UNIVERSITY OF LEEDS

Robotics Coursework Group Project Report

Group 17 Members:
Ali Almoharif
Abderrahamane Bennabet
Adham Hamza
Zaid Shahrouri

May 2024

Contents

1 Introduction

This report presents a comprehensive solution to robotic navigation and target identification problems. The project involves developing a robot capable of autonomously entering the correct room, navigating within it, identifying specific targets, and planning optimal paths to reach them. The solution encompasses various aspects of robotics, including colour detection, motion planning, and image processing. The report is structured as follows:

1. **Design:** Analyses the problem, outlines the proposed solution and details the system components.
2. **Implementation and Results:** Describes the implementation process, machine learning models, motion planning algorithms, and results from simulations and testing.
3. **Real Robot Test:** Discusses the adaptation of the solution for real-world application and evaluates the robot's performance in a natural environment.
4. **Discussion:** Reflects on the project's strengths, weaknesses, and limitations.
5. **Conclusion:** Summarises the findings and suggests future improvements.

2 Design

2.1 Problem Analysis

The problem has been broken down into several distinct yet interconnected issues as illustrated in Figure 1:

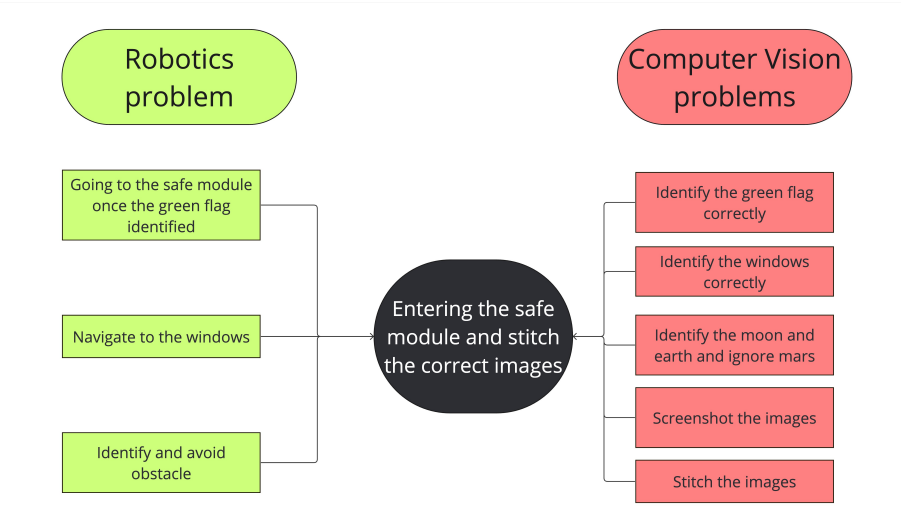


Figure 1: Problems breakdown and analysis

The problems, although defined separately, form a sequence that must be tackled step by step. Initially, the robot must identify green and red flags while ignoring other similarly coloured objects to avoid confusion. Once the flags are detected, the robot should navigate to the safe module (indicated by the green flag). The problem provides the coordinates of the safe and dangerous modules, simplifying access to the correct room. However, this assumes prior knowledge of the environment, posing a challenge if the robot must operate independently without predefined coordinates. After entering the safe module, the robot must detect windows, plan optimal paths, and ensure it captures images containing the moon and Earth, eventually stitching them.

2.2 Proposed Solution

Based on the problem analysis, the proposed solution is structured as follows:

2.2.1 Colour Detection

The robot utilises a combination of two methods for colour detection:

1. **HSV Colour Segmentation and Colour Masks:** This method converts the image to HSV colour space and creates binary masks for specific colours (green, red, and white).
2. **Contour Detection:** This method applies contour detection to these masks to identify the shapes of the coloured regions.

These methods ensure accurate detection of the green and red circles of the entrances while avoiding incorrectly identifying objects with similar colours. Figure 2 displays the flow of how the colours mentioned are detected.

