AUTONOMOUS POLYCULTURE FARMING ROBOT

by

Yavuzhan Erdem

B.S., Mechatronics Engineering, Marmara University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering

Boğaziçi University

2025

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my thesis advisor, Prof. H. Işıl Bozma, who enabled me to work on this project, always supported me, contributed to my development, and inspired me with her dedication, problem-solving approach in challenging situations, and patience.

I also extend my heartfelt thanks to Prof. Yağmur Denizhan and Prof. Hakan Temeltaş for being part of my thesis jury and for their valuable contributions.

Special thanks and love to Semanur Kumaş, Arif Yılmaz, Serhat İşcan, Süleyman Batuhan Vatan, Ahmet Harun Eker, Yunus AS, Görkem Şimşek, Bartu Durmaz, Aybars Safi and the entire ISL family for their support.

I am also deeply grateful to my colleagues in the Baykar Simulation Systems Development team, with whom I had the pleasure of working during my master's studies alongside my professional career.

I owe endless gratitude to my parents and all my family members who have always stood by me and supported me unconditionally.

Finally, I would like to express my heartfelt thanks to my dear wife, Senanur, who stayed by my side during my master's studies, endured the times I was absent, and supported me in every challenging situation, and to my beloved son, Kerem, who brings joy and energy to my life.

This work has been supported in part by ROYAL CB SBB 2019K12-149250.

ABSTRACT

AUTONOMOUS POLYCULTURE FARMING ROBOT

This thesis focuses on the design and development of a three-axis Cartesian robot, TarRob, that has multiple behavioral capabilities and thus can accomplish multiple different tasks associated with the farming of a polyculture garden. In this perspective, the contributions of the thesis are three-fold: First, the robot's hardware and software are overhauled to resolve related problems of the previously developed robot system, as well as forming a robust framework that expands its behavioral capabilities and tasks. Following this, the robot is endowed with the ability to generate a three-dimensional topological garden map. This map is constructed based on the RGB-D data that is acquired by the robot. Finally, The 3DW-APF Navigation Algorithm is proposed as a novel navigation algorithm that enables the movement of the robot among the plants. This algorithm is based on an extension of artificial potential functions to a three-dimensional workspace - considering plants modeled as elliptic cylinders. It is then tested in b. Experimental results from simulations using real RGB-D data as well as on-robot tests both demonstrate that TarRob is able to move safely and effectively among plants even as their canopy foliage and heights grow over time.

ÖZET

OTONOM POLİKÜLTÜR TARIM ROBOTU

Bu tez, çoklu davranışsal yeteneklere sahip ve bu nedenle bir polikültür bahçesinin tarımıyla ilgili birden fazla farklı görevi yerine getirebilen üç eksenli Kartezyen robot TarRob'un tasarım ve geliştirilmesine odaklanmaktadır. Bu bakış açısıyla, tezin katkıları üç ana başlıkta toplanabilir. Birincisi, robotun donanım ve yazılımı, daha önce geliştirilmiş robot sisteminde karşılaşılan ilgili sorunları çözmek ve davranışsal yetenekleri ile görevlerini genişletmeyi mümkün kılacak sağlam bir altyapı oluşturmak amacıyla yenilenmiştir. İkincisi, robota üç boyutlu bir topolojik bahçe haritası oluşturma yeteneği kazandırılmıştır. Bu harita, robot tarafından elde edilen RGB-D verileri temel alınarak oluşturulmuştur.Son olarak, robotun bitkiler arasında hareket etmesini sağlayan yeni bir navigasyon algoritması önerilmiştir. Bu algoritma, yapay potansiyel fonksiyonların üç boyutlu çalışma alanına genişletilmesine dayanmaktadır ve engeller eliptik silindirler olarak modellenmiştir. 3DW-APF Navigasyon Algoritması, gerçek RGB-D verileri kullanılarak bir simülasyon ortamında ve TarRob üzerinde test edilmiştir. Performans sonuçları, TarRob'un bitkiler arasında güvenli ve etkili bir şekilde hareket edebildiğini ve zamanla bitkilerin yaprak örtüsü ile yükseklikleri artsa bile başarılı olduğunu göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

c	Robot pose
d_{min}, d_{max}	Minimum and maximum allowable depth values
d_{NN}	Nearest valid neighbor's depth value
E_i	Crossection of ith Ellipse
F^R	Global Coordinate Frame
f_x, f_y	Focal lengths of the camera in x and y directions
F^Z	ZED Camera Coordinate Frame
g	Goal position
g_{new}	Intermediate target point
H	Homogeneous Matrix
H_i	ith cylinder
I_D	Depth image
I_{RGB}	RGB image
k	Attraction parameter of APF
L_i	Height of ith obstacle
o	Obstacle
$\mathcal{O}$	Obstacles
$\mathcal{P}$	Global point cloud map
P^C	Points in the camera coordinate system
P^R	Points in the robot coordinate system
R	Rotation Matrix
T	Translation Matrix
ρ_i	Radius of ith obstacle
ρ_{i1}, ρ_{i2}	Semi-major and semi-minor axes of the ellipsoid
ρ_r	Radius of the robot
θ_i	Yaw of ith obstacle
ϵ_g, ϵ_o	Error tolerances (e.g., global and obstacle)

ϵ_s	Minimum displacement command to send to stepper motors
ϵ_z	Offset value added through the Z axis at local minima
$\mathcal{B}_{\mathcal{O}}$	Repulsion function of APF
Γ_g	Attraction function of APF
V_x^*, V_y^*, V_z^*	Maximum speed along each axis
$\mathbb{R}^3$	3D Euclidean space

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
APF	Artificial Potential Function
APF-RL	Artificial Potential Functions with Reinforcement Learning
GPU	Graphics Processing Unit
GUI	Graphical User Interface
Inf	Infinity
KD-Tree	K-Dimensional Tree
MARS	Modular Agricultural Robotic System
Nan	Not a Number
OpenCV	Open Source Computer Vision Library
PCL	Point Cloud Library
RAMPS	RepRap Arduino Mega Pololu Shield
RGB	Red Green Blue
RGB-D	Red Green Blue and Depth
ROS	Robot Operating System
UAVs	Unmanned Aerial Vehicles
UGVs	Unmanned Ground Vehicles
YOLO	You Only Look Once

1. INTRODUCTION

Agriculture is faced with two challenges: maintaining soil-plant health and economic feasibility. Agricultural practices such as monoculture farming and using fertilizers and pesticides have significantly affected soil preservation and public food safety. Polyculture farming differs from these practices as it dictates special soil preparation and farming various companion plants together. It has been demonstrated to have numerous benefits, such as enhanced pest control, reduced weed growth, minimized soil erosion, and improved utilization of light, water, and soil nutrients [1].

In parallel, labor shortages and rising operational costs have adversely affected the profitability of farming. This has pushed farming towards agricultural automation and robotics to ensure efficient and sustainable farming practices [2]. The agricultural robotics market is experiencing increased growth with an estimated value of 51 billion USD until 2029 [3].

The development of a robot for agricultural tasks is a complex process that requires the integration of several specialized and interrelated subsystems capable of basic behaviors as well as tasks that require both perception and navigation. The complexity of the robot is exacerbated for polyculture farming due to the varying characteristics of different plants. This thesis is focused on designing and developing a three-axis Cartesian robot, TarRob, that has multiple basic behavioral capabilities and thus can accomplish multiple tasks associated with farming a polyculture garden.

1.1. Related Literature

Agricultural robots are becoming part of high-tech farming. Various types of agricultural robots have been developed. Most robots consist of a mobile robot with a gripper robot capable of navigating among the plants and manipulating them.

They are typically endowed with sensors and computational systems that process the incoming sensory data so the robot can act on it. Comprehensive robotic system architecture Modular Agricultural Robotic System (MARS) for autonomous, multi-purpose system agricultural tasks has been presented in [4]. Some of these robots have been commercialized. For example,a rover robot is designed to assist vegetable farmers in maintaining vegetable beds free of weeds [5]. It achieves this by regularly hoeing the soil's surface, preventing small weeds from taking root. Additionally, while the robot operates in the field, it collects data. Another company has developed a series of agricultural robots that are highlighted for their ability to perform tasks such as autonomous weeding, hoeing, and data collection using advanced RTK GPS guidance systems, promoting sustainable and efficient farming practices [6].

Agricultural robots have been designed to perform various tasks, including planting, harvesting, pruning, and crop monitoring as surveyed in [7]. Various robots have been developed for different produce harvesting, such as lettuce harvesting [8] and citrus harvesting [9]. Work in [10] and [11] focused on developing an autonomous seed placement algorithm utilizing companion plant relationships, alongside proposing policies to improve plant coverage and diversity.

In order to increase accessibility to personal farming automation, the FarmBot project was introduced [12]. This is an open-source that consists of a Cartesian coordinate robot, software, and documentation. Much work has been done on this project. The automation of plant diversity selection and irrigation for polyculture farming has been studied in [13, 14].

Motivated also by the FarmBot project, TarRob has been developed as an indoor farming robot through a series of projects [15–17]. In [18], TarRob's garden has been planted via considering polyculture farming. In addition, the behavioral capabilities of TarRob have been extended to perform visual processing for plant detection, plant segmentation, and point cloud object generation.

1.2. Proposed Approach

The thesis considers the design and development of TarRob in three aspects:

- TarRob hardware and software revisions
- Three-dimensional topological garden mapping
- Navigation among plants

In the first stage, TarRob was evaluated concerning its current hardware and software. As discussed, it had been developed through a series of senior projects. This evaluation made the need to update the hardware and software evident. This meant updating mechanical and electronic components and revising the software infrastructure to have a flexible basis for programming basic behaviors and tasks built using these essential behaviors.

The second stage has focused on the design and development of generating a three-dimensional topological map of the garden. This is based on the RGB-D data obtained by the stereo camera newly installed on the robot. Point cloud data of the plants are obtained using approaches developed in [18]. These are then used to obtain three-dimensional models of the plants, which are then put together to construct the topological garden map.

The final stage has focused on the design and development of navigation among the plants. For this, a reactive approach has been developed to extend the artificial potential functions (APF) to operate in three-dimensional workspaces.

1.3. Contributions

The contributions of this thesis can be summarized as follows:

(i) The reliability of TarRob performance was increased through the modifications and updates introduced to its hardware and software: 1) Various hardware and software problems were resolved for improved reliable performance; 2) Software infrastructure was improved; 3) Behavioral capabilities and tasks were expanded to enable the robot to achieve an extensive range of farming-related tasks.

(ii) A novel three-dimensional topological garden mapping method has been developed. Differing from previous related work, this map corresponds to a more realistic garden model. This is based on constructing the plants' individual point cloud objects based on the RGB-D. Each plant is modeled in this map as an elliptic cylinder with parameter values obtained from the respective point cloud object.

(iii) A novel reactive navigation algorithm has been developed - referred to as three-dimensional workspace artificial potential function (3DW-APF) navigation. Differing from previous related work, this approach enables the robot to move in the three-dimensional space occupied by the plants. In particular, it enables TarRob to move among the plants without hitting them. In this approach, the obstacles are obtained from the three-dimensional topological map. The 3DW-APF navigation algorithm is introduced to handle situations when the robot gets trapped in a local minima.

1.4. Thesis Outline

The organization of this thesis is as follows:

- The details of hardware and software revisions are presented in Chapter 2.
- The construction of the three-dimensional topological garden map is explained in Chapter 3.

- In Chapter 4, the developed 3DW-APF navigation algorithm is presented along with experimental results.
- The thesis concludes with a summary and suggestions for future work in Chapter 5.

2. TARROB HARDWARE AND SOFTWARE REVISIONS

TarRob is an indoor farming robot. Its motion mechanism consists of a three-degree-freedom prismatic robot mounted on top of a 2 m × 0.9 m × 0.4 m garden. The actuation is done through three-step motors controlled by an Arduino card. The robot has a set of tools that can be automatically mounted and detached to pick up seeds and water at a given location. It has some additional mechanical tools such as an air pump that is used for sucking seeds and a water pump that provides water for plants. For environmental sensing, it has a camera and a humidity sensor. A Raspberry Pi processor is used as its processor. The software of the robot is based on ROS/C++. The robot is capable of moving to a given three-dimensional position through a simple motion 2D + V navigation strategy - namely, the first horizontal planar movement followed by vertical movement.

(a) Previous TarRob. (b) Revised TarRob.

Figure 2.1. TarRob evolution.

At the onset of this thesis, an assessment of the robot as shown in Figure 2.1(a) was made in order to determine whether any hardware-software revisions were necessary or not. In particular, the following problems were detected:

- Mechanical problems: Robot's movements along the X and Z axes were not smooth.
- Processing power insufficiency: Raspberry Pi and Raspberry Pi camera would fall short for segmentation, navigation, and storing data in the database. Consequently, electrical and electronic components were up hauled. This also required all the wiring to be controlled and changed as needed.
- Software additions: Robot's behavioral capabilities and tasks needed to be extended including task automation and sensory data readings.

The robot was then revised as to address these issues. The revised parts are as shown in Figure 2.2. The rest of this chapter is as follows: First, the mechanical revisions made to the system are detailed in Section 2.1. Following this, the electronic and software updates are explained in Section 2.2 and Section 2.3, respectively. The revised robot behaviors and tasks are explained in Section 2.4.

2.1. Mechanical Revisions

In order to achieve smooth motion along the axes, the following revisions were introduced:

- The two-piece aluminum profiles on the X-axis are replaced with single-piece profiles. This change resolved the centering issues and eliminated stuttering during transitions between the prior two aluminum segments.
- Furthermore, it is observed that when soil is loaded into the system's middle section, the connection from the midpoint of the X-axis to the lower frame is bent outward due to flex. To remedy this, fixtures at the midsection are strengthened.

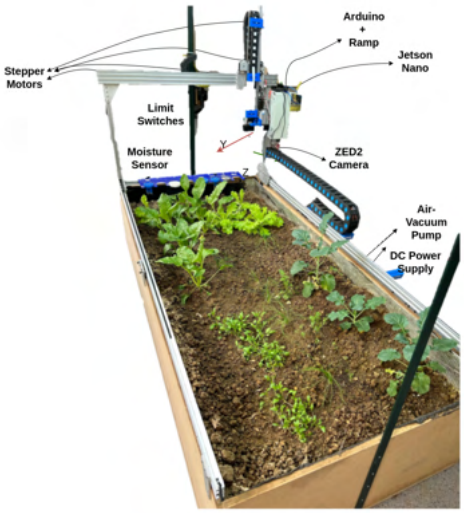


Figure 2.2. TarRob revised components.

- For the Z-axis, the jagged movement is traced to the inappropriate length and weight of the shaft due to its material. To address this, modifications are applied to the motor driver, and the shaft is replaced, effectively rectifying the issue.
- While operating the robot with stepper motors, alignment problems are detected with the pulleys and belts attached to the motor shafts. It is observed that during axes movements, the belts would occasionally rub against the aluminum profiles, creating movement resistance. To address this, a centering adjustment is made between the pulleys on the motor shafts and the endpoints where the belts are attached. This adjustment is conducted with care to prevent the belt from being trapped between the system's moving wheels or rubbing against the aluminum profiles. Moreover, since the belts previously suffered damage, the belt and pulley sets on the X-axis are replaced to finalize the process.

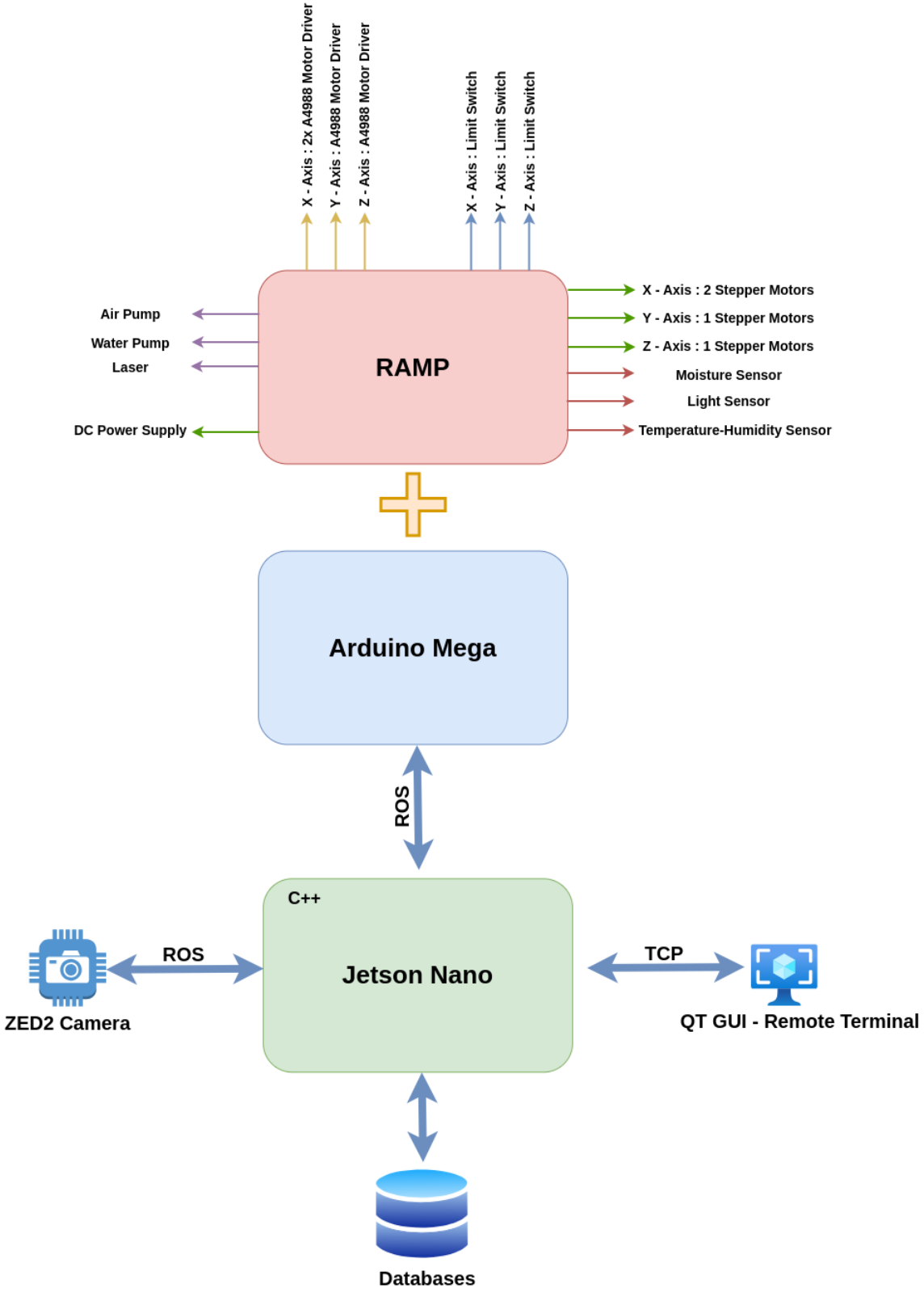


Figure 2.3. System electronic architecture.

2.2. Electrical and Electronic Components

The electrical and electronic components of the robot were overhauled as depicted in Figure 2.3. First, an update of the motion system electronics was done. In the revision, A RAMPS 1.4 Shield is integrated with the Arduino Mega [19]. This update enabled a more modular and systematic implementation of the motor drivers, limit switches, as well as connections for the vacuum and water pumps. Furthermore, it is noted that the system offers adequate infrastructure for the integration of additional sensors. Detailed information about hardware connections and pins used is explained in Appendix C.1.

In the previous system configuration, the EasyDriver Step Motor Driver Board had been used. In particular, the two stepper motors associated with the movement along the X-axis were driven by a single motor driver. In contrast, the Y and Z-axes had two separate motor drivers. It became evident that the single motor driver for the pair of stepper motors on the X-axis was not suitable. Hence, it was decided to switch to the A4988 motor driver owing to its enhanced performance in high torque applications and its compatibility with the RAMP Shield. Incorporating these new motor drivers resulted in a more modular electronic system. This change also helped mitigate stalling problems by raising the current limits of the motor drivers, thereby reducing the incidence of stalls.

The central processor Raspberry PI is replaced with an NVIDIA Jetson Nano 8GB Developer Kit to increase the robot's processing capability. In particular, its RGB camera has been changed to an RGB-D camera such as ZED-2. Furthermore, this will also enable the robot's simple motion strategy to be replaced by a more sophisticated navigation algorithm. The software setup of the Jetson Nano is done. In particular, the following software has been installed on the processor:

- Ubuntu 20.04
- ZED SDK

- ROS Noetic
- MySQL Database
- QT
- OpenCV
- Point Cloud Library (PCL).

Finally, system's connections and the associated cabling were revised. Calibration tests commenced with the new motor drivers to ensure the system's repeatability and precision. These calibration tests are crucial for achieving these qualities. During the tests, necessary adjustments are made to fine-tune the system to a precision of 1 mm. The step count necessary for the motor to move 1 mm linearly is calibrated in Arduino. Precision measurements are marked at designated intervals within the system, confirming the system's consistent ability to perform this task accurately.

2.3. Software Revisions

Substantial software additions have enhanced the robot's behaviors and automation. First, the overall software structure has been revised, as seen in Figure 2.4.

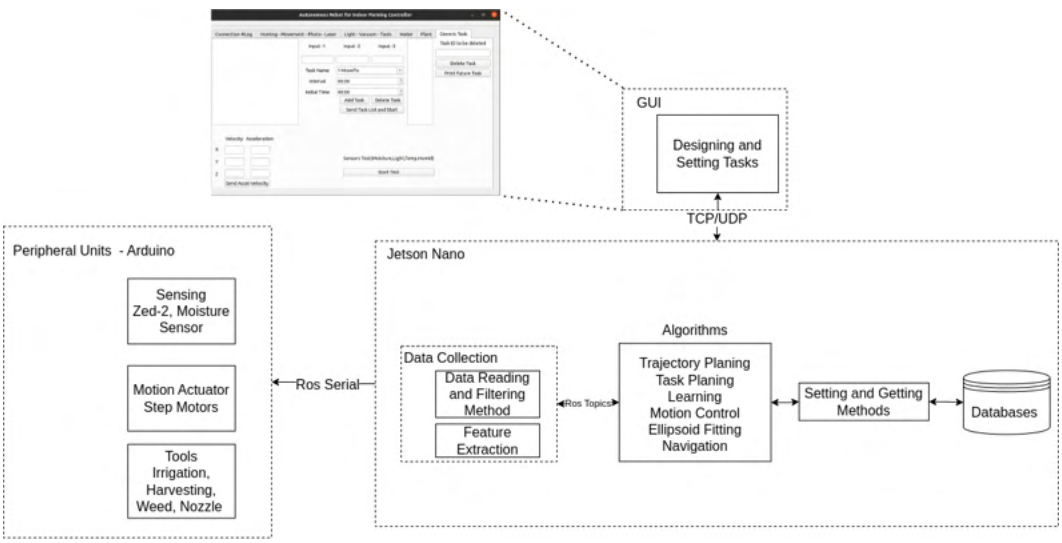


Figure 2.4. Software architecture.

2.3.1. Software Architecture

The primary system software has been upgraded to run on the Jetson Nano, ensuring that user-defined periodic and non-periodic tasks are executed through the GUI. This software controls tasks, sends commands to the electronic system, and handles movement and sensor reading activities. It also processes database read operations. The general structure of the leading software running on the Jetson Nano is shown in Figure 2.5.

The software structure is divided into two modes: autonomous and manual. In manual mode, the system performs no automatic actions or checks the planned tasks in `futureFarmDB`. In this mode, tasks to be executed autonomously are set up. The created tasks are checked and executed when the system switches to autonomous mode. In manual mode, the system waits for commands from the interface and does not perform any tasks automatically. In autonomous mode, there are two threads. The first thread is the consumer thread, which looks at the tasks in `futureFarmDB` and adds those that are due to a vector called `todotaskvec`, determining the tasks to be done. A task-consuming thread is then created to process these tasks. The tasks in the `todotaskvec` vector are executed sequentially. With this developed structure, the system ensures that it does not miss other tasks while performing a task.

