Group 3 - Newton's Pickers

RO47007 Multidisciplinary Project – 2025

I. Agarwal, I. Raducanu, L. Serrano Ruber , J. Verhoog, J. Welgemoed

Contents

Abstract

This project presents the design, implementation, and evaluation of an autonomous apple-picking robot developed by Newton's Pickers. The system addresses critical challenges in an orchard environment, including labor shortages and inconsistent harvesting quality. Built using Mirte and a ROS 2-based modular software stack, the robot integrates visual perception, detection, localization, motion planning, and manipulation to locate, pick, and place apples with minimal human supervision. This design was iteratively validated across 12 tests to meet all its functional requirements, with 10 passing successfully. Key functionalities like apple detection, planning, manipulation, localization, and communication with the farmers were demonstrated reliably in the real-world test conditions. However, the autonomous navigation remains a limitation due to the inability of hardware-software integration. Overall, the results show that the proposed solution is technically feasible and reliable within its operational domain, providing a promising step towards scalable and sustainable automated harvesting systems. "(AGARWAL)"

I. Agarwal, I. Raducanu, L. Serrano Ruber , J. Verhoog, J. Welgemoed
July 28, 2025

1

Start of the Project

1.1. Meet the team

Every successful project starts with a strong team driven by collaboration, curiosity, and a shared goal. We are no exception. We believe in open communication, mutual respect, and the power of effective brainstorming. The brief overview of the responsibilities of the organizational and technical functions is detailed in Table 1.1 and visualized in Appendix A.2. Our team name, Newton's Pickers, matching outfits team photo, as can be seen in Figure 1.1, and team logo, as seen in Appendix A.2, all contribute positively to our team spirit. ”(RADUCANU)””(AGARWAL)”

Table 1.1: Responsibilities of Organizational and Technical Functions ”(RADUCANU)” ”(WELGEMOED)”

Role	Person	Responsibilities
Project Manager/ Systems Engineer / Machine Perception Engineer	Jasper Welgemoed	• Keeping the group on track • Planning and execution of the project • Managing project scope & deliverables • Integration of sub-systems • Obstacle Detection & mapping of the environment
Chair / Planning & Decision Engineer	Ioana Raducanu	• Setting up meeting agendas • Leading meetings & Managing conflicts • Obstacle-free Path Planning of Robot Base and Arm • Execution of Robot Base and Arm Path • Creating logs of the most important changes in Git
Secretary / Human Robot Interaction Engineer	Lucia Serrano Ruber	• Taking meeting minutes • Checking on progress of TODOs • Consulting team regarding safe robot-farmer collaboration • Implementing apple & basket classification, detection, and localization
Promotion Manager/ Planning & Decision Engineer / HR	Ishita Agarwal	• Maintaining a professional environment within the team • Managing conflicts • Obstacle-free Path Planning of Robot Base and Arm • Execution of Robot Base and Arm Path
External Affairs / Quality Control / Machine Perception Engineer	Jolle Verhoog	• Arranging contact with external relations • Doing a final check of our work before submitting • Submitting deadlines on Brightspace • Obstacle Detection & mapping of the environment • Training YOLO models

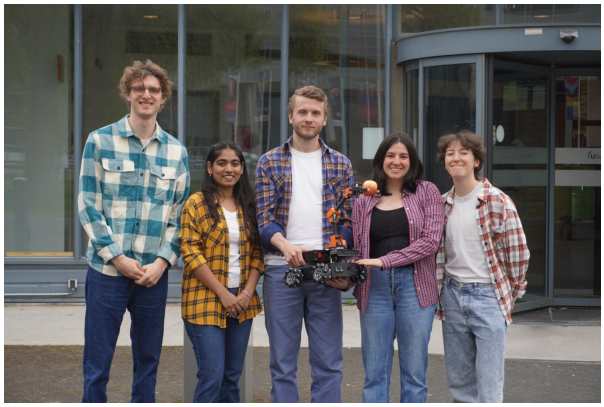


Figure 1.1: Team Photo with Mirte Master ”(WELGEMOED)”

1.2. Ways of working

Due to the multidisciplinary nature of the project, the team needs to constantly be aware of both the high-level (as managers) and low-level (as engineers) aspects of the project. For this reason, it was decided to follow a Systems Engineering (SE) approach. This structured methodology helps break down the project into manageable parts. It ensures that all requirements are clearly defined, verified, and validated throughout each step of the process by keeping the main goals of the project and the client in sight. This approach supports good communication, traceability, and structured decision-making. However, to ensure smooth interaction within the team, a common vision with mutual agreements is needed. To this end, common management roles, as can be seen Table 1.1 and more visually in Appendix A.1, were divided. This division is based on our own learning goals defined in our personal reflection reports. To further align our vision, everyone’s expectations for the project were discussed, and based on these, the following organizational agreements were made:

- **Tri-weekly group work sessions** on Tuesday, Wednesday, Thursday from 08:45-12:45 at our assigned tables. The agenda will be prepared by the Chair Ioana Raducanu.
- **Individual working sessions** worth 6 hours per week to reach the recommended 18 hours per week.
- **Milestone meetings** before important project deadlines to align and review our progress. The Project Manager, Jasper Welgemoed, will be responsible for organizing these.
- **Central documentation** will be kept in Notion. This includes storing sources, preliminary research, brainstorm sessions, meeting minutes, deadlines, requirements validation and verification, personal goals, and trade-off results. Git will be used for the smooth integration of the different departments’ subsystems.
- **Internal communication** will be mostly in-person during the aforementioned work sessions or via WhatsApp when working individually or outside of normal working hours.
- **Usage of generative AI** will be allowed to support learning during coding, but not for writing the report.
- **Interpersonal Relations** will be maintained by organizing team-building activities. HR Manager Ishita Agarwal is responsible for organizing these and for addressing any hiccups in the quality of the relationships between team members.
- **Code management** will be done through *GitLab*. We will check each other’s code through Pull Requests (PRs) where we will have at least two people review a PR before merging.
  ”(WELGEMOED)” ”(VERHOOG)”

1.3. Problem definition

The client wants us to develop a solution that supports the UN’s Sustainable Development Goal (SDG) 2[1]: Zero Hunger by promoting sustainable agriculture and improving food security. UNity Orchard, an ecological apple farm working with the United Nations, is currently facing labor shortages and tough

[1] see `sdgs.un.org/goals` for the UN’s goals for sustainable development

working conditions, making it difficult to harvest apples and compete with large-scale producers.

The robot must operate in a natural orchard environment, characterized by evenly spaced rows of trees, variable weather conditions, uneven terrain with small bumps, muddy ground, and the presence of workers, animals, fallen apples, and high grass. The first two are based on a manual for apple pollination by a sub-organisation of the UN[7]. These dynamic conditions pose challenges to navigate and interact with the environment.
To address this, the client wants us to create a robot that can autonomously navigate the orchard, detect ripe apples, pick and chandle apples carefully, and communicate its status to the farmer, all while avoiding obstacles & workers and minimizing overall disruption to the working conditions. ”(VERHOOG)” ”(AGARWAL)”

1.4. Proposed solution

To address the problem described in Section 1.3, the following solution is proposed. Our goal is to develop an autonomous apple harvesting robot that can communicate its own and the harvest’s status to a human farmer through an interface while it navigates around the artificial orchard, decides which apples on the trees are ripe, and consequently collects those ripe apples into a basket, by using global path planning techniques, multimodal sensor data, and decision-making strategies.

To autonomously navigate around an artificial orchard, the system will use a global path planning method. For this, the system will perceive its environment through the available sensors and generate a map of the orchard with which it can remain aware of its location throughout its run. Additionally, the system will also use a local path planning method to avoid possible dynamic obstacles such as other robots, humans, or animals that could intercept its path.

To pick the ripe apples, the system will use computer vision to localize the red apples in the trees. These locations will be used as inputs to determine the appropriate movements of the robotic arm to pick the apple. From here, the robot will navigate to the basket and place the apple inside.

Lastly, the robot will be communicating its status and data to the farmer. It will continuously keep track of data about the harvest, such as the percentage of ripe apples in the orchard and the number of apples it has picked so far. The system will make use of a user interface to communicate this data to the farmer upon request. The robot will also be able to communicate its status autonomously in the case that it needs human intervention, for example, when it is stuck low on battery. ”(SERRANO RUBER)”

1.5. Requirements

To reach our proposed solution, mandatory and optional requirements are formulated. The mandatory requirements are requirements that the robot can not function without, and the optional requirements add to the benefit of the client. To add further depth to this simple split, a tree hierarchy of functional, design, and performance requirements was set up. These can be found in Appendix A.2. Further details of our test setup are described in Appendix A.3. Each requirement is fitted into one of the three team specializations: planning & control, perception, or human-robot interaction. The mandatory requirements can be found in Table 1.2, and the optional requirements can be found in Table 1.3.”(VERHOOG)”

1.6. Project plan

To ensure a structured and goal-oriented approach throughout the project, a detailed project plan following the SMART framework (Specific, Measurable, Achievable, Relevant, Time-bound) is utilised. A task breakdown can be seen in Appendix B. This lists all the phases of the project and key actions within the phases to realize our proposed solution. It also allocates time and responsibilities in the team based on individual roles and expertise. The tasks are broken down into manageable steps and assigned to specific team members, ensuring accountability and traceability. A Gantt chart has been created to visualize these actions, showing the interdependencies between tasks and providing an overview of the resource allocation as seen in Appendix B.