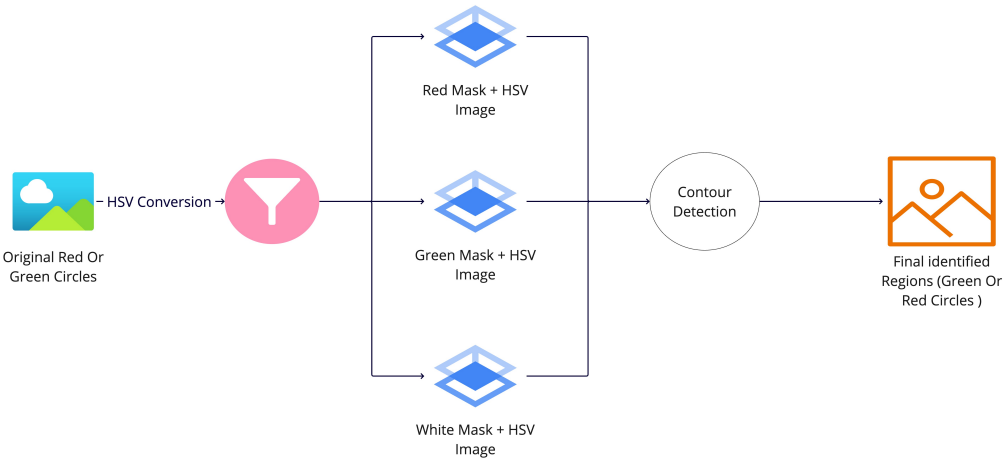


Figure 2: Colour Detection Process

2.2.2 Safe Module Navigation

Our initial plan involved our robot calculating the midpoint of the given entrance coordinates for the current world and moving to it to perform colour identification.

The robot uses its camera to detect coloured signs within its environment. Upon identifying a green sign, it calculates its position within the camera's view and stores this position to decide which room to enter. When both green and red signs are detected, the robot compares their positions about the given centre x and y points for each module and how far they are from the robot frame to identify which of the two modules is green and which is red.

The robot determines whether to move to Module 1 or Module 2 based on the relative positions of the signs compared to the centre points given for each module.

2.2.3 Motion Planning

A custom closed-loop motion planning algorithm was implemented. The algorithm starts by calculating a square centred around the room's centre coordinates, as shown in Figure 3. A closed-loop control system, also known as a feedback control system, continuously adjusts its actions based on real-time feedback. In our approach, the robot uses sensor feedback and a set of flags to determine if the final goal has been reached, making necessary decisions and adjustments while navigating the search area (our square).

The algorithm systematically visits each corner of the calculated square. If both the "moon" and "earth" are not detected after visiting all corners, the algorithm incrementally increases the square's size and repeats until they are detected.

Upon reaching a destination point (i.e. one of the corners of our artificial square), the robot performs a 360-degree rotation about that coordinate. If the robot detects the desired window, it centres it in the middle of its camera frame and moves forward until it reaches a safe distance. After processing the window, the robot proceeds to the next un-visited corner of our square to continue its search.

If the robot detects a window en route to a corner, it cancels its current goal, moves to the detected window for processing, and resumes its path to the following designated vertex. This method ensures comprehensive area coverage while avoiding obstacles and not wasting time visiting a vertex even when it identifies a window. We methodically expand the search until the robot finds the desired objects or the maximum search area is covered.

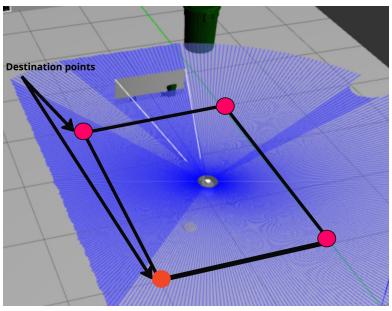


Figure 3: Closed-Loop Control

2.2.4 Window Detection and Planet Classification

The robot uses contour detection in OpenCV (cv2) to identify rectangular shapes representing windows. Once a window is detected and determined to be at an appropriate distance, verified by a flag (`close_enough_to_window`), the region of interest (ROI) is adjusted to focus only on the circle inside that rectangle, which represents the enclosed planet. The robot crops this area, processes it, and passes it to a pre-trained machine learning model to determine whether it is Earth or the Moon. If the identified planet is neither, the model returns 0.

