METR4202 Team Project Report: Turtlebot3 Exploration and Target Search

Team number: 21

Team members: Yiming Ma, Wenjie Ma, Shuyu Liu, Jayden Carter

Student number: 47283330, 47859708, 48017402, 47228340

Date Due: 2025/10/27

Table Of Contents

1. Introduction ..... Error! Bookmark not defined.
   1.1 Project Goal ..... Error! Bookmark not defined.
   1.2 System Setup ..... Error! Bookmark not defined.
   1.3 System Diagram ..... Error! Bookmark not defined.
2. Implementation of Developed Components ..... Error! Bookmark not defined.
   2.1 Autonomous Exploration Strategy ..... Error! Bookmark not defined.
      2.1.1 Detecting Unexplored Areas ..... Error! Bookmark not defined.
      2.1.2 Waypoint Computation ..... Error! Bookmark not defined.
   2.2 Target Detection and Localisation ..... Error! Bookmark not defined.
      2.2.1 ArUco Marker Identification ..... Error! Bookmark not defined.
      2.2.2 Marker Positioning ..... Error! Bookmark not defined.
      2.2.3 Duplicate Detection Processing ..... Error! Bookmark not defined.
      2.2.4 Marker Localisation (Coordinate Transformation) ..... Error! Bookmark not defined.
         2.2.4.1 Image to Camera frame ..... Error! Bookmark not defined.
         2.2.4.2 Camera to Robot Base frame ..... Error! Bookmark not defined.
         2.2.4.3 Robot Base to Map Frame ..... Error! Bookmark not defined.
3. Existing Libraries and Code ..... Error! Bookmark not defined.
   3.1 Waypoint commander ..... Error! Bookmark not defined.
      3.1.1 Launch Files and Mapping ..... Error! Bookmark not defined.
      3.1.2 Waypoint Exploration ..... Error! Bookmark not defined.
   3.2 AruCo detection ..... Error! Bookmark not defined.
4. Testing ..... Error! Bookmark not defined.
   4.1 Testing objectives ..... Error! Bookmark not defined.
   4.2 Map Design ..... Error! Bookmark not defined.
      4.2.1 Map A: ..... Error! Bookmark not defined.
      4.2.2 Map B: ..... Error! Bookmark not defined.
      4.2.3 Map C ..... Error! Bookmark not defined.
   4.3 Testing Procedure ..... Error! Bookmark not defined.
5. Conclusion ..... Error! Bookmark not defined.

Abstract

This project aims to develop a complete autonomous exploration and target localization system for the TurtleBot3 robot. This system is based on the Robot Operating System (ROS 2) framework and developed and tested in the Gazebo simulation environment. The system achieves real-time simultaneous localization and mapping (SLAM) by integrating the slam_toolbox and employing a custom frontier-based exploration algorithm. This algorithm continuously identifies the boundaries between known and unknown areas as potential exploration targets, intelligently guiding the robot to navigate into unknown areas to achieve complete environmental coverage. Simultaneously, the system utilizes an onboard camera and the OpenCV vision library to detect ArUco visual markers in the environment in real time. When the robot's camera scans an ArUco Marker, the system outputs the target's precise coordinates in real time and visually marks them on the map. Ultimately, this system successfully enabled TurtleBot3 to autonomously complete map construction, path planning, and locate all ArUco Marker targets in an unknown environment.

1. Introduction

1.1 Project Goal

The goal of this project was to develop an autonomous exploration strategy for a Turtlebot3 robot and localise targets in both virtual and real-world environments. This was done by using the ROS 2 framework and software simulation platforms Gazebo, Slam and Rviz 2. The project was divided into smaller tasks focusing on exploration and target localisation. To achieve autonomous exploration, algorithms were designed to detect open spaces and generate navigation waypoints that directed the robot to expand its explored area. Target localisation required the Turtlebot3 to detect ArUco markers through the onboard camera, computing coordinate transformations to map their positions accurately, and displaying them as visual markers on the map.

