



Robotic Delivery System for Multi-Storey Condominium Complexes

Nicholas Teo Yong Yeow¹, Tew Yiqi², Yeoh Sing Hsia^{1*}, Yiauw Kah Haur¹,
Lim Ming Jun¹, Ng Yik Heng¹, Tan Hui Shan¹

¹Faculty of Engineering and Technology,
Tunku Abdul Rahman University College, Jalan Genting Kelang, Setapak, Kuala Lumpur, 53300, MALAYSIA

²Faculty of Computing and Information Technology,
Tunku Abdul Rahman University College, Jalan Genting Kelang, Setapak, Kuala Lumpur, 53300, MALAYSIA

*Corresponding Author

DOI: <https://doi.org/10.30880/jeva.2022.03.01.001>

Received 13 December 2021; Accepted 23 May 2022; Available online 30 June 2022

Abstract: In certain environments like condominium complexes, couriers are only allowed to drop off their payloads at designated drop-off stations instead of delivering it directly to the recipient. These regulations have become increasingly popular in condominiums during the Covid-19 pandemic, in part due to government SOPs encouraging their adoption. A partially automated delivery system using robot couriers that deliver the payloads from the drop-off stations directly to the recipient is developed. The system is made up of a central PostgreSQL database, a Webots simulated delivery robot and two Android applications for interfacing with the system. Two individuals operate the system: the system operator and the resident requesting delivery. The resident sends their delivery requests using the Resident android application. The system operator acknowledges the delivery requests, then loads the robot and configures it to carry the package(s) up to the resident's housing unit through the Operator android application. Navigation and obstacle avoidance are achieved through usage of the A* pathfinding algorithm. To receive the package(s) once the robot arrives, a QR code enabled compartment on the robot is unlocked using the Resident android application once again. The PostgreSQL database and Advantech WISE-PaaS Dashboard are used to display delivery requests, residents' data and robot status for system operators. NodeRED and MQTT are used to facilitate communications between robot, applications and database. In the end, the simulated robot is able to navigate a virtual environment with minimal faults and communicate properly with the applications and database. Robot speed is set at 0.27 ms^{-1} when empty, and maximum carrying weight is tested to be 20.5 kg with a delivery time of 1549 seconds for the arbitrary simulated delivery route.

Keywords: Delivery Robot, Webots, NodeRED, MQTT, PostgreSQL, Android, A*, Obstacle Avoidance, Advantech

1. Introduction

Inspired by the restriction of movement during the Covid-19 pandemic, this project involves the development of a delivery system for condominium complexes utilizing robot couriers. In closed complexes, delivery persons are sometimes not allowed to bring deliveries directly to residential units and must instead leave the deliveries at designated drop-off points. The frequency of this inconvenience has increased during the pandemic as more condominiums impose these restrictions, either independently or due their interpretation of government SOPs [1]. To alleviate this burden, our system uses courier robots to transport the deliveries from that drop-off point directly to the residential units, removing the need for residents with disabilities or a lack of time to pick up the packages manually.

*Corresponding author: yeohsh@tarc.edu.my

This also reduces inter-human contact lowering the chances of diseases spreading between people, which is in line with the goals of the Covid-19 SOPs mentioned earlier.

Indoor delivery robots have long been tested in hospitals and boast a long history of iterations and improvements [2]. In recent times these robots have been employed in hotels and production lines as well, with a few young companies pushing for adoption in residential buildings such as Meituan Dianping [3] and Lightline [4]. Indoor delivery robots inherently come with the benefit of operating in a controlled environment, making it easier to develop and account for uncertainties. The environment itself could also be modified should the need arise. To navigate their environment, path finding methods like A* (A-star) and Rapidly-exploring Random Trees (RRT) have been explored in literature [5]. To navigate vertically, operation of elevators either by robotic appendages [6] or wireless communications have been experimented with [7] [8] [9]. As for how the robot will keep its compartment secure, an electric lock unlocked on QR code scan is adopted in this project. A simple QR code verification method involves using Public Key Infrastructure (PKI) [10].

The objectives of this project included (1) developing the transporter robot and its navigation system. The core functions being implemented are path finding algorithms, obstacle avoidance algorithms and a means of operating the elevators in the condominium complexes. Aside from that, (2) the maximum weight for the robot’s payload also needed to be determined and adjusted if necessary. (3) Security measures needed to be set in place to prevent theft of the package during transit and (4) a secure wireless communications protocol needed to be studied and set up between all components of the delivery system. (5) A central database had to be established for use in a central monitoring system. (6) Lastly, a communication channel of sorts needed to be established between the recipient and the control room so that the recipient can make delivery requests.

2. Conceptual Design

The general design splits the system into 4 parts: Android applications, a central database, the robots and the communications protocol as illustrated in Fig.1.

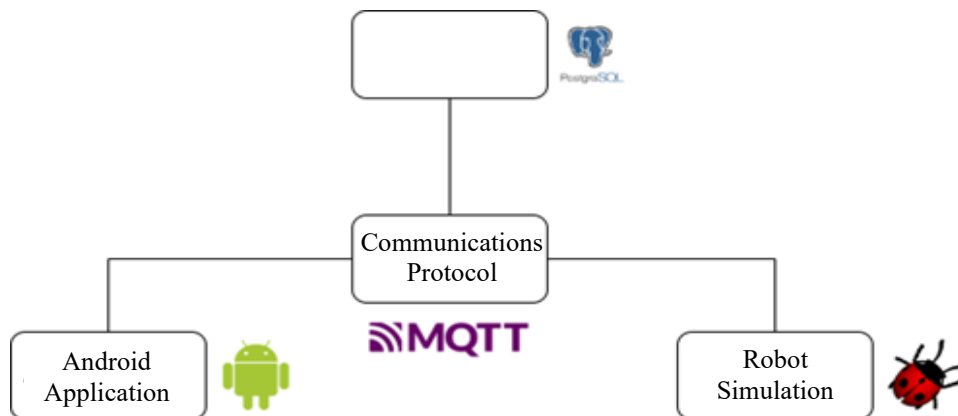


Fig. 1 - System structure

The Android applications are developed in Android Studio due to important modules such as a QR code scanner being available in online communities. PostgreSQL database is selected as its structured SQL format made it easier to understand and implement. Logging to the database is then handled by a Node-RED program and display of the database is handled by the WISE-PaaS Dashboard. Node-RED is a visual IDE used to connect different devices together [11] while the WISE-PaaS Dashboard uses Grafana to display data sources using a variety of graphs and tables [12]. The robot is simulated in Webots, a robot simulator due to travel limitations prohibiting the construction of a physical model. The path finding algorithm decided on is A*. The communications protocol which is MQTT broker (public HiveMQ broker) is applied in this project.

A large part of the decisions made are influenced by Advantech’s sponsorship. PostgreSQL and the use of MQTT as the communications protocol are due, in part to the fact that they are the recommended set up suggested by Advantech for usage with their programs like WISE-PaaS Dashboard.

3. Methodology

3.1 Simulated Robot

The primary goals of the robot are to navigate to specific destinations using the A* pathfinding algorithm and to send and receive instructions via wireless signals. This section discusses the body of the robot and its proposed hardware such as sensors, before diving into its pathfinding and communications features.

A. Robot Model

The robot is modelled in Webots based on an engineering drawing drawn during planning, with some modifications. The engineering drawing is attached as Appendix A and the robot model is shown in Fig. 2.

