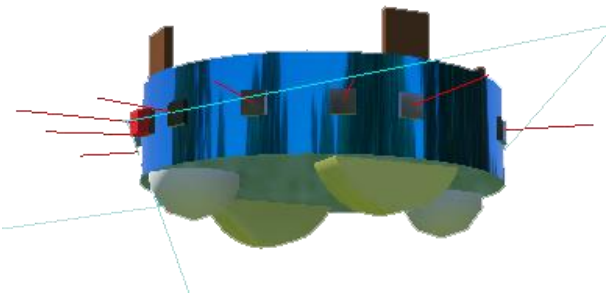
Project: Autonomous Robotic Solution for Warehouse Operations



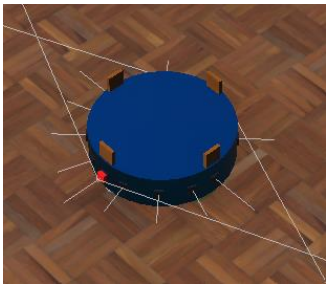
Lavigne Tom - 233777770

I. INTRODUCTION

This report presents a comprehensive solution for an autonomous robotic system designed to operate within a warehouse environment. The primary objective of this system is to facilitate the efficient movement of goods within the facility, utilizing an autonomous vehicle. The warehouse environment is characterized by aisles and loading areas that remain constant over time. Existing infrastructure is in place to load and unload the autonomous robot as needed, and to provide the robot with its goal location.

The challenge is to design a robot and associated navigation tools that will enable it to localize within its environment, map its environment, and navigate to its goal location, all while minimizing collisions. This report details the development process and outcomes for the following key tasks: Robot Design, Pose Determination, Environment Mapping, Autonomous Navigation.

Fig. 1. General view of the robot



II. ROBOT DESIGN

The design of the robot focuses on creating a morphology that is efficient in moving boxes around a warehouse while minimizing collisions. The robot platform, motorization and wheels, and sensors are the key components of the design.

A. *The platform*

The platform of the robot is a disk of 0.4 radius. The choice of a disk (cylinder) shape is due to its ability to avoid collisions more effectively than a rectangular shape. The large surface area of the disk makes it suitable for carrying standard wooden boxes of Webots. Additionally, small stops are placed on the platform to prevent objects from falling off.

B. *The Motorization and Wheels*

The robot is equipped with differential motorization with two moving wheels (hinge joints) on each side, providing the necessary propulsion for forward movement and turning. To enhance stability, two free spheres (ball joints) are placed on the front and rear of the robot. This configuration allows the robot to turn on the spot and move forward without friction. The wheels can look like that : https://ie.rs-online.com/web/p/trolley-wheels/2430377?gb=s

C. *Sensors*

The robot is fitted with a variety of sensors to facilitate localization, mapping, and collision avoidance.

- Two GPS sensors are placed on each side of the robot to compute its exact pose (x, y, θ).
- Two horizontal LIDAR sensors are placed at the front and back of the robot to map the environment. These sensors have a horizontal range of 5π/4 rad, allowing for a 360-degree mapping of the surroundings within a 5-meter radius of the robot. They have 103 ray, it's enough to have a proper map and low latency but if you have a better computer you can make it bigger and you will have less approximations. This one should work for example : https://www.robotshop.com/products/yujin-yrl-series-2d-lidar-w-5m-range
- Eleven distance sensors are placed all around the robot to avoid collisions. The placement of these sensors is at angles 0, π/2, -π/2, 3π/4, -3π/4, π/4, -π/4, 7π/18, -7π/18, π/10, and -π/10. While this may seem excessive, it is a cost-effective solution that ensures comprehensive collision

detection. The range of the distance sensors varies depending on their placement, but most have a range of 0.2 meters. The sensor at 0 rad has a slightly longer range to detect if the robot is heading straight towards an obstacle. The sensors at -π/2 and π/2 have a range of 0.3 meters as they are also used for driving along an obstacle and checking for its presence on the right and left. Those one should work : https://whadda.com/product/ultrasonic-distance-sensor-wpse306n/

