CS 382 – Minor Project Report
Development of Navigation Methods for Mobile Robots in Cluttered Environments

Department of Computer Science and Engineering

National Institute of Technology Meghalaya

Saitsohpen Sohra, East Khasi Hills, Meghalaya, India- 793108

A Project Report

Submitted by

Shivesh Kumar (B22CS038)

Gurijala Meghana (B22CS019)

Karipireddy Surya Teja Gopal Reddy (B22CS022)

Under the Supervision of

Dr. Ngangbam Herojit Singh

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would have not been possible without the kind support and help of many individuals. We extend our heartfelt gratitude to all of them. We are highly indebted **to Dr. Ngangbam Herojit Singh.** who by all means provided us with the resources, facilities and equipment for the experimentation and guided us throughout the process.

We would like to thank NIT Meghalaya for providing the funding to carry out this project. We thank the Head, all the faculty and staff members of Department of Computer Science and Engineering of National Institute of Technology, Meghalaya for their extreme help. This work could never have been possible without the inspiration from the previous works of those whose details are

Saitsohpen Sohra

MAY 2025

Shivesh Kumar **Gurijala Meghana** **Karipireddy Surya Teja**

CERTIFICATE

We hereby certify that the work which is being presented in the B.Tech. Project Report entitled **"Development of Navigation Methods for Mobile Robots in Cluttered Environments"** in partial fulfilment of the requirements for the award of the **Bachelor of Technology** in **Computer Science and Engineering** and submitted to the Department of Computer Science and Engineering of National Institute of Technology Meghalaya is an authentic record of my/our own work carried out during a period from **February 2025 to May 2025 (6th Semester)** under the supervision of **Dr. Ngangbam Herojit Singh.**

The matter presented in this Project Report has not been submitted by us for the award of any other degree elsewhere.

Shivesh Kumar	**Gurijala Meghana**	**Karipireddy Surya Teja**

This is to certify that the above statement made by the students is correct to the best of my knowledge.

Dr. Ngangbam Herojit Singh

Date:

ABSTRACT

This project tackles the problem of enabling robots to move around on their own, especially in messy and complicated settings that resemble real-world situations. The main part of this work involves creating, putting into action, and thoroughly testing several basic ways for robots to navigate. We use a simulation tool called Webots and a small, adaptable robot known as the e-puck. The ability to move around independently is very important for robots to be useful in many different areas, including factory automation, helping people, and search and rescue operations.

The project looks at four main ways for robots to navigate: following a line, where the robot has to stay on a set path; wall following, where the robot keeps a certain distance from a wall; obstacle avoidance, where the robot has to automatically steer around things in its way; and target reaching, where the robot goes from a starting point to a specific target location. For each of these methods, the project explains the basic ideas behind it, what needs to be considered when making it work in the Webots simulation, and what possible problems might come up, particularly in crowded areas.

A big part of the project is explaining the choices made in how to control the robot's movements and how it senses its surroundings. Different ways of controlling the robot and different ways for it to understand its environment were explored and used for each navigation method. The report looks at the trade-offs between these choices, such as how easy they are to use, how well they work, and how reliably the robot performs when things aren't perfect or there are disturbances.

In the end, this project aims to help us better understand the basic problems and potential solutions in the field of robot navigation. By systematically creating and testing these navigation methods in a simulated environment, the project tries to lay a strong foundation for future research and development focused on making robots more autonomous, reliable, and capable of operating in complex and changing real-world environments.

Table of Contents

Abbreviations

OAM	Obstacle Avoidance Module
LLM	Line Leaving Module
OFM	Obstacle Following Module
LEM	Line Entering Module
SLAM	Simultaneous Localization and Mapping
AMR	Autonomous Mobile Robot

LIST OF FIGURES

LIST OF TABLES

1.Introduction

Mobile robots are becoming increasingly important in various fields, including industrial automation, search and rescue, and home services. In factories, they help move materials, assemble parts, and inspect products with great accuracy, boosting efficiency and reducing labour costs. In emergency situations, robots can access dangerous or hard-to-reach areas to assist in rescue missions, using their sensors and navigation tools to gather vital information. At home, robots are taking on tasks like vacuuming, mowing lawns, and supporting elderly or disabled individuals. As their technology improves, they are making daily life more convenient and safer.

For mobile robots to be truly effective, especially in cluttered or dynamic spaces, they must be able to navigate on their own. This report focuses on developing and testing basic navigation methods using the Webots simulation platform and the e-puck robot. The e-puck's small size and rich sensor suite make it ideal for research and teaching in mobile robotics.

2.Simulation Platform

Webots is a powerful and easy-to-use simulation platform designed for developing and testing mobile robots in a virtual environment. It allows users to create realistic 3D worlds where robots can move, sense, and interact with their surroundings—just like in the real world. This makes it an excellent tool for students, researchers, and developers to experiment with robot behaviour, navigation, and control strategies without needing expensive hardware.

One of the best things about Webots is its support for many types of robots, including wheeled robots, drones, and humanoids. It also offers a wide range of sensors (like cameras, distance sensors, and GPS) and actuators (like wheels and motors) that you can easily add to your robot model. The platform uses physics-based simulation, so things like movement, collisions, and friction are realistically modelled.

You can program robots in Webots using popular languages like Python, C, C++, Java, or MATLAB, which makes it flexible for different users. Plus, its built-in 3D editor lets you design your robot or environment visually, without needing to write code from scratch.

Overall, Webots is a great tool for learning and experimenting with robotics in a safe and controlled virtual space before applying those ideas in the real world.

3.Robot

e-puck robot

E-puck is a miniature mobile robot originally developed at EPFL for teaching purposes by the designers of the successful Khepera robot. The hardware and software of e-puck is fully open source, providing low level access to every electronic device and offering unlimited extension possibilities.

The model includes support for the differential wheel motors (encoders are also simulated, as position sensors), the infra-red sensors for proximity and light measurements, the Accelerometer, the Gyro, the Camera, the 8 surrounding LEDs, the body and front LEDs, bluetooth communication (modelled using Emitter / Receiver devices) and ground sensors extension. The other e-puck devices are not yet simulated in the current model.