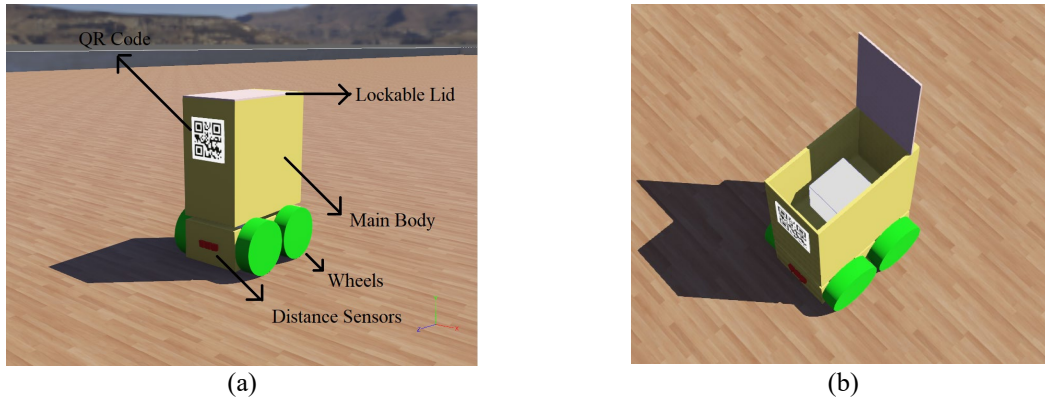


Fig. 2 - (a) Robot model; (b) Robot model with open lid and package

Dimensions of the robot body are estimated based on average of standard conventional elevator door area (800mm x 2100mm) and elevator car floor areas (1400mm x 1250mm) [13]. Characteristics of the robot as defined in Webots are:

- Length: half the depth of elevator car, 1400mm / 2 = 700 mm
- Width: half the width of elevator door, 800mm / 2 = 400 mm
- Height: half the height of elevator door, 2100mm / 2 = 1050 mm
- Weight: 8 kg
- Effective Speed: 0.26704 ms⁻¹

To turn, the robot utilizes skid-steering. This method is commonly used in robotics to turn the robot in a stationary axis, illustrated in Fig. 3.

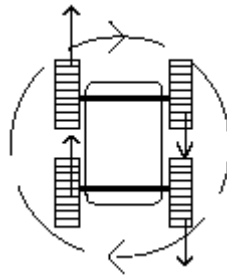


Fig. 3 - Skid steering [14]

B. Robot Sensors

The sensors modules that are attached to the robot model for specific purposes is illustrated as in Table 1.

Table 1 - Sensors used

Sensors	Geomagnetic Sensor (Compass)	Distance Sensors	Position Sensors (Wheel Encoder)
Objective	To accurately feedback the 8 cardinal directions the robot is facing for skid-steering.	To detect dynamic obstacles in order to plan a new path.	To calculate how much the wheel has moved to accurately determine distance travelled.
Implementation in Webots	Adding a Compass Node in the Robot Scene.	Adding a Distance Sensor Node in the Robot Scene.	Adding Position Sensor Node to the Wheels Hinge Joint Node.
Range of Values	0-360°	0-15300 (0 m to 1.53 m detection in the anterior area)	0-infinity (meter/radian)

C. Navigation

The A* path planning algorithm is modified from AtsushiSakai’s algorithm found on Github [15]. The map containing all walls, obstacles, the starting point and goal has to be established first in the code. Then, that map is fed into the path planning algorithm to determine the best path which the robot then follows.

In A* path planning, the map is essentially broken into nodes that are 1 meter apart from each other. The coordinates of every obstacle on the floor plan (walls, pillars etc.) must be manually defined in advance before the robot can generate a path. Once a complete map is defined, a start and an end node are selected and the A* algorithm systematically searches for the shortest path between the two points. In this case, start node is defined as the control room housing the robot while the end node is the resident’s lodgings. Once the path is defined, the robot is programmed to follow it to the destination. The path planning process can be seen using “matplotlib.pyplot”, a public Python library as shown in Fig. 4.

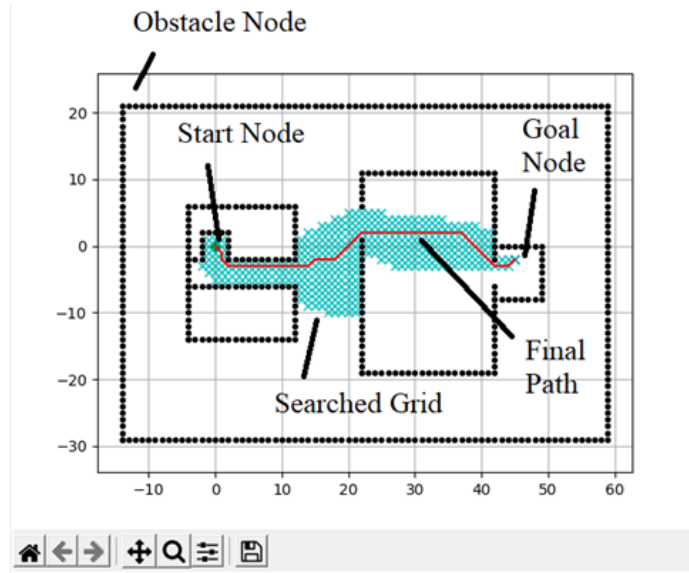


Fig. 4 - A* path planning

Obstacle Avoidance

The obstacle avoidance technique involves scanning the area ahead of it for obstacles and adding them to the A* map if any are detected. Suppose an obstacle is detected, the robot halts its movement and recalculates its path to the destination using the updated map. The new updated path will have considered the obstacle and the robot will navigate around it to the best of its ability.

As mentioned earlier, the map is separated into nodes that are 1 meter apart from each other as shown in Fig. 5. Moving diagonally between nodes results in 1.4142 meters due to Pythagoras’s Theorem. Obstacle avoidance works by using distance sensors to scan up to 1.53 meters ($1.4142 + 0.1158$ error margin) ahead of the robot every time it moves between nodes. If an obstacle is detected, it gets added to the A* map and the path is recalculated. This is repeated every time an obstacle is encountered. If the algorithm determines that there is no available path to the destination, the robot will stall.

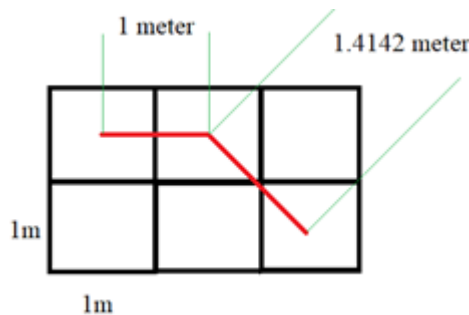


Fig. 5 - Illustration of path tracking

3.2 Communications Processes

Communications with the robot are done over WiFi. Constant connection to the internet is an assumption used for the following communications solutions. Communication is facilitated by the MQTT protocol which is widely used in IoT applications. The protocol is easy to use and is lightweight, allowing it to run on simple or low specification hardware without compromising other functions.

The protocol is structured as a central broker redirecting messages between multiple clients. As described by [16], the protocol is governed by a publish-subscribe model. Publisher clients can publish messages to the broker under specific pseudonyms (topics) and subscriber clients that are subscribed to those topics will receive those messages. Any client can be both a publisher and subscriber at the same time.

In this system, the robot, the database and the application are all clients publishing/subscribing to the same broker (provided by HiveMQ). The relevant components only publish and subscribe to their unique topics, so the messages never go to the wrong clients.