”(WELGEMOED)” ”(RADUCANU)”

Reference	Short description	Specialization
MR1	The robot must localize all visible trees with a spatial offset of maximally 15 centimeters to the tree trunk base.	MP
MR2	The robot must localize the apple basket when it is not obscured and within 3 meters, and with a spatial offset of maximally 5 centimeters.	MP
MR3	The robot must generate a static map of the environment, incorporating LiDAR, sonar, and camera depth map measurements every second.	MP
MR4	The robot base must always avoid static obstacles with a minimum clearance of 10 centimeters	Planning
MR5	The robot base must avoid dynamic obstacles with a minimum clearance of 1 meter	Planning
MR6	The robot base must autonomously plan and execute paths through the orchard with a maximum speed of 1 meter per second, and an absolute path following error of maximally 10 centimeters.	Planning
MR7	The robot must localize all visible apples within 30 centimeters of the robot base, with a spatial offset of maximally 5 centimeters.	MP
MR8	The robot must classify all localized apples by ripeness, based on color, with an F1-score of at least 90%.	MP
MR9	The robot gripper must grip, hold, and drop the apple securely without damaging the apple, with an unwanted apple drop rate per iteration of maximally 10%.	Planning
MR10	The robot gripper must autonomously plan and execute gripper paths towards apples or the basket, with a maximum joint velocity of 1 meter per second and a path following error of maximally 5 centimeters.	Planning
MR11	The robot must communicate data about its battery and critical errors in real-time with a maximum delay of 5 seconds.	HRI
MR12	The robot must have recovery mechanisms to recover from operational errors for sensor faults, grasp failures, and path execution/obstruction issues with a success rate of 80%.	MP/Planning

Table 1.2: List of mandatory requirements in this project. ”(AGARWAL)” ”(WELGEMOED)” ”(VERHOOG)” ”(SERRANO RUBER)”

Reference	Short description	Specialization
OR1	The robot can communicate data about the harvest progress (number of apples picked) with a maximum delay of 5 seconds.	HRI
OR2	The robot can dynamically update the environmental map and track moving objects up to 1Hz.	MP
OR3	The robot can prioritize certain areas of the orchard to determine an intelligent harvesting order.	Planning
OR4	The robot can work efficiently by planning optimal paths for both base and arm.	Planning
OR5	The robot can log sensor and path execution failures at 0.5 Hz	HRI

Table 1.3: List of optional requirements in this project. ”(AGARWAL)” ”(WELGEMOED)” ”(VERHOOG)” ”(SERRANO RUBER)”

2 Functional Architecture

2.1. Functional hierarchy

The purpose of the robot, elaborated upon in the earlier chapter in the form of the problem definition and the requirements, should be realized through the functions that the robot executes. Some of the parent functions, derived mostly from the functional requirements, can be better conceptualized as aggregating other, simpler, functions. The simpler sub functions dissect the parents into more manageable operations; during the implementation stage these sub functions can create simpler self-contained nodes with clear connections and classifications. (”RADUCANU”)

The following subsections describe the parent functions and their respective child functions.

F1. Perceive environment

Perceive environment (F1) processes the sensor data from sonar and LiDAR, and translates these into various modes. The child functions F1.1, F1.2, F1.3, and F1.4 self-localize the robot, generate the static map, generate the dynamic local costmap, and signal the emergency brake, respectively. The LiDAR output is used for mapping and self-localization, while the sonar is used for emergency braking. These functions are related to requirements MR1, MR2, MR3, and OR2. (”RADUCANU”)

F2. Navigate around orchard

This function encompasses the parts required for successfully moving the base of the robot from A to B. This includes calculating collision-free paths (F2.2) where the path is avoiding static objects, an additional function that avoids any other (dynamic) obstacles (F2.1) while driving the robot around with the path executing function (F2.3). These functions are related to requirements MR4, MR5, MR6, OR3, and OR4. (”SERRANO RUBER”)

F3. Detect apples

This function describes the process by which the robot will find the apples it has to pick. It will first localize the apples in a tree with function F3.1, which is used as input to classify whether they are ripe or not (F3.2), and thus whether they should be picked. These functions are related to requirements MR7 and MR8. (”SERRANO RUBER”)

F4. Manage gripper

Manage gripper (F4) describes functions that will control the robot’s arm movement to either pick an apple from a tree or place an apple inside a predefined basket position. The child function F4.1 calculates the trajectory for the joints of the arms, which function F4.2 will then execute. The function also has a child function F4.3 controlling the gripper’s grasping movement. This function relates to requirements MR9, MR10, OR3, and OR4. (”SERRANO RUBER”)

F5. Communicate with the farmer

Communicate with farmer (F6) describes the different ways in which the system will communicate information to the farmer. There are two main types of communication that will work differently. Firstly, in F6.1 the robot will communicate the current state of the orchard and harvest progress. The robot will continuously update its internal state. With this, the interface can be continuously updated, and the farmer can access the information on demand. The second type of communication is an alert or warning about the system status (F6.2). This will include things such as unresolved errors or battery warnings, where the farmer can intervene to help. Furthermore, the farmer will communicate to the robot when to start, stop, or return to start. These functions are related to requirement MR11 and OR1. (”SERRANO RUBER”)

F6. Manage exceptions

This last function handles errors and exceptions thrown by the system. This could be warnings such as ”low battery” or ”basket full.” Some may be able to be handled by the system itself, while others may need human intervention. This function is related to requirements MR12 and OR5. (”SERRANO RUBER”)

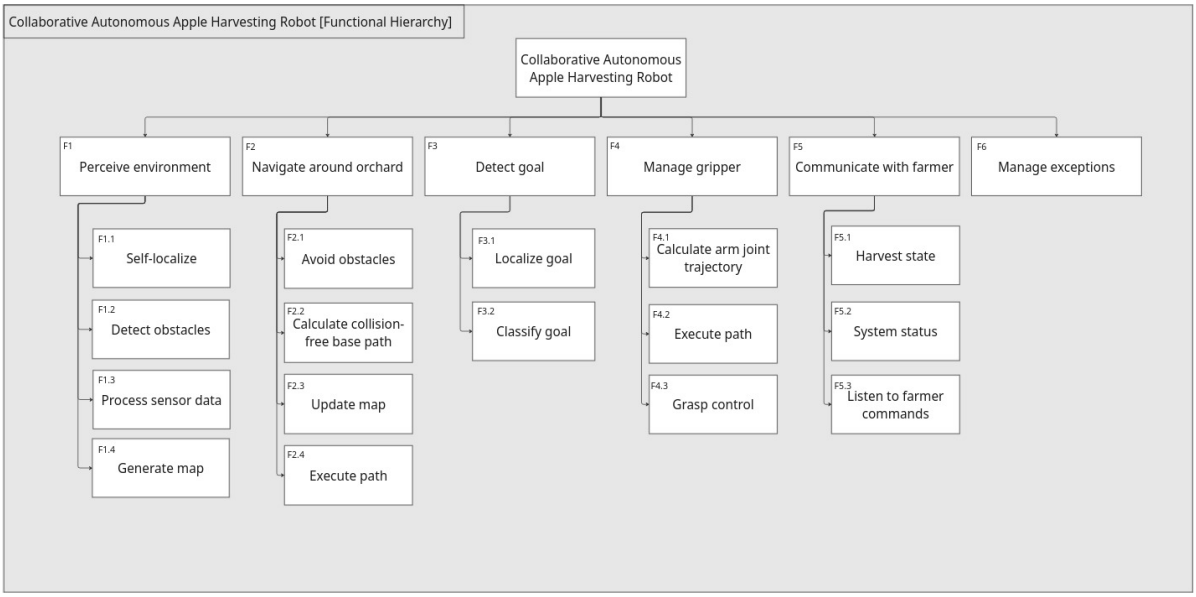


Figure 2.1: Functional hierarchy of the system. ”(SERRANO RUBER)”

2.2. N^2 chart of your system

Within the hierarchical structure from Sec. 2.1, the parent functions are the most appropriate level to reflect upon. Each of these functions interfaces with others. In the N square chart, see Fig. 2.2, the interactions are displayed within a matrix representation, specifically showing the inputs and outputs of the function with respect to the needs of other functions. The external inputs are the sensor readings, split into visual information and point clouds, and the farmers’ commands. The external outputs from the system to the world are the user interface, where the robot communicates with the farmer, and the physical difference of apples now being in the basket. ”(RADUCANU)”

The operation of the robot is interdependent, and thus, there exist some loops in the N2 chart of the system. The input of sonar, LiDAR, and camera depth map clouds is evaluated continuously by the *Perceive environment* function (F1) to obtain a map, where obstacles can be identified and where the robot can self-localize. This information is passed on to the *Navigate around orchard* function (F2). This navigation function also takes inputs from the gripper actuations being finished and from exception management. Depending on whether the gripper has an apple, the robot will move either to the closest unexplored vertical point cloud, a suspected tree, or to the collection basket. There, it will hand over control to the third functionality, *Detect apples* (F3), or the fourth functionality, *Manage gripper* (F4).

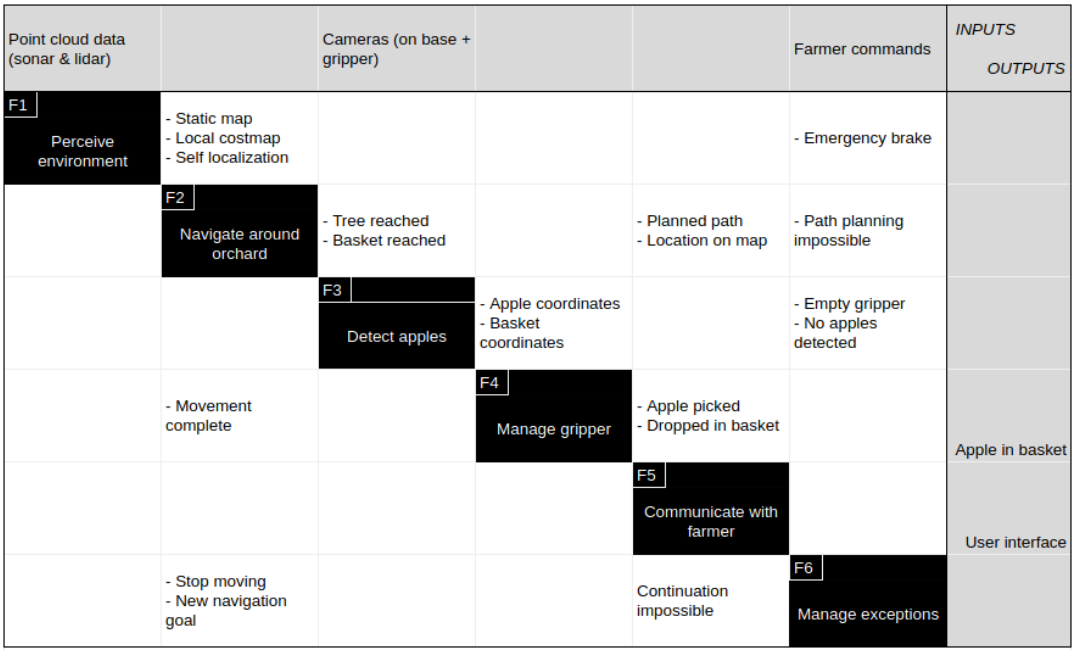


Figure 2.2: N^2 chart of the Collaborative Autonomous Apple Harvesting Robot. ”(VERHOOG)” ”(RADUCANU)” ”(SERRANO RUBER)” ”(AGARWAL)”

The map and the robot’s planned path will be given to the fifth functionality, *Communicate with farmer* (F5). Lastly, if the path planning task is deemed impossible by the robot, this information will be passed to the sixth functionality, *Manage exceptions* (F6). ”(VERHOOG)””(AGARWAL)”