Notable enhancements have been made to the software as detailed in Table 2.4. Furthermore, updates have been incorporated into the communication interface with the GUI and database operations. A threading structure has been introduced to prevent missing periodic tasks during method execution. Within this framework, the first thread carries out the primary function. In contrast, the second thread continuously oversees periodic tasks in the database. This arrangement ensures that recurring tasks will not be missed while the robot is engaged in an operation. Given the extensive use of databases in the system, a new database class has been developed to enhance software usability and modularity.

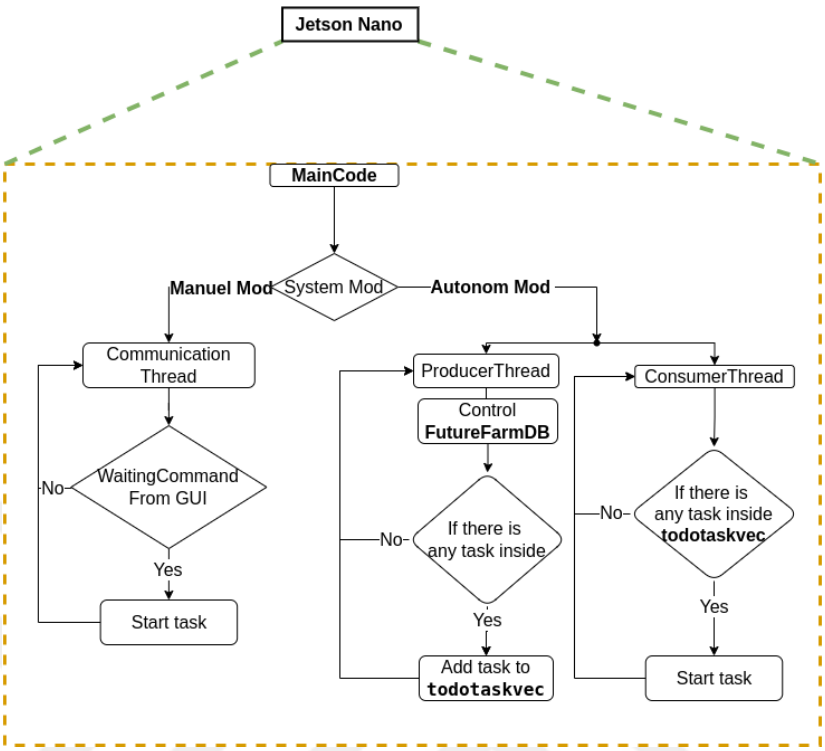


Figure 2.5. Jetson Nano main software architecture.

2.3.2. Qt Based Graphical User Interface Updates

The system utilizes a Qt-based interface for task and scenario creation, which has been significantly enhanced. As shown in Figure 2.6, a new interface has been created to allow users to construct new methods by integrating various sub-methods either periodically or sequentially. This interface provides the capability to add and organize tasks within the system according to preferred schedules and intervals, thereby enabling direct scenario testing on the system.

Additionally, test interfaces have been developed for the moisture sensor, motor acceleration, and velocity commands. These interfaces enable users to perform various tests directly through the GUI without the need to write code. The sections highlighted with green boxes in Figure 2.6 are explained in detail in Table 2.1. The remaining interfaces will be explained in C.2.3 section.

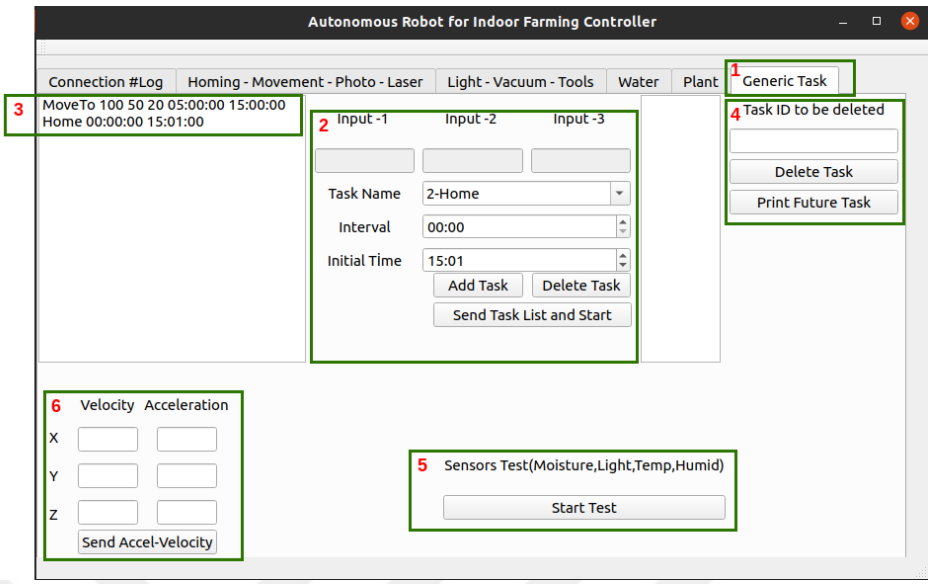


Figure 2.6. GUI - Designed generic task section.

Table 2.1. Generic task GUI explanation.

ID	Description
1	The tab added to open the Generic Task interface.
2	In this section, the desired task is selected from the "Task Name" field. If the selected task requires input, the input field becomes active. After selecting the task name, the start time of the task is set in the "Initial Time" field. If the "Interval" field is set to zero, it indicates that the task is not periodic. After the necessary adjustments are made, clicking the "Add Task" button adds the task to the list shown in section 3. You can send this task to the system by clicking the "Send Task List and Start" button.
3	The information of the added tasks can be viewed in this section.
4	Tasks added to the system can be deleted by checking their IDs in the futureFarmDB database.
5	The test button is used to test the operation of the added sensors.
6	Allows testing by changing the speed and acceleration values of the motors in the system.

2.3.3. MySQL Database Updates

Several updates have been added to the database structure used in previous studies. These updates aim to store multiple types of data related to the system's operation and the conditions of the plants. The data storage structure of databases currently used in the system is illustrated in Figure 2.7. A general explanation of these databases is provided in Table 2.2.

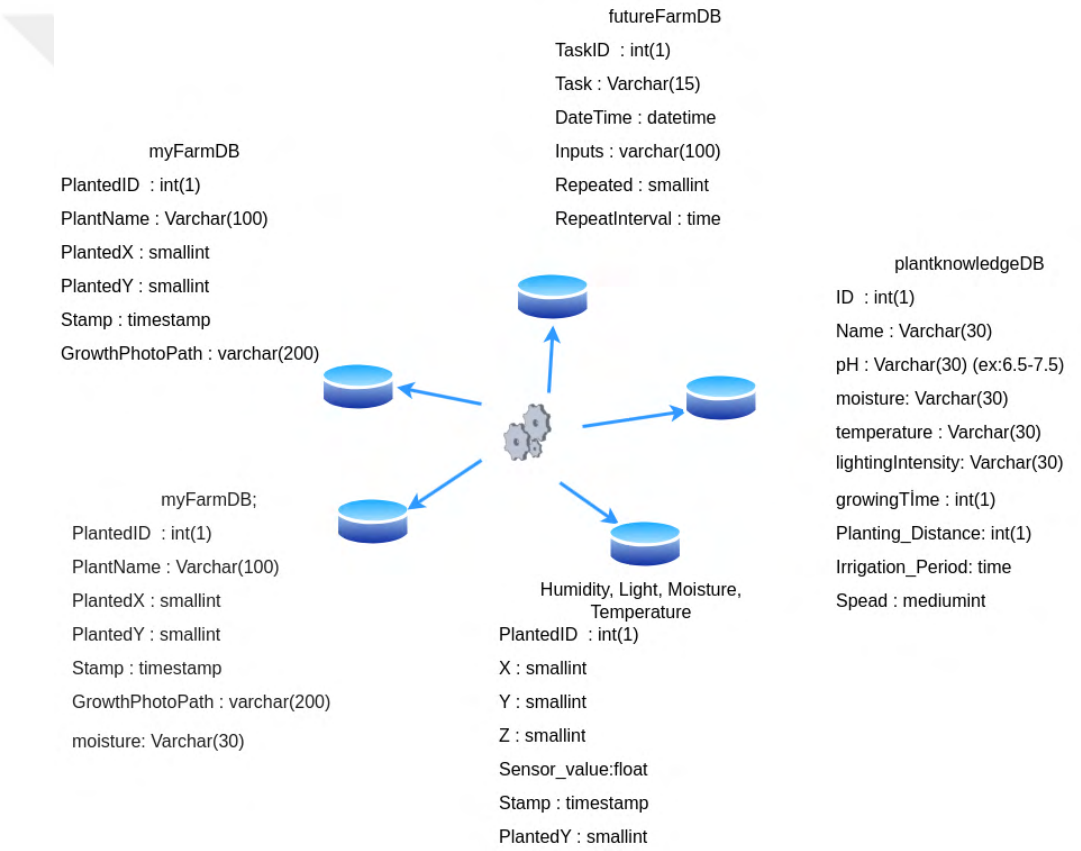


Figure 2.7. Database organization.

Table 2.2. TarRob knowledge bases.

Database Name	Description
futureFarmDB	This database stores time information, inputs, and periodic time intervals for periodic or non-periodic tasks in the system.
myFarmDB	This table contains IDs, names, and X and Y positions of the planted crops. Additionally, the directory information where the photos will be saved is also included in this table.
oplogDB	This database logs the operations performed in the system for retrospective tracking. After each operation, the performed actions are recorded in this table.
plantknowledgeDB	This table stores general information about plants. It includes details such as watering intervals, watering amounts, required temperature, and humidity for different types of plants. These details can be used to decide on the necessary operations.
humiditylevel	This table stores the values obtained from the humidity sensor along with the position where the measurement is taken. The measurement time is also recorded.
lightlevel	This table stores the values obtained from the light sensor along with the position where the measurement is taken. The measurement time is also recorded.
moisturelevel	This table stores the values obtained from the soil moisture sensor along with the position where the measurement is taken. The measurement time is also recorded.
temperaturelevel	This table stores the values obtained from the temperature sensor along with the position where the measurement is taken. The measurement time is also recorded.

2.4. Robot Behaviors and Tasks

Improvements have been made to the system in terms of behaviors and tasks to enable autonomous operation and reduce the need for human intervention. Behavioral and task revisions have been introduced to extend TarRob's usage capabilities to a broader range of farming applications.

2.4.1. Coordinate Frames

The robot's behaviors and tasks are done with respect to a set of coordinate frames as follows:

- F^R - TarRob global coordinate frame.
- F^Z - ZED camera coordinate frame.

The map between the two frames is shown in Figure 2.8 and is defined as given in Table 2.3.

Table 2.3. TarRob coordinate frames.

Robot		**Camera**	
Forward	X_R	Left	$-Y_Z$
Right	Y_R W	Forward	X_Z

Hence, the transformation between the two frames is defined in homogeneous coordinates by the matrix H_Z^R, which consists of a 3×3 rotation matrix R^RZ and a 3×1 translation vector TZ^R,

$$H_Z^R = \begin{bmatrix} R_Z^R & T_Z^R \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

where R^RZ and TZ^R are defined as:

$$R_Z^R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ and } T_Z^R = \begin{bmatrix} x_Z^R & y_Z^R & z_Z^R \end{bmatrix}. \tag{2.2}$$

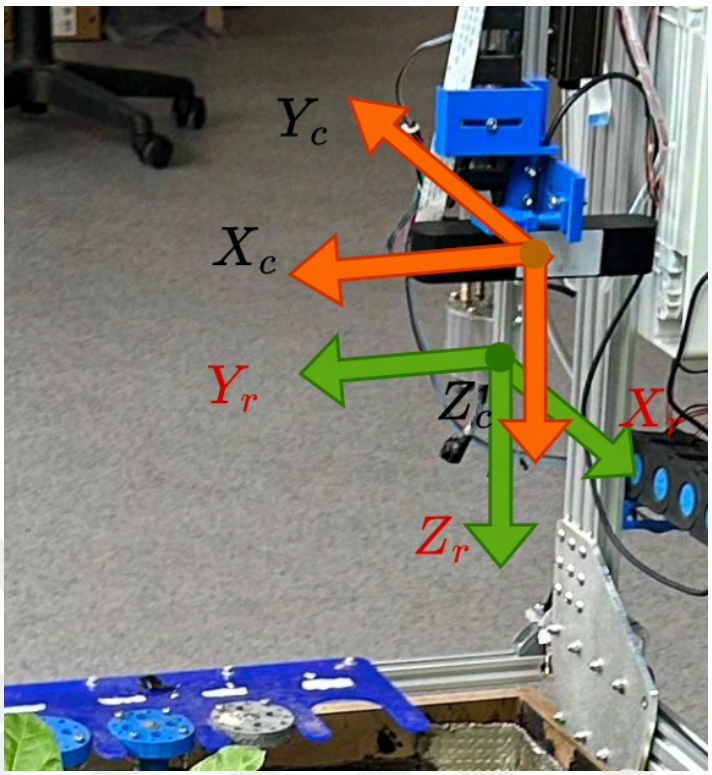


Figure 2.8. Robot and camera coordinate systems.

2.4.2. Behavioral Capabilities

The behavioral capabilities refer to basic methods that the robot can execute. The names and descriptions of the behaviors capabilities present in the existing system and developed within the scope of this project are shown in Table 2.4. Based on the behaviors specified in this table, multiple task capabilities can be added and designed.

2.4.3. Robot Tasks

Tasks are defined based on a sequential combination of behaviors or other tasks. For example, measuring moisture is a system's capability, whereas automatically determining the measurement points for each plant and performing the measurement at those points is defined as a task feature of the system. The tasks designed and developed as part of this project are listed in Table 2.5.

Table 2.4. TarRob behaviors.

Behavior	Description
TakePhoto	Takes a photo, and saves it in the respective folder by calling the GrowthPhoto function.
GrowthPhoto	Retrieves the folder path of the relevant plant from the database and saves the photo in that directory.
TakingTool	Retrieves the position of the specified tool based on its type from the database.
LeavingTool	Leaves currently attached tool based on its type from the database.
MeasureMoisture	Performs the moisture measurement.
Move	TarRob's end-effector moves to the given location.
TempHumidity	Performs the temperature and humidity measurement.
MeasureLight	Performs the light measurement.
WaterMethod	Activates the water pump.

Note that for the navigation task, two alternatives are possible:

- 2D+V Navigation: This is a simple motion algorithm in which horizontal and vertical movements are executed in a decoupled manner.
  Hence, the robot accomplishes each movement in two stages: First, it moves in the X-Y plane with $Z = 0$ to arrive at the $(g_x, g_y, 0)^T$ associated with the goal position. We will refer to this as (Two-dimensional planar motion followed by vertical motion). Following, it moves downwards to the $(g_x, g_y, g_z)^T$. For the next movement, it moves back to $(g_x, g_y, 0)^T$ and starts the next movement.
- 3DW-APF Navigation: This approach is based on 3DW-APF - namely the extension of artificial potential functions (APF) based navigation to three-dimensional workspace (3DW). The details of this task are presented in Chapter 4.

Finally, the performance of these two navigation algorithms will be compared using data obtained from the system.

Table 2.5. TarRob tasks.

Tasks	Description
TakeALLPhoto	This task moves to the position of each plant recorded in the database and calls TakePhoto behaviors.
DailyPlant	This task is designed for periodic photo-taking tasks at predefined intervals.
MeasureMoistureAll	This task moves to the position of each plant recorded in the database and performs the moisture measurement.
AutomaticWaterMethod	Activates the water pump according to the watering amount defined for the respective plant type.
PointCloudMap	Through this task, the DailyPlant method is invoked, and a point cloud map of the garden is generated as photos of each plant are captured.
TopologicalMap	This task enables the automatic creation of the garden's topological map. Using the masks of the plants, elliptical cylinder models are generated to construct the map.
Navigation	This task enables the robot's end-effector to move among the plants.

3. TIME-VARYING THREE DIMENSIONAL GARDEN MAPPING

This chapter presents the construction of a three-dimensional topological garden map. This is a compact map that consists of plant models with respect to the garden ground. Its construction is based on the collected RGB-D data. The map is time-varying since it evolves due to plants' growth states, their light-dependent movements during different times of the day, and non-directional nastic movements. The map construction is based on the periodically collected RGB-D data. Once the RGB-D data of the entire garden is collected, a point cloud map is constructed. Following this, each plant in the resulting map is modeled individually by an elliptic cylinder. These models are then put together to generate a three-dimensional garden map.

3.1. Related Work

Creating a 3D point cloud map of a garden environment is fundamental in developing autonomous agricultural systems. By capturing the spatial structure and features of the garden, point cloud mapping enables precise modeling of plants and their surroundings, forming the basis for navigation, monitoring, and task planning. Multiple representation methods, such as point cloud and voxel representation, are available to model garden environments [20]. In [21], a 3D reconstruction procedure using point cloud representation was developed to monitor crops. Three-dimensional reconstruction from images of farms and crops serves as a realistic approach for monitoring and modeling. Similarly, [22] explores the fusion of camera-based 3D models of farmland with sensor data to create a digital twin of the farmland. Another study [23] proposes a map registration pipeline called AgriColMap, which combines point cloud data obtained from cameras on unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) using data association algorithms. In [24], the 3D point cloud of Sorghum plants was utilized to extract key parameters such as height and width.

In this study [25], RGB images obtained from a UAV platform are used to estimate several vineyard characteristics, including row orientation, height, width, row spacing, canopy cover fraction, and the percentage of missing row segments. The dense point cloud is first extracted from the overlapping RGB images acquired over the vineyard.

The combination of 3D mapping and plant modeling paves the way for advanced robotic applications in polyculture farming, enabling autonomous navigation and interaction with dynamic environments. Various approaches for modeling plants using three-dimensional point cloud maps have been proposed. For instance, [26] employs a two-dimensional circle model for individual plant representation. In [14], an algorithm combining K-means and BFS-based bounding disk fitting is introduced to track plants by enclosing their canopy within the smallest possible radius. Additionally, [13] extends the circle model from [26] by incorporating a height attribute, representing each plant as a cylinder defined by its seed location, radius, and height. The work in [27] modifies the representation of obstacles by adopting ellipsoid approximations, demonstrating that models based on ellipsoids provide more realistic representations of objects. The outer boundary is derived from the ellipse fitted to the convex hull of the laser points. Work In [28], the ROMI Cable Bot is developed to perform complete 3D analyses of plants. It combines a physical scanning station equipped with an RGB camera and a robust image processing pipeline, the Plant Interpreter, to construct a 3D representation of plants. This project focuses on analyzing plants and gathering relevant data. In the study [29], a maize plant detection and mapping algorithm was developed using the FX6 3D Lidar sensor. The algorithm discriminates between ground and plants by combining the robot's movement with measurements from the lidar sensor, creating a global point cloud map. The system was tested in a 3D simulation environment using Gazebo and in real-world conditions with the Volksbot RT-3 robot.

Although data was collected in [28], no study was conducted on creating a model. In [29], point cloud segmentation was performed, and modeling was carried out on a single plant. In [13, 26], plants were modeled using a cylindrical cross-section shape, while in [27], an ellipsoid modeling approach was used.

It has been observed that ellipsoids provide a more accurate representation in object modeling. However, that study was designed for objects with a fixed height. In our study, both the height and the ellipsoid model are dynamically implemented. In this section, the process of constructing a point cloud map of all plant areas and modeling the plants in the garden as elliptic cylinders is explained in detail.

3.2. TarRob Garden Point Cloud Data Construction

The construction of the garden's point cloud data follows the flow of the processing shown in Figure 3.1. As explained, the robot moves over the garden periodically, centers on each plant, and collects RGB-D data, which is then transformed into point cloud data. The generated point cloud undergoes filtering to remove erroneous or poorly calculated results. After filtering, the data is transformed from the camera's coordinate frame to the robot's coordinate frame. As the RGB-D data is acquired systematically through the robot, moving sequentially to each plant's known location, the map is constructed by incrementally stitching the point cloud data associated with each plant into a single significant point cloud data.

Let I_{RGB} and I_D denote the RGB and depth images, respectively. The depth $d(u, v)$ of each pixel (u, v) is converted into a 3D point P^C in the camera coordinate system - based on a mapping between a pixel inside a depth image to its 3D coordinates in the camera's optical frame as defined by the camera's projection model. For a pinhole camera model, it is governed by the following set of equations:

$$P^C(u, v) = \begin{bmatrix} d(u, v)\frac{u - c_u}{f_u} \\ d(u, v)\frac{v - c_v}{f_v} \\ d(u, v) \end{bmatrix} \tag{3.1}$$

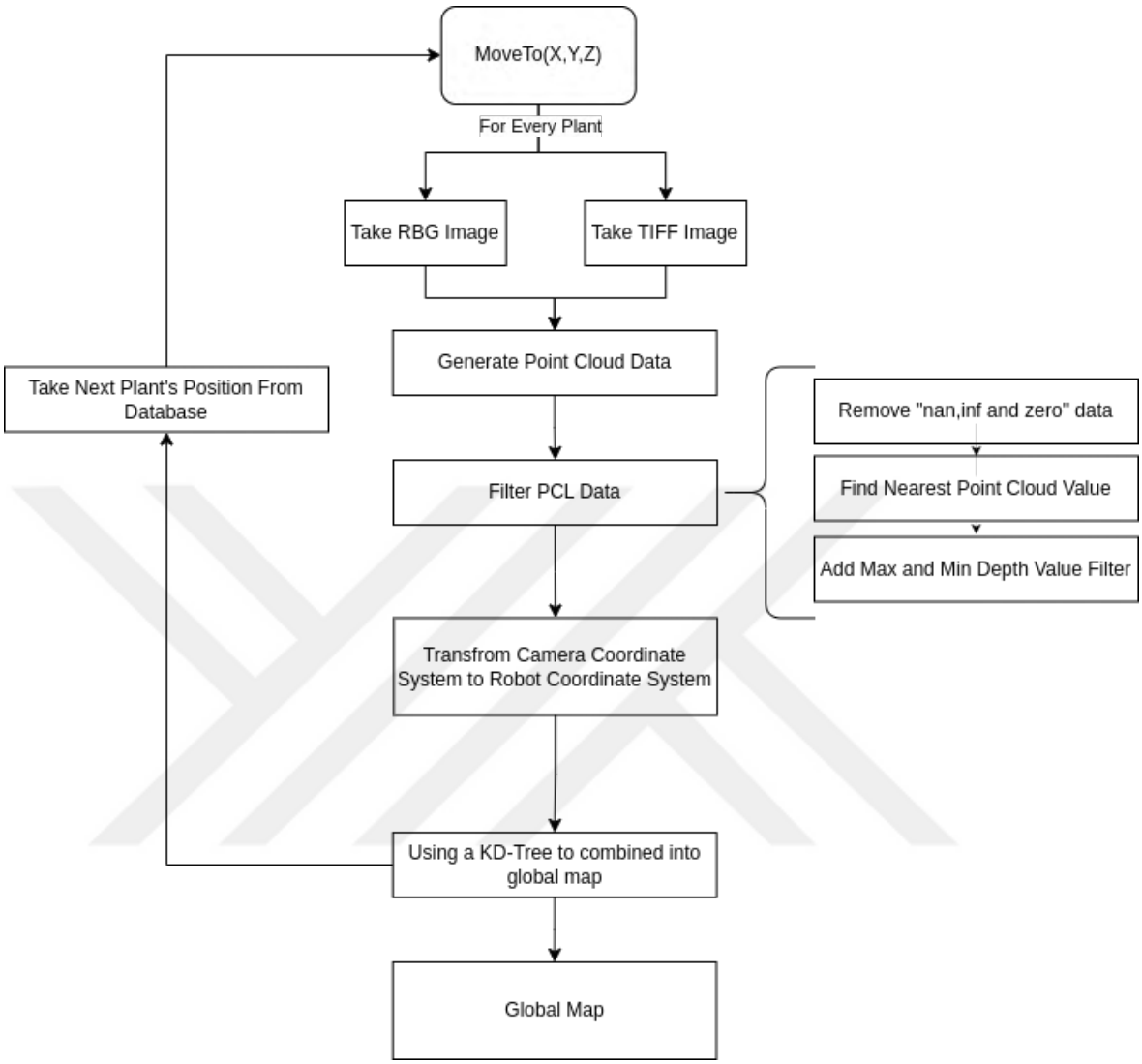


Figure 3.1. Process flow: Garden's point cloud data generation.

where (c_u, c_b) refers to the camera center in the image plane, (f_u, f_v) refers to the camera focal lengths in pixels. The raw point cloud data is observed to contain invalid values such as - NaN, Zero, and Inf. This is attributed to several factors:

- Mismatch between the two cameras on the ZED-2 system, leading to missing or invalid depth values.
- Objects, such as plants, being too close to the camera, resulting in measurement errors.