A. Elevator Communication

To navigate a multi-story building, the robot needs to be able to operate elevators. For this project, operation of the elevator is done wirelessly through MQTT as well. By publishing checks and requests to the same topic that the elevator is subscribed to, the robot is able to successfully send instructions to and operate the elevator.

For example, if the robot needs the elevator to pick it up at Floor 2, the robot can send the message “2” to the topic “Elevator No. 2 Requests”. Elevator No. 2 which is currently subscribed to the “Elevator No. 2 Requests” topic will receive a copy of the message “2”. Knowing how this works, the elevator is then programmed to go to Floor 2 upon receiving the message “2”, so it now services the request and heads to Floor 2. This only illustrates the concept; the actual topics and messages are different in practice and compatibility of the elevator with MQTT is only assumed in this model. It should be noted that the elevator and robot need to be using the same MQTT broker or at least two different but connected brokers.

The information sent to the elevator include the robot’s presence in the elevator, the current floor and the desired floor. The flow of operations and the topics and messages are shown in Fig. 6.

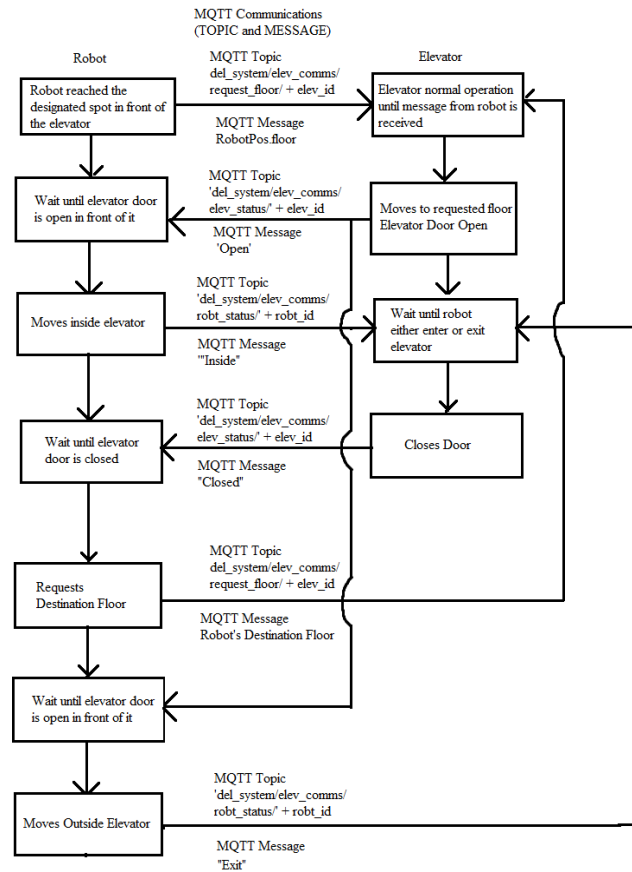


Fig. 6 - Elevator communications flowchart

B. Request of Delivery

Once a resident’s package has arrived at the drop-off point or control room, the resident can make their delivery request. No notification method is included in the scope of this project, so residents are expected to have been notified elsewhere in advance (direct communication with operator or notification from deliveryman/online shopping platform). Requests are handled through the application as they are made through MQTT, so the app is needed to facilitate these communications. Details on the app’s inner workings are explained in section D Android Application.

In brief, the resident keys in their “house number” as the delivery destination and is prompted on whether or not the locked robot should be unlockable by all phones or just this one. Allowing all phones to unlock it lets individuals who live in the same unit unlock the robot in the original person’s place, which is the justification for allowing the option. After keying in their information, the application will send it to the control room via MQTT where it will be logged into the database and notify the operator.

There is a gap in the process here whereupon being notified of the request, the operator themselves must manually configure a robot to send to that destination. This allows flexibility and error checking in this crucial stage so that issues such as a wrong house number, missing packages or un-operable robots can be resolved by the operators. If it were fully automated, problems like these may not be caught in time.

Proceeding on, the operator sends the configurations to the robot using a separate application specifically for operators. The robot’s destination and lock password are set in the app as well as the ID of the robot, these configurations are being sent to. After entering the configurations, the robot begins its journey upon receiving a final confirmation signal from the app. Aside from communications with the elevator, the robot does not communicate during its journey.

C. Reception of Package

Once the robot arrives at its destination, the resident will unlock it by scanning a QR code stuck onto the robot’s body using our developed Resident Application. The process is designed this way because the unlock signal is sent to the robot via MQTT also, so a dedicated application is required to facilitate the MQTT communications. Once the robot receives and verifies the unlock signal, the lid covering the internal compartment will open.

The lid will only close once the resident has confirmed the reception of their package through the application (handled through MQTT also). In cases of human error (forgetting to confirm reception etc.), the robot employs an internal timer that forces it to return to the control room after a set period of waiting. The flow of operations here is detailed in Fig. 7.

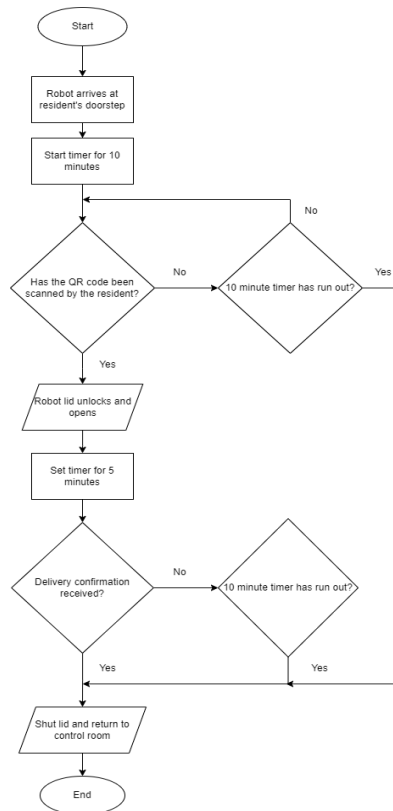


Fig. 7 - Reception of package flowchart

D. Android Application

As mentioned in section B Request of Delivery, the resident application requires the user to key in their “house number” and confirm whether the robot should be unlockable with this phone only. The application achieves the latter by generating a random password and saving it to the phone’s memory upon requesting the delivery. This password is sent to the database and will be configured as the robot’s unlock password. This way, only the phone that requested the delivery will have access to the correct password. If “Limit pickup to this phone” is unchecked, the password sent to the database and the password used to unlock the robot will be taken from the “house number” field. In this case, any person who tries to unlock the robot while the “house number” is keyed into the field will be able to open it.

When the resident scans the QR code on the robot, it sends its password to the robot over MQTT. As long as that password matches the configured one, the lock will disengage. The QR code here contains information on the ‘unlock’ topic that robot is configured for, ensuring that the unlock password is sent to the correct robot upon scanning.

Two major libraries used in Android Studio are Zebra Crossing and Eclipse Paho. Zebra Crossing (ZXing) is a barcode library that contains the QR code scanning functionality [17] and Eclipse Paho a library which enables MQTT functionality [18]. Both are open source and widely available.

Verification of Application Functionality

The Android applications are tested first on whether they could send and receive the correct data using the correct MQTT topics. These functions are tested using MQTTBox, a tool that acts as an easily configurable MQTT client [19]. By configuring MQTTBox to receive messages from the same MQTT topics published by the application, the communications functionality of the application could be verified.