E-puck was designed to fulfilled the following requirements:

- *Elegant design*: the simple mechanical structure, electronics design and software of e-puck is an example of a clean and modern system.
- *Flexibility*: e-puck covers a wide range of educational activities, offering many possibilities with its sensors, processing power and extensions.
- *Simulation software*: e-puck is integrated with Webots simulation software for easy programming, simulation and remote control of the (physical) robot.

- *User friendly*: e-puck is small and easy to setup on a tabletop next to a computer. It doesn't need any cables, providing optimal working comfort.
- *Robustness and maintenance*: e-puck is resilient under student use and is simple to repair.
- *Affordable*: the price tag of e-puck is friendly to university budgets.

Characteristics	**Values**
Diameter	71 mm
Height	50 mm
Wheel radius	20.5 mm
Axle Length	52 mm
Weight	0.16 kg
Max. forward/backward speed	0.25 m/s
Max. rotation speed	6.28 rad/s

Table 3.1. E-puck Model

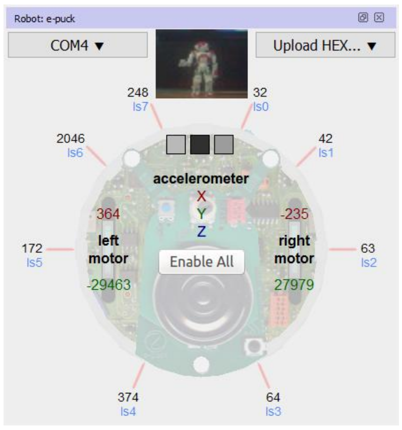


Fig 3.1 e-puck robot window

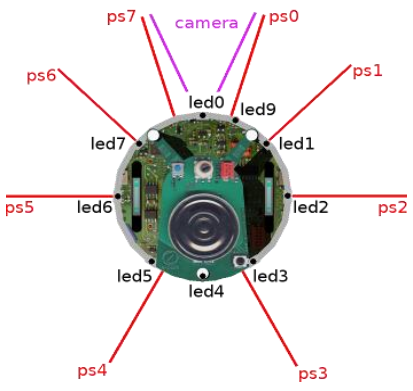
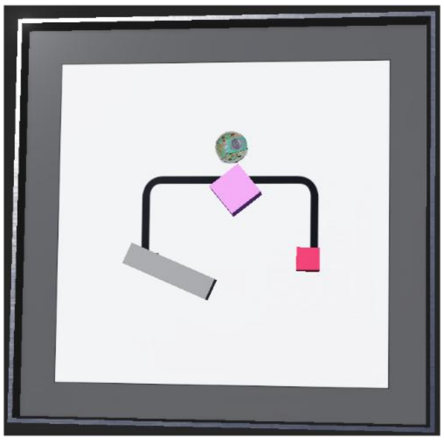


Fig 3.2 Sensors, LEDs and Camera

4.Navigation Methods

4.1. Line Follower

Objective:



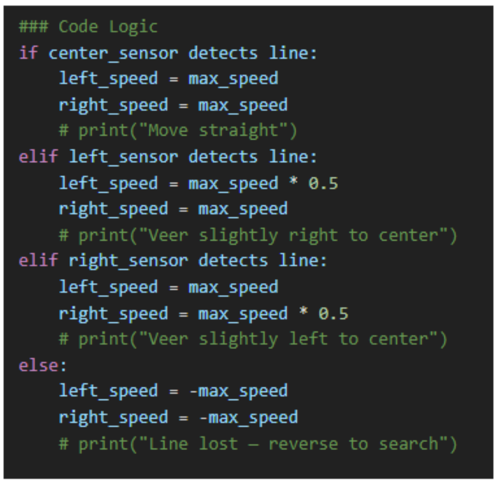
Design and implement a robot in Webots that can:

1. Follow a predefined black line on the ground using IR ground sensors.
2. Detect and avoid obstacles in its path using proximity sensors.
3. Return to the line once the obstacle has been passed.

Behavioural Modules:

The robot's behaviour is broken into modular components, each handling a specific situation:

1. Line Following Module (LFM):



```
### Code Logic
if center_sensor detects line:
    left_speed = max_speed
    right_speed = max_speed
    # print("Move straight")
elif left_sensor detects line:
    left_speed = max_speed * 0.5
    right_speed = max_speed
    # print("Veer slightly right to center")
elif right_sensor detects line:
    left_speed = max_speed
    right_speed = max_speed * 0.5
    # print("Veer slightly left to center")
else:
    left_speed = -max_speed
    right_speed = -max_speed
    # print("Line lost – reverse to search")
```

- Uses 3 ground IR sensors to detect the black line.
- Implements a simple proportional controller to keep the robot aligned.
- Adjusts wheel speeds based on the difference between left and right sensor readings.

2. Obstacle Avoidance Module (OAM):

- Uses 8 proximity sensors to detect nearby obstacles.
- Determines which side the obstacle is on.
- Adjusts speed to steer away using a Braitenberg-like weighted approach.
- Takes over the main control loop when an obstacle is detected.

3. Line Leaving Module (LLM):

- Detects when the robot starts to leave the line.
- Temporarily inhibits other behaviours to allow a smooth transition to obstacle avoidance.

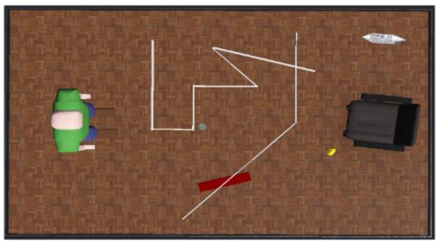
4. Obstacle Following Module (OFM):

- After avoiding the obstacle, this module encourages the robot to follow alongside the obstacle.
- Helps in keeping the robot oriented correctly during detour.

