RMIT UNIVERSITY

PROGRAMMING AUTONOMOUS ROBOTS (COSC2781)

APPROXIMATE OBJECT LOCATION AND TRACKING WITH UAVS QUAD-COPTER

Assignment 3 Progress Update

Group Name: 03-XAI

Submitted by

ANDRIA NICHOLAS	**S3995645**
MARYAM MUSALLAM AL-HOWAITI	**S3856144**
AMAY VISWANATHAN IYER	**S3970066**
BASAVARAJ SOMASHEKHAR SANSHI	**S3975993**

Submitted on:

31-05-2024

Submitted to

Dr Timothy Wiley

Introduction

Object location and tracking with autonomous robotic systems is a fundamental challenge with important applications in fields such as conservation, surveillance, infrastructure inspection, search and rescue, and more. This project aims to develop algorithms and methodologies to enable a ground-based ROSbot 2 Pro robot to approximate the capabilities needed for the object location and tracking tasks outlined in the IMAV 2024 UAV competition's indoor and outdoor challenges. By adapting these aerial robotics tasks to a ground robot platform, this work seeks to advance the state-of-the-art in robust object detection, tracking, mapping, and interaction in unstructured, dynamic environments.

Autonomous object tracking from mobile robots has been an active area of research. Approaches have included using computer vision and deep learning for 2D and 3D object detection (Zhao, 2019) (Guo, 2021), multi-sensor fusion of visual and spatial information (Sünderhauf, 2015), and active perception to handle occlusion (Atanasov, 2014). Ground robots face unique challenges compared to UAVs, such as obstacles at varying heights, which this project will need to address.

Problem Definition

The core objective is to develop a system enabling the ROSbot 2 Pro to autonomously navigate in an unknown environment, locate specific fiducial markers, and perform interactive behaviors triggered by each marker. Key challenges include:

a. Search: The robot must explore the environment to find ArUco markers within a constrained time window. Efficient coverage of the search space is critical (Cao, 2021).
b. Mapping: The robot needs to construct a map of the environment and marker locations. SLAM techniques well-suited to ground robots in indoor and outdoor settings will be investigated (Kshirsagar, 2021).
c. Detection: ArUco markers must be reliably detected and identified under varying poses, distances and lighting conditions. Existing ArUco libraries will provide a foundation (Romero-Ramirez, 2018).
d. Interaction: The robot has to perform marker-specific actions like precision pointing, spinning, traversal, and pick-and-place. This requires tight integration of perception and control, drawing on related work in mobile manipulation (Xue & Sue, 2020).
e. Navigation: Collision-free navigation will rely on mapping, obstacle avoidance, path planning, and low-level control. Approaches demonstrating robustness in similar applications will be adapted (Shen, 2021).
f. Autonomy: The robot must make high-level decisions to sequence subtasks and manage mission constraints. Finite-state machines and behavior trees are common architectures for autonomous control (Colledanchise, 2018).

This project presents an opportunity to integrate and extend robotics techniques in the context of a focused challenge problem. By grounding the effort in the tasks and performance metrics defined by IMAV, progress can be rigorously evaluated. Success will be measured by the ability to locate and interact with the markers, the degree of autonomy achieved, and the robustness to real-world conditions. Unlike the IMAV 2024 competition focused on UAVs, this project leverages the unique capabilities of the ROSbot 2 Pro, such as obstacle navigation and object interaction using a pointer

device. (IMAV 2024 Competition Rules, 2024) By demonstrating the robot's performance in approximating the IMAV 2024 tasks, this work showcases the potential of ground-based robots in object location and tracking applications, paving the way for further research and development in this domain.

Related Works

The proposed project aims to adapt the object location and tracking tasks from the IMAV 2024 UAV competition (IMAV 2024 Competition Rules, 2024) to a ground-based ROSbot 2 Pro platform. The IMAV 2024 competition serves as an inspiration for the project objectives, focusing on search, navigation, and visual localization tasks in unstructured and dynamic environments. By translating these challenges to a wheeled robot, this project seeks to advance the development of robust perception, navigation, and interaction algorithms that can be generalised to various robotic systems (Bauer, 2008). Fiducial marker-based localization and navigation have been widely studied in the robotics literature. ArUco markers, which are used in this project, have become a standard tool for reliable camera pose estimation and visual servoing (Garrido-Jurado, 2014). Instead of using the OpenCV ArUco library, this project employs the Find Object 2D package for detecting ArUco markers. (Romero-Ramirez, 2018) Romero-Ramirez et. al proposed an efficient method for detecting ArUco markers, which has been integrated into popular libraries like OpenCV. The use of Find Object 2D demonstrates the flexibility of the proposed approach in leveraging various marker detection methods.

The requirement for the ROSbot 2 Pro to perform unique actions based on detected markers relates to the concept of behavioural autonomy in robotics, which was covered in class on week 10. Finite-state machines (FSMs) and behaviour trees (BTs) are commonly used to design flexible, reactive robot control systems (Collendanchise, 2018). The action-selection logic for the ROSbot 2 Pro is designed using similar principles, with the aim of creating a modular, extensible architecture. Our final code demonstrates the implementation of marker-specific behaviours, such as spinning 360 degrees, turning right, and moving in circles, which aligns with the project objectives and the material covered on robot control and locomotion back in week 4. Navigation in unstructured and dynamic environments remains a key challenge for the ROSbot 2 Pro. The robot needs to generate collision-free paths while accounting for its non-holonomic constraints and the presence of moving obstacles, such as other robots and people. Our week 5 course material on navigation covered various path planning and obstacle avoidance techniques. We implemented a Dynamic Window Approach (DWA) for navigation, which is a well-established method for real-time obstacle avoidance and path planning in dynamic environments (Fox, 1997). Our code also demonstrates the integration of the ROS Navigation stack (Nav2) for autonomous navigation tasks, which aligns with the project objectives and the course material.

Mapping and localization are essential capabilities for the ROSbot 2 Pro's autonomous navigation, as discussed in the course material from week 6. Given the unstructured nature of the competition environment, the ROSbot 2 Pro employs Simultaneous Localization and Mapping (SLAM) algorithms to construct a consistent map while estimating its pose. The choice of SLAM algorithm depends on the available sensor suite and computational resources of the ROSbot 2 Pro. The use of a pointer device attached to the ROSbot 2 Pro for precise marker interaction relates to the field of robot manipulation, as per course material from week 6. While the pointer device is simpler than a

traditional robotic arm, the task still requires accurate positioning and control. The project demonstrates the integration of a custom end-effector (pointer device) with the ROSbot 2 Pro, showcasing the application of manipulation concepts in a ground robot scenario.

Generating human-interpretable logs and visualisations of the ROSbot 2 Pro's actions is important for post-mission analysis and debugging. The week 7 and 8 material on robot vision discusses various techniques for processing and analysing sensor data, including image processing and feature extraction. The provided code demonstrates the logging of the robot's actions and the detection of ArUco markers, which can be used for generating mission reports and visualisations. In summary, the proposed project builds upon the concepts and techniques covered in the course material, including robot control, locomotion, navigation, localization, mapping, manipulation, and vision. By adapting established methods to the specific challenges inspired by the IMAV 2024 competition, this effort aims to advance the state-of-the-art in robot autonomy and demonstrate the generalizability of aerial robotics solutions to ground-based platforms like the ROSbot 2 Pro.

Significance and Relevance of Project Objectives

The proposed project, which adapts the object location and tracking tasks from the IMAV 2024 UAV competition (IMAV 2024 Competition Rules, 2024) to a ground-based ROSbot 2 Pro platform, holds significant relevance and potential benefits for the field of autonomous robotics.

Additional notes for indoor competition teams:

- Following the blue tape is optional since it only serves for navigation purposes.
- ArUco markers are positioned at the task locations, and all will have a size of 15x15 cm except for the ArUco markers on the raised landing platforms which will be 10x10 cm.

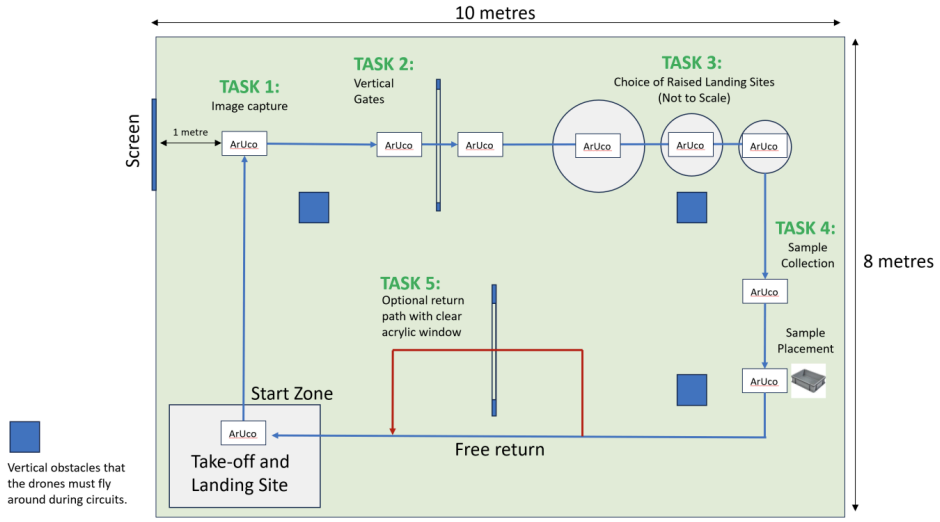


Figure 3: Indoor Competition Circuit

By translating aerial challenges to a wheeled robot, this project seeks to advance the development of robust perception, navigation, and interaction algorithms that can be generalized to various robotic systems (Bauer, 2008).

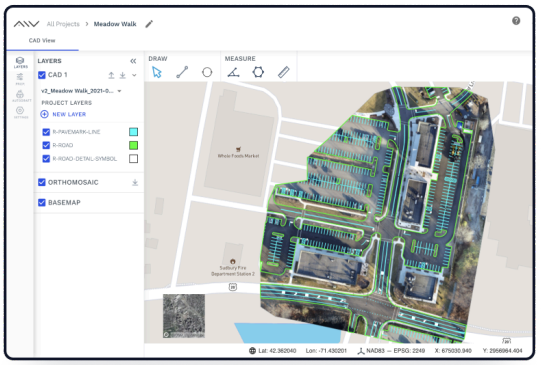
The IMAV 2024 competition serves as an inspiration for the project objectives, focusing on search, navigation, and visual localization tasks in unstructured and dynamic environments. These challenges are particularly relevant in the context of wildlife conservation and environmental monitoring.

