Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Mechanical Engineering

Design and simulation of an autonomous warehouse robot

Ahmed Ali

20335023

Module Co-ordinator: Dr. Adam Kavanagh Coyne
Date: 15 August 2025

Introduction

The field of automation has undergone rapid advances in recent years due to the development of more efficient sensors and mapping techniques alongside complex artificial algorithms. This has given rise to applications in many areas, such as autonomous driving, where self-driving cars can navigate and make adjustments based on input data received from sensors to adjust speed, make lane changes, and stop when needed. Another application is in the robotics field, especially in warehouse operations, where companies such as Amazon utilize robots to organize and move objects by mapping out their environment, planning to respond to different scenarios and picking the most efficient path to achieve the task. Similar to how autonomous systems operate, such robot designs account for the environment for sensor placement, tasks for algorithm and logic design, and performance for supporting loads being transported. This report aims to explore the design and application process of an autonomous robot by exploring design logic and steps to create such one.

This report explores the following goals in design

- A: Create an autonomous robot that can carry a load and traverse around a given path
- B: Create a model suitable for navigating in and out of small areas effectively.
- C: Integrate a light system to indicate to onlookers the current activity taking place.
- D: Reliability of the design and current life applications
- E: Testing and results
- Possible Improvements and additions

Morphology and design choices

The first stage of building the robot involves selecting a suitable design. In general, warehouses require a dynamic and agile body to navigate and traverse environments effectively. The robot also needs to be able to support the load that it carries securely. With the factors above in mind, 3 concept designs were sketched:

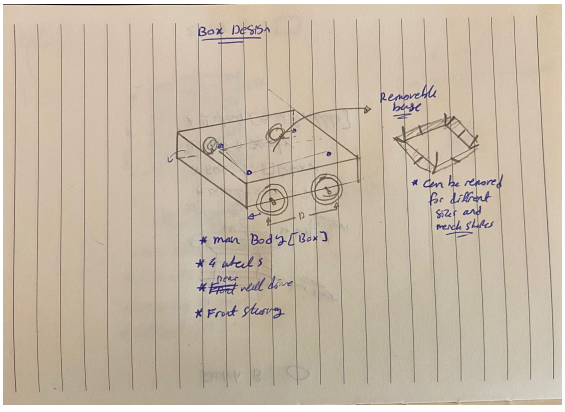

Figure 1: Robot Sketch with a Boxy Base

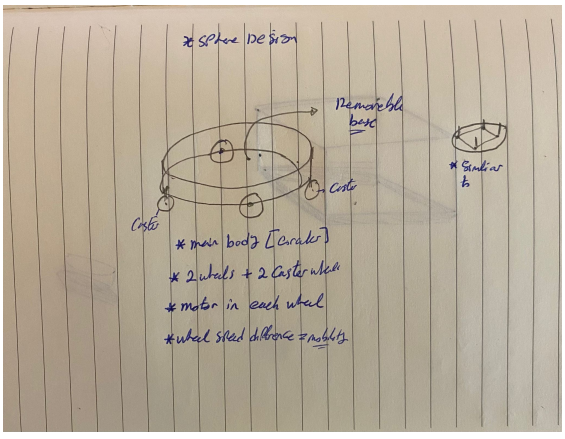

Figure 2: Robot Sketch with a Circular base

The main idea for Figure 1 is that the base acts as an anchor for a removable structure, which allows for the transport of objects with different sizes securely rather than being bound to one shape. In Figure 1, the rectangular body is supported by 4 wheels, each with its own rotational motor, allowing it to change direction by utilizing the wheel rotation. The low base also allows for

better mobility and helps with stability by remaining close to the centre of gravity. This design has a significant limitation where the steering angle is limited due to the connected hinge structure, thus making the robot's turning radius and manoeuvrability through a tight regions limited, to add it will require more setup on Webots simulator, which adds to the complexity of the model.

Figure 2 is an attempt to address the limitation in Figure 1 with 2 major design changes. The first is a change to the base by making it spherical, with the anchor area for cargo remaining dynamic. The drive and steering consist of 2 wheels, each powered by a separate rotational motor, and to ensure the stability of the base, rolling balls are placed in the front and back of the body. The turning motion is controlled by utilizing the rotational difference between the 2 motors to move the robot, giving it 360 degrees of freedom and allowing it to reverse course if needed. A possible weakness of the design is the ability to support the load that it will carry, especially if the load structure can be changed.