- Environmental factors, such as lighting conditions or reflections, can negatively affect depth data quality.

The point cloud data generated from each RGB-D frame is processed to handle such invalid values:

- Depth values that are NaN or Inf are replaced using the nearest neighbor's value within the point cloud.
- Depth values that fall outside the specified range $[d_{\min}, d_{\max}]$ are discarded.

The filtering process is defined as:

$$d(u,v) = \begin{cases} d_{NN}(u,v), & \text{if } d(u,v) = \text{NaN or Inf}, \\ 0, & \text{if } d(u,v) < d_{min} \text{ or } d(u,v) > d_{max} \\ d(u,v) & \text{otherwise} \end{cases} \tag{3.2}$$

where d_{min} refers to the minimum allowable depth value, d_{max} refers to the maximum allowable depth value, and $d_{NN}(u,v)$ refers to the nearest valid neighbor's depth value is calculated based on the Euclidean distance among valid points. By applying these filters, the processed point cloud contains only valid depth values within the allowable range, ensuring better reliability and reducing noise. Once all the data from the entire plant bed is aggregated, a single integrated point cloud data is generated.

Point cloud data obtained in the camera frame is then transformed to the robot frame as seen in Figure 2.8,

$$P^R = H_C^R P^C. \tag{3.3}$$

Points P^R in the robot coordinate system F^R are added to a global point cloud map. When adding a new point cloud data into the so-far-constructed point cloud map, a filtering process is applied based on a distance threshold between the existing and incoming data. This prevents redundant additions of the same point, possibly due to overlapping data from previous positions.

For this, the new point cloud data $\mathcal{P}^R(t)$ is compared against the existing global map $\mathcal{P}$. Using a KD-Tree, for each point $P \in \mathcal{P}^R(t)$, the nearest neighbor P' distance $\delta(P, P')$ exceeds a threshold ϵ - namely:

$$\mathcal{P} = \mathcal{P} \cup \{P : \forall P' \in \mathcal{P}, \delta(P, P') > \epsilon\}. \tag{3.4}$$

3.3. Plant Modeling

The next step is to model each plant individually. For this, an elliptic cylinder model has been selected. Hence, each plant i, $i = 1, \ldots, N$ is modeled by an elliptic cylinder O_i with a cross-section E_i and cylinder H_i:

$$E_i : \quad (Px - Po_i)^T A_i (Px - Po_i) \leq 1, \tag{3.5}$$

$$H_i : \quad o_{iz} - \frac{L_i}{2} \leq h_i \leq o_{iz} + \frac{L_i}{2} \tag{3.6}$$

where P is the 3×2 projection matrix onto the horizontal $X - Y$ plane:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.7}$$

and $o_i = \begin{bmatrix} o_{ix} & o_{iy} & o_{iz} \end{bmatrix}^T$ refers to cylinder center. Hence, each elliptic cylinder is defined by seven parameters. The first five encode the cross-section center (o_{ix}, o_{iy}, h_i), semi-major axis ρ_{i1}, semi-minor axis ρ_{i2}, and orientation θ_i while the height range is specified by o_{iz} and L_i. It should be remarked that this model extends the model proposed by [13] in which a plant is represented using its seed/plant location, radius, and height attribute. This abstraction is both efficient and expressive, as it allows the plant to be modeled during each development stage.

3.4. TarRob 3D Topological Map Generation Task

The flow of processing in the 3D topological map generation task is as follows: First, the robot engages in the DailyPlantTakePhoto task, which enables it to acquire RGB-D data of each plant. This data is then used to construct a 3D point cloud model of the entire garden. The construction of these models are shown in Figure 3.2.

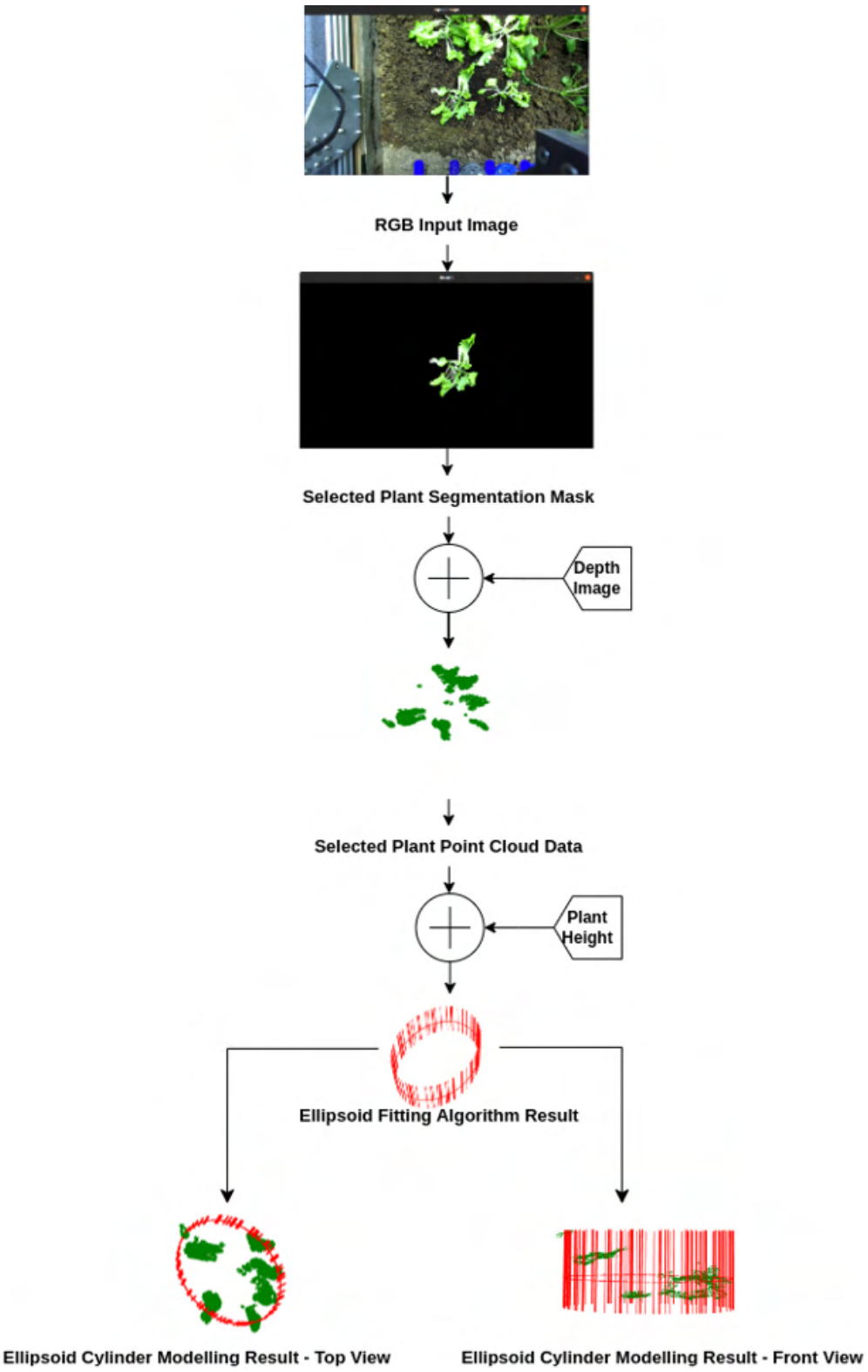


Figure 3.2. Plant model construction.

Following, elliptic cylinder models of all the plants are autonomously constructed. It is based on the point cloud data $\mathcal{P}_i$ corresponding to each plant $i = 1$. In TarRob, the point cloud data of each plant is determined autonomously from the point cloud map using the plant mask associated with each plant. The masks are obtained from RGB data based on a deep-learning based algorithm [18]. The point cloud data is then generated through augmenting the mask data with the corresponding the depth values.

The parameters of the ellipse cross-section E_i are determined using an ellipsoid fitting algorithm that is applied to the point cloud data associated with the canopy of the plant. In particular, Khachiyan's algorithm is used [30]- [31]. The details of this algorithm are given in Appendix A. The cylinder height is obtained directly from the height of the respective point cloud object. This 3D point cloud data is then processed to obtain the minimum ellipse that encapsulates the point cloud data that defines to the plant's canopy area.

Finally, a map consisting of elliptic cylinders is generated to facilitate autonomous operations, such as watering and sensor measurements, using the developed navigation algorithm.

3.5. Experiments and Results

This section describes experimental results regarding the construction of each plant's individual point cloud data as well as that of the whole garden's point cloud data and 3D topological garden map construction.

3.5.1. Point Cloud Data Verification

First, the accuracy of point cloud data of each plant is validated through running a calibration test. For this, data are collected from three different positions $(120, 180, 0), (300, 180, 0)$, and $(300, 320, 0)$ with overlapping fields of view as seen in Figure 3.3. These areas were colored red, brown, and pink, respectively.

When the point cloud data associated with each image are merged into a single point cloud data, the overlapping regions are correctly aligned, confirming the accuracy of the calibration.

3.5.2. Garden Point Cloud Data

Initially, a point cloud map of the garden was generated using RGB-D data. The robot visited positions $(120, 180, 0), (120, 350, 0), (120, 530, 0) ve (120, 700, 0)$ and performed the processes described in Figure 3.1. A point cloud data for four plants is obtained, as shown in Figure 3.4.

The steps shown in these four images are applied to all plant data collected on 2024-02-16. After merging the data, a point cloud map of the entire garden is obtained. Using the data obtained for each plant recorded in the database, the point cloud map of the entire garden has been created. The top view of the generated model is shown in Figure 3.5, while the right-side view is displayed in Figure 3.6.

3.5.3. Plant Modeling

The process of generating plant elliptic cylinder models is carried out using RGB-D data obtained from the actual system. Based on the data collected on 2024-02-25, the elliptic cylinder models for all plants are shown in Figure 3.7. The integration of the generated elliptical cylinder models into a three-dimensional garden map is illustrated in Figure 3.8.

(a) Position : (120 mm,180 mm,0 mm)

(b) Position : (300 mm,180 mm,0 mm)

(c) Position : (300 mm,320 mm,0 mm)

(d) Merged point cloud data

Figure 3.3. Verification of merging point cloud data collected at different positions. The overlapping regions are marked with red (a), brown (b), and pink (c).

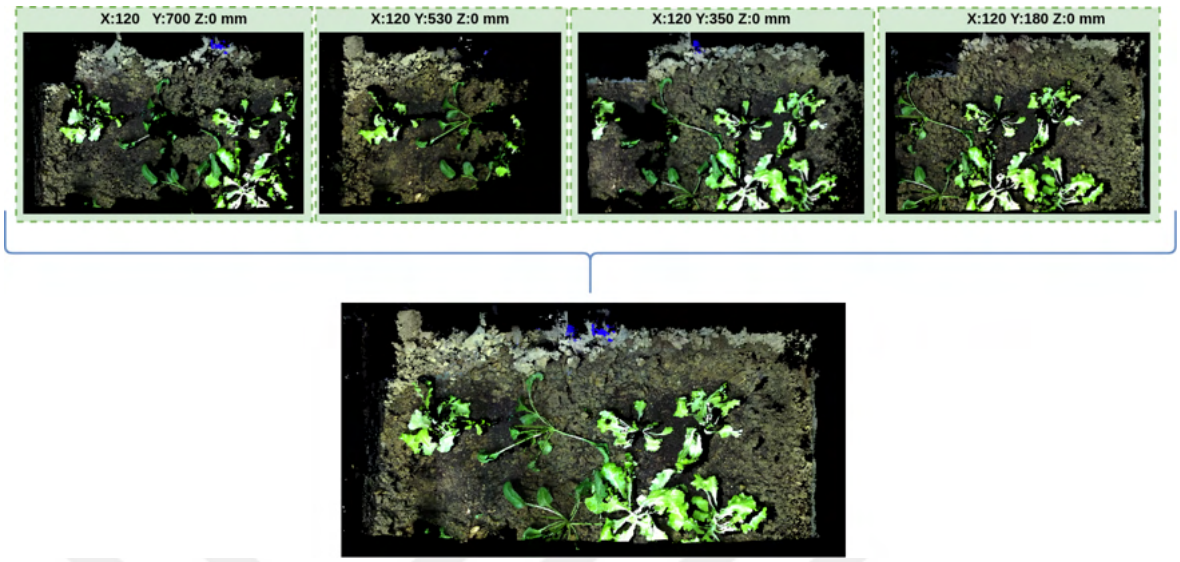


Figure 3.4. Sample garden point cloud data construction.

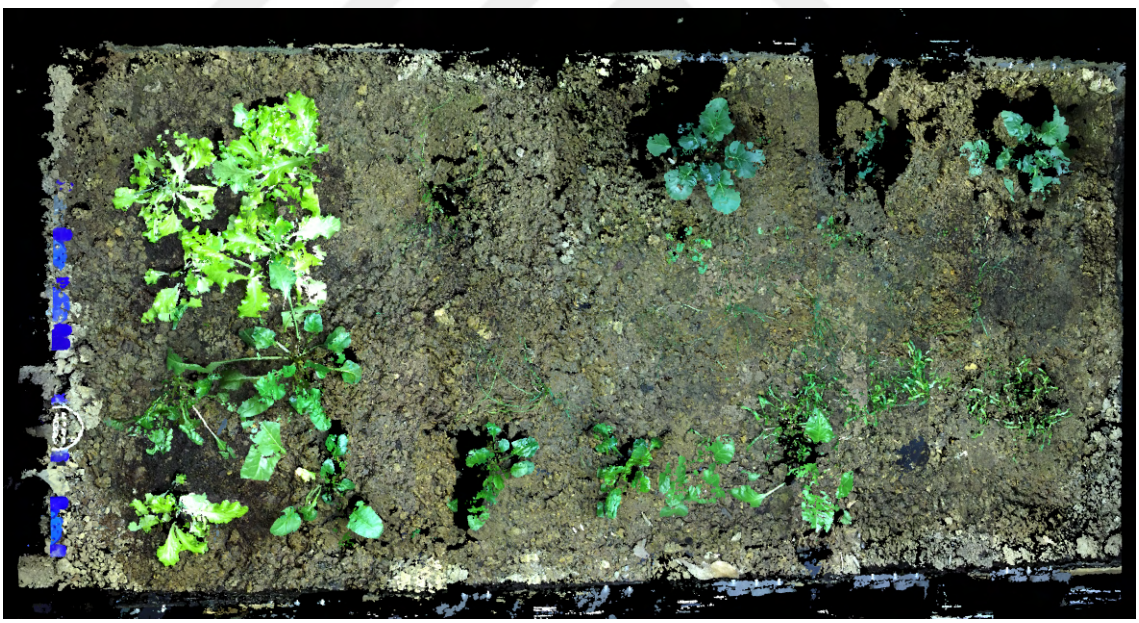


Figure 3.5. Top view of garden's point cloud data.

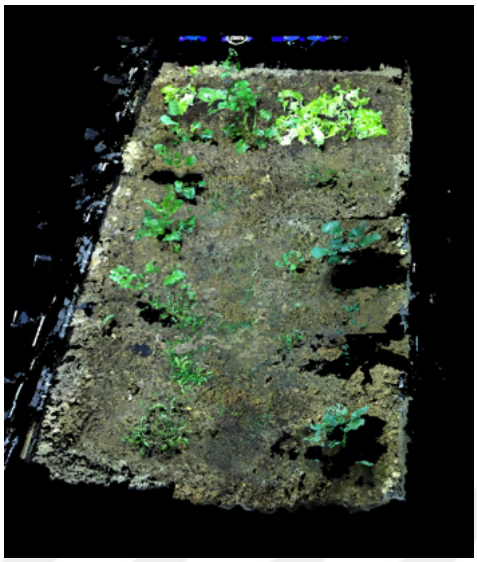
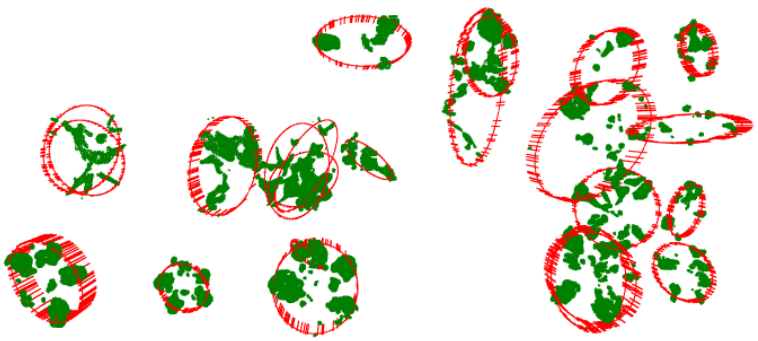


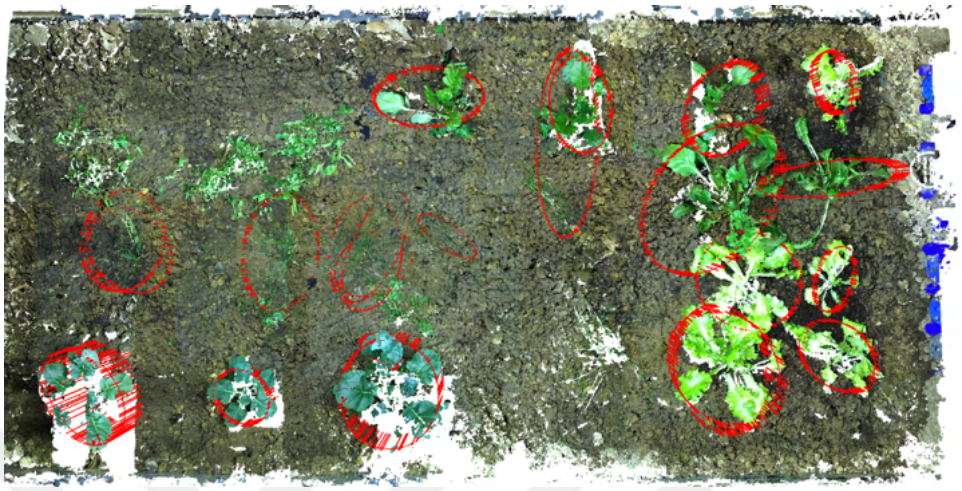
Figure 3.6. Frontal (from the right corner) view of garden's point cloud data.



(a) Top View

(b) Front View

Figure 3.7. Elliptic cylinder models of plants.



(a) Top view

(b) Front view

Figure 3.8. Three-dimensional topological garden map.

4. NAVIGATION AMONG PLANTS

This chapter is focused on achieving navigation among the plants in the garden. While navigating, the robot must avoid collisions with the plants so as not to damage them. Hence, all the plants in the garden become obstacles when navigating. Such a navigation ability necessary for a variety of tasks, such as watering, seeding, and taking soil measurements. Furthermore, it must be easily applicable with the changing garden topology since the plants' shapes change due to growth stage, lighting, and nastic movements. A novel reactive navigation approach is proposed. In particular, planar navigation using artificial potential functions are extended to navigation in the garden - considering its three-dimensional plants' topology. The remainder of the chapter is organized as follows: First, related work is summarized briefly. Following, the proposed three-dimensional workspace navigation approach is presented. Finally, the results of off-line and real-time experiments are evaluated.

4.1. Related Work

Traditionally, the navigation problem has been viewed as a special case of the general open-loop motion planning problem. Hence, the kinematics of planning are separated from the dynamics of execution. A geometric planner produces a trajectory in the joint configuration space of the robots connecting a prespecified initial condition to the fixed goal configuration [32]. This plan is then 'guarded'in real-time execution by a local tracking controller. Most geometric approaches are based on road maps or cell decomposition [33]. Furthermore, depending on how the planning is achieved, they are either classified as being centralized or decentralized [34].

Alternatively, rather than separating the kinematics and dynamics stages, the two are solved together. These are referred to as reactive approaches. For the simpler problem of planar robot motion among stationary obstacles, a variety of artificial potential function (APF) based approaches have been presented in the literature [35–39].

The reader is referred to [40] for a comprehensive survey. However, most of this work suffers from local minima. An optimization-based approach has been proposed to improve the repulsive force pointing angle and the gravitational field function [41]. This enables the robot to detach itself from mechanical equilibrium states caused by obstacles. Local minima problem is addressed using the A* optimal search algorithm with limited step size to find a feasible neighboring node to proceed [42]. Alternatively, the introduction of 'navigation functions' has established the existence of potential functions free from spurious minima that guarantee obstacle-free gradient-based navigation [36, 43]. This has been extended to multiple robots in geometrically simple one-dimensional settings that are entirely [44] and partially [45–47] actuated robots. However, the applicability of these work in realistic scenarios is limited due to respective assumptions regarding the workspace. An enhanced APF algorithm modifies the coefficient of the repulsive force depending on its proximity to the goal location [48]. As the proposed approaches typically assume two-dimensional workspaces, they are not applicable in 3D environments [49]. While an extension of APF to aerial navigation in three-dimensional workspace has been introduced, it is not applicable in scenarios in which the robot cannot be assumed to be modeled as a point charge [50].

4.2. Three Dimensional Workspace Artificial Potential Function

Consider the robot with end effector positioned at $c \in \mathbb{R}^3$ where $c = [c_x \ c_y \ c_z]^T$ with radius ρ_r. Suppose the goal is given by $g \in \mathbb{R}^3$. Let $\mathcal{P}$ denote the set of N_p plants where each plant i is modeled as elliptic cylinder o_i. Also assume the four fences surrounding the garden bed to be also modeled by four elliptic cylinders o_f, $f = 1, \ldots, 4$. Let $\mathcal{O}$ denote the union elliptic cylinders associated with the plants and fences. Let ϵ_o denote the minimal neighborhood of a plant or a fence that needs to be avoided, and ϵ_g denote the neighborhood of a goal in which the robot is considered to have reached the goal.

