MIRTE Master-based Laboratory Clean-up Robot

Authors: Matthew de Lannoy, Otto Meijerhof, Max van 't Hart, Florian van den Berg
Delft University of Technology

12th Jun, 2026

Abstract

In laboratories, items of many kinds may fall onto the floor. While implementations to clean such floors - like robotic vacuum cleaners - already exist, they do not work with trash-separation mechanics and end up disposing of all waste into the same bin. Yet, in laboratories, one may want specific types of dropped items to be returned and reused, such as electronic components. This project aims to use the open-source MIRTE Master robotic platform to combine both cleaning and reusing, by implementing a sorting system. This project presents an automated trash-detecting and sorting robot, using a 4-DOF robotic arm with gripping end-effector for grabbing, a 2-sided bin for sorting, an RGB-D camera for object-detection, MoveIt! 2 for manipulation and Nav2 combined with SLAM for navigation, built on top of the MIRTE Master platform. During this research, methods for path planning and object detection have been tested, compared and implemented with the goal of making the robot as accurate as possible while maintaining operational velocity.

Preface

This report marks the end of our Bachelor's degree at TU Delft. Over the past few months, our team has engaged in designing an autonomous mobile laboratory-cleaning robot with a waste-separation mechanism. This has been achieved by integrating a 4-DOF manipulator, an RGBD camera and a software stack based on ROS2. We were drawn to this project because of the potential for creative freedom it seemed to offer, given that the main goal wasn't set in stone. This provided us with a hands-on learning experience that combines the theoretical and physical sides of robotics. Having a tangible robotic prototype as an end result was, as expected, a good motivator to keep on working diligently on the project. During the preparation of this paper, ChatGPT and Gemini were used as a supporting tool for brainstorming, improving sentence structure, refining argumentation, debugging, ideating, and checking the clarity of written sections.

We would like to express our sincere gratitude towards our project supervisors, Martijn Wisse and Martin Klomp for their technical insights and feedback regarding the project. In addition, we would like to thank Arend-Jan van Hilten for helping us occasionally, despite not actually being involved in the project directly.

Contents

1 Introduction

In laboratory environments, maintaining a good level of cleanliness is often a challenge. While in modern domestic situations, an autonomous floor-cleaning robot is regularly used nowadays, in laboratory environments this is often not yet the case. In a laboratory, what actually falls onto the floor usually consists of graspable objects, while in domestic spaces these types of robots are generally made to remove dust and hair instead. Hence, a robot for laboratory-cleaning purposes should be able to grab objects instead of dust or hair. But as one may not always want to throw away all objects found in a laboratory setting, one would also want to separate trash from other fallen objects, such as electronics. Hence, the goal for this project was to develop an autonomous laboratory-cleaning robot that can detect, separate and dispose of two different types of objects, all based on the MIRTE Master robotic platform.

This report outlines the roadmap and methodology required to make such a robot.
Section 1 (Materials) includes information about the hardware and software used to realize the robot.
Section 2 (Theory) explains in detail how the theory behind the implemented ideas works.
Section 3 (System Overview) breaks down the implementation of all major aspects of the robot into the robot.
Section 4 (Experimental Methods) shows how the implementations are tested.
Section 5 (Results) covers the results of the tests that have been performed.
Section 6 (Conclusion), at the end, revisits the purpose of the project and reflects on that using the results that have been obtained and explained in Section 5 (Results).

1.1 Related Works

Previous work has shown that combining 3D perception with grasp-planning can allow robotic platforms to be able to efficiently manipulate objects in its environment [1]. This is relevant for our robot because we will be combining navigation, object detection, and manipulation for our lab-cleaning robot. Instead of general domestic service tasks, our robot will be combining this aspects for use in a laboratory cleanup scenario based on the MIRTE Master platform.

1.2 Contribution

This work serves as a practical application of the MIRTE Master platform used for testing the capabilities of MIRTE Master platform itself. By integrating perception, navigation, and object manipulation in a real-world setup, the project provides insight into the platform's practical strengths and limitations.

2 System Overview

The setup consists of a mobile manipulator (MIRTE Master) tasked with autonomously exploring an indoor laboratory environment, identifying and localizing objects, distinguishing between electronics and other objects and sorting these objects accordingly.

The main task of the robot can be broken down into sub-tasks, these then fit into specific niches of the entire system architecture:

1. **Motion planning**
2. **Perception**
3. **Navigation**

Figure 1 showcases this idea.

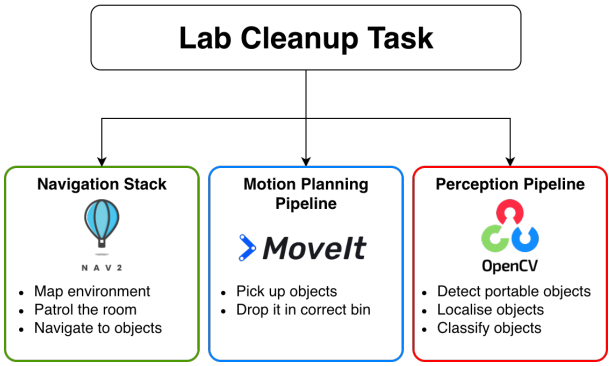

Figure 1: Three main tasks of the Robot

2.1 System Architecture

The tasks as described above are used as a guideline to construct the full system behaviour. The standard in robotics for such navigation and perception heavy tasks is to use a global behaviour tree that describes how the robot ought to behave in certain situations. This system level architecture as shown in Figure 23 is also used here to describe and execute the cleaning strategy. Unlike the packages described in the following subsections, a standard, widely adopted package for behaviour trees in robotics frameworks does not exist. A smaller, more niche python package, in the form of Py Trees for ROS is used [2].

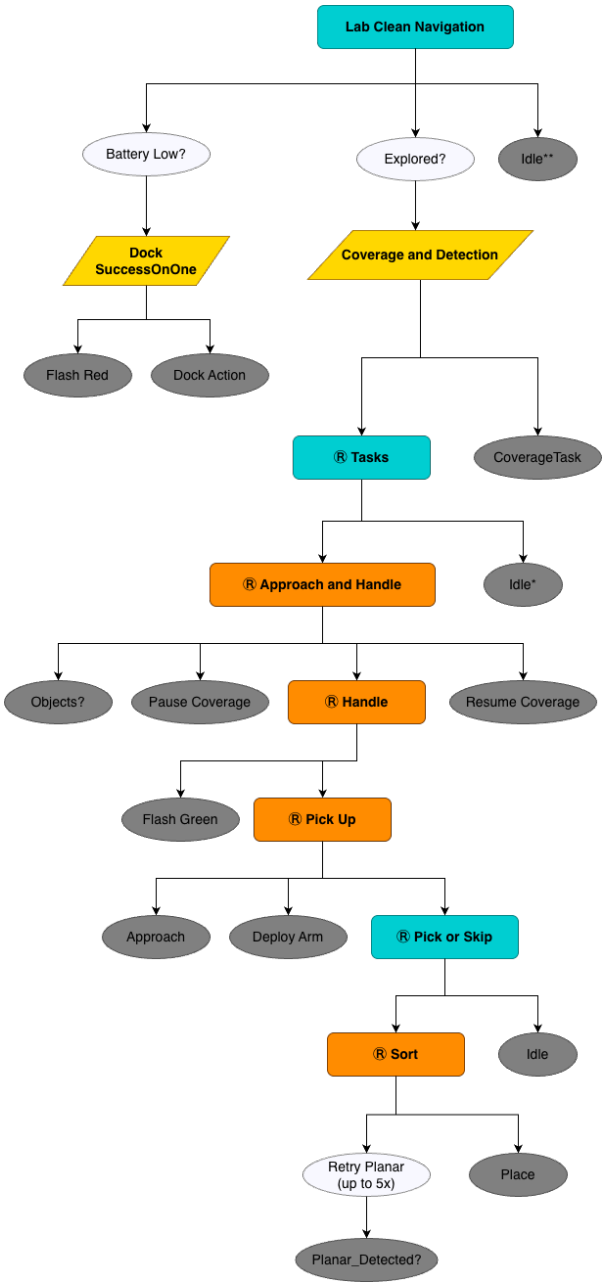

Figure 2: Behaviour Tree

The behaviour tree provides a hierarchical approach for coordinating navigation, perception and cleaning actions, cascading through the modules. It also can easily be visualised, simplifying debugging significantly. As can be seen from the tree, the robot initially covers the environment. Then, when an object is detected, the coverage navigation task is paused. Then the robot approaches, checks if the object can still be detected, picks it up and places it in the correct basket. Please view the documentation of the code base for a more detailed description of how the entire system runs. Pause handling is shown in Algorithm 3.