The QR code scanner function is also tested on outputting the correct MQTT signal upon successful scan, although the scanning page itself is not shown in the test results. These tests are carried out as shown in Fig. 8, Fig. 9 and Fig.10. In the end, the functionality of the application is verified to produce the appropriate responses.



Fig. 8 - (a) Resident application page; (b) MQTTBox output

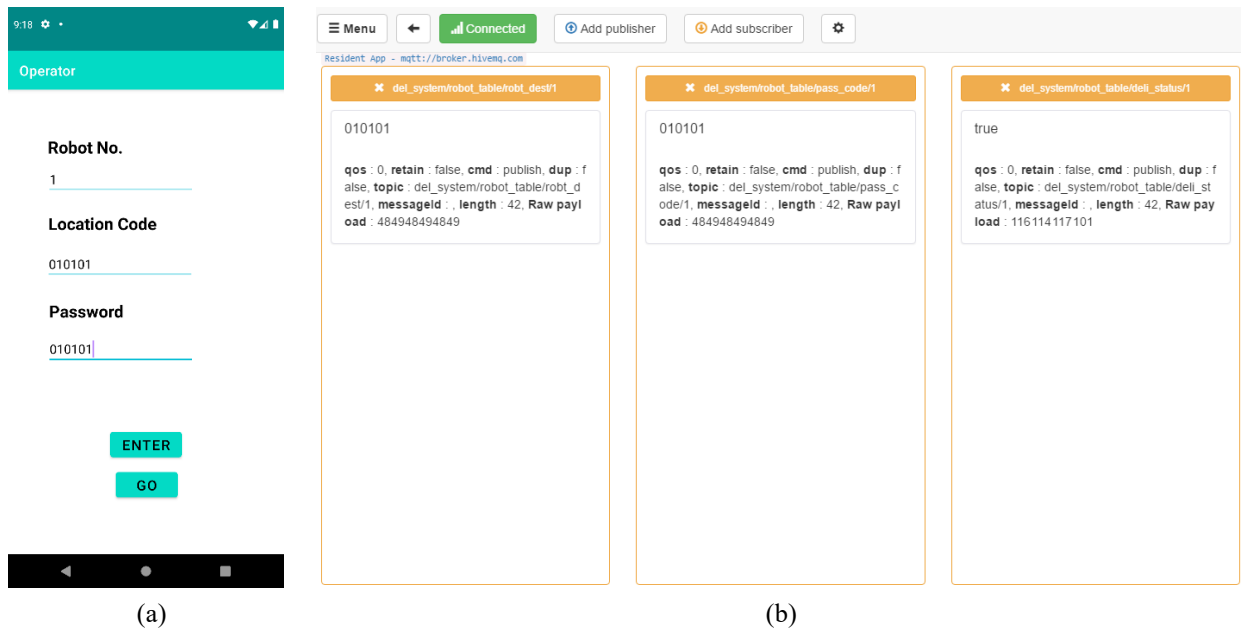


Fig. 9 - (a) Operator application page; (b) MQTTBox output

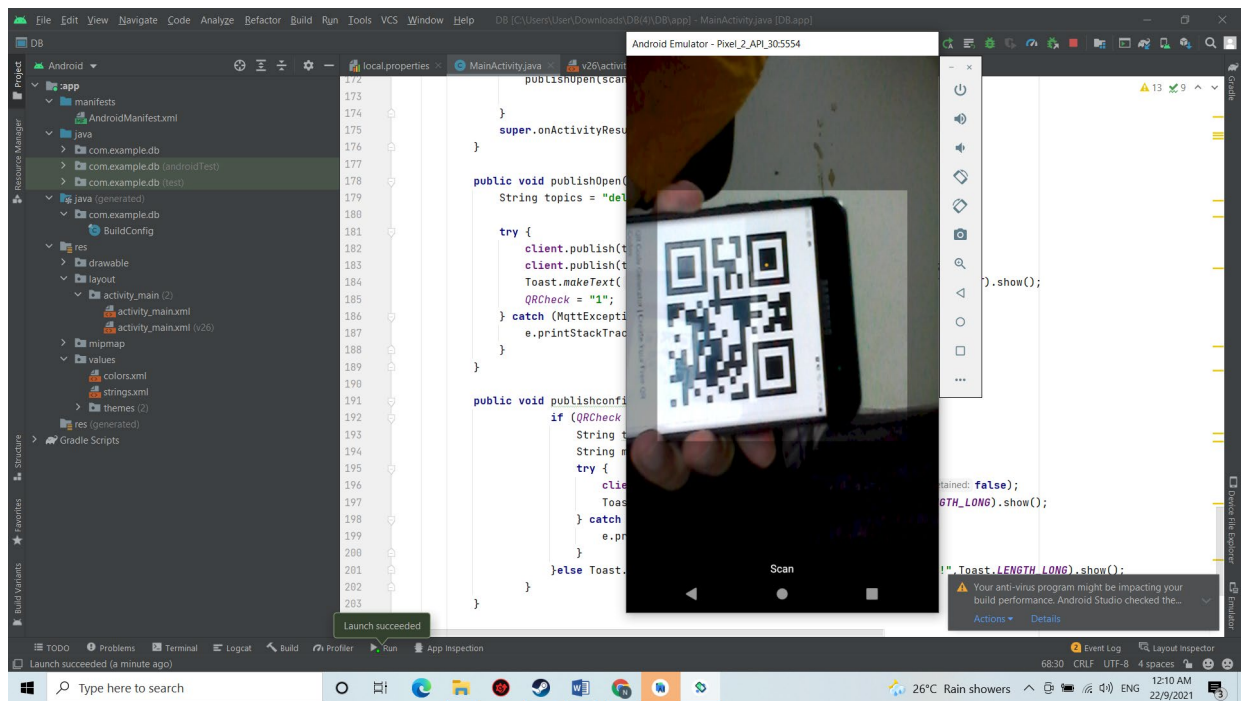


Fig. 10 - QR code scanner tested in android studio

3.3 General Operation

The usage of the delivery system begins with the resident using their Resident Application to send a delivery request (call) to the database consisting of their address (location) and a ‘password’. The request will appear in the database and notify the operator of the request. Once the operator is aware of it, they can use their Operator Application to configure a free robot to service that delivery. Important configurations are the delivery location and the password for the robot’s locked package compartment. Two configurations: ‘location’ and ‘password’ are sent by the Resident Application earlier.

Once the robot has been configured and loaded with the package, it would be sent on its delivery journey. By using A* pathfinding, the robot calculates a path to the resident’s address using map data that has been pre-programmed into it from the beginning. A prototype dynamic path finding algorithm allows the robot to avoid unexpected obstacles

under the correct conditions. Once it reaches the elevator, the robot uses wireless communications through the MQTT protocol to operate it to the correct floor. From there it navigates to the correct unit and waits to be unlocked.

To unlock the compartment, the resident uses Resident Application to scan the QR code sticker on the robot. The QR code sticker contains the MQTT topic needed to communicate with the robot. The application sends the same ‘password’ as before and if it matches with the ‘password’ inputted into the robot by the operator, the robot will unlock. After it unlocks, the resident removes the package and confirms their receipt through their Resident Application. The robot will then navigate back to its starting point and reset all delivery parameters relevant to itself in the database. This process is summarized in Fig. 11.