5. Line Entering Module (LEM):

- Once the robot passes the obstacle, this module helps it detect and re-enter the black line.
- Uses a state machine with logic to search, detect, and align with the line.

4.2. Wall Follower



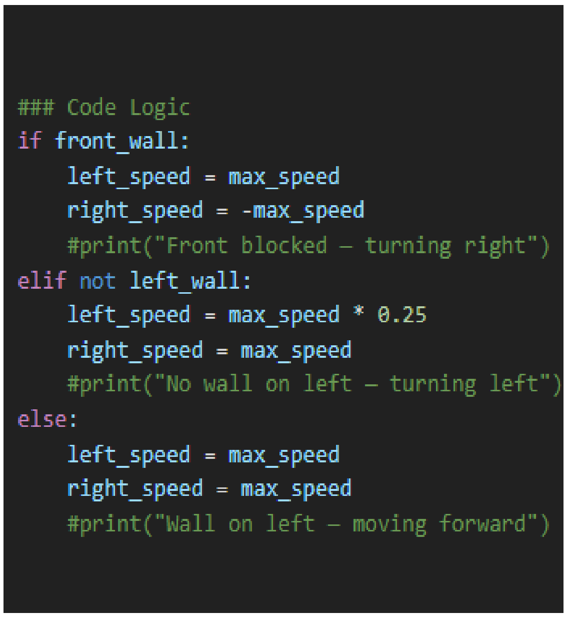
Objective:

To design and implement a mobile robot in the Webots simulation environment that can autonomously follow a wall using proximity sensors and simple reactive control logic.

Working Principle:

The robot constantly reads its proximity sensors and makes navigation decisions based on detected obstacles. The left-hand wall-following algorithm operates with the following logic:

1. **Front wall detected:** Turn right in place.
2. **No left wall detected:** Turn left to search for a wall.
3. **Wall on the left:** Move forward to follow the wall.



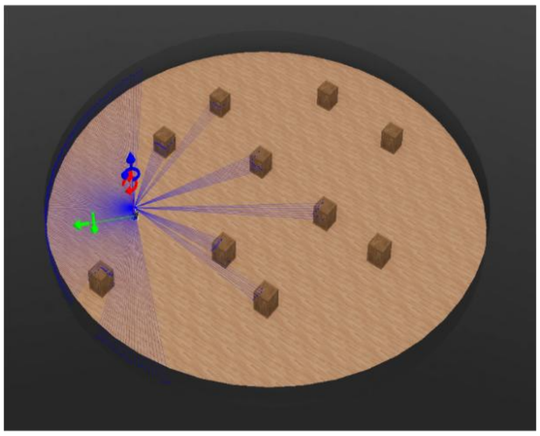
```
### Code Logic
if front_wall:
    left_speed = max_speed
    right_speed = -max_speed
    #print("Front blocked – turning right")
elif not left_wall:
    left_speed = max_speed * 0.25
    right_speed = max_speed
    #print("No wall on left – turning left")
else:
    left_speed = max_speed
    right_speed = max_speed
    #print("Wall on left – moving forward")
```

Conclusion:

The project demonstrates a simple yet effective wall-following robot using proximity sensors and reactive behaviour.

4.3. Avoid Obstacles

Objective:

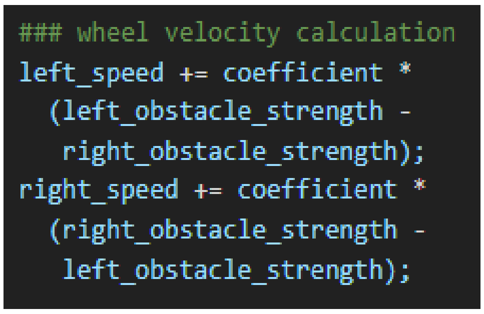


This project implements a reactive obstacle avoidance strategy using LIDAR-based sensing and a behaviour inspired by Braitenberg vehicles. The robot detects obstacles in its surroundings and dynamically adjusts wheel speeds to avoid collisions, even in dense and complex environments.

Working Principle: Braitenberg Vehicle Behavior

Sensor Input:

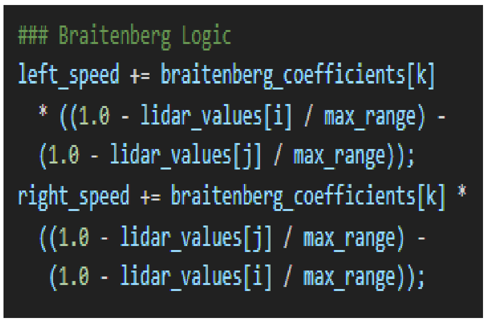
- The LIDAR sensor returns a 2D range image representing distances to surrounding obstacles.
- A specific angular window (from 0.25 to 0.5 of the scan width) is processed for obstacle detection.



```
### wheel velocity calculation
left_speed += coefficient *
  (left_obstacle_strength -
    right_obstacle_strength);
right_speed += coefficient *
  (right_obstacle_strength -
    left_obstacle_strength);
```

Control Strategy:

- For each LIDAR reading pair (left and right side of the scan), the robot computes a differential correction to the base wheel speeds.
- The formula considers how close the object is, and biases motor speeds to turn away from closer objects.
- The closer an obstacle is, the stronger the repulsion effect applied to the respective wheel.



```
### Braitenberg Logic
left_speed += braitenberg_coefficients[k]
  * ((1.0 - lidar_values[i] / max_range) -
  (1.0 - lidar_values[j] / max_range));
right_speed += braitenberg_coefficients[k] *
  ((1.0 - lidar_values[j] / max_range) -
   (1.0 - lidar_values[i] / max_range));
```

Gaussian Weighting:

- A Gaussian function gives higher weights to sensors in the front and lower to the sides, focusing the robot’s attention on what's ahead.

Note: Adjusting the left and right wheel of robot speeds based on the difference in obstacle proximity on each side. The robot turns away from closer obstacles by increasing the speed of the wheel farther from the obstacle, using Braitenberg coefficients to scale the response based on sensor angle and proximity intensity.

