School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Group Report

Minoru Kishinami, Lai Ting Yeung, Michael Wiciak, Jack Jolley

COMP3631 Intelligent Systems and Robotics

Contents

Chapter 1

Design of Solutions

We decomposed the problem into more manageable chunks and integrated them at a later stage of the project. All the approaches we tried, the current approach, and the limitations and challenges faced are described below.

1.1 Window Detection

The robot sends every frame of its camera to window detection for verification. The identification of window frames within an image relies on several specific criteria to ensure accurate detection. Contours with areas below a predefined threshold are initially filtered out to eliminate noise or insignificant details. Window frames are assumed to have a rectangular shape, so contours with bounding rectangles below certain dimensions are excluded. Additionally, the aspect ratio of the bounding rectangle is considered, with contours possessing an aspect ratio greater than 0.5 being prioritised. Moreover, contours with an insufficient number of vertices are disregarded to ensure the presence of well-defined rectangular shapes. Another criterion involves evaluating the similarity between the area enclosed by the contour and the area of its bounding rectangle, aiming for a close match to reinforce the assumption of rectangularity. These criteria collectively serve to identify potential window frames, although it's important to acknowledge that they are based on assumptions about typical window frame characteristics and may not universally apply in all scenarios. Variations in image quality and perspective distortion could potentially lead to false positives or missed detections.

1.2 Planet Detection

Once we identified a window inside of a module, the following are the steps taken.

1.2.1 Stars Removal

To optimise the accuracy of the Hough Transform method for circle detection in low-resolution images, a preprocessing step was implemented. This involved the removal of stars from the images which could interfere with the detection process. The process begins by loading the image and applying a white mask to isolate regions of high intensity, indicating potential star locations. The white mask is generated by identifying pixels with RGB values falling within a specified range (200-255 for each channel). Subsequently, the resulting mask is applied to the image using a bitwise AND operation to retain only the white regions.

To enhance the visibility of the white regions and facilitate subsequent processing, the image is converted to greyscale. A binary threshold is then applied to create a binary image where white pixels represent the detected stars. To increase the size of these white regions and effectively remove the stars, dilation is performed using a circular structuring element with a kernel size of 40x40 pixels. This process expands the white regions, effectively covering and removing the stars from the image.

Finally, the processed image is utilized to create a mask for star removal. The mask is applied to a copy of the original image, effectively setting the pixel values of the detected stars to black (0,0,0),

thereby removing them from the image. This results in an image devoid of stars, ensuring improved accuracy in subsequent circle detection tasks, particularly in low-resolution images.

1.2.2 Planet Separation

The input image is initially converted to greyscale to simplify processing, followed by the application of Gaussian blur to reduce noise and enhance circle detection. Subsequently, the Hough Circle Transform algorithm is employed to detect circles in the image, based on gradient values and accumulator thresholds. Detected circles are then iterated over, and their coordinates and radii are used to create Planet objects representing the detected planets, which are stored in a list for further processing. If circles are found, they are drawn on the image for visualization, and the resulting image is saved. The function returns a list of Planet objects representing the detected planets. Additionally, after planet detection, the function isolates each planet from the original image using the generated Planet objects. This is accomplished by creating a mask around each detected planet and applying it to the image. The resulting cropped images of individual planets are then saved to a designated folder, enabling further analysis and processing of celestial objects.

1.2.3 Template Matching

Once the planets have been extracted from the image using circle coordinates and radii, the next step involves planet identification. Initially, we employed template matching techniques, comparing the extracted planet images with predefined templates for Earth and Moon. This method involves correlating the pixel intensities of the extracted images with those of the templates, producing correlation scores that indicate similarity. The confidence scores obtained from template matching are then compared against a predefined threshold to determine the planet's identity.

However, this approach faced challenges in accurately identifying planets under varying conditions, such as differences in brightness levels and image quality, and it is unrealistic to assume that the templates will be able to cover variations in the image quality. This is especially because of a problem when the planets could have different rotations, rendering this matching approach useless. To address these limitations, we transitioned to a machine learning (ML) approach for planet identification.