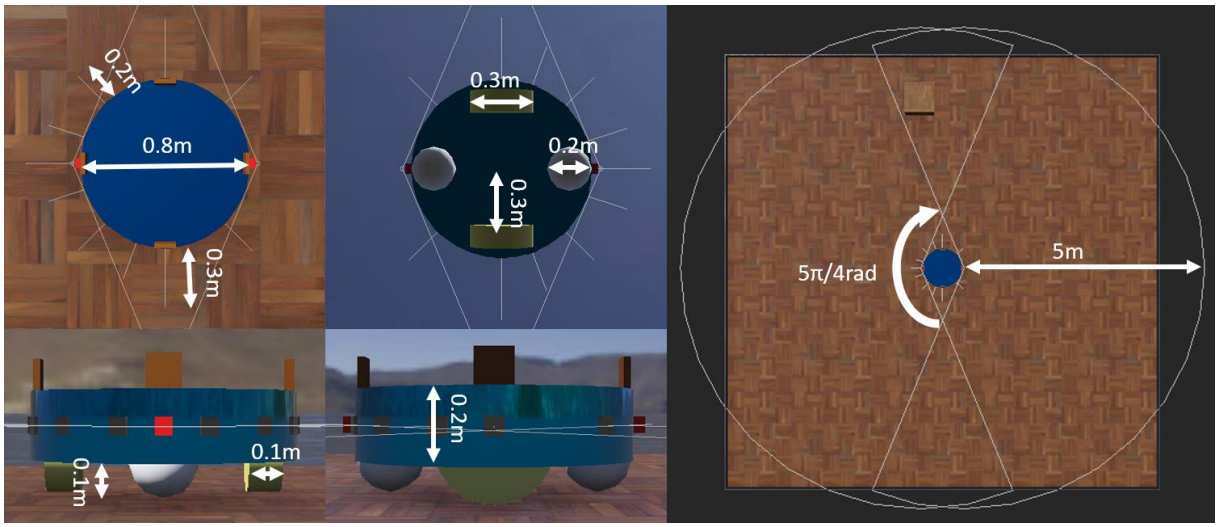


Fig. 2. Dimensions and elements of the robot

III. POSE DETERMINATION

To determine the pose of the robot within the warehouse environment, a controller has been developed that utilizes the Webots GPS sensors. This section provides a detailed explanation of the controller's functionality and the underlying code. The controller consists of two primary functions: getGpsData() and getRobotPose().

- getGpsData(gps, value): This function retrieves the GPS data from the sensors. It iterates through the list of GPS sensors and updates the corresponding values in the value list using the getValues() method provided by the Webots API. The function then returns the updated value list.
- getRobotPose(gps_value): This function calculates the robot's pose (x, y, θ) based on the GPS data. It first initializes a pose list with default values. The orientation (θ) is calculated using the np.arctan2() function, which computes the angle between the positive x-axis and the line connecting the two GPS points.

$$\theta = \arctan2(y2 - y1, x2 - x1) \text{ and normalized}\_\theta = ((\theta + 180) \% 360) - 180$$

The result is then converted from radians to degrees. The x and y coordinates are calculated using trigonometric functions based on the position of the GPS (+0.2 and -0.2 on y in robot coordinates) and the GPS values. The function returns the calculated pose.

$$x = gps\_value[1][0] - 0.2 * \sin(angle)$$

$$y = gps\_value[1][1] + 0.2 * \cos(angle)$$

IV. Environment Mapping

The mapping controller is designed to transform an 80 by 80 grid of -1 into a grid of 0 and 1, where 0 represents an empty cell and 1 represents an occupied cell. This transformation is achieved through the use of front and rear LIDARs mounted on the robot.

