The display has 300x300 pixels. Come up with a way to convert from world coordinates in meters to display coordinates in pixels so that the 300x300 display covers the entire 1m x 1m floor.
Hint: You can use the (`display.drawLine(int x1, int y1, int x2, y2)`) to plot beams from the LIDAR

Please use red for drawing the robot pose, blue for the obstacles detected by the LiDAR, and white for the free space. Pixels in unmapped areas should remain black.
The hexadecimal values of the above colors are shown below:
0xFF0000 - Red - Robot pose
0xFFFFFF - White - Free space
0x0000FF - Blue - Obstacle

2. Use your findings from above to draw a red pixel into the map wherever the robot is (`pose_x, pose_y`). Test your map by setting the controller state to "line_follower". You can store your pose and draw them last so that they are not hidden by other pixels.

Part 3: Transform Lidar readings

1. Write down the homogenous transform to convert readings from your LIDAR's frame of reference $\left(x_{lidar}\right)$ into the robot's frame of reference $\left(x_r, y_r\right)$ and then into world frame coordinates $\left(x_{world}, y_{world}\right)$.
   Testing Tip: If the robot is at pose $\left(0, 0, 0^o\right)$ and the LIDAR beam is facing forward (i.e., the reading at index 10 which is offset by 0 degrees) and detects an object 5cm away, your homogenous transform should return $(.05, 0)$ for the object's position. Gradually add complexity to your tests.

2. Implement a loop that cycles through all readings and converts them into world coordinates using the equations that you derived. Use the "stationary" state and drag the robot around manually to understand the data you are getting. Once you are confident that your code is working reasonably well, change the controller state from "stationary" to "line_follower" at the top of your code.
   Implementation Hint: You'll need to convert from (bin,distance) to (x,y) coordinates in the robot's frame of reference. You can assume the LIDAR is mounted at the robot's origin. Next, you'll need to take the (x,y) point in the robot's frame and convert it to (x,y) in the world frame, using a homogeneous transform. Make sure you ignore values that are infinite range!

Part 4: Visualization and Path Planning Prep

1. Add all the sensor readings from your laser scanner to the map using blue pixels. If successful, you should see the outline of your objects emerging as the robot traverses the space.

2. Draw the free space on your map using white pixels, covering each pixel along the LIDAR beam from its origin to its object detection. You can use an algorithm like Bresenham's Line Algorithm or `display.drawLine` to figure out which pixels to fill in.
   Implementation Hint: Fill in the white pixels before filling in your object detection (blue) pixels, so that you can essentially overwrite the white pixel with the object detection.

Part 5: Lab Report

Create a report that answers each of these questions:

1. What are the names of everyone in your lab group?
2. Roughly how much time did your group spend programming this lab?
3. The display is setup with 300 rows and 300 columns to cover a 1m x 1m area. What is the spatial resolution of each pixel in the map? Show how you arrived at your answer to get credit.
4. How would mapping be affected if the odometry is not perfect and has errors?
5. How could you choose a good resolution for your map? Elaborate on what happens if your resolution is too low or too high.
6. You have been using the starting line to perform “loop closure”, recognizing where the robot is to reset its odometry and prevent localization drift. How can you use the LIDAR sensor to accomplish a similar goal?