1.2.4 Usage of pre-trained CNN

In our pursuit of a more robust solution for planet identification, we opted to employ a pre-trained convolutional neural network (CNN) and adapt it to our specific task. We utilized a pre-trained CNN called MobileNetV2, a popular architecture known for its efficiency and effectiveness in various computer vision tasks. This pre-trained model serves as a feature extractor, capable of capturing meaningful features from input images.

To adapt the pre-trained model for planet detection, we fine-tuned it using a custom dataset consisting of images containing the Earth, the Moon, and other celestial bodies. The dataset was preprocessed and split into training, validation, and testing sets. We modified the classifier of the MobileNetV2 model to accommodate our classification task, replacing the original classifier with a new fully connected layer with the appropriate number of output classes.

During the training phase, we employed techniques such as data augmentation and regularization to prevent overfitting and improve generalization. The model was trained using the Adam optimizer with a learning rate of 0.001 and a weight decay of 1e-5. We monitored the training process, logging training

loss and evaluating validation loss and accuracy to ensure model performance. Upon completion of training, the model was evaluated on a separate test set to assess its performance on unseen data. The trained model achieved competitive performance, demonstrating high accuracy in classifying planets.

With the trained model in hand, we can now utilize it for planet identification by loading it and providing input images. This ML-based approach offers a more robust and adaptable solution compared to template matching methods, capable of accurately identifying planets across various conditions and image variations.

To enhance the robustness of our machine learning model and ensure sufficient data diversity, we employed a technique known as data augmentation. By applying variations to the existing dataset, we aimed to mimic real-world scenarios and introduce variability that the model might encounter during inference. Specifically, we augmented the dataset by pixelating, Gaussian blurring, and resizing the images, thereby creating variations that capture different aspects of real-world data.

Through data augmentation, we were able to generate over 50,000 distinct images from the original 1,434 images in the dataset. This significant increase in data volume enabled the model to learn from a more diverse range of examples, improving its ability to generalize and perform accurately on unseen data.

1.3 Correct Module Detection

Note that this uses the window detection from the previous chapter to identify where the green/red circle should be. Whether the windows has a green or red circle uses the code from the lab.

Moving to the right module involves a strategic approach to efficiently locate and assess modules for further exploration, as simplified in **??**. Initially, our approach was simplistic: heading towards the entrance of Module 1. However, this method lacked efficiency, as Module 1's entrance may not always be the closest. Our current approach addresses this limitation by navigating to the closest entrance between the two modules. While more efficient, this method relies on the speed of pose publishing.

Upon reaching a module's entrance, the robot executes a 360-degree rotation to assess the room's safety based on the circle's size. However, this method assumes several conditions: the presence of only two modules and clear visibility of the green/red circles from the entrance's coordinates, which may not always be the case.

This approach offers robustness as it can adapt to various scenarios, as indicated by the 360-degree rotation. However, it is not the most efficient due to the time-consuming rotation process. Moreover, its effectiveness is limited as the robot may misclassify rooms if another module appears closer due to its larger size.

To address misclassification, an alternative approach was considered: halting the rotation once a module is found and moving closer to it. If proximity to the module's centre is achieved, accurate classification can be ensured. However, this solution is hindered by the lack of constantly published pose information, limiting its efficiency improvement potential.