Final Design & Sensor Placement

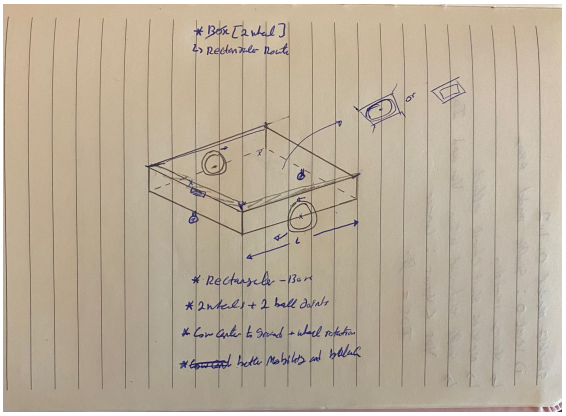

Figure 3: Open-loop control diagram

Based on the points from designs 1 and 2, a combination of strong elements from both designs resulted in the 3rd design shown in Figure 3. The base is box-shaped, serving as a strong support structure, with a simple drive and steering system supported by two joint balls at the front and rear. This design simplifies the structure of the robot on Webots,,, reducing the risk of causing the simulator tocrash;; the holding mechanism of the load was also simplified by using a box frame that sits on top of the robot base, the frame secures the cargo in place and keeps the design compact, the diminsions of the robot are (**0.6x0.6x0.4**)
The Final model created is shown in figure 4, aligning closely with the sketch in Figure 3, with the addition of 4 Led's at the corners of the base working as a state indicator.

Sensor Placement

Accounting for real world applications was crucial for the sensor placement, to keep the logic and overall design simple and realistic 3 main sensors are used to assist with movement and traversal, 2 GPS nodes are placed in front and back of the robot, a single GPS would only approximate the direction that the robot is facing (orientation) making it unreliable, a front and back setup allows for the calculation the the centre position of the robot consistently which can be used alongside mathematical functions to calculate the orientation, the LIDAR was initially placed on top of the robot but when tested against low clearance objects it failed to detect it, for that reason it was moved to the middle where it can detect low clearance object while still detecting long objects.

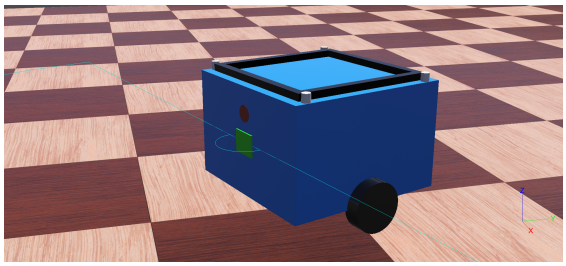

Figure 4: Robot side view

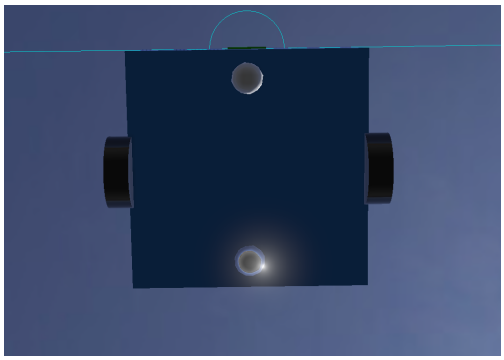

Figure 5: Robot bottom view

Actuators

As mentioned in the morphology section, the robot consists of 2 wheels, each with an independent motor to drive the robot forward and also to control the direction by using the rotational difference between the wheels to turn, reverse, or even spin in place. Due to the structural shape of the robot, having a 2-wheel system requires a balance method. The ball joints, located at the front and rear of the robot's base, provide the necessary balance and enable a higher degree of mobility, which is beneficial for navigating tight areas, such as the warehouse floor used in the project.

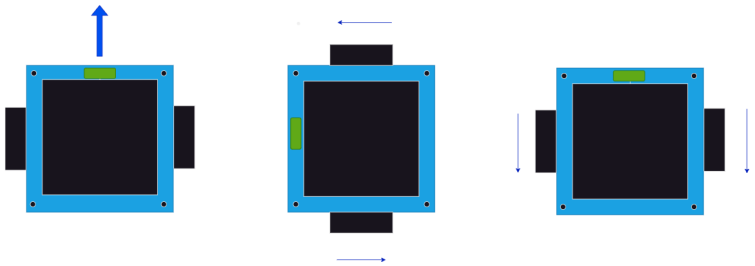

Figure 6: Robot Moving Forward, Turning Left and Reversing

Localization Plan

In order to traverse a grid path and avoid objects along the way, the robot needs to know its current location, the location it is heading to, and the direction it is facing. To keep the project simple, a combination of GPS and LIDAR systems is used to be able to find the robot coordinates and the direction that it's facing, mapping out a suitable path while also being able to avoid objects along the way.

