AY23/24 Semester 2

EG2310 Group 13 G1 Report

Name	Student ID
Dylan Liew Yan Hong	A0255312N
Jervin Tan Si Kai	A0286659E
Kashfy Ilxilim Bin Zulkarna'in	A0272245H
Lim Jing Heng	A0272417E
Toh Yan Ting	A0261533L

1. ***Problem Definition & Requirements***

The overall objective of this project is to construct and design a robot that is able to navigate through a maze while producing an accurate map of it. The robot will then be faced with two doors. In order to know which door to go through, the robot has to communicate with a web server, which will also unlock the selected door. After locating and entering the unlocked door, the robot will have to launch five ping pong balls into a red bucket. All this has to be done within 25 minutes.

Mission Requirements
Mapping: The robot must map out the maze elements and develop a full SLAM map of the area.
Navigation: The robot must navigate through the maze in order to reach the "elevator".
Localisation (Coordinates): The robot must locate the door based on coordinates given.
Communication: The robot must unlock a door by making a HTTP call to a web server.
Localisation (Object): The robot must locate the red bucket in the room.
Launcher: The robot must launch five ping pong balls (payload) into the red bucket

2. ***Literature Review/Research***

Based on the deconstructed mission requirements, we have researched and compiled a list of technologies and algorithms available that we could use to accomplish the mission objectives.

a. Mapping

In order to produce an accurate map of the maze, simultaneous localization and mapping (SLAM) algorithms can be used to simultaneously fuse sensor data while the robot is moving through the maze. This produces a map of the unknown maze environment. With this map, we can also subsequently run path-finding algorithms for the robot to exit the maze.

Method	**Pros**	**Cons**
Visual SLAM - making use of image data from cameras, tracking key features such as edges and contours to build the SLAM map.	- 3D map, more information - Cameras are cheaper	- High computational requirements - Less precise distances - More lighting-dependent
LiDAR SLAM - making use of the 360° LiDAR distance readings, and matching together consecutive scans to build a larger SLAM map.	- LiDAR is included - Less computational requirement - Precise distances	- LiDAR is bulky and costly - LiDAR update rate is slower - Unable to detect black or transparent objects

b. Navigation

i. Markers to aid with Navigation

We are able to place any markers in the map and use them to guide our robot during this mission, either to point the robot towards the two doors, or to guide the robot out of the maze.

Method	Pros	Cons
Line Following - Lay black tape to trace out the route the robot should take to complete the maze	- Cheap, simple sensors - Easy software implementation	- Time-consuming to lay and retrieve tape every test session
Waypoints - Paste coloured paper / AprilTags to denote waypoints, let the robot know if it should go left or right when it reaches specific junctions	- Easy (physical) setup and teardown	- Requires a camera and related vision implementation, which might be computationally expensive

ii. Path Finding Algorithms

Method	Pros	Cons
Depth / Breadth First Search - algorithms utilise a stack / queue to explore different nodes in a maze, until it finds the desired location.	- Easy implementation	- Inefficient back-and-forth movement while exploring nodes
A* - algorithm which extends Dijikstra's algorithms, utilising a heuristic function that estimates the "cost" from each closest node to the intended location.	- Efficient and effective	- Difficult implementation - Requires maze to be fully mapped

c. Localisation (Coordinates)

At the start of the mission, the coordinates of the two doors are made known to the robot. In order to head towards those coordinates, the robot would first need to know its own current coordinates and location.

Method	Pros	Cons
Inertial Measurement Unit (IMU) / Encoders - The 6-DOF IMU and encoders on drive motors provide the robot with heading and displacement data.	- Easy software implementation	- Gyro is prone to drift - Encoders are prone to wheel slippage
Local Positioning System - Uses short-range beacons and a receiver to triangulate signals in order to obtain accurate absolute coordinates	- Very accurate - Easy setup	- Difficult implementation - Expensive beacons
Markers - By placing markers (e.g. coloured paper on floor) at pre-set coordinates, the robot can localise itself when it detects these markers	- Easy software implementation	- Difficult to set up to the exact location
SLAM - generated SLAM maps, when matched to current LiDAR data, can help localise the robot.	- Easy software implementation	- Accuracy depends on starting position

d. Localisation (Object)

The location of the bucket in the room is not known in advance. Hence, the robot has to be able to detect and localise the bucket before it can deposit the payload inside.

Method	**Pros**	**Cons**
Camera - Image data can be analysed to detect the bucket via its shape and colour	- Red bucket should be easily identifiable by colour	- Computationally expensive - Limited distance data
Depth Camera - Depth maps can be analysed to detect the bucket via its shape	- Accurate distance data for aiming of ball	- Expensive - Difficult implementation
LiDAR - 2D distance data can be analysed to detect the bucket via its shape	- Accurate distance data for aiming of ball	- Difficult implementation and detection
Colour Sensor - The bucket is the only red object, which the colour sensor would be able to detect.	- Cheap sensor - Easy implementation	- Very close proximity required before triggering

e. Launcher

Method	**Pros**	**Cons**
Flywheel - a fast-spinning wheel can propel the payload into the target	- Controllable strength on-the-fly	- Potentially inconsistent and unreliable - Bulky
Catapult - Loads an arm (e.g. with a rubber band) and flings the payload towards the target.	- Simple mechanism	- Potentially inconsistent and unreliable - Strength of spring / band might degrade over time - One payload only
"Piston" Shooter - Loads a "piston" with a spring or elastic, to hit and impact the payload upon release.	- Straight trajectory	
Extendable Arm - An actuated arm that can extend above the bucket to deposit the payload	- Accurate, consistent	- Bulky, cause imbalance in Centre of Gravity
Conveyor Belt - Load the payload on an inclined conveyor belt, which will then drop into the bucket at the peak of the belt	- Looks cool	- Bulky, prone to failure due to many moving parts