1.2 System Setup

The system consists of many different hardware components on the Turtlebot3 that will interact with each other based on the code and algorithms that were created. All core components, programs, and code required to make the system fully operational have been provided below:

- **Lidar Sensor:** The hardware component that gathers the data that is used in the exploration program
- **Camera:** The hardware component that is able to detect the ArUco marker and passes the information to a program to localise it onto the map
- **Motors and Odometry:** Hardware used to move around and travel, based off commands from Navigation and Slam

- **Waypoint Commander:** The python script that enables the robot to automatically identify unknown areas using frontiers and generate waypoints for further autonomous exploration.
- **Navigation and SLAM:** Utilises nav2 for path planning and slam_toolbox for real-time map generation.
- **ArUco Detector:** Python scripts that detect the markers from the camera feed, transforms their coordinates, and records each marker's position.
- **RViz2 Visualisation:** Provides real-time visualisation of the robot's exploration progress, displaying the evolving map, navigation paths, and detected markers that can be visualised on the map. This enables clear observation of frontier coverage, target identification, and overall exploration behavior.
- **Gazebo Visualisation:** Provides a platform and real time visualisation of the robot's exploration progress, which displays the map with the turtlebot in it from a 3D point of view

1.3 System Diagram

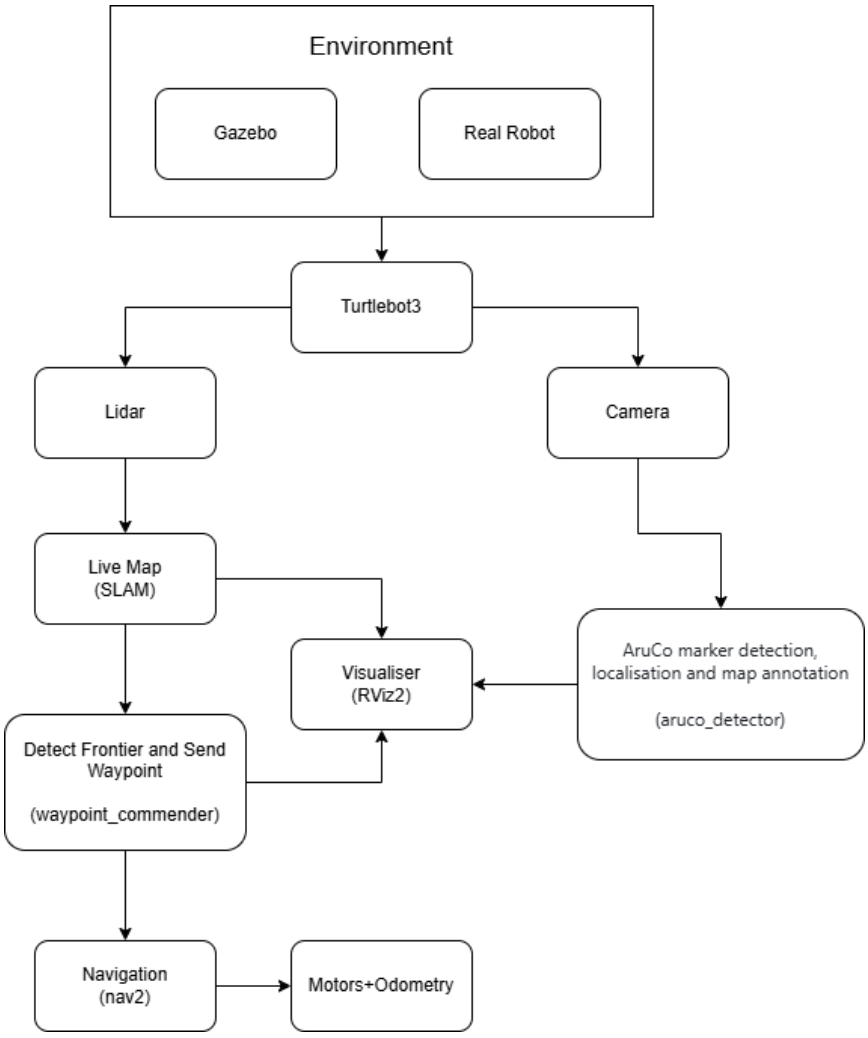

Diagram 1: A system flow chart that shows how all the components interact with the required programs