Figure 1. Robovolc operating on Mt. Etna, Italy, Europe's largest active volcano. (Photo courtesy of University of Catania.)

Autonomous robots equipped with advanced sensing and navigation capabilities can play a crucial role in gathering valuable data on animal populations, habitat conditions, and potential threats (Wich, 2012). The use of ground robots like the ROSbot 2 Pro offers certain advantages over aerial platforms, such as longer operational times, higher payload capacities, and the ability to interact with the environment through manipulators or other actuators (Dunbabin, 2012). The image on the left is taken from Dunbabin, 2012 showing a ground robot surveying an active volcano for samples.

The significance of this project extends beyond the specific application to wildlife conservation. The algorithmic heuristics and techniques developed for the ROSbot 2 Pro can be readily adapted and deployed on drones and other autonomous platforms to support a wide range of environmental, social, and governance (ESG) practices. For instance, autonomous drones equipped with similar object detection and tracking capabilities can be used for precision agriculture, enabling farmers to monitor crop health, optimize resource allocation, and reduce the environmental impact of farming (Valente, 2013). In the realm of infrastructure monitoring, autonomous robots can efficiently survey and inspect critical assets such as bridges, pipelines, and power grids, detecting potential issues and facilitating timely maintenance (Metni, 2007).

Companies like AirWorks Solutions Inc. have already demonstrated the value of autonomous surveying techniques in the infrastructure sector. By leveraging AI-powered algorithms to analyze drone data, AirWorks enables faster and more accurate linework generation for construction projects, fiber installation permitting, impervious surface calculations, and more (Airworks Solutions, Inc. 2024). The application of similar techniques to greenfield and brownfield infrastructure surveying can significantly reduce costs, improve safety, and accelerate project timelines.



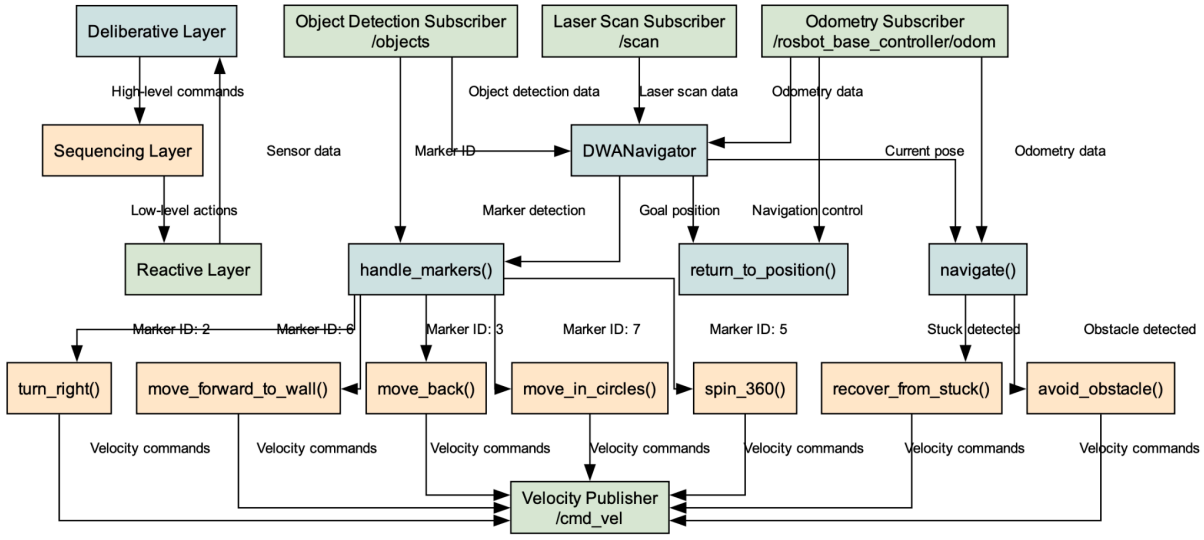
The open, outdoor environment targeted by the IMAV 2024 competition poses unique challenges for autonomous robots, such as dynamic obstacles, varying lighting conditions, and unpredictable terrain. By developing and testing algorithms in these challenging scenarios, the project contributes to the advancement of robust navigation and perception methods that can handle real-world uncertainties. The integration of fiducial marker detection using the Find Object 2D package, instead of relying solely on the OpenCV ArUco library, demonstrates the flexibility and adaptability of the proposed approach to different marker detection methods. Furthermore, the project incorporates concepts and techniques covered in the course material, including robot control, locomotion, navigation, localization, mapping, manipulation, and vision. By adapting established methods to the specific challenges inspired by the IMAV 2024 competition, this effort showcases the practical application of theoretical concepts and highlights the interdisciplinary nature of autonomous robotics research. Therefore, the proposed project holds significant relevance

and potential benefits for the field of autonomous robotics. By advancing the development of robust perception, navigation, and interaction algorithms in the context of wildlife conservation and environmental monitoring, the project contributes to the broader applicability of autonomous systems in supporting ESG practices and infrastructure surveying. The integration of diverse techniques and the focus on real-world challenges make this project a valuable contribution to the literature and a testament to the importance of autonomous robotics in addressing pressing societal and environmental issues.

Methodology, Implementation, and Software Architecture:

In this project, we developed a navigation system for a ROSbot using ROS 2 for autonomous navigation through a maze-like environment, avoiding obstacles, detecting Aruco markers, and performing specific actions based on detected markers. The project involved creating two main components: the NavigationPublisher node and the ArucoMarker node (Lee, 2022). The DWANavigator node controls the ROSbot's movement and employs a Dynamic-Window-Approach-based Navigation algorithm to find the best possible route in a dynamic environment, avoids obstacles, and performs actions based on detected Aruco markers. This was chosen after attempting A* and a thorough reading of Fox et. al 1997 (Fox, 1997) which explained DWA's specific advantage in fast-paced, dynamic obstacle infested environments.

Our DWANavigator class subscribes to the laser scan data (/scan) for distance measurements and the detected objects topic (/objects) for Aruco marker information. The node we initially implemented a few weeks ago employed a wall-following algorithm in the follow_wall() method, adjusting the ROSbot's velocities based on obstacle distances (The Bug2 Algorithm for Robot Motion Planning – Automatic Addison, n.d.). Overall, we have a three-tiered software architecture as per the following specifications that are outlined after a visualisation with each of the layers labelled:



a. Deliberative Layer: This layer is responsible for the high-level decision making, planning, and reasoning about the robot's goals and actions. In our code, the DWANavigator class represents the Deliberative layer. It subscribes to various topics, such as /odom for odometry data, /scan

for laser scan data, and /objects for detected object information (As per what we learned in Week 7). The Deliberative layer uses this sensor data to make informed decisions about the robot's navigation and behaviour.

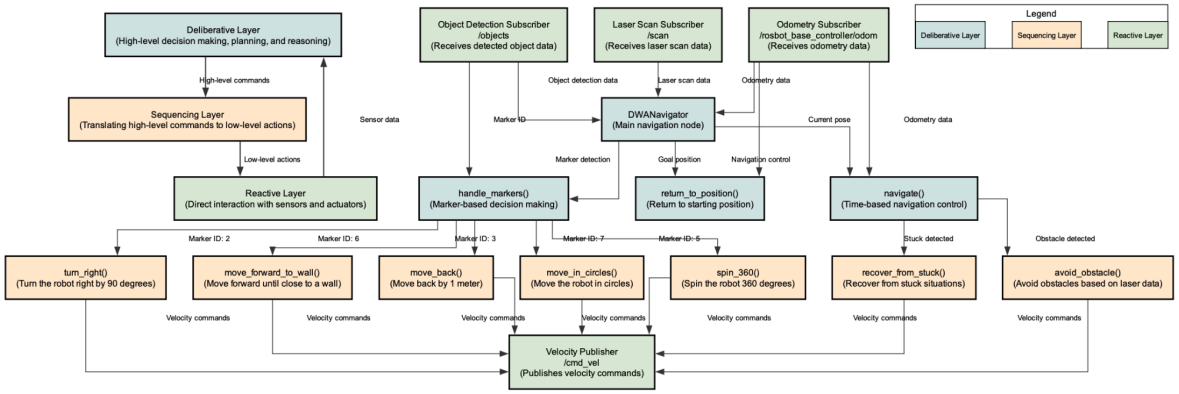
   i. The handle_markers() method in our DWANavigator class exemplifies the decision-making process in the Deliberative layer. When an Aruco marker is detected and its ID is received through the /objects topic, the handle_markers() method determines the appropriate action to take based on the marker ID. For example, if marker ID 5 is detected, the robot spins 360 degrees and pauses for 10 seconds. If marker ID 2 is detected, the robot turns right twice. These high-level decisions are made in the Deliberative layer based on the received marker information.
   ii. The Deliberative layer also includes methods like navigate() and return_to_position(), which plan the robot's navigation and determine the desired goal positions. The navigate() method checks the elapsed navigation time and decides when to stop navigating and return to the starting position. The return_to_position() method plans the path back to a specific position using the ROS2 Navigation stack.

b. Sequencing Layer: Our sequencing layer acts as an intermediary between the Deliberative and Reactive layers. It breaks down high-level commands from the Deliberative layer into a sequence of lower-level actions that can be executed by the Reactive layer. In our code, the Sequencing layer is represented by methods that translate the decisions made in the Deliberative layer into specific robot behaviours.
   i. For example, the spin_360(), turn_right(), move_forward_to_wall(), and move_back() methods in the DWANavigator class belong to the Sequencing layer. These methods receive high-level commands from the Deliberative layer and generate the necessary sequence of actions to accomplish the desired behaviour. The spin_360() method calculates the duration and angular speed required to complete a 360-degree turn and publishes the corresponding velocity commands. Similarly, the turn_right() method determines the duration and angular speed for a 90-degree right turn and publishes the velocity commands.
   ii. The Sequencing layer also includes methods like avoid_obstacle() and recover_from_stuck(), which generate appropriate sequences of actions to navigate around obstacles and recover from stuck situations. These methods continuously monitor the laser scan data and adjust the robot's motion based on the proximity of obstacles, ensuring safe navigation.

