Project Report: Navigating a Mobile Robot Using Bug 1 & Bug 2 Algorithms in Webots

Author: Kevin Rahimi 201563237

Date: January 2024

Implementation Language: Java

Abstract

This report presents a detailed analysis of navigating a mobile robot from a given start location to a prescribed goal in the Webots simulation environment using Java-implemented controllers based on Bug 1 and Bug 2 algorithms. These algorithms are designed to navigate complex environments by combining movement-to-goal and boundary-following behaviours, adjusting the robot's path based on the distance & orientation to the goal.

Introduction

The challenge of autonomous robot navigation in environments with obstacles is a fundamental issue in robotics. The ability to reach a designated goal efficiently without collisions is crucial for various applications, from industrial automation to search and rescue operations. This report describes the implementation and comparative analysis of two algorithms, Bug 1 and Bug 2, designed to address this challenge.

Problem Statement

The aim is to navigate a mobile robot from a specified start location to a predetermined goal in the Webots software environment. The controller, written in Java, implements the Bug algorithms to combine motion-to-goal and boundary-following behaviours. The goal is achieved efficiently without obstruction by obstacles, adjusting the path based on the heuristic distance.

Implementation Overview

Bug 1 Algorithm (MyController.java):

- **Approach**: The robot moves towards the goal until it encounters an obstacle. It then circumnavigates the obstacle until returning to the initial encounter point, effectively walking around the obstruction. It marks a minimum point where the distance to the goal from the obstacle is the shortest and when it meets that point it switches states to moving towards the goal.
- **Environment & Sensor Integration**: The robot is equipped with a front sensor to detect obstacles. In the absence of obstacles, it moves directly towards the goal. Upon obstacle detection, it switches to boundary-following mode, circling around the obstacle to find the nearest point aligned with the goal.

Bug 2 Algorithm (MyController2.java):

- **Approach**: On encountering an obstacle, Bug 2 shifts to obstacle avoidance, following the obstacle's edge while calculating the line slope from its current position to the destination using theta to find the orientation of the robot (when 0 robot is facing the target). This continuous calculation allows for informed decisions on when to follow the wall and when to head towards the target.
- **Advanced State Management**: Unlike Bug 1, Bug 2 employs a sophisticated state management system, making it more efficient in pathfinding and reducing unnecessary circumnavigation of obstacles.

Detailed Analysis and Comparison

- **Motion Modes and Navigation States**: Both algorithms operate in 'Go to Target' and 'Wall Follow' modes. In 'Go to Target', the robot can be in 'FORWARD', 'ARC', or 'STOP'. 'FORWARD' corresponds to direct movement towards the goal, 'ARC' involves circumventing obstacles, and 'STOP' indicates a pause. In 'Wall Follow', the robot adheres to the 'Wall Follow' navigation state, moving along obstacle contours.
- **Performance Metrics**: Bug 2 outperforms Bug 1 in both time taken and distance traveled due to its real-time decision-making and state management capabilities. While Bug 1 is simpler and may be more robust in certain scenarios, Bug 2's efficiency makes it preferable in environments where speed and precision are vital. Bug 1 reached the goal in an average time of 227.5s* while travelling 65.5m* on average. On the other hand Bug 2 reached the goal in an average 72s while

travelling only 18m*. Therefore the Bug 2 algorithm is more efficient in this situation as it reaches the goal in less time and travels a smaller distance.

*average was calculated by running the simulator 5 times, results ranged only slightly.

Challenges Encountered and Resolutions

Implementing the Bug algorithms involved intricate tuning of PID parameters, selecting optimal threshold distances from walls, and implementing effective edge detection mechanisms. This required a rigorous process of trial and error, experimenting with various combinations of parameters to ensure smooth and efficient operation. The iterative process involved selecting values based on performance outcomes and adjusting them to optimise the robot's navigation behaviour. Also, Display issues with memory caused it to crash several times causing me to lose a lot of my work over and over. However, this issue was mostly solved by buying a new CPU, motherboard and 64 GB ram so now it should be working. I have kept the console prints as well just incase the display refuses to function when marking.

I had also found it hard to get the pioneer robot to take corners smoothly. I mostly resolved this issue by increasing the velocity at which it takes corners on one wheel. This reduced the robot overlapping the corner.

Another issue, I had was switching states, however, I had implemented something similar before where I use 2 enumerators, one in the controller class and one in the Nav class. For this project I used “Move States” in the nav class and I used “Global States” in the Controller class. I implement the method to move towards the goal in the controller class instead as when I tried to implement it in the navigation class it was hard to switch states.