Formula Used (for Braitenberg Behavior)

Let,

Di = Distance measured by LIDAR on the left side (e.g., angle i)

Dj = Distance measured by LIDAR on the right side (mirror of i)

R = Maximum LIDAR range

Gk = Braitenberg coefficient at position k (a function of angle, typically Gaussian)

Then the formula becomes:

left_speed += Gk * ((1 - Di/R) - (1 - Dj/R)) = Gk * (Dj - Di)/R

right_speed += Gk * ((1 - Dj/R) - (1 - Di/R)) = Gk * (Di - Dj)/R

Interpretation: -

If Di < Dj: there's an obstacle closer on the left, so the robot turns right.

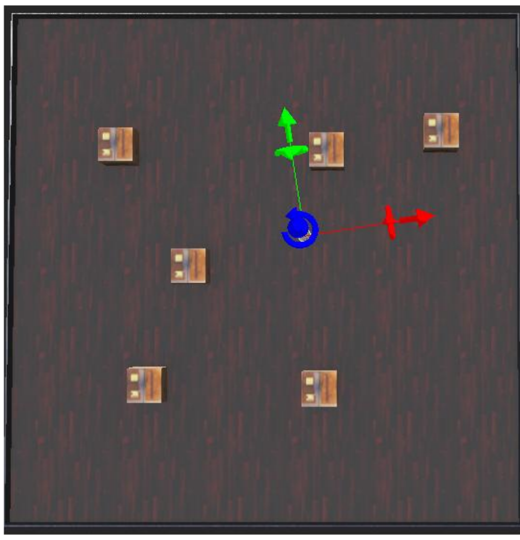
If Dj < Di: there's an obstacle closer on the right, so the robot turns left.

If Di is approximately equal to Dj: obstacles are equally distant; the robot moves forward.

The Gaussian-weighted coefficient Gk ensures that central obstacles have more influence than peripheral ones. This behaviour mimics a Braitenberg vehicle that reacts to sensor stimuli to avoid obstacles in a smooth, biologically-inspired way.

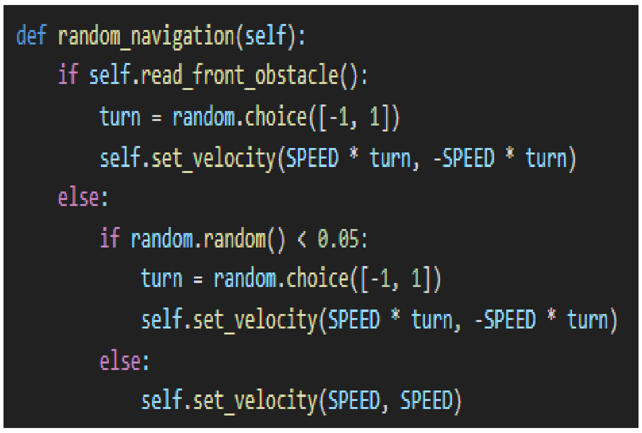
4.4. Robot Vacuum Navigation:

Objective:



Optimizing robot vacuum navigation involves improving how efficiently and effectively the robot covers an area while avoiding obstacles and returning to the dock. Several algorithms for the given navigation method are as follow:

1. Random Algorithm
2. Zigzag Algorithm
3. Spiral Algorithm
4. Snake Algorithm



```
def random_navigation(self):
    if self.read_front_obstacle():
        turn = random.choice([-1, 1])
        self.set_velocity(SPEED * turn, -SPEED * turn)
    else:
        if random.random() < 0.05:
            turn = random.choice([-1, 1])
            self.set_velocity(SPEED * turn, -SPEED * turn)
        else:
            self.set_velocity(SPEED, SPEED)
```

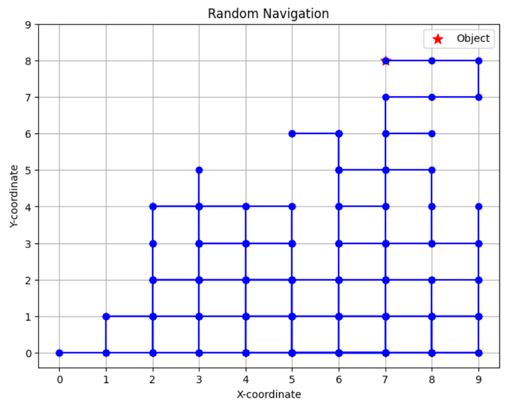
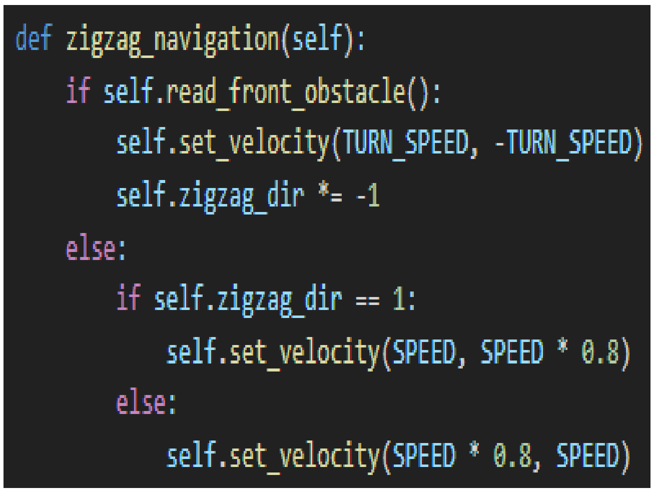


Fig 4.4.1 code logic and theoritical path for random algorithm



```python
def zigzag_navigation(self):
    if self.read_front_obstacle():
        self.set_velocity(TURN_SPEED, -TURN_SPEED)
        self.zigzag_dir *= -1
    else:
        if self.zigzag_dir == 1:
            self.set_velocity(SPEED, SPEED * 0.8)
        else:
            self.set_velocity(SPEED * 0.8, SPEED)
```