First, define $\phi : \mathbb{R}^3 \to \mathbb{R}^{\geq 0}$ as:

$$\phi(c; g) = \frac{\gamma(c; g)^k}{\beta(c)}. \tag{4.1}$$

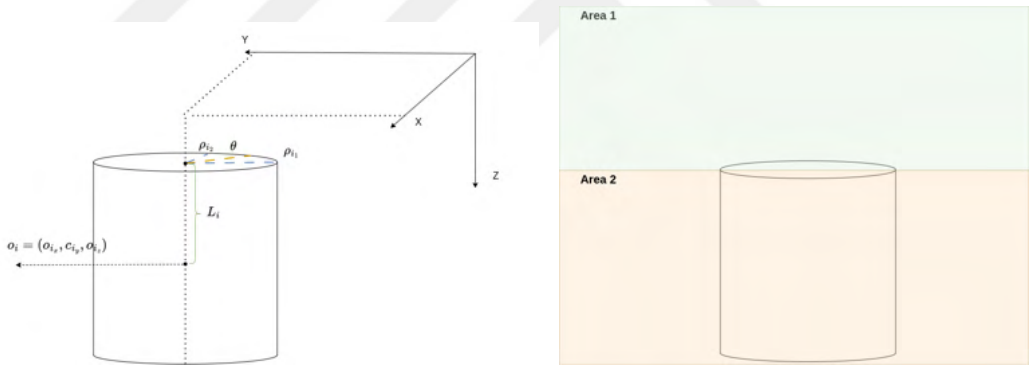
The numerator term $\gamma : \mathbb{R}^3 \rightarrow R^{\geq 0}$ encodes the distance to current goal $g \in \mathbb{R}^3$ as measured by $\gamma(c)$ as weighted by $k-$parameter,

$$\gamma(c; g) = \begin{cases} (c-g)^T(c-g) & \text{if} \quad \epsilon_o \leq ||c-g|| \\ 0 & \text{if} \quad \text{otherwise.} \end{cases} \tag{4.2}$$

The denominator term $\beta : \mathbb{R}^3 \rightarrow R^{\geq 0}$ encodes the plants as obstacles and is defined so that as the robot approaches a plant $o_i \in \mathcal{O}$, t $\beta(c) \rightarrow 0$. Thus, it ensures collision-free navigation. It is defined as:

$$\beta(c) = \prod_{o_i \in \mathcal{O}} \beta_i(c). \tag{4.3}$$

The $\beta_i : \mathbb{R}^3 \rightarrow R^{\geq 0}$ functions are defined based on the robot's position relative to the elliptic cylinder. In particular, the spatial region around it is partitioned into two regions as shown in Fig. 4.1(b).



(a) Relative geometry with respect to an elliptic cylinder.

(b) Regions around elliptic cylinder.

Figure 4.1. Plant as an obstacle.

- Region 1: Region above the elliptic cylinder: $(o_{i_z} - \frac{L_i}{2})) \leq c_z \leq (o_{i_z} + \frac{L_i}{2})$.
- Region 2: Region next to the elliptic cylinder: $\|P(c - o_i)\| > \rho_r + \rho_{o_i}$.

The definitions of β functions formed in the direction of the areas determined around elliptic cylinders are as follows:

$$\beta_i(c) = \begin{cases} (c-o_i)^T(c-o_i) - (\rho_r + \rho_{o_i})^2 & \text{if} \quad \begin{array}{l} (o_{i_z} - \frac{L_i}{2})) \leq c_z \leq (o_{i_z} + \frac{L_i}{2}) \\ \|P(c-o_i)\| > \rho_r + \rho_{o_i} \end{array} \\ (c-o_i)^T(c-o_i) - (L_r + \frac{L_i}{2})^2 & \text{if} \quad 0 \leq c_z < (o_{i_z} - \frac{L_i}{2})). \end{cases} \tag{4.4}$$

Here, the term ρ_{o_i} refers to the distance between the elliptic cylinder o_i as defined by the distance between the closest elliptic cross-section and robot's position [27],

$$\rho_{o_i}(\theta_i) = \frac{\rho_{i_1}\rho_{i_2}}{\sqrt{\rho_{i_1}^2 \sin^2(\theta_i) + \rho_{i_2}^2 \cos^2(\theta_i)}}. \tag{4.5}$$

The $k-$parameter designates the weight of the goal distance $\gamma(c)$ with respect to the obstacle function $\beta(c)$. It is is critical in setting the balance between attractive (goal) forces and repelling (obstacle) forces. A heuristic function is used as defined:

$$\frac{\gamma(c)^k}{\beta(c)} > 1 \rightarrow \gamma^k > \beta \rightarrow k \log \gamma(c) > \log \beta(c) \rightarrow k = round(\frac{\log \beta(c)}{\log \gamma(c)}) + 1. \tag{4.6}$$

Finally, the 3DW artificial potential function $\varphi : \mathbb{R}^3 \rightarrow \mathbb{R}^{\geq 0}$ is defined as:

$$\varphi(c; g) = \sigma_d \circ \sigma \circ \phi(c; g) = \frac{\gamma(c; g)}{(\beta(c) + \gamma(c; g)^k)^{1/k}} \tag{4.7}$$

where $\sigma : [0, \infty) \rightarrow [0, 1]$ and $\sigma_d : [0, 1] \rightarrow [0, 1]$ are defined as:

$$\sigma(x) = \frac{x}{1 + x}, \qquad \sigma_d(x) = x^{\frac{1}{k}}. \tag{4.8}$$

The displacement commands Δc to the step motors of the robot are obtained from the negative gradient of φ as:

$$\Delta c = \begin{cases} -\frac{\nabla_c \varphi}{\|\nabla_c \varphi\|} & \text{if } \|\nabla_c \varphi\| > 1 \\ -\nabla_c \varphi & \text{otherwise} \end{cases}, \tag{4.9}$$

$$c(t + \delta t) = c + \Delta c \tag{4.10}$$

The resulting navigation path for a sample case is shown in Figure 4.2. The total displacement Δc is monitored until it reaches 1 mm. Once it reaches 1 mm, a command is sent to the stepper motors. As the necessary calibration processes and step adjustments are performed within the Arduino, the number of steps corresponding to 1 mm is provided to the `moveRobotSpecialMethod` function.

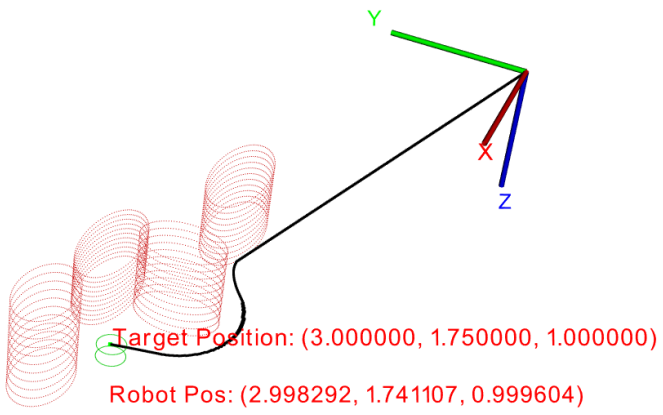


Figure 4.2. Sample path resulting from 3DW-APF.

4.3. 3DW APF Navigation Algorithm

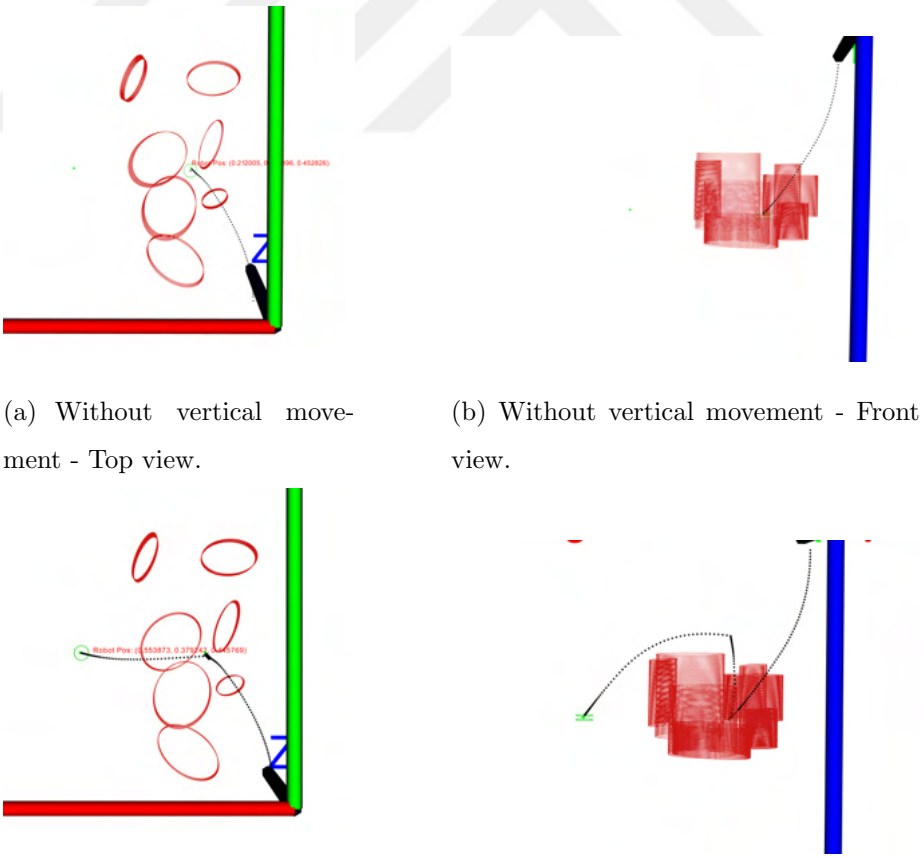
One of the shortcomings of artificial potential function based approaches pertains to the robot getting stuck in a local minima before reaching the goal location. The occurrence of this is known to depend on the workspace topology. When moving among the plants in the garden, as plants grow and the foliage in the garden becomes denser, chances of getting stuck in local minima - namely $\bigtriangledown_c \varphi(c) = 0$ increase. In order to address such cases, a 3DW-APF navigation algorithm has been designed as given by Algorithm 1. In this algorithm, if the robot gets stuck in a local minima prior to reaching the goal position, a virtual intermediate goal g' is given to the robot. Given the 3D topological garden map, the virtual intermediate target point can simply be assigned along the z-axis direction. Hence, the location of the virtual intermediate goal g' is obtained simply via perturbing its location vertically by a small amount ϵ_z - namely,

$$g' = c + \begin{bmatrix} 0 & 0 & \epsilon_z \end{bmatrix}^T. \tag{4.11}$$

This is repeated until $\bigtriangledown_c \varphi(c) \neq 0$. The corresponding vertical distance of $\delta z = M\epsilon_z$ that is moved by the robot depends on the number M of times this is repeated. A sample case is shown in Figure 4.3 in which the robot needs to navigate to a goal location $g = \begin{bmatrix} 0.55 & 0.38 & 0.49 \end{bmatrix}^T$.

Algorithm 1 3DW-APF Navigation Algorithm.

1: c, g, $\mathcal{O}$
2: **while** $||\delta c|| > \epsilon_s$ **do**
3: $\quad c(t + \delta t) = c(t) + \Delta c$
4: $\quad$ **while** $\delta c = 0$ **do**
5: $\quad\quad g' \leftarrow c + \delta\,[0, 0, \epsilon_z]$
6: $\quad\quad g \leftarrow g'$
7: $\quad\quad c(t + \delta t) = c(t) + \Delta c$
8: $\quad$ **end while**
9: **end while**



(a) Without vertical movement - Top view.

(b) Without vertical movement - Front view.

(c) With vertical movement - Top view.

(d) With vertical movement - Front view.

Figure 4.3. Navigation paths without and with vertical movements.

However, just using 3DW-APF, it gets stuck at $c(t) = \begin{bmatrix} 0.21 & 0.38 & 0.494 \end{bmatrix}^T$. A sample application of the proposed 3DW-APF navigation algorithm is shown in Figure 4.3. The robot moves vertically until it is able to navigate towards its goal location using equation 4.11. The robot utilizes this intermediate point to escape from the local minimum and continue its movement effectively. Throughout the tests, it is observed that this added perturbation effect allowed the system to escape from the local minima. As plants grow and canopy foliage gets denser, free workspace will become more packed and chances of doing such vertical movements will increase.

4.4. TarRob Navigation Task

The flow of processing of the navigation task is shown in Figure 4.4. It assumes that a three-dimensional topological garden map has been constructed as explained in Chapter 3. It also assumes that a sequence of goal points for the robot is given. The robot moves to each goal position sequentially using the proposed 3DW APF navigation algorithm. Each movement is achieved through a series of steps that are calculated based on the 3DW APF algorithm. This process is repeated for each goal location and once all points have been reached, the robot is considered to have successfully achieved its task and is then moved to its home position.

4.5. Experiments and Results

This section presents the experimental evaluation of the navigation performance of TarRob. In the experiments, the robot is given a sequence of N workspace goal positions $g_k = (g_{kx}, g_{ky}, g_{kz})^T,\ k = 1, \ldots, N$. The goal locations are selected quasi randomly - namely, these locations are subjected to the constraints of a watering task. As such, they are random ground locations that are in close proximity to each plant in the garden. The robot needs to move to these target locations one by one while avoiding any collisions with the plants. For 3DW-APF algorithm, $\epsilon_z = 0.2,\ \epsilon_o = 0.01,$ and $\epsilon_s = 0.001$ meters. For ablation studies, the goodness of the generated place proposals is evaluated in comparison to its previous simple motion strategy - 2D + V navigation.

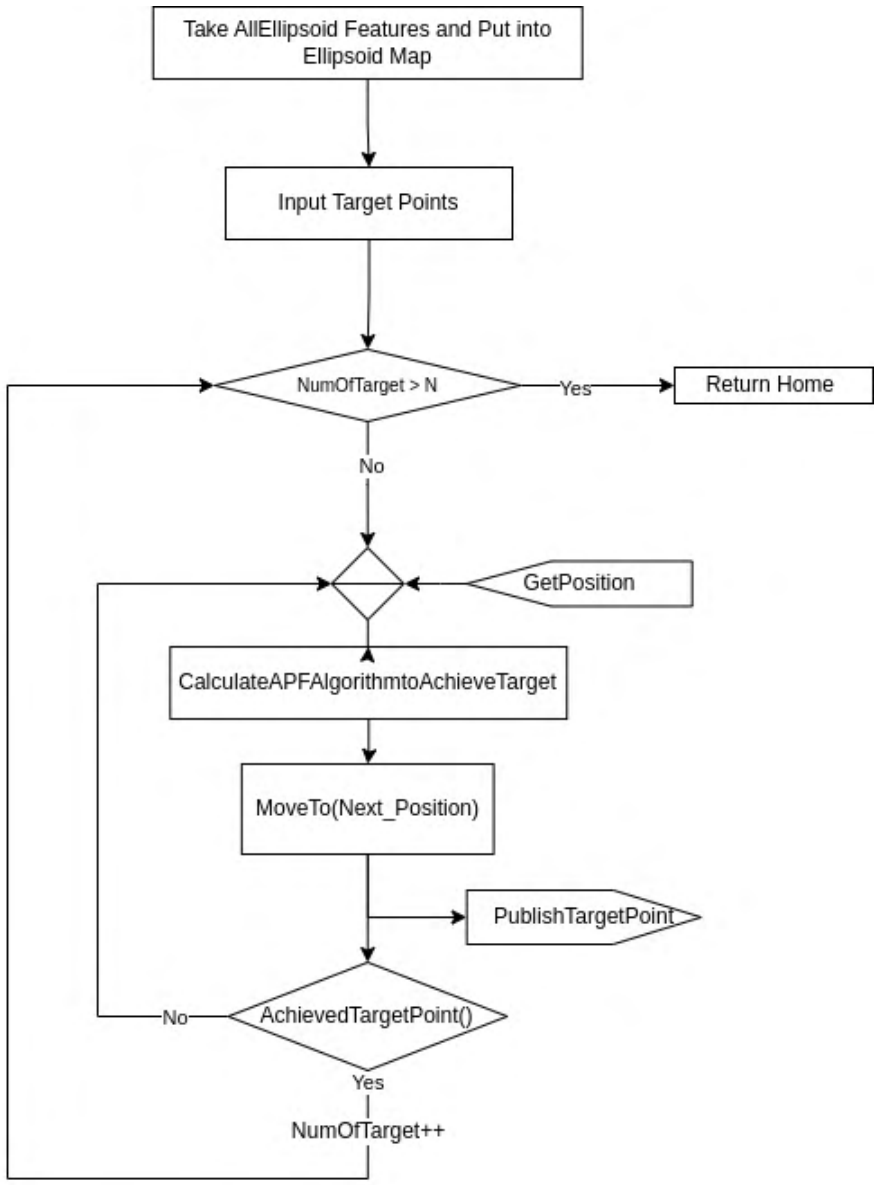


Figure 4.4. Navigation task process flow.

First, statistical results from an extensive set of experiments are presented. In these experiments, the garden models are based on real RGB-D data collected by the robot. The results of the on-robot experiments are discussed next. In both sets of experiments, the outer boundary of the garden is modeled as a rectangular prism, which is defined by six faces: four side faces, one bottom, and one top (virtual). Note that the top face is necessary so that the robot minimizes its vertical movements as much as possible. Each face is modeled by an elliptic cylinder with parameters as given in Table 4.1.

The resulting elliptic cylinders are as seen in Figure 4.5.

Table 4.1. Garden outer boundary modeling as elliptic cylinders.

Side i	Position o_i	ρ_{i1}, ρ_{i2}	Height L_i
0	$-0.04, 0.35, 0.35$	0.02, 0.4	0.35
1	$1.77, 0.35, 0.35$	0.02, 0.4	0.35
2	$0.86, -0.04, 0.35$	0.1, 0.2	0.35
3	$0.86, 0.75, 0.35$	0.1, 0.2	0.35
4	$0.86, 0.35, 0.345$	1.9, 0.0.78	0.001
5	$-0.04, 0.35, 0$	1.9,0.78	0.001

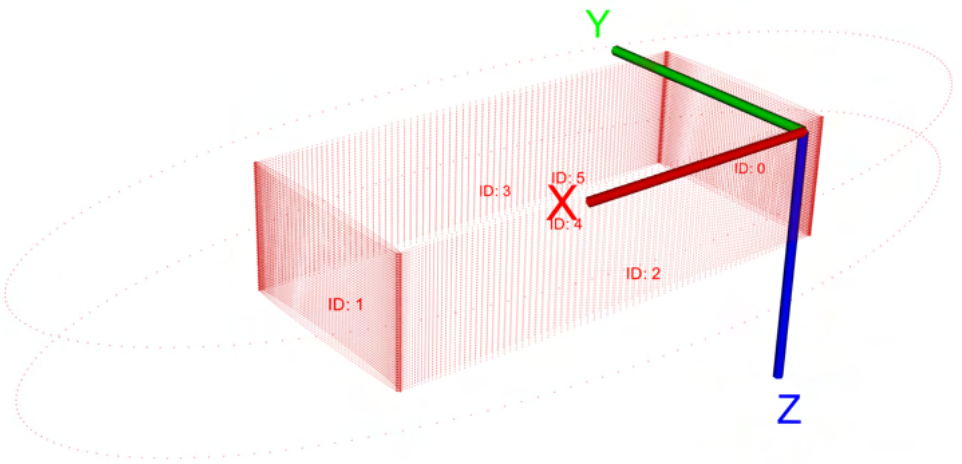


Figure 4.5. TarRob garden: Outer boundary modeling.

Navigation performance is evaluated based on the following criteria:

- Success rate (S.R.) indicates the percentage of target points created in each case that the robot successfully reaches. It is defined as:

$$\text{S.R.}(\%) = \frac{N_s}{N} \times 100 \tag{4.12}$$

where N is the total number of target points, N_s is the number of successfully reached target points.

The evaluation of whether the target is successfully reached is based on the robot approaching the designated target point within a threshold distance of 0.02 meters. If the robot remains within this threshold, the attempt is considered successful; otherwise, it is deemed unsuccessful. S.R.(%) represents the success rate as a percentage.

- Average path length efficiency η is determined by taking the sum of the ratio of path length between consecutive goal locations with respect to their Euclidean distance,

$$\eta = \frac{\sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} ||\delta c(t)|| dt}{\sum_{i=0}^{N-1} ||(c(t_i) - g_{i+1})||}. \tag{4.13}$$

  Note that $\eta \geq 1$. As $\eta \to 1$, efficiency will increase. Conversely, as η gets larger, efficiency decreases.

- Average number of intermediate goals N_I. $N_I = 0$ indicates no local minima.

- Average path length to intermediate goals η_I is determined by taking the ratio of total intermediate vertical path lengths to the overall path length,

$$\eta_I = \frac{\sum_{j=0}^{N_I} \int_{t_i}^{t_{i+1}} \delta z_i dt}{\sum_{j=0}^{N_I} \int_{t_i}^{t_{i+1}} \delta c(t) dt} \tag{4.14}$$

  where δz_i represents incremental vertical movements. This is a second measure of efficiency since a smaller η_I indicates less effort spent during vertical movements for getting out of trapping locations.

4.5.1. Real Data Simulation Experiments

Simulation experiments were conducted using real data from TarRob's garden. A simulation environment has been developed using the C++ programming language and Point Cloud Library. The data is sampled from the RGB-D data that TarRob has collected over five weeks. The stages of growth of the plants in the garden are considered one week apart. As such, each week, the plants have grown with respect to the previous week. In the first week, there were 15 plants, consisting of 5 lettuces, one broccoli, five chards, and four spring onions. In the second week, after the first week, two spring onions, two broccolis, and three spinaches were added, bringing the total to 22 plants.

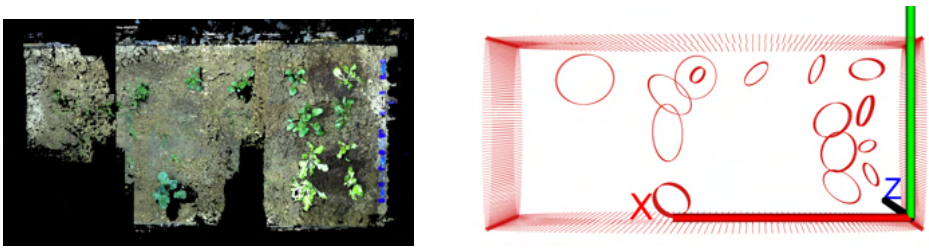
In the third week, one carrot was added, increasing the total number of plants to 23. No updates were made to the bed in the fourth and fifth weeks. The respective point cloud data for each week is as shown in Figure 4.6. It is observed that as time progresses, the plant canopy areas and heights increase as expected.

Table 4.2. TarRob garden plant model.

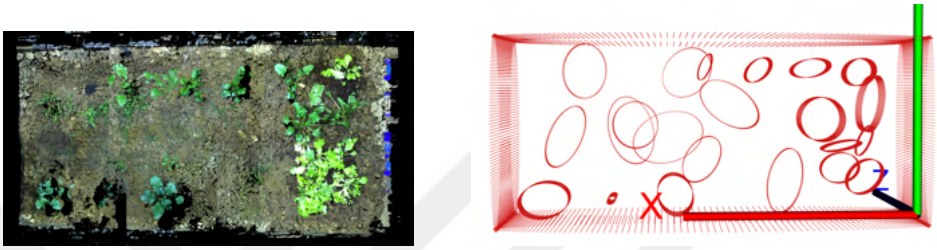
Week	Date	Plants	Height	Canopy length-width	
			Min-Max L_i (m)	Min-Max ρ_{i1} (m)	Min-Max ρ_{i2}
1	2024-02-09	15	0.041-0.192	0.039-0.133	0.023-0.114
2	2024-02-16	22	0.057-0.209	0.041-0.197	0.014-0.113
3	2024-02-25	23	0.034-0.309	0.046-0.222	0.014-0.130
4	2024-02-03	23	0.034-0.480	0.068-0.220	0.037-0.143
5	2024-02-13	23	0.046-0.496	0.029-0.280	0.016-0.234

The robot constructs a topological map of the garden for each data as explained in Chapter 3. The resulting 3D topological garden map for each week is shown in Figure 4.6. As expected, the packness of the garden's workspace increases as the plants grow.

