Lab Report for Lab 5: Mapping & Path Planning

Pedro Gonzalez-Guzman
University of South Florida
Date: 12/08/2024

Introduction

In Webots Lab 5, Task 1, students create a Python program to compute the robot's shortest path to a goal cell using Dijkstra's algorithm. This algorithm is combined with a predefined cell wall configuration and sensor inputs, such as data from cameras, an IMU, or LiDAR. The program employs a data structure to model the maze, detailing cell connectivity and obstacles. By leveraging the predefined wall configuration, the robot identifies and avoids blocked cells on its path to the goal. The testing scenario involves a 5x5 grid maze, referred to as Maze 8, as shown in Figure 5. Maze 8 consists of 25 cells with wall obstacles. The program predefines the shortest path and total distance to the goal before the robot begins its movement. As the robot progresses through the maze, the program outputs the cell coordinates, cell number, and the robot's orientation at each step.

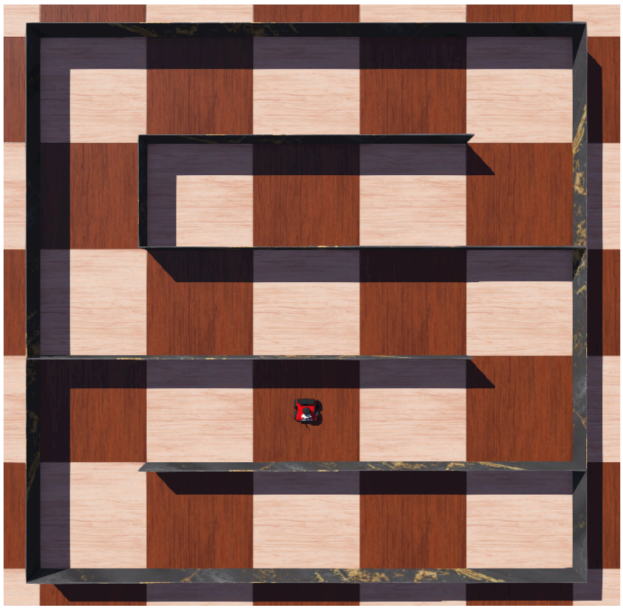


FIGURE 5. Maze environment for Task 2, with known wall configurations for Bayesian inference-based localization (maze file *maze8.xml*).

Task 1: Path Planning

Mathematical Computations

Students calculated the left and right servos velocities given the input parameters from the robot RGBD camera and LiDAR sensors. They apply the PID controller formulas, assumptions about the LiDAR range, tolerance distances, gain constants, etc. With PID control formulas the velocities of the left and right motors are calculated based on the error $e(t)$ between the robot's current distance from the goal or wall $y(t)$ and the target distance from the wall or goal $r(t)$.

Input parameters

# Known wall configuration
GRID_SIZE = 5 # 5x5 grid
CELL_SIZE = 1.0 # Each grid cell is 1x1 meter

# Cardinal Orientations
EAST = 0°
NORTH = 90°
WEST = 180°
SOUTH = 270°
CARDINALS = [EAST, NORTH, WEST, SOUTH]

# PID Controller Tolerances
THRESHOLD_FROM_WALL = 1 # distance from wall tolerance
ORIENTATION_TOLERANCE = math.radians(0.1) # setting orientation angle tolerance
MOVEMENT_TOLERANCE = 0.001 # PID movement tolerance

# Max and Min motor velocities
MAX_ROTATIONAL = 4.0 m/s
MAX_VELOCITY = robot.max_motor_velocity = 26 m/s
ADD_VELOCITY = 3.0 m/s

# Camera constants
CAMERA_WIDTH = robot.rgb_camera.getWidth()
CAMERA_CENTER_X = CAMERA_WIDTH / 2

$Optimal\ combination\ of\ gain\ constants$
$Kp_{rotat} = 8\,;\ Ki_{rotat} = 0.01\,;\ Kd_{rotat} = 2.0$ # robot specified orientation
$Kp_{forw} = 15\,;\ Ki_{forw} = 0.05\,;\ Kd_{forw} = 0.5$ # robot moving forward

