CS 733/833 Introduction to Mobile Robotics

Fall 2021, Prof. Begum

Lab Assignment # 5: Obstacle Avoidance with Bug 2 Algorithm [7 points]

Introduction:

In this lab, you will write a ROS package for obstacle avoidance using Bug2 algorithm. The theory of Bug2 has been discussed in the class. You will implement a Bug2 algorithm to enable your robot to reach a goal position from a start position in a given map while avoiding an unplanned obstacle. This lab has two components and will be completed in two weeks.

Lab Work:

Navigating to a pre-defined goal while avoiding an unplanned obstacle using Bug2 algorithm requires implementation of two behaviors in the robot: "`goal-seek`" (where the robot will follow a planned trajectory to reach the goal) and "`wall-follow`" (where the robot will follow the wall of the obstacle until it reaches the m-line). The robot needs to continuously localize itself while executing these two behaviors. You will use AMCL for localizing the robot (the same way you used AMCL in Lab 4).

We have provided a skeleton sub-class where you can implement the two listed behaviors. The sub-class builds upon your earlier work and therefore requires you to have a copy of your earlier code (from Lab 3 or 4) in the same scripts directory where you store this lab's skeleton code.

This lab marks a substantial increase in challenge and freedom over earlier labs. To that end you will find the skeleton code is far more sparse than earlier codes. We strongly encourage you to consider the requirements of the given task and make any modifications to the code that you deem necessary to generate the desired result. (Note: you are free to use code/information from outside of the class to solve the assignment but remember to cite your sources properly in your report to avoid penalty for plagiarism)

[Additional Resources on Bug 2 Implementation: The Algorithm 2 in the reference book *Principle of Robot Motion* is a Bug 2 algorithm that you may find useful to organize your thoughts. Section 2.3 of the same book discusses some implementation tricks for the wall follow behavior that you may find useful]

Goal-Seek Behavior:

We will evaluate this behavior of your robot as it traverses a rectangular path from any starting point and stops after returning to that point. Throughout this process you will use a map that you built in Lab 4. This part of the lab is due on November 5.

1. Subscribe to the 'amcl_pose' rostopic.

2. Complete the **goToGoal** and **driveStraightAMCL** functions. The functions should use the controller you designed in Lab 3 for smooth robot motion.
3. Modify **executeTrajectory** so that the robot navigates along a rectangular path of your choice and returns to where it started. Note: We will be checking to make sure that AMCL is used to identify the locations along the path
4. Ensure that you publish visualization markers for each goal to RViZ and can see them in the application. The starter code provides an example of how to create visualization markers. You will have to explicitly include those markers in the RViZ following these steps:
    a. Press the add button located in the bottom left,
    b. select Marker,
    c. then select the appropriate topic name.

Wall-Follow Behavior:

We will evaluate this behavior of your robot as it follows the wall of a newly encountered obstacle while travelling on a given map. This part of the lab is due on October 29.

1. Subscribe to the 'scan' and 'amcl_pose' rostopics (discussed in Lab 4).
2. Investigate the LaserScan message to understand its component parts. Note: It may also help to look at the *laserscan_to_image* function we provided you in Lab4 for additional context.
3. In the **laserscanCallback** modify the variable "**self.wall_following**" variable to *True* if an obstacle is detected so that the robot begins executing the **WallFollowProtocol** function (it should print out a message to let you know the function has been called).
4. Complete the **getTangentLine** and **WallFollowProtocol** functions. There are various ways you can identify the tangent line to a set of points (corresponding to the laser scan of the obstacle). A simple **slope-based** method was discussed in the theory class. The **appendix 1** of this manual discusses a more robust **regression-based** method for finding tangent lines and the starter code also provides a pseudo code (Note: linear regression was not discussed in the theory class; use this method if you are comfortable with regression). You can, however, use any method other than these two for calculating tangent lines.

Complete Bug 2 Demonstration:

Using the `goal-seek` and `wall-follow` behaviors, you will demonstrate how your robot avoids a newly encountered obstacle and reach a pre-defined goal location in a given map. This part of the lab is due on November 5.

