Lab 4: Mapping (with ROS)

CSCI 3302: Introduction to Robotics

Due 3/10/20 @ 11:59pm

The goals of this lab are to:

- Implement the line-following behavior and loop closure on Sparki using ROS
- Implement a basic discrete map representation and map an environment
- Use coordinate transforms to map sensor readings into world coordinates
- Understand discrete and algorithmic aspects of mapping

You need:

- A working ROS Installation
- The Lab 4 ROS base code: sparki-simulator.py and associated image files, and lab4-base.py (for your lab code)

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 is uncertain, maps are quickly corrupted, and both mapping and navigation have to deal with limited on-board resources. In this exercise you will use Sparki's ultrasonic sensor to map objects in the environment. You will initially display the objects' coordinates on the screen and then implement data structures and helper functions that (1) allow you to store a map in Sparki'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:

- The group's code (1 .py file)
- A single lab report. (Please put the number of each question next to your answers, rather than turning in your answers as an essay)

Part 1: Preliminaries and Line Following

1. In a terminal, start **roscore** on the system

2. Run the **sparki-simulator.py** node (e.g., 'python sparki-simulator.py')

3. Verify that the node is running – type '**rostopic echo /sparki/state** ' in a new terminal. You should see a streaming text indicating readings from the robot's IR sensors

4. Using the **lab4-ros.py** base code and your previous C implementation as a reference, implement line following behavior (**with loop closure**) in the main loop of the program

   Tip: You'll want to establish global variables to track your robot's pose and state. Since your updates to these variables will be coming asynchronously over ROS topics, **your subscriber callbacks should update these variables** .

   Tip: To command the robot's motors, you will need to publish a **std_msgs/Float32MultiArray** message on the **/sparki/motor_command** topic with data [left_motor, right_motor] (e.g., [1.0, 1.0])

   Tip: To reset the robot's odometry (during loop closure), send a **geometry_msgs/Pose2D** message over the **/sparki/ set_odometry** topic.

5. Before your program's main loop, publish a message to set the servo angle so that it will point inwards toward the center of the line following map (The servo can be set between [-80,80] degrees)

6. Implement a 50ms maximum CYCLE_TIME for your program's main loop so you don't flood the robot driver with commands. Use time.time() and rospy.sleep(float) for this.

7. Have your robot take a distance reading using the ultrasonic sensor each run of your main loop by publishing the appropriate message.

   (Hint: "rostopic list" shows all available topics, and "rostopic info /topic" will show details)

Part 2: Coordinate Transforms

1. Write down the homogenous transform to convert readings from Sparki's Ultrasonic Sensor's frame of reference $(x_{ultrasonic})$ into the robot's frame of reference (x_{robot}, y_{robot}) and then into world frame coordinates (x_{world}, y_{world}).

   Tip: If Sparki is at pose $(0,0,0)$ and the ultrasonic sensor is facing forward (0 degrees) and detects an object 5cm away, your homogenous transform should return $(.05,0)$ for the object's position.

2. Write a Python function that will transform a point given in Sparki's Ultrasonic Sensor frame of reference into world frame coordinates.

Part 3: Map Construction

1. Choose a resolution for your map (e.g., 3cm per cell). Create a data structure world_map[#y_cells][#x_cells] to store and maintain it.

   The approximate size of the line-following course is ~60cm wide by ~42cm tall.

2. Create a function that takes a float (x, y) coordinate and returns the map/grid location that it corresponds to (i, j).

3. Create a function that takes a grid location (i, j) and returns the (x, y) world coordinate at its center.

4. Using the value "1" to indicate obstacles and "0" to indicate navigable space, use the functions from Part 2 with the ultrasonic sensor readings received via the robot's state broadcast to populate your map as Sparki traverses the line course.

Part 4: Visualization and Path Planning Prep

1. Write a function that will visualize your map as your program runs. You will have to pick a resolution for each 'cell' from your map, drawing navigable space as empty (or empty boxes) and occupied space as filled-in. You may implement this as terminal output (e.g., using Python's print statements) or as a graphical visualization in a manner of your choosing. You should also visualize the current position of the robot on the map.

2. Algorithms such as Dijkstra's are not made for grid maps, but rather for graphs in general. To compute the distances from a given start position to an end position, you will need three functions which you are to implement to complete the lab:
   a. A function that assigns a single integer to each grid cell of your map $f: (i, j) \rightarrow Int$ (e.g., $f(i, j) = j * width + i$)
   b. A function that takes as input an integer (cell index) and returns the corresponding 2D coordinates $f: Int \rightarrow (i, j)$
   c. A function that takes as input two integers representing the indices of two different cells and returns the "cost" to move between them $f: (Int, Int) \rightarrow Int$. This function should return a low value (e.g., 1) if both cells are adjacent and unoccupied, and a high value (e.g., 99) otherwise.

Part 5: Lab Report

Create a report that answers each of these questions:

1. What is the drawback of a data structure that stores distances between every possible pair of nodes in the graph? How does the implementation in 4 -2 address this problem?

2. List the subscribers that your code has. What message types are they receiving? Please provide a brief description.

3. List the publishers that your code has. What message types are they sending? Please provide a brief description.

4. Please paste the resulting final map as rendered by your code that shows all the obstacles found and mapped by your robot. Is there a mismatch with the provided obstacle image file (obstacles.png in the simulator)? If so, why? How could you make it better?

5. Do you like ROS?

6. Roughly how much time did your group spend programming this lab?