GPS

As seen in Figure 4, the model utilizes 2 GPS nodes, a forward GPS, and a back GPS. A single GPS proved to be a challenge, as it only provided the coordinates on the front and did not account for the changes required should an object be detected. It also made the orientation prediction more difficult by limiting the options for the orientation calculation to use the arctan function,

which has a smaller range [pi / 2 , pi / 2], reducing accuracy.

To calculate the orientation more accurately, the 2 GPS nodes are used to find the exact centre of the robot for a consistent value and for orientation, for orientation, arctan2 was used, as the centre orientation allows for a more accurate reading through all 4 quadrants. The code for a single GPS node is expanded to include the middle coordinates and uses arctan2 to find the orientation, reducing the complexity of the overall code.

The (X, Y) coordinates for each GPS node and the orientation are computed as shown in Table 1.

Name	**Formula**
Front GPS position	$(X_f,\ Y_f)$
Back GPS position	$(X_b,\ Y_b)$
Midpoint (centre)	$X = \frac{X_f + X_b}{2}, \quad Y = \frac{Y_f + Y_b}{2}$
Difference vector	$\Delta X = X_f - X_b, \quad \Delta Y = Y_f - Y_b$
Heading	$\Theta = \text{atan2}(\Delta Y, \Delta X)$

Table 1: Formulas for computing robot position and heading from dual GPS sensors.

The GPS controller (`412_GPS.py`) uses these formulas to print the (X, Y) position and Θ, converting the angle from radians to degrees. The GPS controller (`412_GPS.py`) uses the above formulas to print the (X, Y) position and Θ (converted from radians to degrees), the midpoint is used as a constant refrence point for the position

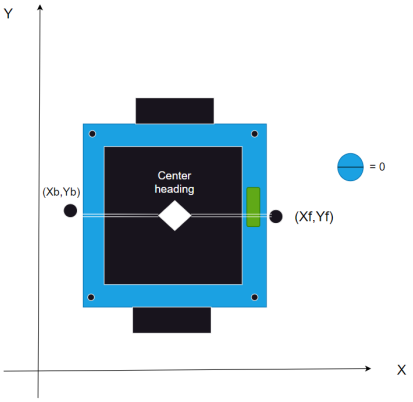

Figure 7: Default Orientation

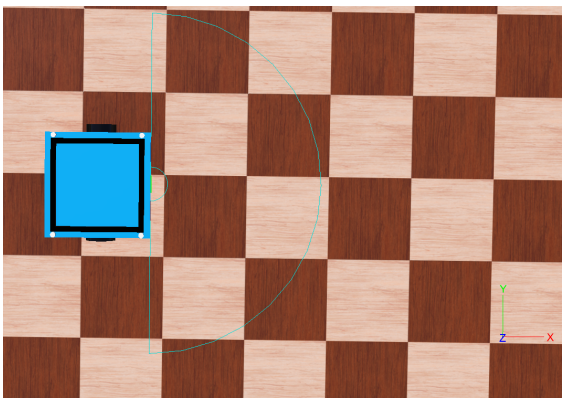

Figure 8: Robot view in Webots

A* Path Planning

2 python files were provided as part of this project (A*.py), and (grid.csv) maps out the occupancy grid, noting the start position and the goal position alongside the goal angle, and prints out the shortest path using the grid. In addition, a second python file called (WebotsNavigationAlgorithm.py) deals with local navigation by estimating the location using the Midpoint GPS position reading alongside theta, the initial testing consisted of fixing the Y values in the coordinates while setting new values for X and Orientation, the initial test consisted of travelling on a straight line and using new values for X and theta, to verify functionality, the path planning was expanded further after the implementation of the object avoidance function as seen in the next section.

Object Avoidance

LIDAR

With the setup and verification that the robot can transmit its coordinates and give an accurate orientation, the second phase involved setting up a way to be able to detect objects and navigate its way around them to maintain the path to the destination. A LIDAR (Light detection and ranging) was chosen because it gives a wide range of view and allows for the robot to clear tight areas, which in the context of this application environment (warehouses) is a suitable choice

The LIDAR ray is split into 3 paths: left, middle, and right To simplify the code, the reason behind this is that having no defined sections for the rays causes the behaviour to be limited, as the overload of information due to the possibility of detecting 2 or more rays, which on its own, causes the robot to freeze in place.