3. Concept(s) Design

a. Launching Mechanism

Out of all the possible payload launchers, we chose not to deal with any mechanism that "flings" the payload into the air. We did not want to worry about the unreliability and inconsistencies that come with launching a payload and subjecting it to air resistance or other external forces, hence, we decided to increase the reliability of our system by decreasing the contact with its external environment. We settled with a "Turn and Pour" extendable arm mechanism, which will let the payload roll off at a slow speed, directly into the bucket, to decrease any chances of external interference.

Issue 1: The turning arm would have to be sufficiently long for it to reach above the bucket. However, the weight of a long arm greatly affects the robot's centre of gravity, making movements less accurate and reliable. Additionally, it would hinder the turtlebot's ability to move through tight corners of the maze as easily.

Solution: We aim to separate each turning arm into two separate beams which are connected using a pin. This helps to ensure that the arm can be folded when not in use, allowing for a smaller overall footprint of the turtlebot.

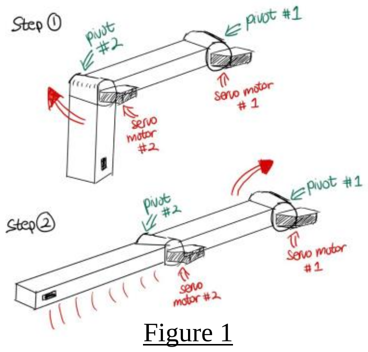


Figure 1

Issue 2: Having a foldable turning arm would mean that additional operations are required. The first operation is to unfold the arm such that it is fully extended, represented in Figure 1. The second operation requires turning the extended arm such that the payload's basket is above the top of the basket in order to tip the payload into the bucket. The use of one motor for each operation as shown in Figure 2 would be able to rectify this issue, but this would cause the overall weight of the arm to be heavier. Motor 1 in Figure 2 would have to generate more torque to lift both the payload and the arm mechanism.

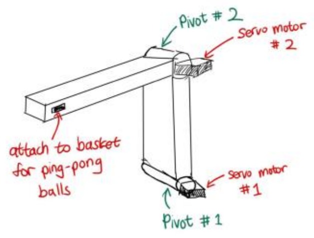


Figure 2

Solution: Reducing the number of servo motors from 2 to 1 through the use of a belt and gear mechanism. Referring back to Figure 2, motor 2 is replaced with a gear which is driven by motor 1 via a belt system. This helps in reducing the weight issue caused by mounting another servo motor onto the turning arm.

Issue 3: Having chosen the method of using an extendable arm mechanism, we would require a carrier to house and deliver the payload. Typically, most designs would utilise a carrier with a trapdoor that would release the payload after deployment. However, this causes the end of the launching mechanism to be top-heavy, thus increasing the height of the robot's centre of gravity and making it unstable. Additionally, the system would be over-complicated and unreliable due to many different moving parts needed, such as wiring and a servo motor.

Solution: Since having more moving parts increases the chance of a part failure, we have decided to design a simple carrier without any moving parts. The design of the carrier would aim to maximise payload release height while minimising material use. This would reduce overall weight and complexity to our proposed solution.

b. Electrical System

Several sensors, namely the LiDAR, 6-DOF IMU and Motor Encoders are included within the TurtleBot package. The primary functions of these sensors are for the robot to plot an accurate SLAM map of its environment and also provide precise data of its own location.

These sensors are favoured over other localisation methods (such as LPS) due to its ease of implementation, with no additional set up required beforehand. Methods such as markers will not produce precise data regarding the location of the robot, and is only able to specify preset points on the map where the robot is within range.

A camera is to be mounted to the front of the robot in order to detect colour. This will be used primarily to detect the red bucket once the robot enters the room, allowing the robot to deliver the payload into the bucket. As the bucket is the only red object in the room, a camera is preferred over the use of a LiDAR or depth camera to detect the bucket as it is less prone to errors or false detections. This colour detection capability will also allow the robot to identify coloured markers placed in the maze if necessary, to aid the robot in its navigation.

c. Navigation System

In order to produce a map of the maze, we will utilise the LiDAR on the robot to run LiDAR SLAM algorithms. With the limited time available to complete this project, the software work required to implement Visual SLAM algorithms is too much, and for the purposes of this mission a 2D LiDAR map would suffice.

The navigation system will consist of two parts - a traversing algorithm that allows the robot to explore the entire maze in order to produce an accurate and full map, and a path finding algorithm to get the robot to the exit after the maze has been mapped. This is to prioritise speed in obtaining the full maze map, in order to gain the highest possible amount of points. The program flow is depicted below in Figure 7. By implementing a camera for object detection, we are also able to switch and utilise waypoints if the full implementation of LiDAR SLAM navigation turns out to be unfeasible.