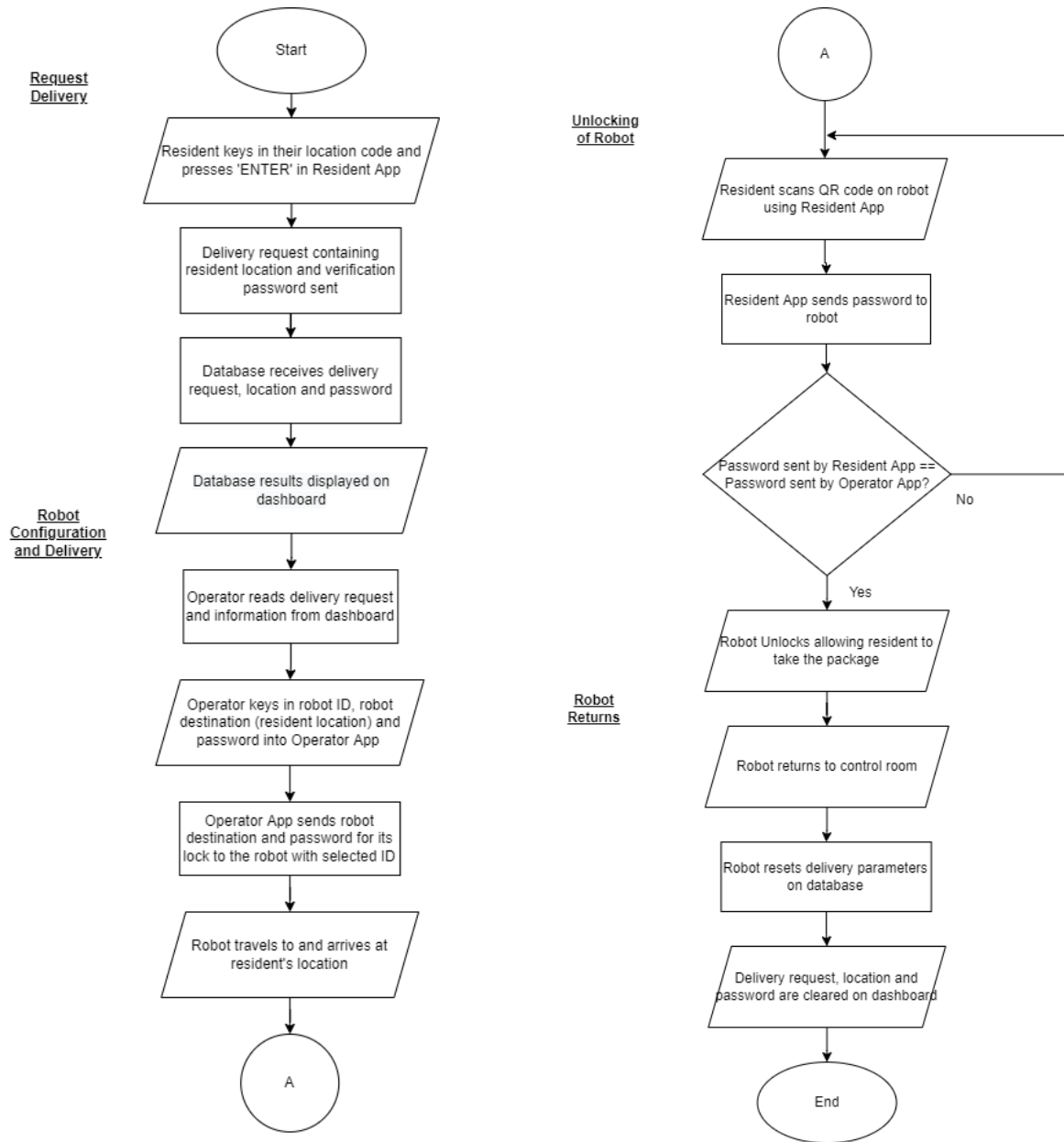


Fig. 11 - Flowchart of operational process

3.4 Communications Framework

In general, all communications between the Android applications and the simulated robot go through the MQTT broker since they communicate using the MQTT protocol. MQTT works by sending messages under specified ‘topics’ to a central MQTT broker. That broker redirects the messages to all clients subscribed to those topics [13], effectively delineating which clients should receive which messages. Communications between those two components are done directly across the broker only.

As all messages are passed through the broker, some of the relevant messages are taken and logged in the database as necessary. The Node-RED program is in charge of ‘listening’ in on the specified MQTT topics, receiving their messages and using a combination of the topic name and message content to log data into the database. The WISE-PaaS Dashboard which is connected to the PostgreSQL database data source displays the data and updates in real-time.

The ‘pgAdmin’ is a backend tool used to configure the PostgreSQL database. While it is not used in general operations, it is useful in keying in the initial parameters in the table like residents’ names and addresses as well as modifying data directly should the need arise. The whole communication framework is illustrated in Fig. 12.

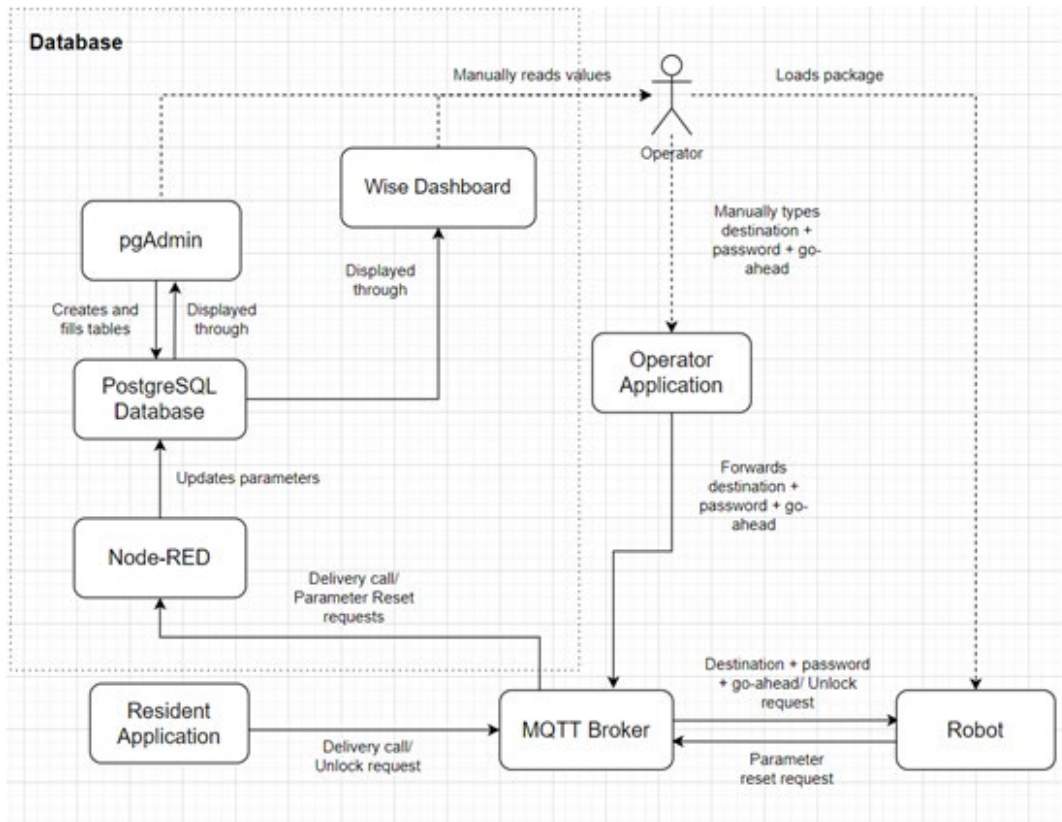


Fig. 12 - Overall communications diagram

3.5 Database

As mentioned in earlier sections, a database is used to hold data on each household’s location code, delivery request (CALL) status and requested unlock password. The database used is a PostgreSQL database, hosted by Advantech on their servers. As the name suggests, it is a SQL database which means it is ideal for cataloguing structured data. Structured data meant it would be easy to understand as well as display in tables, which is why it is chosen.