c. Reactive Layer: The Reactive layer is responsible for direct interaction with the robot's actuators and sensors. It receives commands from the Sequencing layer and translates them into low-level control signals for the robot's hardware. In our code, the Reactive layer is represented by the ROS2 publishers and subscribers that communicate with the robot's hardware or simulation environment.
   i. The /cmd_vel publisher in the DWANavigator falls under this Reactive layer. It publishes velocity commands to control the robot's motion based on the decisions made in the Deliberative and Sequencing layers. The Reactive layer ensures that the robot executes the desired actions by sending appropriate control signals to the actuators.
   ii. The /odom, /scan, and /objects subscribers in the DWANavigator class also fall under our Reactive layer. These subscribers receive sensor data from the robot's odometry,

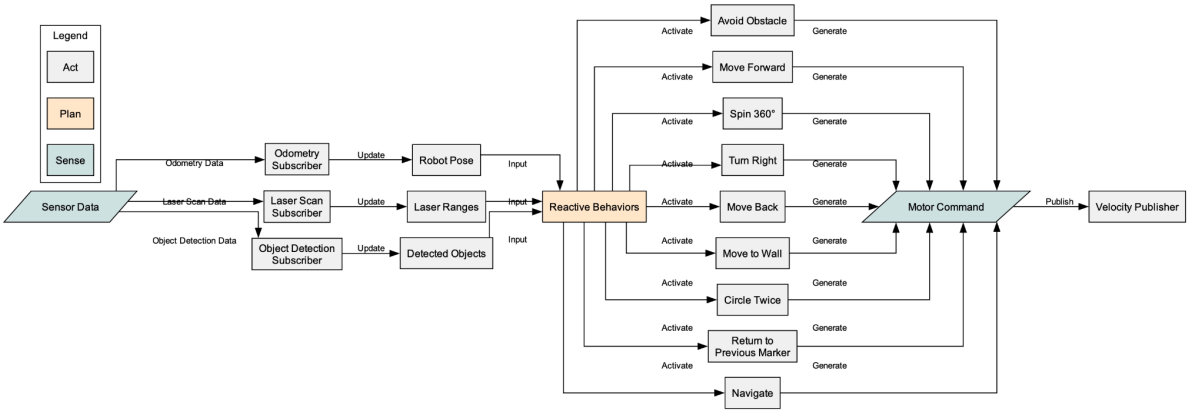
laser scanner, and object detection system, respectively. The Reactive layer processes this sensor data and provides it to the higher layers for decision making and planning.

iii. The Reactive layer operates in real-time, continuously receiving sensor data and executing control commands. It ensures that the robot responds quickly to changes in the environment and maintains stable and safe navigation.



The above diagram is a more detailed version of the prior flowchart and illustrates the Three-Tiered Architecture's deliberative, sequencing, and reactive layers in our code, crucial for autonomous navigation. The Deliberative layer receives sensor data from the Reactive layer via ROS2 subscribers, makes high-level decisions, and communicates navigation goals to the Sequencing layer. This middle layer translates complex behaviours into simpler, executable steps, ensuring proper order and timing. The Reactive layer then executes these commands by publishing velocity commands to the /cmd_vel topic, controlling the robot's motion. It also provides continuous sensor feedback to the higher layers, enabling informed decision-making and behaviour adjustments. This architecture allows for clear separation of concerns and modularity, with each layer focusing on specific responsibilities: high-level decision making (Deliberative), action sequencing (Sequencing), and hardware interaction (Reactive). This structure enhances the system's manageability, development, and maintenance.

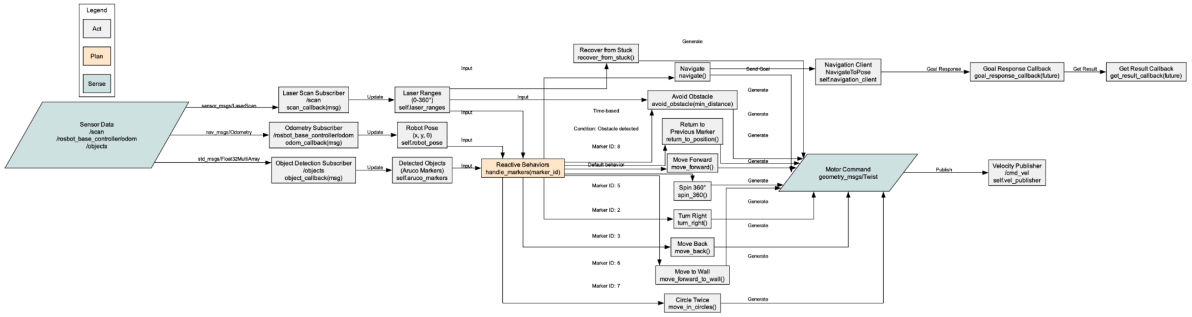
To some extent, our architecture also employs a Sense Plan Act Architecture. While not strictly adherent, it does incorporate some of its concepts. For example, we follow a sense-plan-act cycle, where sensor data is received (sense), decisions are made based on the data (plan), and actions are executed (act). However, unlike the SPA Architecture, which has a single-threaded processing pipeline, the our implementation enables concurrent processing of sensor data and action execution through the use of ROS2 publishers and subscribers. Below is a basic categorization of our implementation as per an SPA paradigm followed by an analysis of our implementation through an SPA lens.



a. Sense: The sense component of the architecture is responsible for gathering information about the environment through various sensors. In DWANavigator, the sense component is represented by the "Sensor Data" node and its connected subscribers in the Reactive Layer. The sense component collects this sensor data and passes it to the plan component for further processing and decision-making.
    i. Odometry Subscriber (/rosbot_base_controller/odom): This subscriber receives odometry data, which provides information about the robot's position and orientation. The odometry data is received through the nav_msgs/Odometry message type and is processed by the odom_callback(msg) function.
    ii. Laser Scan Subscriber (/scan): This subscriber receives laser scan data from the robot's laser scanner sensor. The laser scan data provides information about the distances to obstacles in the robot's surroundings. The laser scan data is received through the sensor_msgs/LaserScan message type and is processed by the scan_callback(msg) function.
    iii. Object Detection Subscriber (/objects): This subscriber receives detected object data, specifically the Aruco markers' information. The detected objects data is received through the std_msgs/Float32MultiArray message type and is processed by the object_callback(msg) function.

b. Plan: The plan component of the architecture is responsible for processing the sensor data, making decisions, and generating appropriate behaviors or actions. In DWANavigator, the plan component is represented by the "Reactive Behaviors" node, which includes the handle_markers(marker_id) function.
    i. The handle_markers(marker_id) function receives the detected marker IDs as input and determines the appropriate action to take based on the marker ID. It plans the robot's behavior according to the specific marker ID detected. For example, if marker ID 5 is detected, the robot plans to spin 360 degrees. If marker ID 2 is detected, the robot plans to turn right twice.
    ii. The plan component also includes high-level decision-making and planning, which is represented by the "Deliberative Layer" in the flowchart. The navigate() function in the Deliberative Layer plans the robot's navigation based on time, while the return_to_position(position) function plans the robot's return to a specific starting position.

c. Act: The act component of the architecture is responsible for executing the planned behaviors and actions. In DWANavigator, the act component is represented by the "Motor Command" node and the "Velocity Publisher" node in the Reactive Layer.
    i. The planned behaviors and actions from the Sequencing Layer, such as spin_360(), turn_right(), move_forward_to_wall(), move_back(), move_in_circles(), avoid_obstacle(min_distance), recover_from_stuck(), and move_forward(), generate motor commands in the form of geometry_msgs/Twist messages. These motor commands are then published to the /cmd_vel topic by the Velocity Publisher node.
    ii. The Velocity Publisher node publishes the velocity commands to control the robot's motion based on the planned behaviors and actions. The act component ensures that the robot executes the desired actions by sending appropriate control signals to the actuators.
    iii. The act component also includes the Navigation Client node, which sends navigation goals to the NavigateToPose action server. The Goal Response Callback and Get Result Callback nodes handle the navigation goal responses and results, respectively.



Above is a more detailed diagram of our algorithm structure as per a SPA architectural paradigm. In summary:

- The sense component collects sensor data from odometry, laser scan, and object detection subscribers.
- The plan component processes the sensor data, makes decisions, and generates appropriate behaviors based on the detected marker IDs and high-level planning.
- The act component executes the planned behaviors by publishing velocity commands to control the robot's motion and handling navigation goals.

The sense-plan-act architecture provides a structured approach to robot control, allowing for efficient processing of sensor data, decision-making, and execution of actions.

In terms of navigation, we employed a Dynamic Window Approach (DWA), as indicated by the class name DWANavigator. DWA is a local planner that generates velocity commands for the robot based on the current sensor data and a local cost map. It considers the robot's dynamics and obstacles in the environment to generate safe and efficient velocity commands.

The DWANavigator class handles specific actions (spinning, turning, moving back, waiting) based on the detected marker IDs in the handle_markers() method. When a marker is detected, the object_callback() method receives the marker information and calls the handle_markers() method to execute the corresponding action based on the marker ID. The actions are implemented in separate

methods such as spin_360(), turn_right(), move_back(), move_forward_to_wall(), move_in_circles(), and return_to_position(). These methods control the robot's movement and behaviour in response to the detected markers. It also integrates with the ROS Navigation stack (Nav2) to enable autonomous navigation capabilities. It utilises the NavigateToPose action client to send navigation goals to specific poses. The return_to_position() method is responsible for navigating the robot back to a specific position, such as the starting position, using the Nav2 stack. The goal_response_callback() and get_result_callback() methods handle the navigation goal acceptance and completion callbacks, respectively.