$PID - DT = 0.032\,s$ $\#\ Time\ step\ for\ the\ PID\ controller\ in\ Webots$

```
# Maze8 Aadjacency List Configuration
CELL_GRAPH = {
    1: [ 2,  6],      2: [ 1,  3],   3: [ 2,  4],   4: [ 3,  5],   5: [ 4, 10],
    6: [ 1, 11],      7: [ 8    ],   8: [ 7,  9],   9: [ 8, 10],  10: [ 5,  9],
   11: [ 6, 12, 16],  12: [11, 13],  13: [12, 14],  14: [13, 15],  15: [14, 20],
   16: [11, 17, 21],  17: [16, 18],  18: [17, 19],  19: [18, 20],  20: [15, 19, 25],
   21: [16, 22],      22: [21, 23],  23: [22, 24],  24: [23, 25],  25: [20, 24]
}

# Maze8 Wall Configuration
WALL_CONFIG = [
   ['O', 'W', 'W', 'O'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['W', 'W', 'O', 'O'],
   ['W', 'O', 'W', 'O'], ['O', 'W', 'W', 'W'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['W', 'O', 'O', 'W'],
   ['O', 'O', 'W', 'O'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['W', 'W', 'O', 'O'],
   ['O', 'O', 'W', 'O'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['W', 'O', 'O', 'O'],
   ['O', 'O', 'W', 'W'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['O', 'W', 'O', 'W'], ['W', 'O', 'O', 'W'],
]
```

Servos Rotational and Linear Velocity:

Distance Error Formula:

The error is the difference between the current distance and the target distance for rotational and linear velocity: $e(t) = y(t) - r(t)$

- Rotational velocity PID Control is used for adjusting robot orientation after detecting the landmarks and before moving forward to the next cell. Also, for centering the landmark in the camera frame using the function *goal_object.getPositionOnImage()*.
  Where:
  $e(t): Distance\ Error$
  $r(t): Camera\ Center$
  $y(t): Camera\ Landmark\ Position$

- Linear velocity PID Control is used to move the robot forward through the cells, $based\ on\ the\ encoder\ readings$
  Where:
  $e(t): Distance\ Error = Target\ distance - Distance\ traveled$
  $r(t): Target\ distance$: $CELL\_SIZE$ $when\ moving\ forward\ or\ the\ returning\ distance\ to\ the\ Starting\ Position$
  $y(t): Distance\ traveled\ to\ reach\ the\ target\ distance$

PID Velocity: Distance Control Formula:

The PID controller calculates the control signal using the gain constants Kp , integral Ki, and derivative Kd terms.

Note: These formulas can be used for the rotational and linear velocity PID control calculation:

$$u(t) = Kp \cdot e(t) + Ki \cdot \sum_{0}^{t} e(t)\,dt + Kd \cdot \frac{e(t) - e(t-1)}{dt}$$

For programming purposes, students follow the Formulas:

$$u(t) = P(t) + Ki \cdot I(t) + Kd \cdot D(t)$$

$$P(t) = Kp \cdot e(t)$$

$$I(t) = I(t - 1) + e(t) \cdot dt$$

$$D(t) = \frac{e(t) - e(t-1)}{dt}$$

Where:

$u(t)$: $Motor\ Controlled\ Velocity$

$e(t)$: $Distance\ Error$

$I(t)$: $Integral\ term$

$D(t)$: $Derivative\ term$

$Kp,\ Ki\ ,\ Kd$: $gain\ constants$

Saturation Motor Control Velocity:

$$u_r(t) = f_{Sat}(u(t))$$

$$f_{Sat}(u(t)) = u_r(t) = c_{max}, \quad if\ u(t) > c_{max} \quad \text{or}$$

$$= u_r(t) = u(t), \quad if\ c_{min} \leq u(t) \leq c_{max} \quad \text{or}$$

$$= u_r(t) = c_{min}, \quad if\ u(t) < c_{min}$$

Where:

$u_r(t)$: $Motor\ velocity\ saturated\ control\ function$

$f_{Sat}(u(t))$: $Saturation\ motor\ velocity\ control\ functions\ limited\ by\ constants\ c_{max},\ c_{min}$