2.2 Mechanical

The CAD-model of this rendition of the MIRTE Master can be found using this link: CAD-Model.

2.2.1 Original MIRTE Master Platform

The project is based on the MIRTE Master V2. The platform consists of a mobile base propelled by four individually driven mecanum wheels and a robotic arm, allowing the robot to navigate through an environment and interact with objects.

The original MIRTE Master hardware consists of three main parts:

1. **Mobile base**
2. **Electronics**
3. **Robotic arm**

The mobile base utilizes four geared DC motors with Hall encoders, connected to 97 mm mecanum wheels. This allows the robot to move in multiple directions without having to turn first, which is useful for navigating in small indoor spaces. The base also contains various sensors, such as rearward facing ultrasonic sensors, an IMU, a 2D LiDAR and an RGBD (depth) camera mounted on the robotic arm.

The electronics of the original platform are based on an Orange Pi 3B and a Raspberry Pi Pico H. The Orange Pi functions as the main computer, while the Pico is used for lower-level control tasks such as controlling the motors and general interfacing. The MIRTE Master also includes a custom-made PCB to connect the motors, sensors, and power system in a structured manner. The main power source for the MIRTE Master is a 5Ah 12V Li-ion Parkside drill battery. No significant changes were made to the electronics used for this project.

The robotic arm is mounted on top of the mobile base and has four degrees of freedom, which are driven by four serial bus servo motors of varying strengths. A gripper and RGBD camera are mounted at the end of of the arm for object scanning and manipulation.

2.2.2 Hardware Components

The most important original hardware components can be seen in Materials.

These components provide a useful base platform for a mobile manipulation task. However, for this project, the original configuration needed some changes. The robot had to be adapted for a laboratory-cleanup task, where it should be able to move around in a laboratory environment, detect waste on the floor, and eventually collect and move objects into bins.

2.2.3 Design Goals for the Modified Robot

The goal of the hardware modifications was to make the MIRTE Master more suitable for collecting and transporting small waste objects. This required changes to the chassis and gripper design.

The main design goals were:

- Add space for collecting and separating waste.
- Mount the camera in a useful position for object recognition.
- Improve the battery mounting system.
- Improve cable management.
- Modify the gripper so it can pick up waste objects more reliably.

2.2.4 Chassis Modifications

The original MIRTE Master chassis was designed as a general-purpose mobile manipulator base. For this use case, the robot also needed to carry waste bins and support the manipulator during picking tasks. Therefore, the chassis was modified to improve strength and usability.

The chassis consists of six side panels and a top and a bottom plate. In the original model, the top and bottom plates were made out of clear PMMA. While PMMA allows the user to see the status of the electronics clearly by looking at the LEDs on the inside, it also has a tendency to shatter easily if bent too much. During early testing of the MIRTE Master it has already shown that the PMMA was not a good choice for any type of loading resulting in shattered PMMA. Therefore a change to the material of all plates from PMMA to aluminium with a similar stiffness was essential, the difference of this being that aluminium can bend more without shattering. To still be able to see the status of the electronics by the LEDs, the six side panel elements were 3D-printed using a slightly translucent material.

Two separate waste bins were added to the robot. These bins allow the robot to collect, store and separate objects after picking them up.

2.2.5 Gripper Modifications

The gripper is an important part of the laboratory-cleanup robot because it directly interacts with the objects that need to be collected. The original MIRTE Master gripper is useful as a general robotic gripper, but the cleanup task requires it to handle different small objects reliably.

The original gripper geometry is suitable for holding small and large objects, however, the gripping surface is not. Electronic trash tends to have small contact points for the gripper, so good gripping performance is a must-have. For improving the grip on objects, the contact patches are layered with a layer of soft 83A TPE, coated with an additional layer of Bison Rubber anti-slip coating.

During the initial testing phase of the RGBD camera, it was found that the mounting location for it was too low to the ground for it to be able to confidently display any accurate results regarding the distance towards objects. While this was the intended way for the camera to be implemented, for this project it was a better plan to mount this RGBD camera onto the end of the arm, right above the gripper. This gives it elevation which makes it able to look slightly more downward and therefore it could more accurately display distances. This also makes the need for the secondary RGB camera mounted onto the gripper insignificant, as the RGBD camera can also be used for object detection.

2.2.6 Current Design

The finalised design is available for viewing here.

2.3 Navigation

The navigation stack of this robot is responsible for two main tasks:

- Mapping
- Coverage

2.3.1 Mapping

Before the robot can navigate properly through the environment and ensure proper coverage, this environment must first be known. That is where mapping approaches can be helpful. Several advanced and specialised mapping approaches already exist, but in this paper only one was considered: 'frontier-based mapping'.

Unlike less advanced alternatives, frontier-based mapping allows the robot to start navigating the space and propagate its behaviour further down the tree as soon as the environment is mapped sufficiently. Once the environment is mapped sufficiently, the robot is allowed to navigate the space and propagate its behaviour further down the tree.

2.3.2 Coverage Planning Algorithms

This section presents the algorithms employed for the discovery of portable objects. While a comprehensive technical analysis of these algorithms is beyond the scope of this work, they are described with sufficient detail to enable understanding of their application within our framework. For in-depth technical details and formal derivations, the reader is referred to the original publications cited herein. To demonstrate the different navigators, a costmap of the Pulse Breakout room is used. This map was obtained during testing of the coverage methods. Furthermore Nav2 [3] is used for low level motion planning and execution as well as real-time control and costmap generation.

Pre-processing: Before the map can be used to plan a path, a common pre-processing algorithm can be abstracted. Coverage path planners generally use either a binary map of the free space, (free=1, occupied=0) or the polygons constructed from the boundaries of the free space.

The input to the algorithm can be any 2D image displaying the free and occupied pixels. In this case the input will be the costmap directly. First the costmap is thresholded to create a Figure 3, which is the first output.

Figure 3: Binary costmap generated by thresholding the Nav2 costmap into free and occupied space.

Then, using this binary costmap, the contours in Figure 4 are extracted.

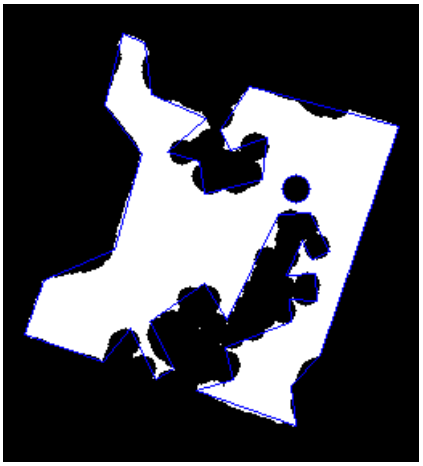

Figure 4: Contours extracted from the binary costmap representing the boundaries of free space.

Morphology-Based Skeleton Path Planner: The first implemented method, based on the work of [4], [5], derives a coverage path from the morphological skeleton of the free space. The algorithm is explained in Algorithm 1.

The free space is extracted using the methods discussed in the previous section. The polygons are used and converted to a binary map as to not produce too many branches for the robot to navigate through. This map is then skeletonised to extract waypoints, see Figure 5.

