Optional Lab
Robot Path Planning

Instructor: Dr.Charbel Azzi
MTE 140 - Algorithms and Data Structures
University of Waterloo

Due: 8:00 PM, Wednesday March 26, 2025

1 Lab Objectives

In this lab, you will develop a generic path planning algorithm using Doubly Linked List and Dynamic Memory Allocation with a VEX Robot moving on planet Mars.

2 Lab Description

You have successfully sent a robot to planet Mars for exploration. On Mars, the robot does not have any human assistance nor a full map of the planet, making exploring Mars is tricky. One challenge in robotics is called path planning. This is the process of the robot figuring out how to navigate between two points within some environment.

The goal is to get the robot to a safe location marked on the map by carefully planning its path. The robot will encounter waypoints (obstacles) which have been marked on the map. The robot has to move in small steps, check its surrounding make sure it is clear, then continue moving. If the steps are too big, the robot might ran into obstacles like rocks and damage itself beyond repairs.

3 Map Description

NASA have provided us a map of a small area it mapped on Mars. The map has 4 marked locations as dangerous due large rocks, trenches, or sand dunes. The robot has to find its way around them to get to the final destination. In this assignment, you will implement a simplified path planning algorithm for a robot moving in a simple 2D map mimicking the area on planet Mars.

The map can be described with the following:

1. Starting location $(0, 0)$: the robot (dashed lines) needs to be facing down in the middle of the orange zone. Note that the positive x-y axes indicate East and North directions respectively
2. Black empty squares: represent an empty or open space where the robot can safely move. Each square represent one unit - in this map $1unit = 2cm$
3. Red squares: represent the location of the obstacles - in this assignment you will need to place the 4 Lego obstacles in their assigned location on the map
4. Blue Square: represents the final safe location the robot needs to reach. In this project it will be $(18, 18)$ units

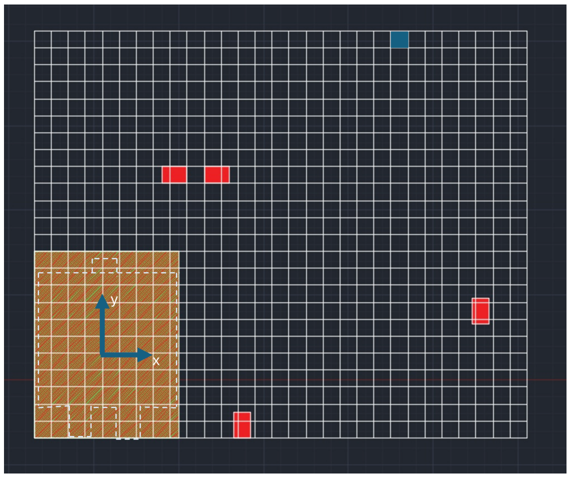


Figure 1: Map

4 Path Planning Algorithm

4.1 Doubly Linked Lists

First we will explain the doubly linked list node and class implementation.

The doubly linked list node should have:

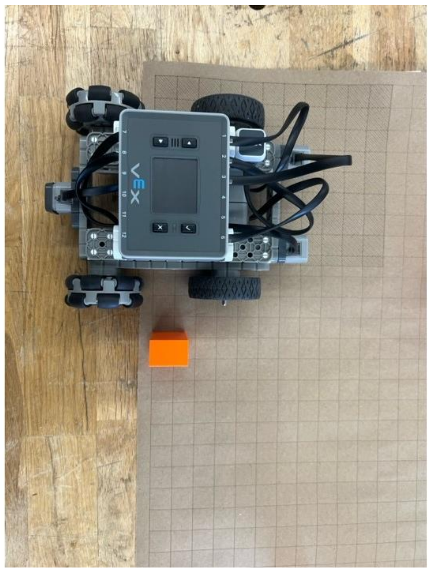
1. x (East) and y(North) location of the robot in units i.e. if the robot is as (3, 2) that means it is (6, 4) cm
2. Node Type: Specify if this is a necessary node or a regular node. Necessary nodes are: 1) Start location Node, 2) Turning Node, and 3) Object Detection Node. A regular node is a node belonging to an empty square.
3. Previous pointer
4. Next pointer

Your doubly linked list class should have the following members: **head, tail, size, and capacity**. It should also constructors (**max capacity is set to 10** in this lab) and destructor, **insert, and remove** function. The insert function will insert nodes at the end of the list, whereas the remove function will take in an index and remove the node (or multiples nodes). The implementation of the insert and remove will be explained later.

4.2 The Algorithm

After creating your doubly linked list class, we now explain the path planning algorithm that implements the doubly linked list.

1. **Start from the Starting Location:**
   1.1. Begin at coordinates (0,0) as shown in Figure 1
   1.2. Call the provided `cali_inertial()` function to calibrate the IMU (see appendix)
   1.3. Set your north angle to 180 degrees, i.e., your robot needs to be in the positive y direction as its starting position as shown in the figure
   1.4. Configure the drivetrain speed (or motor speed, depending on your setup) to 10 RPM.