1.4 Finding windows in the correct module

1.4.1 Looking for windows

In our pursuit of an efficient window detection system within modules, we explored various methodologies to address the challenge. Initially, a simple rotational approach was tested by positioning the

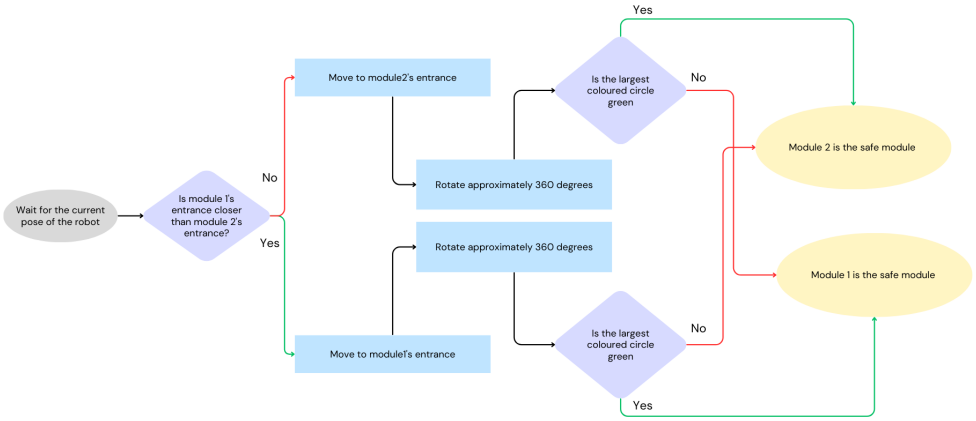


Figure 1.1: Flowchart of the process of finding the safe module.

robot at the module's centre. However, this method proved inadequate as it failed to detect windows obstructed by obstacles. Subsequently, we adopted a randomized sampling and rotational scanning technique as our primary strategy.

The current approach involves randomly selecting goals within the module's vicinity and directing the robot to these points. Upon reaching each goal, the robot executes a 360-degree rotation to scan for windows. This method offers resilience in navigating around obstacles through the ActionClient interface, ensuring adaptability to diverse environmental conditions. Nonetheless, it lacks asynchronous goal-sending capability, rendering the system unable to cancel actions or halt upon window detection. However, immediate cessation of actions is deemed non-essential given the nature of window detection. If the robot is unable to capture both the moon and the earth through a number of trials, the range from which goals are selected is enlarged, allowing the robot to navigate in a larger search space.

To enhance efficiency and image quality, a mechanism was implemented to verify the capture status of detected windows. If a window is identified but not previously captured, the robot adjusts its position for optimal capture without significant time overhead. Post-capture, the randomized sampling procedure continues to ensure continuous window detection. Internal flags track the capture status of Earth and Moon windows, halting motion once both are captured, assuming precise detection.

Challenges were encountered during attempts to navigate around obstacles, exacerbated by complexities in converting map files to usable coordinates. These challenges were further compounded by constraints imposed by the provided action client interface.

1.4.2 Moving to windows

In the process of implementing the movement towards windows, shown in 1.2 several key steps were involved. Initially, upon identifying a window, the robot accessed its X and Y coordinates within the ROS field of view. Leveraging this data, including the window's X, the camera's X (set at 960), and the field of view degree (approximately 62.200011761), the system computed the pixels per degree. From there, it determined the target offset X value by subtracting the camera's X centre from the window's target X value. Subsequently, the target offset degree value was calculated by dividing the target offset X value by the pixels per degree, providing the necessary angle of rotation for ROS to

align directly with the window.