The robot checks the distance to the window by using predefined width and height thresholds; if the detected window exceeds these thresholds, it is considered close enough. To avoid duplicates, the robot uses the ORB (Oriented FAST and Rotated BRIEF) feature detector. ORB is a robust and efficient method for detecting and describing local features in images, allowing the ability to recognise previously seen windows and avoid redundant screenshots.

Additionally, the robot ignores fake windows (present in the complex map) by counting the number of circles within the border of the windows, assisted by the machine learning model. This approach ensures efficient identification of the correct celestial bodies, capturing and saving screenshots only when the robot finds the appropriate windows.

Figure 4 summarises the planet detection pipeline, integrating steps from window detection and cropping the planet to passing it to the model and receiving output.

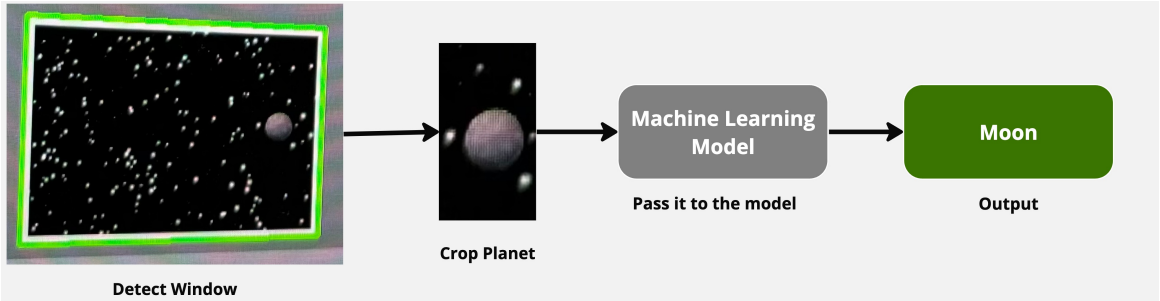


Figure 4: Planet Detection Pipeline

2.2.5 Image Stitching

The process begins with loading images in full colour, which is crucial for accurate colour detection and manipulation. The images are then converted to the HSV colour space, making it easier to specify colour thresholds. We created a binary mask that uses these thresholds to isolate specific colours. This mask is refined through dilation, expanding the highlighted regions, making the isolated objects more pronounced and accessible to extract.

Next, the algorithm blends the isolated image into a base image. It determines the optimal position for overlaying the isolated image, ensuring it fits within the borders to maintain visual continuity. If the isolated image is too large, the algorithm resizes both images accordingly. The blending process uses bitwise operations to mask the isolated image, preserving only the desired object and then combining it with the base image. This method ensures a seamless merge with no abrupt transitions.

This method is robust and efficient for several reasons:

1. Accurate Color Detection: Using HSV colour space and defined thresholds allows for precise colour isolation.
2. Enhanced Object Extraction: Dilation refines the mask, making objects more prominent and easier to utilise.
3. Optimal Positioning and Resizing: Ensure the isolated image fits within the base image's borders to maintain visual continuity.
4. Seamless Blending: Bitwise operations ensure smooth integration of images without abrupt transitions.

This approach provides a detailed and uninterrupted visual representation of the robot's surroundings, which is crucial for effective navigation and interaction in complex environments.

2.3 Overall Workflow

The workflow of the algorithm is summarised in the flow diagram shown in Figure 5:

