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

Assignment: PA1 - Report

Author: Carter Kruse

Date: April 13, 2023

Implementation / Description

Overview

The program is a simple ROS node that allows the robot to draw various shapes, including trapezoids, "d" shapes, and polylines/polygons.

Trapezoid

The robot draws a simplified "D" represented as an isosceles trapezoid. The robot starts at the center of the longer base, with the robot local frame x-axis perpendicular to it and the vertices of the shorter base vertices are determined by the vectors of radius r and angle $+/- 45$ *degrees*.

"D" Shape

Instead of the trapezoid, the robot draws a proper "D" where the right stroke is represented with a half circle centered at the starting location of the robot with radius r.

Polyline / Polygon

The code is generalized to allow the robot to draw general arbitrary polygonal shapes. The coordinates of the vertices of such a polygon are given as input (represented as a polyline) in the global reference frame odom. The terminal displays the points of the polyline in a local reference frame, where the origin is on the first point of the list, and the x-axis is aligned with the line segment determined by the first two points.

The terminal also shows the 2D homogeneous transformation matrix in 2D between the two frames (from local to global).

Design Decisions

In developing the program, various design decisions were made, primarily to ensure the accuracy/completeness of the code. Specifically, the angle and distance tolerances had to be closely set, so that the robot would rotate or move linearly as expected, allowing room for error.

Rather than simply use an angular or linear velocity, along with a time (which leads to inaccuracies), the program incorporates a closed-loop that continues to operate the robot until a certain specification is met. This is where the tolerances come into play.

By using this method, rather than the velocity/time method, we are able to ensure that the error in the robot's movements does not carry forward, which would greatly impact the positioning of the robot.

Primary Method

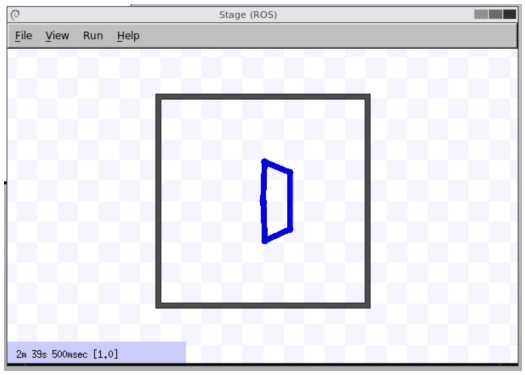
The main method used throughout the program is the *move_to_position()* method, which incorporates a rotation, translation, rotation algorithm. In particular, the robot does not overshoot the rotation specified, as the angular velocity remains relatively low (compared to the tolerance), and we allow the robot to make adjustments in the case of overshooting. The robot does not overshoot the position/location specified, as the linear velocity remains relatively low (compared to the tolerance).

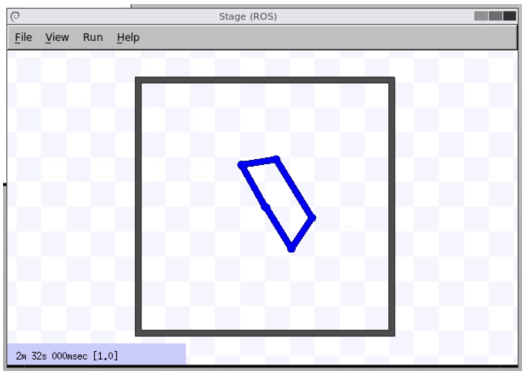
Evaluation

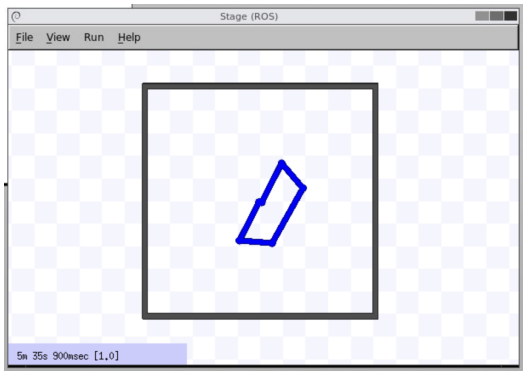
To verify the accuracy/completeness of the program, various experiments were conducted, with given parameters, highlighted as follows.