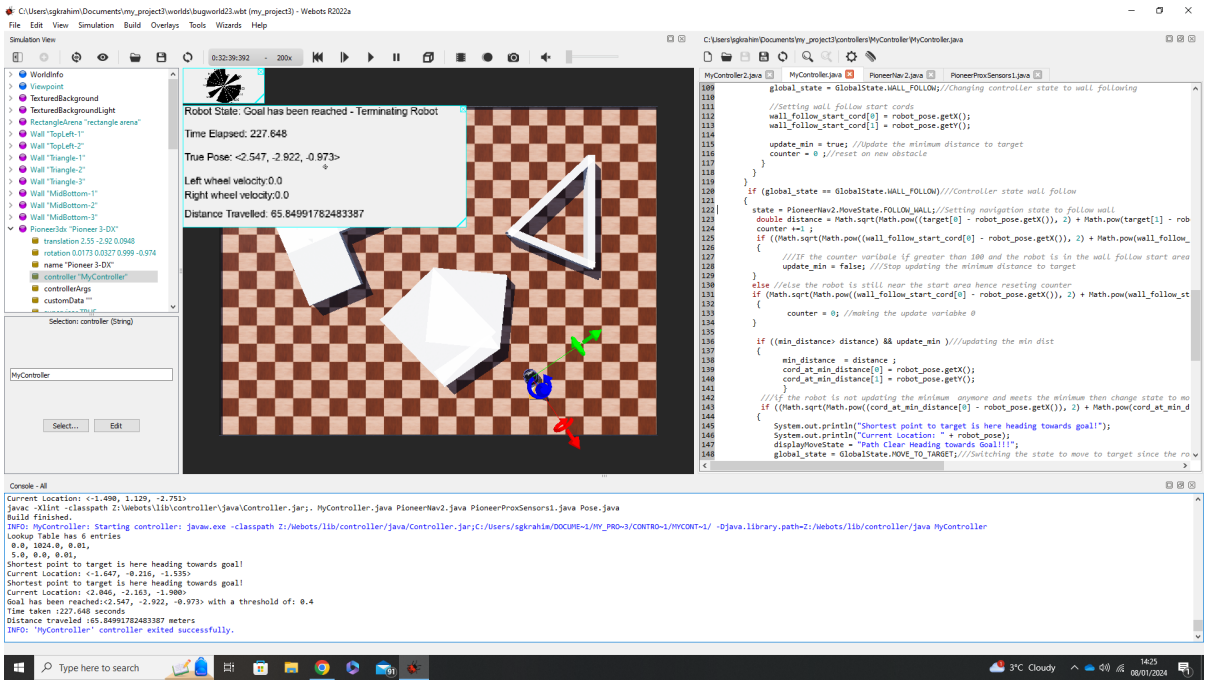
Reflection

The comparative analysis between Bug 1 and Bug 2 provides insights into the trade-offs between simplicity and efficiency in robot navigation strategies. While Bug 1's straightforward approach offers predictability and ease of implementation, Bug 2's advanced features lead to superior performance in terms of speed and distance. I would also use GitHub commits more as there was many times that I either completely butchered my code and had to start all over again or Webots decided to crash and leave me back at the start. To be optimistic, this also helped me in understanding the code more as I was writing the same algorithms over and over.

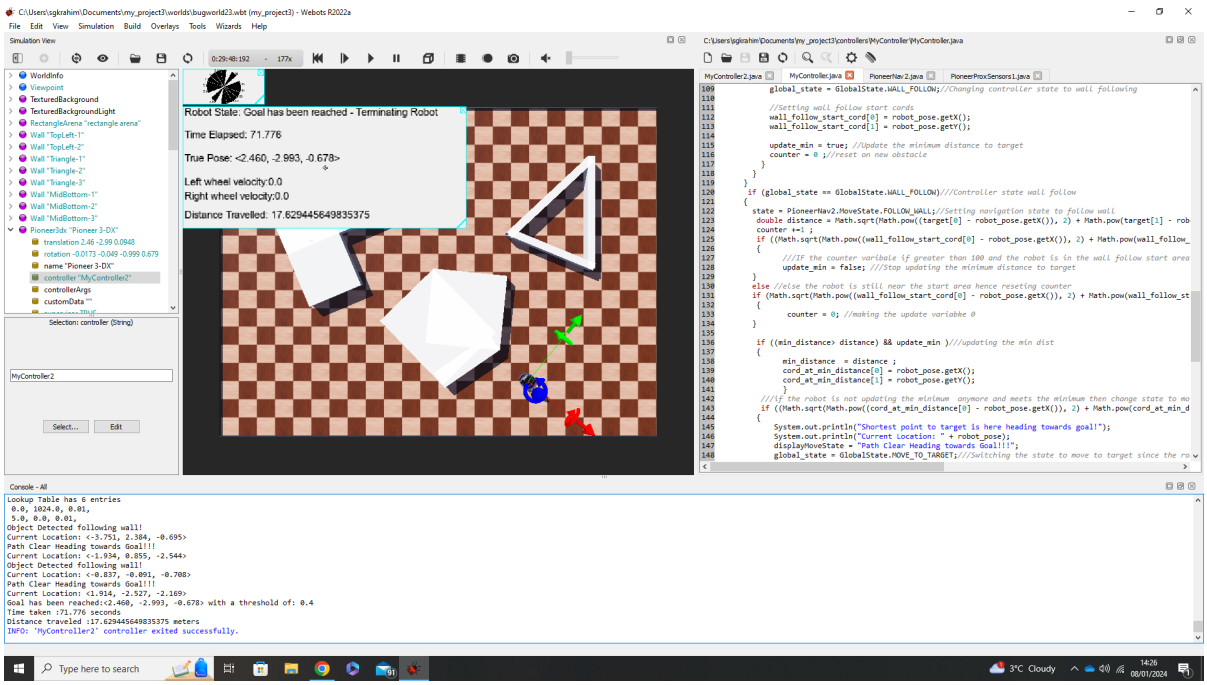
Conclusion

Both Bug 1 and Bug 2 algorithms exhibit the potential to navigate complex environments and reach designated goals efficiently. The choice between the two depends on the specific requirements of the environment and the desired balance between simplicity and efficiency. By understanding the strengths and limitations of each algorithm, developers can make informed decisions to optimise each algorithm to their needs.

▼ Screenshots from both solutions:



Bug 1 Algorithm



Bug 2 Algorithm