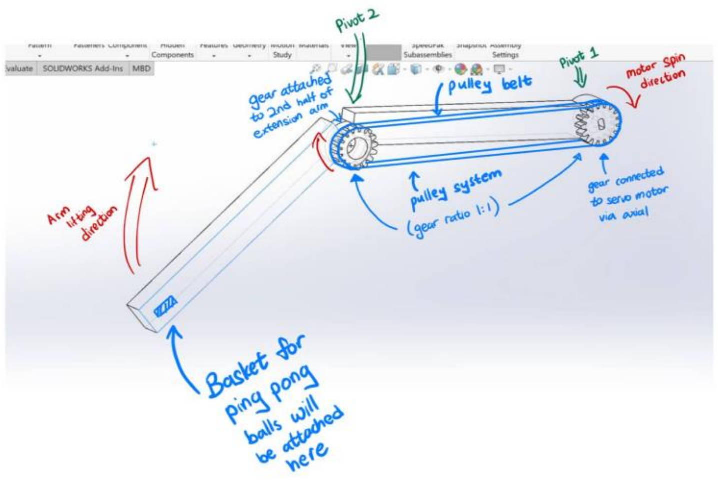
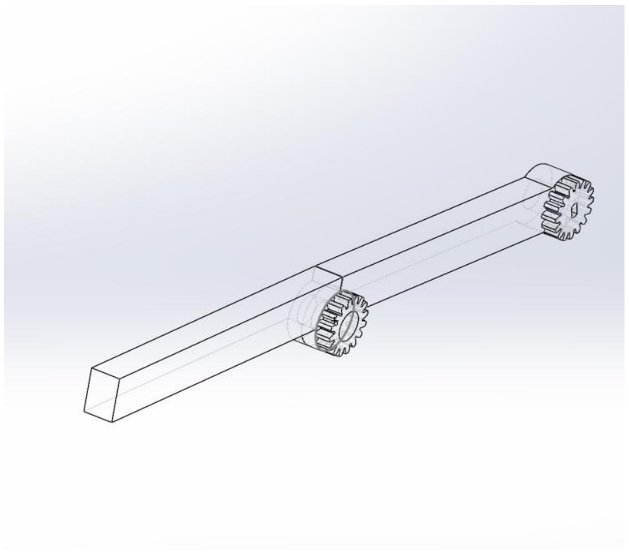
After passing through the maze, object detection would be run on the images obtained by the camera, thresholding the bucket primarily through its colour and size.

4. ***BOGAT***

Considerations	Mechanical	Electrical	Software
Mapping: LiDAR-SLAM	- LiDAR accuracy heavily impacted if moving object is in view	- Highly dependent on the functionality and accuracy of the LiDAR	- Requires a lot of computational power
Navigation: SLAM-based traversal, followed by path-finding to exit maze	- Traversing through the whole maze first before depositing the payload might result in unintended loss of the payload in the maze	- Navigation algorithms may require unnecessary movements from the robot resulting in greater power loss and battery drain	- Algorithms are highly reliant on accurate mapping - Maze solving algorithm has to be run simultaneously with other processes
Navigation: Camera Waypoint Detection	- If the camera is not mounted well and shifts while the robot is moving, this method would be more inaccurate	- Highly dependent on the functionality and accuracy of the camera	- Commands must be pre-coordinated for the behaviour of the robot upon detection of each waypoint
Localisation: LiDAR, 6-DOF IMU, Encoders	- Easily disrupted by unintended displacement or inaccurate starting positions of the robot	- Highly dependent on the functionality and accuracy of various sensors, prone to drift	- Intensive; requires virtual mapping of the surroundings in order to localise itself in its environment
Localisation: Camera Object Detection	- Might not maintain accurate distance to the bucket to drop the payload reliably	- Highly dependent on the functionality and accuracy of the camera	- Requires object detection/recognition software
Launching Mechanism: “Turn and Pour”	- Timing Belt might be worn out or dislodged after repeated use (Wear and Tear).	- Sufficient power needs to be supplied to the actuators to generate the required torque	- Need to ensure that the robot is at the correct distance and orientation with respect to the bucket

5. Preliminary Design

a. Launching Mechanism

 Pivot 2 Pivot 1 motor Spin direction gear attached to 2nd half of extension arm pulley belt Arm lifting direction pulley system (gear ratio 1:1) gear connected to servo motor via axial Basket for ping pong balls will be attached here	
Figure 3.1: Launching arm (Belt and gear system)	Figure 3.2: Launching arm (Fully extended)

The robot will utilise a launching arm mechanism in order to hold and transfer the payload into the red bucket. The arm will have two pivot points.

The first pivot will be a joint between the frame of the robot and the arm of the robot. This pivot will be fixed to a servo motor on each side, which will control the angle of elevation of the robot arm.

The second pivot connects the two halves of the launching arm together. Not only does this enable the launching arm to be straightened to achieve a longer launching arm but will allow for the arm to be folded into a smaller form factor to be mounted to the turtlebot.

This is crucial given the rather small base of the turtlebot. An extrusion of the robot arm from the base of the turtlebot could affect the balance, mobility and precise direction control of the turtlebot. In addition, the belt and gear system enables each arm to be extended using only one servo motor.