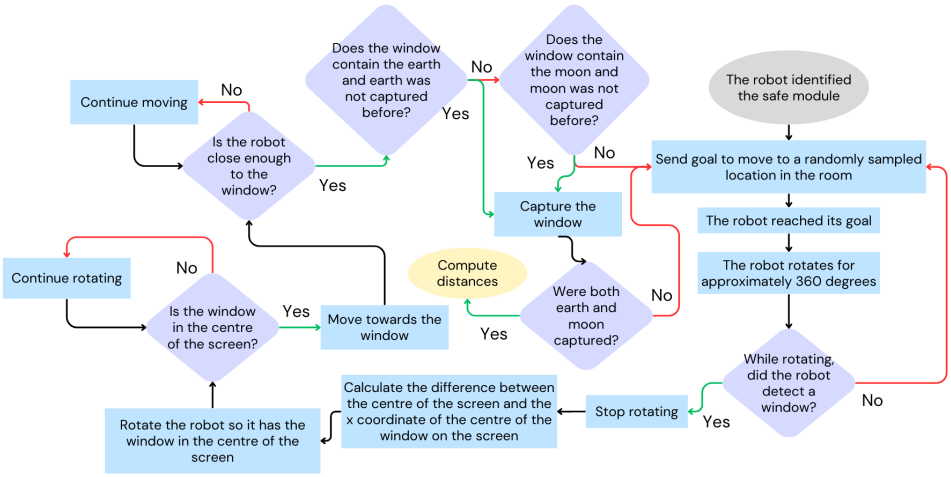


Figure 1.2: Flowchart of the process of finding windows without the time-out functionality.

Once the rotation angle was computed, the robot initiated rotation using a specific method. If the angle was negative, indicating a counterclockwise rotation was required, the robot executed accordingly. Conversely, for positive angles, indicating a clockwise rotation, the robot adjusted its movement accordingly. This design choice aimed to optimize the rotation process, minimizing unnecessary movement and ensuring the most efficient path to alignment. The calculated angular velocity was then published to ROS for execution.

However, immediate movement forward after rotation initiation posed challenges, as the rotation had not yet been completed. To address this, the system implemented functionality to ensure precise alignment with the window before proceeding. This involved periodic checks of the difference between ROS's orientation and the window's X coordinate, allowing for a permissible error range. Movement forward was only permitted once the desired angle alignment was achieved, ensuring accurate positioning before proceeding.

Upon achieving precise alignment with the window, the robot proceeded to move towards it for capture. The window's size served as the stopping distance, with the robot halting once the window size surpassed a predetermined threshold. Following capture, the robot's goal reverted to "Finding windows," allowing it to continue scanning for Earth and Moon windows.

To mitigate potential issues such as robot entrapment or prolonged capture times, a time-out functionality was integrated into the system. Various time-out values were tested across different rooms to ensure accuracy and efficiency, allowing the robot to continue its mission of window detection without delay.

1.4.3 Ensuring no duplicate windows captured

To ensure the exclusion of duplicate window captures, a technique leveraging image comparison was employed. This technique utilizes the Scale-Invariant Feature Transform (SIFT) algorithm to detect KeyPoints and descriptors within the images. By applying the Brute-Force Matcher (BFMatcher) algorithm, matches between the descriptors of two images are identified. A ratio test is then conducted to determine the similarity between the matched KeyPoints, with matches below a certain threshold

considered as potential duplicates. Specifically, matches where the distance of the closest keypoint is significantly smaller than that of the second-closest keypoint are categorized as "good" matches, while the rest are marked as "bad" matches. The ratio of good matches to bad matches is subsequently calculated to provide a quantitative measure of similarity between the images. This technique allows for the reliable identification and filtering of duplicate window captures based on their visual similarity, enhancing the accuracy and efficiency of the window detection process. However, this approach has limitations. Firstly, the performance of the SIFT algorithm can degrade in scenarios involving significant variations in scale, rotation, or illumination between images, potentially leading to mismatches or false positives. Additionally, the efficacy of the ratio test depends on the chosen threshold value, which may require manual tuning and could vary based on the specific dataset or application context. Furthermore, the computational cost of feature extraction and matching using SIFT can be relatively high, particularly for large datasets or real-time applications, potentially impacting the overall processing speed. Finally, the technique may struggle with images containing complex scenes or occlusions where the keypoint matching becomes challenging, reducing its robustness in such scenarios. However, most of these are not an issue during experimentation so we decided to stay with this technique.

1.5 Image Stitching

To stitch the images of the Earth and the Moon together, the function employs several key techniques. Initially, it detects KeyPoints and computes descriptors for both images using the Scale-Invariant Feature Transform (SIFT) algorithm. These KeyPoints and descriptors are then matched between the Earth and Moon images using a brute-force matching technique. Matches are sorted based on distance, and the top matches are selected for further processing.