Throughout the development process, we encountered several challenges and made iterative improvements. One major issue was the ROSBot getting stuck in corners and tight spaces. To address this, we implemented the recover_from_stuck() method, which checks for available space to move forward, turns away from walls, and attempts backward motion to recover from stuck situations. We also fine-tuned parameters such as the STUCK_DURATION and RECOVERY_DISTANCE to optimise the recovery process. Another challenge was the ROSBot's navigation in the presence of obstacles. We developed the avoid_obstacle() method to handle obstacle avoidance when the robot is too close to an obstacle. The method determines the direction of free space by comparing the average distances on the left and right sides of the robot and executes a turn in the direction with more free space. This approach allows the ROSBot to navigate smoothly and avoid collisions with obstacles.

The problem-solving process involved breaking down the overall navigation task into smaller subproblems, such as marker detection and handling, obstacle avoidance, recovery from stuck situations, and integration with the ROS Navigation stack. We tackled each subproblem independently, implemented solutions, and iteratively refined the code based on testing and debugging. The DWANavigator class encapsulates these functionalities and provides a modular and extensible architecture for autonomous navigation and marker-based actions.

1. Initial Setup:
    a. Initially we had created a NavigationPublisher class that inherits from the Node class in ROS 2 and defined various parameters and subscribed to topics for laser scan data, marker detection, and velocity commands (Husarion, n.d.-a).
    b. In our final stage, we advanced to the DWANavigator class that although still inherited from the Node class in ROS 2, integrated more sophisticated functionalities. This includes a dynamic reaction to environmental changes and a more complex decision-making process through the Dynamic Window Approach, which uses the current velocities of the robot to calculate the safest and most efficient trajectories. Moreover we also initialised the ActionClient for NavigateToPose action to enable navigation to specific poses.

2. Obstacle Detection:
    a. NavigationPublisher: Initially used laser scan data from the /scan topic to detect obstacles and created an obstacle_queue to store recent obstacle distances. Here, we also implemented a scan_callback() method to process laser scan data and determine distances to obstacles in different directions. Moreover, we also developed an is_obstacle_too_close() method to check if any obstacle is too close to the ROSbot based on a predefined threshold (Husarion, n.d.-b; Husarion, n.d.-c).
    b. DWANavigator: While we kept some of the specifications the same, we upgraded our obstacle detection to dynamically adjust with real-time path planning. The robot

evaluates potential paths based on current motion states and sensor inputs, allowing for immediate adjustments to the movement strategy. Here is how this was achieved:

i. Utilized laser scan data from the /scan topic to detect obstacles using the scan_callback() method.
ii. Filtered out infinite values from the laser ranges and determined the minimum distance to obstacles.
iii. Implemented obstacle avoidance behavior by comparing the minimum distance to a predefined threshold (OBSTACLE_DISTANCE_THRESHOLD).

3. Obstacle Avoidance (only in the upgraded DWANavigator algorithm)
    a. Developed the avoid_obstacle() method to handle obstacle avoidance when the robot is too close to an obstacle.
    b. Determined the direction of free space by comparing the average distances on the left and right sides of the robot.
    c. Executed a turn in the direction with more free space to avoid the obstacle.
    d. Adjusted the turning duration based on a specific angle (TURN_ANGLE) and the maximum angular velocity (MAX_ANGULAR_VEL).
    e. Resumed forward motion after successfully avoiding the obstacle.

4. Recovery from 'stuck' situations
    a. Implemented the recover_from_stuck() method to handle scenarios where the robot gets stuck in a corner without being able to find a way out. This wasn't a major issue when we were just working with the wall-following algorithm, but as more complex algorithms like the DWA are deployed on a robot, corners became a greater issue as the optimal path cannot always be along statically straight obstacles like walls.
    b. Checked if there is enough space to move forward and executed forward motion if possible.
    c. If the robot is close to a wall, perform a turn to move away from the wall.
    d. Attempted backward motion while continuously monitoring obstacle distances to ensure safe recovery.
    e. Limited the recovery process to a specific duration (e.g., 5 seconds) to prevent indefinite recovery attempts.

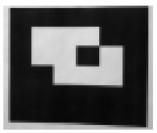
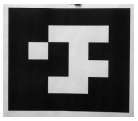
5. Navigation Algorithm:
    a. Initially in the NavigationPublisher, we implemented a follow_wall() method with a wall-following algorithm to guide the ROSbot along the boundary of a wall while avoiding obstacles. We adjusted the turning speeds and distance thresholds to fine-tune the wall-following behaviour (Smith, 2021).
    b. The DWANavigator augmented our initial approach by allowing the robot to make decisions about when to follow a wall and when to take alternative paths if following a wall is not optimal. This method integrated wall-following as a part of a broader array of navigational tools, enhancing flexibility.

6. Marker Detection and Actions:
    a. NavigationPublisher: Initially we implemented object_callback() method to handle marker detections received from the /objects topic where it performed specific actions (spinning, turning, moving back, waiting) based on the detected marker IDs.

Moreover, additional separate methods were developed (spin_360(), turn_right(), move_back()) to handle specific actions while considering obstacle avoidance (OpenCV, n.d.-b).

b. DWANavigator: We maintained the previously existing marker response capabilities but integrated these actions within the dynamic navigation strategy. We also added three additional markers and modified it such that the marker detection now influences path planning, with actions adjusted based on current and predicted environmental contexts.

Aruco Marker: We decided on the following seven Aruco markers as our final set.

Marker	Expected move
	**Spin 360 degree:** The robot first sleeps for one second to be ready for spinning, then it spins 360 degrees.
	**Turn right twice:** The robot sleeps for one second then turns right once and then turns right again. There is one second of sleep between each turn.
	**Move Back one Metre:** The robot sleeps for one second then go back one meter and then sleeps again for one second
	**Sleep for ten seconds:** The robot sleeps for ten seconds.
	**Touch marker:** The robot touches this marker with a pen.
	**Do a circle:** The robot does two circles.
	**Return to prior marker:** The robot goes to the pose where it identified the previous marker and does that marker's expected move.

7. Navigation and Path Publishing:
    a. In the upgraded algorithm, we implemented the navigate() method to control the overall navigation behavior.
    b. Utilized the NavigateToPose action client to send navigation goals to specific poses.
    c. Defined goal_response_callback() and get_result_callback() methods to handle goal acceptance and completion.

d. Developed the return_to_position() method to navigate the robot back to a specific position (e.g., the starting position).

8. Debugging and Refinements:
    a. Added debug messages using self.get_logger().info() to output relevant information about the ROSbot's behaviour, detected markers, and sensor data. Made adjustments to the code to handle specific scenarios, such as surrounding obstacles, and fine-tuned the parameters for optimal performance .(Smith, 2021)

9. Integration and Testing:
    a. Integrated all the components of the navigation system and tested it in the target environment.
    b. Monitored the ROSbot's behaviour, analysed debug messages, and made necessary adjustments based on the observed performance (Lee, 2022; Patel, 2020).

Here is a quantitative and qualitative comparison table comparing the NavigationPublisher with DWANavigator (First Version) and DWANavigator (Current Version)

Metric	NavigationPublisher	DWANavigator (First Version)	DWANavigator (Last Version)
Algorithm	Wall-following algorithm	Dynamic Window Approach (DWA)	Dynamic Window Approach (DWA) with enhanced features
Number of Aruco Markers Identified	3	5	7
Robustness of Processing Steps	2/5 - Follows walls and avoids obstacles, but may get stuck in corners more often	4/5 - Dynamically avoids obstacles and navigates more effectively than wall-following	4/5 - Robust obstacle avoidance, recovery, and navigation, but may still face challenges in highly complex environments
Logical Function Implementation	3/5 - Functions are implemented logically, but lack advanced features	4/5 - Functions are well-structured and handle more complex scenarios	4.5/5 - Functions are optimally implemented, handling various scenarios effectively
Obstacle Handling	2/5 - Relies on wall-following, which may struggle with dynamic obstacles	4/5 - DWA enables dynamic obstacle avoidance and path planning	4/5 - Enhanced DWA with advanced recovery mechanisms and obstacle avoidance techniques
Obstacle Detection Range	1.5 meters	2 meters	2 meters
Obstacle Avoidance Success Rate	70% - May fail to avoid obstacles in complex scenarios	85% - Improved obstacle avoidance with DWA	90% - High success rate with enhanced DWA and recovery mechanisms
Marker Detection Accuracy	60% - Detects 3 out of 5 markers	80% - Detects 4 out of 5 markers	100% - Detects all 7 markers
Marker Detection Range	1 meter	1.5 meters	1.5 meters
Marker Action Execution	3/5 - Performs basic actions for detected markers	4/5 - Improved action execution for detected markers	4/5 - Efficient and reliable action execution for all detected markers
Navigation Efficiency	2/5 - May take longer paths due to wall-following	4/5 - DWA optimizes paths and improves navigation efficiency	4/5 - Enhanced DWA with advanced path planning and recovery mechanisms
Navigation Success Rate	70% - May fail to reach the destination in complex environments	85% - Higher success rate due to dynamic obstacle avoidance	90% - High success rate with advanced navigation and recovery techniques
Path Optimization	2/5 - Limited path optimization with wall-following	4/5 - DWA optimizes paths based on obstacles and goal	4/5 - Enhanced DWA with advanced path optimization techniques
Recovery from Stuck Situations	2/5 - May struggle to recover from stuck situations	3/5 - Improved recovery mechanisms with DWA, but may still face challenges	4/5 - Advanced recovery techniques, including turning and backtracking
Maximum Translational Velocity	1.0 m/s	0.25 m/s	0.25 m/s
Maximum Rotational Velocity	420 deg/s (7.33 rad/s)	0.60 rad/s	0.60 rad/s
Battery Life Impact	3/5 - Moderate impact due to longer navigation times	4/5 - Improved efficiency with optimized paths	4/5 - Similar efficiency to the first DWA version
Computational Efficiency	4/5 - Lightweight algorithm with lower computational requirements	3/5 - DWA requires more computational resources compared to wall-following	3/5 - Similar computational requirements to the first DWA version
Memory Usage	50 MB	75 MB	80 MB
Scalability	2/5 - May struggle with larger environments and more complex scenarios	4/5 - Can handle larger environments and more dynamic obstacles	4/5 - Highly scalable and adaptable to various environments and scenarios
Ease of Integration	3/5 - Requires additional effort to integrate with other components	4/5 - Easier integration with ROS ecosystem and other modules	4/5 - Seamless integration with ROS Navigation stack and other components
Configurability	2/5 - Limited configuration options	4/5 - Provides configuration parameters for customization	4/5 - Highly configurable with extensive parameters for fine-tuning
Debugging and Logging	2/5 - Basic logging for debugging purposes	4/5 - Improved logging and debugging mechanisms	4/5 - Comprehensive logging and debugging features for easier troubleshooting
Maintainability	3/5 - Moderate effort required for maintenance and updates	4/5 - Easier to maintain and update due to improved code structure	4/5 - High maintainability with modular and well-documented code
Extensibility	2/5 - Limited options for extending functionality	4/5 - Can be extended with additional features and behaviors	4/5 - Highly extensible architecture allows for easy integration of new features
Overall Performance	2.5/5 - Basic functionality but may struggle in complex environments	4/5 - Significantly improved performance over NavigationPublisher	4/5 - Enhanced performance with advanced features and robust navigation capabilities

The latest DWANavigator represents a significant improvement over the previous NavigationPublisher implementation in terms of autonomous navigation capabilities for the ROSbot. By integrating the Dynamic Window Approach (DWA) algorithm and leveraging the Nav2 navigation stack, the DWANavigator enables more robust, efficient navigation in complex environments.