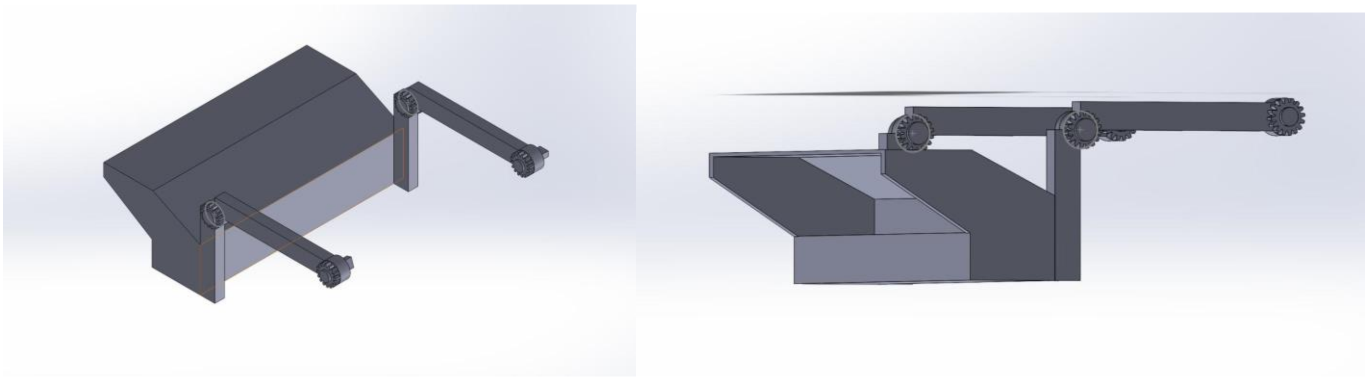


Figure 4 : Assembly model of launching arm with carrier

As shown in Figure 4, the launching arm will be folded at a 90 degree angle for its initial resting position.

Both the arms will be mounted on the 4th layer of the turtlebot where each arm is connected to a servo motor.

i. *Payload Carrier*

The payload carrier will house the five ping pong balls, with a design similar to a box with a semi-closed lid. The balls cannot roll out before and during the process of deploying the launching mechanism, and will only roll out of the carrier when the launching mechanism is fully deployed (upside down).

b. *Electrical System*

Multiple electrical components and subsystems will be connected to the Turtlebot via the Raspberry Pi or the STM32F7. The connections between these components are depicted below in Figure 11.

i. *LiDAR*

A LDS-02 Light Detection and Ranging (LiDAR) sensor was included in the Turtlebot 3 package, and is mounted on the top of the robot. This sensor is used to plot the environment accurately, allowing the robot to avoid obstacles, and with the DFS algorithm, run path planning on the resultant SLAM maps to determine routes to an intended destination. The mapping and navigation process is shown below in Figure 8.

ii. *6-DOF IMU*

A 6-DOF IMU is included on the OpenCR board. This sensor provides the robot with accurate heading information, which augments the SLAM algorithm while mapping out the LiDAR data.

iii. *Motor Encoders*

Encoders are included inside both drive motors. This sensor provides rotation data of each wheel, which gives the robot odometry data that can also augment the SLAM algorithm to provide a more accurate occupancy map.

iv. *Camera*

An OV5647 (Pi Camera v1) camera mounted onto the robot will provide visual information of its surroundings, such as colour. We are able to threshold the resultant image from the camera to obtain location information of the red bucket in the room. This will allow the robot to detect the bucket, as it is the only red object present in the room.

This camera will also be able to detect other colours, allowing us to utilise different coloured papers as markers throughout the maze as waypoints or checkpoints for navigation in the event that the original navigation system fails.

c. *Navigation System*

The navigation system will first utilise a Depth-First Search (DFS) algorithm to map out the full maze. The DFS algorithm is able to traverse and explore the maze while it is still unknown, by prioritising the open path that is in the preferred direction of travel. Hence, while the robot is mapping out the maze, it will use DFS to explore the furthest unvisited node in the maze, and keep repeating this process of exploring the furthest unvisited node until the map is over 95% complete.

After the map of the maze is complete, an A* path-finding algorithm will be run to determine the shortest path from the robot's current location to the exit of the maze. The full flowchart can be seen in Figure 10 below.

i. DFS vs BFS

DFS was chosen as our maze traversal algorithm due to its efficiency for our physical system. In solving a maze, DFS begins at the first intersection (i.e. node) and traverses nodes in one direction until a dead end or visited node is found. This is preferred over BFS, which visits all the nodes closest to its starting location before progressing to further nodes. As seen in Figure 9, while BFS might work in a virtual system or on code, the physical robot would have to move back and forth too much, which would take up too much time and be too inefficient.