Using the selected matches, a homography matrix is estimated using the Random Sample Consensus (RANSAC) algorithm. This matrix represents the transformation between the KeyPoints of the Earth and the Moon images, allowing for alignment and perspective correction. With the homography matrix obtained, the Moon image is warped to match the perspective of the Earth image using the `cv2.warpPerspective` function.

To blend the warped Moon image with the Earth image seamlessly, alpha blending is applied. This involves creating a new image large enough to accommodate both images, blending them together using a specified blending factor, and producing the final stitched panorama.

1.6 Distance Calculations

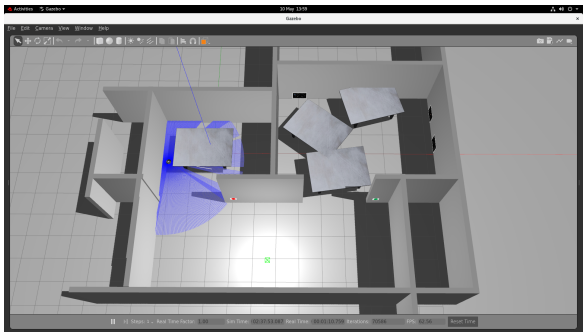
The distance computation relies on triangulation principles, utilizing the detected positions of Earth and the Moon in the captured images. Initially, the distance to Earth is determined based on the detected Earth object, while the distance to the Moon is calculated using the detected Moon object. These distances are then utilized to calculate the distance between Earth and the Moon. The approach follows a triangulation method, which assumes certain conditions such as a direct line of sight between the spacecraft and the celestial bodies, consistent object sizes, and a fixed perspective. However, it's essential to note that real-world scenarios may introduce complexities, such as partial obstructions and variations in the spacecraft's viewpoint, which can impact the accuracy of distance calculations. Therefore, while the method provides a simplified means of estimating distances, its applicability to real-world situations may be limited due to these inherent assumptions and potential complications.

Chapter 2

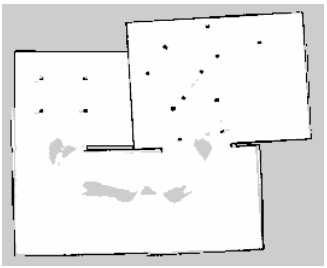
Implementation and Results

2.1 Performance on Gazebo Simulations

In the simulations conducted on Gazebo, the robot demonstrated remarkable efficiency and accuracy in various tasks. It efficiently navigated to different rooms using planning and searching methods, showcasing its ability to effectively traverse the simulated environment. Furthermore, the robot exhibited high accuracy in identifying module colours and windows, particularly distinguishing between Earth and Moon representations. An additional environment termed "Interesting", was introduced to provide further challenges. This environment was unique in that each module was approximately half the size of the provided maps, and it contained numerous tables that could disrupt motion planning. Despite the added complexity, the robot managed to navigate through the environment adeptly, showcasing robustness in handling obstacles. This environment placed significant stress on the motion planning algorithm, highlighting the need for resilience in real-world scenarios. With computer vision tested extensively in real-life scenarios, the focus shifted towards stressing motion planning capabilities, ensuring the robot's readiness for challenging environments.



(a) "Interesting" Environment in Gazebo



(b) .PGM file of the "Interesting" Environment

Table 2.1: Summary of the Performance during Simulations

World	Time To Complete	Did it Complete	How many windows it captured
Easy	4.25	yes	2 windows (1 Earth, 1 Moon)
Moderate	4.04	yes	2 windows (1 Earth, 1 Moon)
Hard	2.29	yes	2 windows (1 Earth, 1 Moon)
Interesting	2.47	yes	3 windows (1 Earth, 1 Mars, 1 Moon)

Table 2.2: Measurements.txt data for each world (km and 2 d.p.)

	Spacecraft to Earth distance	Spacecraft to Moon distance	Earth to Moon distance
Easy	93614.69	10425.00	93032.42
Moderate	92669.01	10341.27	92090.27
Hard	117618.46	10343.24	117162.79
Interesting	94579.79	12671.04	93727.17

In Table 2.1, Time To Complete is represented in the Minute:Second format.