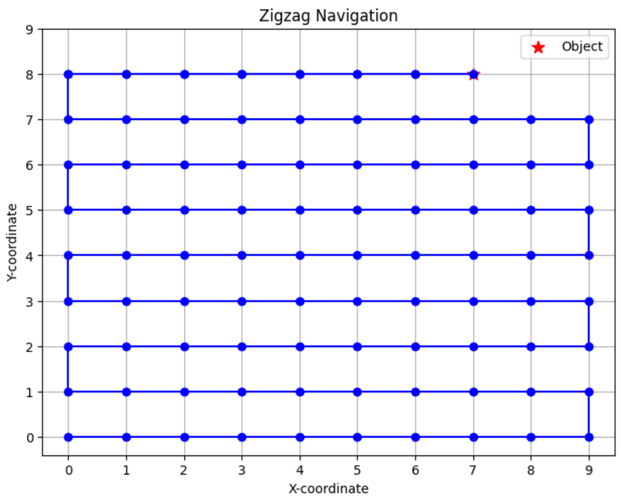
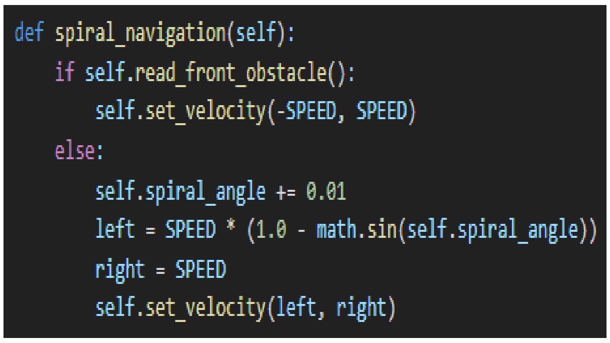


Fig 4.4.2 code logic and theoritical path for zigzag algorithm



```python
def spiral_navigation(self):
    if self.read_front_obstacle():
        self.set_velocity(-SPEED, SPEED)
    else:
        self.spiral_angle += 0.01
        left = SPEED * (1.0 - math.sin(self.spiral_angle))
        right = SPEED
        self.set_velocity(left, right)
```

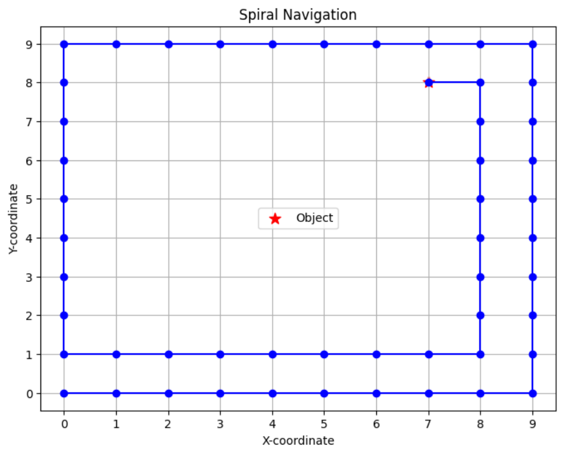
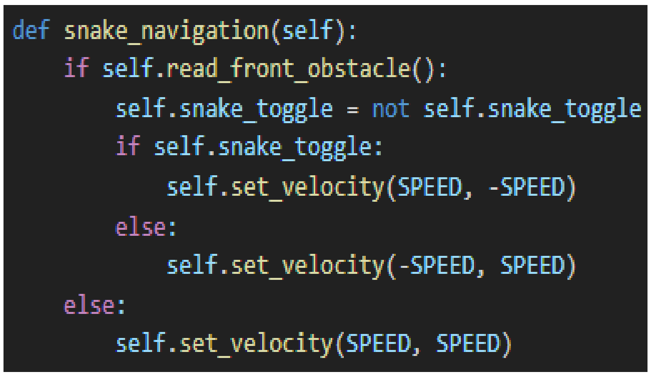


Fig 4.4.3 code logic and theoritical path for spiral algorithm



```python
def snake_navigation(self):
    if self.read_front_obstacle():
        self.snake_toggle = not self.snake_toggle
        if self.snake_toggle:
            self.set_velocity(SPEED, -SPEED)
        else:
            self.set_velocity(-SPEED, SPEED)
    else:
        self.set_velocity(SPEED, SPEED)
```

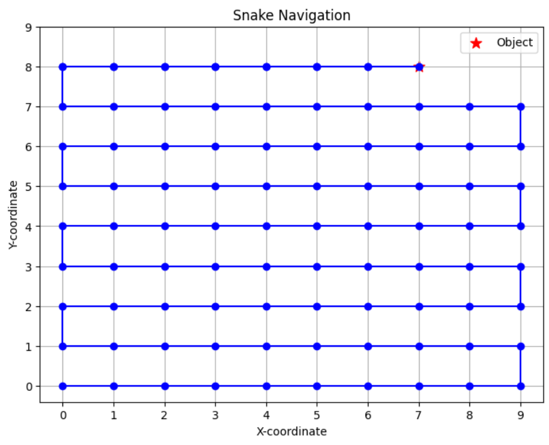


Fig 4.4.4 code logic and theoritical path for snake algorithm

Description for each navigation algorithm:

Random Navigation Algorithm

In this method, the robot moves around freely and changes direction when it bumps into something or gets close to an obstacle using its sensors. If it hits something on the left side, for example, it will back up and turn randomly to the right, choosing an angle between 0° and 180°. This randomness helps the robot explore its surroundings without getting stuck, even though it doesn't follow a set path.

Zigzag Navigation Algorithm

The zigzag algorithm helps the robot cover an area by moving back and forth in straight lines at a slight angle, making a zigzag pattern. When the robot hits an obstacle, it turns sharply—almost 180°—but just slightly off, so it doesn't go back exactly the same way. This small change in angle helps the robot gradually move forward and cover the entire area. There might be a bit of overlap, but it makes sure nothing is missed.