Starting Position after completing step 1

2. **Movement Restrictions:** You may only move in 2 directions:
   (a) North (180 degrees), in the positive y direction
   (b) East (270 degrees) in the positive x direction

3. **Movement Priority:**
   (a) Always move towards the final safe destination $(x_f, y_f) = (18, 18)$ units by prioritizing the direction that brings you closer to the target. e.g., if $x_{current} - x_f > y_{current} - y_f$ move towards y_f until this condition change
   (b) If both the x and y coordinates are equally distant from the destination (x_f, y_f), prioritize moving in the x direction first (East) before the y direction (North)
   (c) Move in 1-unit intervals, where 1 unit equals 2 cm

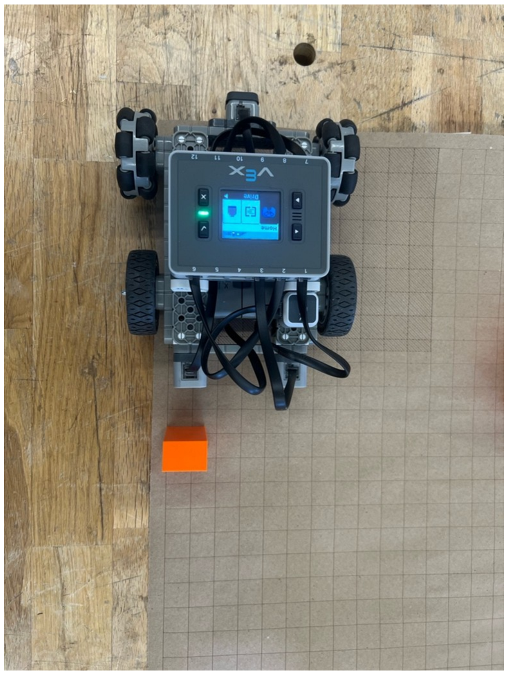
4. **Pathing & Planning:**

   4.1. Robot moves into an empty squared according to the movement rules in steps 2 and 3

   4.2. After every movement, create a new data point according to Step 5

   4.3. Repeat steps 4.1 & 4.2 until one of the following conditions is triggered:

      i. Turning → Go to Step 6
      ii. Object Detected → Go to step 7
      iii. Max Capacity Reached → Go to step 8
      iv. Arrived at Final Destination → You have successfully Pathed Mars!


Robot detecting an object after having moved East due to prioritizing x_f

5. **Data Point Creation:**

   (a) After every movement, create a new data point i.e. a new doubly linked list node

   (b) Make sure you to specify if it is necessary node or regular node (according to the description in Section 4.1)

   (c) Print the newly node on the robot screen

6. **Turning:**

   (a) **Movement Priority Trigger:**
   If a movement priority from Step 3 changed then a turning is triggered:

      i. Update the Doubly-Linked List → Go to step 8
      ii. If robot was prioritizing x_f and that condition changes → Go to step 9 to turn North to prioritize y_f
      iii. If robot was prioritizing y_f and that condition changes → Go to step 9 to turn East to prioritize x_f
      iv. Resume moving → Go back to Step 4.1

   (b) **Object Detection Trigger:**
   If an object is detected from Step 7 then a turning is triggered:

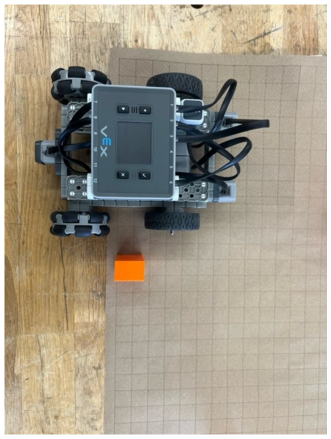
i. Update the Doubly-Linked List → Go to step 8
ii. Go to step 9 to turn North or East to avoid the object
iii. Move to a safe distance (3-4 units i.e. 6-8 cm) away from the object
iv. Resume moving → Go back to Step 4.1

7. **Object Detected:**

   7.1. Continuously monitor any detected objects

   7.2. Mark the object as detected when the distance between the robot and object is $\leq=$ 2 units

   7.3. Stop the robot → Go to step 6



8. **Update the Doubly Linked List:**

   (a) **Capacity Trigger**: If your Doubly Linked List hit capacity (10):

      i. Delete all regular nodes along that straight path between the current position and the latest/newest necessary node
      ii. You must not delete any necessary node
      iii. Print the current Nodes after deletion on the robot screen

   (b) **Turning Trigger**: If you triggered a Turning condition from Step 6:

      i. Delete all regular nodes along that straight path between the current necessary node triggering the turn and the previous one
      ii. You must not delete any necessary node
      iii. Print the current Nodes after deletion on the robot screen

9. **Make a Turn (East or North)**:

   (a) Turn the robot 90 degrees in the appropriate direction

   (b) Read the angle after turning using $imu.angle(degrees)$ (you will notice the robot turned more then 90 degrees)

   (c) Correct the angle by subtracting the $imu.angle(degrees)$ from 90 in that direction

5 Deliverables

Please submit the following 2 files to LEARN:

1. **Your code:** please write comments ans feedback to help us know where you struggle and if you have any suggestions for improvements

2. **A demo video:** a demo video showing your progress (as far as you reached if you did not finish it fully). The video should show the robot pathing and the printed nodes while pathing

Appendix

```cpp
void cali_inertial() {
  left_motor.setVelocity(15, rpm);
  right_motor.setVelocity(15, rpm);

  imu.calibrate();
  while (imu.isCalibrating()) {}
  wait(1, seconds);
  Drivetrain.turn(right);
  wait(400, msec);

  while (imu.angle(degrees) < 166.0) {}
  Drivetrain.stop(brake);
  wait(1, seconds);
}
```