Due to motion planning constraints, the robot may occasionally attempt to reach the same window multiple times. Although the likelihood of this occurrence is low, the robot is programmed not to capture repeated windows. However, it's intriguing to observe how this concept unfolds. This behaviour is illustrated in Table 2.4.

Table 2.3: ML Model Prediction Time

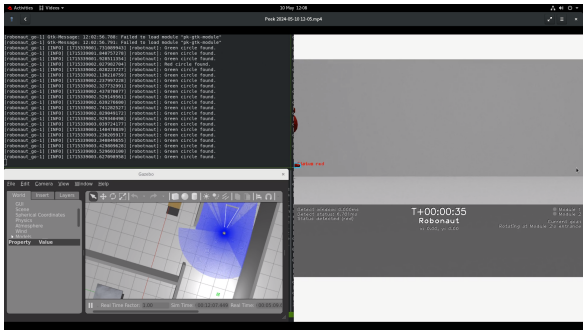
Statistic	Time (ms)
Average	2.3598
Median	2.1843
Max	150.1545
Min	1.8241

Table 2.4: Window Navigation Statistics

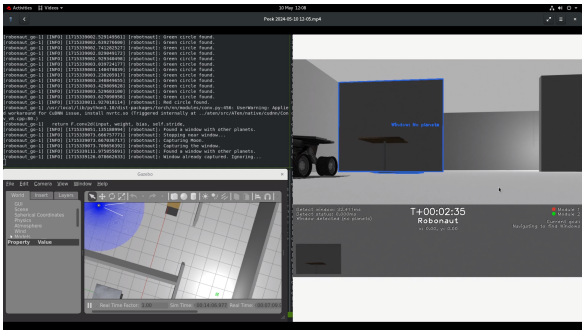
Environment	Total Windows Navigated	Repeated Windows
Easy	3	1
Moderate	2	0
Hard	2	0
Interesting	3	0

The performance on the ML was tested in Table 2.3. It was tested on 61,594 images that we data augmented for the ML to have more data.

2.2 Limitations and interesting observations from testing



(a) Detecting rectangular part of the fire hydrant as a module circle



(b) Small rectangular wall detected as window.

Despite the advancements in the solution, there are notable limitations that need to be addressed. One significant issue arises when the robot encounters locations on the map where it becomes stuck and disrupts the simulation, even with the utilization of the ActionClient for goal-setting. While the asynchronous goal setting approach could potentially resolve this issue, it significantly slows down the robot's performance, rendering it impractical for use. Thus, the chosen solution represents the best compromise given the available options. Another challenge lies in accurately determining whether a window has been captured solely based on motion planning and map details (coordinates). Although similarity detection is employed for this purpose, it is not flawless and requires parameter optimization, particularly when transitioning between real-world and simulated environments due to variations in camera quality. Other problems are shown in Figures 2.2a and 2.2b. These weren't an issue in the simulations but were bugs with the implementation of our design.