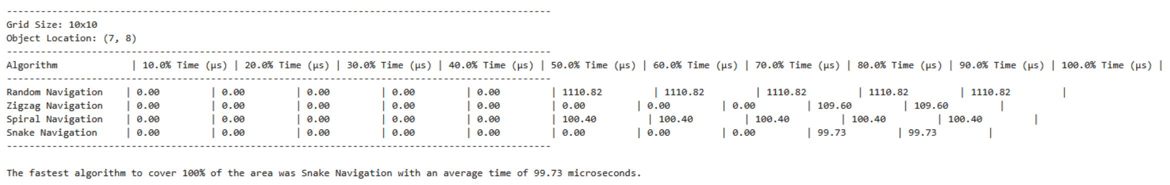
Rectangular Spiral Navigation Algorithm (Spiral Algorithm)

This method starts with the robot in the center of the area. It moves straight in one direction until it hits a wall or reaches a certain distance, then turns 90° to continue on the next side. After every two turns, it increases the length of the path, forming a bigger and bigger rectangle around the starting point. The robot keeps doing this, creating a spiral-like pattern with straight lines. When it hits a boundary, it stops expanding in that direction but continues spiraling in the other directions until the entire area is covered.

Snake Navigation Algorithm

The snake algorithm is a more organized version of the zigzag pattern. Instead of using angled lines, the robot moves in straight, parallel paths—like how a typewriter or printer head moves. After finishing one line, it shifts slightly by the width of its cleaning area and then goes in the opposite direction. This pattern ensures the robot doesn't miss any spots and avoids unnecessary overlap. It looks like the smooth motion of a snake, which is where it gets its name.

Comparing the result of the four algorithms:



```
--------------------------------------------------------------------------------------------
Grid Size: 10x10
Object Location: (7, 8)
--------------------------------------------------------------------------------------------
Algorithm            | 10.0% Time (µs) | 20.0% Time (µs) | 30.0% Time (µs) | 40.0% Time (µs) | 50.0% Time (µs) | 60.0% Time (µs) | 70.0% Time (µs) | 80.0% Time (µs) | 90.0% Time (µs) | 100.0% Time (µs) |
--------------------------------------------------------------------------------------------
Random Navigation   | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 1110.82         | 1110.82         | 1110.82         | 1110.82         | 1110.82         |
Zigzag Navigation   | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 109.60          | 109.60          |
Spiral Navigation   | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 100.40          | 100.40          | 100.40          | 100.40          | 100.40          |
Snake Navigation    | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 0.00          | 99.73          | 99.73          |
--------------------------------------------------------------------------------------------

The fastest algorithm to cover 100% of the area was Snake Navigation with an average time of 99.73 microseconds.
```

Algorithm	**Description**	**Pros**	**Cons**
Random Navigation	Robot moves in random directions until the entire area is covered.	Simple to implement.	* Highly inefficient; takes the longest time to cover the area. * Unpredictable path, not suitable for applications requiring systematic coverage.
Zigzag Navigation	Robot moves in a back-and-forth pattern, covering each row or column sequentially.	* Guarantees full coverage. * Relatively simple to implement. * More efficient than random.	Can be inefficient in certain grid shapes or if the object is found early.
Spiral Navigation	Robot moves in a spiral pattern, starting from the outer edge and moving towards the center.	* Guarantees full coverage. * Relatively efficient and covers area in a systematic way. * Good for finding objects located near the center.	* Slightly more complex to implement than Zigzag. * May not be the most efficient if the object is near the starting point.
Snake Navigation	Similar to Zigzag, but the robot moves along the rows/columns, changing direction at the end of each row.	* Guarantees full coverage * Simple implementation * More efficient than Random and Zigzag in most cases.	Path can be longer than spiral if the object is located in the center of the grid.

Table 4.4 Comparing the result of the four algorithms

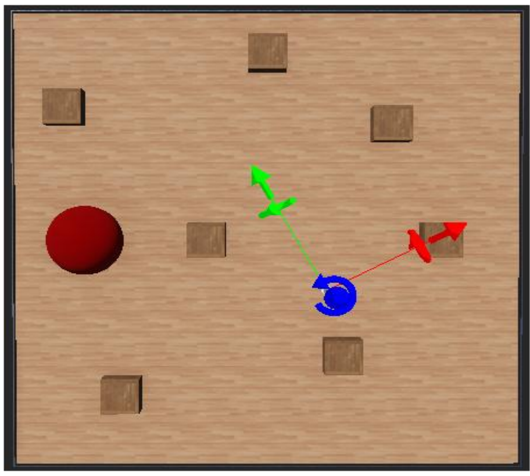
Conclusion

This study compared four robot vacuum navigation methods—random, zigzag, rectangular spiral, and snake—inside a 5m x 5m empty room using the Webots simulator. Among them, the rectangular spiral and snake algorithms performed the best. They covered the area more efficiently, with less overlapping, shorter travel distances, and faster cleaning times compared to the random and zigzag methods.

However, it's important to remember that the test was done in a simple, obstacle-free environment. If obstacles were added, the performance of spiral and snake patterns might not be as good. In such cases, more advanced navigation techniques like SLAM (Simultaneous Localization and Mapping) might be needed to handle the complexity.

Overall, this study shows that Webots is a great platform for testing and comparing robot navigation methods, and it provides a solid starting point for future research focused on improving robot vacuum navigation in more realistic and complex environments.

4.5. Target-Finding and Obstacle Avoidance



Objective

To develop and evaluate a lightweight and efficient navigation system that allows the e-puck robot to:

- Locate a specified target (e.g., a coloured object or a beacon).
- Avoid static obstacles using its onboard infrared proximity sensors.
- Reach the goal using a deterministic and memory-efficient strategy.

Algorithm Overview – Bug2

Principle

Bug2 is a simple path-planning algorithm designed for robots with minimal computational resources. It works as follows:

- The robot moves in a straight line (called the M-line) from the start point toward the goal.
- When it detects an obstacle blocking the path, it switches to wall-following mode.
- Once the robot re-encounters the M-line on the other side of the obstacle (closer to the goal than before), it resumes straight-line motion.