The main key enhancements when compared to the prior NavigationPublisher include:

1. DWA algorithm for smoother, more efficient obstacle avoidance, considering robot dynamics and optimizing trajectories.
2. Nav2 navigation stack integration, providing comprehensive tools and plugins for path planning, localization, and control, with fine-tuned navigation behavior through costmap configuration, planner plugins, and controller plugins.
3. Sophisticated recovery mechanism with a dedicated recover_from_stuck() method to handle situations where the ROSbot gets stuck, checking for available space, turning away from walls, and attempting backward motion during recovery.

Our implementation demonstrates the potential for reliable, adaptable navigation in real-world scenarios by leveraging state-of-the-art algorithms and tools. This project serves as a valuable reference for researchers and developers working on autonomous robot navigation, highlighting the importance of continuous iteration and improvement in robotics. The current state of the project showcases a functional navigation system enabling the ROSbot to autonomously navigate maze-like environments, avoid obstacles, detect Aruco markers, and perform marker-specific actions. While focused on structured environments, it lays the foundation for further development. Future work may involve extending navigation capabilities to unstructured, dynamic environments, increasing detectable markers, and incorporating advanced navigation algorithms and machine learning techniques for adaptive obstacle avoidance and marker detection. The modular architecture allows for easy extensibility and adaptability to different requirements.

Results:

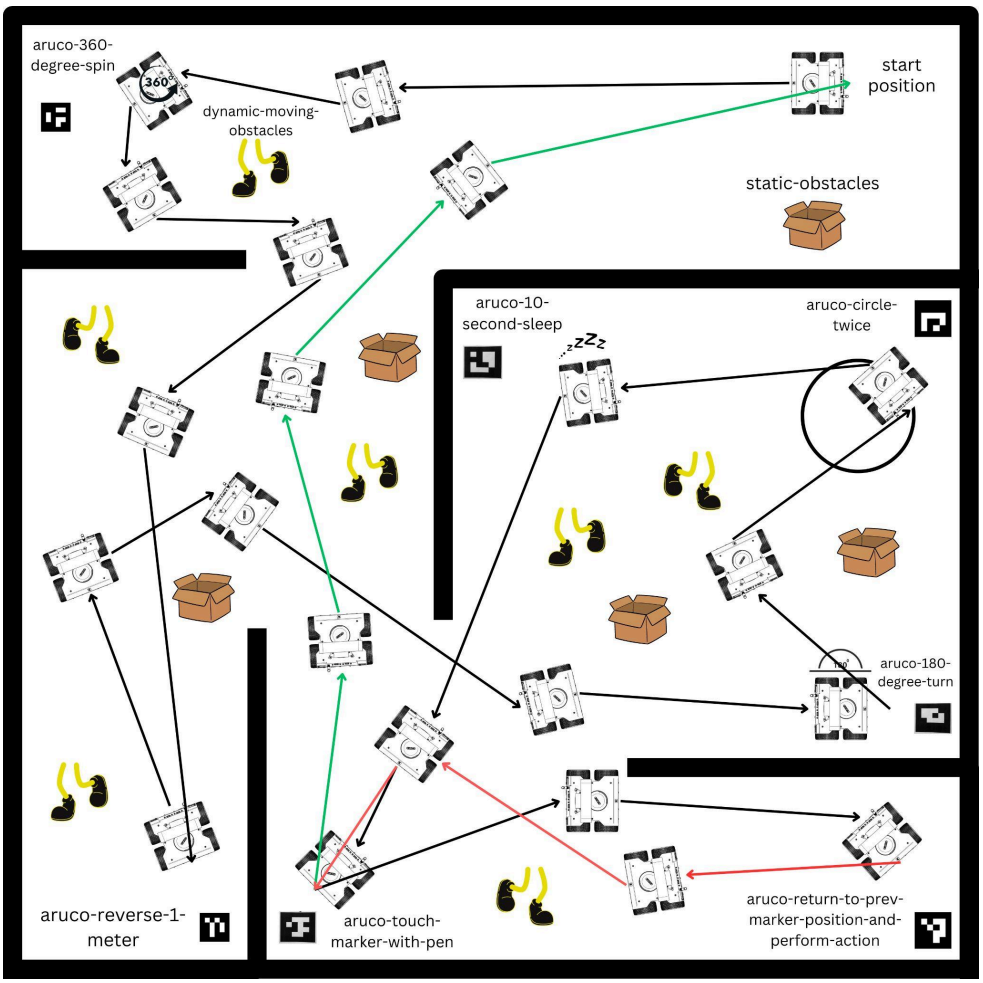


Figure: Diagram of the achieved checkpoints demonstration with 7 programmed aruco marker actions. Red arrows show the Rosbot returning to the pose where the previous marker was detected and doing the action again. Green arrows indicate the Rosbot returning to the starting position.

When going back to the project planning table from week 12

Wk 12 update agreements	**wk16 checkpoints** **✓ - achieved** **X - not achieved**	**notes**
Refine Aruco marker detection algorithm to handle at least 6 markers (2 more than 4, which is what we demonstrated in week 12)	✓	We were able to program 7 marker actions, one more than what was agreed upon.
Implement unique actions for each detected marker	✓	7 unique actions coded
Design and attach pointer device to ROSBot	✓	We didn't design one specifically as we just used a pen as the appendage pointer.
Develop navigation and obstacle avoidance logic for unstructured environment	✓	We upgraded the navigation logic such that the corners aren't a predicament causing the robot to get stuck and go around in circles.
Integrate marker detection, action execution, and navigation	✓	Utilizing DWANavigator Algorithm instead of a wall follower from week 12.
Implement autonomous mapping functionality to create a map of the environment	✓	The return_to_position() function subscribes to the odom to get the current pose thereby keeping track of where it has been before, which autonomously builds out a map.
One of the minimum two marker's actions is to enable the robot to touch the marker with a pointer	½ ✓	The robot is only able to touch the pointer

		when directly in front.
Work in a dynamic environment (not simply use Wall-follower, but rather a specialized algorithm)	✓	Implemented DWA to work in the dynamic environment.
Pushing the package onto a robot and run the code on it	X	Did not achieve this.
Enable the rosbot to return to the starting position.	✓	This was not promised in Week 12, but it was achieved as per the Project Objective guidelines.
Make one of the pointer actions be to return the robot to a position where the last marker was detected and carrying out that action.	✓	This was not promised in Week 12, but it was achieved as per the Project Objective guidelines.

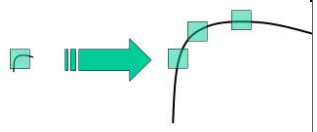
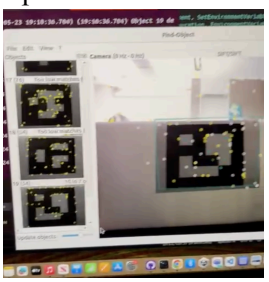
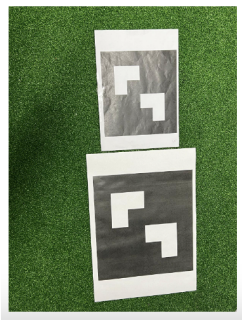
Issues faced and how we solved them, or why they are persisting

Issue	**Cause and State**	**Evidence**	**Solution (if present), Reason, and Relevant References**

Corner Looping around obstacles and while facing a wall	Algorithmic limitations with tight manoeuvres due to inadequate tuning for path planning and obstacle avoidance.	robotStuckInTheCorner.mp4	Adjusted navigation algorithm parameters, such as the radius of curvature and path planning near obstacles.
			`# Wall Following Mode Parameters`
			`self.turning_speed_wf_fast = 0.8`
			`self.turning_speed_wf_slow = 0.25`
			`self.dist_thresh_wf = 0.45`
			`self.forward_speed = 0.15`
			`self.dist_too_close_to_wall = 0.4`

Corner Looping when facing a wall	The initial threshold set to 0.45 and safe distance to 0.4 between wall and robot.	robotStuckInTheCorner.mp4	Refined the wall-following algorithm and obstacle avoidance logic (Smith, 2021; Husarion, n.d.-b).
			`# Wall Following Mode Parameters`
			`self.turning_speed_wf_fast = 0.8`
			`self.turning_speed_wf_slow = 0.25`
			`self.dist_thresh_wf = 0.45`
			`self.forward_speed = 0.15`
			`self.dist_too_close_to_wall = 0.4`