2.3 Link to the overall video

https://www.youtube.com/watch?v=UU7TQqW6gh0

Chapter 3

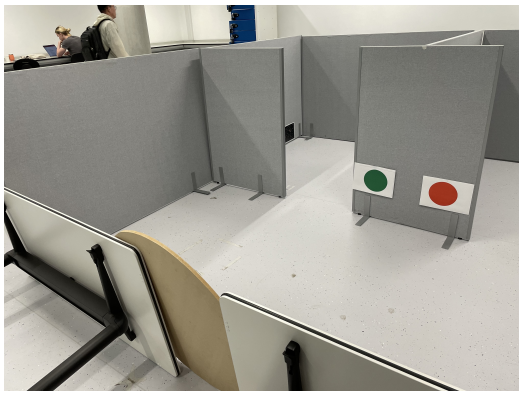
Real robot test

3.1 Required adaptation of the solution

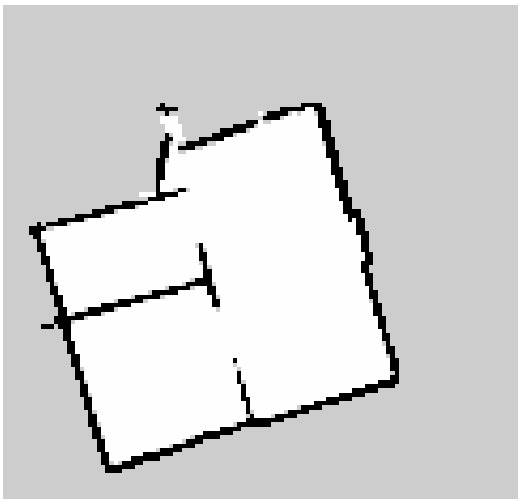
Adapting our robot algorithms for real-world testing involved addressing various challenges to ensure robust performance. Firstly, the robustness of computer vision algorithms was crucial considering factors such as varying brightness and shadows in the environment. This necessitated the design of machine learning models to handle the complexities present in real-world scenarios. Additionally, the presence of small handles on the walls posed challenges as the robot moved towards them, requiring padding to prevent collisions. Data augmentation techniques were employed to enhance the model's robustness against such variations in data. Shadows on the walls further complicate the detection process, necessitating adjustments in the algorithms. Moreover, the physical module's smaller size and lack of enclosing walls in the corridor introduced variations in green/red circle detection. Lower comparison thresholds were utilized due to the poor quality of images, and linear velocity was reduced to prevent unintended behaviours. The range of goals was adjusted accordingly, and timeouts were set to smaller intervals to accommodate the module's size constraints.

3.2 Performance and Results

The robot was tested on a real world environment that was created by us, trying to mimic the Gazebo Simulations provided. The environment created is shown in Figure 3.1a and 3.1b



(a) Picture of the Real World Environment



(b) PGM Map of the Real World Environment

The real simulation testing of the solution revealed several key observations. The robot successfully navigates to the closest module, identifies the correct (safe) module based on the detected circle, and proceeds to locate and approach the windows within each module to capture images. However, it encounters difficulty in accurately detecting the Earth's circle, often misclassifying it as "Other" instead. This issue stems from the low quality of the robot's camera, causing RGB values to be indistinguishable from the surrounding space. Example of such image is shown in Figure 3.3. As

a result, the machine learning model trained on images including real-world data fails to function effectively. Even a dedicated ML model trained solely on low-quality real-world data struggles due to subtle differences in brightness and background noise, leading to oscillating predictions between Earth, Moon, and Other classifications.

Despite this limitation, other components of the solution such as ignoring similar images and calculations performed for measurements.txt, function properly on the real robot. However, the detection of planets emerges as a bottleneck in the design. In hindsight, implementing more advanced training techniques and establishing a dedicated validation set using real-life photos could have ensured that the model captures all necessary aspects of the project requirements. This experience underscores the importance of thorough testing and fine-tuning, particularly when dealing with real-world scenarios and data of varying quality.

A bar chart of what the robot took time on when ran in the real world scenario is shown in Figure 3.3.