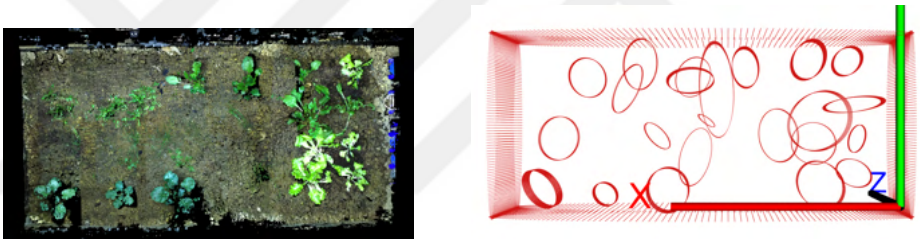
In the datasets collected on 2024-03-03 and 2024-03-13, measurement accuracy is reduced as some plants' heights exceed the range of the ZED-2 camera. Consequently, some plants appear to go out of the garden's bounds. Weekly statistical values for the plants' model parameters including minimum and maximum height, semi-major and semi-minor axis length measurements are presented in Table 4.2. Throughout the weeks, the heights and canopy areas of the plants are observed to increase. Let it be noted that the minimum height measurements do not increase as expected due to the growth patterns of plants such as carrots and spring onions, which spread horizontally.



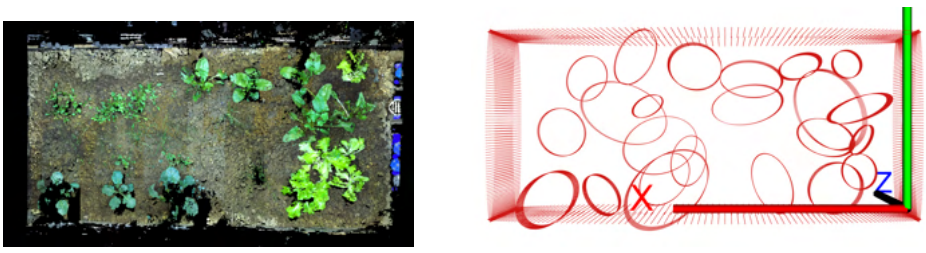
(a) 2024-02-09



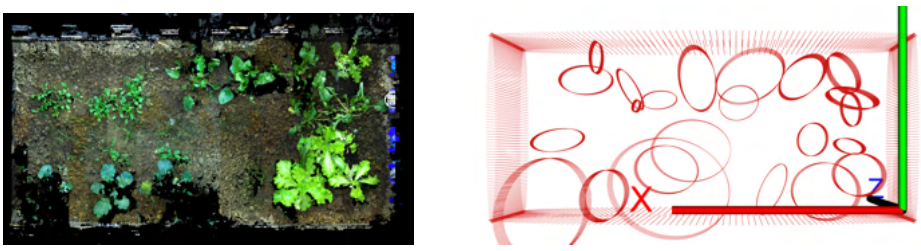
(b) 2024-02-16



(c) 2024-02-25



(d) 2024-03-03



(e) 2024-03-13

Figure 4.6. Real data based simulation tests: Left: Top view of gardens's point cloud data; Right: 3D topological garden maps.

Since these plants grow close to the soil and have thin leaf and stem surfaces, the accuracy of the segmentation masks have decreased, resulting in inconsistencies in the minimum height data. The robot uses elliptic cylinder models of plants for navigation. The navigation paths are depicted using a set of graphical entities as explained in Figure 4.7.

- Red elliptical cylinders: Garden outer boundary and plants' canopies.
- Green points: Goal locations.
- Black points: Navigation path with numbers indicating movement order.

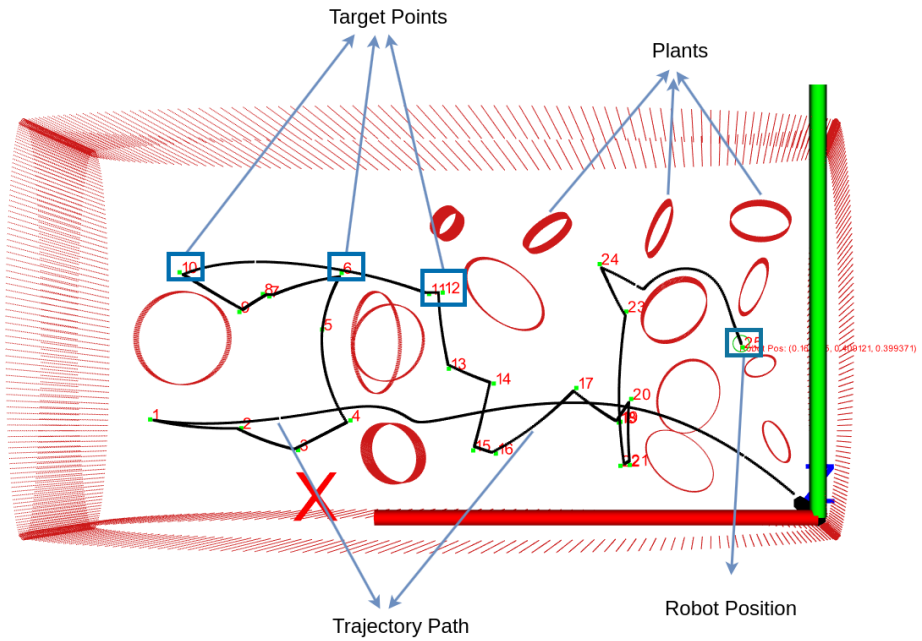


Figure 4.7. Simulated navigation result graphics.

4.5.2. Statistical Results

A statistical study of navigation performance has been conducted considering navigation tasks with $N = 25, 50, 75, 100$ goal locations. Each case is repeated 50 different goal locations. As explained, for ablation studies, we also consider the 2D+V algorithms.

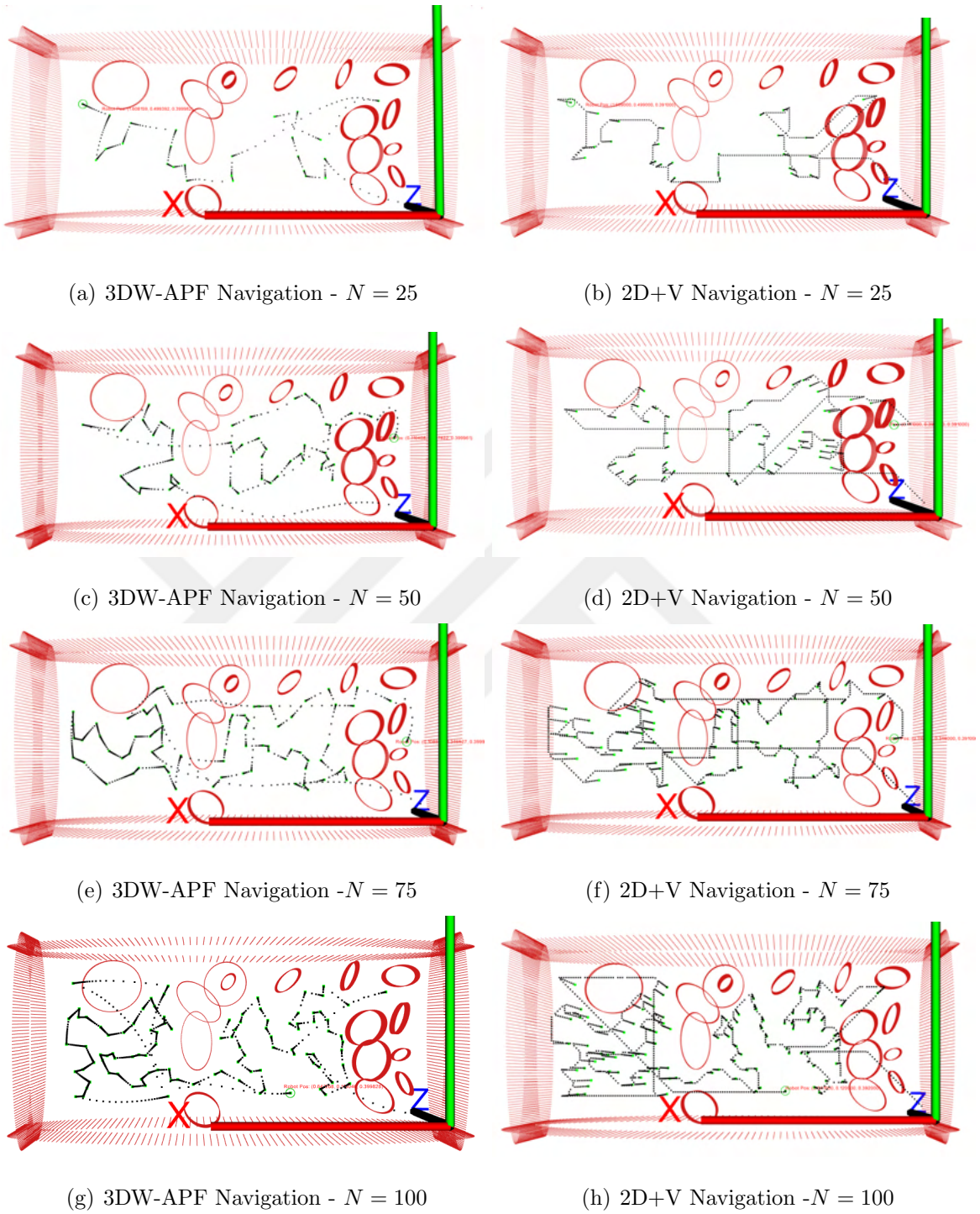
Sample resulting navigation paths from the first week for each different N value are as shown in Figure 4.8. In this growth stage, plants have small canopy areas and obstruction is low. Table 4.3 gives comparative results for the first week.

Table 4.3. Comparative results: 2024-02-09.

N	**3DW-APF navigation**				**2D+V Navigation**			
	$\eta_I(\%)$	η	N_I	S.R. (%)	$\eta_I(\%)$	η	N_I	S.R. (%)
25	0.0226318	1.027	0.111	98.8148	38.6495	4.87682	25	100
50	0.0411333	1.033	0.2	97.76	41.6429	6.50155	50	100
75	0.0285336	1.021	0.14	97.3867	43.20	7.92985	75	100
100	0.0286709	1.014	0.14	98.08	44.1495	9.16231	100	100

Hence, η_I value is close to zero. It is observed that efficiency for the proposed 3DW-APF Navigation Algorithm remains the same regardless of the number of goal locations - in contrast to the 2D+V Navigation. Furthermore, it is about 4-10 fold better for the proposed 3DW-APF Navigation Algorithm than 2D+V Navigation. In contrast, the η value of the 2D+V algorithm is significantly higher than that of the 3DW-APF algorithm. For the 2D+V algorithm, this increase occurred proportionally with the increase in N. Additionally, it is noted that the 3DW-APF algorithm followed a more aggressive trajectory. For this week, increasing the number N of targets directly impacted performance. It is also observed that the distance of the randomly selected target points from the plants played a critical role in the algorithms' performance.

Sample navigation paths on the second-week dataset are as given in Figure 4.9. It is observed that the canopy areas of the plants have increased and had greater heights compared to the first week. Some plants overlapped with others, which reduced the robot's maneuvering space. The results are given Table 4.5. The 3DW-APF algorithm is determined to have values η and η_I greater than those compared to the previous week.



(a) 3DW-APF Navigation - $N = 25$

(b) 2D+V Navigation - $N = 25$

(c) 3DW-APF Navigation - $N = 50$

(d) 2D+V Navigation - $N = 50$

(e) 3DW-APF Navigation -$N = 75$

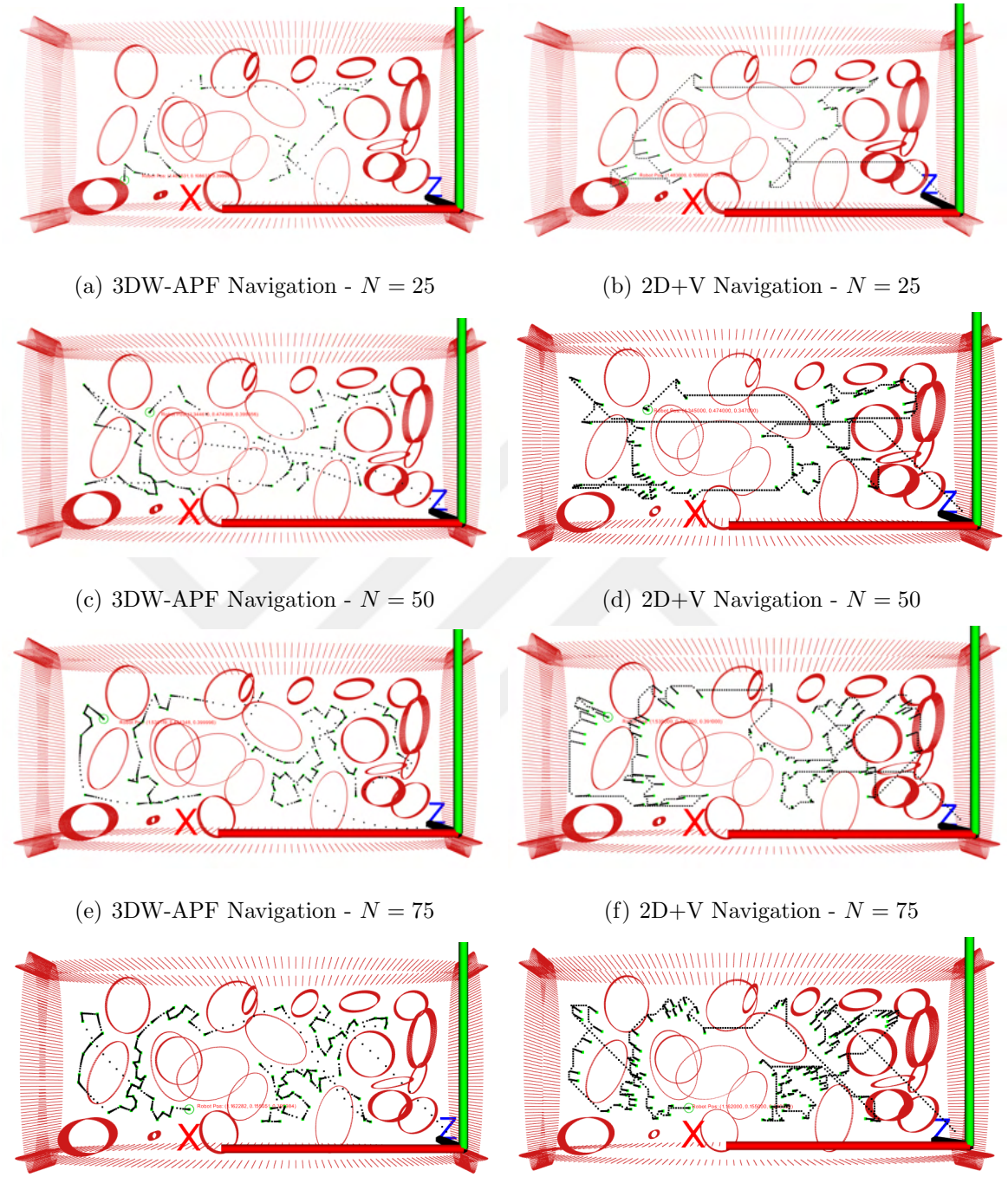
(f) 2D+V Navigation - $N = 75$

(g) 3DW-APF Navigation - $N = 100$

(h) 2D+V Navigation -$N = 100$

Figure 4.8. Simulation results - 2024-02-09 dataset.

On the other hand, in the 2D+V algorithm, since it moves upward along the Z-axis by the number of points, the η_I value did not change, and despite the increased complexity of the environment, the η value produced similar results.



(a) 3DW-APF Navigation - $N = 25$

(b) 2D+V Navigation - $N = 25$

(c) 3DW-APF Navigation - $N = 50$

(d) 2D+V Navigation - $N = 50$

(e) 3DW-APF Navigation - $N = 75$

(f) 2D+V Navigation - $N = 75$

(g) 3DW-APF Navigation - $N = 100$

(h) 2D+V Navigation -$N = 100$

Figure 4.9. Simulation results: 2024-02-16 dataset.

The efficiency of the 3DW-APF Navigation Algorithm is about 1.5-5 fold better, as seen in Table 4.4. This behavior is attributed to the planar movements of the 2D+V algorithm.

However, it was observed that the trajectories followed by the 3DW-APF algorithm varied depending on the environment and the added target points. For this week, it was determined that the positions of the randomly selected target points being concentrated in areas with a high density of plants significantly affected both the complexity of the environment and the performance of the algorithms.

The results of Figure 4.10 are from the following week. It was observed that the plants had grown significantly, and due to the randomly selected target points being farther apart. However, the η value varied depending on the order and positions of the goal locations. The 3DW-APF algorithm is determined to have values η and η_I greater than those compared to the previous week. On the other hand, in the 2D+V algorithm, since it moves upward along the Z-axis by the number of points, the η_I value did not change, and despite the increased complexity of the environment, the η value produced similar results. The results of the 2D+V algorithm were once again consistent with those of previous weeks. For this week, the increased distance between the randomly chosen targets increased the η_I value. In contrast, the effect of goal sequencing became more pronounced. This demonstrated the effect of complex and widely distributed goal locations on performance.

Table 4.4. Comparative results: 2024-02-16.

N	**3DW-APF navigation**				**2D+V Navigation**			
	$\eta_I(\%)$	η	$N_I.$	S.R. (%)	$\eta_I(\%)$	η	$N_I.$	S.R. (%)
25	0.7354	3.47424	0.622951	99.3115	39.2395	5.14064	25	100
50	0.9197	2.6147	0.77381	98.8571	42.5124	7.27343	50	100
75	0.9152	1.91087	0.64	99.8667	44.0520	9.09509	75	100
100	0.7248	1.7786	0.52	99.42	44.9257	10.5628	100	100

Sample paths taken in the garden in the fourth week are as given Figure 4.11. It is observed that the plants covered most of the garden area and that the robot placed intermediate points to reach the targets as indicated by the increase in the η_I value.

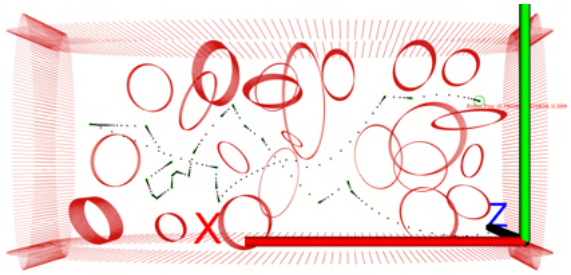
Despite the increased complexity of the environment compared to previous weeks, the low η value and the high success rate demonstrate the robustness of the 3DW-APF algorithm. The 2D+V algorithm, on the other hand, consistently produced high η and η_I values. For this week, the dense placement of random targets among the plants highlighted the effectiveness of intermediate point placement strategies. Despite the complexity of the environment, the high success rate underscores the superiority of the 3DW-APF algorithm.

In the last week, the garden's topological complexity is observed to increase as plants have grown. Sample navigation paths for each different N case are shown in Figure 4.10. The increase in the success rate demonstrates that the 3DW-APF algorithm performs reliably despite the increasing complexity of the environment. Additionally, η_I and η values were found to be consistent with those of previous weeks.

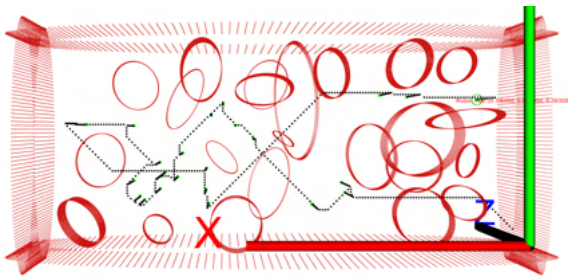
Simulation results demonstrate that the developed 3DW-APF navigation algorithm is significantly more efficient than the 2D+V algorithm. Additionally, the intermediate path length and the number of intermediate points are noticeably lower in the 3DW-APF navigation algorithm than in the 2D+V algorithm. However, it is also observed that the 2D+V algorithm achieves an average success rate of approximately 5% higher. The success rate of the 3DW+APF navigation algorithm can be improved using a target points policy instead of a random target generator.

Table 4.5. Comparative results: 2024-02-25.

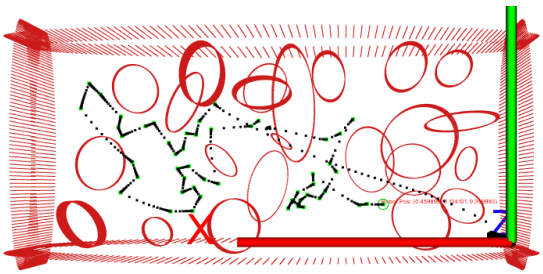
N	**3DW-APF navigation**				**2D+V Navigation**			
	$\eta_I(\%)$	η	$N_I.$	S.R. (%)	$\eta_I(\%)$	η	$N_I.$	S.R. (%)
25	0.8833	1.33242	0.26	96.24	39	5.05118	25	100
50	0.6628	1.41991	0.28	98.8	42.6521	7.42171	50	100
75	0.7279	1.53295	0.4	98.4	44.0566	9.08471	75	100
100	0.9102	1.38973	0.52	98.46	44.8213	10.3799	100	100



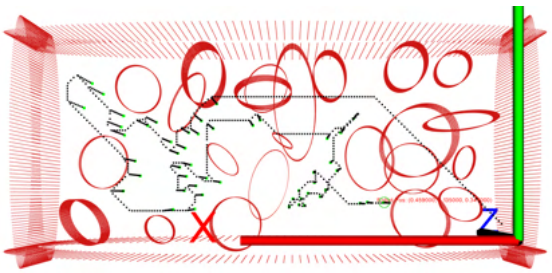
(a) 3DW-APF Navigation - $N = 25$



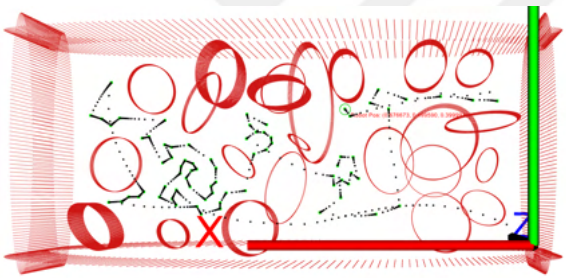
(b) 2D+V Navigation -$N = 25$



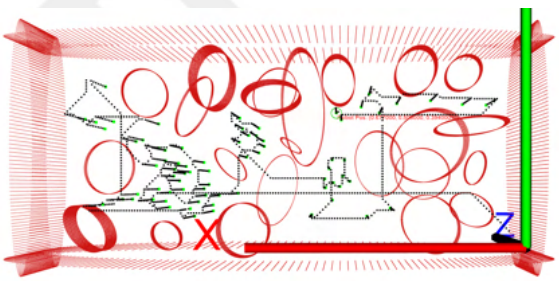
(c) 3DW-APF Navigation - $N = 50$



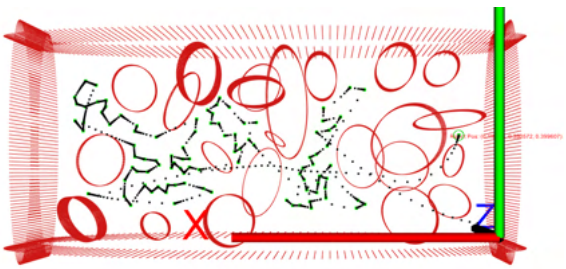
(d) 2D+V Navigation -$N = 50$



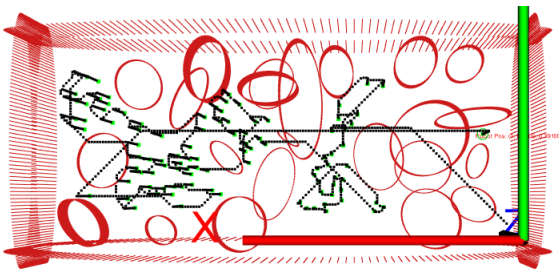
(e) 3DW-APF Navigation - $N = 75$



(f) 2D+V Navigation -$N = 75$



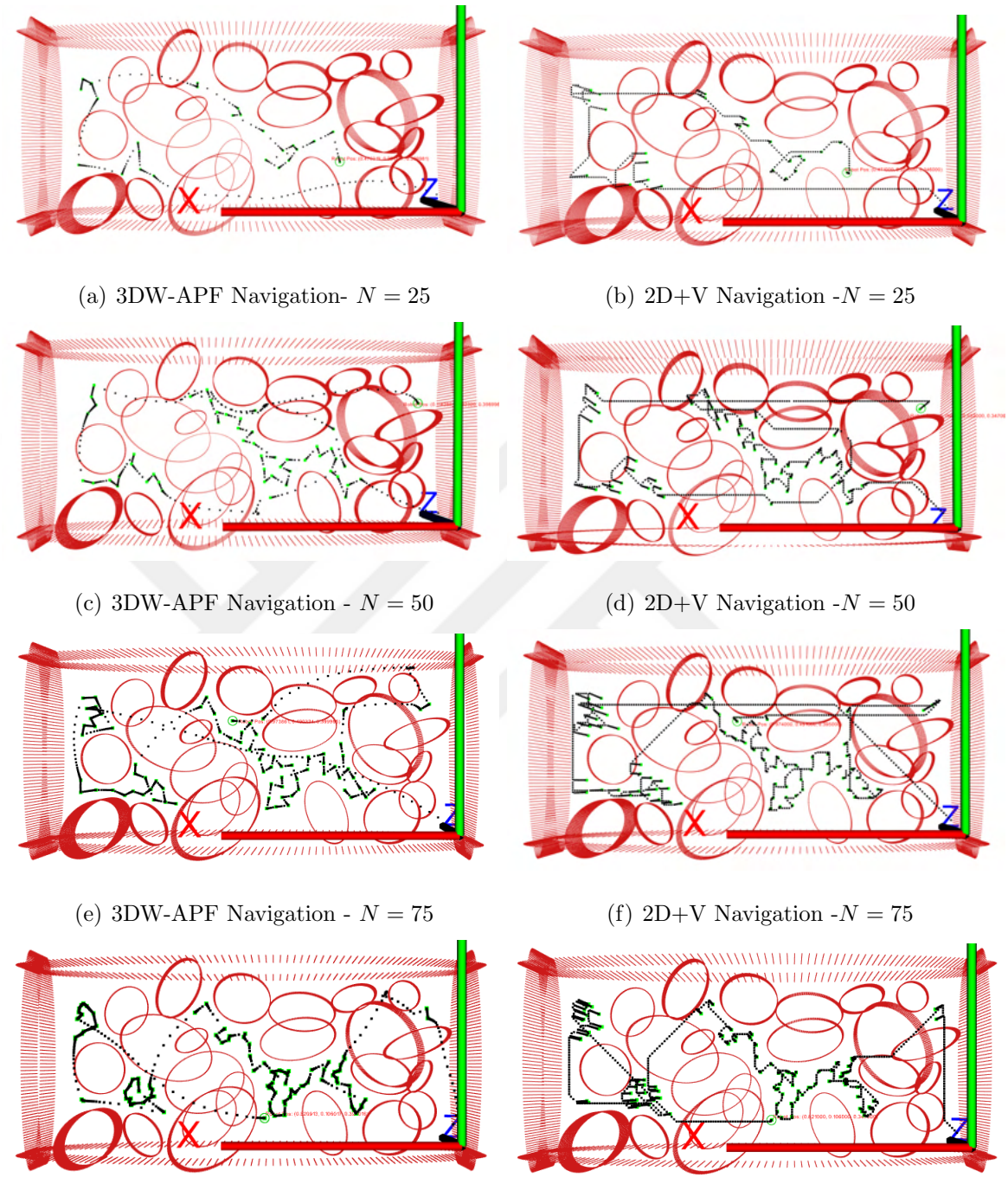
(g) 3DW-APF Navigation - $N = 100$



(h) 2D+V Navigation - $N = 100$

Figure 4.10. Simulation results: 2024-02-25 dataset.