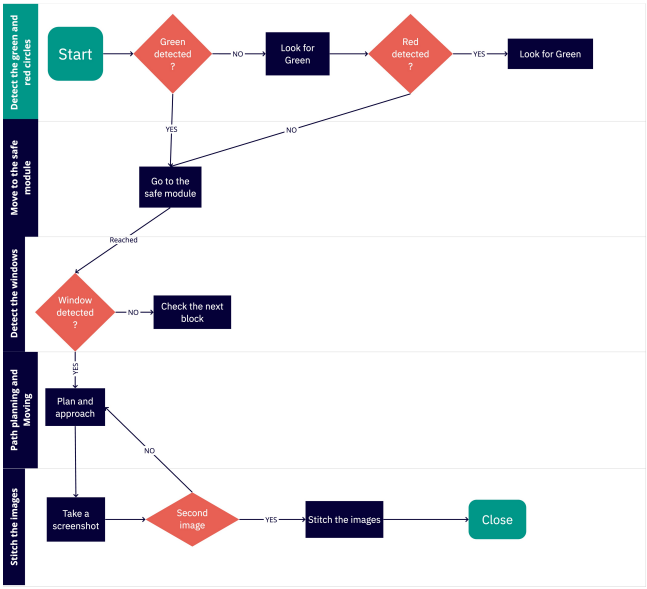


Figure 5: Proposed Algorithm Workflow

3 Implementation and Results

Figure 9 illustrates each node of the algorithm developed in the preceding section. The architecture is made both robust and modular by allocating distinct functionalities to individual classes, where each class corresponds to a ROS node.

Figure 6: Overview class diagram

3.1 Machine Learning Model for Planet Processing

3.1.1 Data Collection and Cleaning

Model training began with data collection from two primary sources: a custom dataset using the Google Chrome extension "Download All Images" and images from the Planets and Moons Dataset on GitHub. A custom Python script pre-processed our data by filtering non-relevant file extensions and removing corrupted files, resulting in a high-quality and clean dataset.

3.1.2 Data Structure

We organised the dataset into a structured folder hierarchy, with training and validation sets containing Earth and Moon image subfolders. This organisation maintained a clear separation between learning and evaluation data.

3.1.3 Data Preparation and Augmentation

1. **Training Data Augmentation**: An `ImageDataGenerator` applied augmentation techniques, including rescaling, random rotations, shifts, shear, zooms, and flips, to enhance the model's generalisation.
2. **Validation Data Preparation**: A separate `ImageDataGenerator` was used for rescaling validation data to normalise pixel values.
3. **Image Loading**: Images were loaded using `flow_from_directory`, resized to 256x256 pixels, and batched into groups of 20. The class mode was set to 'binary' for the most suitable classification tasks.

3.1.4 Model Architecture

The model architecture balanced performance and efficiency, starting with an `InputLayer` for the expected input shape. It included `SeparableConv2D` and `BatchNormalization` layers for computational efficiency and regularisation. `MaxPooling2D` layers reduced spatial dimensions, and a `Dropout` layer prevented overfitting. A `GlobalAveragePooling2D` layer reduced complexity, and the final `Dense` layer with a sigmoid activation function was used for binary classification.

3.1.5 Model Compilation and Training

The model was compiled with the Adam optimiser and binary cross-entropy loss function, using accuracy as the metric. Two callbacks, `EarlyStopping` and `ModelCheckpoint`, enhanced the training process:

- **EarlyStopping**: Halted training if no improvement was seen after a specified number of epochs and restored the best weights.
- **ModelCheckpoint**: Saved the model with the best validation accuracy.

The training involved data augmentation and validation on rescaled data. Although a set for 100 epochs was made, training could stop earlier if the model met our set conditions.

3.2 Motion Planning

3.2.1 Entering the Room

The `callback` method processes incoming images to detect red and green colours that indicate room entrances. OpenCV converts images to the HSV colour space, and colour thresholds are applied to create red, green, and white masks. Contours are extracted from these masks to identify the positions of the red and green signs and the white rectangles surrounding each within the camera's view.

The initial step was for the robot to move to the entrance coordinates of module 1. The TurtleBot then rotates counter-clockwise with an angular velocity of 0.2, picked to accurately identify the red and green coordinates until it identifies the coordinates of the red and green signs relative to the robot frame. For this step, to distinguish between the entrance signs and other objects that may be of the same or similar colours, we identified entrance signs by their given colour and the white square surrounding them. This process was done using the following steps;

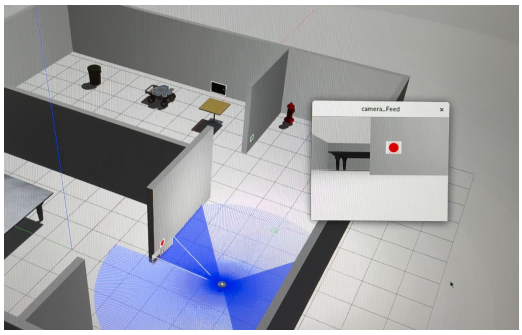
1. Identify red and green contours
2. Check for the presence of white contours

3. Compare coordinates of red/green (Whichever is spotted) and the white contours
4. If they are the same or close (their difference is under a set threshold), that colour is identified as the entrance sign.