Annex

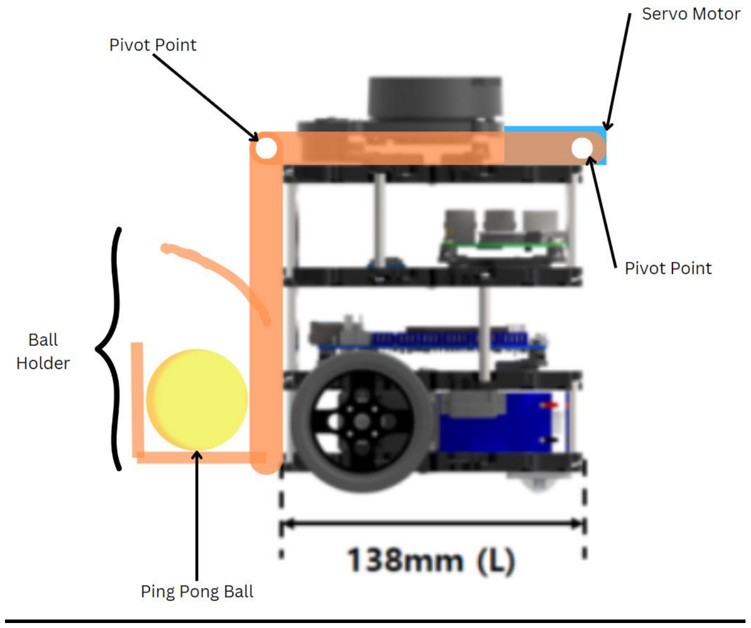


Figure 5: Labelled Drawing of the Launching Mechanism (in orange)

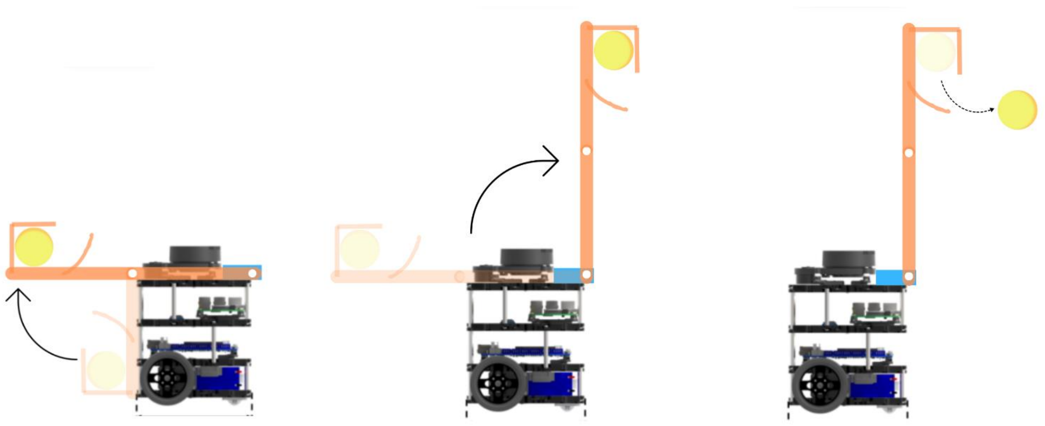


Figure 6: Proposed Movement of the Launching Mechanism

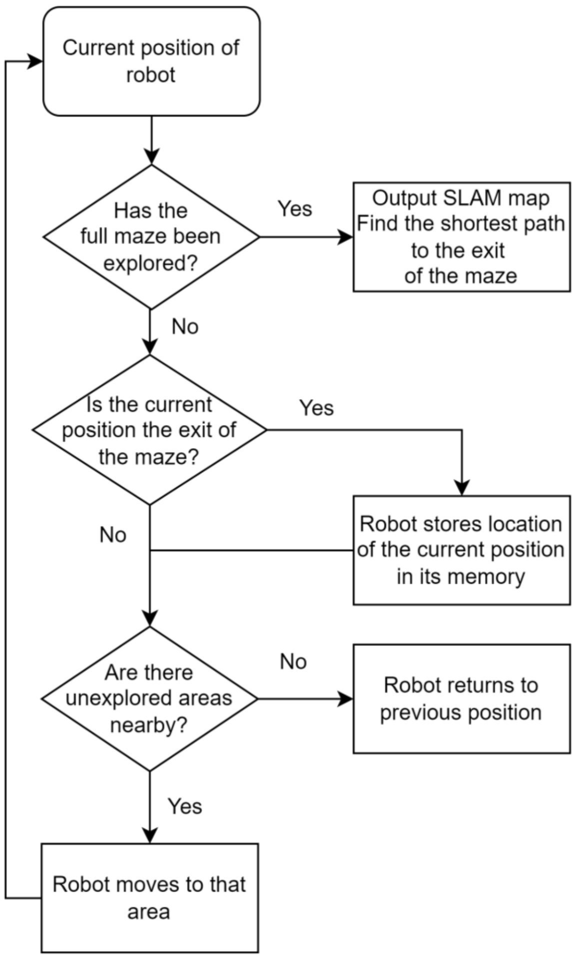

Current position of robot
Has the full maze been explored?
Yes
Output SLAM map Find the shortest path to the exit of the maze
No
Is the current position the exit of the maze?
Yes
Robot stores location of the current position in its memory
No
Are there unexplored areas nearby?
No
Robot returns to previous position
Yes
Robot moves to that area