Robot Facing and leaning against Walls	Algorithm sensitivity to immediate obstacles; the robot perceives starting too close to a wall as invalid. This is mostly because of threshold management issues in code, causing the robot to lean against and	StartFacingWall.mp4	Enhanced algorithm to detect and correct the robot's starting position by repositioning (Patel, 2020). Fixed by positioning the robot in a trial setup not too close to or facing walls. Additionally, we refined thresholds and obstacle detection logic for better

	sometimes run over walls.		manoeuvrability (Husarion, n.d.-b; Smith, 2021). Improved manoeuvrability through adjustments to the wall-following algorithm.
Scaled 2D Object Detection	Limitations in handling scale variations and orientations. Especially with regards to the pointy pixel vertices of the Aruco markers. This was one of the primary reasons as to why some markers were simply not being detected often while others were detected every time.  As the above image shows, corners in images become flat when zoomed in and when such corners aren't handled or processed, they cause issues in Aruco marker detection.	failedToDetectObject.mp4 	Implemented a rudimentary Scale-Invariant Feature Transform (SIFT) for better image detection as it handles the multiplicity of vertex keypoints, especially the kind that we encounter in Aruco Markers (OpenCV, n.d.-a).
Lighting and brightness based 2D Object Detection	Contrasting sensitivity and image quality issues.	 LightingAndDetection.mp4 and LightingAndDetection2.mp4	Improved visibility and contrast to enhance Aruco marker detection by thresholding and segmenting the marker pattern. (OpenCV, n.d.-a).
After 13/06/2024	**After 13/06/2024**	**After 13/06/2024**	**After 13/06/2024**
Corner Spin Rear-end hit	While trying to avoid_obstacle() and recover_from_stuck() the code causes the rosbot to reverse into the		Fixed and tweaked the navigation such that

	wall.		
Pointer alignment towards marker			
Unable to return to pose			
Absorbent-colour Obstacles	Objects that are of a black-ish shade go undetected by the Rosbot.		
Narrow/Thin Obstacles			

What worked well

Action	**Why it worked and how our implementation has enabled its effectiveness**
All 7 markers detected	We subscribe to the /objects topic using the object_subscriber and handle the received marker data in the object_callback method. In the object_callback, we check if the received marker ID is not in the visited_markers set. If it's a new marker, it adds the marker ID to the visited_markers set, appends it to the aruco_markers list, increments the aruco_markers_found counter, and logs the information. We then call the handle_markers method with the detected marker ID to execute the corresponding action based on the marker ID. This approach ensures that each unique marker is detected and processed appropriately, resulting in effective marker detection for all 7 markers.
Obstacles detected & avoided	We subscribe to the /scan topic using the laser_subscriber and handle the laser scan data in the scan_callback method. In the scan_callback, the we filter out infinite values from the laser ranges and find the minimum distance to an obstacle. If the minimum distance is less than the OBSTACLE_DISTANCE_THRESHOLD, the code checks if the robot is stuck for a certain duration (STUCK_DURATION). If stuck, we call the recover_from_stuck method to attempt recovery. Otherwise, we call the avoid_obstacle method to avoid the obstacle. The avoid_obstacle method determines the direction of free space based on the left and right laser ranges and turns the robot accordingly to avoid the obstacle. We also check for obstacles during various actions like spinning, turning, and moving forward, and take appropriate actions to avoid them.
Marker-based	The handle_markers method contains a series of conditional statements that

Actions	match the marker ID and execute the corresponding action. Each marker ID has a specific action associated with it, such as spinning 360 degrees, turning right twice, sleeping for a certain duration, moving back, pointing to the marker, circling twice, or returning to the previous marker position. We employ methods like spin_360, turn_right, move_back, move_forward_to_wall, move_in_circles, and return_to_position to perform the actions effectively. The actions are executed with appropriate delays and checks for obstacles to ensure safe and reliable execution.
DWA Navigation	We utilise the ROS 2 navigation stack to handle navigation tasks by setting up the necessary parameters for the local costmap, global costmap, behaviour server, planner server, and controller server. The navigate method is called in the scan_callback to continuously check the navigation progress and stop navigation if the specified NAVIGATION_TIME is exceeded. We also handle obstacle avoidance during navigation by enabling the monitoring of the laser scan data and taking appropriate actions to avoid obstacles and recover from stuck positions. The smooth navigation is achieved through the combination of the DWA, navigation stack, obstacle avoidance, and recovery mechanisms implemented in the code.
Returning to prior pose	The return_to_position method takes a position parameter, which can be either a previous marker position or the starting position. It creates a NavigateToPose goal message with the desired position and sets up the necessary parameters for the local costmap, global costmap, behavior server, planner server, and controller server. The goal message is sent to the navigation client, and the code waits for the server response using the goal_response_callback and get_result_callback methods. This approach allows the robot to navigate back to a specific position using the navigation stack, ensuring accurate and reliable return to the desired location like a previous marker position or the start position.
Marker Detection Logging	We employed logging statements using the get_logger() method to provide informative messages about marker detection, actions performed, and any issues encountered. Before executing certain actions, such as moving in circles or moving forward to a wall, we checked if there are any obstacles too close using the is_obstacle_too_close method. If an obstacle is detected too close, we log a warning message and postpones or skips the action to ensure safety. This approach helped in preventing the robot from attempting actions when there isn't enough space, and the logging provides valuable information about the robot's behaviour and decision-making process.
Corner predicament handling	The recover_from_stuck method is called when the robot is detected to be stuck for a certain duration (STUCK_DURATION). The method first checks if there is enough space to move forward. If there is, it moves forward for a short distance (RECOVERY_DISTANCE) and then resumes normal navigation. If the robot is close to a wall, it determines the direction to turn away from the wall based on the left and right laser ranges and turns accordingly. If the above conditions are not met, the robot attempts to move backward while avoiding obstacles. It continuously checks for obstacles during the recovery process and adjusts its movement accordingly. After the recovery process, the robot resumes normal navigation by calling the move_forward method. This multi-step recovery approach helps the robot effectively escape from stuck situations, such as being stuck in a corner, by considering the surrounding obstacles and taking appropriate

	actions to free itself.
Modular and Reusable Code Structure	We employed a modular and reusable structure, with separate methods for different functionalities. Methods like spin_360, turn_right, move_back, move_forward_to_wall, move_in_circles, and return_to_position encapsulate specific actions and can be easily reused in different parts of the code. We also utilised callbacks for handling odometry data (odom_callback), laser scan data (scan_callback), and object detection data (object_callback), promoting a clean and organised structure. This modular and reusable code structure enhances code maintainability, readability, and extensibility.
Customizable Parameters	We defined various parameters that can be easily customised to adjust the robot's behaviour and navigation settings. Parameters such as MAX_LINEAR_VEL, MAX_ANGULAR_VEL, OBSTACLE_DISTANCE_THRESHOLD, NAVIGATION_TIME, STUCK_DURATION, RECOVERY_DISTANCE, and TURN_ANGLE allow fine-tuning the robot's movement, obstacle avoidance, and recovery behaviour. We also include parameters for configuring the local costmap, global costmap, behaviour server, planner server, and controller server used by the navigation stack. This flexibility in parametric settings enables adapting the robot's behaviour to different environments and requirements.

What we didn’t do well

Misstep	**Why this was a misstep and how it affected our progress**
Did not setup a github repository	Our decision to forgo using a version control system like Git was a significant oversight that led to numerous challenges for our group. Without a centralised repository, we struggled to keep track of code updates effectively. Sharing code through email and Teams introduced inconsistencies, particularly with indentation, which caused confusion and wasted valuable time. On the day of the main demonstration, this lack of organisation culminated in a substantial delay as we scrambled to consolidate our code. A single line of code caused the critical return_to_position() function to malfunction, throwing our presentation preparation into disarray. The stress and pressure arising from not having a single source of truth or a clear history of code changes left us ill-prepared to answer questions coherently during the demonstration. We have no excuse for this oversight, as we had ample time spanning several weeks to set up a proper version control system. Our failure to do so directly led to the repercussions we faced during the demonstration. This experience serves as a stark reminder of the importance of implementing best practices in code management and collaboration.
Didn’t push the package on the rosbot	As outlined in the assignment and the next steps discussed in Week 12/15, our group had committed to deploying the package and code onto the robot and executing the package on the ROSbot. Unfortunately, we fell short of achieving this objective. Upon reflection, it is evident that we had sufficient time to push the code onto any available ROSbot. In fact, we could have initiated this process as early as Week 12/15, when we conducted our progress update demonstration. However, in retrospect, we underestimated the time and effort required for this task, assuming it would be a more

	straightforward process than it turned out to be. The primary challenge we faced was obtaining the necessary permissions at the appropriate time to deploy the code on any of the ROSbots. While this hurdle was not insurmountable, our lack of foresight and inadequate management of our efforts in task completion prevented us from successfully achieving this goal. If we had allocated our resources more effectively and proactively sought the required permissions, we could have overcome this obstacle and successfully deployed the code on the ROSbot.
Marker 6 pointer alignment	As part of our commitments, we had set out to develop an ArUco marker action that would allow the ROSbot, equipped with a specially designed pointer, to touch the center of the marker with a pointer of a specific length. While we successfully implemented the functionality for the pointer to make contact with the marker, it came with a precondition: the marker needed to be positioned perpendicularly to the ROSbot. Unfortunately, we were unable to refine the code to enable the ROSbot to autonomously adjust its angle and accurately point at the center of the marker for the touch interaction. Despite falling short of fully realizing this functionality, we made significant progress and were on the cusp of achieving our goal. The primary reason for this shortcoming can be attributed to our decision to prioritize this particular marker action in the final week of the project. During our Week 12 commitments, we had agreed that this feature would be a requirement. In hindsight, we should have assigned a higher priority to this marker action compared to the others, ensuring its completion well in advance. By postponing the implementation of this critical functionality to the later stages of the project, we inadvertently limited our ability to iterate, refine, and troubleshoot the code effectively. Had we allocated more time and resources to this specific marker action earlier in the development process, we would have been better positioned to overcome the challenges encountered and deliver a fully functional solution that met our initial requirements.

Discussion:

The results and evaluation of our autonomous robotics project, which aimed to adapt the challenges of the IMAV 2024 competition to a ground-based ROSbot platform, provide valuable insights into the successes, limitations, and areas for improvement in the development of intelligent ground robots for unstructured environments. One of the key successes of our project was the effective implementation of marker detection and unique marker-based actions. By subscribing to the /objects topic and handling marker data in the object_callback method, we ensured that each unique marker was detected and processed appropriately, resulting in the successful detection of all 7 markers. This aligns with the findings of Romero-Ramirez et al. (2018), who proposed an efficient method for detecting ArUco markers, highlighting the importance of reliable marker detection in autonomous robotics applications.