2. Implementation of Developed Components

2.1 Autonomous Exploration Strategy

2.1.1 Detecting Unexplored Areas

To detect unexplored areas, the code uses a frontier based algorithm approach. A frontier is the next set of possible paths or locations that will be the most effective for the robot to explore next.

When the robot receives a new global cost map (once *ros2 run waypoint_commander waypoint_cycler* is executed in the terminal), the function *detect_frontiers()* is called which executes a step by step process to identify unexplored regions.

The process begins by first iterating through all the cells in the costmap data. The value of each cell represents a degree of occupancy: -1 means it is an unknown area, 0-70 is specifically considered a free cell, and 70-100 is recognised as high cost/ an obstacle. The code then checks each free cell's 4 neighbouring cells (up, down, left, right); If the neighbouring cell has a value of -1, it is classified as frontier. These cells are then added to a list of frontier cells that represent edges of unexplored areas the robot should go to next. To avoid using previously visited frontier cells, the node removes any frontiers that are too close to already visited waypoints.

2.1.2 Waypoint Computation

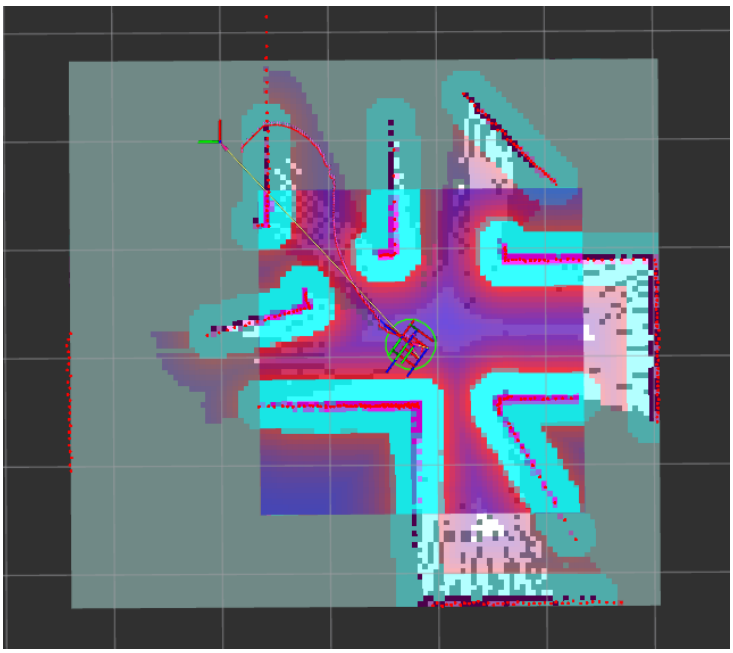

Figure 1: Simulation of Tutlebot3 in an unexplored map

Once all the frontiers are identified, the code selects the most suitable frontier point to set as the waypoint, which is done in two stages. For the first waypoint, the robot moves to the farthest