Figure 7: Basic maze solver algorithm

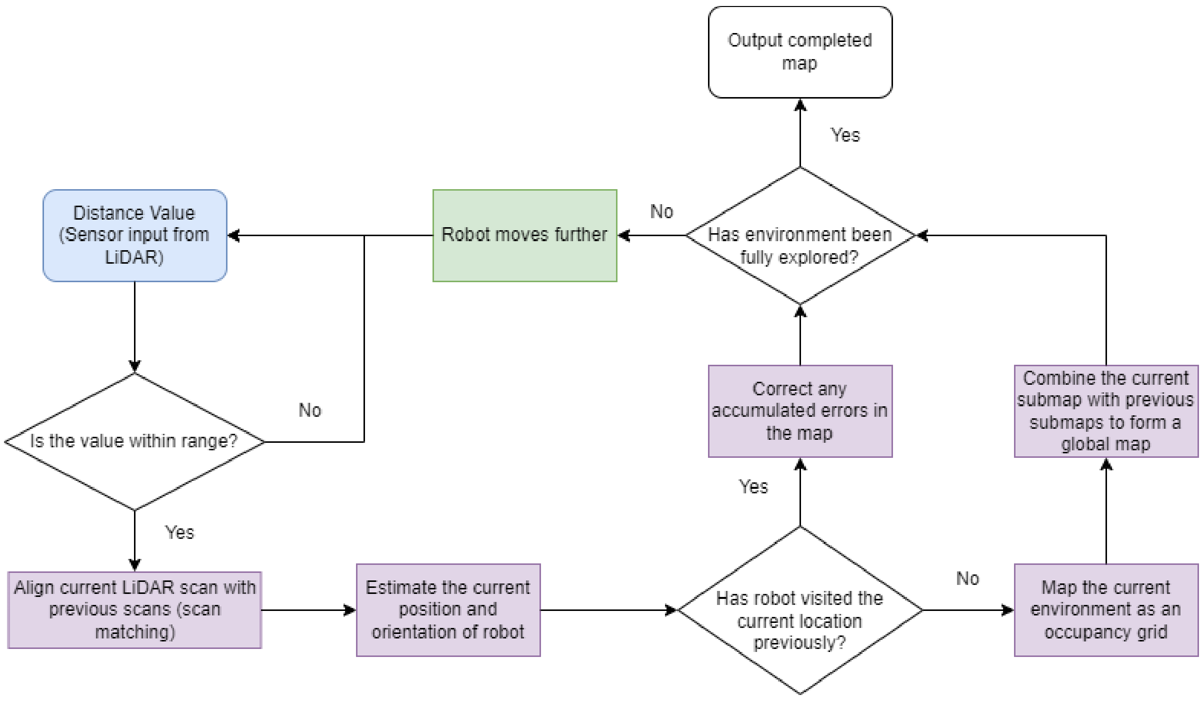

Output completed map
Yes
Distance Value (Sensor input from LiDAR)
Robot moves further
No
Has environment been fully explored?
Is the value within range?
No
Correct any accumulated errors in the map
Combine the current submap with previous submaps to form a global map
Yes
Yes
Align current LiDAR scan with previous scans (scan matching)
Estimate the current position and orientation of robot
Has robot visited the current location previously?
No
Map the current environment as an occupancy grid