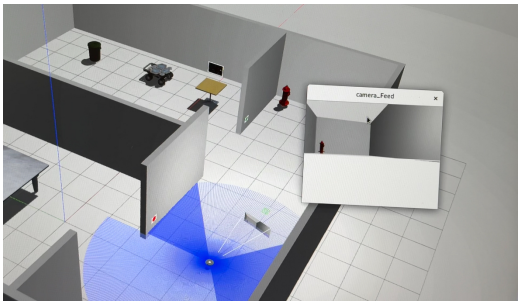
This is clearly shown in both sub-figures in Figure 5. Part (a) shows a red bounding box around the red sign, indicating that it has been identified as the sign and not a dummy object of the same colour. Part (b) shows the fire hydrant (of colour red) not being bounded by a rectangle, which proves that our approach works correctly and does not misclassify similarly coloured objects as the entrances. The `GoToPose` class sends navigation goals to the robot, directing it to the centre of the identified module. The robot stops upon reaching the entrance to ensure it has successfully arrived.

After reaching the entrance, the robot navigates to the centre of the selected module using the `send_goal` method of the `GoToPose` class. The robot adjusts its movements based on feedback from the detected signs to ensure accurate entry into the room. Once the robot achieves the goal, it stops completing the task.

This approach integrates colour detection, real-time image processing, and precise movement commands, allowing the robot to navigate and enter rooms efficiently and autonomously. The combination of these techniques ensures robust and accurate performance in various environments.



(a) Red Sign: Correct Bounding Box



(b) Fire Hydrant: No bounding box

Figure 7: Comparison of images

3.2.2 Room Exploration

Implementing the motion planning algorithm involves several key steps and calculations to ensure accurate and efficient navigation.

Calculation of Square Vertices An artificial square was created to cover all parts of the walls in the room. Our algorithm calculates the vertices of this square based on the centre coordinates $(\text{center\_x}, \text{center\_y})$ using the following formula:

$$\left(\text{center\_x} + \left(\frac{\text{length}}{2} \times \cos\left(\theta + \frac{\pi}{2}\right)\right), \text{center\_y} + \left(\frac{\text{length}}{2} \times \sin\left(\theta + \frac{\pi}{2}\right)\right)\right)$$

where θ takes the values $0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$ for the four vertices of the square. The variable length is predefined and adjustable.

Movement Logic The algorithm visits each corner of the square sequentially. If the target objects ("moon" and "earth") are not detected after visiting all corners, the square's length is incrementally increased, and the search continues.

Destination Handling When the robot reaches a destination point (corner), it performs a 360-degree rotation to scan for the desired window. If detected, the robot centres the window in its camera frame, moves forward to process it, and then continues to the next unvisited corner of the square.

Dynamic Goal Adjustment Suppose the robot detects a window while en route to a destination. In that case, it cancels the current goal, processes the detected window, and then resumes its path to the following designated vertex. This dynamic adjustment ensures efficient navigation and comprehensive area coverage.

Goal Canceling To handle the scenario where the robot spots a window while on the way to a destination, we have implemented a new function, `cancel_goal`. This function cancels the current goal and increments the index to the next vertex in the square, ensuring that the robot continues its navigation seamlessly after processing the detected window.

Moving Forward to the window When moving forward to the window, the system aligns the window frame with the centre of the camera frame by calculating the difference in the x-coordinate between the centre of the camera frame and the centre of the rectangle frame. If the difference is negative, the robot rotates counter-clockwise; otherwise, it rotates clockwise. The robot checks if the window is centred using the following formulas:

$$\text{center\_x} = x + \frac{\text{width}}{2}$$

$$\text{center\_y} = y + \frac{\text{height}}{2}$$

$$\text{image\_center\_x} = \frac{\text{image.shape[1]}}{2}$$

$$\text{image\_center\_y} = \frac{\text{image.shape[0]}}{2}$$

$$\text{error\_x} = \text{center\_x} - \text{image\_center\_x}$$

If the absolute value of error_x is less than 10% of the image width, the robot stops rotating and adjusts its velocity based on the distance to the window. This ensures that the window is centred correctly in the camera frame for accurate processing.