A. Update the map

The mapping process is initiated with the robot updating the map with information from the LIDARs. The LIDARs retrieve the distances of 103 rays each and we have a table with the 103 corresponding angles, resulting in a total of 206 rays. The "infinite" values, which represent no obstacle detection, are filtered out. For each distance, a formula is applied to obtain the x and y positions of the obstacles. This formula takes into account the position and orientation of the robot, the position of the LIDAR on the robot (±0.45m on the y axis of the robot), the angle of the measurement and the measurement itself.

x= x_robot+0.45×cos(angle_robot)+distance_front×cos(angle_robot +filtered_lidar_angles_front[x])

y==y_robot+0.45×sin(angle_robot)+distance_front×sin(angle_robot +filtered_lidar_angles_front[x])

The x and y positions of the obstacles detected by the LIDARs are then transformed into grid positions. The corresponding cells in the grid are set to 1, indicating the presence of an obstacle. A Bresenham algorithm is applied to set all cells between the robot and the detected obstacle to 0, with a sensitivity threshold of 0.3. Bresenham's line algorithm is a line drawing algorithm that determines the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, subtraction, and bit shifting, all of which are very cheap operations in historically common computer architectures.

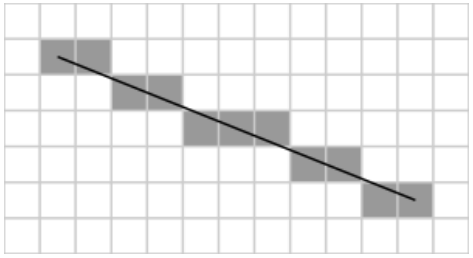


Fig. 3. Example of the Bresenham algorithm

In addition, all values within the robot's radius are set to 0, as they are considered empty because the robot is currently there.

B. Discover the map

While the global goal of mapping the entire environment is not achieved, the robot continually updates the map with new information from the LIDARs. To improve efficiency, this discovery mechanism is only performed every 20 steps, as LIDAR calculations can be time-consuming. The robot then identifies the point closest to it in Euclidean distance that has not yet been discovered and navigates towards it. To do that, It will find the angle of the target and turn until it reaches this angle (in an effective way by choosing to turn right or left depending on the angle), then advance to this point. Each step it will verify that he is still in the good orientation.

A collision avoidance mechanism is also in place thanks to distance sensors. If an obstacle is detected in front of the robot, it will turn right if there's an obstacle to the left, and vice versa. If the robot detects a presence between itself and the goal, it will turn to put it next to it, and then go around it. It will do that by verifying there is nothing in front thanks to distance sensors and still the obstacle on a side thanks to those at $\pi/2$ and $-\pi/2$,. The robot checks every 20 consecutive bypass steps whether it is still heading in the right direction. The collision avoidance mechanism is deactivated as soon as the target is approached (0.2m).

Once the map no longer contains a "-1", indicating that all areas have been explored, it exports the map as a .csv file. I've also added a small function that eliminates imperfections of 0.1 in size. If a previous file exists, it is replaced by the new one.

C. Limitations

A critical aspect of the mapping mechanism is its ability to ensure that the robot has traversed the entire map without missing a single point. This is achieved through a comprehensive exploration strategy that systematically updates the map with new information from the LIDARs.

The exploration strategy can be adjusted to balance the trade-off between mapping speed and quality. To accelerate mapping, the percentage of uncovered pixels can be lowered, and the precision of the Bresenham algorithm can be reduced. However, this comes at the cost of lower mapping quality. Conversely, mapping quality can be improved by adding more rays to the LIDAR and updating the map every step instead of every 20 steps.

As a limitation, we can add that the robot pass very close to the walls and a good thing to do with more time is to make the minimum distance from the walls a bit further to enhance the security.

One limitation of the current mapping algorithm is that it represents obstacles with a thickness of 0.1m greater than their actual size. This is likely due to the rounding process when applying the formula to determine the position of obstacles. Sometimes the algorithm rounds up to the nearest 0.1, and other times it rounds down to the nearest 0.1. While this issue has not been corrected, it is arguably better to overestimate the size of an obstacle to ensure safe navigation.