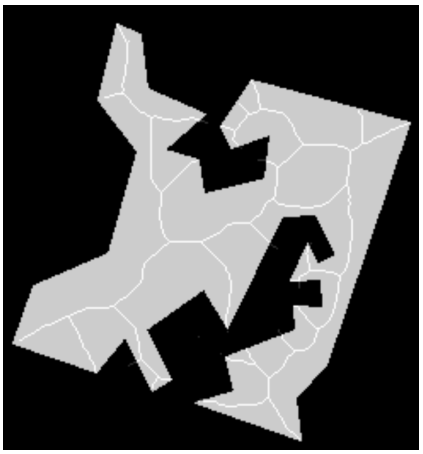

Figure 5: Morphological skeleton extracted from the free-space representation of the environment.

Once the waypoints are obtained, a graph is constructed and leaf nodes are extracted. In the graph leaf nodes are defined by their single connection to the rest of the skeleton, these are seen as endpoints of the path.

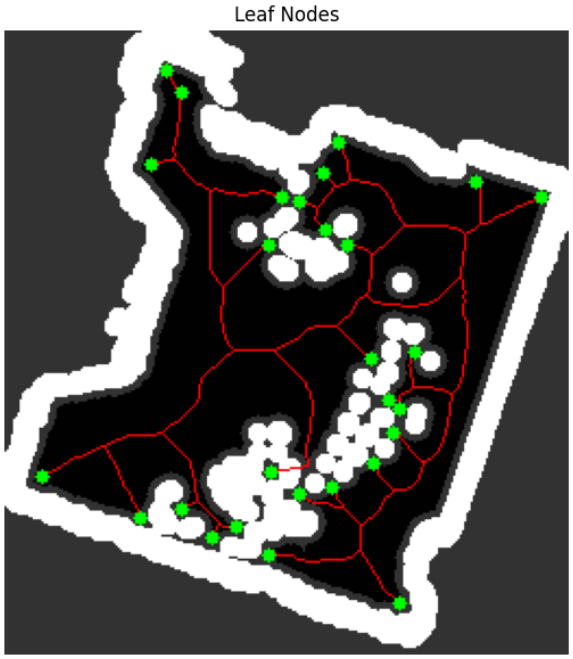

Figure 6: Leaf nodes identified on the skeleton graph, representing endpoints used during path generation.

Finally, starting from the closest leaf node to the robot, the full path is planned by recursively tracing the graph from the current leaf node to the nearest unvisited leaf node, following a nearest-neighbor heuristic [6]. This can be seen in Figure 7. A visualization of how the path is traced can be viewed here.

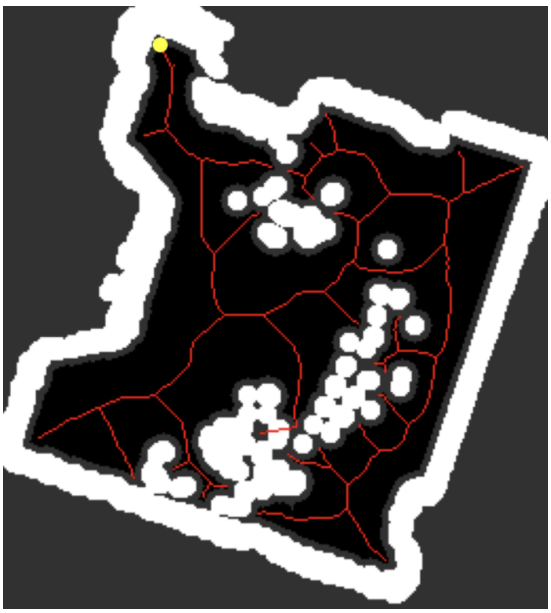

Figure 7: Fully planned path.

Grid-Based Spanning Tree Planner: Secondly, a grid-based spanning tree algorithm is used to plan a full coverage path over the map. This planner is based on the work of [7].

As opposed to the Skeleton planner, this planner makes use of the binary costmap directly. Using this binary costmap, it is further discretised to a grid which is the fraction of the resolution of the original costmap, see Figure 8.

Figure 8: Downsampled free-space grid used to construct the spanning tree graph.

A degree-first search (DFS) is conducted over the free cells, from which sparse waypoints are created. This can be seen in Figure 9.

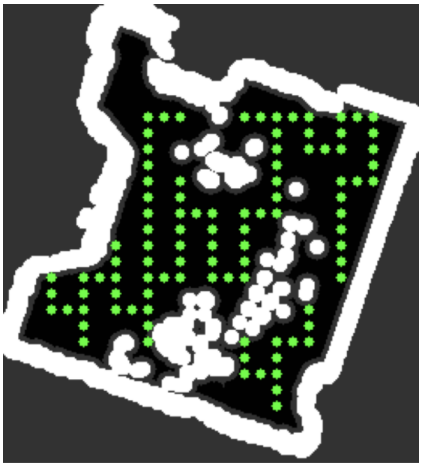

Figure 9: Depth-first search spanning tree generated over the free-space grid.

A black image, of the same resolution as the original costmap is then created and white squares, with half the cell width, are placed in the center of each cell and in between each cell. Waypoints are created from the contours of this tree. These waypoints represent a circumnavigation around the DFS tree.

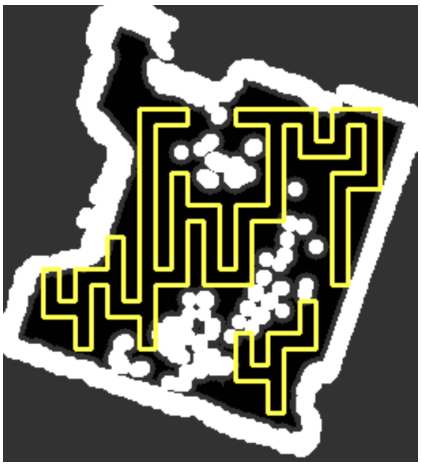

Figure 10: Coverage waypoints generated by tracing the contour around the spanning tree structure.

A segment from this circumnavigation is then removed from the path as can be seen in Figure 11. This entire process is described in Algorithm 2.

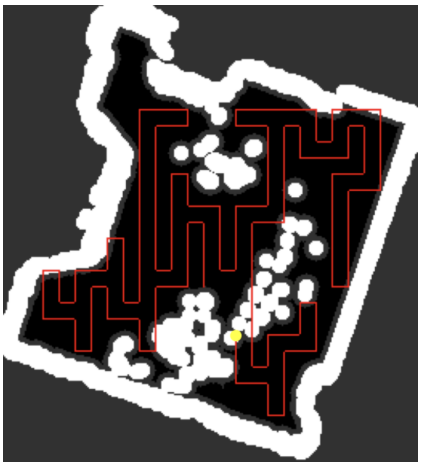

Figure 11: The path with circumnavigation removed.

A lab cleanup robot must not blindly follow waypoints. If it does it might collide with obstacles near the map boundary or collide with moving objects.
Nav2 implements several low level real-time trajectory planners that plan inside the live updating costmap [8]. They handle how the waypoints are followed and how much influence the map ought to have on the generated trajectory. The system described in this section implements an MPPI controller [9] due to its superior performance in highly dynamic environments [10], like labs where students or people frequently walk. Other planners that were considered include DWB (Dynamic Window Approach Behaviour-based) [11] and RPP (Regulated Pure Pursuit) [12].

2.4 Perception

Perception is responsible for object detection, classification and spatial localisation. For this study the object detection and localisation approaches were limited to two. A **2D detection and depth projection** approach and 3D segmentation with **point cloud clustering**. As for classification methods, this study evaluates a classical cv approach, as well as a classification model based approach.

Originally, objects were categorised into:

1. Graspable vs non-graspable
2. Electronics vs non-electronics

However, due to the limitations of the hardware of the standard MIRTE Master package (relatively low-quality camera and limited computing power) the scope got reduced to:

1. Graspable vs non-graspable
2. Colourful vs greyscale

2.4.1 2D Perception

For determining the type, and possibly the distance of objects, a YOLO26 ('You Only Look Once-26') object classification model was used. A dataset was trained by recording videos using the USB camera included in the MIRTE Master hardware package, of which one in every fifteen frames got extracted to use for the training dataset. A figure of four frames in different environments is shown below in Figure 12.

Figure 12: Picture of four individual frames.