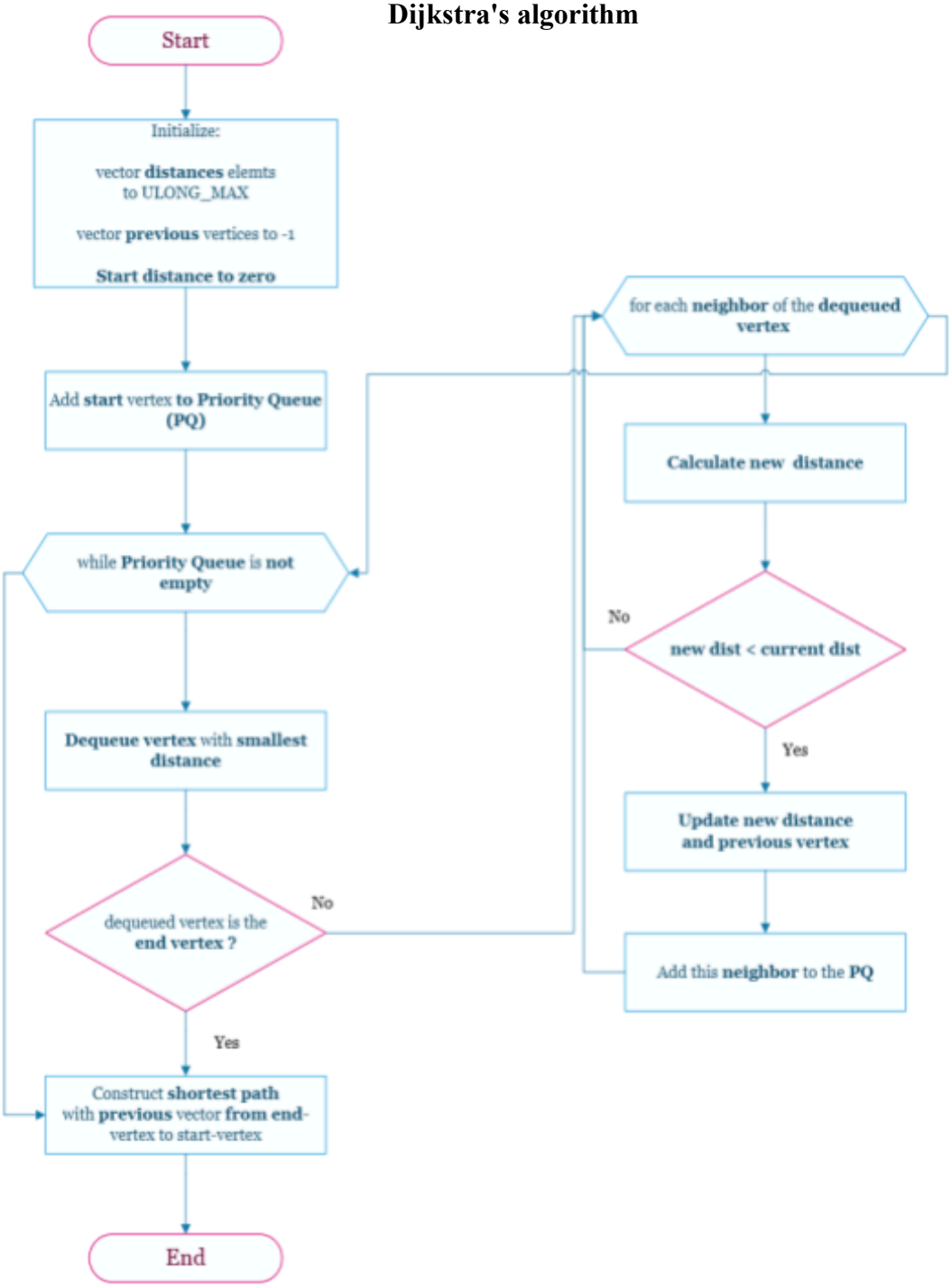
$c_{max},\ c_{min}$: $max\ and\ min\ motor\ velocities$

$Note:\ c_{max},\ c_{min}\ are\ different\ for\ rotational\ and\ linear\ velocities).\ See,\ input\ parameters.$

Determine the Robot's Shortest Path and Distance to the Goal:

In Task 1 students implement a solution for navigating a robot along the shortest path from a starting cell to a specified goal cell within a maze 8. The shortest path is found through Dijkstra's algorithm implementing a priority queue. The priority queue always keeps on its top the cell that accumulates the shortest distance from the start by comparing all cell distances.