Once the third functionality, *Detect apples* (F3), is prompted, the visual information from both cameras will be used as the input. This function can be called when the robot has reached a suspected tree or when an apple is dropped from the gripper. The base will be stationary, and the cameras mounted on the base and the gripper will be utilized to detect apples. If an apple is successfully found, the next functionality, *Manage gripper* (F4), is called. If there are no apples found by the cameras or if the apple is dropped, this information will be sent to the function *Manage exceptions* (F6).”(VERHOOG)””(AGARWAL)”

Manage gripper (F4) follows after the apple has been detected. This function describes the gripper arm actuation to grasp the apple. Once it is successful, it will tell the base navigation (F2) that it is okay to move again, and it will tell the farmer that an apple is picked (F4). If the apple were to be dropped, the function *Detect apples* (F3) catches the exception and looks for a new apple while remaining stationary. If this fails, the Navigate around orchard function is prompted to move. ”(VERHOOG)””(AGARWAL)”

For the reverse operation of picking, namely placing them in the basket (F4), the apple detection (F3) is not needed. The function *Manage gripper (F4) is started again* when it is told that the robot is standing next to the basket, and once it has completed its task, that information is relayed to the function *Navigate around orchard* (F2) and *Communicate with farmer* (F5). An output for this will be the difference in the real world, since now there is an apple added to the basket.”(VERHOOG)”

Lastly, there is a separate functionality defined for exception handling. The inputs of the farmer are seen as exceptions, for the robot should be autonomous. The farmer could indicate to the robot to come back to its start or to stop completely. All errors that this function collects will be shared with the function *Communicate with farmer* (F5). It can also give the signal to detect potentially fallen apples to function *Detect apples*, and it can signal *Navigate around orchard* to navigate to the start location, the next suspected tree, or stop moving altogether. ”(VERHOOG)”

2.3. Functional Flow of your system

As aforementioned in Sec 2.2, there are quite a few loops within the collaborative autonomous apple harvesting robot system. These loops are difficult to envision using only the N^2 chart; in the activity flow, 2.3, the sequences and flow of actions within the main operational scenario are described with a more technical approach. We will let the operator use 3 different commands, which are the start, return to start, and stop commands."(WELGEMOED)" "(RADUCANU)"

The functional flow diagram can be seen in Figure 2.4 and Figure 2.3. First, the main node starts. Together with the "Perceive Environment" which at a certain interval looks for objects on a planned path, the "Detect Apples" node, which at a certain interval tries to detect apples in the gripper camera and the base camera, and the "Obtain Farmer Commands" node which at a certain interval listens to commands of the farmer. When the main node first starts up, it waits for the farmer's command. This can be "Stop", "Return to start", or an "Order Request".
If the "Stop" command is given, the robot is stopped (as an emergency) and can be started again when the node is manually started up again.

Suppose the "Return to start" command is given. In that case, the navigation part activates, canceling the current task, running through the loop of obtaining a trajectory to the goal and monitoring its movement to the goal while listening to the Perception Node. If an obstacle is sensed, it cancels the current trajectory and tries to obtain a trajectory to the goal again. If the goal is reached, the gripper actuation starts. "(WELGEMOED)"

If an "Order Request" is provided, the robot checks if there are apples left. If there are, the robot will obtain a map of the orchard and a trajectory to the tree. If there are no static obstacles in its way, it will move to its goal while constantly checking for dynamic obstacles. If an obstacle is sensed nearby, the trajectory gets cancelled, and a new trajectory will be made. When the tree is reached, the robot will scan for apples with its camera. If no apple is detected in the gripper camera or in the base camera, then there is nothing to pick. In that case, it needs to go back to localizing and navigating to another possible position to look for apples.
If an apple is detected in the gripper, the apple needs to be placed in the basket. Then, it will obtain the location and trajectory of the basket and move towards it. During this movement, the gripper status is monitored to ensure the apple has not fallen. After the apple is placed in the basket, the robot will detect apples again to see if there are more apples to pick. If no apple is detected in the gripper, but there is an apple detected in the base camera, then it means the apple needs to be picked. Then we obtain a trajectory to the apple and move the gripper to pick it. Then it will go back to detecting apples, to see if the apple is successfully picked or if there are still ripe apples to be picked.

After the apple has been placed in the basket, a new weight estimation will be communicated to the farmer. After this, the loop will start over again. "(WELGEMOED)"

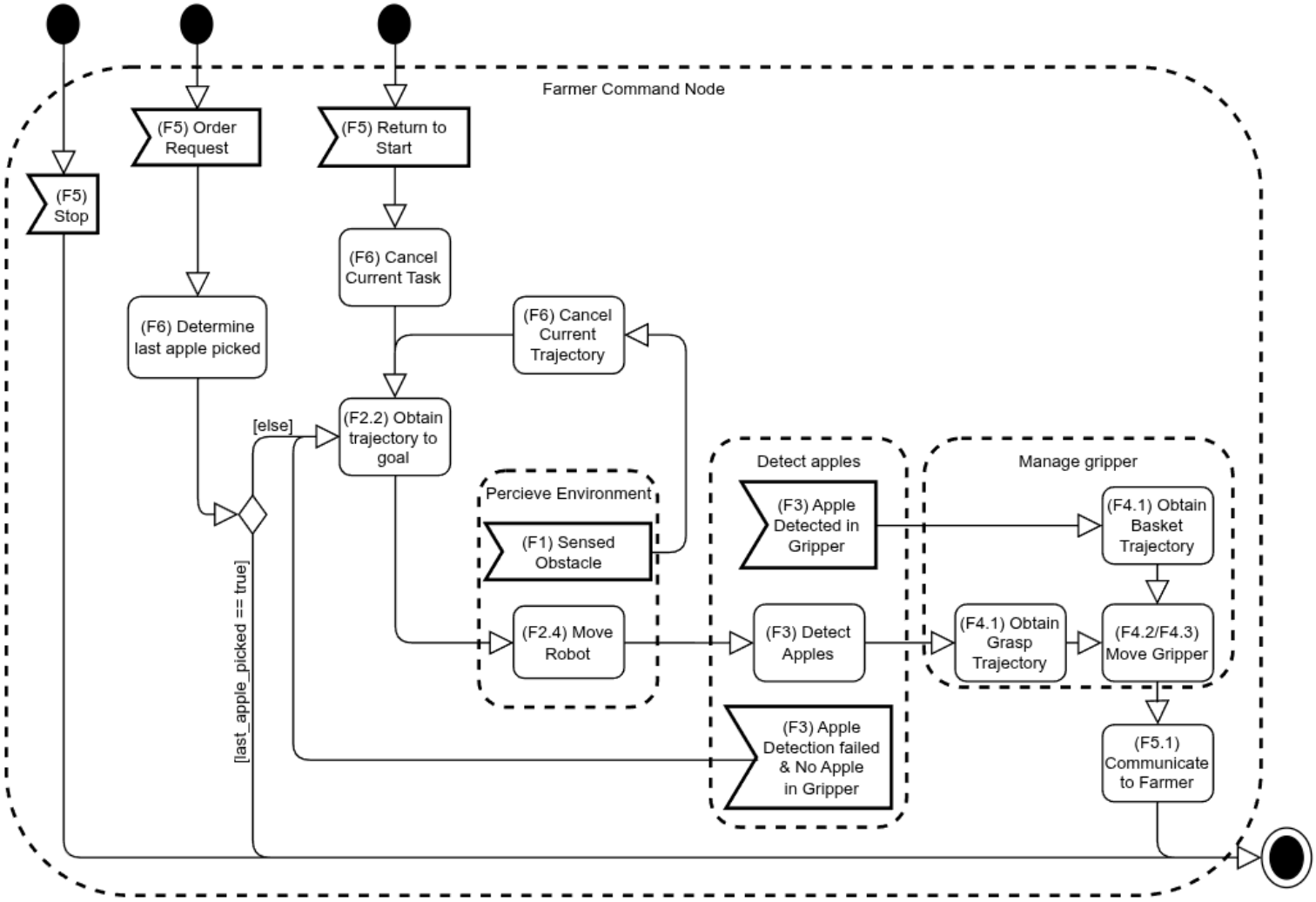


Figure 2.3: Functional Flow Diagram Main ”(WELGEMOED)””(AGARWAL)””(VERHOOG)”

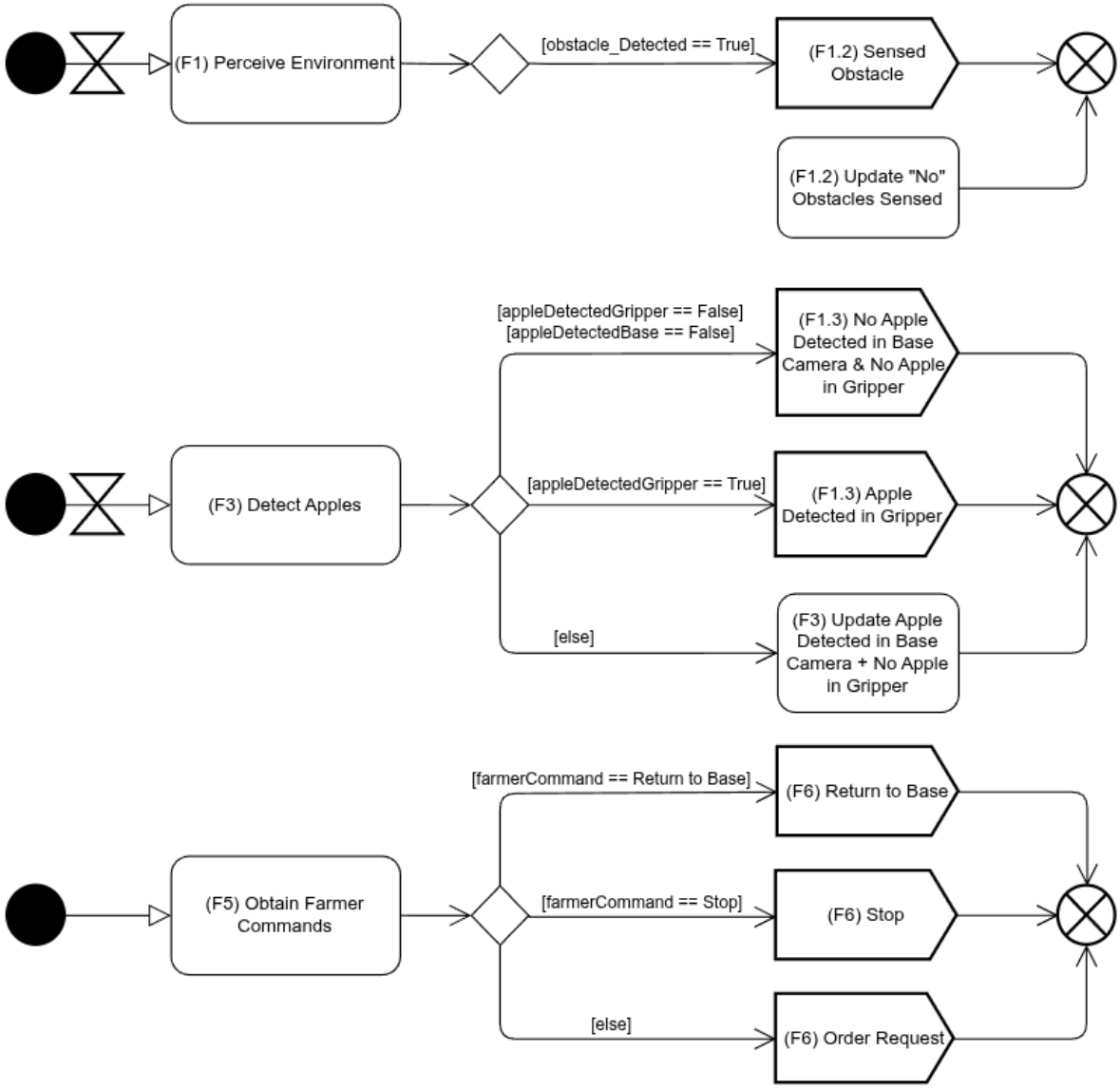


Figure 2.4: Sequential Nodes ”(WELGEMOED)””(VERHOOG)””(AGARWAL)”