frontier from its current position to try and cover as much space as possible. Then for subsequent waypoints, the algorithm takes the closest 30% of frontiers and clusters them by their 4 neighbouring frontier cells. From this the algorithm selects the largest cluster - as it will likely represent a large unexplored space and sets the centre of the cluster as the waypoint. The waypoint location is then converted into a pose-stamped message in the map frame. The robot then navigates to that point via Nav2, whilst the node monitors progress.

```
will@Will:~$ ros2 run waypoint_commander_team21 waypoint_cycler_team21
[INFO] [1761522004.853183024] [frontier_detector]: New waypoint sent → (-0.11, -0.25)
[WARN] [1761522035.718211868] [frontier_detector]: Stuck for > 30s — trying to escape…
[INFO] [1761522035.776224875] [frontier_detector]: Escape: open area (≥ 0.30 m clearance) → (-1.06, 0.23)
[INFO] [1761522035.777088698] [frontier_detector]: New waypoint sent → (-1.06, 0.23)
[INFO] [1761522045.555185711] [frontier_detector]: Reached goal → ready for the next frontier
[INFO] [1761522045.557944231] [frontier_detector]: Reached goal → ready for the next frontier
[INFO] [1761522046.975348252] [frontier_detector]: New waypoint sent → (-0.26, -1.82)
[WARN] [1761522077.717667553] [frontier_detector]: Stuck for > 30s — trying to escape…
[INFO] [1761522077.808450910] [frontier_detector]: Escape: open area (≥ 0.30 m clearance) → (-1.71, -1.42)
[INFO] [1761522077.812708438] [frontier_detector]: New waypoint sent → (-1.71, -1.42)
```

Figure 2: Terminal output of the Waypoint Commander node

To avoid the robot going back to the same waypoints, it checks each new frontier with the other waypoints it has explored already. If the robot hasn't reached its next waypoint in 30 seconds, it then builds a mask of free enough cells that are less than 30. This then finds connected open areas and picks a cell in it with enough clearance as a temporary escape waypoint. This means the robot won't get trapped in narrow corners or areas.

2.2 Target Detection and Localisation

The TurtleBot3 uses a simulated or onboard camera to detect ArUco markers in the environment, allowing for automatic target identification and localisation. The module subscribes to the robot's camera image topic and processes each frame using a predefined ArUco marker dictionary. When a marker is detected, the node computes its position and orientation relative to the camera and transforms these coordinates into the global map frame. This process enables the robot to accurately record the positions of target objects within its environment.

2.2.1 ArUco Marker Identification

The module captures raw image data from the robot's camera processes it through the ArUco detection algorithm. The algorithm uses edge detection and geometric filtering to locate possible square marker regions and further analyses the internal binary pattern to verify and identify valid ArUco markers and their IDs. For each detected marker, the system draws a bounding box on the original image to indicate the recognition result and extracts the coordinates of the four corner points of the marker's pixels and the marker ID.

2.2.2 Marker Positioning

Next, by utilising the camera's internal parameters and the detected actual side length of the marker, the algorithm uses a perspective-n-point solver to calculate the marker's position relative to the camera coordinate system. OpenCV's ArUco library provides built-in functions for directly calculating the 3D pose of each marker based on the detected corner pixel coordinates, real-world dimensions, and camera calibration parameters. The result is a set of translation and rotation vectors that describe the marker's pose within the camera coordinate system.

2.2.3 Duplicate Detection Processing

To improve detection stability and system efficiency, the module prevents repeated processing of markers with the same ID. When the robot re-detects an ArUco marker it has already identified, the system recognises it as an existing target rather than a new one. This approach eliminates redundant computations caused by revisiting the same marker and minimizes the effects of duplicate data. As a result, each physical marker is consistently mapped to a single, unique target entity, ensuring reliable recognition and a stable, jitter-free mapping process.

2.2.4 Marker Localisation (Coordinate Transformation)

After detecting a marker and estimating its pose relative to the camera, the data is transformed into the global map frame through three coordinate transformations:

```
will@Will:~$ ros2 run aruco_detector_team21 aruco_detector_team21
[INFO] [1761522001.012488251] [aruco_map_marker]: ArUco map marker node has started.
[INFO] [1761522035.623634830] [aruco_map_marker]: --> New marker found! ID: 1, Map Coordinates: x=0.32, y=0.54
[INFO] [1761522035.639479991] [aruco_map_marker]:    Published green marker on the map for ID: 1.
[INFO] [1761522049.045108954] [aruco_map_marker]: --> New marker found! ID: 2, Map Coordinates: x=-1.26, y=-2.
16
[INFO] [1761522049.049226175] [aruco_map_marker]:    Published green marker on the map for ID: 2.
[INFO] [1761522095.281350376] [aruco_map_marker]: --> New marker found! ID: 0, Map Coordinates: x=-2.86, y=0.2
4
[INFO] [1761522095.284377358] [aruco_map_marker]:    Published green marker on the map for ID: 0.
```