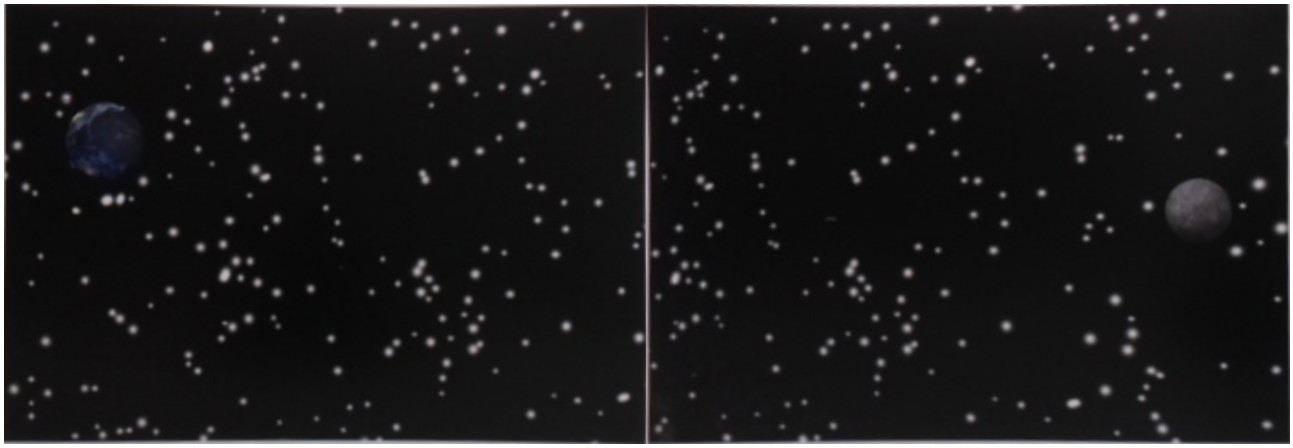


Figure 3.2: Capture of what the Robot Camera "sees" when it encounters a window with the Earth or the Moon.

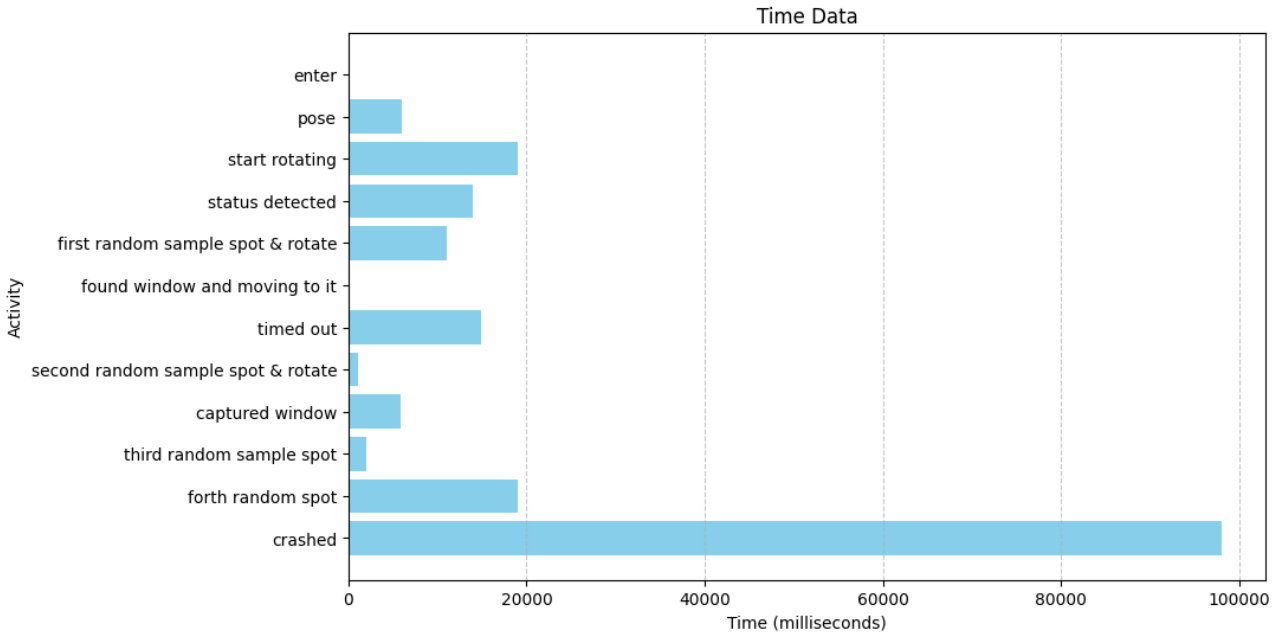


Figure 3.3: Time taken by the Robot for various tasks during the real world simulation.