3

Description of the Robot Software

3.1. Nodes Overview

Given our functional descriptions in Chapter 2, the designed system has been split into nine main nodes, as can be seen in Figure 3.1. The seven functionalities have each been allocated their own node, merged into a single node, or split into multiple nodes. To orchestrate the inner workings of the system, a Finite State Machine (FSM) node serves as the central controller. This node constantly publishes the current state of the robot and listens to the active nodes to determine when control should be transferred. This aligns the process with the flow of the N2-chart in Section 2.2. The system begins when the finite state machine node receives the farmer's 'start' command. Once a navigation goal is reached, the apple detection node starts up. After that node gives the apple count and locations, the gripper planner node starts planning and executing a trajectory for the arm. All top-level exceptions are handled in the logic of the finite state machine, but exceptions at a node level will be handled in that respective node.

The first functionality, Perceive Environment, has been split into the nodes SLAM, and EmergencyBrakeCheck. The second functionality, Navigate around the orchard, has become a dedicated planner node NavPlanner. The third functionality, Detect apples, has also become a separate node, AppleDetection, while the fourth and fifth functionalities are merged into one node: GripperPlanner. The sixth functionality, communicate with farmer, is encapsulated in the FarmerInterface node. The final functionality, manage exceptions, is handled internally in nodes when possible, and otherwise incorporated into the finite state machine internal logic.
The functional flow diagram of our system has also been updated to align with this structure, as can be found in Figure 3.2. Furthermore, swim lanes for the six different self-made nodes are included. The main changes include the removal of excess nodes and the assignment of functional (sub-)requirements to each node. This ensures the system still aligns with our functional requirements and adheres more closely to SysML diagram conventions. ”(VERHOOG)”

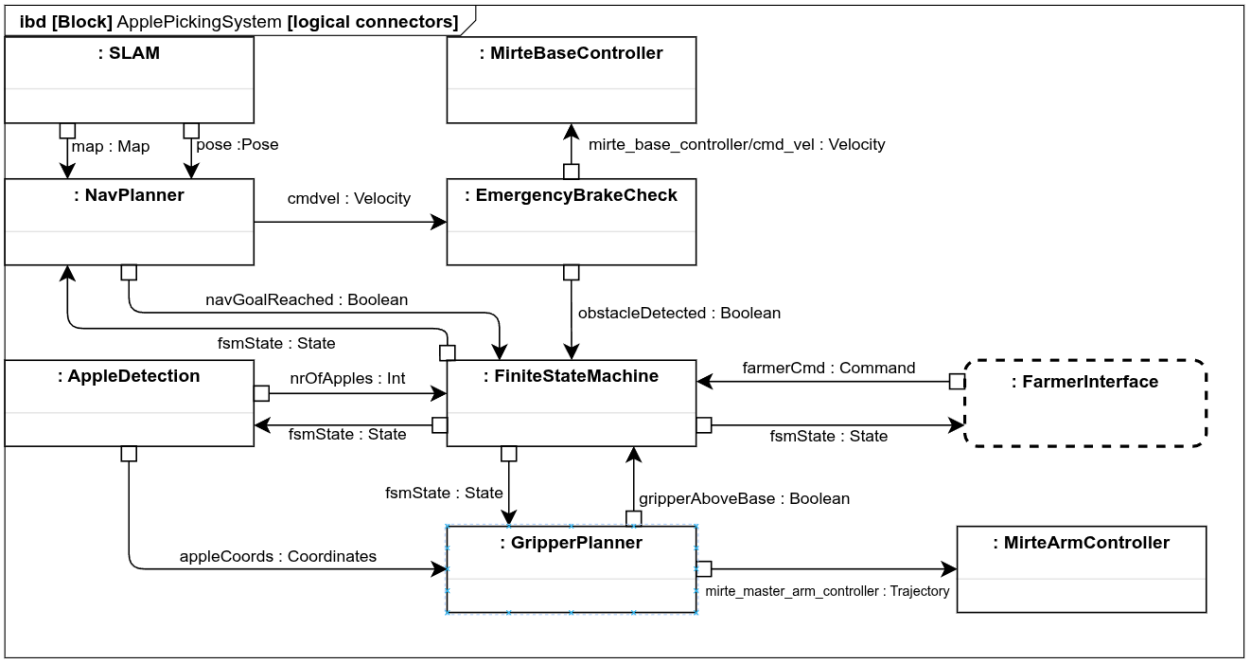


Figure 3.1: SysML Internal Block Diagram presenting how the ROS nodes in the system are connected. ”(VERHOOG)”

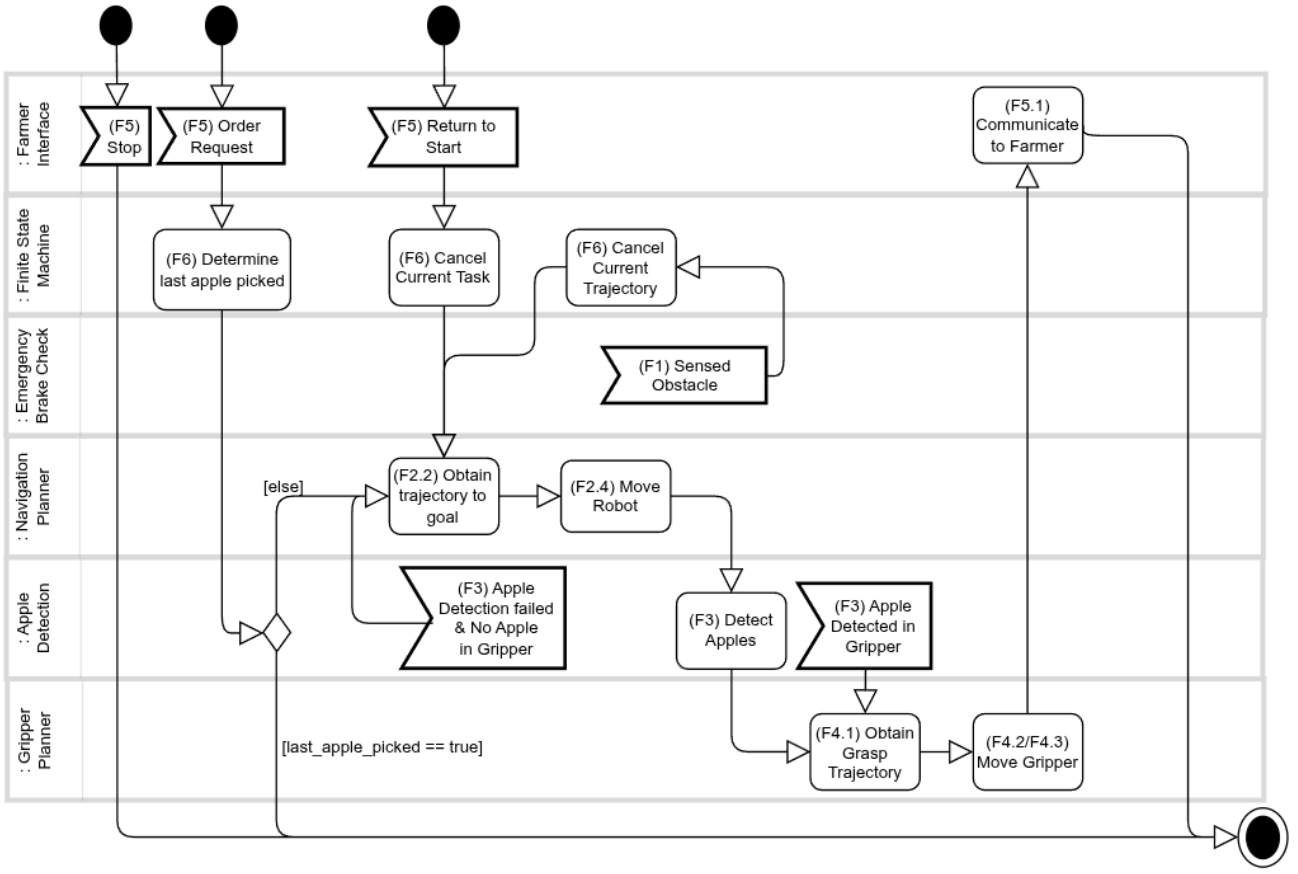


Figure 3.2: Updated Functional Flow Diagram with swimlanes for node allocation. Only the six self-made nodes are included. ”(VERHOOG)””(AGARWAL)”

3.2. Design choices

After the previous high-level overview of the nodes in the system, in this section, the main design decisions in terms of the requirements shall be addressed. Three of the above nodes are pre-existing nodes that will be used as they are. These are the controllers for the robot arm and base, as found on the Github of the course [3], and the split-up self-localisation on a given map, and mapping of a new map with Slam Toolbox by Steve Macenski [2] ”(RADUCANU)”.

:SLAM

The SLAM Node can separately run the localization and mapping, only mapping the static environment once before operation and then localizing on that map.

- Existing or self-made packages used (and reasoning): slam_toolbox runs the original localization.launch.py. This package is preferred over the alternative nav2_bringup localization.launch.py as this package came with transform errors that were avoidable by using the SLAM toolbox instead.
- Information on configuring the node’s parameter values: Within the parameter file, there are several parameters that are still being investigated based on the refresh time and accuracy of the map. However, this can only be defined properly once the other nodes are fully running as well.
- Other affiliated requirements (in reference codes): MR1, MR2, MR3, OR2

:NavPlanner

The Navigation Node plans the trajectory throughout the static map obtained, avoiding the obstacles.