Why YOLO26 was used: The YOLO object-detection family is well established, with YOLO26 [13] being the most recent generation. YOLO26 was chosen because the robot has to detect and classify objects from a live camera on the MIRTE Master platform, which has limited processing power as it uses an OrangePi 3B which has limited computational capabilities. Object detection with limited processing power requires a lightweight model with fast inference times while still maintaining accuracy. YOLO26 is suitable for this because it was designed with deployment on low-power devices in mind. It changed the post-processing pipeline to NMS-free inference, simplified bounding-box predictions and supports deployment formats commonly used for low-power devices [14].

In this case, the RK3566 NPU of the OrangePi 3B can be used for improving the performance, reducing the total inference time [15]. For this model, the Nano variant of YOLO26 was chosen to minimize inference times. Another benefit of the newer YOLO26 variant over the older variants is that YOLO26 allows for oriented bounding boxes. Unlike standard bounding boxes, oriented bounding boxes also estimate the rotation of recognised objects [16]. This could be useful for complex grasping tasks where orientation of the gripper is crucial. However, this project only uses standard object detection.

Training: Training was done manually using Label Studio. Initially, roughly 280 frames were captured in four different environments. Due to a large amount of blurring within the images, about 100 pictures were deleted. Of the remaining 180 pictures, 60 pictures were imported into Label Studio and labelled per colour: green, red, purple, white, light-grey, dark-grey and black. In total, 1206 bounding boxes were drawn. A test of the perception model performed during training is shown in the figure below.

Figure 13: The tests performed during training.

The curves in the figure below show that the model converged during training. This means that the model actually learns while training instead of making random or unstable decisions.

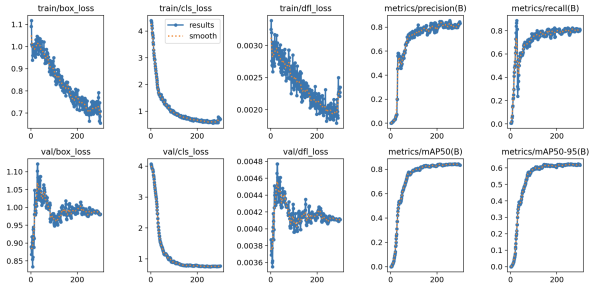

Figure 14: Plots of training results per epoch.

The normalised confusion matrix shows how well classes are recognised by the model. As shown in the figure shown below, most classes are classified correctly, as indicated by the strong diagonal values. However, light-grey objects were among the least correctly identified objects, this is a result of exposure and lighting changing enough to where the model will classify them as background. Some of objects were missed as background, a small amount of objects were misclassified, and some objects like dark-grey and white were confused with light-grey.

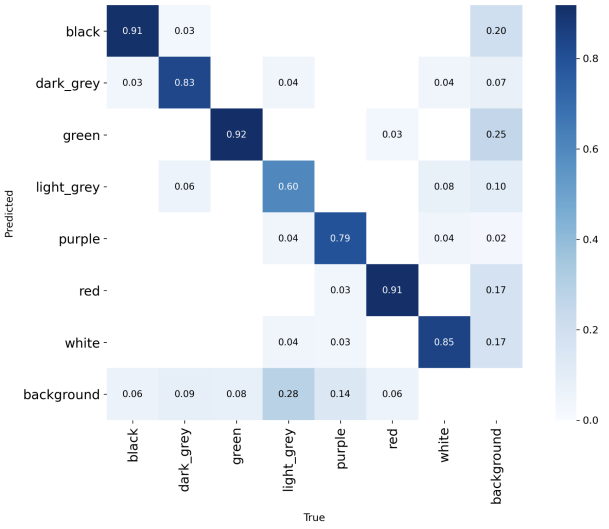

Figure 15: Normalised confusion matrix.

A figure of the F1 confidence curve is shown below. This curve shows how the F1 metric, which combines precision (how many objects were correctly predicted) and recall (how many objects were correctly found), changes with the confidence threshold of the model. The curve shows that a good starting confidence threshold would be 0.3-0.35.

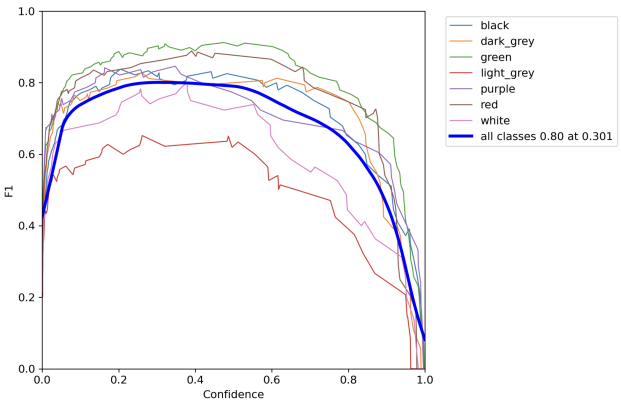

Figure 16: BoxF1 curve.

2.4.2 3D Perception

Point Cloud-based Object localisation: To determine the exact location of graspable objects around the MIRTE Master, the data gathered and transmitted by the camera must be interpreted. One effective method for achieving this is through the use of point clouds.

As the name suggests, point clouds are collections of points that represent a three-dimensional space. These points may contain a variety of information, but the most important characteristic of a point during this project is its position, represented by its x, y, and z coordinates. During this project, the `Open3D` library was used to process the point cloud. This particular library has been selected because of its user-friendly API and popularity over PCL [17].

Detecting planes will be done using the Random Sample Consensus model, or RANSAC for short. This algorithm picks a number of points from the point cloud at random and fits a plane through them. It then evaluates which points in the point cloud are within the given distance from the plane [18]. With this method, point clouds can easily and quickly be filtered such that only the important parts, namely the clusters of points representing small objects on the ground, remain.

Next, an algorithm for discovering clusters is used called DBSCAN (Density-Based Spatial Clustering of Applications with Noise). The model evaluates the remaining points in the point cloud and determines the cluster to which they belong [19]. Afterwards, the identified clusters can be converted into bounding boxes, deciding an object's position and orientation, as well as its dimensions.

Below, Figure 17 and Figure 18 show the input to output the constructed point cloud pipeline.

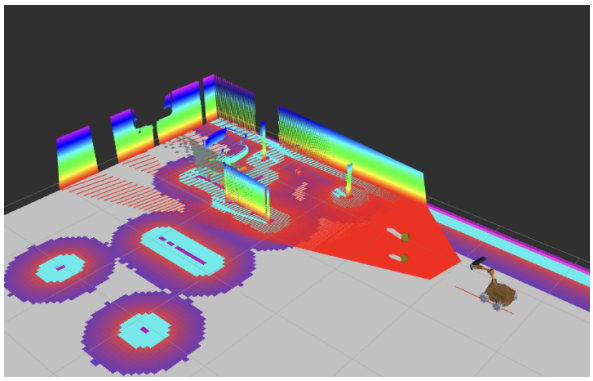

Figure 17: The point cloud as received from the camera topic.

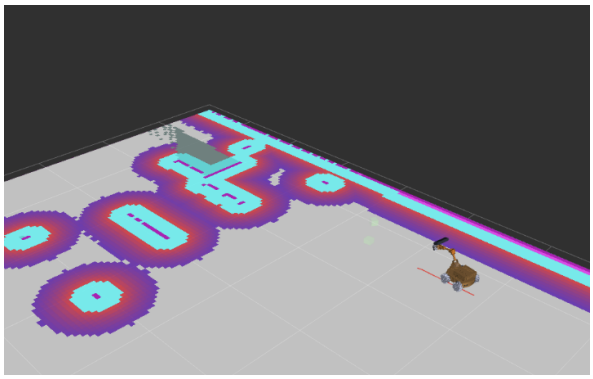

Figure 18: The resulting bounding boxes.

Any objects that are in the costmap are excluded, resulting in only these bounding boxes remaining.

3 Methodology

This section describes the transition from the simulation-based implementation to real-world testing on the MIRTE Master platform and the practical requirements to do so.