The superior efficiency of the developed 3DW-APF navigation algorithm makes it highly suitable for tasks such as irrigation, sensor measurements, or operations requiring interaction with multiple plants or points.



(a) 3DW-APF Navigation- $N = 25$

(b) 2D+V Navigation -$N = 25$

(c) 3DW-APF Navigation - $N = 50$

(d) 2D+V Navigation -$N = 50$

(e) 3DW-APF Navigation - $N = 75$

(f) 2D+V Navigation -$N = 75$

(g) 3DW-APF Navigation - 100 Target Points

(h) 2D+V Navigation-100 Target Points

Figure 4.11. Simulation results: 2024-03-03 dataset.

While the 2D+V algorithm achieves a higher success rate due to its nearly ideal planar movements, the 3DW-APF navigation algorithm still maintains a high success rate in various scenarios. As the plants in the system grow, the area they occupy increases over the weeks.

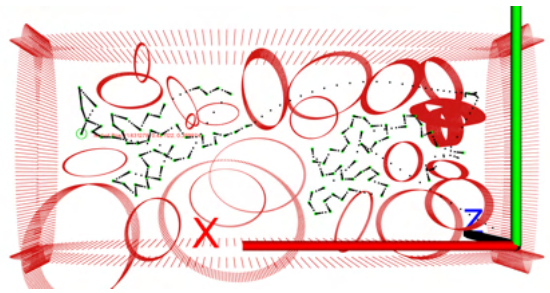
Table 4.6. Comparative results: 2024-03-03.

N	**3DW-APF navigation**				**2D+V Navigation**			
	$\eta_I(\%)$	η	$N_I.$	S.R. (%)	$\eta_I(\%)$	η	$N_I.$	S.R. (%)
25	1.9548	1.07011	0.44	96.72	39.5733	5.31695	25	100
50	1.9248	1.06892	0.56	95.2	42.7087	7.46745	50	100
75	1.7526	1.04951	0.58	95.6	44.2353	9.35073	75	100
100	2.4952	1.06431	0.94	93.2	45.0519	10.8275	100	100

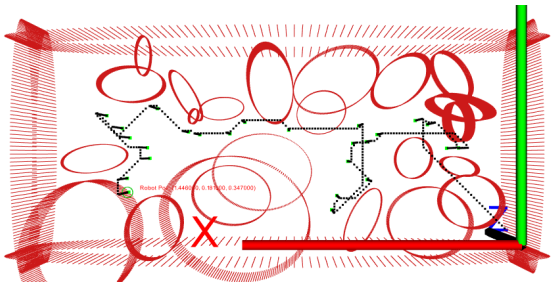
Table 4.7. Comparative results: 2024-03-13.

N	**3DW-APF navigation**				**2D+V Navigation**			
	$\eta_I(\%)$	η	$N_I.$	S.R. (%)	$\eta_I(\%)$	η	$N_I.$	S.R. (%)
25	1.3625	1.0745	0.3	94.8	39.8011	5.48065	25	100
50	1.9230	1.13927	0.6	99.72	43.2187	8.10438	50	100
75	2.1187	1.11778	0.78	99.6533	44.4218	9.73441	75	100
100	2.2873	1.2497	1.1	99.92	45.1296	11.0736	100	100

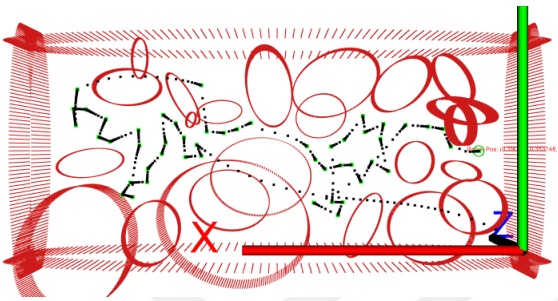
The developed random target assignment algorithm ensures that target points are assigned while maintaining a predefined threshold distance from both the plants and each other. As the plant surface area increases, the assigned target points tend to be positioned closer together and distributed within a specific range. Due to this, in the initial weeks, target points are also assigned between the ellipsoid and the outer boundary. However, over time, the number of targets in this region decreases. This phenomenon directly affects the positions and spatial distribution of the assigned targets, thereby influencing the algorithm's performance. Finally, when integrated into the real system, it is believed that developing an algorithm that assigns target points based on plant locations rather than using a random target assignment approach will yield more successful results.



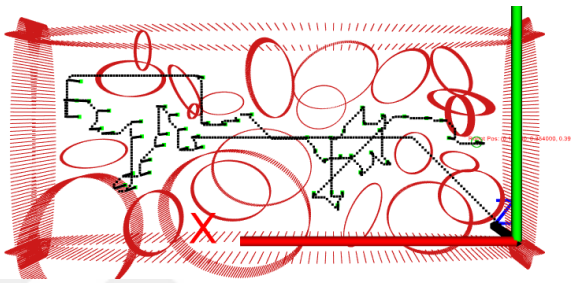
(a) 3DW-APF Navigation - $N = 25$



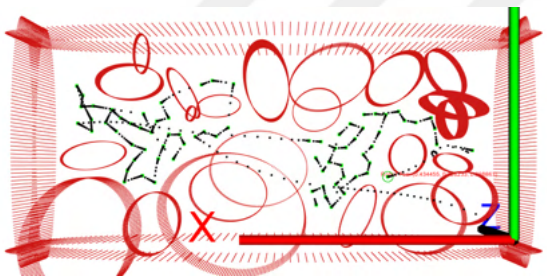
(b) 2D+V Navigation - $N = 25$



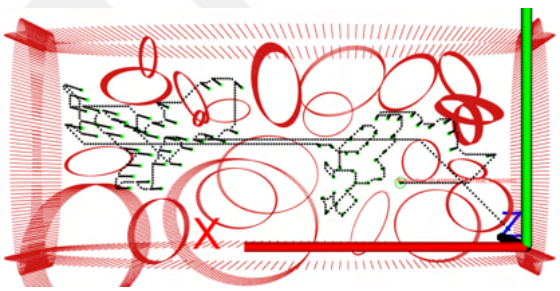
(c) 3DW-APF Navigation - $N = 50$



(d) 2D+V Navigation - $N = 50$

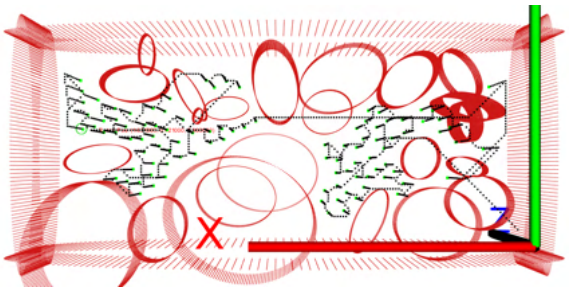


(e) 3DW-APF - $N = 75$



(f) 2D+V Navigation - $N = 75$

(g) 3DW-APF Navigation - $N = 100$



(h) 2D+V Navigation - $N = 100$

Figure 4.12. Simulation results: 2024-03-13 dataset.

As the weeks progress and the plant surface area increases, the target points naturally become closer to each other, which is considered one of the factors contributing to a high success rate.

A significant limitation of the 2D+V algorithm is its insensitivity to dynamic environments, significantly reducing its efficiency in practical applications. Conversely, the 3DW-APF navigation algorithm is more aggressive and adaptive to dynamic environments. Despite its higher sensitivity to dynamic obstacles, the 3DW-APF navigation algorithm consistently delivers excellent efficiency and success rate results. As the operational area grows and the number of required tasks increases, efficiency becomes a critical parameter. In this context, the 3DW-APF navigation algorithm is a more effective solution, particularly in complex and dynamic environments where adaptability and efficiency are essential.

4.5.3. TarRob Application

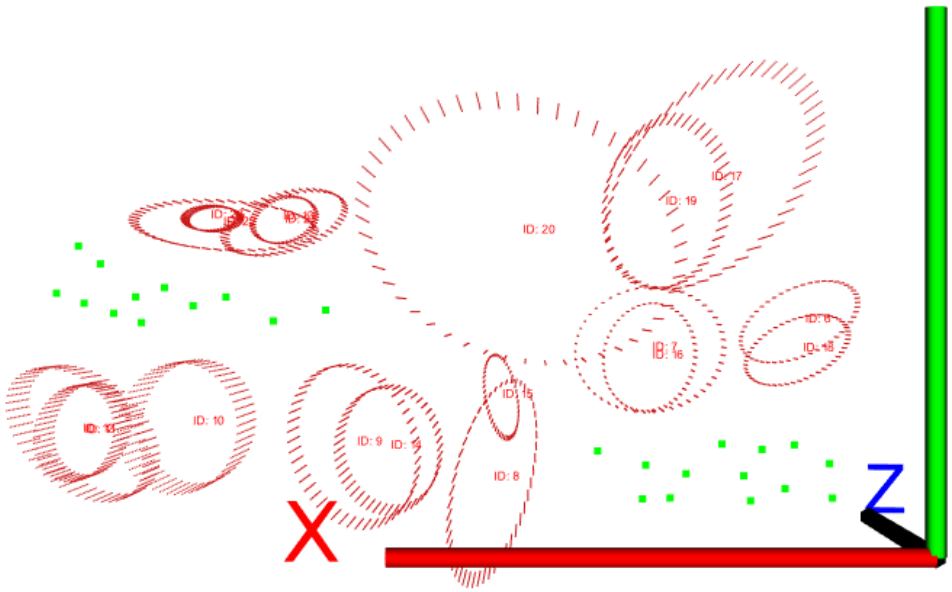
The proposed approach has been applied on Tarrob. The garden layout is illustrated in Figure 4.13(a). It consists of a total of 21 plants. First, the DailyPlant method is executed to obtain data on every plant in the garden. After capturing an RGB-D image, the PlantGrowthMonitoring algorithm and the Ellipsoid Fitting algorithm are run sequentially to generate an ellipsoid map of the entire garden. The resulting 3D topological garden map is shown in Figure 4.13(b). Among these plants, the minimum height was measured as 0.071 m, while the maximum height was 0.39 m. The minimum semi-major and semi-minor axes values were measured as 0.053 m and 0.024 m, respectively, while the maximum semi-major and semi-minor axes values were 0.314 m and 0.239 m, respectively

Following, the 3D-APF algorithm is deployed for navigating in this garden. The robot is given a sequence of random goal points that are determined using the constructed 3D garden map as shown in Figure 4.14(a). Random target points were automatically generated within ellipsoids, ensuring that each point maintained a predefined minimum distance of 0.05 meters from the others. The resulting path is depicted in Figure 4.14(b). It was observed that the system successfully reached 21 different randomly assigned target points while avoiding plants in the real environment. Thus, the study demonstrating the system's functionality was successfully completed.

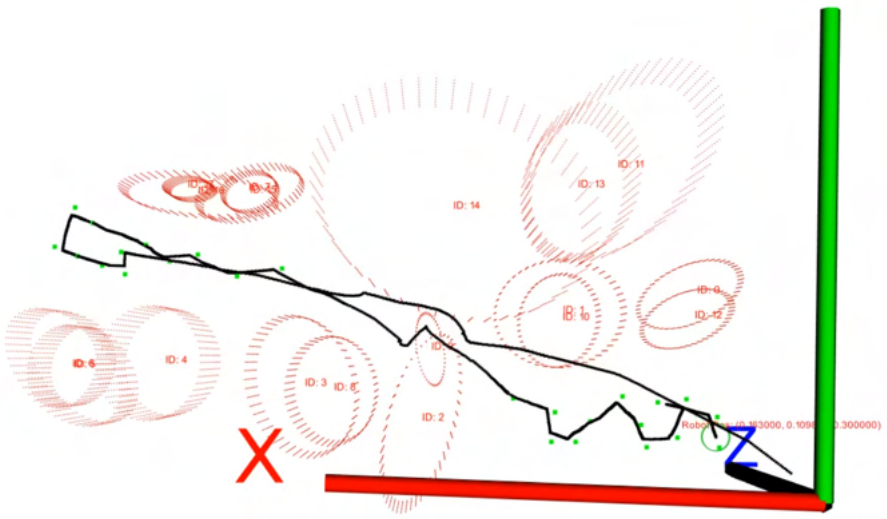
(a) TarRob garden top view.

(b) 3D garden map.

Figure 4.13. TarRob garden layout and 3D garden map.



(a) Sequence of 25 goal points.



(b) 3DW-APF navigation path.

Figure 4.14. TarRob navigates through a path of given goal points.

5. CONCLUSION

This thesis has focused on the design and development of TarRob in regard to three related aspects: i) Its hardware and software overhaul, ii) The construction of a three-dimensional topological garden map, and iii) The formulation of reactive navigation in a three-dimensional workspace.

In Chapter 2, the developments made on the TarRob system are described in detail. Mechanical, electronic, and software issues in the existing system were resolved, and the robot was made capable of robust operation. In addition, electronic and software updates were made to enhance the system's capabilities. For future work, the infrastructure for collecting more data from sensors can be utilized to increase the amount of data obtained from the system. The part to which the ZED-2 Camera is connected can be made mobile, and depth measurements can be made in multiple directions. Additionally, the motor driver could be upgraded to one with higher torque and no overheating issues.

In Chapter 3, the Construction of a three-dimensional topological garden map has been presented. This is based on the RGB-D data collected by the robot. It should be noted that the Construction needs to be repeated after each RGB-D acquisition. Segmentation masks corresponding to the individual plants are used to determine the point cloud objects corresponding to the plants in the garden. Each point cloud object is then modeled as an elliptic cylinder. These elliptic cylinders are then put together to construct the topological map. In future work, the accuracy of the garden and plant models can be improved by increasing the RGB-D data collected to reduce gaps in the generated garden point cloud maps. In addition, the leaf detection algorithm can be improved to model the plant leaves to increase the plant representation's accuracy. Therefore, a plant can represent more than an elliptic cylinder.

In Chapter 4, artificial potential function-based navigation is extended to be applicable to a three-dimensional workspace in which the robot is modeled as a cylinder and obstacles (plants) are modeled as elliptic cylinders. The resulting 3DW-APF enables the robot to autonomously move from one location in its workspace to another while ensuring that it does not hit any of the plants. The local minima problem is addressed by introducing an algorithm that results in vertical movement whenever it happens. The resulting 3DW-APF navigation algorithm was tested in a simulation environment using data obtained from the system and compared with the results of the existing planar motion algorithm. Finally, tests were conducted on the actual system. For future work, improvements can be made to achieve smoother movements, and the developed APF algorithm can be further refined to make it more comprehensive and robust.

REFERENCES

1. Mollison, B., *An Introduction to Permaculture*, Yankee Permaculture, Barking Frogs Permaculture Center, Sparr, Florida, 2001.

2. Ozguven, M., *The Digital Age in Agriculture*, CRC Press, Boca Raton, Florida, 2023.

3. "Agricultural Robot Market - Global Forecast to 2025", `https://www.marketsandmarkets.com/Market-Reports/agricultural-robot-market-17`, accessed on January 13, 2025.

4. Xu, R. and C. Li, "A Modular Agricultural Robotic System (MARS) for Precision Farming: Concept and Implementation", *Journal of Field Robotics*, Vol. 39, No. 4, p. 387–409, 2022.

5. PROJECT, R., "Robotics for Microfarms Develops Open Technologies to Assist Organic, Diversified, Vegetable Farmers.", 2025, https://romi-project.eu/, accessed on January 12, 2025.

6. "Naïo Technologies", https://www.naio-technologies.com/en/home/, accessed on October 5, 2023.

7. Rajendran, V., B. Debnath, S. Mghames, W. Mandil, S. Parsa, S. Parsons and A. Ghalamzan-E., "Towards Autonomous Selective Harvesting: A Review of Robot Perception, Robot Design, Motion Planning and Control", *Journal of Field Robotics*, Vol. 41, No. 7, pp. 2247–2279, 2024.

8. Pham, V.-C., H.-G. Nguyen, T.-K. Doan and G.-B. Huynh, "A Computer Vision Based Robotic Harvesting System for Lettuce", *International Journal of Mechanical Engineering and Robotics Research*, Vol. 9, No. 11, pp. 1526–1531, 2020.

9. Yin, H., Q. Sun, X. Ren, J. Guo, Y. Yang, Y. Wei, B. Huang, X. Chai and M. Zhong, "Development, Integration, and Field Evaluation of An Autonomous Citrus-Harvesting Robot", *Journal of Field Robotics*, Vol. 40, No. 6, pp. 1363–1387, 2023.

10. Avigal, Y., A. Deza, W. Wong, S. Oehme, M. Presten, M. Theis, J. Chui, P. Shao, H. Huang, A. Kotani, S. Sharma, R. Parikh, M. Luo, S. Mukherjee, S. Carpin, J. H. Viers, S. Vougioukas and K. Goldberg, "Learning Seed Placements and Automation Policies for Polyculture Farming with Companion Plants", *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 902–908, Xi'an, China, 2021.

11. Kamat, V., S. Aeron, A. Gu, H. Jalan, S. Adebola and K. Goldberg, "PolyPoD: An Algorithm for Polyculture Seed Placement", *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pp. 1–6, Chicago, Illinois, USA, 2023.

12. FarmBot, "FarmBot: Open-Source CNC Farming", 2023, `https://farm.bot`, accessed on July 22, 2023.

13. Avigal, Y., W. Wong, M. Presten, M. Theis, S. Aeron, A. Deza, S. Sharma, R. Parikh, S. Oehme, S. Carpin, J. H. Viers, S. Vougioukas and K. Y. Goldberg, "Simulating Polyculture Farming to Learn Automation Policies for Plant Diversity and Precision Irrigation", *IEEE Transactions on Automation Science and Engineering*, Vol. 19, No. 3, pp. 1352–1364, 2022.

14. Adebola, S., M. Presten, R. Parikh, S. Aeron, S. Mukherjee, S. Sharma, M. Theis, W. Teitelbaum, E. Solowjow and K. Goldberg, "Automated Pruning and Irrigation of Polyculture Plants", *IEEE Transactions on Automation Science and Engineering*, Vol. 21, No. 3, pp. 2199–2210, 2024.

15. Doksansoglu, H., "CNC Robot for Indoor Farming", EE492 Senior Project,

Bogazici University, Electrical & Electronics Engineering, 2017.

16. Dogan, M. and U. Soylu, "CNC-Based Intelligent System for Automated Indoor Farming", EE491-492 Senior Project, Bogazici University, Electrical & Electronics Engineering, 2018.

17. Takım, M. B. and C. B. Cakır, "Intelligent Autonomous Robot for Precise Indoor Farming", EE491-492 Senior Project, Bogazici University, Electrical & Electronics Engineering, 2019.

18. Yılmaz, A., *Vision-Based Robotic Plant Growth Monitoring and Learning in Polyculter Farming*, Master's Thesis, Boğaziçi University, Electrical and Electronic Engineering, 2024.

19. RepRap, "RAMPS 1.4", https://reprap.org/wiki/RAMPS_1.4, 2025, accessed on January 6, 2025.

20. Okura, F., "3D Modeling and Reconstruction of Plants and Trees: A Cross-Cutting Review Across Computer Graphics, Vision, and Plant Phenotyping", *Breeding Science*, Vol. 72, No. 1, pp. 31–47, 2022.

21. Madokoro, H., S. Yamamoto, Y. Nishimura, S. Nix, H. Woo and K. Sato, "Calibration and 3D Reconstruction of Images Obtained Using Spherical Panoramic Camera", pp. 311–318, 2021.

22. Yamada, M., N. Yoshimoto and Y. Nakayama, "Integrated 3D Farm Modeling with Photogrammetry and Optical Camera Communication", *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pp. 1–5, Seoul, South Korea, 2024.

23. Potena, C., R. Khanna, J. Nieto, R. Siegwart, D. Nardi and A. Pretto, "AgriColMap: Aerial-Ground Collaborative 3D Mapping for Precision Farming", *IEEE Robotics and Automation Letters*, Vol. 4, No. 2, pp. 1085–1092, 2019.

24. Bao, Y., L. Tang, M. W. Breitzman and P. S. Schnable, "Field-based Robotic Phenotyping of Sorghum Plant Architecture Using Stereo Vision", *Journal of Field Robotics*, Vol. 36, No. 2, pp. 397–415, 2019.

25. Weiss, M. and F. Baret, "Using 3D Point Clouds Derived from UAV RGB Imagery to Describe Vineyard 3D Macro-Structure", *Remote Sensing*, Vol. 9, No. 2, p. 111, 2017.

