COSC 081/281 - Principles of Robot Design and Programming (Term: Spring, Year: 2023)

Assignment: PA0 - Report

Author: Carter Kruse

Date: April 3, 2023

Implementation / Description

Overview

The program is a simple ROS node that allows the robot to move in a random walk.

The program processes the laser scan to identify the minimum range value within a field of view. The minimum range value is compared with a threshold value to set a flag variable identifying that the robot is too close to an obstacle. This is within a callback function.

When the flag is not set, the robot moves forward. When the flag is set, the robot rotates a random angle between *-pi* and *pi*, unsets the flag, and moves forward again. The robot rotates clockwise or counterclockwise depending on the value of the angle. If the value of the angle is in the interval *(0, pi)*, it rotates counterclockwise (positive angular velocity). If the value of the angle is in the interval *(-pi, 0)*, it rotates clockwise (negative angular velocity). This is within the main loop, so the callback function might set the flag back if there is still an obstacle.

Design Decisions

In developing the program, various design decisions were made, primarily to ensure the accuracy/completeness of the code. The global variables *LINEAR_VELOCITY* and *ANGULAR_VELOCITY* remained consistent with what was given, as each allows for the user to see the execution of the robot's movements. The global variable *MIN_THRESHOLD_DISTANCE* remained consistent with what was given, as it did not need to be modified. If the minimum clearing distance were decreased, it is possible that the robot would become stuck against a wall (or other obstacle) when trying to rotate.

The global variables *MIN_SCAN_ANGLE_RAD* and *MAX_SCAN_ANGLE_RAD* were modified to be ±60º (rather than ±10º) to allow a greater field of view of the robot (in radians). This was done to allow the robot to detect obstacles at a small incident angle, to prevent the robot from crashing into obstacles.

Laser Callback Method

The *_laser_callback()* method acts as a way to handle messages passed by the LaserScan. The structure of the method is as follows. Given that the flag is not already set, the minimum distance (*min_distance*) to the nearest obstacle is set to the maximum range of the sensor (*msg.range_max*). To determine the minimum and maximum indices of the LaserScan corresponding to the minimum and maximum scan angles (*MIN_SCAN_ANGLE_RAD, MAX_SCAN_ANGLE_RAD*), the middle index is determined according to the total number of indices (the length of *msg.ranges*). This value is used to calculate the minimum and maximum indices by the following formulas:

$$min\_index = int(mid\_index + self.scan\_angle[0] \,/\, msg.angle\_increment)$$
$$max\_index = int(mid\_index + self.scan\_angle[1] \,/\, msg.angle\_increment)$$

Following this, the minimum range value between the minimum and maximum scan angles is determined by cycling through *min_index* to *max_index*. If a given range value is less than the current minimum, the minimum is reset via *min_distance = msg.ranges[i]*. The program further checks to ensure that the minimum range value is within the appropriate range (*msg.range_min < msg.ranges[i] < msg.range_max*), as indicated by the documentation for LaserScan.

If the minimum range value (*min_distance*) is less than the threshold value (*min_threshold_distance*), the flag is set.

Spin Method

The *spin(self)* method acts as a way to handle the state of the flag; if the flag is not set, the robot moves forward, otherwise the robot rotates a random angle between *-pi* and *pi* and unsets the flag.

If the flag is not set (*not self._close_obstacle*), the *move()* method is called with the appropriate linear velocity, and an angular velocity of zero (*self.move(self.linear_velocity, 0)*).

If the flag is set, a random angle is determined using the *random.uniform()* method, and this is converted into the appropriate *duration*, according to the angular velocity. The *duration* is used to specify the length of time that the robot should rotate with a given angular velocity to reach the desired angle. The rotation of the robot is determined according to the specifications; if the value of the angle is in the interval *(0, pi)*, it rotates counterclockwise (positive angular velocity) and if the value of the angle is in the interval *(-pi, 0)*, it rotates clockwise (negative angular velocity). After the robot has finished rotating, the flag is unset so that the robot moves forward.

Evaluation

To verify the accuracy/completeness of the program, various experiments were conducted with the given parameters, highlighted as follows.

Modifying LINEAR_VELOCITY & ANGULAR_VELOCITY

By increasing the linear velocity, the robot moves more rapidly in straight directions, which leads to issues in which the robot collides with obstacles. This mimics the real-world situation in which there is not enough time to read and interpret sensor data before making a decision.

Similarly, by increasing the angular velocity, the robot moves more rapidly when spinning around, which leads to issues in which the robot collides with obstacles. This is due to the fact that the robot takes an incremental step forward after spinning, as the flag is unset before the obstacle is detected again.

By decreasing the linear and angular velocities to an appropriate value, *0.2 m/s* and *pi / 4 rad/s*, respectively, the robot does not encounter any issues in colliding with obstacles.

Modifying MIN_THRESHOLD_DISTANCE

By decreasing the minimum clearing distance, the robot moves more closely to obstacles before spinning. Thus, there are issues in which the robot collides with obstacles, typically when spinning in place.

By increasing the minimum clearing distance, the robot is guaranteed to not run into issues in which it collides with obstacles. However, the minimum clearing distance cannot be increased so much to the point that it prevents the robot from navigating through the obstacles, particularly small openings. The initial value of *0.5 m* is appropriate for this context.

Modifying MIN_SCAN_ANGLE_RAD & MAX_SCAN_ANGLE_RAD

By increasing the minimum and maximum scanning angle (to ±60º rather than ±10º), the program allows for a greater field of view of the robot. As indicated, this was done to allow the robot to detect obstacles at a small incident angle, to prevent the robot from crashing into obstacles. Increasing the field of view does not seem to have any significant (negative) impact on the complexity or runtime of the program operation, thus it is advantageous to modify this variable.

By decreasing the minimum and maximum scanning angle, the robot scans only within a limited field of view, which leads to issues in which the robot collides with obstacles. This is due to the fact that the algorithm disregards obstacles placed at the exterior of the sensor view (when limiting FOV).

Conclusion

The program functions as expected, according to the specifications. The robot does not seem to encounter difficulties navigating the obstacles, and rarely (if ever) runs into issues in which it collides with obstacles. Further, the robot does not become stuck when incident with a corner; the algorithm implemented typically results in a double rotation after the obstacle is detected after the first spin.

Credits

Alex Fick (*April 3, 2023)* → High-level discussion regarding the *_laser_callback(self, msg)* function, specifically various ways to find the minimum and maximum index.

Kevin Cao (*April 3, 2023)* → Discussion regarding the results of the program, alongside a cursory comparison of the visual output.

Laser Scan Documentation → http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html