Figure 3: Terminal output of the AruCo Detector node

2.2.4.1 Image to Camera frame

Using the camera's intrinsic parameters, pixel coordinates are converted into 3D camera coordinates. The pose estimation process provides the marker's position (tvec, in meters) and orientation (rvec) relative to the camera. In this frame, the x-axis points right, y-axis down, and z-axis forward.

2.2.4.2 Camera to Robot Base frame

The known static relationship between the camera and the robot base allows conversion of the marker's pose from the camera frame to the robot's base frame. This transformation is handled by ROS 2's TF system, which stores the static camera to base_link transform in the TF tree. The system can then query and compute the marker's pose in the base frame automatically.

2.2.4.3 Robot Base to Map Frame

Using the robot's pose from the SLAM module, the marker's pose is transformed into the global map frame. ROS continuously publishes the dynamic map to base_link transform. By combining

the map-to-base_link and base_link-to-marker transforms, the system computes the map-to-marker transform, providing the marker's precise 3D coordinates in the global map frame.

2.2.5 Marker Visualisation and Map Annotation：

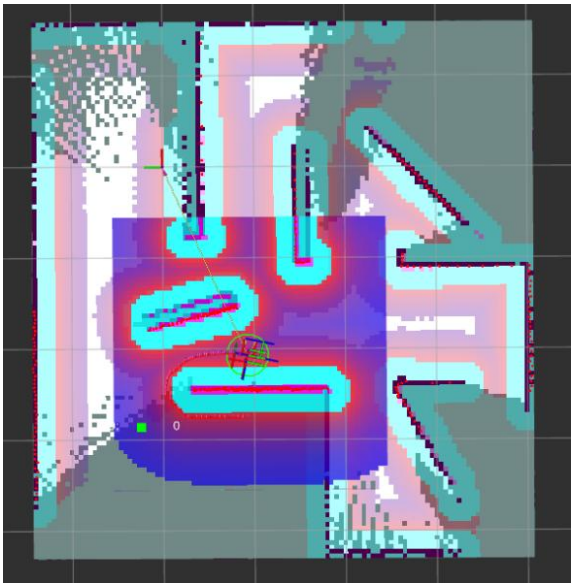

Figure 4: Simulation of TurtleBot3 detecting an ArUco marker. The green dot represents the estimated position of the detected marker on the map.

Once ArUco marker positions are determined, they are visualised in Rviz 2 using the *visualization_msgs/Marker* message. The marker's reference frame (*frame_id*) is set to the global "map" frame, ensuring it remains fixed at its true location regardless of robot motion.

Each detected marker is represented by a green dot positioned according to its pose in the map frame. The dot's size is adjusted for clear visibility, and unique namespaces and IDs ensure consistency between updates without duplicate objects. The marker's lifetime is set to permanent, so it remains visible throughout the session.

This visualization provides an intuitive, real-time overview of identified markers within the environment, effectively annotating the SLAM map and enhancing situational awareness in RViz.

3. Existing Libraries and Code

3.1 Waypoint commander

The project is built upon the existing ROS 2 and TurtleBot3 packages, with modifications made where necessary to achieve the specific requirements of the exploration and target localization system.

3.1.1 Launch Files and Mapping

The launch file used to open and initialize the map was adapted from the official *TurtleBot3* launch file turtlebot3_world.launch.py. The file path and the robot's initial pose were modified to match the test environment, and the new launch file was named testmap.launch.py.