Functions:

All units should be in meters, degrees, and meters/second.

goToGoal(speed, goal_pos)

Using AMCL pose information, rotate towards the given point and then drive straight towards it. AMCL is used to calculate the rotational and distance needed to be travelled but you can use Odomoetry-based functions to move the actual robot.

driveStraightAMCL(speed, distance)

Using AMCL pose information to travel the given distance. We strongly recommend using this in conjunction with the "goToGoal" function.

getTangentLine()

From the depth data (in the 'scan' rostopic), identify a tangent to the contour of the obstacle that is visible in the heading direction of the robot. If you use slope-based method, from the robot's coordinate system, take an average of the obstacle points that are left and right of the robot. Find the angle of the line that connects these points relative to the robot and use this angle to calculate the robot's next heading. If you want to use regression, see the **appendix 1** of this manual.

wallFollowProtocol()

Perform the Wall Following Protocol: drive parallel to the obstacle and frequently check to make ensure if it is still nearby. Continue doing these actions until you reach the m-line at which point you should start moving towards the goal. The protocol may consist of these primary steps:

1. Drive your robot along the tangent for δt time. Try between 1.5 to 4 seconds, at a low robot velocity to identify a suitable value for the δt such that the robot moves alongside the obstacle (test with a circular and rectangular/square-shaped obstacles).
2. Check if your current position is along the m-line. You can use various strategies to recognize that the robot has re-encountered the m-line. See **appendix 2** of this manual for one example strategy. You may have to use an error margin to make this decision. Otherwise, your robot may overshoot, and you may not be able to detect the m-line at all.
3. If yes (from step 2), trigger the "`goal-seek`" behavior to reach the goal.
4. If no (from step 2), orient toward the obstacle and repeat steps 1 and 2. To re-orient yourself you will need to find the angle to turn from your current heading. There are several ways to calculate this position. One way is:
    a. As you start following the tangent of the obstacle (e.g. after turning right), keep track of the closest obstacle point (i.e. from the range reading within -90° to 0°)
    b. After δt time, use the position of the closest obstacle point (from the most recent laser scan) to orient toward the obstacle.

Note: make sure to set the "**self.wall_following**" variable to *False* after you have crossed the m-line so that the robot can return to normal operation.

Suggestions

Based on responses from previous years, here are some suggestions that may help your system work more effectively:

- Using a large obstacle has sometimes generated better results as it is easier to re-locate the obstacle after you have begun the wall following protocol.
- Try to test in a spacious location. If you are near walls, you may accidentally begin following those.
- Consider the scenario where your scan sees both the obstacle and the wall behind it. What steps can you take to make sure your algorithm generates a tangent line from the obstacle as opposed to sensor data from other objects in the environment.

Deliverables and Deadlines

I. The specific deliverables are:
   1. Demonstrating the `goal-seek` behavior without any obstacle.
   2. Calculating the tangent line from the depth data and publishing the angle in world coordinates to the topic /tangent_angle and using the tangent line information to execute the `wall-follow` behavior.
   3. Demonstrating that the `wall-follow` behavior becomes operational when an obstacle is sensed. You need to publish a Boolean message to the topic `/wall_following`.
   4. Recognizing the robot has reached the m-line and changing its behavior to `goal-seek`. You need to publish a Boolean message to the topic `/wall_following`.
   5. The robot stops after reaching the goal position.

II. **October 29:** We will evaluate the `wall-follow` behavior of your robot (i.e. the **w**allFollowProtocol() and related functions, deliverable 2). Place your robot in front of an obstacle of your choice and make it follow the obstacle. We will not evaluate any stopping condition; the robot may continue following the wall indefinitely. Calculating the tangent line and thereby correctly demonstrating the wall-follow behavior is worth 3.5 points and you will be graded for this task. There is no report due on this day.

III. **November 05:** You will demonstrate the `goal-seek` behavior along a rectangle (deliverable 1) and the complete Bug2 algorithm (deliverables 2-5 but you will not be graded on deliverable 2 which was done on October 29, and we assume it to be functional). The report and codes are also due on this day.

You will submit the following:

1. Copy of the sign-off sheet.
2. Compressed folder containing your package.
3. A **brief** report communicating your findings regarding the lab. Report on the difficulties that you encountered while implementing the Bug2 algorithm and the ways you tackled them. Discuss the parameters you choose on a trial-and-error basis. The complete document should not exceed 4 typed pages.

4. If you used external sample code other than the packages listed in this lab document, please also submit a document called References.txt that lists all such sources, as well as a very brief description of what you used from each one.

CS 733/833 Lab 5: **Obstacle Avoidance with Bug 2 Algorithm**

Name: _____

Task **[Due date]**	**Signature**	**Grade**
Modified executeTrajectory (Goal Seek behavior) [November 5]		1.5
Triggering Wall Follow Behavior [November 5]		0.25
Calculate tangent line and demonstration of Wall-follow behavior [October 29]		3.5
Stops at the m-line [November 5]		0.5
Arrives at Goal Position [November 5]		0.25
Report [November 5]		1

Appendix 1:

Linear regression for tangent calculation

Performing linear regression on the measured data points provides a more accurate tangent line. Assume there is a vector for the x and y coordinates of the laser scan points: $p_x \in \mathbb{R}^N$ and $p_y \in \mathbb{R}^N$, where N is the number of scanned points. Also assume $\theta \in \mathbb{R}^N$ is a vector of the scan point angles. Concatenating these two vectors horizontally results in a vector that represents all points in the scan: $p_{scan} \in \mathbb{R}^{Nx2}$. We propose the following linear equation to model the points.