Setup of the database during development is aided by ‘pgadmin’, a backend tool that allows direct configuration of database settings and modification of database data. It is frequently used to change variable names and setup test cases.

Even though the database is set up, a visual dashboard is needed to allow operators to observe the necessary data without needing to utilise backend programs like ‘pgadmin’. WISE-PaaS Dashboard is used to display the data from the PostgreSQL database in a clean and easy to understand manner. The database is tested to ensure it is updated properly when data is sent, confirming that the WISE-PaaS Dashboard and Node-RED program work correctly, in addition to the PostgreSQL database being properly set up. These tests are similar to the application functionality tests, testing the database updates in response to the Resident App, Operator App and Simulated Robot. In the end, the database updates correctly after launching as observed in Fig. 13 and Fig. 14.



Fig. 13 - (a) Android application; (b) WISE-PaaS dashboard showing database

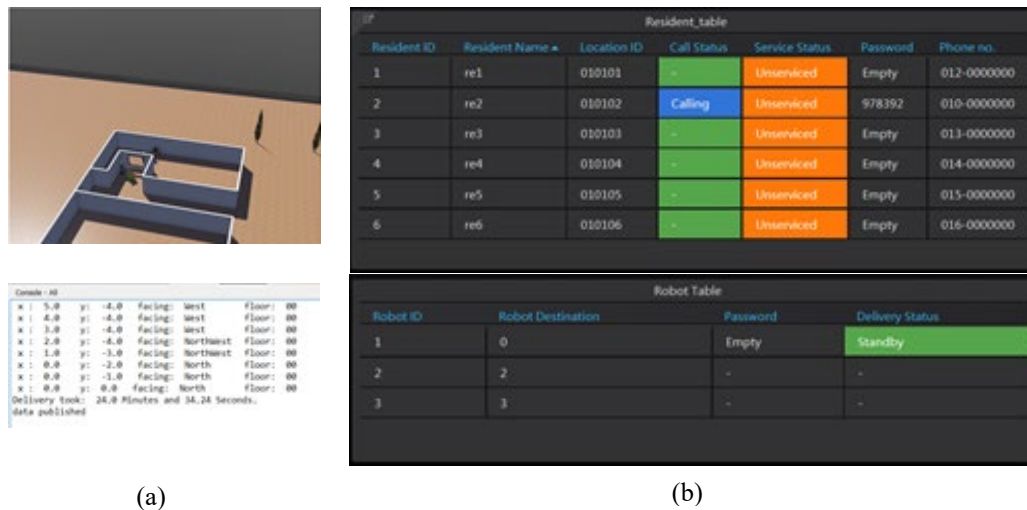


Fig. 14 - (a) Simulated robot; (b) WISE-PaaS dashboard showing database

4. Discussion and Results

4.1 Simulated Robot

Tests on the robot focused on the path finding, obstacle avoidance and package weight limit of the robot.

A. Verification of Delivery Operations

Verification is done modelling a simulated apartment building in Webots and having the robot navigate the environment. Isometric and planar views of the test environment are shown in Fig. 15. The elevator is modelled as a single room with doors on both sides as modelling a vertically moving elevator would be unnecessarily complicated. Its dimensions follow those of the reference elevator [13] used for selection of the robot’s size. The dimensions of the entire test environment are shown in the sketch in Fig. 16.

During the tests, the robot is found to travel correctly to the destination housing units after receiving its instructions from Operator App. Path finding and obstacle avoidance work correctly in most cases, successfully navigating to the correct unit despite unexpected obstacles. Table 2 shows tests where the robot follows an A* path between its starting location and destination housing unit. A third test show a robot’s updated A* map after sensing an unexpected obstacle ahead of it during the return trip.

Obstacle avoidance is unable to be implemented when entering and exiting the elevator during to the unpredictability of people walking in and out of such a cramped space. The robot also exhibits a small positioning error when returning to its starting point, which adds up to compromise its navigation after repeated use without reset.

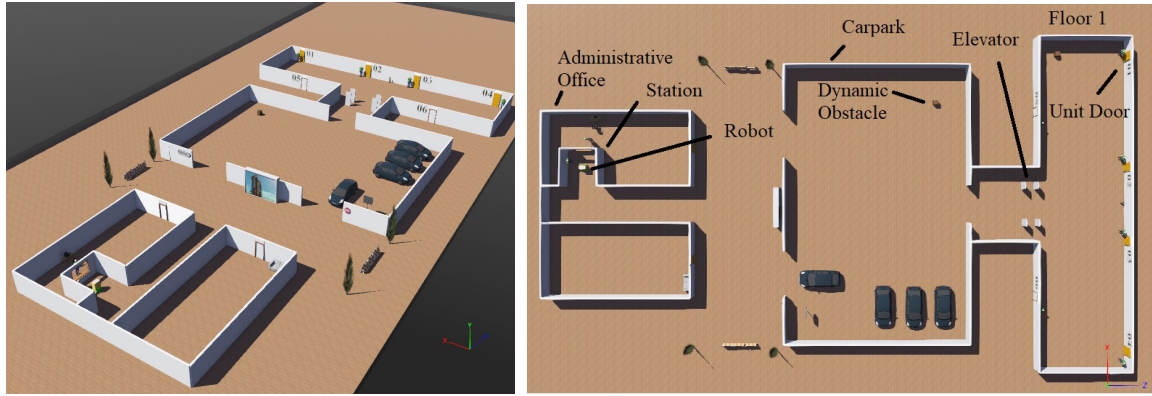


Fig. 15 - Robot test environment in Webots

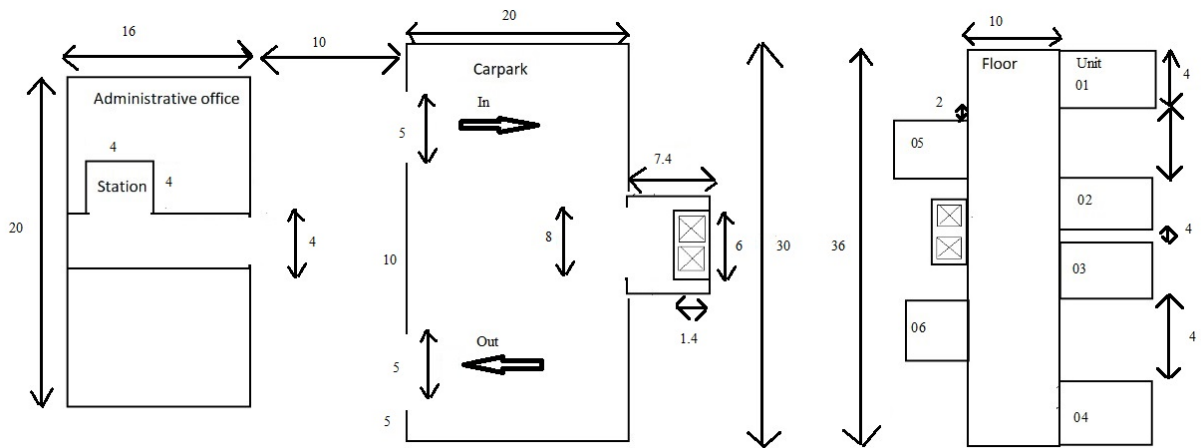


Fig. 16 - Rough sketch of the test environment with dimensions in meters (m)

B. Package weight limitations

The maximum package weight the robot accommodated is 20.5 kg after testing repeatedly with increasing loads. The test path is between the starting area and unit 6, time taken to complete the delivery cycle is timed using the simulator's built-in timer. Based on the dimensions in Fig. 16, distance travelled is very roughly estimated as 55.87×2 m, though the actual distance travelled by the A* path is likely to differ.