Why Bug2 for e-puck

- Requires minimal memory and computation.
- Works effectively with the e-puck’s proximity sensors.
- Adapts well to dynamic or partially known environments.
- Provides a balance between simplicity and goal-oriented behaviour.

Behaviour Design

- **Move toward target**: Robot aligns itself and moves forward toward target.
- **Detect obstacle**: If an IR sensor crosses a proximity threshold, obstacle is considered detected.
- **Wall-following**: Robot turns and follows the boundary of the obstacle using left- or right-hand rule.
- **Check M-line**: While wall-following, the robot checks if it has returned to the direct line to the target. If so, it resumes moving straight toward the goal.

Observations

- **High success rate** in reaching the target in open and moderately cluttered spaces.
- **Average time and distance** were significantly lower than random or coverage-based algorithms (zigzag, spiral).
- **Robustness**: Able to recover from complex obstacle shapes without getting stuck.

5.Result and Discussion

The results of this minor project demonstrate the successful implementation and evaluation of several fundamental navigation methods for the e-puck robot within the Webots simulation environment. Specifically, the **Bug2 algorithm** proved effective for target-finding and obstacle avoidance in moderately cluttered spaces, achieving a high success rate and generally shorter paths compared to simpler methods, though it showed limitations in complex obstacle scenarios. The **wall-following** behaviour, based on reactive proximity sensing, allowed the robot to navigate alongside walls. The **line follower**, integrating ground and proximity sensors, could track a line while also avoiding obstacles and returning to the path. The **LIDAR-based obstacle avoidance**, inspired by Braitenberg vehicles, enabled robust navigation in densely cluttered environments. Finally, in the context of robot vacuum navigation, the **spiral and snake algorithms** demonstrated superior area coverage efficiency compared to random and zigzag patterns in an obstacle-free setting. Overall, the project successfully showcased the e-puck's capabilities and the effectiveness of different algorithmic approaches for achieving basic autonomous navigation in various simulated scenarios.

The discussion surrounding this minor project on developing navigation methods for mobile robots in cluttered environments using the Webots simulator and the e-puck robot can be multifaceted, touching upon the effectiveness of the chosen algorithms, the capabilities of the simulation platform and the robot, and the broader implications for real-world robotics.

Effectiveness of the Navigation Methods:

The successful implementation of the Bug2 algorithm for target-finding and obstacle avoidance highlights the viability of simple, memory-efficient strategies for goal-oriented navigation in moderately complex environments. The comparison with basic exploration methods underscores the importance of incorporating goal-directed behavior for efficiency. However, the identified limitations of Bug2 in handling certain obstacle configurations raise important questions about the robustness of purely reactive approaches and the potential need for more sophisticated spatial reasoning or memory in challenging scenarios.

The wall-following behavior, while basic, demonstrates a fundamental reactive strategy for navigating along boundaries. Its simplicity makes it computationally inexpensive and easy to implement, but its effectiveness is limited in open spaces or environments lacking continuous walls.

The line follower, integrating multiple sensors and modular behaviors, showcases a more complex control architecture capable of handling distinct navigation tasks. The successful switching between line following and obstacle avoidance demonstrates the potential for creating layered control systems that can adapt to different environmental conditions. However, the complexity of managing these transitions and ensuring smooth recovery onto the line presents design challenges.

The Braitenberg-inspired obstacle avoidance using simulated LIDAR highlights the power of sensor-driven reactive behaviours in navigating highly cluttered spaces. The smooth, biologically-inspired movements are advantageous for avoiding collisions without getting stuck in oscillations. However, the reliance on a relatively sophisticated sensor like LIDAR might not be feasible for all low-cost robots like the basic e-puck, suggesting a trade-off between sensor richness and cost-effectiveness.

The comparison of robot vacuum navigation algorithms provides valuable insights into the efficiency of different coverage strategies. The superior performance of spiral and snake patterns in a clean environment underscores the importance of systematic movement for thorough area coverage. However, the acknowledged limitation of these algorithms in the presence of obstacles emphasizes the need for integrating robust obstacle avoidance and potentially SLAM techniques for real-world robot vacuum applications.

Capabilities of Webots and the e-puck:

The project effectively demonstrates Webots as a powerful and versatile simulation platform for robotics research and education. Its realistic physics engine, diverse sensor and actuator models, and support for multiple programming languages facilitate the development and testing of complex robot behaviours in a controlled virtual environment. The ease of creating different arena configurations and introducing obstacles allows for systematic evaluation of navigation algorithms under varying conditions.

The e-puck robot, with its compact size and rich sensor suite (including infrared proximity and ground sensors, and the potential for a camera), proves to be an excellent platform for exploring fundamental navigation concepts. Its limitations in terms of processing power and memory, as highlighted by the rationale for choosing Bug2, encourage the development of efficient and lightweight algorithms. The simulated encoders and motor control allow for realistic modelling of the robot's movement.

Broader Implications and Future Directions:

The findings of this project contribute to the broader understanding of mobile robot navigation in cluttered environments. The exploration of different algorithmic approaches and sensor modalities provides valuable insights for developing autonomous robots for various applications. The identified limitations of simpler algorithms in complex scenarios underscore the need for more advanced techniques, such as incorporating environmental mapping (SLAM), path planning, and more sophisticated control strategies.

6.Conclusion

This minor project successfully explored and evaluated several fundamental navigation methods for the e-puck robot within the Webots simulation environment. The implementation of the memory-efficient Bug2 algorithm demonstrated a viable approach for target-finding and obstacle avoidance in moderately cluttered spaces, albeit with identified limitations in highly complex scenarios. Basic wall-following and line-following behaviours, along with a Braitenberg-inspired obstacle avoidance strategy utilizing simulated LIDAR, showcased different reactive control mechanisms for navigating specific environmental constraints. Furthermore, the comparison of robot vacuum coverage algorithms highlighted the efficiency of systematic patterns in unobstructed areas. While each method presented its own strengths and weaknesses, the project collectively underscores the e-puck's potential as a platform for studying autonomous navigation and the effectiveness of Webots as a tool for developing and testing robotic algorithms. The findings provide a solid foundation for future work focused on enhancing robustness, integrating more sophisticated sensing, and ultimately translating these simulated capabilities to real-world robotic applications in cluttered environments.