3.1 From Simulation to Real-World Application

For the transition from simulation towards real-world testing, the codebase needed to be changed. The launch files were changed so that Gazebo was no longer used and `use_sim_time` parameters were set to `false`. To reduce the risk of collisions, the inflation radius and the robot radius in the Nav2 parameters were increased. Another change was the migration of processing tasks from the Orange Pi 3B to a laptop connected to the MIRTE.

Finally, all the sensors were tested and it was concluded that the depth camera was not positioned correctly and thus its position changed from the front of the robot to the gripper, allowing its position and viewing angle to be adjusted.

3.2 Choice of testing environment

For testing, an open, medium-sized room was chosen. The room needed to have sufficiently large open spaces for the robot to be able to correctly plan its trajectories for the object detection phase. Due to the use of mecanum wheels and camera-based object detection, the room also needed to have a mostly-smooth, flat floor with a uniform colour.

3.3 Choice of Testing Objects

Due to low video output quality of the included USB camera module, including significant motion blur and difficulties with exposure under different lighting conditions, the success rate of recognition and classification of the electronics was too low.

Additionally, most electronics waste had too low of a profile to be recognised by the depth camera. As a result, it was chosen to 3D-print coloured shapes with a textured outer wall for grip. A custom dataset was created for the object classification model to improve object detection performance.

4 Results

This section covers the observed characteristics of the implemented coverage planners, perception methods and solutions regarding manipulation.

4.1 Coverage Planners

Three path planners have been compared:

- Boustrophedon Coverage
- Morphology-based Skeleton Coverage
- Spanning Tree coverage

The Boustrophedon coverage planner package had compatibility issues with other packages used for navigation, this resulted in Boustrophedon not working on the MIRTE Master and it could therefore not be tested.

The morphology-based skeleton coverage planner performed well in laboratory-like environments but worse in larger open spaces. This planner allowed the robot to maneuver efficiently between objects such as tables and chairs while still being able to get into tight places.

The morphology-based skeleton coverage planner also resulted in the least amount of sharp turns which, due to the simple nature of the MIRTE Master odometry (wheel encoders only), helped the robot not to lose track of where it was and thus keeping the map clean. However, the skeleton planner performed slightly worse in terms of coverage performance. This resulted from the lower density of the planned path, which resulted in the robot sometimes missing the edges of larger free spaces that needs to be scanned for objects.

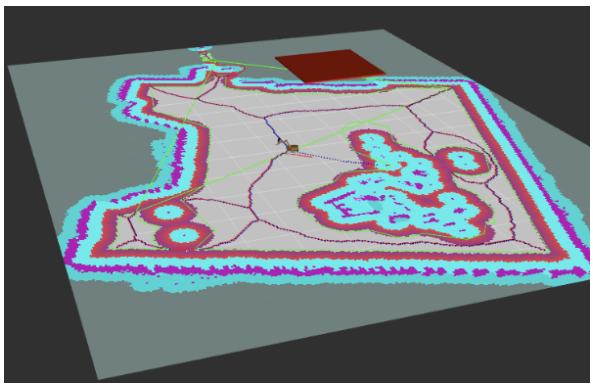

Figure 19: Skeleton path in laboratory environment.

The spanning tree coverage planner performed well in large open spaces, but unlike the morphology-based skeleton coverage planner, it struggled more with planning paths in tighter spaces such as laboratory environments (see Figure 20), resulting in separate path loops being stitched together and possibly missing tighter spots between the loops. Also, due to the nature of the coverage pattern, a lot of smaller radius turns are generated which resulted in the robot sometimes slowly accumulating drift in the odometry, resulting in less accurate localisation and a less accurate map.

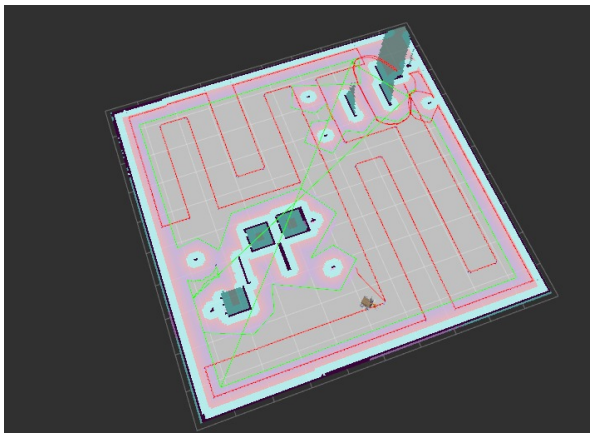

Figure 20: Spanning tree path in laboratory environment.

4.2 Object Localization and Classification

For object detection, a localization and classification method was developed. Objects are represented by bounding boxes that are constructed by means of point clouds, which they themselves are made with the RGBD camera's depth images. This particular object detector was able to detect objects sized 1*cm* to 6*cm*. Figure 21 below shows a point cloud and the corresponding bounding boxes constructed within. The custom-made YOLO26 object detection model worked as intended, being able to classify the localised objects correctly and consistently. Figure 21 also shows this behaviour in action.

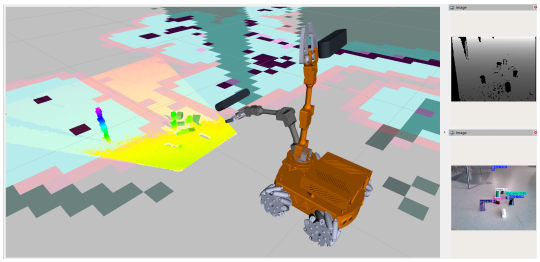

Figure 21: Bounding box drawn around found objects (center) and YOLO26 working simultaneously (bottom right).

4.3 Manipulation

Movement accuracy The manipulation of the actuator has seen major improvements regarding movement accuracy. The initial accuracy of the actuator was approximately 15*cm*. After reworking the code, the actuator was controlled by a function that did not rely on joint approximation. This reduced the error significantly, resulting in a movement error of just a couple millimetres.

Gripping Performance The gripping performance of the original robot was slightly increased by adding rubberised patches to gripping surfaces to the gripper. This has helped gripping the smooth, plastic testing objects when the gripper approached the objects at a sub-optimal angle of approach.

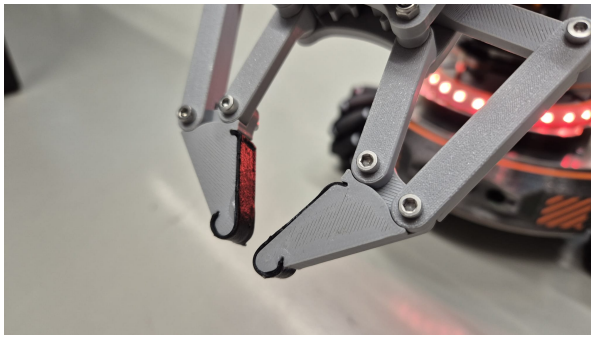

Figure 22: Rubberised gripping surfaces.

5 Conclusion

This work has demonstrated that the different coverage planners each have their strengths and weaknesses: The morphology-based skeleton coverage planner excelled in cluttered environments with tight spaces, but the resulting path did not provide complete coverage of the free space in larger rooms. conversely, the spanning tree coverage planner achieved better coverage at the expense a large number of sharp turns, reducing localisation accuracy.

The developed perception pipeline successfully localised and classified small objects, demonstrating that this combination is an effective solution for object recognition for the MIRTE Master.

The modifications to the manipulator have increased its accuracy nearly two orders of magnitude, as the system no longer relies on approximate joint values. This together with the improved gripper surface made it so objects are graspable even at odd angles.

Overall, this work has successfully demonstrated the MIRTE Master can accommodate all components necessary to realise a laboratory-cleaning robot, and is therefore a viable platform for performing this task.

6 Discussion

6.1 Delays

We were caught in the middle of a platform migration to a new design. This migration happened at the time that a MSc course started which used up all of the MIRTE Master robots. This meant that, for us, no robot was available from the start of our project. We also had no clarity about when a unit would be received and if more components needed to be ordered. Furthermore, not all components in the MIRTE Master could be ordered given the BEP budget. This meant there was an unanticipated waiting period that extended far into the project timeline, during which testing and integration was supposed to have happened. In the end, the components were received in week 13 of the 16-week long project timeline. While the software team was able to frontload the majority of software development using Gazebo simulations, this still meant that the team only had three weeks to find out all the quirks of the MIRTE Master, troubleshoot them and simultaneously integrate the software which severely limited our ability to perform real-world quantitative and qualitative experiments.