- Existing or self-made packages used (and reasoning): Within the existing nav2_bringup package, the navigation.launch.py is run with an alternative parameter file from the mirte_navigation package [1]. For actual movement, the NavigateToPose action within the nav2_bt_navigator module within ROS Humble [4] is used.
- Information on configuring the node’s parameter values: Within the parameter file, the type of driving planner is changed to a SMAC state lattice planner. This is done with an additional YAML file where only spinning and straight driving is allowed. Although there are reasons why omnicontrol would be preferred, to have proper data for the emergency brake check, this limited movement is preferred (see EmergencyBrake node).
- Other affiliated requirements (in reference codes): MR4

:EmergencyBrakeCheck

The EmergencyBrake is used for dynamic obstacle detection and avoidance. Since there is no live mapping for dynamic obstacles, the sonars will independently pull an emergency brake and move depending on which sensor is affected.

- Existing or self-made packages used (and reasoning): This is a self-made node. It listens to cmd_vel from navplanner and if and only if there is no obstacle in front of the front-facing sonars within a certain distance, it will relay to /mirte_base_controller/cmd_vel, therefore being the node that processes dynamic obstacles.
- Information on configuring the node’s parameter values: The main parameter value considered is the distance at which it should perform an emergency brake, since this distance is not differentiated in the code between static or dynamic, it is simply set to distance=20 cm. As aforementioned, the robot now only has spin-straight-spin forward control such that the sonars can properly avoid on-heading dynamic obstacles whilst still being able to reach a goal within a reachable distance (MR5)
- Other affiliated requirements (in reference codes): MR12

:AppleDetection

Where SLAM focuses on the global mapping and robot localization, the apple detection is based on the transformations within the robot and finding the apples on the trees through sonar and camera sensors (gripper and base)

- Existent or self-made packages used (and reasoning): OpenCV [5] is used for image processing and detecting the tree. The Ultralytics [8] package is used to train and use a YOLO machine learning model to detect and localize the apples. These packages were chosen for their balance between detection accuracy and computational efficiency.
- Information on configuring the node’s parameter values (including affiliated requirement reference codes): The choice of YOLOv8 variant is based on a trade-off between model complexity and available computational resources. This configuration remains under evaluation to ensure optimal real-time performance in the field. (MR7, MR8)
- Other affiliated requirements (in reference codes): N/A

:FiniteStateMachine

The Finite State Machine (FSM) is the core reasoner for the robot, activating and managing transitions between nodes. This node is the overarching reason for traceability and organization.

- Existing or self-made packages used (and reasoning): Fully self-made node. This was necessary to tightly couple the FSM logic to custom-defined states and transitions, which are specific to the agricultural workflow and robot capabilities. Built using ROS 2 with Python3, leveraging standard messaging types (std_msgs) for inter-node communication.
- Information on configuring the node's parameter values (including affiliated requirement reference codes): N/A
- Other affiliated requirements (in reference codes): All nodes are loosely affiliated as this serves as the connection; therefore, most requirements are as well.

:FarmerInterface

The Farmer Interface node provides a three-option simplistic graphical user interface for the farmer (emergency stop, return to base, order request). As well as a monitoring station for the robot's current state.

- Existing or self-made packages used (and reasoning): Using the standard Python GUI library tkinter [6]. This was chosen for its simplicity, native Python integration, and ease of deployment across platforms without requiring additional dependencies. Other packages, such as PyQt/ PySide, are heavy and overkill, and rqt plugins, although also well integrated, lack flexibility for a non-technical customized styling.
- Information on configuring the node's parameter values (including affiliated requirement reference codes): N/A
- Other affiliated requirements (in reference codes): MP11, OR5

:GripperPlanner

This node converts high-level Cartesian space goals (3D positions) obtained from the apple detection node into low-level joint trajectories for a robotic arm.

- Existing or self-made packages used (and reasoning): Several approaches are explored, both through MoveIt and through decoupling inverse kinematics and trajectory control. Where MoveIt is an extensive package, this also sometimes leads to unexpected irrelevant errors based on graphical interfaces; on the other hand, the inverse kinematics and trajectory control method takes a lot more time, although it allows for better modularity and clearer debugging.
- Information on configuring the node's parameter values (including affiliated requirement reference codes): The parameters that have been adjusted from the existing packages are simply the ones that describe the robot link lengths.
- Other affiliated requirements (in reference codes): MP9, MP10

:MirteBaseController

This node is used to control the base of the Mirte Master. This package is unchanged, and no parameters were tuned.

:MirteArmController

This node is used to control the base of the Mirte Master. This package is unchanged, and no parameters were tuned."(RADUCANU)"

4

Validation of the Robotic Solution

This chapter evaluates whether the robot meets the functional and performance requirements through a combination of verification and validation. Verification was done primarily through unit tests, simulation, and manual inspection of individual components such as the FSM, camera node, and gripper control logic. These tests ensured each module behaved according to its specification.

Validation is performed through system-level integration tests conducted in a physical demo arena. Each test focused on one or more functional requirements and measured the robot's behavior under realistic task conditions. Metrics such as success rate, spatial accuracy, timing, and reactivity were collected using a mix of ROS diagnostics, video recordings, and manual measurements. The results of these tests are presented below. ”(RADUCANU)”

4.1. Test procedures

To verify the robot fulfills its functionalities and to gauge the performance, multiple test procedures are devised. Each procedure is designed to test at least one of the functions of operation, and all tests together should fulfill at least the mandatory requirements. The testing phase was closed off by a full integration test in the test arena which setup can be seen in figure A.3 and A.4. The scenarios, what is checked for, and the result of the test will be included in a separate subsection for each test. A summary of these tests and which functionalities they relate to is given in Table 4.1. ”(VERHOOG)(WELGEMOED)”

Reference	Short description	Function
T1	Startup of the robot	F1, F6
T2	Self localization on the map	F1
T3	Moving to a goal location	F2
T4	Detecting ripe apples with the cameras	F3
T5	Determining relative apple position	F3
T6	Picking an apple from the tree	F4
T7	Determining relative basket location	F1
T8	Placing an apple in the basket	F5
T9	Moving the gripper above the base	F4, F5
T10	Avoiding dynamic obstacles	F1, F2
T11	Managing different robot states with the FSM	F7, F6
T12	Communicating to the farmer	F6

Table 4.1: List of conducted tests in this project.”(VERHOOG)”

4.1.1. Test T1: Startup of the robot
Tested the startup sequence, functions F1 & F6, by booting the robot from the battery in the demo area. Verified that the robot waits for the farmer to give the 'start' command, and all required nodes started spinning. This was inspected manually by looking at the active ROS elements and listening to the FSM state topic. This test succeeded. "(VERHOOG)"

4.1.2. Test T2: Self localization on the map
We generated and saved a static map which can seen in A.5. Lifted the robot to different locations within the demo area, let it drive around, and checked if the localization converged to the correct location with a margin of 10 centimeters. After a few seconds of moving, the Monte Carlo particle filter converged to the correct location with the required accuracy. The offset was invisible with the blind eye, estimated to be less than 5 centimeters, and thus the test was successful. "(VERHOOG)(WELGEMOED)"

4.1.3. Test T3: Moving to a goal location
Set a goal position in the demo area and verified that the robot moved to the specified location with a margin of 10 centimeters. It should also publish to the finite state machine that the goal has been reached. By looking at the topic published by the FSM and measuring the real-life distance from the real goal position to the robot position, it was found that the robot could not use the navigation node in real life like it had done in simulation. This test proves further research should be done on the possibility of the use of nav2 actions directly with the Mirte robot. Therefore, the application did not pass the test. "(VERHOOG)" "(RADUCANU)"

4.1.4. Test T4: Detecting ripe apples with the cameras
Placed the robot at multiple different distances from the tree and verified that it correctly detects at least two of the four apples in the demo area. For this, both cameras are used simultaneously. Checked that the apple detection accuracy within 2 meters in front of the robot is at least 80%. This test succeeded with an apple detection accuracy of 90%. "(VERHOOG)""(AGARWAL)"

4.1.5. Test T5: Determining relative apple position
Given the apple locations in the camera image and the sonar distances to the tree, it was calculated the relative coordinates of the apples were calculated and verified by measuring these. The average measured offset was 2 centimeters, and hence this test passed "(VERHOOG)""(AGARWAL)"

4.1.6. Test T6: Picking an apple from the tree
Placed the robot in front of the tree with the relative apple coordinates given, and verified that the gripper can pick 80% of all possibly pickable apples without moving. Determined all the pickable apples by manually moving the gripper around the apples on the tree. One by one, let the gripper pick these apples. The gripper arm achieved a successful picking rate of 85%, resulting in this test being successful. "(AGARWAL)" "(VERHOOG)"

4.1.7. Test T7: Determining relative basket location
Placed the robot in front of the basket and verified that the basket is detected. Also verified that the predicted relative center basket position, based on the detection and the sonars, aligns with real-life measurements with an offset of maximally 5 centimeters. The average offset of the center of the basket location was 1.5 centimeters, and hence this test succeeded. "(VERHOOG)" "(SERRANO RUBER)"

4.1.8. Test T8: Placing an apple in the basket
Placed the robot in front of the basket with the relative basket location given. Verified that the gripper arm can place the apple inside the basket with a success rate of at least 80%. The average success rate of this test was 100%, resulting in passing the test "(VERHOOG)" "(AGARWAL)"

4.1.9. Test T9: Moving the gripper above the base
Moved the gripper arm into various positions and verified that the robot moved the arm back to above its base with a 100% success rate. After ten different successful test runs, the test was concluded and marked as passed. "(VERHOOG)"

4.1.10. Test T10: Avoiding dynamic obstacles

Let the robot drive to a goal position in the simulation, and it was verified that the robot moved in a twist-straight-twist style. Also verified that when something is in front of or back of the robot within 20 centimeters during movement in that direction, the robot stops moving. Planned multiple different paths in the demo area, and placed new obstacles that were not on the static map, in the path. The robot was unable to navigate, but by putting obstacles in front of it, it did ring the alarm sound and replanned according to the GUI. The application failed the testing, but shows promise if the navigation node were fully working ”(VERHOOG)” ”(RADUCANU)”.