$$\tilde{p}_i = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \theta_i + \begin{bmatrix} w_3 \\ w_4 \end{bmatrix} \tag{1}$$

Equation 1 is a parametric line that start at (w_3, w_4) and move in the (w_1, w_2) direction. The subscript i designate the ith element of $\tilde{p}$ and θ. The x and y components of the models can be expressed as follows.

$$\tilde{p}_{xi} = w_1\theta_i + w_3 \tag{2}$$

$$\tilde{p}_{yi} = w_2\theta_i + w_4 \tag{3}$$

Equations (2) and (3) can be re-written as matrices.

$$\tilde{p}_{xi} = \begin{bmatrix} \theta_1 & 1 \\ \theta_2 & 1 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} w_1 \\ w_3 \end{bmatrix} = A_x \begin{bmatrix} w_1 \\ w_3 \end{bmatrix} = A_x w_x \tag{4}$$

$$\tilde{p}_{yi} = \begin{bmatrix} \theta_1 & 1 \\ \theta_2 & 1 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} w_2 \\ w_4 \end{bmatrix} = A_y \begin{bmatrix} w_2 \\ w_4 \end{bmatrix} = A_y w_y \tag{5}$$

Our objective is to minimize the sum of the inner products of the x and y errors. We define the following quadratic objective.

$$J = (A_x w_x - p_x)^T (A_x w_x - p_x) + (A_y w_y - p_y)^T (A_y w_y - p_y)$$

$$J = w_x^T A_x^T A_x w_x - 2w_x^T A_x^T p_x + p_x^T p_x + w_y^T A_y^T A_y w_y - 2w_y^T A_y^T p_y + p_y^T p_y$$

The objective is minimized when the gradient is equal to zero.

$$\nabla_x J = 2A_x^T A_x w_x - 2A_x^T p_x = 0$$

$$w_x = (A_x^T A_x)^{-1} A_x^T p_x$$

$$\nabla_y J = 2A_y^T A_y w_y - 2A_y^T p_y = 0$$

$$w_y = (A_y^T A_y)^{-1} A_y^T p_y$$

Appendix 2:

Change of basis for tangent calculation

Transforming the robot's position from the world coordinate system to the m-line coordinate system will help to detect when the robot re-encounters the m-line. As shown in the figure below, first, the robot's position should be expressed in the translated world coordinate system. The translated world coordinate system has its origin at the beginning of the m-line. We then need a rotation matrix that rotates the translated world coordinate system to the "m-line coordinates". Let $\hat{L}$ be a unit vector in the direction of the m-line in world coordinate system. $\hat{L}_x$ and $\hat{L}_y$ designate the x and y coordinates of the m-line unit vector. The m-line x and y basis vectors in translated world coordinates are $(\hat{L}_x, \hat{L}_y)$ and $(-\hat{L}_y, \hat{L}_x)$, respectively. We want a rotation matrix ${}^{m}R_w$ that rotates those basis vectors until they equal $(1, 0)$ and $(0,\ 1)$.

$$ {}^{m}R_w \begin{bmatrix} \hat{L}_x & -\hat{L}_y \\ \hat{L}_y & \hat{L}_x \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} $$

$$ {}^{m}R_w = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{L}_x & -\hat{L}_y \\ \hat{L}_y & \hat{L}_x \end{bmatrix}^{-1} $$

$$ {}^{m}R_w = \begin{bmatrix} \hat{L}_x & -\hat{L}_y \\ \hat{L}_y & \hat{L}_x \end{bmatrix}^{-1} $$

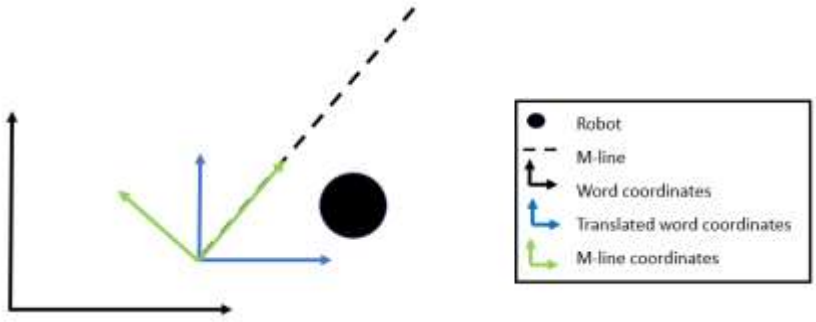

Figure 1: visualization of the reference coordinate systems