6.2 Gazebo

Gazebo was a great tool for testing the software quickly and in an isolated manner. It also provided a way to run the programs without needing a real robot. However, learning to work with the simulator not only had a steep learning curve, it also required a lot of additional effort to set up a realistic environment: making custom worlds using Blender with objects that could be interacted with. It also required a further setup to be compatible with the existing MIRTE framework.

When transitioning from a simulated environment to the real world, many issues arose. Firstly, every ROS node dependent on up-to-date transforms and other timestamped messages now needed to be reconfigured not to use the simulated clock (which starts at zero when initialized). Secondly, the team discovered that the robot sensors produce significantly less accurate, up-to-date, and reliable feedback than the ones configured for the simulated MIRTE.

6.3 The MIRTE Master Platform

The MIRTE Master proved to be a viable starting platform for prototyping a lab-floor cleaning robot. A lot of aspects came pre-configured: MoveIt configuration was largely set up, Nav2 parameters for MIRTE existed, a selection of configuration files for the robot were by and large integrated, operational and all the calibration and control functioned.

However, it seemed to us that some parts of the design did not seem to be thoroughly tested prior to integration within the MIRTE Master platform. The most significant friction occurred with the components used for perception. Firstly, the RGBD camera was mounted in the front, near the bottom of the MIRTE Master which resulted in the depth camera not being able to create a usable depth image for the point cloud. This issue was discovered by using the OrbbecViewer tool and testing for different mounting positions. From our testing, we concluded that the depth camera should never have been mounted low to the ground. Secondly, the included USB camera module was excluded from our final design because of poor performance: the camera was too blurry for use while moving and the frame rate dropped severely in different lighting and background conditions due to its automatic exposure control.

The final major issue was time-synchronization between the MIRTE Master and other machines. It was discovered quite quickly that the MIRTE Master's OrangePi 3B was not up to the task of running the software stack on its own, so a lot of the processing (lab-cleanup stack and point cloud creation) was outsourced to a laptop. For this to work correctly, time needed to be synchronised between the two machines. The MIRTE Master does not possess an accurate clock and when a laptop is directly connected to the MIRTE Master, no internet connection is available. This resulted in attempts to manually sync the clocks between devices, which did not have a high success-rate. Later, Chrony was implemented as a more permanent solution which automatically synced the clock between devices within tolerances. This almost eliminated clock synchronization issues, but some errors still persisted.

6.4 Future Work

Using the software packages created as a part of this work, future research can be done to evaluate the performance of the robot by testing the different coverage planners in isolation and in combination with the rest of the system. Additionally, other coverage planners, such as Voronoi-based [20] and boustrophedon [21] coverage planners, can be implemented and tested. It can also be interesting to observe the influence of different environments on the performance of the system. One could also research if porting the MIRTE Master software stack to a more suitable SBC is possible.

7 Appendix

7.1 Algorithms

Algorithm 1: Skeleton-Based Coverage Path Planning

Input: Occupancy grid $\mathcal{M}$, robot start position $\mathbf{p}_0 \in \mathbb{R}^2$
Output: Ordered list of waypoint segments $\mathcal{P} = [P_1, P_2, \ldots, P_k]$

```
/* Map preprocessing                                                    */
B ← threshold(M, lethal_threshold) ;                  // binary free-space mask
C ← extractContours(B) ;                              // OpenCV RETR_TREE
G_poly ← groupContoursByContainment(C) ;              // outer + holes
foreach polygon group g ∈ G_poly do
    S ← medialAxis(B) ;                               // skimage skeletonize
    W ← skeletonToWorldCoords(S);
    G_graph ← buildKDTreeGraph(W, radius = r_path);
    bridgeDisconnectedComponents(G_graph);
    L ← findLeafNodes(G_graph) ;                      // nodes with degree 1
    v ← nearestLeaf(p_0, L);
    Q ← [v],  V_visited ← ∅;
    while V_visited ≠ L do
        V_visited ← V_visited ∪ {v};
        r ← nearestLeafByGraphDistance(v, L \ V_visited, G_graph);
        π ← shortestPath(v, r, G_graph) ;                          // Dijkstra
        append π to Q;
        v ← r;
    P_i ← [W[u] ∀ u ∈ Q];
    append P_i to P;
    p_0 ← P_i[last] ;                                 // chain segments
sanitiseWaypoints(P, M) ;                             // clamp to costmap bounds
return P
```

Algorithm 2: Spanning-Tree Coverage Path Planning

Input: Occupancy grid $\mathcal{M}$, robot start position $\mathbf{p}_0 \in \mathbb{R}^2$, scale factor $s = 0.06$
Output: Ordered list of waypoint segments $\mathcal{P} = [P_1, P_2, \ldots, P_k]$

```
/* Map preprocessing                                                    */
B ← threshold(M, lethal_threshold) ;                  // binary free-space mask
B_down ← resize(B, scale = s, interpolation = NEAREST) ;        // downsample
/* Build spanning tree over free cells                                  */
G ← gridGraph(B_down) ;                               // 4-connected grid graph
foreach edge (u, v) ∈ G do
    if B_down[u] = 0 or B_down[v] = 0 then
        remove (u, v) from G;
T ← ∅;
foreach connected component C ∈ G do
    T ← T ∪ DFS_tree(C);
/* Edge subdivision | insert midpoint nodes                             */
foreach edge (u, v) ∈ T do
    m ← (u+v)/2;
    replace (u, v) with (u, m) and (m, v);
/* Generate circumnavigation contours                                   */
P_perim ← 0^{H×W} ;                                   // blank perimeter map
r ← ⌊1/(4s)⌋ ;                                // rectangle half-size in pixels
foreach node p ∈ T do
    p_img ← (p+0.5)/s ;                          // scale back to image coords
    drawFilledRect(P_perim, centre = p_img, halfsize = r);
C ← findContours(P_perim, RETR_EXTERNAL);
/* Convert contours to world-frame waypoints and plan                   */
P ← [ ];
foreach contour c ∈ C do
    W ← pixelToWorld(c);
    if |W| < 2 then
        continue;
    idx ← arg min_i ‖W[i] − p_0‖_2 ;                  // start from nearest point
    P_i ← [W[idx], W[idx + 1], ..., W[−1], W[0], ..., W[idx − 1]];
    append P_i to P;
    p_0 ← P_i[last];
sanitiseWaypoints(P, M);
return P
```

Algorithm 3: Coverage Task Execution with Pause/Resume

Input: Coverage segments $\mathcal{P}$, planner type τ
Output: Task result (success / cancelled / stopped)
$\mathcal{P} \leftarrow$ sortByLength($\mathcal{P}$, descending);
$n \leftarrow |\mathcal{P}|,\ i \leftarrow 0$;
while $\mathcal{P} \neq \emptyset$ **do**
  $P_i \leftarrow$ dequeue($\mathcal{P}$), $i \leftarrow i + 1$;
  nav2.goThroughPoses(toROSPath(P_i));
  repeat
    $f \leftarrow$ nav2.getFeedback();
    publishFeedback(i, n, f.remainingPoses);
    if *cancelRequested* **then**
      nav2.cancelTask();
      return CANCELLED;
    if *stopRequested* **then**
      nav2.cancelTask();
      return STOPPED;
    if *pauseRequested* **then**
      nav2.cancelTask();
      $P_{rem} \leftarrow$ computeRemainingSegment(P_i, getRobotPos());
      prepend P_{rem} to $\mathcal{P}$; `// resume from current pose`
      repeat
        spinOnce()
      until *not* *pauseRequested*;
      ;
      nav2.goThroughPoses(toROSPath(P_{rem}));
  until *nav2.isTaskComplete()*;
  publishFeedback(i, n, 0);
return SUCCESS