From the tests, the relation following relation shown in Fig. 17 is obtained. For a wide range of mass (0.5-18 kg), the delivery time remains below 1530 seconds (25.5 minutes) but increases to above 1550 seconds (25.8 minutes) for masses 18.5 kg and up. The delivery cycle assumed no scanning of QR code to remove human error, so 10 minutes of waiting at the doorstep are included in the final times. It should be noted that the robot does not translate when it is turning, so estimated distance*speed = time taken will not produce accurate results.

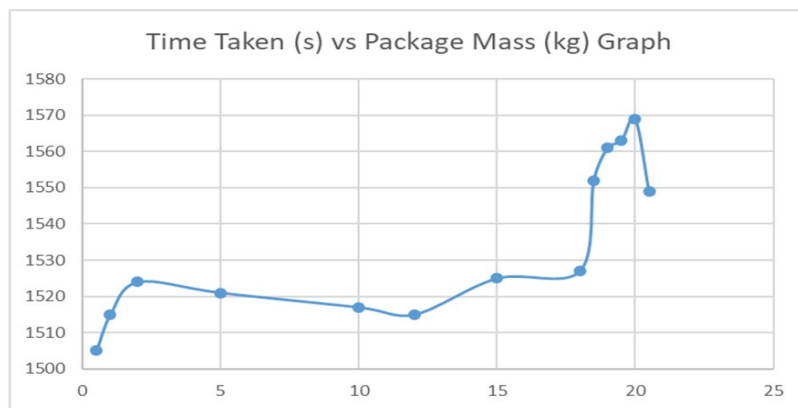
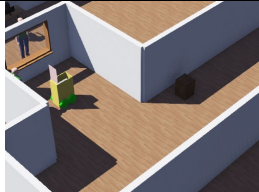
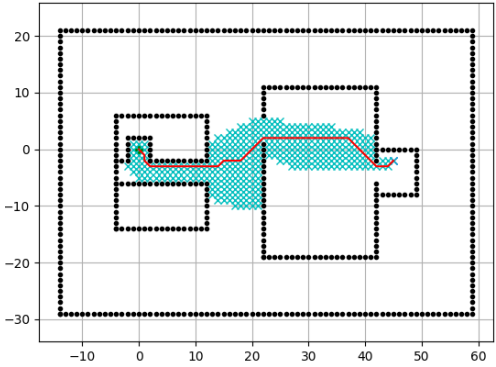
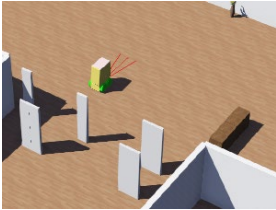
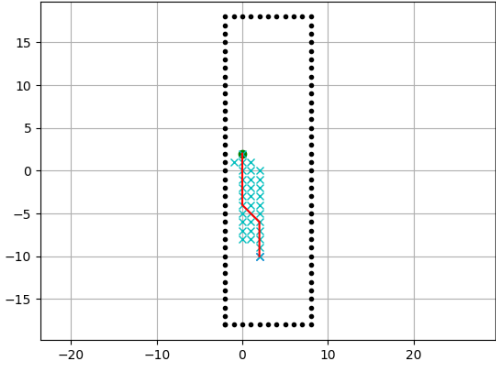
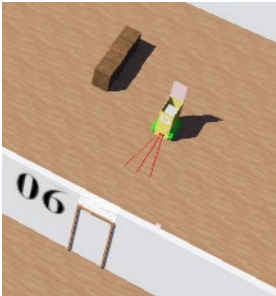
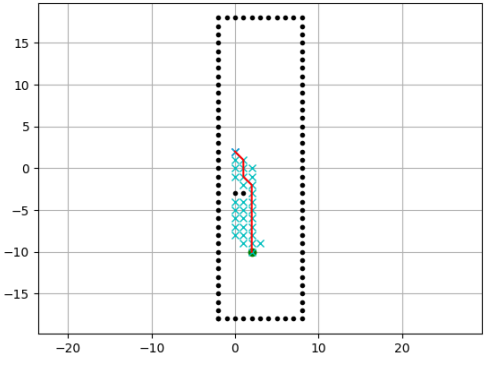


Fig. 17 - Time taken against package mass

The effect on mass on the delivery time is overall negligible (~1%), though for masses 21 kg and above, the robot ceases to function properly. The excessive weight increases the robot’s odometry calculation error, hence accumulatively it will start to move for a little less than the distance required to reach the center of each node within the A* map. As a result, in cramped areas the robot misjudges its travelled distance and moves less than the distance required to reach the designated spots. In this case, it fails to move the necessary amount of distance to enter the elevator, causing the elevator door to clash with the robot body. This is considered as a fail state, hence weights above 21 kg are considered unacceptable.

Table 2 - Obstacle avoidance in A*

Path Planning No.	Robot Translation in Webots (x, y)	3D Window	matplotlib A* Path Planner Representation
1 (Exiting Standby Phase)	(0,0)		
2 (First Floor Reached)	(52.3, -1.98)		
3 (Returning from housing unit)	(54.4, -13.9)		

5. Conclusion

A delivery robot has been developed in Webots that successfully implements the A* pathfinding algorithm to navigate and avoid obstacles to deliver loads to selected, predefined destinations. It is able to receive instructions wirelessly and communicate with the PostgreSQL database and Android applications to coordinate delivery location, delivery status and unlocking of the package compartment. Two simple applications are also developed to simplify sending instructions to the robot to improve ease-of-operation.