4.1.11. Test T11: Managing different robot states with the FSM

Communicated different feedback from the different nodes to the finite state machine and verified by listening to the FSM that the exceptions are handled correctly. Also checked that the FSM listens to the farmers ’stop’ and ’return to start’ commands. After sending all possible node communication signals to the FSM and checking that the logic handles the state transitions as expected, this test was marked as passed successfully with a reasonable delay of less than one second. ”(VERHOOG)”

4.1.12. Test T12: Communicating to the farmer

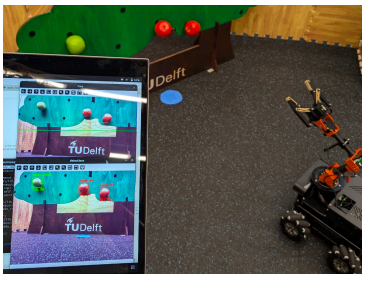
Opened the graphical user interface (GUI) (see figure A.6) and checked that the FSM state is displayed correctly at all times with a maximum delay of 5 seconds. Also verified that warning and startup noises play correctly. This was tested by sending all the different nodes feedback to the FSM and checking that the FSM state aligned with the farmer’s GUI. This was the case with a maximum delay of 1 second, and thus this test passed. ”(VERHOOG)(WELGEMOED)”

4.2. Validation results

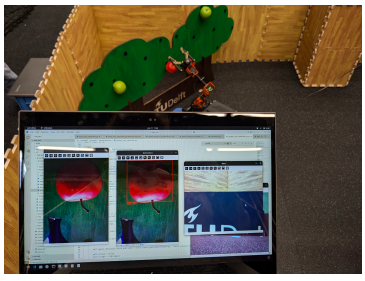
Out of the 12 conducted tests (T1–T12) listed in Tab. 4.1, the robot successfully passed 10 tests. Two tests did not fully pass due to limitations primarily linked to incomplete physical integration with the navigation stack.

The robot demonstrated consistent and reliable performance in the following areas:

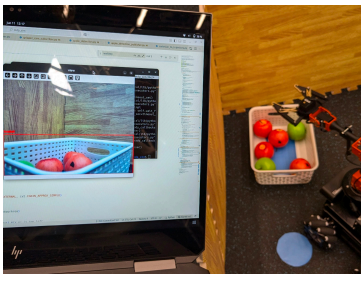
- **Startup and System Readiness (T1, T11, T12):** All system components initialized correctly, the GUI responded as expected, and state transitions occurred smoothly. This fulfilled requirements F1, F6, and F7.
- **Localization and State Awareness (T2, T7):** The Monte Carlo particle filter achieved accurate localization with an error of less than 5 cm. The basket position estimation was reasonably precise, demonstrating situational awareness. Requirement F1 was satisfied.
- **Camera Perception and Object Detection (T4, T5, T7):** Apples were consistently detected with over 80% accuracy within 2 meters. The relative position estimates for apples and the basket had low average errors, meeting the F3 requirement.
- **Manipulation (T6, T8, T9):** Apple picking, placing, and arm retraction back to the base had a success rate above 85%, reliably satisfying requirements F4 and F5.



(a) Apple detection through base camera

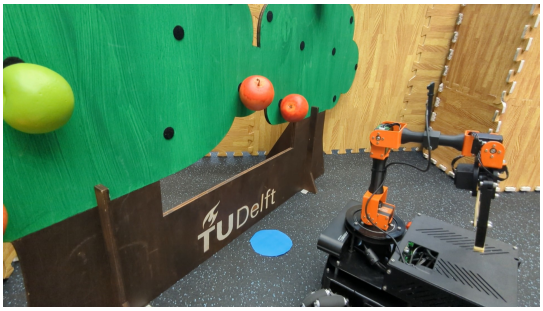


(b) Apple detection through gripper camera

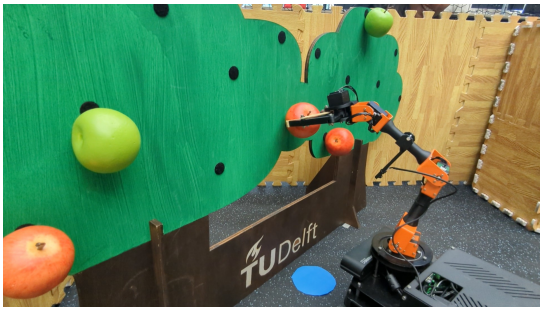


(c) Basket detection through base camera

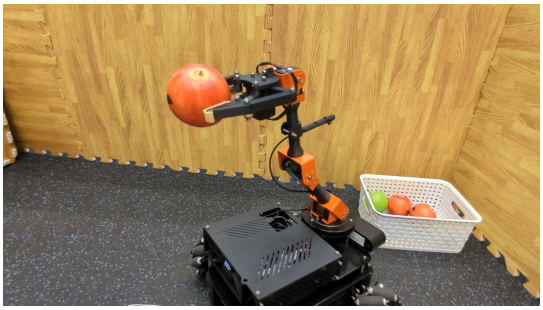
Figure 4.1: Examples of MIRTE detecting ripe/unripe apples and the basket



(a) Rest position



(b) Apple picking pose



(c) Apple holding pose

(d) Apple placing pose (before open gripper)

Figure 4.2: MIRTE's various poses throughout an apple-picking cycle "(SERRANO RUBER)"

The following limitations were observed during testing:

- **Autonomous Navigation (T3):** While navigation performed well in simulation, it could not be executed due to incomplete integration of the nav2 stack with the MIRTE. As a result, requirement F2 was not met.
- **Dynamic Obstacle Avoidance (T10):** The robot detected unexpected obstacles and triggered audible alerts. Although the GUI indicated that replanning was attempted, the robot could not navigate around the obstacle or resume its path, making the overall behavior insufficient for a full pass.

Overall, the robot met all of its core functional requirements except for F2 (autonomous navigation), demonstrating strong performance in perception, manipulation, and system robustness. "(AGARWAL)"

4.3. Software changes

Based on the Validation Results, no major changes were required for most of the modules. These components consistently performed as expected in the demo environment. However, the navigation-related functionalities did not pass the real-world tests. While the navigation worked well in the simulation, the Mirte was unable to execute the navigation goals using nav2 actions.

Given the limited time and the complexity of fully integrating the nav2 planner and controller with Mirte, we opted for the teleop keys for moving around the demo environment. These tests highlight clear areas for future improvement, such as implementing a minimal action server to interface with nav2 or replacing the planner and controller plugins to suit Mirte's integration. "(AGARWAL)"

5

Project Conclusions

This chapter summarizes the final conclusions of the project, reflecting on the robot's performance, its operational capabilities, and its limitations. After an intensive development and testing phase, we conclude that the robot successfully fulfilled the majority of the mandatory requirements, particularly those related to perception, control, and communication. However, several challenges emerged during physical deployment, especially regarding navigation. These insights help define the system's operational boundaries and point toward clear directions for future development. ”(RADUCANU)”

5.1. Concluding remarks

Newton's Pickers' autonomous apple-picking robot is designed to address key challenges in modern orchard management, such as labor shortages, rising operational costs, and inconsistencies in manual harvesting. By integrating perception, planning, and control modules within a robust and modular ROS-based software stack, we developed a system capable of autonomously navigating through an orchard, detecting ripe apples, and picking them with precision and care. The robot is also equipped with safety features such as emergency stop, obstacle detection, and live monitoring via a web-based interface. Most of the mandatory requirements were successfully met, with partial success on optional goals, demonstrating the feasibility of the solution for real-world agricultural deployment. ”(RADUCANU)”

Beyond automation, the robot delivers substantial value to clients. It operates continuously for the duration of its battery life without the need for constant human supervision, and ensures consistent picking quality. This reduces food waste compared to manual labor due to overcoming of the labor shortages that lead to unpicked apples. Additionally, the intuitive web interface provides real-time status updates, manual overrides, and system health diagnostics. Its modular architecture, organized into multiple nodes, allows for straightforward maintenance and upgrade ability. With these advantages, our robotic platform positions itself as a practical and sustainable alternative to traditional fruit-picking practices. ”(RADUCANU)” ”(AGARWAL)”

5.2. Deployment steps

Deploying the apple-picking robot involves a combination of hardware setup and software initialization. The following high-level steps provide a clear overview of the deployment process:
Initial setup:

- Run the map creation utility to generate a digital map of the orchard and annotate tree and basket positions using simple commands.
- Train or adapt the apple detection model/basket detection if the visual appearance of those differs significantly from the training dataset.

Operational deployment:

- Load the orchard map and predefined positions into the robot.
- Ensure that baskets are placed at the designated collection points.
- Start the robot and connect with it via WiFi
- Launch the full solution launch file.

Use the GUI to initiate the picking routine. The robot will navigate, identify apples, pick them, and place them in the baskets autonomously.
These steps ensure that the system can be deployed with minimal seasonal overhead and without the need for frequent retraining or reconfiguration. The deployment process is streamlined for ease of use by agricultural workers and farm operators. ”(RADUCANU)”

5.3. Operational design domain

The robot was developed and validated within a 2 × 3 meter indoor testing arena under controlled conditions. From these trials, the following operational constraints and assumptions were established for reliable deployment:

- **Smooth, flat ground is essential.** The robot relies on twist-based motion for obstacle avoidance and precise localization. High-friction surfaces, such as carpet, introduce too much resistance, disrupting this behavior. Since the MIRTE robot lacks lateral sonar sensors, its obstacle avoidance strategy—based on alternating straight and twist motions—is only viable on low-resistance surfaces.
- **Apple visibility and position are critical.** Apples must hang freely within the field of view of the cameras and should not be occluded by branches or leaves. The perception system is not robust to partial visibility, which is common in real-world orchard environments.
- **Tree and fruit characteristics must match the training data.** The apple detection model was trained on green trees with red or green apples. It does not generalize well to other types of trees or fruits, and it may falsely classify other objects as apples if they resemble those seen in training.
- **Localisation required structured environments.** The Monte Carlo particle filter used for localization assumes known static maps and reliable ground truth from initial position estimation. It is not designed to handle dynamic or GPS-denied outdoor environments without adaptation.
- **Navigation is intended for sparse, static scenes.** The robot’s navigation system is designed for environments with low obstacle density and minimal dynamic changes. Complex or crowded areas are outside the current design domain.

These constraints define the robot’s operational design domain (ODD). Within these conditions, the robot can operate safely and effectively. Operation outside this domain requires further development or manual supervision. ”(SERRANO RUBER)” ”(RADUCANU)” ”(AGARWAL)”

5.4. Known issues

Despite the successful demonstration of core functionalities, several known issues remain unresolved. These represent opportunities for future work:

- **Unreliable connectivity.** Communication between the robot and the GUI over WiFi was sometimes unstable, affecting responsiveness and monitoring reliability.
- **Dropped apples go undetected.** If the apple falls out of the gripper during transport, the robot completes the placement action regardless. There is no feedback mechanism to detect or correct this failure.
- **Weak gripper design.** The current gripper is a basic two-finger design, which is not well-suited for gripping soft or variably shaped apples. A 3-point or adaptive gripper would likely provide better grip stability and reduce the rate of dropped apples.
- **Limited detection of occluded apples.** Apples that are partially hidden by leaves or branches are often missed by the vision system. This makes detection unreliable in more realistic or complex tree configurations.
- **Inaccurate depth estimation in complex trees.** Our current system estimates the depth of apples using sonar readings based on a flat reference. In practice, trees are 3D structures with varying depth, making this method inaccurate. A more reliable approach would combine camera data with LiDAR to generate a 3D point cloud and localize apples via 3D bounding boxes.
- **Incomplete navigation implementation.** As shown in Tests T3 and T10, the robot could not navigate to goal locations in the real world. Although simulation results were promising, the navigation stack needs further tuning and integration with MIRTE to function properly in practice.

