# COMP329 Programming Assignment

Java - utilises Java utility libraries List and ArrayList.

**Adam Cain - 201572027**

## Abstract Navigation

I developed an abstract Navigation class to encapsulate common functionalities shared by both the Bug algorithms, with subclass-specific logic being implemented for movement state transitions and specific logic. This superclass includes methods for wall-following, robot alignment to walls, linear movements and other common functions.

A method was made for moving the robot towards the target. This would check if the robot is facing the robot by continuously calculating the difference in the robot's pose theta and the bearing between the robot and target. It then rotated the robot until it was within the bearing. Then move forward until it detects a wall in front of it on its forward facing sensors.

Once a wall is detected it then aligns the robot's side, depending on the configuration, to the right or left. By rotating the robot until the configured sides sensors are approximately equal, then starting the wall following algorithm.

The movement of the robot in the previous cases are inversely related to its target state, accelerating when distant from its intended state and decelerating as it nears it. Such as the robot moving faster when far away from a wall or turning faster when the difference between sensors are higher.

To improve wall-following from Lab 4, I altered the sensor usage for wall detection. Previously, the robot used the outermost sensor on the opposite side of wall-following, but this occasionally led to tracking unintended obstacles, especially problematic in the Bug 1 algorithm where it hindered the robot from returning to its original path. Additionally, I refined how the robot measures its distance from the wall. Instead of relying on the minimum sensor value, the robot now calculates an average distance for a more accurate assessment of its proximity to the setpoint. I also introduced a method to determine the robot's angle relative to the wall by comparing distances from the two side sensors, creating a secondary error metric. Combined with PID control, these enhancements significantly improved the robot's navigation accuracy in maintaining the desired setpoint and angle to the wall.

An abstract method 'step()' is implemented in the class which will be called on every timestep of the simulation. This is where the implementation of the state machine and logic for each different algorithm should be overridden and implemented in the subclass.

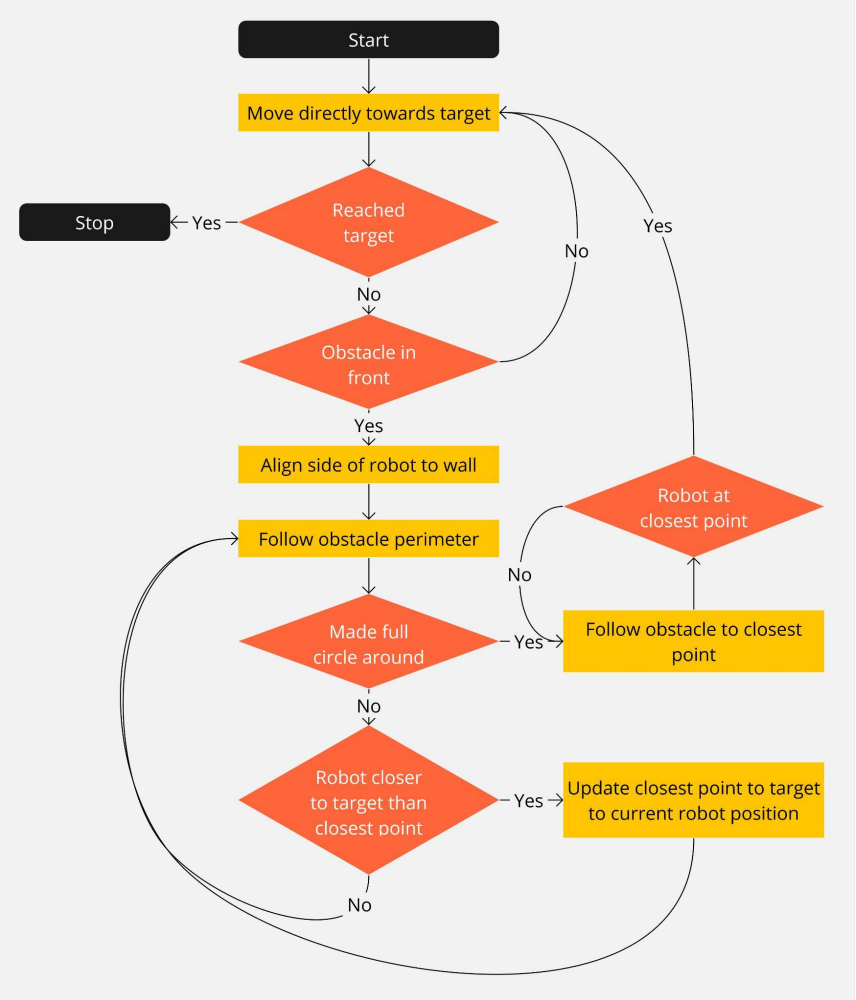
## Telemetry & Display

The dashboard displays the robot's position, orientation, sensor ranges, detected walls, and the robot's path in the arena graphically. It also indicates the start and end points. For the Bug 1 algorithm, it also highlights the closest point on an object's perimeter to the target

while following a wall. For Bug 2, it also displays the M-Line connecting the start to the target where the robot should terminate its wall following.