3.1.2 Waypoint Exploration

The waypoint generation and exploration strategy were implemented in a custom Python node named waypoint_cycler_team21.py.

This script was primarily developed by the team, referring to the example code provided in Practical 4 (prac4) as a starting point. Most of the logic, including frontier detection, waypoint selection, and navigation sequencing, was independently developed and refined to support autonomous exploration.

3.2 AruCo detection

The ArUco detection system combines existing ROS 2 and OpenCV libraries with custom-developed logic. The OpenCV ArUco module handles marker identification, corner extraction,

and pose estimation using the camera's intrinsic parameters. A custom Python script was written to integrate these functionalities

4. Testing

4.1 Testing objectives

Simulation and testing in Gazebo are essential preliminary steps before deploying algorithms to the real robot. Testing in simulation allows developers to validate algorithms and system integration without risking hardware damage or safety concerns. Based on the system configuration described in 1.2, the following testing objectives were established:

- **Waypoint and Navigation Validation:** Verify that waypoints are correctly generated from the identified frontiers and that the robot navigates to each target point without collision, prioritising unexplored regions to maintain high search efficiency.
- **ArUco Detection Evaluation:** Assess the accuracy of ArUco marker detection and ensure that detected markers are correctly represented and positioned in RViz2.
- **Mapping Performance Assessment:** Evaluate overall map coverage, consistency, and system robustness during different exploration maps.

4.2 Map Design

Three simulation maps were created for evaluation:

4.2.1 Map A:

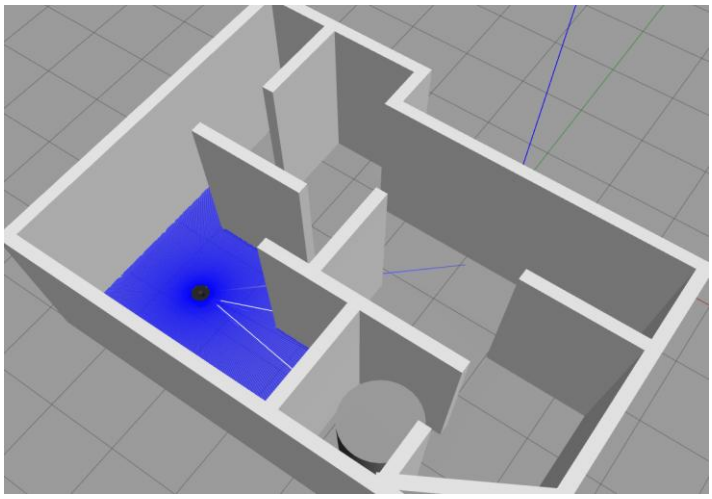

Figure 5: Gazebo-generated simulation environment (Map A)

This map combines both narrow corridors and open-space environments to evaluate the robot's performance under varying levels of environmental complexity. The design tests the robot's ability to navigate and generate frontiers in both complex and sparse areas, assessing its capability to maintain accurate localisation and waypoint generation when varying amounts of environmental information are available for SLAM matching. The map was constructed using the Gazebo Building Editor and is primarily used to assess the first testing objective (waypoint and navigation validation).

4.2.2 Map B:

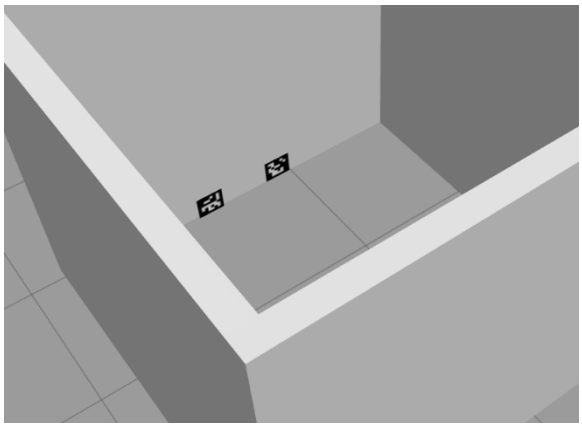

Figure 6: Gazebo-generated simulation environment (Map B)

The second map is enclosed by square walls, with the TurtleBot3 spawned directly facing two ArUco markers. This setup tests the marker detection and localisation system, verifying that markers can be detected and are correctly visualised in RViz2 with green indicators. The ArUco

targets were generated using a Python script that automated model creation by duplicating a base "marker0" template, replacing texture images, and updating configuration files. Each processed image produced a unique compatible 3D model placed in the Gazebo simulation world.