Addressing these known issues would significantly improve the system’s robustness and bring it closer to real-world deployment readiness. ”(SERRANO RUBER)” ”(RADUCANU)”

A

Appendix 1

A.1. Meet the team

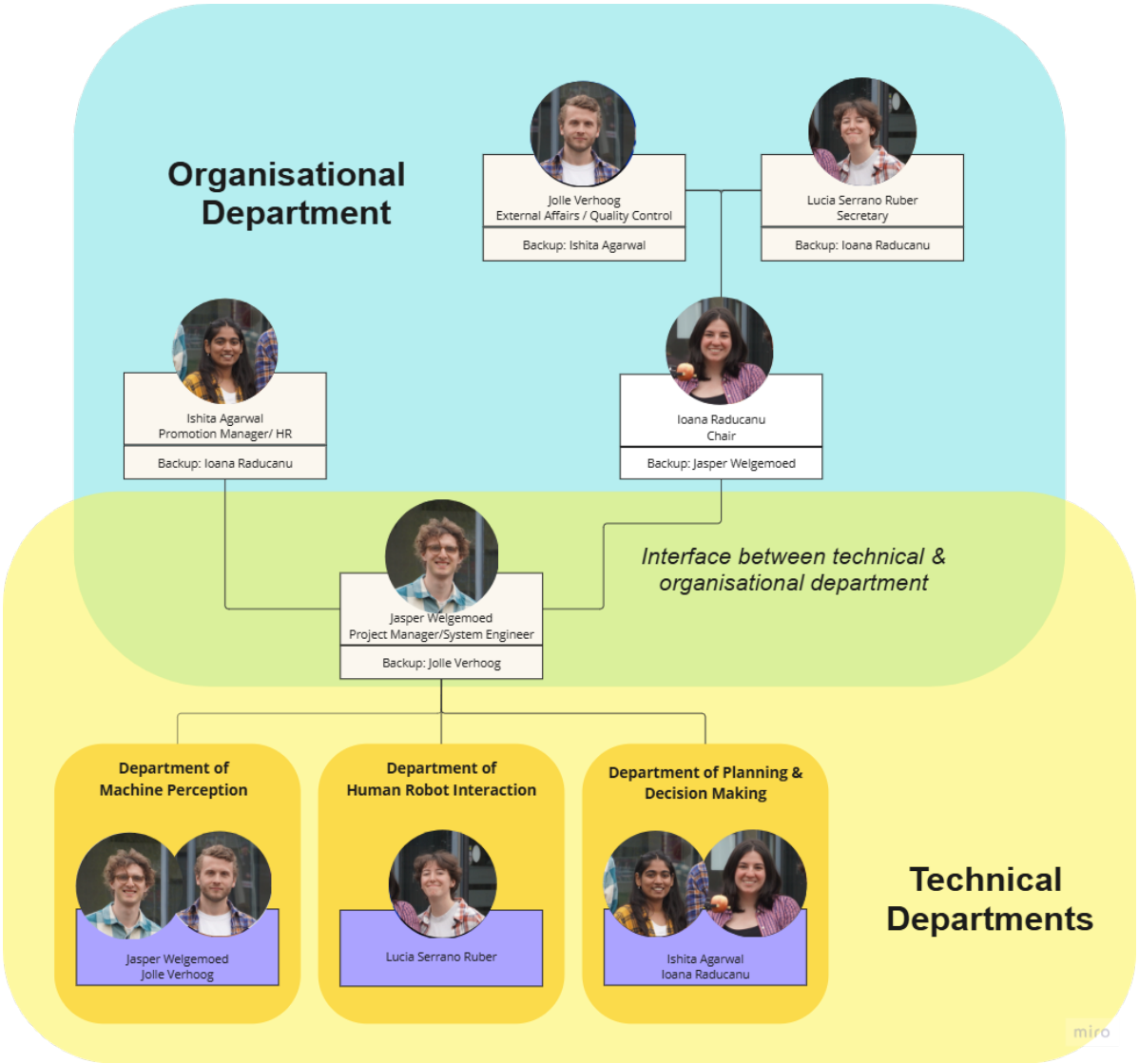


Figure A.1: Organogram ”(RADUCANU)”

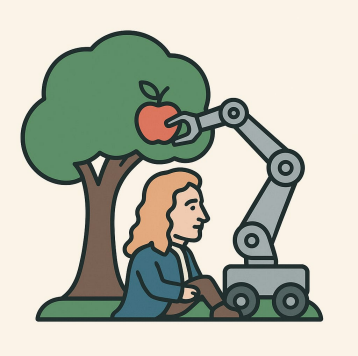


Figure A.2: Team Logo ”(WELGEMOED)”

A.2. Requirements

A.2.1. Functional requirements

The functional, high-level requirements can be found in Table A.1.

Reference	Short description	Specialization
FUNC-1	The system shall autonomously detect, classify, and collect ripe apples from the orchard.	MP/PC
FUNC-2	The system shall work cooperatively in the orchard environment.	HRI

Table A.1: List of functional requirements in this project. ”(WELGEMOED)” ”(AGARWAL)”

A.2.2. Design requirements

The design, mid-level requirements can be found in Table A.2.

Reference	Short description	Specialization
DES-1.1	The system shall grip and hold apples securely.	PC
DES-1.2	The system shall navigate the gripper to a set goal position.	PC
DES-1.3	The system shall drop picked apples into a basket.	PC
DES-2.1	The system shall navigate the orchard autonomously.	PC
DES-2.2	The system shall avoid static obstacles.	PC
DES-2.3	The system shall avoid dynamic obstacles.	PC
DES-2.4	The system shall localize the selling basket.	MP
DES-3.1	The system shall localize ripe apples on a tree.	MP
DES-3.2	The system shall classify ripe and unripe apples in an orchard environment.	MP
DES-3.3	The system shall localize itself within the orchard.	MP
DES-3.4	The system shall generate and dynamically update an environmental map.	MP
DES-4.1	The system shall communicate data about the orchard.	HRI
DES-4.2	The system shall communicate data about its state.	HRI
DES-4.3	The system shall listen to the commands of the farmer.	HRI
DES-4.4	The system shall have failure detection and recovery mechanisms.	HRI

Table A.2: List of design requirements in this project. ”(WELGEMOED)” ”(AGARWAL)”

A.2.3. Performance requirements

The performance, low-level requirements can be found in Table A.3.

Reference	Short description	Specialization
PERF-1.1	The system shall localize ripe apples with an accuracy of at least **TBD%** detection rate and no more than **TBD%** false positives.	MP
PERF-1.2	The system shall classify the ripeness of apples with an accuracy of at least **TBD%** within **TBD seconds** per apple.	MP
PERF-1.3	The system shall localize the selling basket with an error of no more than **TBD cm**.	MP
PERF-1.4	The system shall perform self-localization within a positional error range of **TBD cm** to **TBD cm**.	MP
PERF-1.5	The system shall create and maintain a map of the environment with a spatial accuracy of **TBD cm** and update its state at least **TBD Hz**.	MP
PERF-2.1	The system shall achieve at least **TBD%** collision-free runs in the orchard environment.	PC
PERF-2.2	The system shall avoid at least **TBD%** of static obstacles during operation.	PC
PERF-2.3	The system shall avoid at least **TBD%** of dynamic obstacles during operation.	PC
PERF-3.1	The system shall not drop more than **TBD%** of all apples grabbed during picking and placing.	PC
PERF-3.2	The system shall securely store up to **TBD apples** in the basket before requiring intervention.	PC
PERF-4.1	The system shall provide status and orchard updates with a success rate of at least **TBD%**.	HRI
PERF-4.2	The system shall maintain a false alert rate of no more than **TBD%** during communication.	HRI
PERF-4.3	The system shall respond to farmer commands within **TBD seconds** of receiving the command.	HRI

Table A.3: List of performance requirements in this project. ”(WELGEMOED)” ”(AGARWAL)”

A.3. Test setup

To evaluate the robot’s performance it is necessary to define the workspace, configuration space, and any assumptions made for the design and testing phase. The workspace will be defined as R^3 as the robot will be moving in 3D space, and the configuration space as $R^2 * S^1_{\text{(wheel base)}} * S^1_{\text{(base of arm)}} * R^5_{\text{(limited movement of all joints in arm)}}$. The following assumptions further define the test setup:

Robot payload and size: The robot can accommodate the necessary payload for carrying itself and apples in the environment. It is not too big to maneuver around trees or other obstacles.

Time: Battery life is sufficient to complete the entire task the robot is assigned, even when the robot has to take a sub-optimal path to circumvent obstacles or workers.

Ground type: The terrain will be modelled as a smooth, flat surface with no slip.

Simplified orchard: The orchard will be simplified. The trees will be represented by a vertical wooden board with plastic hollow apples in either green or red attached to this board. The apples will be at predictable apple heights, without ambiguous apple colors.

Weather: The weather is assumed to be constant. The setup will be consistently clearly lit and without precipitation or wind or fog.

Reliable and available hardware: The test setup assumes that the necessary hardware is readily available and works as expected.

Farmer interaction: The farmer will not actively try to block the robot, but will also not move out of the way for it.