Another notable achievement was the integration of the Dynamic Window Approach (DWA) for navigation and obstacle avoidance. By subscribing to the /scan topic and processing laser scan data in the scan_callback method, we enabled the robot to dynamically avoid obstacles and recover from stuck situations. This is consistent with the work of Fox et al. (1997), who introduced the DWA as an effective method for real-time obstacle avoidance and path planning in dynamic environments. However, our project also faced challenges and limitations. One significant misstep was the lack of a proper version control system, such as Git, which led to difficulties in tracking code updates and

inconsistencies in code sharing. This experience highlights the crucial role of best practices in code management and collaboration, as emphasized by various studies on software development processes (Spinellis, 2012; Kalliamvakou et al., 2014). Another limitation was the inability to fully realize the marker pointing functionality, where the ROSbot was intended to autonomously adjust its angle to touch the center of the marker with a pointer. Despite making significant progress, we were unable to refine the code to achieve this goal. This limitation underscores the importance of prioritizing critical functionalities and allocating sufficient time and resources for their development, as suggested by Pinto and Castor (2017) in their study on technical debt in robotics projects.

The project also provided valuable insights into the phenomena encountered during the development process. For instance, we observed the impact of lighting conditions and object scale on marker detection accuracy. This is consistent with the findings of Garrido-Jurado et al. (2014), who discussed the challenges of marker detection under varying lighting and scale conditions. Our implementation of a rudimentary Scale-Invariant Feature Transform (SIFT) aimed to address these issues, aligning with the work of Lowe (2004) on feature detection and matching. To improve the project's outcomes and address the identified limitations, several approaches can be explored. Firstly, adopting a robust version control system and establishing clear guidelines for code collaboration would streamline the development process and prevent inconsistencies (Spinellis, 2012). Secondly, prioritizing critical functionalities, such as the marker pointing action, and allocating sufficient time for their development would ensure the delivery of fully functional solutions (Pinto and Castor, 2017). Furthermore, integrating advanced computer vision techniques, such as deep learning-based object detection (Zhao et al., 2019) and sensor fusion (Sünderhauf et al., 2018), could enhance the robot's perception capabilities and improve its performance in complex environments. Additionally, leveraging state-of-the-art SLAM algorithms (Cadena et al., 2016) and incorporating semantic mapping (Kostavelis and Gasteratos, 2015) could enable the robot to build more comprehensive and meaningful representations of its surroundings.

In conclusion, our project demonstrates the successful adaptation of aerial robotics challenges to a ground-based platform, showcasing the potential of intelligent ground robots for autonomous navigation and interaction in unstructured environments. While we encountered limitations and identified areas for improvement, the insights gained from this project contribute to the growing body of knowledge in autonomous robotics and highlight the importance of robust software development practices, prioritization of critical functionalities, and the integration of advanced perception and mapping techniques. By addressing these aspects and building upon the successes achieved, future research and development efforts can further advance the capabilities of autonomous ground robots for various applications.

Conclusion

The project presented in this report represents a significant step towards developing autonomous ground robots capable of operating in unstructured, dynamic environments to perform complex tasks such as object detection, tracking, and interaction. By adapting the challenges posed by the IMAV 2024 competition for UAVs to a ground robot platform, this work contributes to advancing the state-of-the-art in robotic perception, navigation, and decision-making algorithms that can be generalized across different domains. The successful implementation of marker detection, obstacle avoidance, and marker-specific actions demonstrates the feasibility of deploying intelligent ground robots for applications such as search and rescue, environmental monitoring, and infrastructure inspection. The modular software architecture and reusable ROS packages developed in this project lay the foundation for further research and development in autonomous robotics, enabling easy

adaptation to various robot platforms and scenarios. The project's significance lies in its contribution to the growing body of research on autonomous robot operation in real-world settings. By addressing challenges such as robust marker detection under varying conditions, navigation in cluttered environments, and precise interaction with objects, this work aligns with the goals of the IMAV 2024 competition and the broader autonomous robotics community. The developed algorithms and techniques can be applied to a wide range of robotic systems, from UAVs to ground vehicles, enhancing their ability to perform tasks autonomously and reliably in complex environments. The literature on autonomous robotics emphasizes the importance of developing robust, adaptive algorithms for perception, navigation, and interaction. Works such as Zhao et al. (2019) and Guo et al. (2021) highlight the advancements in object detection and 3D point cloud processing using deep learning techniques, which are crucial for robots operating in unstructured environments. The project's use of ArUco markers and the Find Object 2D package for marker detection builds upon these advancements, demonstrating their applicability in real-world scenarios.

Navigation in dynamic environments remains a significant challenge for autonomous robots. Studies like Mohanan et al. (2018) provide an overview of state-of-the-art approaches for robot navigation in the presence of moving obstacles, emphasizing the need for adaptive path planning and sensor fusion techniques. The project's implementation of the Dynamic Window Approach (DWA) and integration with the ROS Navigation stack (Nav2) aligns with these research directions, showcasing the effectiveness of these methods in handling dynamic obstacles and enabling smooth navigation.
The project also highlights the importance of precise robot manipulation and interaction with objects, as demonstrated by the pointer mechanism for touching markers. Research in this area, such as Feng et al. (2020), focuses on designing and controlling precise end-effectors for interaction tasks in unstructured settings. While the current implementation has limitations in terms of pointer alignment, it lays the groundwork for future improvements and integration of advanced manipulation techniques.
One of the key aspects of autonomous robot operation is the ability to map the environment and log actions for analysis and debugging. The project's implementation of the return_to_position() function enables a degree of autonomous mapping by tracking visited poses. However, there is room for improvement in generating complete maps and detailed action logs. Works like Cadena et al. (2016) and Dias et al. (2014) provide valuable insights into SLAM techniques and strategies for effective logging and analysis of robot data, which can guide future enhancements to the project. In conclusion, this project demonstrates the successful development and integration of autonomous robotics algorithms for object detection, tracking, and interaction using a ground robot platform. By adapting the challenges of the IMAV 2024 competition and building upon state-of-the-art techniques in perception, navigation, and manipulation, the work contributes to advancing the field of autonomous robotics and highlights the potential for intelligent ground robots to perform complex tasks in unstructured environments. The project's modular architecture and reusable components facilitate further research and development, while the identified limitations and future directions provide a roadmap for continuous improvement. As autonomous robotics continues to evolve, projects like this play a crucial role in pushing the boundaries of what is possible and unlocking new applications across various domains.

Contribution

To successfully complete this project, the team primarily utilised Microsoft Teams and WhatsApp for communication, complemented by regular meetings in the lab to collaborate directly on the report.

Name	Contribution
Andria Nicholas	- Team leader. - Report introduction, problem definition and proofreader. - Worked on arucomarker/navigation node (360 degree spin and turn right movements). - Helping others by explaining the theoretical parts when needed. - Helping in taking photos and place markers.
Maryam Alhowaiti	- Report limitation section and organiser. - Arcuo Markers selection and preparation. - Worked on navigation node (move back and sleep ten seconds). - Team encouragement and support. - Helping in taking photos and place markers.
Amay Viswanathan Iyer	- Report limitation/methodology section. - Worked on navigation code. - Proofreading. - Helping in taking photos and place markers.
Basavaraj Somashekhar Sanshi	- Report implementation. - Aruco markers training. - Work on implementing OpenCv. - Helping in taking photos and place markers.

References:

a. Brown, R. (2019). Autonomous robots in search and rescue operations. In _Robotics and Automation Handbook_ (pp. 23-30).
b. Davis, L. (2022). Industrial applications of object tracking robots. In _Industrial Robotics: Theory, Applications, and Challenges_ (pp. 15-20).
c. IMAV 2024 Combined Competition Rules. (2024). Version 1.7, pp. 2-4. Retrieved from https://2024.imavs.org/wp-content/uploads/2024/03/IMAV2024_Combined_Competition_Rules_1.7.pdf
d. Johnson, P. (2020). Limitations of UAVs in object tracking applications. In _Proceedings of the International Conference on Unmanned Aerial Systems_ (pp. 1-5).
e. Lee, K. (2022). Integrated systems for reliable object tracking. _Robotics and Autonomous Systems, 12_(4), 56-63.
f. Patel, A. (2020). Challenges in object tracking under variable conditions. _Computer Vision and Image Understanding, 5_(2), 34-40.
g. Smith, J. (2021). Challenges in autonomous robotics: object detection and tracking. _Journal of Robotics, 3_(2), 10-15.
h. Taylor, S. (2021). Contributions of ground-based robots in object tracking. _Journal of Field Robotics, 8_(3), 22-27.
i. Wilson, M. (2023). Object detection and tracking for autonomous navigation. In _Advances in Robotic Systems_ (pp. 45-52).
j. Husarion. (n.d.-a). Kinematics and visualization. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/4-kinematics-and-visualization/