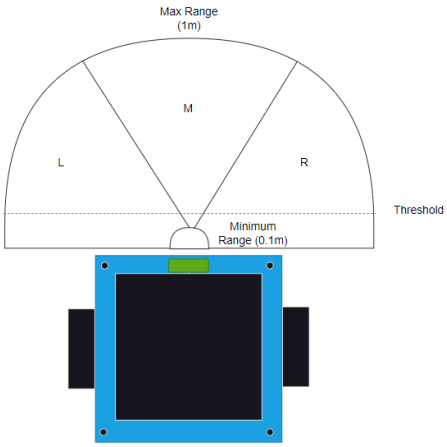

Figure 9: LIDAR field of view

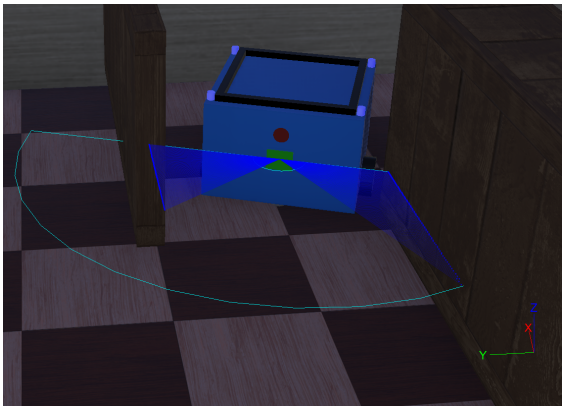

Figure 10: LIDAR navigation

As shown in figure 9, the LIDAR range is set to be (0.1 to 1) meter, giving it a wide range to anticipate the path ahead, the threshold to trigger the avoidance algorithm is 0.85 m, giving a 0.15 m range of clearance which, as shown in figure 10 allows for the navigation of tight areas and between shelves with the robot detecting objects in its threshold sector and then taking steps to find the clear path while giving itself enough clearance to avoid hitting objects, the LIDAR logic by itself is has a slight limitation especially around tight areas where the robot movement is crude and is hard to tune on its own, to overcome this drawback, a small PID function can be used to smooth out the travel.

PID

Proportional Integral Derivative (PID) is a popular control mechanism that helps in maintaining a desired output while minimizing the overshoot, the values used for the code were found by trial and error, where only the P value (Proportional gain) was set to a small value to complement the LIDAR object avoidance function

Factor	Formula
K_P, K_I, K_D	$(0.002,\ 0,\ 0)$
Heading error	$e = \theta_{\text{goal}} - \theta_{\text{current}}$
Error (deg)	$e_{\text{norm}} = \text{atan2}(\sin e, \cos e)$
Proportional term	$P_{\text{out}} = K_P \cdot e_{\text{norm}}$
PID output	$u = K_P e_{\text{rr}}$
Wheel speeds	$v_L = v_{\text{base}} - u, \quad v_R = v_{\text{base}} + u$

Table 2: Formulas used in the PID steering controller.

The formulas in table 2 were taken from the control system notes where the P (since I and D are set to zero) focuses on controlling the heading, using the two motors to modify the angle, the heading error formula is then added to the PID output, where if (u is greater than 0) then the direction nudges to the left by adding the P value to the right wheel and subtracting it from the left wheel, the reverse happens when (u is less than 0), since Kp values are small, the output produced by the changes is gradual and small, resulting in a smooth realignment towards the target, the robot speed is set low to ensure balance.

LED setup

As part of the project requirement was for the robot to have a mechanism to indicate its current activity, the approach taken to address this was to add 4 LED bulbs at the corner of the robot's upper base as shown in the previous figures, using Webots template for Led's (led.py), a function was built with 4 different colours representing 4 different states

- Blue: Path clear, follow the path
- Yellow: Robot is changing orientation
- Green: Reached (X,Y ,theta) target
- Red: Object detected!, avoid

A loop function with index i to represent each state is initialized, and is stored to be called after the end for certain tasks, for example, if an object is detected, state 1 for the LED's is called, turning the colour from Blue to Red until the object is cleared, the same process applies when the robot reaching the goal destination and heading where state 2 sets all the LED's to green, this approach and placement of the LED's ensures that the current activity can be deduced when looking at the robot from any direction, increasing awareness and safety.