4.2.3 Map C

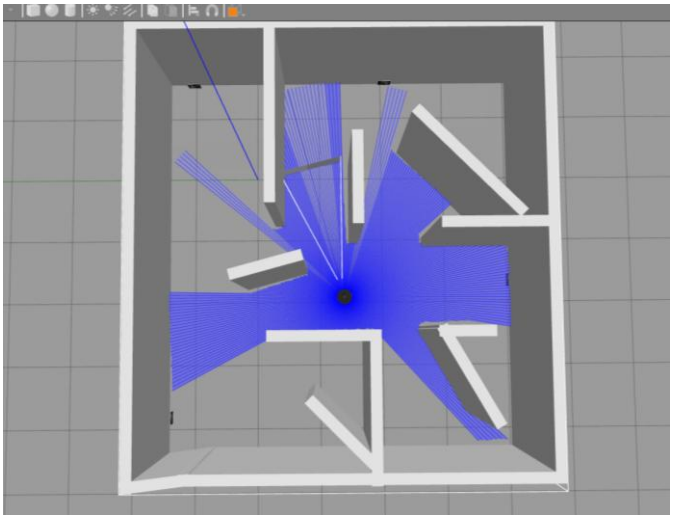

Figure 7: Gazebo-generated simulation environment (Map C)

Map C is a more complex map that combines the features of the previous maps to evaluate the integrated performance of the system under a realistic and challenging environment. It assesses how effectively the robot can maintain localisation accuracy while exploring, detect and register ArUco markers during movement, and continue to generate valid frontiers for exploration. By integrating all system components within a single environment, it evaluates the robot's overall capability to achieve the project goal of autonomous exploration, mapping, and target detection.

4.3 Testing Procedure

Testing was performed systematically in the Gazebo simulation environment to verify both the exploration and target localisation subsystems. For each map, the robot was spawned at a fixed initial pose, and the corresponding launch file was executed to initiate SLAM, navigation,

waypoint generation, and ArUco detection. The exploration process was allowed to run until the robot completed full map coverage or reached all accessible frontiers. During testing, the system's performance was monitored through RViz2, observing navigation accuracy, mapping progress, and visualisation of detected ArUco markers. Marker detection results were cross-checked against their known positions in Gazebo to confirm localisation accuracy. All tests followed consistent parameters and README instructions to ensure reliable comparison of system performance across different environments.

5. Conclusion

This project successfully developed and demonstrated a fully autonomous exploration and target localisation system for the TurtleBot3 using the ROS 2 framework. Through the integration of the slam_toolbox, Nav2, and custom Python nodes for frontier-based exploration and ArUco detection, the robot was able to perform complete mapping, navigation, and target recognition tasks within simulated and real-world environments.

The frontier-based exploration algorithm enabled the TurtleBot3 to dynamically identify unexplored regions and navigate efficiently across various maps, ensuring complete environmental coverage without redundant paths. Simultaneously, the ArUco detection module achieved accurate real-time recognition and localisation of visual targets, transforming their positions into global map coordinates and visualising them clearly in RViz2.

Testing across multiple simulated maps confirmed that the system performed robustly under diverse conditions. The robot successfully balanced exploration efficiency, mapping accuracy,

and target detection reliability. The results validated that the integration of SLAM, path planning, and computer vision could yield a practical autonomous mapping and localisation framework.