The distance the robot travelled is calculated using the position sensors from each motor, the distance from each sensor is converted from the angular value in radians to metres travelled by each wheel. An average for both wheels is then calculated to find an overall total distance the robot has travelled.

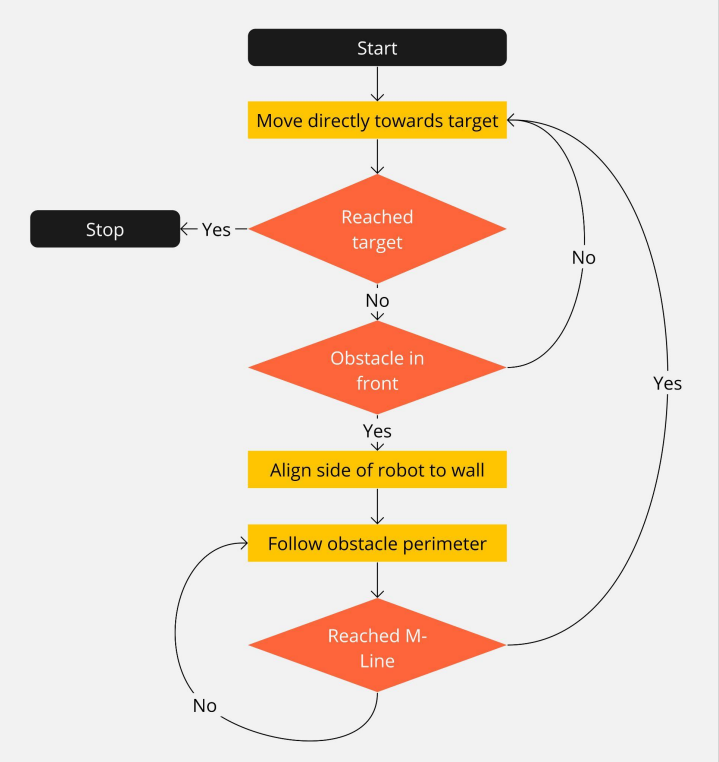
# Bug1




*Bug 1 Algorithm Logic Flowchart*

The Bug 1 implementation builds upon the navigation class by enhancing the wall-following functionality. It incorporates checks to determine if the robot's current distance to the target is shorter than the previous point. If so, the current distance becomes the new point. Additionally, this implementation includes mechanisms to detect when the robot completes a full loop around an obstacle. Upon completing, the robot retraces its path back to the closest point. After reaching this point, the robot resumes its original behaviour, moving towards the target.