Robot Behaviour setup

After the successful verification of each independent functions of the code, all of them were combined into a single program, the logic was then tested in phases to ensure all functions work with each other.

Testing

Based on the information given by the project brief, the track layout was replicated in a custom Webots world to refine the logic process. The layout of the environment is an 8x8 grid with a 0.1 floor tile size. The testing was carried out in 3 phases

- Phase 1: Verification of the A* path following and occupancy grid loading
- Phase 2: Course traversal to a hardcoded point in a straight path
- Phase 3: Course traversal through the full track

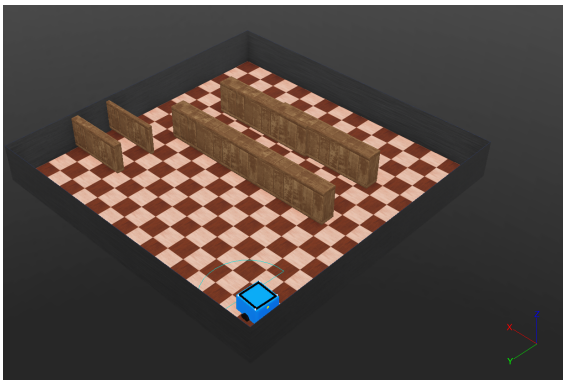

Figure 11: Track used for testing

in phase 1 the *Occupancygrid.csv* file loading verification was done using a simple conditional statement, if the robot or the goal path is in an occupied grid, a *Fail!!* statement is printed out, and the code stops executing. If the grid is clear, then *grid* will load. The A* algorithm runs once and generates the (X,Y) points from the current position until the goal point.

With the path planning verified, phase 2 involved setting hardcoded coordinates and heading angle, for example, goal (4,2) with theta (0.0) degrees is the goal point, the grid loads and prints out the path, then the robot starts driving towards the point, to prevent it from being stuck in a loop attempting to get in position for the goal points, small tolerance values for the occupancy grid, (X,Y, theta) co-ordinates

The 3rd and final phase involved a full course traversal from all functions (A*, LIDAR, PID and LED states) were active. The main goal of phase 3 was to ensure the robot could clear tight regions and between shelves, following the path without hitting objects, the course was completed without collisions and reaches the end goals within acceptable margins, this phase confirmed the correct integration of localization and object avoidance alongside LED states into a unified behaviour.

Results

The code compiles as expected and starts driving towards the goal. The travel speed is low, as going too fast causes the body to lose balance, with the front half lifting, which in real applications could cause the box being transported to fall over, opsetacle avoidance worked as intended with the robot wide range being able to detect objects up to 1m ahead and only reacts if the object is detected in the threshold region, which changes the LED state and uses the wheel rotational difference alongside the PID.

Some drawbacks were noted, the robot behaviour can be inconsistent at times, where the robot will reach the goal point but does not stop, driving around the track, avoiding objects, and stopping after a period. There were also cases were if the goal coordinates are positioned between shelves, the robot cannot position itself correctly to corner towards the shelves, causing it to take a longer path, a possible reason can be traced back to the LIDAR avoidance, which prioritizes clearance over cornering, causing detours even when the current position is close to the path.

Conclusion and Future Application

Overall, the goals of the project listed in the introduction are met both in terms of robot design and controller programming. The design can be replicated in real life using the components used in Webots (GPS nodes setup, LIDAR, LEDs, Rotational motors, and ball joints). The controller integrates all the data collected by the sensors to plan.

However the current navigation and avoidance approach lacks a more robust algorithm, the model is reactive rather than being able to build and update its environment, this application can be explored with the LIDAR using simultaneous localisation and mapping (SLAM) to map out the occupancy grid in real time and plan out a new A* path dynamically, allowing for the robot to anticipate objects and blocked paths and generating a new path with before even reaching the area mapped out, the biggest drawback is the increased computational load created by the constant mapping and path replanning, potentially affecting stability if not optimised.

The design process and implementation explored in the report provide a solid baseline for expansion and integration of more complex algorithms while using a less computationally taxing approach.

Sources

[1]: https://www.geeksforgeeks.org/python/numpy-arctan2-python/
[2]:https://www.w3schools.com/python/ref$_{m}ath_{h}ypot.asp$
$[3] : https://www.youtube.com/watch?v = qexnI4 - Cdxkt = 38s$
$[4] : https://github.com/cyberbotics/webots/blob/master/projects/samples/devices/controllers/led/led.py$