Figure 8: Mapping and navigation process

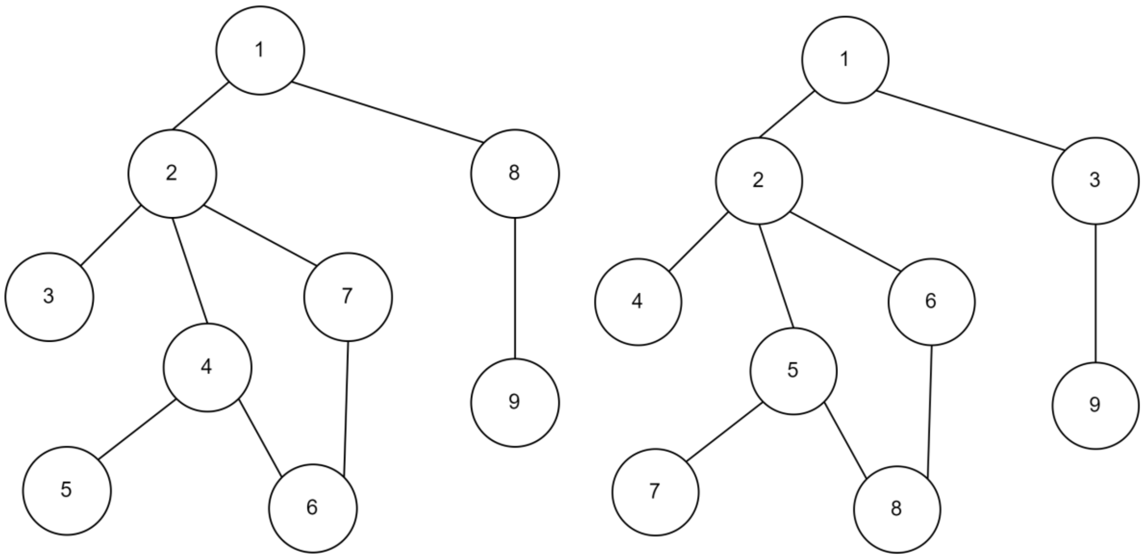


Figure 9: Visual representation of DFS and BFS

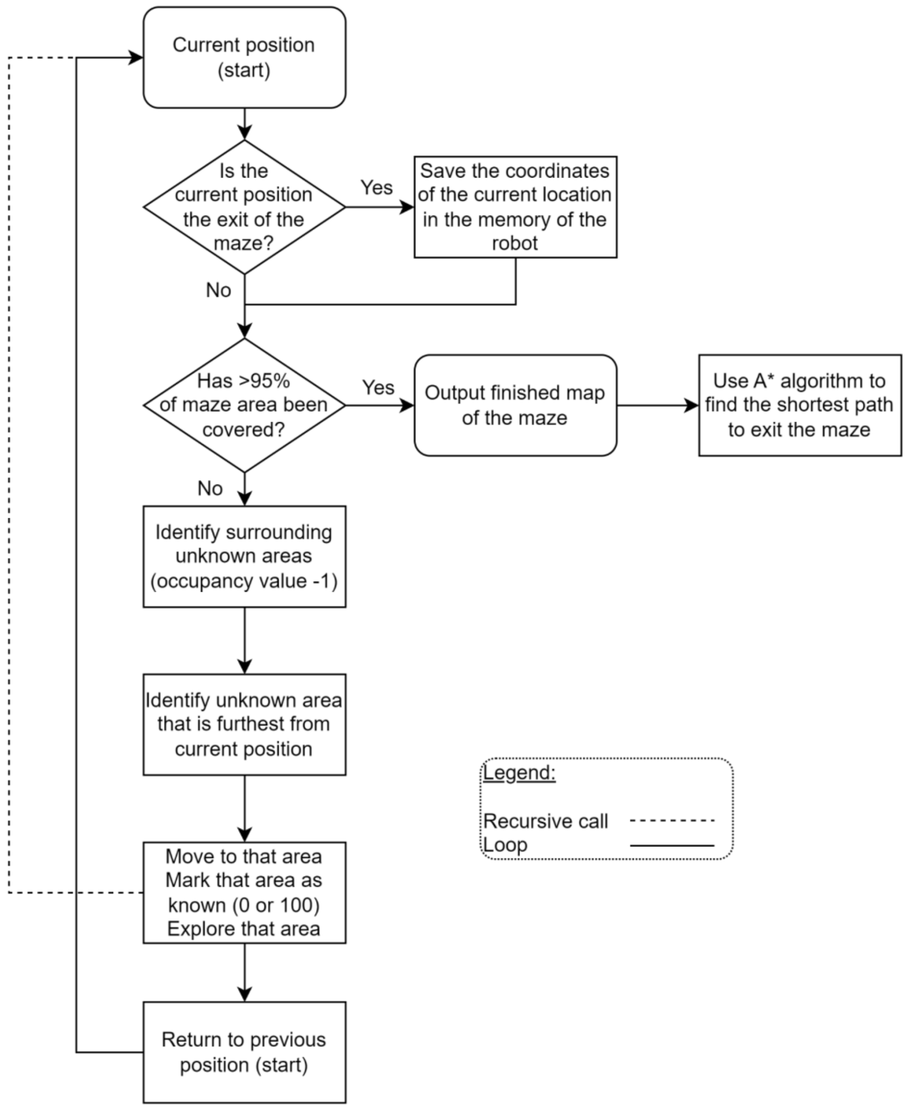

Current position (start)
Is the current position the exit of the maze?
Yes
Save the coordinates of the current location in the memory of the robot
No
Has >95% of maze area been covered?
Yes
Output finished map of the maze
Use A* algorithm to find the shortest path to exit the maze
No
Identify surrounding unknown areas (occupancy value -1)
Identify unknown area that is furthest from current position
Move to that area Mark that area as known (0 or 100) Explore that area
Return to previous position (start)
Legend:
Recursive call
Loop