TRAPEZOID

The trapezoid method uses two parameters, *radius* and *start_angle*, which determines the way in which the robot draws out the trapezoid. Given various arguments, the program functioned as expected. Examples of the output are as follows (for angles *0, pi / 6,* and *-pi / 6*).

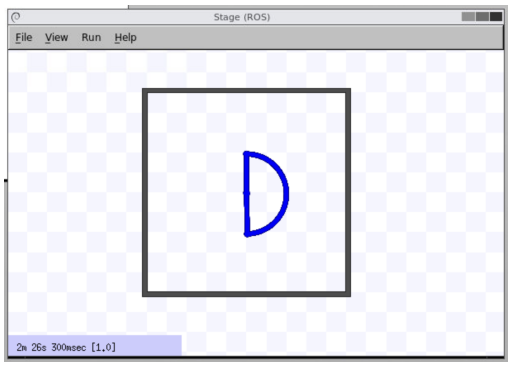


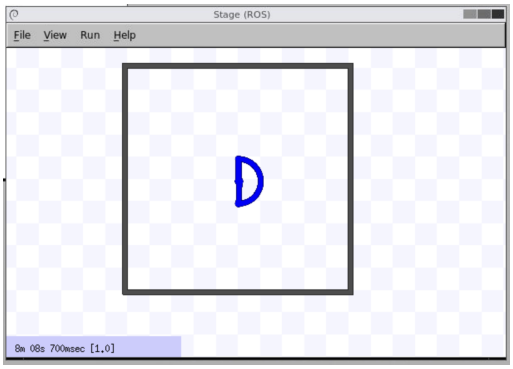




D SHAPE

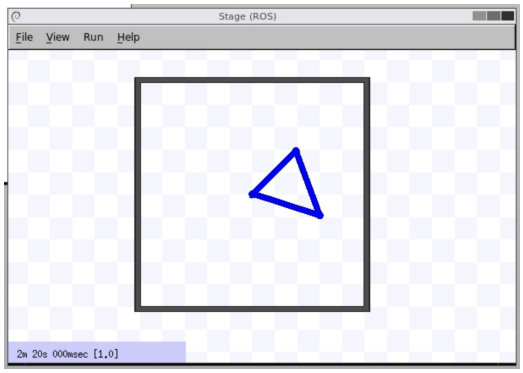
The "D" shape uses a single parameter, *radius*, which determines the way in which the robot draws out the "D" shape. Given various arguments, the program functioned as expected. Examples of the output are as follows (for radii *2* and *1*).

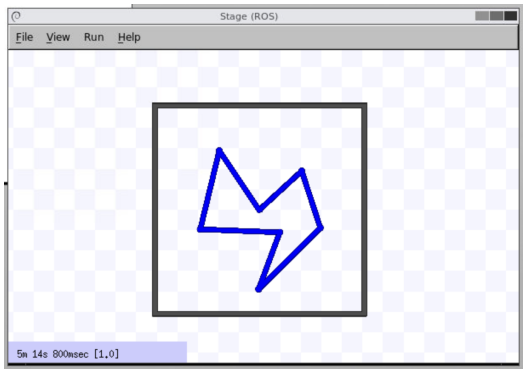




POLYLINE / POLYGON

The polyline / polygon uses a single parameter, *points*, which contains all of the points to be visited by the robot. Given various arguments, the program functioned as expected. Examples of the output are as follows.





The transformation matrix shown corresponds to the polyline/polygon on the right. This was the corresponding output to the terminal.

```
Transformation Matrix:
[[ 0.70710678 -0.70710678  0.          5.5       ]
 [ 0.70710678  0.70710678  0.          5.5       ]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]

Points:
(0.0, -0.0)
(2.828, 0.0)
(1.414, -2.828)
(-2.828, -2.828)
(0.0, -1.414)
(-2.828, 1.414)
(0.707, 3.536)
(0.0, -0.0)
```

Conclusion

The program functions as expected, according to the specifications.

The robot does not seem to encounter difficulties, in that it does not run into issues in which it collides with obstacles, if the appropriate positions are given. Further, the robot does not become stuck, oscillating between positions or rotations; the algorithm implemented includes the appropriate tolerances for the angle and distances.

Credits

Quaternion/Euler Angles Documentation
https://docs.ros.org/en/melodic/api/tf/html/python/transformations.html#tf.transformations.euler_from_quaternion