This test setup provides a basis for evaluating the robot's performance and assessing which requirements are met. These may be changed during the progress of the project based on testing results or unforeseen circumstances. ”(VERHOOG)”

A.4. Test Images

”(WELGEMOED)”

Figure A.3: Test Setup

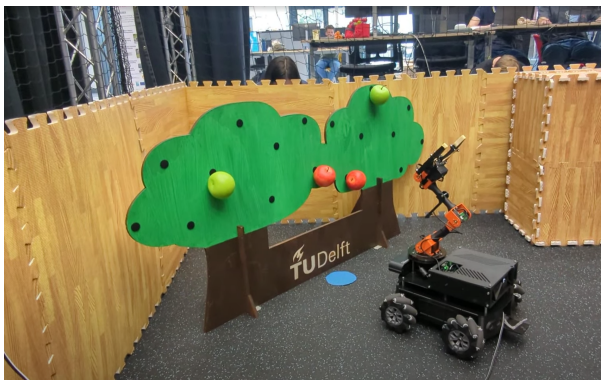


Figure A.4: Physical Test Arena

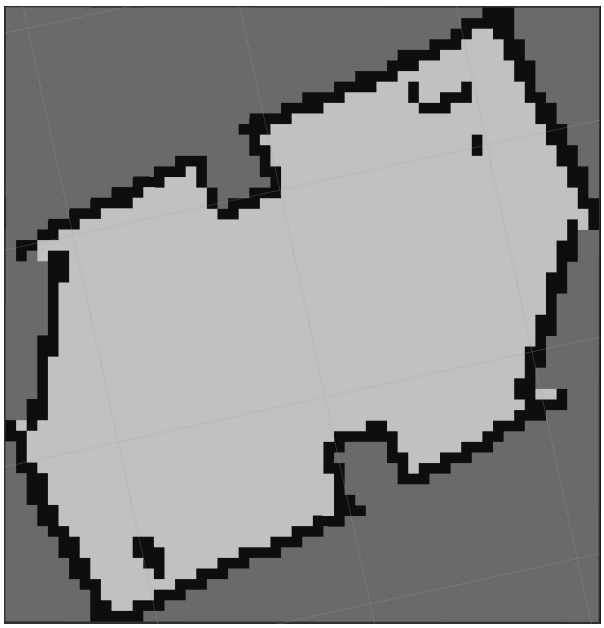


Figure A.5: Static Map

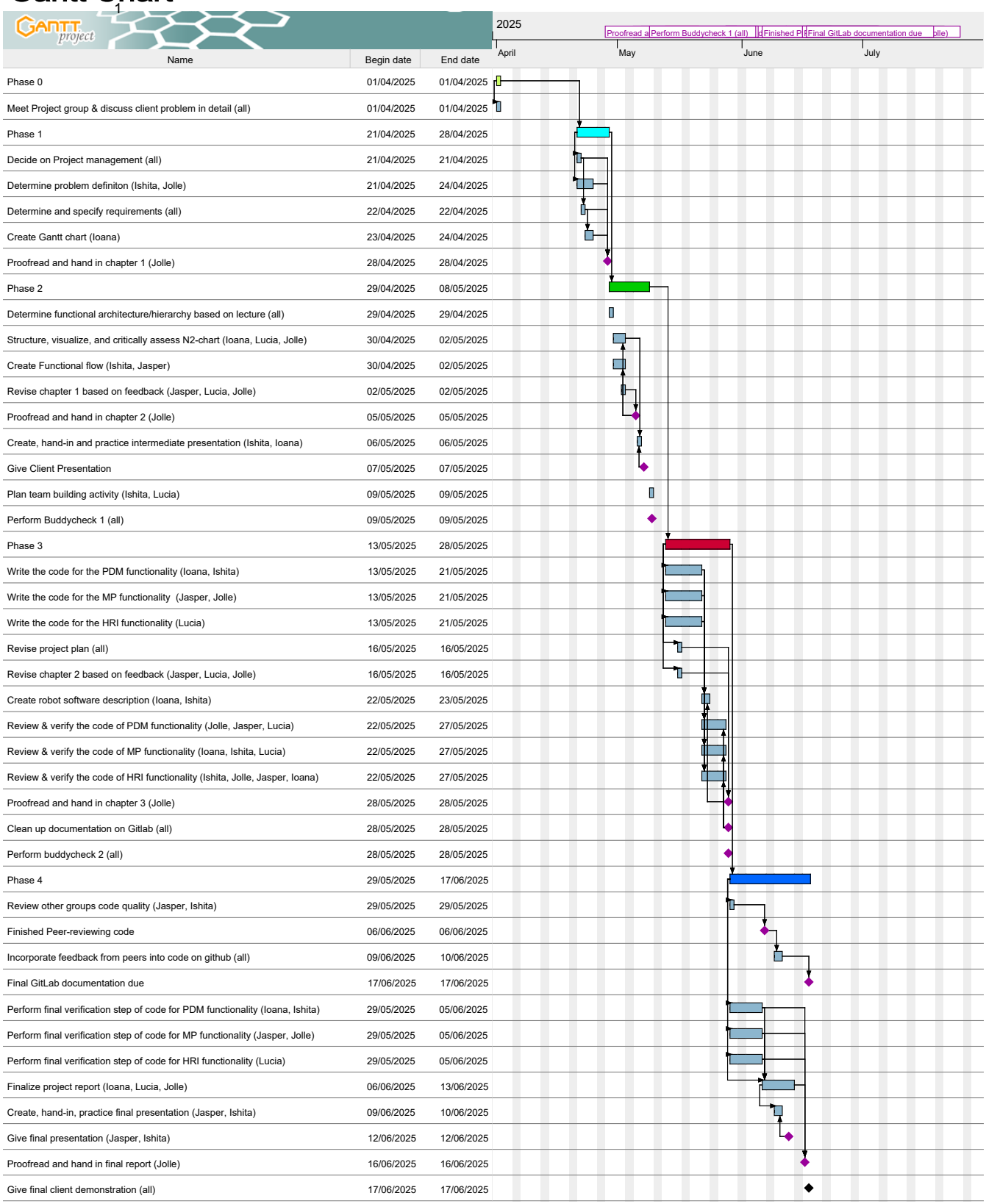
Figure A.6: Farmer Interface

B

Gantt chart

Gantt Chart

[1]



Name	Begin date	End date
Phase 0	01/04/2025	01/04/2025
Meet Project group & discuss client problem in detail (all)	01/04/2025	01/04/2025
Phase 1	21/04/2025	28/04/2025
Decide on Project management (all)	21/04/2025	21/04/2025
Determine problem definiton (Ishita, Jolle)	21/04/2025	24/04/2025
Determine and specify requirements (all)	22/04/2025	22/04/2025
Create Gantt chart (Ioana)	23/04/2025	24/04/2025
Proofread and hand in chapter 1 (Jolle)	28/04/2025	28/04/2025
Phase 2	29/04/2025	08/05/2025
Determine functional architecture/hierarchy based on lecture (all)	29/04/2025	29/04/2025
Structure, visualize, and critically assess N2-chart (Ioana, Lucia, Jolle)	30/04/2025	02/05/2025
Create Functional flow (Ishita, Jasper)	30/04/2025	02/05/2025
Revise chapter 1 based on feedback (Jasper, Lucia, Jolle)	02/05/2025	02/05/2025
Proofread and hand in chapter 2 (Jolle)	05/05/2025	05/05/2025
Create, hand-in and practice intermediate presentation (Ishita, Ioana)	06/05/2025	06/05/2025
Give Client Presentation	07/05/2025	07/05/2025
Plan team building activity (Ishita, Lucia)	09/05/2025	09/05/2025
Perform Buddycheck 1 (all)	09/05/2025	09/05/2025
Phase 3	13/05/2025	28/05/2025
Write the code for the PDM functionality (Ioana, Ishita)	13/05/2025	21/05/2025
Write the code for the MP functionality (Jasper, Jolle)	13/05/2025	21/05/2025
Write the code for the HRI functionality (Lucia)	13/05/2025	21/05/2025
Revise project plan (all)	16/05/2025	16/05/2025
Revise chapter 2 based on feedback (Jasper, Lucia, Jolle)	16/05/2025	16/05/2025
Create robot software description (Ioana, Ishita)	22/05/2025	23/05/2025
Review & verify the code of PDM functionality (Jolle, Jasper, Lucia)	22/05/2025	27/05/2025
Review & verify the code of MP functionality (Ioana, Ishita, Lucia)	22/05/2025	27/05/2025
Review & verify the code of HRI functionality (Ishita, Jolle, Jasper, Ioana)	22/05/2025	27/05/2025
Proofread and hand in chapter 3 (Jolle)	28/05/2025	28/05/2025
Clean up documentation on Gitlab (all)	28/05/2025	28/05/2025
Perform buddycheck 2 (all)	28/05/2025	28/05/2025
Phase 4	29/05/2025	17/06/2025
Review other groups code quality (Jasper, Ishita)	29/05/2025	29/05/2025
Finished Peer-reviewing code	06/06/2025	06/06/2025
Incorporate feedback from peers into code on github (all)	09/06/2025	10/06/2025
Final GitLab documentation due	17/06/2025	17/06/2025
Perform final verification step of code for PDM functionality (Ioana, Ishita)	29/05/2025	05/06/2025
Perform final verification step of code for MP functionality (Jasper, Jolle)	29/05/2025	05/06/2025
Perform final verification step of code for HRI functionality (Lucia)	29/05/2025	05/06/2025
Finalize project report (Ioana, Lucia, Jolle)	06/06/2025	13/06/2025
Create, hand-in, practice final presentation (Jasper, Ishita)	09/06/2025	10/06/2025
Give final presentation (Jasper, Ishita)	12/06/2025	12/06/2025
Proofread and hand in final report (Jolle)	16/06/2025	16/06/2025
Give final client demonstration (all)	17/06/2025	17/06/2025

[1] Gantt Chart for Newton's Pickers "(RADUCANU)"

Bibliography

[1] KAS-Lab. *mirte_navigation: Navigation stack for the Mirte robot (physical_robot branch)*. GitHub repository, https://github.com/kas-lab/mirte_navigation/tree/physical_robot. Accessed: 27 May 2025. 2025.

[2] Steve Macenski. *Slam Toolbox for lifelong mapping and localization in potentially massive maps with ROS*. GitHub repository, https://github.com/SteveMacenski/slam_toolbox. Accessed: 27 May 2025. 2025.

[3] mirte-robot. *mirte-ros-packages: ROS packages for the Mirte robot (develop branch)*. GitHub repository, https://github.com/mirte-robot/mirte-ros-packages/tree/develop. Accessed: 27 May 2025. 2025.

[4] Open Robotics. *ROS 2 Documentation: Humble Hawksbill*. https://docs.ros.org/en/humble/index.html. Accessed: 27 May 2025; distribution: Humble Hawksbill. 2025.

[5] OpenCV team. *OpenCV – Open Computer Vision Library*. https://opencv.org/. Accessed: 27 May 2025. 2025.

[6] Python Software Foundation. *tkinter — Python interface to Tcl/Tk*. https://docs.python.org/3/library/tkinter.html. Accessed: 27 May 2025; Python 3.13.3 documentation. 2025.

[7] Cory S. Sheffield, Hien T. Ngo, and Nadine Azzu. *A Manual on Apple Pollination*. Pollination Services for Sustainable Agriculture series. Rome: Food and Agriculture Organization of the United Nations, 2016. ISBN: 978-92-5-109171-5. URL: https://www.fao.org/3/i5527e/i5527e.pdf.

[8] Ultralytics Inc. *Ultralytics | Revolutionizing the World of Vision AI*. https://www.ultralytics.com/. Accessed: 27 May 2025. 2025.