26. Price, W. J., B. Shafii and D. C. Thill, "An Individual-Plant Growth Simulation Model for Quantifying Plant Competition", *Proceedings of the 6th Annual Conference on Applied Statistics in Agriculture*, pp. 1–10, Kansas State University, Manhattan, KS, USA, 1994.

27. Bektaş, K. and H. I. Bozma, "APF-RL: Safe Mapless Navigation in Unknown Environments", *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7299–7305, Philadelphia, Pennsylvania, USA, 2022.

28. PROJECT, R., "Robotics for Microfarms Develops Open Technologies to Assist Organic, Diversified, Vegetable Farmers.", https://romi-project.eu/, 2025, accessed on January 12, 2025.

29. Weiss, U. and P. Biber, "Plant Detection and Mapping for Agricultural Robots Using A 3D LIDAR Sensor", *Robotics and Autonomous Systems*, Vol. 59, No. 5, pp. 265–273, 2011.

30. Khachiyan, L. G., "A Polynomial Algorithm in Linear Programming", *Doklady Akademii Nauk SSSR*, Vol. 244, No. 5, pp. 1093–1096, 1979, translated in *Soviet Mathematics Doklady*, Vol. 20, pp. 191–194, 1979.

31. Todd, M. J. and E. A. Yıldırım, "On Khachiyan's Algorithm for The Computation of Minimum-Volume Enclosing Ellipsoids", *Discrete Applied Mathematics*, Vol. 155, No. 13, pp. 1731–1744, 2007.

32. Sharir, M., *Algorithmic Motion Planning*, CRC Press, Boca Raton, FL, 1997.

33. Choset, H., K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun, *Principles of Robot Motion: Theory, Algorithms and Implementations*, MIT Press, Cambridge, Massachusetts, 2005.

34. Yeung, D.-Y. and G. A. Bekey, "A Decentralized Approach to The Motion Planning Problem for Multiple Mobile Robots", *IEEE International Conference on Robotics and Automation*, Vol. 4, pp. 1779–1784, Raleigh, North Carolina, USA, 1987.

35. Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", *IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 500–505, San Francisco, California, USA, 1985.

36. Rimon, E. and D. E. Koditschek, "Exact Robot Navigation Using Artificial Potential Functions", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5, pp. 501–518, 1992.

37. Warren, C., "Multiple Robot Path Coordination Using Artificial Potential Fields", *IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 500–505, San Francisco, California, USA, 1990.

38. Kim, J. O. and P. K. Khosla, "Real-Time Obstacle Avoidance Using Harmonic Potential Functions", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 3, pp. 338–349, 1992.

39. Reif, H. and H. Wang, "Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots", *Workshop on Algorithmic Foundations of Robotics*, pp. 431–459, Cambridge, Massachusetts, USA, 1994.

40. Hwang, Y. and N. Ahuja, "A Potential Field Approach to Path Planning", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 1, pp. 23–32, 1992.

41. Li, H., "Robotic Path Planning Strategy Based on Improved Artificial Potential Field", *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, pp. 67–71, IEEE, Beijing, China, 2020.

42. Tan, J., L. Zhao, Y. Wang, Y. Zhang and L. Li, "The 3D Path Planning Based on A Algorithm and Artificial Potential Field for the Rotary-Wing Flying Robot", *Proceedings of the 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, pp. 551–556, Hangzhou, China, 2016.

43. Koditschek, D. E. and E. Rimon, "Robot Navigation Functions on Manifolds with Boundary", *Advances in Applied Mathematics*, Vol. 11, No. 4, pp. 412–442, 1990.

44. Koditschek, D. E., "An Approach to Autonomous Robot Assembly", *Robotica*, Vol. 12, pp. 137–155, 1994.

45. Karagöz, C. S., *A Game-Theoretic Approach to Objects' Moving Problem With Mobile Robots"*, Ph.D. Thesis, Bogazici University, Istanbul, Turkey, 2001.

46. Karagöz, C., H. I. Bozma and D. E. Koditschek, "Feedback-Based Event-Driven Parts Moving", *IEEE Transactions on Robotics*, Vol. 20, No. 6, pp. 1012–1018, 2004.

47. Bozma, H. I. and D. E. Koditschek, "Assembly as A Noncooperative Game of Its Pieces: Analysis of 1D Sphere Assemblies", *Robotica*, Vol. 19, No. 1, pp. 93–108, 2001.

48. Y., H., H. Li, Y. Dai, G. Lu and M. A. Duan, "A 3D Path Planning Algorithm for UAVs Based on an Improved Artificial Potential Field and Bidirectional RRT*", *Drones*, Vol. 8, No. 12, p. 760, 2024.

49. Yang, L., J. Qi, D. Song, J. Xiao, J. Han and Y. Xia, "Survey of Robot 3D Path Planning Algorithms", *Hindawi Publishing Corporation Journal of Control Science and Engineering*, Vol. 2016, No. 7426913, p. 22, 2016.

50. Ahmad, S., Z. N. Sunberg and J. S. Humbert, "APF-PF: Probabilistic Depth Perception for 3D Reactive Obstacle Avoidance", *2021 American Control Conference (ACC)*, pp. 32–39, IEEE, New Orleans, LA, USA, 2021.

51. Larson, C., "MVEE - Minimum Volume Enclosing Ellipsoid", `https://github.com/chrislarson1/MVEE`, accessed on December 24, 2024.

APPENDIX A: ELLIPSOID FITTING

Khachiyan's Algorithm is used for the ellipsoid fitting [30]. Its general flow is shown in Algorithm 2.

Algorithm 2 Khachiyan's Algorithm.

$N_i = \mathcal{P}_i$
$\mathcal{Q}_i \leftarrow \emptyset$
$n \leftarrow 0$
while $n < N_i$ **do** ▷ Find canopy points
    $q_n = (p_{nx}, p_{ny}, 1)^T$
    $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{q_n\}$
    $U \leftarrow I_D$
end while
$X = Q_i U Q_i^T + \epsilon I$

For the initialization of the parameters, the initial weight vector $\mathbf{u}$ is equally distributed:

$$u = \frac{1}{N}\begin{bmatrix}1 & \dots & 1\end{bmatrix}. \tag{A.1}$$

Ellipse matrix $\mathbf{X}$ is computed as:

$$X = Q \cdot \mathrm{diag}(u) \cdot Q^\top + \epsilon I \tag{A.2}$$

where ϵ is a regularization term to avoid singularities, and $\mathbf{I}$ is the identity matrix. The diagonal elements of $\mathbf{M}$ are computed as:

$$M = \mathrm{diag}\left(Q^\top \cdot X^{-1} \cdot Q\right). \tag{A.3}$$

Let $m_{\max}$ be the maximum element in $\mathbf{M}$ and its index be $i_{\max}$. The step size is computed as:

$$\text{step\_size} = \frac{m_{\max} - 2 - 1}{3 \cdot (m_{\max} - 1)}. \tag{A.4}$$

The weights **u** are updated as:

$$u = (1 - \text{step\_size}) \cdot u, \quad u_{i_{\max}} = u_{i_{\max}} + \text{step\_size}. \tag{A.5}$$

After convergence, the ellipse parameters are computed as: - The matrix A:

$$A = \left(P \cdot \text{diag}(u) \cdot P^\top - (P \cdot u) \cdot (P \cdot u)^\top\right)^{-1} \cdot \frac{1}{2} + \epsilon I. \tag{A.6}$$

After determining the optimal weights vector **u**, the parameters of the enclosing ellipse can be calculated as follows:

- Elliptic Cross-section Center (o_{ix}, o_{iy}): The center of the ellipse o_i is given by:

$$o_i(o_{ix}, o_{iy}) = P \cdot u \tag{A.7}$$

  where P is the matrix of input points and u is the optimal weights vector. Expanding this for 2D coordinates:

$$o_{ix} = \sum_{i=1}^{N} u_i x_i, \quad o_{iy} = \sum_{i=1}^{N} u_i y_i. \tag{A.8}$$

- Ellipse major and minor axis lengths (ρ_{i1}, ρ_{i2}): The semi-major axis (ρ_{i1}) and semi-minor axis (ρ_{i2}) are derived from the eigenvalues of the ellipse's shape matrix **A**. If λ_1 and λ_2 are the eigenvalues of **A**, the axes lengths are given by:

$$\rho_{i1} = \sqrt{\frac{1}{\lambda_1}}, \quad \rho_{i2} = \sqrt{\frac{1}{\lambda_2}}. \tag{A.9}$$

- Ellipse Orientation (θ): The orientation of the ellipseis determined by the eigenvector corresponding to the largest eigenvalue (λ_1). Let $\mathbf{v}_1 = \begin{bmatrix} v_{1x} \\ v_{1y} \end{bmatrix}$ be this eigenvector. The orientation angle θ is computed as:

$$\theta = \arctan\left(\frac{v_{1y}}{v_{1x}}\right). \tag{A.10}$$

APPENDIX B: APF COMPUTATIONS

In this section, the detailed steps of the formulas provided in Section 4.2 will be explained. The gradient $\bigtriangledown_c \varphi$ of φ is computed using chain rule:

$$\bigtriangledown_c \varphi = \frac{(\beta + \gamma^k)^{1/k} \bigtriangledown_c \gamma - \gamma \bigtriangledown_c (\beta + \gamma^k)^{1/k}}{(\beta + \gamma^k)^{2/k}}. \tag{B.1}$$

The gradient of $(\beta + \gamma^k)^{1/k}$ term with respect to c is:

$$\bigtriangledown (\beta + \gamma^k)^{1/k} = \frac{1}{k} (\beta + \gamma^k)^{\frac{1-k}{k}} \bigtriangledown_c (\beta + \gamma^k). \tag{B.2}$$

The gradient of $(\beta + \gamma^k)$ term with respect to c is:

$$\bigtriangledown_c (\beta + \gamma^k) = \bigtriangledown_c \beta + k\gamma^{k-1} \bigtriangledown_c \gamma. \tag{B.3}$$

Combining everything Eq. B.1 becomes as:

$$\bigtriangledown_c \varphi = \frac{(\beta + \gamma^k)^{1/k}(-2(g-c)) - \gamma(\frac{1}{k}((\beta + \gamma^k)^{\frac{1-k}{k}}(\bigtriangledown_c \beta - 2k\gamma^{k-1}(g-c))}{(\beta + \gamma^k)^{2/k}}. \tag{B.4}$$

The gradient of γ function is defined as:

$$\bigtriangledown_c \gamma(c; g) = 2(c - g). \tag{B.5}$$

The gradient $\bigtriangledown_c \beta(c)$ of β function with respect to c is defined as:

$$\bigtriangledown_c \beta_(c) = \sum_{o_i \in \mathcal{O}} \frac{\beta(c)}{\beta_i(c)} \bigtriangledown_c \beta_i(c) \tag{B.6}$$

Recall that the definition of $\beta_i(c)$ depends on where the robot is with respect to the respective obstacle:

$$\beta_i(c) = \begin{cases} (c - o_i)^T (c - o_i) - (\rho_r + \rho_{o_i})^2 & \text{if} \quad (o_{i_z} - \frac{L_i}{2})) \leq c_z \leq (o_{i_z} + \frac{L_i}{2}) \\ & \|P(c - o_i)\| > \rho_r + \rho_{o_i} \\ (c - o_i)^T (c - o_i) - (L_r + \frac{L_i}{2})^2 & \text{if} \quad 0 \leq c_z < (o_{i_z} - \frac{L_i}{2})). \end{cases} \tag{B.7}$$

Hence, the term $\bigtriangledown_c \beta_i$ is computed accordingly:

$$\bigtriangledown_c \beta_i(c) = \begin{cases} \bigtriangledown_c (c - o_i)^T (c - o_i) + & \text{if} \quad (o_{i_z} - \frac{L_i}{2})) \leq c_z \leq (o_{i_z} + \frac{L_i}{2}) \\ \quad 2(\rho_r + \rho_{o_i}) \bigtriangledown_c \rho_{o_i} & \quad ||P(c - o_i|| < (\rho_{o_i} + \rho_r) \\ - - - - - - - - - - \quad -- \quad - - - - - - - - - - -- & \\ 2(c - o_i) & \text{if} \quad 0 \leq c_z < (o_{i_z} - \frac{L_i}{2})) \ . \end{cases} \tag{B.8}$$

In the first region, the gradient of β_i with respect to c is:

$$\bigtriangledown_c \beta_i = \bigtriangledown_c (c - o_i)^T (c - o_i) + 2(\rho_r + \rho_{o_i}) \bigtriangledown_c \rho_{o_i}. \tag{B.9}$$

The term $\bigtriangledown_c (c - o_i)^T (c - o_i)$ is equivalent to:

$$\bigtriangledown_c (c - o_i)^T (c - o_i) = 2(c - o_i). \tag{B.10}$$

Since ρ_{o_i} is a function of $\theta_i(c)$, using chain rule, the gradient of ρ_{o_i} with respect to c is:

$$\bigtriangledown_c \rho_{o_i} = \bigtriangledown_{\theta_i} \rho_{o_i} \bigtriangledown_c \theta_i. \tag{B.11}$$

The gradient $\bigtriangledown_{\theta_i} \rho_{o_i}$ of ρ_{o_i} with respect to θ_i is:

$$\bigtriangledown_{\theta_i} \rho_{o_i} = \frac{\partial \rho_{o_i}}{\partial \theta_i} = -\frac{ab(a^2 - b^2)\sin(\theta_i)\cos(\theta_i)}{(a^2 \sin^2(\theta_i) + b^2 \cos^2(\theta_i))^{3/2}}. \tag{B.12}$$

The gradient of θ_i with respect to c is:

$$\bigtriangledown_c \theta_i = \begin{bmatrix} -\frac{c_y - o_{i_y}}{(c_x - o_{i_x})^2 + (c_y - o_{i_y})^2} \\ \frac{c_x - o_{i_x}}{(c_x - o_{i_x})^2 + (c_y - o_{i_y})^2} \\ 0 \end{bmatrix}. \tag{B.13}$$

Using Eq. B.12 and B.13, Eq. B.11 becomes:

$$\bigtriangledown_c \rho_{o_i} = -\frac{ab(a^2 - b^2) * \sin(\theta_i)\cos(\theta_i)}{(a^2 \sin^2(\theta_i) + b^2 \cos^2(\theta_i))^{3/2}((c_x - o_{i_x})^2 + (c_y - o_{i_y})^2)} \begin{bmatrix} c_y - o_{i_y} \\ c_x - o_{i_x} \\ 0 \end{bmatrix}. \tag{B.14}$$

In the second region, the gradient $\bigtriangledown_c \beta_i(c)$ of $\beta_i(c)$ with respect to c is:

$$\bigtriangledown_c b_{o_i} = 2(c - o_i). \tag{B.15}$$

APPENDIX C: TARROB HARDWARE AND SOFTWARE DETAILS

C.1. Hardware Setup

This section gives information about the connection diagrams of the hardware used on the system and the structures of the pins used. TarRob hardware consists of the following components:

- Mechanical part: A gantry robot with three degrees of freedom.
- Head and Nozzle System: Several different end effector components are designed. The robot can automatically pick up and release tools. The head and nozzles are mounted using magnets. The Seed Injector, Watering Nozzle, and Moisture Sensor Nozzle can be attached to the robot's end effector.
- Electromechanical part: The system is driven by four stepper motors, which operate across three axes. The stepper motors have the following specifications: Rated Voltage: 2.8V, Current: 1.68A, Holding Torque: 4.4 kg.cm.
  - X-axis: 2 NEMA-17 stepper motors
  - Y-axis: 1 NEMA-17 stepper motor
  - Z-axis: 1 NEMA-17 stepper motor
- Electronic components:
  - Processor: Jetson Nano
  - Arduino Mega
  - Ramps 1.4
  - ZED2 Stereo Camera
  - DHT22 Humidity and Temperature sensor
  - HC-SR04 Ultrasonic Sensor
  - 100GPH Bilge Water Pump
  - DC Air Pump
  - 220V AC Led Lighting and Mechanical Fixture

– 445nm 2.3W Focus-able Blue Laser Module Laser Sensor

The robot's system components are connected as shown in Figure C.1. The connections of Arduino Mega are detailed in Table C.1.

Table C.1. Arduino pin-out configuration.

ID	Hardware Name	Pin Number
1	X-Axis Step Motor-1	Step: D46, Direction : D48, Enable: D66
2	X-Axis Step Motor-2	Step: D26, Direction : D34, Enable: D36
3	Y-Axis Step Motor	Step: D60, Direction : D61, Enable: D56
4	Z-Axis Step Motor	Step: D54, Direction : D55, Enable: D38
5	X-Axis Limit Switch	D18
6	Y-Axis Limit Switch	D14
7	Z-Axis Limit Switch	D3
8	Light Relay Pin	D42
9	Moisture Sensor	Enable: D27 Data: D57
10	Water Pump	PWM: D8
11	Vacuum Motor	PWM: D9
12	BH1750 Light Sensor	SCL : D20, SDA: D21
13	DHT22 Sensor	D2
14	Laser Pin	D6

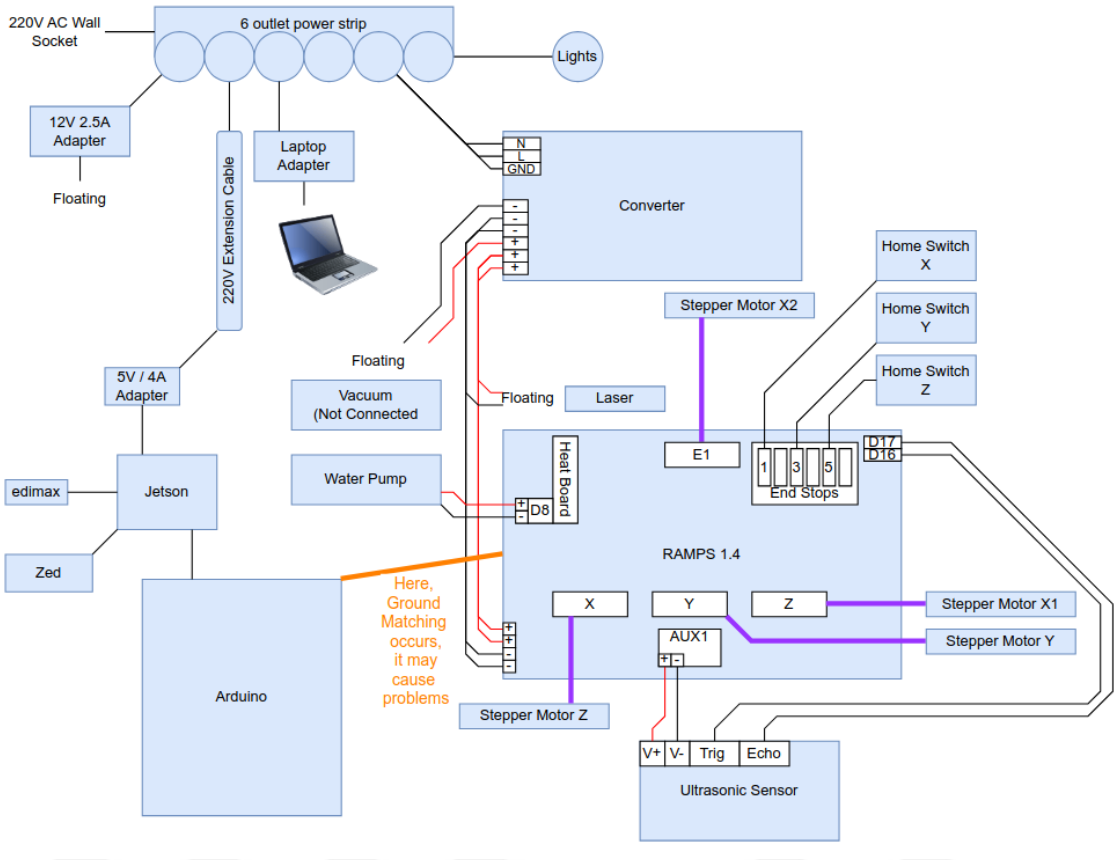


Figure C.1. Connection schematic of TarRob system components.

C.2. Software Organization

The general software structure of the robot consists of

- Jetson software:
- Arduino software
- Remote client software

C.2.1. Jetson Software

The software running on Jetson consists of several components. These components are designed to have different functionalities as listed as below:

- Network software
- ROS/C++ software
- Qt-Based GUI software
- MYSQL Database

C.2.1.1. Network Software. A Geekworm Jetson Nano Wi-Fi Adapter is mounted on the Jetson to create a local Wi-Fi network for communication between remote client computers running the Qt-based GUI, which sends commands to the main software on Jetson. Additionally, the main software can share the states of the TarRob system over the Wi-Fi network. Every time the system is restarted, clients wishing to send commands from a remote desktop must connect to this network and establish the necessary connections.

C.2.1.2. ROS/C++ Software. Through this project, general communication between algorithms running inside Jetson, interaction with the Arduino and RGB-D data acquisition from ZED camera are designed based on the ROS topic protocol. In this protocol, all desired information and command are shared via topics. The software has been developed using ROS and C++.

`/master_node` is the main software that manages all algorithm processes and data. They take commands from GUI using TCP protocol with determined packets, sending commands and reading data from Arduino using `/ros_serial` topics, and trigger related algorithms(Ellipsoid Fitting, APF Navigation) with topics. Arduino's sensor measurement data and other physical system features are also over ROS Serial communication between Jetson Nano. Data transfer is provided using ROS topics. ROS topic diagram of the software running on the main processor of the robot is as shown in Figure C.2.

Ellipsoid Fitting algorithm is developed as a ROS node. The ellipsoid fitting function is developed motivated by [51] project. There is an implementation of Khachiyan's algorithm [30]. General ROS topic diagrams of the algorithms is shown in Figure C.3. Ellipsoid fitting software subscribes to the `/depth_path`, `/plant_length`, `/plant_width`, `/plant_height` topics published by the Plant Growth Monitoring System algorithm [18], which was developed in a previous project.

C.2.1.3. MYSQL Database. In this project, databases have been used to store information about the plants maintained in the garden, log the performed operations, record the acquired sensor data, and manage both periodic and non-periodic tasks. Figure 2.7 illustrates the general database diagram structure. Additionally, Table 2.2 provides an explanation of the functions of each database.

To facilitate and manage these operations, the `MySQLHelper` class has been developed. This class includes methods for tasks such as initializing the MySQL database and reading data from tables, allowing essential operations to be executed efficiently within the main project. This approach has been adopted to ensure more reliable operation management and enhance fault tolerance.

First, a terminal is opened on Jetson, and the `sudo mysql` command is entered. The system database is accessed using the appropriate username and password. Then, the following commands are executed in sequence in the terminal to view the table names in the system database:

- `use AutofarmDB;`
- `show tables;`

The outputs seen in the system terminal are shown in Figure C.4. The descriptions of these tables are provided in Section 2.3.3.