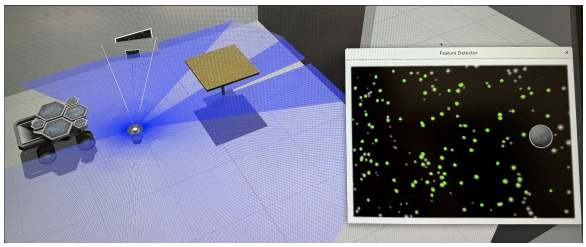
3.2.3 Window Detection

The `find_window_border` method detects windows by converting the image to a grey scale and applying thresholding to create a binary image. Morphological operations enhance contour detection, approximating contours to identify rectangular shapes. Rectangles that match size and shape criteria are considered window borders.

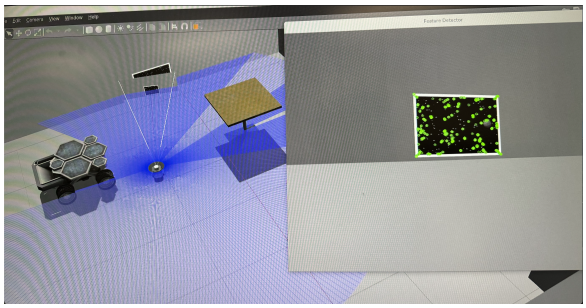
Once the robot identifies a window, the `find_circle_inside_window` method determines if a celestial body (Earth or Moon) is visible. It calculates the bounding rectangle for the window, crops the region of interest (ROI), and uses the Hough Circle Transform to detect circular shapes within the ROI. Detected circles suggest the presence of a celestial body.

To ensure accurate detection, the robot calculates the error between the centre of the detected window and the image centre, adjusting its position using the `RobotController` module. The method `crop_circle` crops the celestial body from the window area, and `predict_planets` uses a pre-trained model to classify the image as Earth, Moon, or neither.

The modular design of the window detection and image processing system ensures efficiency and robustness. Distinct methods for detecting windows, identifying celestial bodies, and adjusting the robot's position allow for straightforward task integration. Real-time data processing and adaptive movement strategies enable the robot to accurately detect and classify windows and celestial bodies in various environments.



(a) Bounding Box around Planet



(b) Feature Identification

Figure 7 displays how bounding boxes are placed around detected windows and the planets, showing an accurate representation of our approach and how different planets are distinguished. Part (a) shows a combination of the feature detection in our Moon window and the bounding box around the moon itself. Part (b) displays the feature identification for our window and the bounding box around it.

3.3 Image Stitching

After loading the images, the `isolate_earth_by_color` function isolates the Earth in its image by converting it to HSV colour space and creating a mask based on colour. Morphological operations, such as dilation, fill gaps in the mask, and the isolated Earth image is obtained by applying the mask to the original image.

The `blend_images` method integrates the isolated Earth image with the star background. It calculates the position to place the Earth image near the left border of the background, ensuring it is vertically centred. If the region of interest exceeds image boundaries, the algorithm resizes the images accordingly. The Earth image is blended into the star background using bitwise operations, resulting in a seamless composite image.

The `stitch_images` method conducts the entire stitching process. It loads the images, isolates the Earth image, and blends it with the star background. The final stitched image is saved and displayed in the specified output directory. This method also invokes the `detect_and_measure_planets` method to detect and measure celestial bodies in the final image.

The `detect_and_measure_planets` method converts the final stitched image to grayscale and applies a median blur to reduce noise. It uses the Hough Circle Transform to detect circular shapes, indicating celestial bodies. Detected circles are marked, and their coordinates and radii are recorded. Using the formula:

$$\text{Distance} = \left(\frac{\text{Actual Radius (km)}}{\text{Pixel Radius}} \right) \times \text{Scale Factor}$$

The `PlanetaryMeasurements` class calculates the distances between the Earth and Moon based on these circles, saving the measurements to a text file.

The image stitching system's modular design ensures both accuracy and robustness. Distinct methods for loading, isolating, blending, and measuring images allow for explicit task integration. The system handles image size and positioning variations, ensuring an accurate and visually cohesive final stitched image.