Algorithm 4: Depth-Based Object Detection

Input: Point cloud $\mathcal{C}$ (camera frame), costmap $\mathcal{M}$, TF tree
Output: Detected object array $\mathcal{O}$
$\mathcal{C} \leftarrow$ removeNaN($\mathcal{C}$);
$\mathcal{C}_{LQ} \leftarrow \mathcal{C},\ mask \leftarrow \mathbf{1}^{|\mathcal{C}|}$;
`/* Iterative RANSAC plane removal */`
repeat
  $(\mathbf{n}, d) \leftarrow$ RANSAC_segmentPlane($\mathcal{C}_{LQ}$,;
    $\epsilon = 0.003$,;
  $n_{iter} = 2000$);
  $\mathcal{C}_{LQ} \leftarrow \mathcal{C}_{LQ} \setminus$ inliers;
  $mask \leftarrow mask \ \wedge\ (\text{dist}(\mathcal{C}, \Pi) > 0.008)$;
until $|\mathcal{C}_{LQ}| < 100$ ***or*** $|inliers| < 500$;
$\mathcal{C}_{obj} \leftarrow \mathcal{C}[mask]$;
`/* Coordinate transform to map frame */`
$T_{cam \to base} \leftarrow$ TF.lookup(base_link, camera_frame);
$T_{base \to map} \leftarrow$ TF.lookup(map, base_link);
$\mathcal{C}_{map} \leftarrow T_{base \to map} \cdot T_{cam \to base} \cdot \mathcal{C}_{obj}$;
`/* Costmap filtering */`
$\mathcal{C}_{map} \leftarrow \{\mathbf{p} \in \mathcal{C}_{map} : \mathcal{M}(\mathbf{p}) < \delta_{occ}\}$;
`/* Clustering and bounding box extraction */`
$L \leftarrow$ DBSCAN($\mathcal{C}_{map}$, $\varepsilon = 0.03$, minPts= 20);
$\mathcal{O} \leftarrow [\,]$;
foreach *cluster* $k \in L,\ k \neq -1$ **do**
  $OBB_k \leftarrow$ getOrientedBoundingBox($\mathcal{C}_{map}[L{=}k]$);
  if $\mathcal{M}(OBB_k.center) < \delta_{occ}$ **then**
    $\psi \leftarrow$ arctan2($R_k[1,0],\ R_k[0,0]$) ; `// yaw only`
    append DetectedObject(OBB_k.center, OBB_k.extent, ψ) to $\mathcal{O}$;
return $\mathcal{O}$

7.2 Materials

7.2.1 Hardware Specifications

The majority of the hardware used for this implementation of the MIRTE Master stems from the standard components implemented by the MIRTE team. On the web-page of this Materials section, below, the list of all hardware parts can be seen, excluding only trivial components such as nuts and bolts. Given that MIRTE doesn't work with large quantities of robots, meaning no standardised parts are available, 3D-printing was a viable choice to establish the necessary parts of the frame using (for this specific project slightly adjusted versions of) the CAD-models provided. These parts can be divided into two groups: chassis and manipulator.

Chassis The chassis consists of a top, bottom and manipulator-mounting plate, for which aluminium plates with a thickness of 1.5 mm were used, as well as several side panels. These side panels don't only act as chassis support elements but also as mounting components for several electronics parts. For both of these purposes, PETG was chosen as a good filament to use for 3D-printing the panels. This filament been used for this application due to its relatively high strength, in addition to having enough ductility, it is easy to print and is relatively cheap. The ductility not only makes the robot more resistant to impact due to it being less brittle, it also allows for some slight bending of the chassis which has the added benefit that the wheels are more likely to keep traction on the floor.

The variant of PETG used for this project was translucent which, aside from being aesthetically pleasing, also enables the user to look at the status lights on the inside of the chassis. This would otherwise not have been possible due to the aluminium top and bottom plates. During printing, orange PETG accent lines have also been added to the robot for both aesthetic purposes and it standing out more to people walking by.

Manipulator The manipulator consists of several components that can be found within the list at the top of the Materials webpage. This system can be divided into four groups: brackets, limbs, servo-motors and the gripping mechanism. Given that there are four servo-motors, the arm is defined to have four degrees of freedom to be able to reach all places around the robot.

There is a fifth servo-motor mounted, but that only actuates the gripping mechanism and therefore doesn't add any degree of freedom to the system. All of this, including the chassis, can be seen in the interactive display near the bottom of the Mechanical overview page.

7.2.2 Software specifications

The MIRTE Master robotic platform has free open-source software available for any user to install.
The latest stable release is used in this robotic system. This software comes packaged inside a ROS application [22], a standardised framework for developing distributed robotic systems. This allows for the use of a wide variety of cross-compatible plugins and additional software packages, making rapid prototyping and development more efficient.

The particular ROS distribution the MIRTE Master platform implements is ROS2 Humble Hawksbill on Ubuntu 22.04.

SLAM Before the robot is able to perform any complex task in an environment, the environment must first be mapped. For this, SLAM (Simultaneous Localization and Mapping) is used. This way the robot can dynamically update its environment based on measurements from its LiDAR scanner. The robot creates an occupancy grid map as an image while estimating the robot's pose. The two most widely adopted SLAM frameworks in the ROS ecosystem are Cartographer [23] and SLAM Toolbox [24]. For the MIRTE Master platform, SLAM Toolbox was selected due to its superior mapping accuracy and localization performance when reliable odometry is available [25]. Since the MIRTE Master is dedicated to running the core robotics software stack and is not burdened by other computationally intensive workloads, the higher computational requirements of SLAM Toolbox do not pose a significant limitation. Consequently, prioritizing mapping accuracy over computational efficiency makes SLAM Toolbox the most suitable choice for this application. To implement SLAM into the MIRTE Master robot, SLAM Toolbox is used. This software package was chosen due to its native ROS2 support and good integration with other ROS2 packages. SLAM Toolbox allows for straight forward modification of mapping behaviour using the set of parameters it provides. In the case of the MIRTE Master, there are several projects that have implemented SLAM Toolbox, so finding a ready-to-use set of SLAM parameters for this robot is relatively simple. The MIRTE Navigation ROS package is used as a starting point for configuration files and SLAM parameter settings. It was assumed the robot would only navigate in unknown environments without predefined or reoccurring maps. Therefore localization (using AMCL) was disabled in the application.

Manipulation While the arm on the MIRTE Master can be controlled in joint-space, the implementation relies on task-space (cartesian-space) control over the end effector position.
The manner which in motion planners are typically implemented requires the integration of several complex subsystems like inverse and forward kinematic solvers, trajectory planners and path planners. To accomplish the goal of task-space control over the arm, MoveIt 2 ([26]) was used.

Navigation Robot navigation also has a challenge analogous to that of robot manipulation, namely that of workspace control. The robot must be able to navigate to or through a set of waypoints given in the coordinate system of the map, while also implementing a real-time controller for obstacle avoidance. Similar to MoveIt 2, Nav2 is used as a solution to this problem ([3]). Aside from path planning and real-time control, Nav2 also provides a Costmap.