# Bug 2

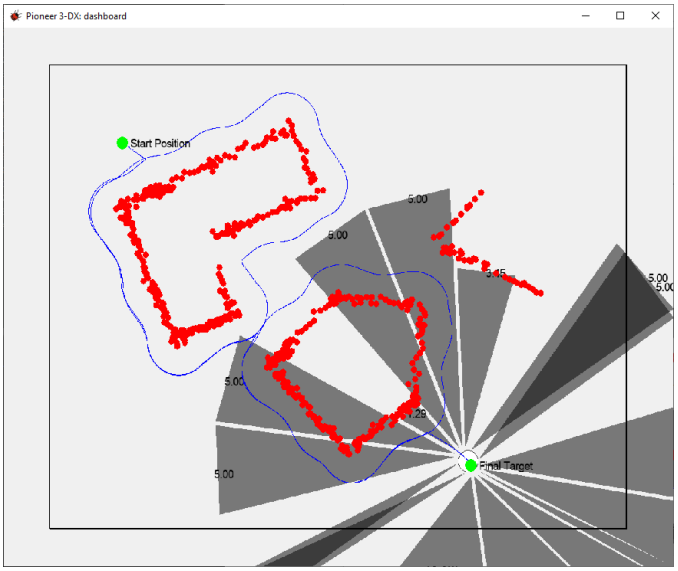
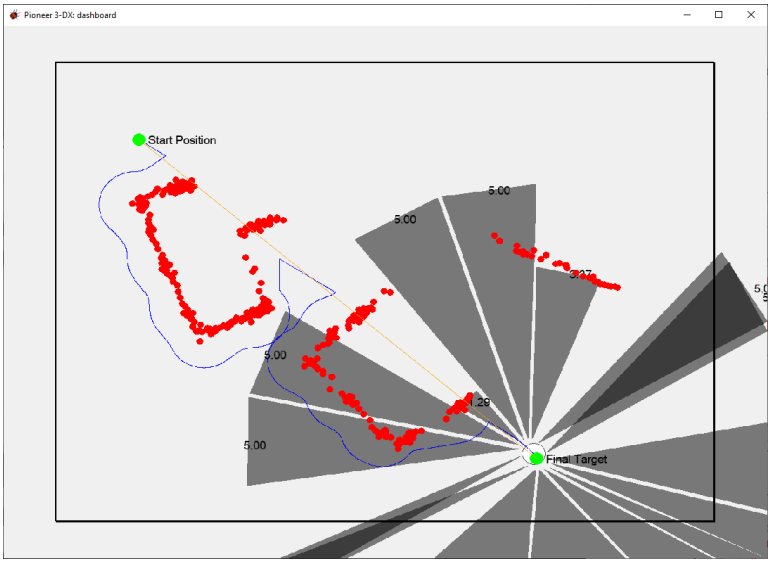
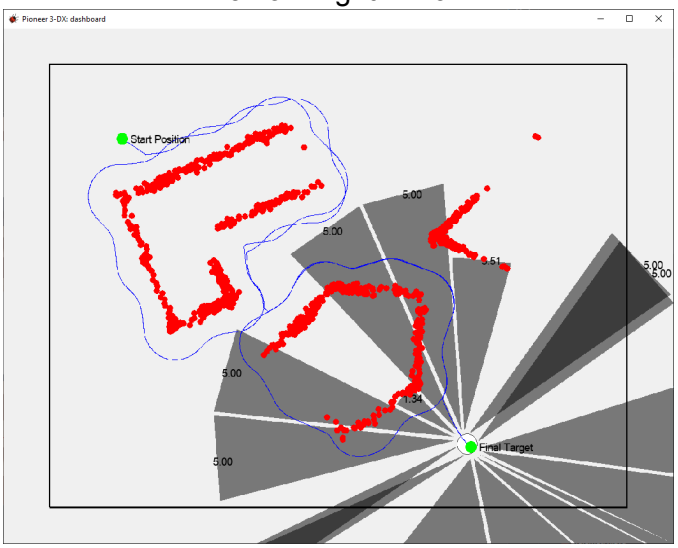
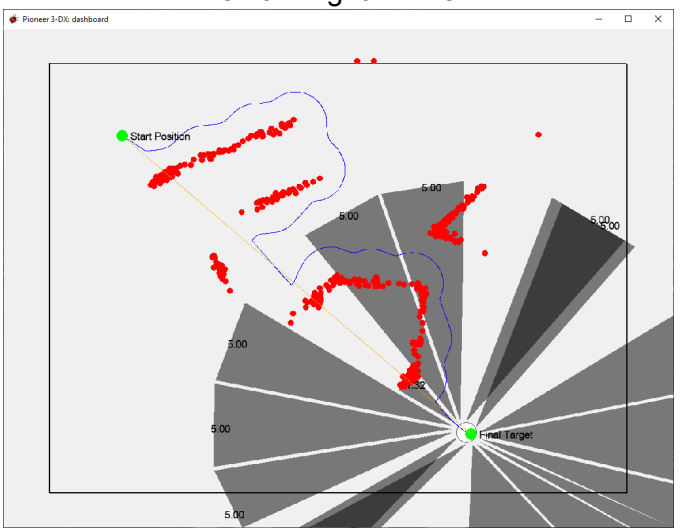



*Bug 2 Algorithm Logic Flowchart*

The implemented bug 2 algorithm is a lot simpler than bug 1. The only needed functionality was for checks if the robot had left where it started following the wall and if the robot was on the M-line. Which if it was it continued its original behaviour moving towards the target.

# Testing and Results

I evaluated both algorithms in the specified environment and manually fine-tuned the PID controller, using the robot's travel distance as a performance metric. Initially setting all PID values to zero, I gradually increased Kp to achieve a steady oscillation rate. Further adjustments to KI and Kp were made, considering factors like the bug algorithm used and which wall side the robot followed. These changes aimed to minimise travel distance, but sometimes led to issues such as overshooting or colliding with walls. The final PID values were Kp = 0.4, Ki = 0.0, and Kd = 0.0, with the robot's path and total distances below indicating their effectiveness.

| Bug 1 | Bug 2 |
| --- | --- |
| Wall on Left: 45m<br>  | Wall on Left: 18.56m<br>  |
| Wall on Right: 47.9m<br>  | Wall on Right: 17.25m<br>  |

# Reflection and Challenges

1. An initial challenge was a limited understanding of the Pioneer robot's sensors. I learned that its distance sensors use sonar and may not accurately detect distances if an object is at certain angles, further understood by looking at the Webots documentation.
2. A challenge arose from the final target, a cone, due to its irregular, small, and curved shape, making detection difficult for the sensors. I resolved this by removing the cone, though ideally, a point-in-polygon test would have been used to determine if the target was inside an object.
3. I encountered a telemetry challenge involving the display of polygons representing the sensor's detected range. The issue was that the coordinates extended beyond

the display's constraints, causing them to "wrap around" and give a bounced-off appearance. Although I attempted to use the Sutherland-Hodgman clipping algorithm to address this, it didn't yield better results. Consequently, I limited the polygon coordinates to fit within the screen's constraints.

4. Another challenge was during testing, tuning the PID controller took a large amount of time to test for each parameter such as either bug algorithm and the side of the wall the robot follows. I would ideally have automated this in some way so that multiple tests could have run to find best PID values possibly using the distance and if the robot made it to the final target as metrics to measure certain values success.