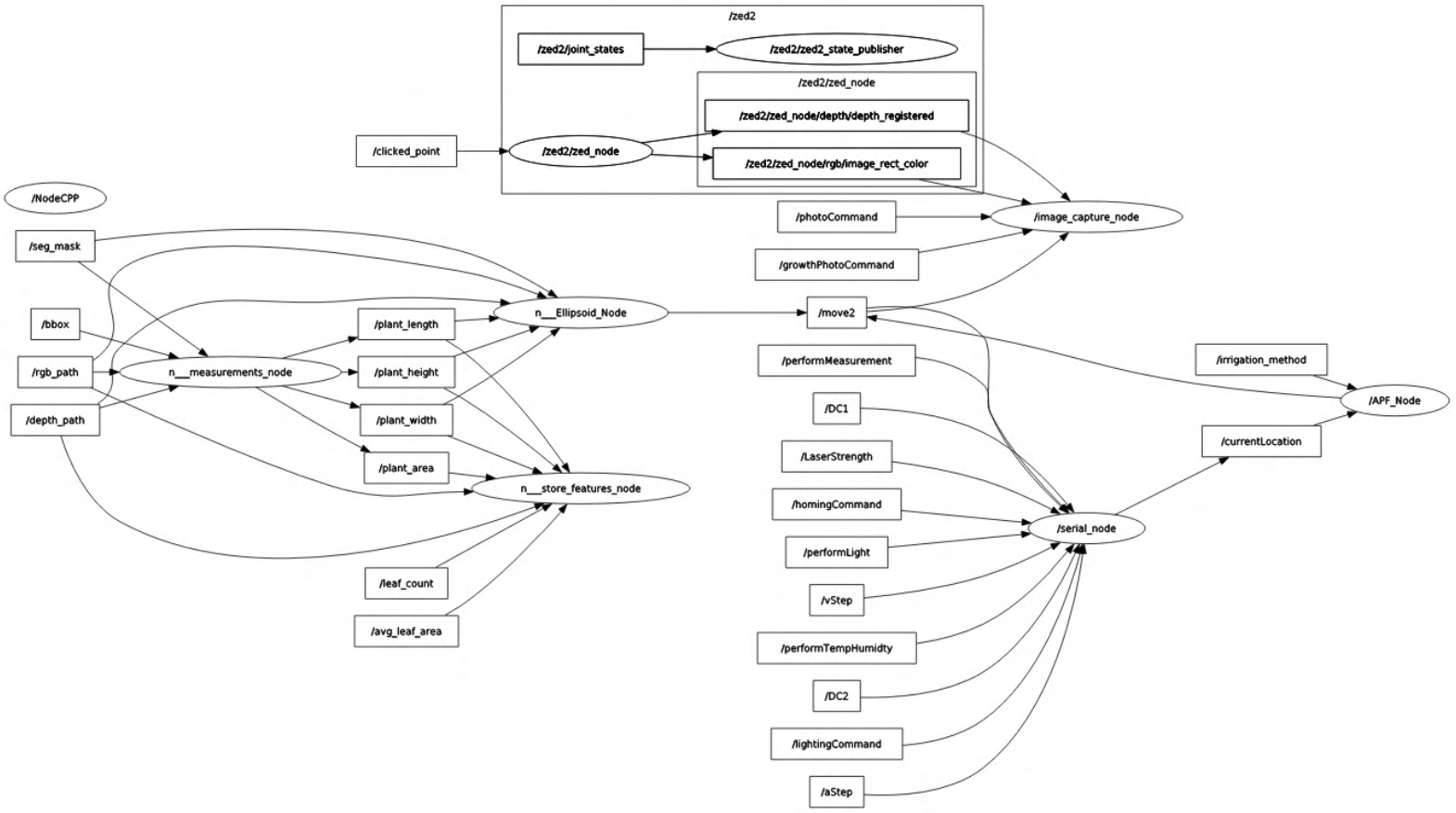


Figure C.2. ROS topic diagram of the entire system.

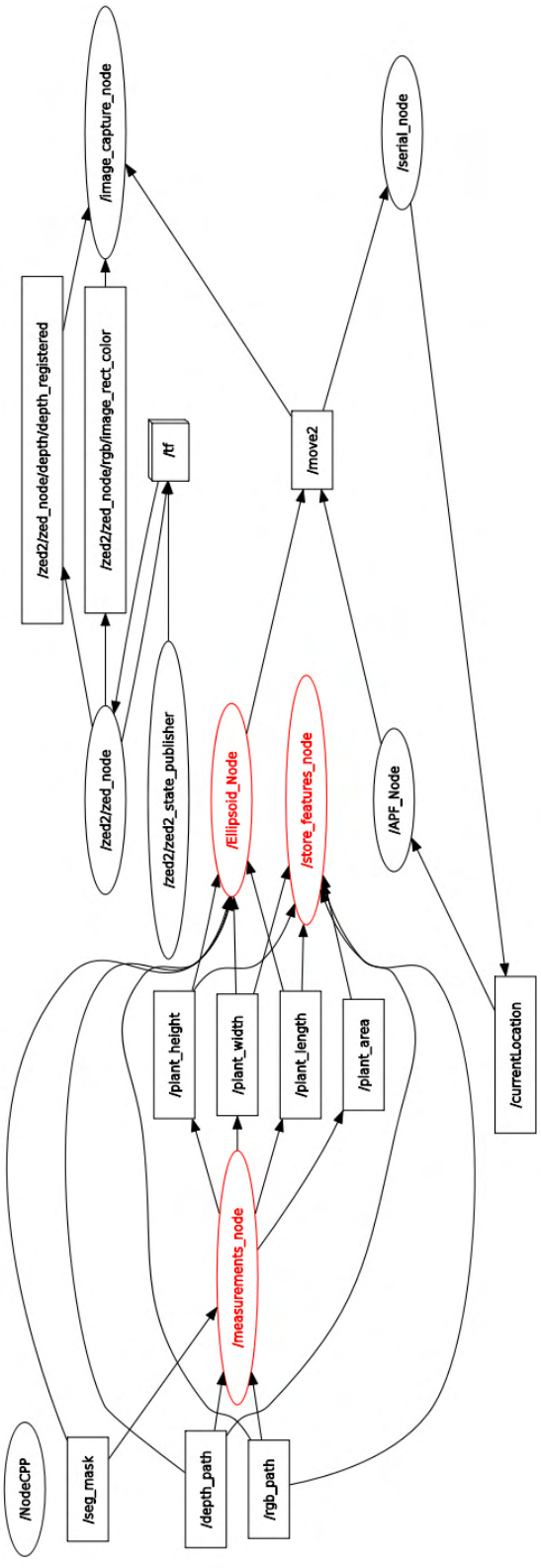


Figure C.3. ROS topic diagram of the ellipsoid fitting algorithm.

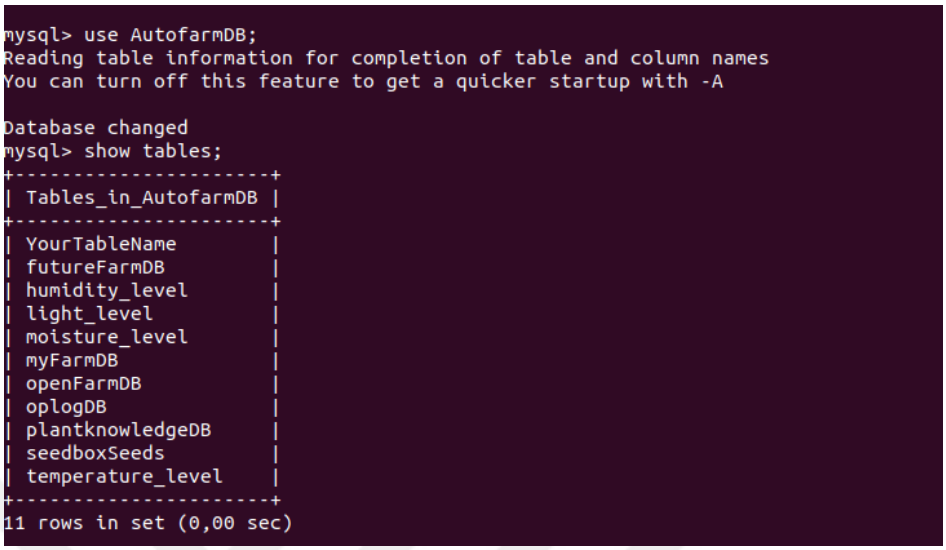


Figure C.4. MYSQL terminal outputs.

C.2.2. Arduino Software

The software developed on the Arduino processes commands received via ROS Serial. The main software running on Jetson is designed to execute commands coming from `/master_node`. The Arduino software handles stepper motor step calibration, sensor data filtering, reading the status of limit switches, and controlling the water and pump DC motors.

- Stepper Motors: Movement and calibration process of the stepper motor is developed in Arduino. The motors are controlled using the `MultiStepper` and `AccelStepper` libraries. The desired maximum speed in steps per second is set for the X, Y, and Z axes using the `setMaxSpeed` function as follows:

  $$\mathrm{X} = 2400, \quad \mathrm{Y} = 2400, \quad \mathrm{Z} = 3600.$$

  The desired acceleration in steps per second squared is set for the X, Y, and Z axes using the `setAcceleration` function as:

  $$\mathrm{X} = 1000, \quad \mathrm{Y} = 1000, \quad \mathrm{Z} = 1200.$$

The desired target positions for the three axes are provided using the `moveTo` function, which takes the desired position as parameters. Finally, the `run` function is called to move the system to the target positions.

- Sensors: The BH1750 light sensor is managed using the `BH1750.h` library available in Arduino. The DHT22 temperature and humidity sensor is handled using its dedicated `DHT.h` library. Moisture sensor readings are obtained by reading the analog signal, mapping the 0-1023 range to a meaningful 0-100 range. Limit switch states are read through digital input/output pins. Water and vacuum pumps are controlled using analog outputs, providing PWM signals to the DC motors.
- Jetson Nano: Communication with Jetson Nano is established using ROS topics.

For each operation to be performed, specific topics have been defined. Commands are detected and executed through these topics. The Arduino subscribes to the following topics:

- SubMove: The topic where the target position of the system is published.
- SubHoming: The topic corresponding to the home position command. When this command is received, the Arduino drives the stepper motors until they reach the limit switches.
- SubFiatLux: The topic used to control the LEDs on the system.
- SubDC1: The topic for receiving the PWM command required for the water pump.
- SubDC2: The topic for receiving the PWM command required for the vacuum pump.
- SubLaser: The topic where commands for the laser sensor are sent.
- SubMeasure: The topic where commands for moisture sensor readings are published.
- SubvSte: The topic used for setting the speed of the stepper motors.
- SubaStep: The topic used for setting the acceleration of the stepper motors.

- SubTempAndHumidity: The topic used to trigger measurements from the DHT22 sensor.
- SubLight: The topic defined for light sensor measurements. When this topic is received, the relevant measurements are performed.

The topics where data and system status obtained from the Arduino are published are as follows:

- CurrentLocation: The topic where the real-time positions of the stepper motors are published.
- Moisture: The topic where the measured moisture sensor values are published.
- TempLevel: The topic where the measured temperature sensor values are published.
- HumidLevel: The topic where the measured humidity sensor values are published.
- LightLevel: The topic where the measured light sensor values are published.

C.2.3. Qt-Based GUI Software

The Qt-based GUI software can send and read states using TCP communication. A service software has been developed to automatically create the Wi-Fi network after Jetson boots. QT-Based GUI consists of six different tabs. Through this thesis project, the Generic Task Tab has been developed and implemented as detailed in Section 2.3.2. The descriptions of the remaining tabs are as follows:

- Connection and Log: In this tab, the connection between the computer running the GUI and Jetson can be established or terminated by configuring the Jetson IP address and port settings and pressing the *Connect* or *Disconnect* button. Messages sent to the system and logs of system operations can be viewed in the *Communication Log* section. The system's autonomous mode can be toggled using the *Autonomous Mode* option.

In autonomous mode, the system checks the scheduled tasks in the database, while in manual mode, it listens for commands from the interface. When *Terminal Mode* is activated, the main software running on Jetson waits for input from the terminal. The system can be shut down using the *Exit and Turn Off* button, while the GUI can be closed using the *Quit* button. The general view of this tab is shown in Figure C.5.

- Homing - Movement - Photo - Laser: This tab allows manual control of the system via commands. The general view of this tab is shown in Figure iii.. Pressing the *Home* button returns the system to the home position. In the *Movement Commands* section, movements along the X-Y-Z axes can be controlled. The *Move* command first sets Z = 0 and then moves sequentially along the X and Y axes to the specified position. Finally, it moves along the Z-axis to the desired position, ensuring controlled movement. The *Move All Three Axis* command allows the system to move more aggressively along all three axes simultaneously. The *Move Relative* command enables movement relative to the current position by entering X, Y, and Z values. The *Take Photo* button captures a manual snapshot using the ZED camera. The *Laser Commands* section allows manual firing of the laser with a specified power level.
- Light - Vacuum - Tools: This tab includes the *Lighting Commands* section for manually turning system lights on and off. The *Vacuum Commands* section allows manual control of the vacuum motors. The *Tool Commands* section is designed for changing the system's end effectors.

  The *Get Current Tool* button retrieves information about the currently attached tool, while the *Leave Tool* command detaches it. The general view of this tab is shown in Figure iii..
- Water: This tab provides manual control of the water pump in the system. The *Duration* section allows setting the duration of water flow. The *Just Water* button activates the water pump at the system's current position for the specified time. The *Water Location* button moves the system to the specified X-Y-Z position before watering for the defined duration.

The *Water All* button sequentially waters all plants recorded in the database for the specified duration. The general view of this tab is shown in Figure iii..

- Plant: This tab includes functions developed for the system's capability to automatically plant seeds. The *Plant Seed* section contains a list of available tools for different seed types. The *SeedBox X-Y* fields specify the location where the seed will be picked and the target X-Y position where it will be planted. After configuring the necessary settings, pressing the *Plant Seed* button ensures the selected seed is planted at the desired position. The *Delete Plant* section allows removing plants from the database based on their IDs. The *List My Plants* button displays all plants recorded in the system. The general view of this tab is shown in Figure C.9.

C.3. TarRob Operation

The operation of TarRob is done as follows:

i. Connect the DC power supply to the system and power it on (make sure the Jetson Nano and Arduino show power indicators, such as LED lights).

ii. After powering on the system, connect to the "Jetson" network via Wi-Fi to operate the system remotely. Jetson Nano password is as follows:

   - Password: isl492492

   For this, the accompanying laptop may be used. Please make sure that the laptop is booted as to run in Ubuntu operating system. The laptop user id and password are as follows:

   - Password: isl492492

iii. Use the following command to access the Jetson Nano via SSH:

```
ssh -X jetson@10.42.0.1 gnome-terminal --disable-factory
```

   Jetson Nano password is as follows:

   - Password: jetson

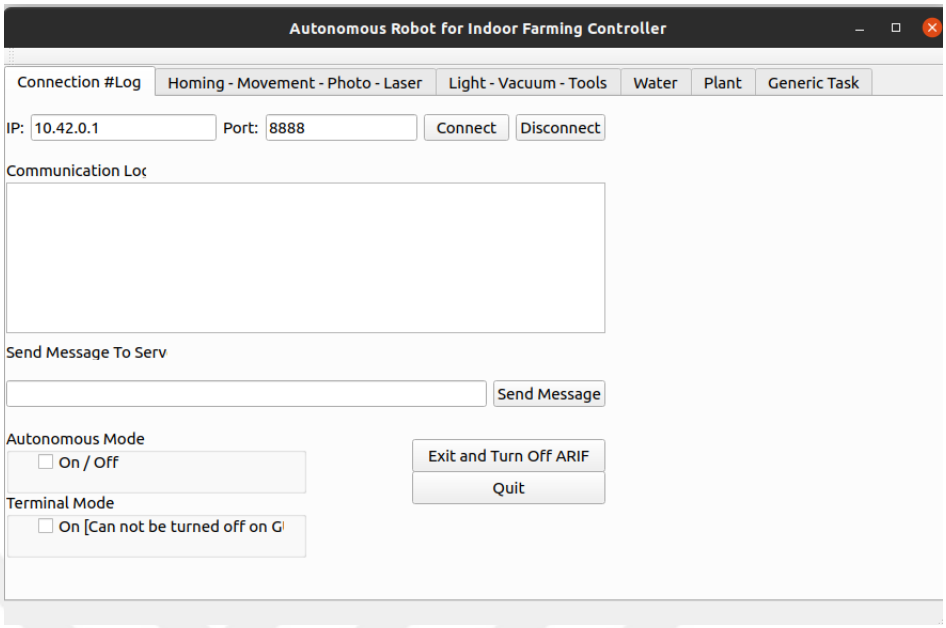


Figure C.5. Tab - Connection and Log.

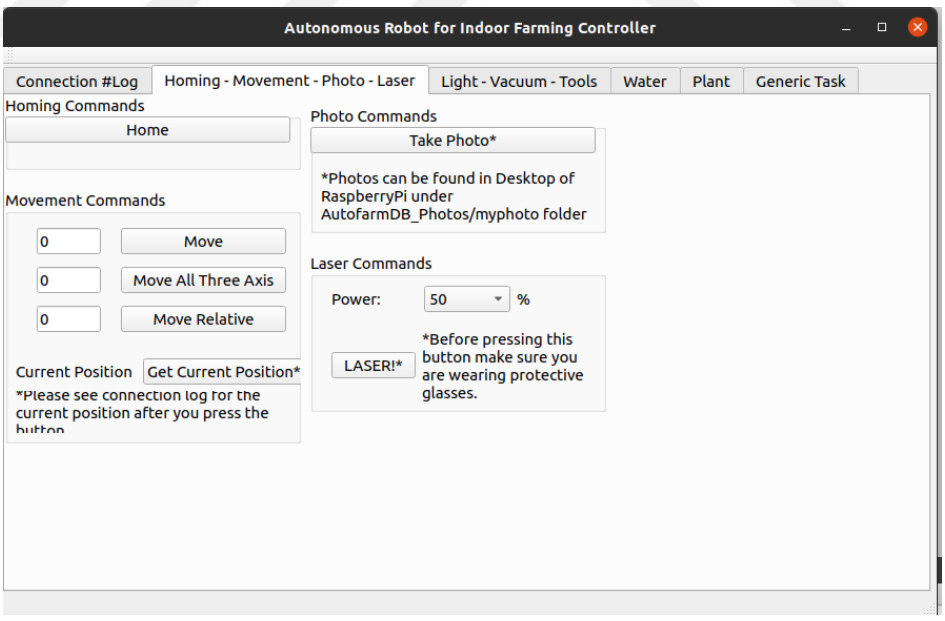


Figure C.6. Tab - Homing - Movement - Photo - Laser.

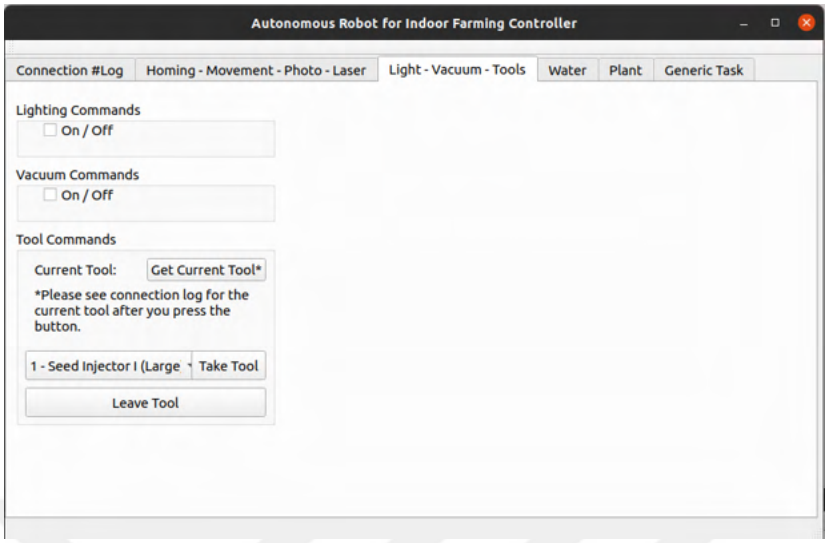


Figure C.7. Tab - Light - Vacuum - Tools.

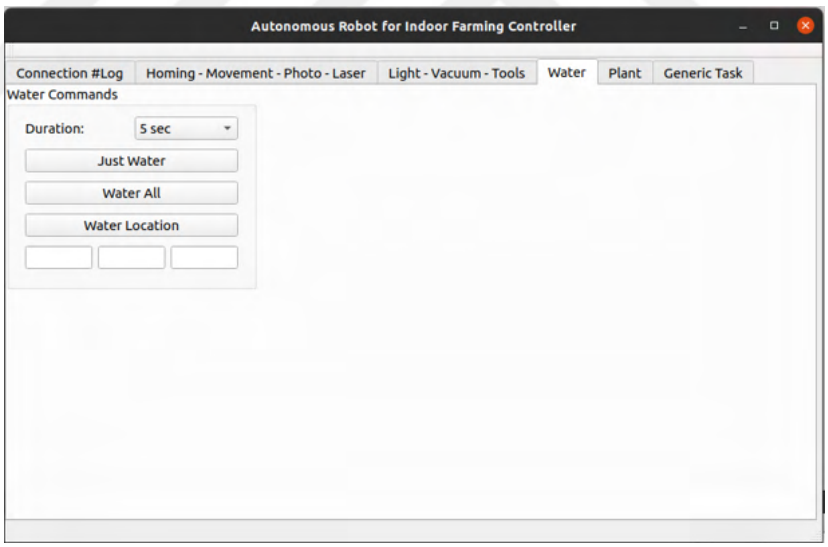


Figure C.8. Tab - Water.

iv. Once connected, navigate to the desktop and run the `capture_node.sh` script to activate the following nodes in order:

- `serial_node`
- `zed_wrapper`
- `capture_node`
- `MasterNode`

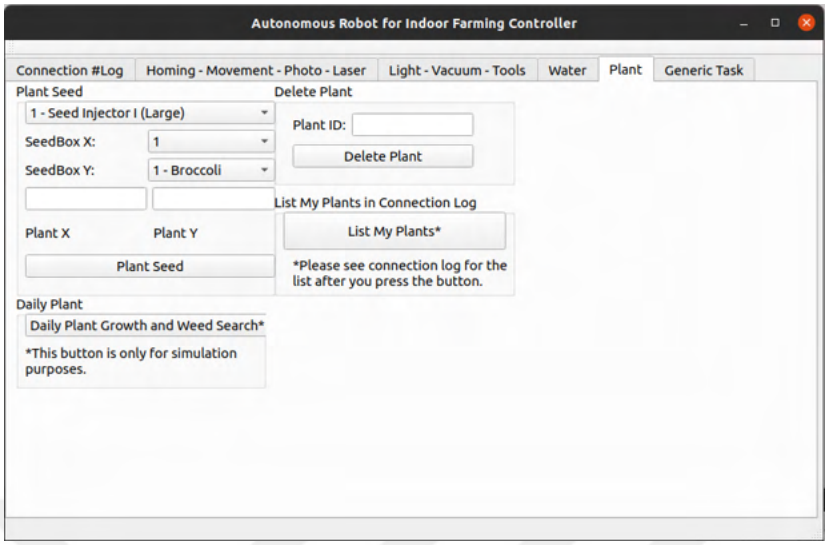


Figure C.9. Tab - Plant.

v. Launch the Qt interface from the remote desktop to connect to the system, making it ready to receive and execute commands.

C.4. 3D Point Cloud Map Generation

The software has been developed using ROS and C++. In this development, RGB-D data is combined to generate a point cloud model of the entire garden.

i. `roscore` – Start ROS Core

ii. `rosrun MapGeneration pointcloudmap_node` – Open a terminal and execute the command to activate the algorithm.

C.5. Ellipsoid Cylinder Fitting

The Ellipsoid Fitting algorithm is executed as RGB-D data is received, and the information about the generated cylinder is written to a text file. Execution steps are as follows:

i. `roscore` – Start ROS Core

ii. `./plant_growth.sh` – This script activates the nodes necessary to run the Plant Growth Monitoring System algorithm from the previous project [18].

iii. `rosrun mvee_pcd ellipsoid_node` – Open a terminal and execute the command to run the elliptic cylinder algorithm.

C.6. 3DW-APF Navigation Algorithm

The 3DW-APF navigation algorithm is executed as follows: The ellipsoid cylinders generated from the Elliptic Fitting algorithm are read from the file, creating a map of these ellipsoids. The system is executed by feeding target points defined within the created area. `APF_Node` subscribe `/currentLocation` and publishes `/move2` topic to move the system to target point.

i. `roscore` – Start ROS Core

ii. `rosrun APFNavigation APF_Node` – Execute the algorithm.