7.3 Figures

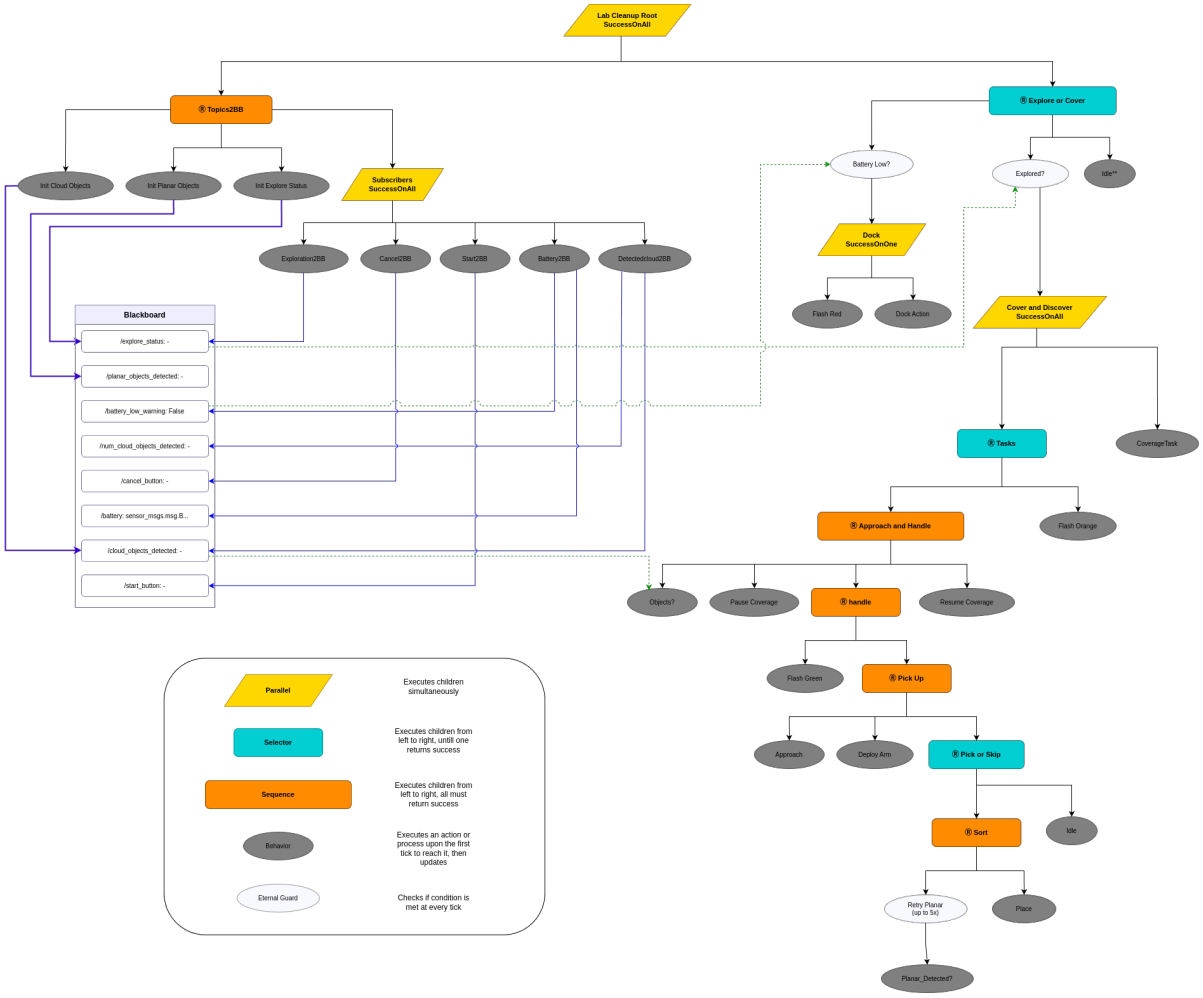

Figure 23: Full Behaviour Tree

References

[1] J. Stückler, R. Steffens, D. Holz, and S. Behnke, "Efficient 3d object perception and grasp planning for mobile manipulation in domestic environments", *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1106–1115, 2013, Selected Papers from the 5th European Conference on Mobile Robots (ECMR 2011), ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2012.08.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889012001297.

[2] D. Stonier, *Py_trees_ros documentation*, Accessed: 2026-06-12, 2024. [Online]. Available: https://py-trees-ros.readthedocs.io.

[3] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The Marathon 2: A Navigation System", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. [Online]. Available: https://github.com/ros-planning/navigation2.

[4] A. J. Becoy, K. Khomenko, L. Peternel, and R. T. Rajan, "Autonomous navigation of quadrupeds using coverage path planning with morphological skeleton maps", *Frontiers in Robotics and AI*, vol. Volume 12 - 2025, 2025, ISSN: 2296-9144. DOI: 10.3389/frobt.2025.1601862. [Online]. Available: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2025.1601862.

[5] H. Niu, X. Ji, L. Zhang, F. Wen, R. Ying, and P. Liu, *A Skeleton-Based Topological Planner for Exploration in Complex Unknown Environments*, https://arxiv.org/abs/2412.13664, 2025. [Online]. Available: https://arxiv.org/abs/2412.13664.

[6] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem", *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977. DOI: 10.1137/0206041.

[7] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot", in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, 2001, 1927–1933 vol.2. DOI: 10.1109/ROBOT.2001.932890.

[8] S. Macenski, T. Moore, D. V. Lu, A. Merzlyakov, and M. Ferguson, "From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2", *Robotics and Autonomous Systems*, vol. 168, p. 104 493, 2023. DOI: 10.1016/j.robot.2023.104493.

[9] G. Williams, A. Aldrich, and E. A. Theodorou, "Model Predictive Path Integral Control: From Theory to Parallel Computation", *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[10] S. M. Elbouhy, A. Adamanov, P. M. Braun, and H. W. Rose, *Comparative Analysis of Local Trajectory Planning Algorithms in ROS2*, Unpublished manuscript, 2024.

[11] S. Thrun, D. Fox, and W. Burgard, "The dynamic window approach to collision avoidance", *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[12] S. Macenski, S. Singh, F. Martin, and J. Ginés Clavero, "Regulated Pure Pursuit for Robot Path Tracking", *Autonomous Robots*, vol. 47, no. 6, pp. 685–694, 2023. DOI: 10.1007/s10514-023-10097-6.

[13] Ultralytics, *Ultralytics YOLO26*, https://docs.ultralytics.com/models/yolo26/, Accessed: 2026-06-11, 2026. [Online]. Available: https://docs.ultralytics.com/models/yolo26/.

[14] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee, "Yolo26: Key Architectural Enhancements and Performance Benchmarking for Real-Time Object Detection", 2026. [Online]. Available: https://arxiv.org/abs/2509.25164.

[15] A. Limaran, A. Wicaksono, and P. Herwanto, "Performance enhancement of embedded object detection via neural hardware acceleration", *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 24, p. 126, Feb. 2026. DOI: 10.12928/telkomnika.v24i1.27448.

[16] R. Sapkota and M. Karkee, "Ultralytics YOLO Evolution: An Overview of YOLO26, YOLO11, YOLOv8 and YOLOv5 Object Detectors for Computer Vision and Pattern Recognition", 2026. [Online]. Available: https://arxiv.org/abs/2510.09653.

[17] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing", *arXiv preprint arXiv:1801.09847*, 2018.

[18] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography", *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. [Online]. Available: https://doi.org/10.1145/358669.358692.

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, AAAI Press, Portland, Oregon, 1996, pp. 226–231.

[20] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi Tessellations: Applications and Algorithms", *SIAM Review*, vol. 41, pp. 637–676, Aug. 2006. DOI: 10.1137/S0036144599352836.

[21] H. Choset and P. Pignon, "Coverage Path Planning: The Boustrophedon Cellular Decomposition", in *Proceedings of 1st International Conference on Field and Service Robotics (FSR '97)*, 1997, pp. 216–222.

[22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild", *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[23] P. Kicman, P. Silson, A. Tsourdos, and A. Savvaris, "Cartographer - A Terrain Mapping and Landmark-based Localization System for Small Robots", in *AIAA Infotech at Aerospace Conference and Exhibit 2011*, Mar. 2011, ISBN: 978-1-60086-944-0. DOI: 10.2514/6.2011-1513.

[24] S. Macenski and I. Jambrecic, "Slam Toolbox: Slam for the dynamic world", *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021. DOI: 10.21105/joss.02783. [Online]. Available: https://doi.org/10.21105/joss.02783.

[25] İ. İnce, D. Yiltas-Kaplan, and F. Keleş, "From simulation to reality: Comparative performance analysis of SLAM Toolbox and Cartographer in ROS 2", *Electronics*, vol. 14, no. 24, p. 4822, 2025, ISSN: 2079-9292. DOI: 10.3390/electronics14244822. [Online]. Available: https://www.mdpi.com/2079-9292/14/24/4822.

[26] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the Barrier to Entry of Complex Robotic Software: A MoveIt! Case Study", *Journal of Software Engineering for Robotics*, vol. 5, no. 1, pp. 3–16, 2014. DOI: 10.6092/JOSER_2014_05_01_p3.