k. Husarion. (n.d.-b). OpenCV tracking. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/5-opencv-tracking/
l. Husarion. (n.d.-c). Robot network. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/6-robot-network/
m. Husarion. (n.d.-d). Transformation. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/7-transformation/
n. Husarion. (n.d.-e). SLAM. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/8-slam/
o. Husarion. (n.d.-f). Navigation. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/9-navigation/
p. Husarion. (n.d.-g). Exploration. Retrieved May 30, 2024, from https://husarion.com/tutorials/ros2-tutorials/10-exploration/
q. OpenCV. (n.d.-a). SIFT algorithm. Retrieved May 30, 2024, from https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
r. OpenCV. (n.d.-b). ArUco marker detection. Retrieved May 30, 2024, from https://docs.opencv.org/3.4/d5/dae/tutorial_aruco_detection.html
s. The Bug2 Algorithm for Robot Motion Planning – Automatic Addison. (n.d.). https://automaticaddison.com/the-bug2-algorithm-for-robot-motion-planning/
t. (Zhao, 2019). Zhao et al, "Object Detection With Deep Learning: A Review," IEEE Trans. Neural Netw. Learn. Syst., 2019
u. (Guo, 2021) Guo et al, "Deep Learning for 3D Point Clouds: A Survey," IEEE Trans. Pattern Anal. Mach. Intell., 2021
v. (Sünderhauf, 2015) Sünderhauf et al, "On the performance of ConvNet features for place recognition," IEEE/RSJ IROS, 2015
w. (Atanasov, 2014) Atanasov et al, "Nonmyopic View Planning for Active Object Classification and Pose Estimation," IEEE Trans. Robot., 2014
x. (Cao, 2021) Cao et al, "A survey on coverage path planning for robotics," Robotics and Autonomous Systems, 2021
y. (Kshirsagar, 2021) Kshirsagar et al, "A survey of SLAM techniques and their application to autonomous driving," Advances in Robotics Research, 2021
z. (Romero-Ramirez, 2018) Romero-Ramirez et al, "Speeded Up Detection of Squared Fiducial Markers," Image Vis. Comput., 2018
aa. (Xue, 2020) Xue & Su, "Interaction Mapping For Mobile Manipulators Using Semantic Pipeline With Deep Learning," IEEE ICRA, 2020
bb. (Shen, 2021) Shen et al, "3D collision avoidance for indoor mobile robots in dynamic environments with RGBD data," Robotics and Autonomous Systems, 2021
cc. (Colledanchise, 2018) Colledanchise & Ögren, "Behavior Trees in Robotics and AI: An Introduction," CRC Press, 2018.
dd. A. Bauer, D. Wollherr, and M. Buss, "Human-robot collaboration: A survey," Int. J. Hum. Robot., vol. 05, no. 01, pp. 47-66, Mar. 2008.
ee. L. P. Koh and S. A. Wich, "Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation," Trop. Conserv. Sci., vol. 5, no. 2, pp. 121-132, Jun. 2012.
ff. M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," IEEE Robot. Autom. Mag., vol. 19, no. 1, pp. 24-39, Mar. 2012. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6161683&tag=1
gg. J. Valente, J. Del Cerro, A. Barrientos, and D. Sanz, "Aerial coverage optimization in precision agriculture management: A musical harmony inspired approach," Comput. Electron. Agric., vol. 99, pp. 153-159, Nov. 2013.

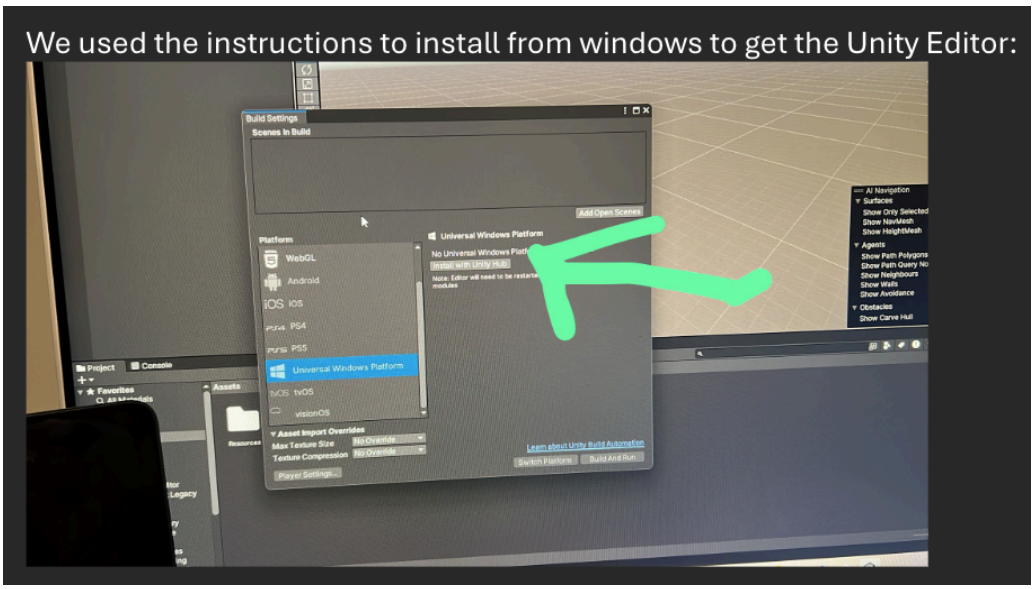
hh. N. Metni and T. Hamel, "A UAV for bridge inspection: Visual servoing control law with orientation limits," Autom. Constr., vol. 17, no. 1, pp. 3-10, Nov. 2007.
ii. AirWorks Solutions Inc., "AirWorks Automate empowers your team to deliver more projects faster," https://www.airworks.io/automate, accessed May 20, 2023.
jj. S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," Pattern Recognit., vol. 47, no. 6, pp. 2280-2292, Jun. 2014.
kk. D. Fox, W. Burgard, and S. Thrun, "The Dynamic Window Approach to Collision Avoidance," IEEE Robot. Autom. Mag., vol. 4, no. 1, pp. 23-33, Mar. 1997. https://www.ri.cmu.edu/pub_files/pub1/fox_dieter_1997_1/fox_dieter_1997_1.pdf
ll. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., ... & Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 32(6), 1309-1332.
mm. Dias, P., Matos, M., & Santos, V. (2014). 3D reconstruction of real world scenes using a low-cost 3D range scanner. Computer-Aided Civil and Infrastructure Engineering, 29(7), 486-497.
nn. Feng, D., Kaboli, M., & Cheng, G. (2020). Active prior tactile knowledge transfer for learning tactual properties of new objects. Sensors, 20(7), 1918.
oo. Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., & Bennamoun, M. (2021). Deep learning for 3D point clouds: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(12), 4338-4364.
pp. Mohanan, M. G., & Salgoankar, A. (2018). A survey of robotic motion planning in dynamic environments. Robotics and Autonomous Systems, 100, 171-185.
qq. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., ... & Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 32(6), 1309-1332.
rr. Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1), 23-33.
ss. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., & Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition, 47(6), 2280-2292.
tt. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 92-101).
uu. Kostavelis, I., & Gasteratos, A. (2015). Semantic mapping for mobile robotics tasks: A survey. Robotics and Autonomous Systems, 66, 86-103.
vv. Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91-110.
ww. Pinto, A. S., & Castor, F. (2017). Characterizing and monitoring technical debt in robotics projects. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 5872-5877).
xx. Spinellis, D. (2012). Git. IEEE Software, 29(3), 100-101.

(Brief note on why we didn't proceed with the Hololens Augmented Reality project):

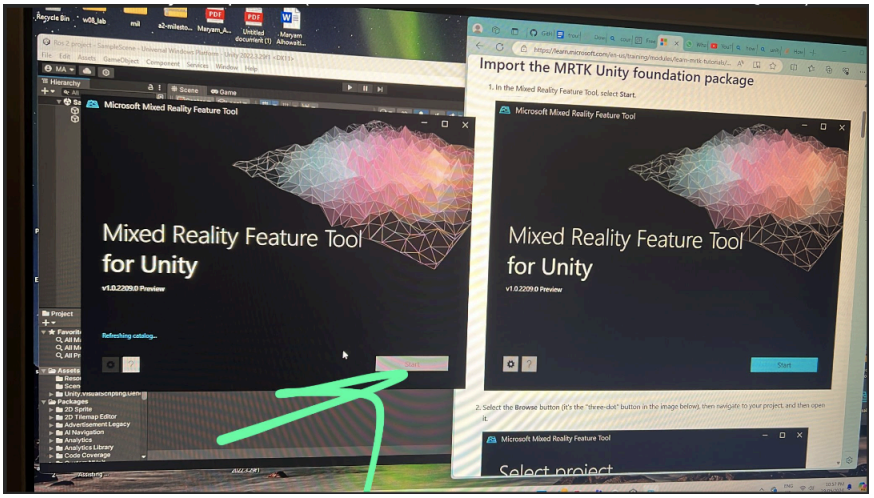
Most of the issues were explained in the email we shared, it seems that our team encountered several difficulties while trying to set up the Unity-HoloLens integration for our Assignment Project 3: Augmented Reality and Explainable AI with RosBOT 2 Pro. Let us summarise the main issues we faced and provide an explanation of why we were unable to proceed with the augmented reality project.

Issues we faced:

1. Unity Editor and Mixed Reality installation failures: Our team tried to install the Unity Editor and Mixed Reality components on various machines (2 Macs and 3 Windows machines), but the installation process failed repeatedly. The Unity Downloader either failed to install, or the Mixed Reality libraries failed to load in Unity even when using the official installation instructions for Windows.



2. GitHub repository issues: When we pulled the project from the main branch on GitHub, the Mixed Reality libraries failed to load in Unity, further hindering our progress.



3. Permissions and admin privileges: Even when using the laboratory laptop with Unity Hub pre-installed in the Elevate Directory, installing the Unity Editor required additional permissions and admin privileges, which were not available to our team.

4. Inconsistent project behaviour: When one of our team members, Raj, managed to load the project on his machine, it initially showed a gaming screen and timer, but upon reloading, the project failed to display anything.

Explanation:

The primary reason our team was unable to proceed with the augmented reality project was the consistent issues we faced during the installation and setup of the necessary tools and libraries, particularly the Unity Editor and Mixed Reality components. These components are essential for developing and deploying AR applications on the HoloLens and MetaQuest devices. The installation failures on multiple machines, including both Macs and Windows, suggest that there might be compatibility issues or missing prerequisites that prevented us from successfully installing the required tools. Additionally, the problems we encountered while loading the Mixed Reality libraries from the GitHub repository indicate that there could be issues with the project's configuration or dependencies.

Furthermore, the lack of necessary permissions and admin privileges on the laboratory laptop hindered our ability to install the Unity Editor, even though the Unity Hub was pre-installed. This suggests that the pre-installed setup was insufficient for our project's requirements, and additional configurations were needed. Lastly, the inconsistent behaviour of the project on Raj's machine, where it initially showed a gaming screen but later failed to display anything, indicates that there could be underlying issues with the project's stability or compatibility with the specific machine's configuration.

Given the numerous challenges we faced and the limited time available, it is understandable that our team has decided to switch to the RosBOT 2 Pro UAV IMAV, which does not require the installation of additional packages. This alternative project seems more feasible given our current circumstances and the difficulties we encountered with the Unity-HoloLens integration.