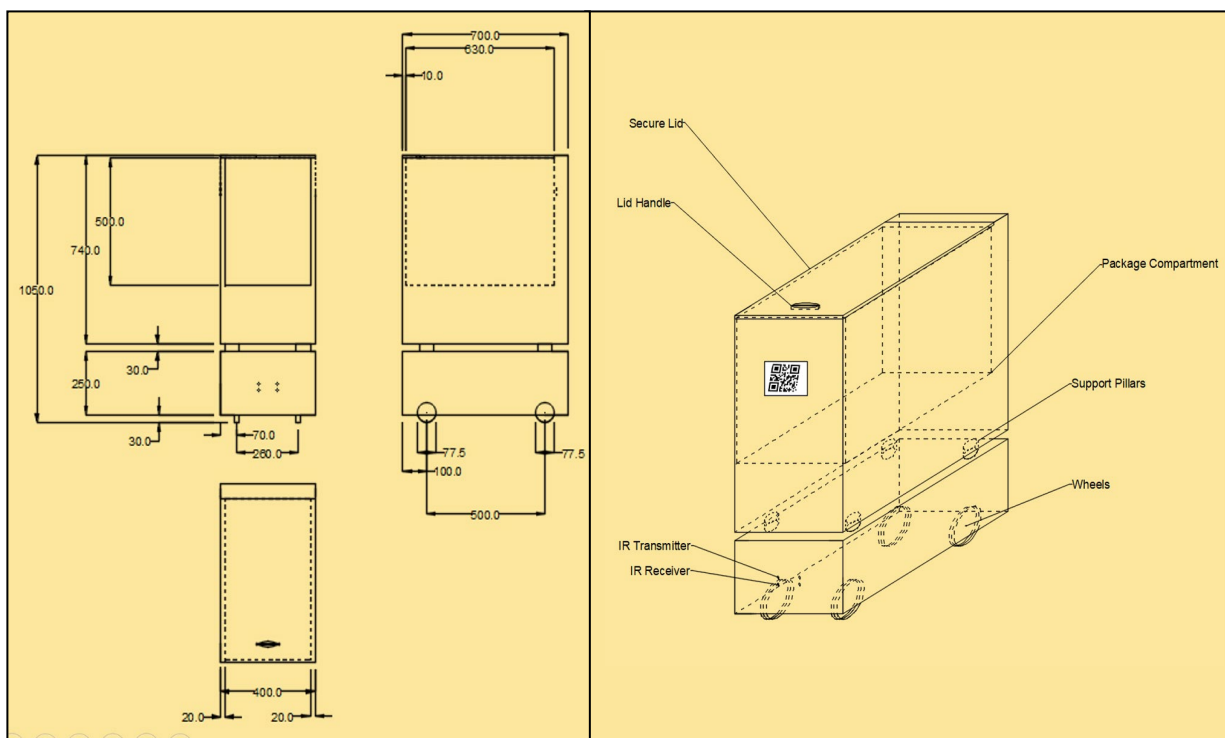
In the end, the traversal and communications ability of the robot are verified through simulations in Webots, performing consistently to deliver loads to the defined locations whilst navigating around unexpected obstacles. The effect of increasing load weight on the robot is also investigated, showing consistent delivery times of around 1525 seconds (25.5 minutes) for package weights below 18 kg, with negligible differences up to 20.5 kg. Maximum capacity is determined to be 20.5 kg with delivery time of 1549 seconds, though weights of 21 kg and above resulted in failed traversal of the robot.

Small calibration errors were noted across successive trips which eventually caused performance issues. While the calibration error can be corrected manually in practical applications, the eradication of the errors remains an avenue of improvement.

Acknowledgment

The authors would like to thank Advantech for supporting the project with their WISE-PaaS Dashboard and PostgreSQL servers, as well as providing financial support as the sponsor for this project.

Appendix A: Engineering Drawings of Robot



(a) Overall design with dimensions via AutoCAD (b) Isometric overall design with labels via AutoCAD

References

- [1] Majlis Keamanan Negeri. (2021). KPKT PKP SOP Pengurusan Strata Kemaskini 11 Jun 2021 [Online]. Available at: <https://asset.mkn.gov.my/web/wp-content/uploads/sites/3/2019/08/KPKT-PKP-SOP-Pengurusan-Strata-Kemaskini-11-Jun-2021-converted.pdf> [Accessed: 9 Sept 2021]
- [2] Niechwiadowicz, K. & Khan. Z. (2008). Robot Based Logistics System for Hospitals-Survey [Online]. IDT Workshop on Interesting Results in Computer Science and Engineering, August. Available at: <http://www.idt.mdh.se/kurser/ct3340/archives/ht08/papersRM08/30.pdf> [Accessed: 9 Sept 2021]
- [3] Soon, S. (2019). In Japan and China, robots could soon deliver food to your doorstep [Online]. Available at: <https://www.cnbc.com/2019/10/02/in-japan-and-china-robots-could-soon-deliver-food-to-your-doorstep.html> [Accessed: 18 Feb 2021].
- [4] Light Line Delivery Corporation. (2021). Light Line Delivery website [Online]. Available at: <https://www.lightlinedelivery.com/> [Accessed: 27 Feb 2021].
- [5] Gul, F., Rahiman, W., & Alhady, S. S. N. (2019). A comprehensive study for robot navigation techniques [Online]. Cogent Engineering, 6(1). Available at: <https://doi.org/10.1080/23311916.2019.1632046> [Accessed: 15 Feb 2021].

- [6] Wang, F., Chen, G., & Hauser, K. (2018). Robot Button Pressing in Human Environments. Proceedings - IEEE International Conference on Robotics and Automation, 7173-7180. <https://doi.org/10.1109/ICRA.2018.8463180> [Accessed: 13 Feb 2021].
- [7] Stricker, R. et al. (2012). Konrad and Suse, two robots guiding visitors in a university building [Online]. Informatik Aktuell, 2012, 49-58. Available at: https://doi.org/10.1007/978-3-642-32217-4_6 [Accessed: 15 Feb 2021].
- [8] Cavallo, F. et al. (2014). Development of a Socially Believable Development of a Socially Believable Multi-Robot Solution from Town to Home [Online]. Cognitive Computation, 6(4), 954-967. Available at: <https://doi.org/10.1007/s12559-014-9290-z> [Accessed: 13 Feb 2021].
- [9] Chang-won, L. (2020). Autonomous food delivery robots to be introduced at new apartment complex [Online]. Available at: <https://www.ajudaily.com/view/20200703091051590> [Accessed: 10 Feb 2021].
- [10] Purnomo, A. T., Gondokaryono, Y. S. and Kim, C. (2016) Mutual authentication in securing mobile payment system using encrypted QR code based on Public Key Infrastructure [Online] 2016 6th International Conference on System Engineering and Technology (ICSET), pp. 194-198. doi: 10.1109/ICSEngT.2016.7849649 [Accessed: 6 September 2021]
- [11] Node-RED. (n.d.) Node-RED Website [Online] Available at: <https://nodered.org/> [Accessed: 8 Sept 2021]
- [12] Advantech. (n.d.) Advantech Wise-Marketplace [Online] Available at: <https://wise-paas.advantech.com/en-us/marketplace/product/advantech.wise-paas-dashboard> [Accessed: 8 Sept 2021]
- [13] Mitsubishi Electric. (2020) Passenger Elevators, [Online] Available at: https://www.mitsubishielectric.com/elevator/products/basic/elevators/nexiez_mr/pdf/product_guide.pdf/ [Accessed: 6 September 2021]
- [14] Shuang, G. et al, (2007), Skid Steering in 4-Wheel-Drive Electric Vehicle [Online]. Semantic Scholar. Available at: www.semanticscholar.org/paper/Skid-Steering-in-4-Wheel-Drive-Electric-Vehicle-Shuang-Cheung/5ca8d2b5b49532ad385d95d961c5e863f1d666b8 [Accessed: 6 September 2021].
- [15] AtsushiSakai, (2020). GitHub robotics algorithms. [Online] Available at: github.com/AtsushiSakai/PythonRobotics (Accessed: 6 September 2021).
- [16] Paessler. (n.d.) IT Explained: MQTT [Online]. Available at: <https://www.paessler.com/it-explained/mqtt> [Accessed: 8 Sept 2021]
- [17] ZXing Project. (n.d.) ZXing Project [Online] Available at: <https://github.com/zxing> [Accessed: 8 Sept 2021]
- [18] Craggs, I., (2021) Eclipse Paho | The Eclipse Foundation. [Online] Available at: <https://www.eclipse.org/paho/> [Accessed: 5 September 2021]
- [19] The Things Stack (n.d.) MQTTBox [Online] Available at: <https://www.thethingsindustries.com/docs/integrations/mqtt/mqttclients/mqttbox/#:~:text=MQTTBox%20is%20a%20cross%20platform,The%20Things%20Stack%20MQTT%20Server.> [Accessed: 5 September 2021]