Figure 10: DFS maze solving algorithm to be implemented

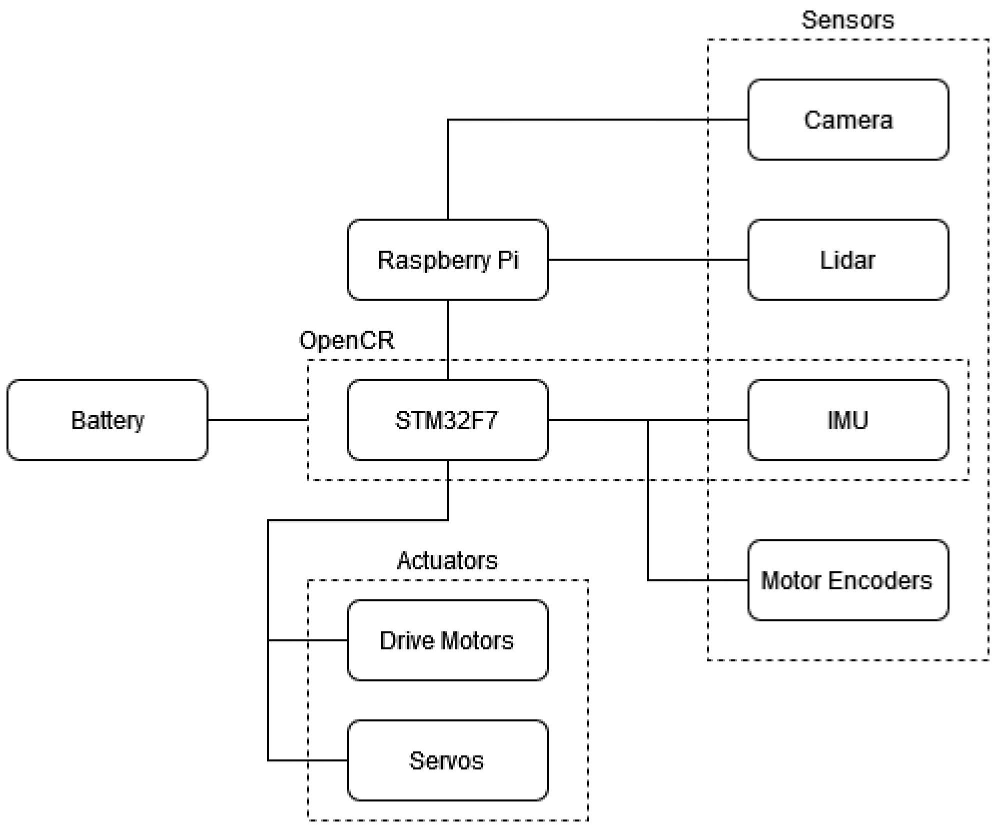


Figure 11: Block Diagram of Turtlebot 3

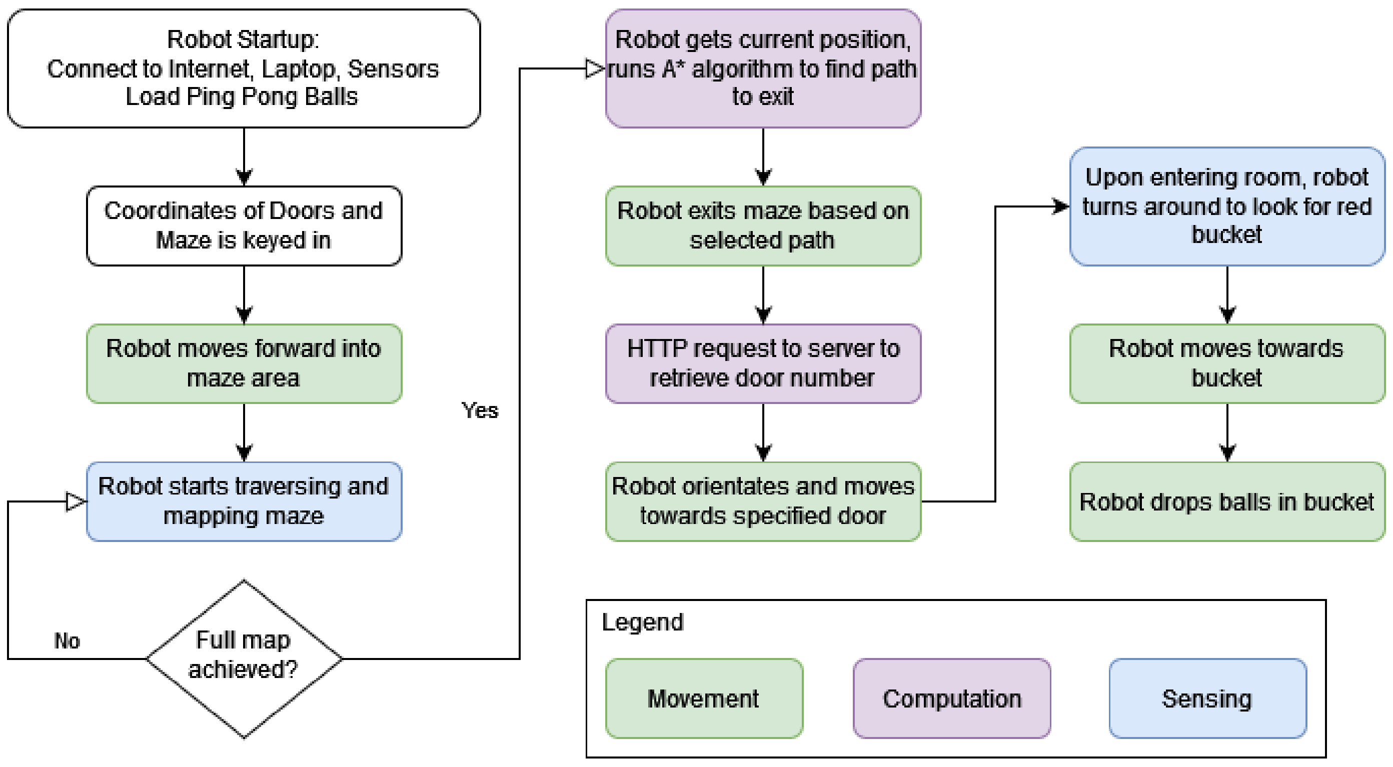

Robot Startup:
Connect to Internet, Laptop, Sensors
Load Ping Pong Balls
Coordinates of Doors and Maze is keyed in
Robot moves forward into maze area
Robot starts traversing and mapping maze
Full map achieved?
No
Yes
Robot gets current position, runs A* algorithm to find path to exit
Robot exits maze based on selected path
HTTP request to server to retrieve door number
Robot orientates and moves towards specified door
Upon entering room, robot turns around to look for red bucket
Robot moves towards bucket
Robot drops balls in bucket
Legend
Movement
Computation
Sensing

Figure 12: General flow of project