Dijkstra's algorithm



Results

The linear PID servomotor velocity varies from 16.0 to 4.0 m/s, approximately.

The rotational PID servomotor velocity when the robot sets its orientation is 4.00 m/s.

Examples- Robot's Shortest Path and Distance to the Goal:

INFO: pedrog_Lab5_task1: Starting controller: C:\FAIRIS-Lite\venv\Scripts\python.exe -u pedrog_Lab5_task1.py

Program Start.

Goal shall be cell number: 7

Path: 10|9|8|7

Shortest Distance: 3.00 meters

Navigation: W W W

Pose Estimation:

(x:2.0, y:1.0, n:10, theta:90)

(x:1.0, y:1.0, n:9, theta:180)

(x:0.0, y:1.0, n:8, theta:180)

(x:-1.0, y:1.0, n:7, theta:180)

Destination has been reached.

Total travel time: 39.89 seconds

INFO: 'pedrog_Lab5_task1' controller exited successfully.

INFO: pedrog_Lab5_task1: Starting controller: C:\FAIRIS-Lite\venv\Scripts\python.exe -u pedrog_Lab5_task1.py

Program Start.

Goal shall be cell number: 7

Path: 25|20|15|14|13|12|11|6|1|2|3|4|5|10|9|8|7

Shortest Distance: 16.00 meters

Navigation: N N W W W W N N E E E E S W W W

Pose Estimation:

(x:2.0, y:-2.0, n:25, theta:90)

(x:2.0, y:-1.0, n:20, theta:90)

(x:2.0, y:0.0, n:15, theta:90)

(x:1.0, y:0.0, n:14, theta:180)

(x:0.0, y:0.0, n:13, theta:180)

(x:-1.0, y:0.0, n:12, theta:180)

(x:-2.0, y:0.0, n:11, theta:180)

(x:-2.0, y:1.0, n:6, theta:90)

(x:-2.0, y:2.0, n:1, theta:90)

(x:-1.0, y:2.0, n:2, theta:0)

(x:0.0, y:2.0, n:3, theta:0)

(x:1.0, y:2.0, n:4, theta:0)

(x:2.0, y:2.0, n:5, theta:0)

(x:2.0, y:1.0, n:10, theta:270)

(x:1.0, y:1.0, n:9, theta:180)

(x:0.0, y:1.0, n:8, theta:180)

(x:-1.0, y:1.0, n:7, theta:180)

Destination has been reached.

Total travel time: 209.54 seconds

INFO: 'pedrog_Lab5_task1' controller exited successfully.

Conclusions

Task 1 has been completed, demonstrating the robot's ability to accurately compute the shortest path and distance between two cells within a maze environment. This achievement highlights the integration of advanced path-planning techniques and sensor-based navigation. Throughout the lab, students gained valuable hands-on experience in implementing and managing various data structures, which were essential for representing the maze layout, including cell connectivity and obstacle information.

The use of Dijkstra's algorithm allowed students to explore its practical application in robotics, enabling the robot to navigate effectively Maze 8 with a predefined wall configuration. By incorporating real-time sensor readings from cameras, IMU, and LiDAR, students learned how to enhance the robot's situational awareness, refine obstacle detection, and adapt its path planning accordingly.

This lab not only solidified the theoretical understanding of algorithms and data structures but also emphasized their importance in solving real-world problems. Furthermore, it provided insights into the integration of robotics hardware and software, fostering skills in algorithm optimization, sensor fusion, and navigation in constrained environments. These experiences prepare students for more complex robotics challenges, where accurate and efficient path planning is crucial.

Video

Task 1 Video:

Authorship Statement

I developed the code and wrote the report.

Pedro Gonzalez Guzman: , 12/08/2024