Another obvious limitation is that mapping currently only works for an 80 by 80 grid, but it seems feasible to adapt this so that the robot can explore an unfamiliar environment. In fact, it would be enough to look at the obstacles with the largest x and y values, and this would give us an idea of the size of the arena.

In summary, the mapping mechanism employs a systematic exploration strategy to guarantee complete coverage of the environment. The strategy can be adjusted to balance mapping speed and quality, and while there is a limitation in the representation of obstacle size, it is a conservative approach that prioritizes safety.

V. NAVIGATION

The navigation controller is responsible for enabling the robot to autonomously navigate to any goal location in the environment from any other location. The first approach, which is the simplest and safest, is the one that we have chosen to retain.

A. First version

This approach involves expanding the walls with a buffer equal to the size of the robot's radius, which is 0.4m. This ensures that the robot maintains a safe distance from obstacles. We then apply the A* algorithm, which provides us with a list of coordinates corresponding to the shortest path to the goal, if one exists. A* is a pathfinding algorithm used to find the shortest path from a starting point to a goal point on a graph or grid. It works by evaluating nodes based on the combined cost of reaching them from the start node (G-cost) and the estimated cost to reach the goal from there (H-cost).

The robot then navigates to each of these points in sequence to reach the goal. Similar to the mapping mechanism, the robot turns to achieve the correct angle and advances while verifying this angle until it reaches the desired point. To improve fluidity, we set a larger tolerance for intermediate points to avoid zigzags created by the A* algorithm. Since it cannot move diagonally, it creates diagonal staircases, and the tolerance eliminates this issue. The final goal precision is 0.02m. With this approach, the robot can reach any point on the map without encountering obstacles.

B. Second version

The second part of the navigation section focuses on an alternative, less secure approach that was developed to address a specific challenge in the warehouse environment.

In the northern part of the map, there is a narrow passage that is 1m wide. When a 0.4m buffer was applied to the walls, it blocked the passage, as the passage was only 1m wide diagonally. Since the robot is 0.8m wide, it is technically capable of passing through the passage. However, the magnification of the walls caused the A* algorithm to suggest a significant detour.

To address this challenge, a second version of the navigation controller was developed. This version employed the same algorithm as the previous version but included collision avoidance mapping. With this approach, it was possible to magnify walls whose length was less than the robot's radius. This resulted in a significant gain in efficiency, as it saved a lot of time due to the zigzags created by collision avoidance.

However, this approach had a drawback. The robot passed very close to the walls, which was not ideal for safety. In fact, the robot came within 0.1m of the walls, which increased the risk of collision. After evaluating the trade-off between efficiency and safety, it was decided not to use this solution. While it provided a significant gain in efficiency, the risk of collision was deemed too high. The first approach, which involved expanding the walls with a buffer equal to the size of the robot's radius, was determined to be the safest and most reliable approach.

VI. CONCLUSION

The report presents an autonomous robotic system for warehouse operations, featuring a robust robot design, efficient pose determination, accurate environment mapping, and reliable navigation. The system uses GPS and LIDAR sensors to determine the robot's position and map the warehouse, enabling effective navigation and obstacle avoidance. The system has been tested and validated, demonstrating its potential to advance warehouse automation technology. Future work could focus on performance improvement especially on mapping and the production of a real prototype.

REFERENCES

[1] A* algorithms, (n.d.). In Wikipedia. Retrieved February 28, 2023, from https://en.wikipedia.org/wiki/A*_search_algorithm

[2] QuickCS, *"Bresenham's Line Drawing Algorithm With Example"* on https://youtu.be/GKnzwTMmO1Y?si=wruz7dgl1m22RY9N

[3] Ginn, C. M. (2024, February). Lecture 2 to Lecture 5 from sensors to kinematics to navigation. Ireland.