7.Application and Future Scope

The exploration of navigation methods for mobile robots in cluttered environments, as undertaken in this minor project, holds significant promise for various real-world applications and suggests several compelling avenues for future research and development.

Applications:

The fundamental navigation capabilities investigated have direct relevance to a multitude of robotic applications:

- **Small-Scale Robotics and Education:** The simplicity and resource-efficiency of algorithms like Bug2 and basic reactive behaviors make them ideal for educational robots like the e-puck, enabling students to grasp core concepts of autonomous movement and sensor integration.
- **Service Robotics in Cluttered Indoor Environments:** Robots designed for domestic tasks (e.g., cleaning, delivery) or assisting the elderly and individuals with disabilities must navigate cluttered homes and offices. The principles of obstacle avoidance and target finding explored here are foundational for such applications.
- **Inspection and Surveillance:** Small, agile robots capable of navigating complex or confined spaces are valuable for inspection tasks in industrial settings or for

surveillance in areas inaccessible or dangerous for humans. The wall-following and obstacle avoidance strategies are particularly relevant.

- **Search and Rescue Operations:** In disaster scenarios, robots need to navigate through debris and cluttered environments to locate survivors. Robust obstacle avoidance and efficient exploration strategies, potentially building upon the Bug2 algorithm with enhanced recovery mechanisms, are crucial.
- **Logistics and Warehouse Automation:** While often relying on more sophisticated systems, basic navigation primitives like line following and obstacle avoidance can be integral to the movement and coordination of autonomous mobile robots (AMRs) in warehouses and logistics centers.

Future Scope:

The findings and limitations of this project point towards several exciting directions for future research and development:

- **Enhanced Robustness and Error Handling:** Developing more sophisticated mechanisms for detecting and recovering from navigation failures, such as "stuck conditions" in Bug2 or loss of line in the line follower, is crucial for real-world reliability. This could involve incorporating learning-based approaches or more complex state machines.
- **Integration of Advanced Sensing Modalities:** Exploring the fusion of data from multiple sensors (e.g., combining proximity sensors with cameras, inertial measurement units (IMUs), or even rudimentary 3D sensors) can lead to more robust and context-aware navigation. For instance, visual information could enhance target recognition and obstacle understanding.
- **Development of Mapping and Localization Capabilities (SLAM):** For truly autonomous navigation in unknown or dynamic environments, integrating Simultaneous Localization and Mapping (SLAM) algorithms with the explored control strategies would be a significant advancement. This would allow the robot to build a map of its surroundings and localize itself within it.
- **Learning-Based Navigation:** Investigating the application of machine learning techniques, such as reinforcement learning or imitation learning, to train robots to navigate cluttered environments more effectively and adapt to novel situations. This could potentially lead to more efficient and human-like navigation behaviours.
- **Multi-Robot Systems and Coordination:** Extending these navigation methods to multi-robot systems opens up possibilities for cooperative tasks like distributed exploration, simultaneous mapping, or coordinated object manipulation in cluttered environments.

REFERENCES

1. R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. MIT Press, 2011.
2. F.Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. C. Zufferey, D. Floreano, and A. Martinoli, “The e-puck, a Robot Designed for Education in Engineering,” *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 2009.
3. O.Michel, “Webots™: Professional Mobile Robot Simulation,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.
4. E. Rimon and D. E. Koditschek, “Exact robot navigation using artificial potential functions,” *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.
5. A. Lumelsky and T. Skewis, “Incorporating range sensing in the robot navigation function,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 5, pp. 1058–1068, 1990. [Bug Algorithms]
6. V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, MIT Press, 1986. [Braitenberg Vehicle Behavior]
7. H. Choset, “Coverage for robotics – A survey of recent results,” *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1–4, pp. 113–126, 2001.
8. C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.
9. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, MIT Press, 2005.
   — A foundational book covering localization, mapping, SLAM, and navigation under uncertainty.
10. H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part I,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, June 2006.
    — A classic tutorial paper on SLAM, relevant for your "Future Scope" section.
11. B. Yamauchi, “Frontier-Based Exploration Using Multiple Robots,” *Proceedings of the Second International Conference on Autonomous Agents*, 1998.
    — Useful for future extensions of your work to multi-robot exploration.
12. D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
    — Relevant to your obstacle avoidance methods.
13. J. Borenstein and Y. Koren, “The vector field histogram—fast obstacle avoidance for mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, June 1991.
    — Important for understanding advanced sensor-based avoidance algorithms.

14. L. E. Parker, “Current state of the art in distributed autonomous mobile robotics,” *Distributed Autonomous Robotic Systems 4*, Springer, Tokyo, 2000, pp. 3–12.
— Discusses coordination in multi-robot systems; useful for the multi-robot extension.
15. M. Quigley et al., “ROS: an open-source Robot Operating System,” *ICRA Workshop on Open-Source Software*, vol. 3, no. 3.2, 2009.
— Can be mentioned in the future work section, as ROS is often used with Webots.
16. R. Arkin, *Behavior-Based Robotics*, MIT Press, 1998.
— Great reference for the behavioral modules like wall following and Braitenberg behavior.
17. D. C. Hsu, J.-C. Latombe, and R. Motwani, “Path planning in expansive configuration spaces,” *International Journal of Computational Geometry & Applications*, vol. 9, no. 04n05, pp. 495–512, 1999.
— For discussing more advanced path planning, beyond Bug2.
18. M. Himmelsbach, T. Luettel, and H.-J. Wuensche, “Real-time object classification in 3D point clouds using point feature histograms,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.
— If you plan to mention future use of LIDAR in more detail.