3.4 Results

For the simulation testing in both the easy, moderate, and complex world types, the robot successfully demonstrated its capability to navigate and perform tasks within the environment. It quickly detected both entrance signs (red and green) and successfully entered the designated room module. The robot then scanned the room by rotating about the centre and ensured a clear path for navigation by utilising our square method algorithm.

Its use of the window and feature detection algorithms enabled it to identify windows and maintain an appropriate distance from them for optimal capturing. Furthermore, the feature detection algorithm effectively recognised celestial bodies, allowing the machine learning model to classify them accurately and store the information in the designated path. The stitching algorithm successfully combined both images, providing a comprehensive visual output of the environment. All this was done in under 5 minutes, as shown in Table 1.

The robot faced some difficulty in the world setting. Initially, it seemed to detect an obstacle collision, causing it to mis-calibrate, but it was able to overcome the problem and proceed with the execution smoothly. The initial obstacle caused the robot to exceed the time limit of 5 minutes, as shown in Table 1

Utilising the model in section 3.1, it took the robot between 11ms to 18ms to correctly classify the celestial bodies, indicating the high performance of the ML model in classifying the planets.

World Type	Time
Easy	3 min and 17 sec
Moderate	3 min and 32 sec
Hard	6 min and 17 sec
Real	NA

Table 1: Performance times of the robot in different complexity levels of the testing environment.

Link to our solution on the Easy, Moderate and Hard worlds: Demo on Gazebo

4 Real Robot Test

Our algorithm has been tested against the real-world map to ensure that this algorithm, used in reality, would work effectively.

4.1 Adaptation for Real Robot

Navigation and mapping systems were essential for the robot to navigate and operate in real-world environments effectively. We developed a new environmental mapping, as demonstrated in the attached visualisation.

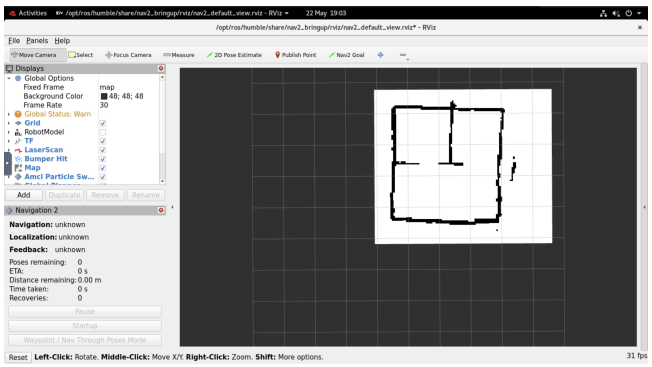


Figure 9: Real World Mapping

4.2 Performance in Real World

During real-world testing, The robot was able to navigate correctly to the correct green room; however, it failed in window detection. High variability in brightness led to the frequent misidentification of colours while processing latency delayed timely decision-making.

Additionally, image compression introduced artefacts that distorted colour information, complicating detection accuracy. The initial use of an RGB-based colour detection method proved less effective under varied lighting conditions, failing to detect essential cues such as white frames. These experiences underline the necessity of adopting an HSV (Hue-Saturation-Value) colour model, which could better differentiate colours independently of lighting conditions.

Link to demonstration on the real robot: Real Robot Demo

5 Discussion

In this section, we analyse our project's strengths, weaknesses, and limitations, reflecting on the overall effectiveness and implementation of our proposed solutions.

5.1 Strengths

Robust Colour Detection: HSV Colour Segmentation and Contour Detection proved highly effective in identifying the green and red flags. This method provided accurate and reliable detection, crucial for the robot's navigation and decision-making processes. Combining these techniques ensured the robot could distinguish between the target colours and other similarly coloured objects in the environment, eliminating false positives.

Advanced Image Stitching: The image stitching process was designed to be robust and efficient. The method ensured high-quality, comprehensive visual outputs by converting images to the HSV colour space and using bitwise operations for seamless blending. This was essential for the robot's capturing and stitching images of the Earth and Moon, providing accurate and uninterrupted visual representations.

Machine Learning Model for Image Processing: Another strength was the machine learning model developed for differentiating celestial bodies. The model's architecture balanced performance and efficiency, including SeparableConv2D and BatchNormalization layers. Data augmentation techniques enhanced the model's generalisation ability, ensuring reliable performance in varied conditions.

5.2 Weaknesses

Dependency on Predefined Coordinates : One of the primary areas for improvement was the reliance on predefined coordinates for the safe and dangerous modules. This dependency limits the robot's ability to operate autonomously in unseen environments. The robot may not always access predefined coordinates in real-world scenarios, requiring more advanced environment mapping and localisation techniques.

Limited Real-World Testing: While the simulation results were promising, the real-world testing was limited. Adapting the implementation to the real robot faced challenges, mainly with environmental variations and hardware limitations not accounted for. This highlighted a gap between simulation performance and real-world applicability.

Inability to Identify Windows behind obstacles: When our TurtleBot is at the co-ordinates of a square corner, and there is an obstacle right at the corner blocking the view to our window, our algorithm fails to detect that blocked window and proceeds by moving to the next corner of our artificial square.

5.3 Limitations

Environmental Assumptions: The project assumed certain environmental conditions, such as the availability of clear and distinguishable green and red signs and predefined coordinates. These assumptions may not hold in more complex or cluttered environments, affecting the robot's ability to perform its tasks accurately.

Sensor Accuracy and Limitations: The performance of the colour detection and image processing algorithms heavily depends on the quality and accuracy of the robot's sensors. Variations in lighting conditions, sensor noise, and other factors can impact the reliability of the detection and classification processes.

Scalability: The solutions implemented were tailored for the specific problem set defined in this project. Scaling these solutions to more complex tasks or larger environments would require significant modifications and additional considerations, such as enhanced localisation techniques and more sophisticated environment mapping.

6 Conclusion

Reflecting on our project, we recognise the considerable progress in addressing the robotic navigation and image processing challenges. The strengths of our approaches demonstrate the potential for effective robotic solutions in controlled environments. However, the identified weaknesses and limitations emphasise the need for further development and testing, particularly in real-world scenarios and more complex environments. Future work should focus on enhancing the robot's autonomous capabilities, improving sensor accuracy, and ensuring the scalability of our solutions to broader applications.

7 Future Improvements

Several improvements can be made to address the weaknesses and limitations identified:

Advanced Environment Mapping and Localization: To reduce dependency on predefined coordinates, implementing more advanced environment mapping and localisation techniques, such as Simultaneous Localization and Mapping (SLAM), would enable the robot to operate autonomously in unfamiliar environments. This approach would allow the robot to create a map of its surroundings and use it for navigation.

Enhanced Real-World Testing: Increasing the extent and variety of real-world testing would help bridge the gap between simulation and real-world performance. Testing the robot in diverse environments with different lighting conditions, obstacles, and layout variations would provide valuable insights into its robustness and adaptability.

Improving Path Planning Algorithms: Developing complete and robust path planning algorithms that guarantee finding a solution, even in dynamic and obstacle-rich environments, would enhance the robot's navigation capabilities. Algorithms like A* or hybrid approaches combining multiple planning strategies could be utilised.

Scalability Enhancements: To scale the solutions for more complex tasks or larger environments, incorporating advanced machine learning techniques for real-time decision-making and more sophisticated environment mapping methods would be essential. Ensuring the system is modular and adaptable would facilitate easier scaling and integration of new functionalities.

Dynamic Rotation Strategy: Implementing a dynamic rotation strategy for room inspection would optimise the robot's operational efficiency. Currently, the robot's consistent anti-clockwise rotation may not always be the most effective approach in every scenario. By developing an adaptive rotation algorithm, the robot could determine the most efficient rotation direction based on each room's specific layout and obstacles. This flexibility would enhance navigation speed and reduce the time spent in obstacle negotiation, leading to more efficient task execution.

Reducing Time Delay: The robot is programmed to pause operation using time.sleep function upon reaching the entrance of Module 1 in the initial step of the solution. For efficiency reasons, it would be beneficial to implement a condition-based waiting system where the pause duration is dynamically adjusted based on real-time assessments of the robot's readiness and the callback value returned. This approach would prevent unnecessary delays and optimise the robot's task execution flow.