Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Row Detection and Graph-Based Localization in Tree Nurseries Using a 3D LiDAR

Ionut Vintu

Course of Study:	Computer Science
Examiner:	Dr. Daniel Hennes
Supervisor:	Dr. Ruth Schulz, Dr. Stefan Laible
Commenced:	December 6, 2017
Completed:	June 5, 2018

Abstract

The increasing need of eliminating pesticides and substituting chemical-based weeding techniques with manual and mechanical methods have gave way to the development of agricultural robots that will greatly improve and reduce the time spent on growing more healthy plant fields. Because most crops are cultivated in rows, the development of robust and reliable algorithms for the detection of plant rows and individual plants is the foundation for autonomous navigation within the plant fields.

A number of field robots have been developed by various research groups and companies during the past decades. Although they use expensive sensors for detecting rows, these methods lack a certain degree of robustness with regard to the variability of different fields. They are typically built with a specific project or purpose in mind and their design limits the possibility of using them for other purposes. Using big robot platforms that usually span several plant rows limits greatly the size of the rows and the size of the plants in many types of fields.

This thesis proposes instead an algorithm that makes use of cheaper sensors and has a higher variability by combining row detection algorithms with graph-based localization methods as they are used in Simultaneous Localization and Mapping (SLAM). The considered method focuses on row detection in different types of fields and on using graph-based localization to improve individual plant detection and deal with exception handling, like row gaps, which are falsely detected as end of rows.

Testing the developed algorithm in a variety of simulated fields shows that the additional information obtained from localization provides a boost in performance over methods that rely purely on perception to navigate. The framework built within the scope of this thesis can be further used to integrate data from additional sensors, with the goal of achieving even better results.

This Master thesis achieves the goal of implementing a robust perception algorithm that uses a LiDAR sensor and graph-based localization techniques. The entire framework allows a small-sized robot to navigate autonomously inside tree nurseries populated by trees of different shapes and sizes.

Acknowledgements

This Master thesis is part of a publicly funded project called Autonome Mechanische Unkraut-bekämpfung Robot - the autonomous weeding robot (AMU-Bot), initiated by the Fraunhofer IPA. The project is supported (was supported) by funds of the Federal Ministry of Food and Agriculture (BMEL), based on a decision of the Parliament of the Federal Republic of Germany via the Federal Office for Agriculture and Food (BLE) under the innovation support programme. The aim of the project is the development and demonstration of a cost-effective autonomous platform for the control of weeds in sensitive crops of different planting distances in tree nurseries. The robot platform is equipped with an inertial measurement unit, wheel encoders and a 3D LiDAR sensor.

Contents

List of Figures

List of Algorithms

List of Tables

List of Abbreviations

g2o A General Framework for Graph Optimization. 15

GPS Global Positioning System. 13

I/O lin Input/Output linearization. 15

IMU Inertial Measurement Unit. 14

LiDAR Light Detection and Ranging. 14

Mid Line Middle line between two 2D lines. 39

PCL Point Cloud Library. 15

PID Proportional Integral Derivative. 15

RANSAC RANdom SAmple Consensus. 15

ROS Robot Operating System. 15

SLAM Simultaneous Localization and Mapping. 15

1 Introduction

Agricultural robotics is a growing field. Farmers are pressured to keep up with the higher demand of products because of increasing world population. An increase in cost and use of pesticides demands a change in the way agriculture is done. A growing organic market and consumer demand for more natural products fuel the need for cleaner ways of growing crops. Agricultural robots could step in to help with the increasing workload and try to reduce the amount of toxic substances used on plants and affecting the environment. Research has been done also towards completely eliminating pesticides from agriculture by developing mechanical methods of removing weeds ([ÅB02], [THM02]).

Manual weeding is a hard and tedious work. The mean input of manual weed control in organic grown crops in the Netherlands is circa 45 hours per hectare for planted vegetables, but increases to more than 175 hours per hectare for direct-sown onion ([VBA+08]). In 1998, on average, 73 hours per hectare of organic grown sugar beet were spent on hand weeding in the Netherlands ([VLBG02]).

Autonomous robots seem to be a good alternative to deal with these problems by replacing tedious human work with small robots, which could work tirelessly throughout the day. Replacing manual weeding with autonomous robots could mean an enormous stimulus for organic farming. In the future, autonomous robotic weed control systems may replace human labor for hand weeding and may also provide a cost effective alternative for herbicides ([SGD08], [VBA+08]).

Considering that most fields are organized in rows, the detection of plant rows is the most important aspect with respect to autonomous navigation of agricultural robots inside fields of crops. A number of field robots have been developed by various research groups and companies during the past decades, such as the Autonomous Mechanisation System (AMS) ([BGN+04]), HortiBot ([RSM+07]), Weeding Robot ([BBM+10]) and BoniRob ([RBD+09]). Previous and current projects like AgriApps ([FFM+15]) and Flourish ([LPK+16]) use expensive sensors for detecting rows, and yet these methods lack a certain degree of robustness with regard to the variability of different fields. These projects focus on deploying the robots on fields with small crops, which allow the platform to drive above them.

Autonomous robot platforms specially designed for weed control have been described by [THM98], [ÅB02], [RSM+07] and [RKL+06]. These platforms use machine vision to navigate within rows of crops. Autonomous navigation with a robotic platform for agricultural field operations, including weed control, has also been carried out by absolute navigation with Global Positioning System (GPS) ([BJ04], [NRZ+98], [NUK+04], [BGN+04], [NGNS08]).

This Master thesis aims to develop a perception algorithm that allows a robust detection of rows and individual plants. Achieving this goal is done by using graph-based algorithms that provide a better pose estimation of the robot. This permits the creation of a local map for the robot to reliably

detect the position of individual plants needed for precision weeding. The local map can also help to better detect plant rows and avoid common problems in such scenarios, like false end of row detection, when there are gaps in the plant rows.

A small robot platform is used because it is better equipped for intra row weed management. It can be deployed in more versatile environments, which are represented by different distances between plant rows and also suitable for any dimensions of plants. When applied to outdoor applications, video cameras often encounter illumination changes. This is a common vulnerability of video sensors and it can greatly affect their performance. A 3D Light Detection and Ranging (LiDAR) is better equipped to deal with the challenge of illumination changes. Hence, it has an advantage over video cameras when it comes to outdoor applications. Given this reason, a 3D LiDAR is chosen as a visual sensor for the application developed in this thesis.

For testing the efficiency of the proposed methods, 100 fields of trees in different growth stages, arranged in rows, are simulated. Beech and cherry laurel plants are selected for simulation because they convey, in their different growth stages, the diversity a weeding robot might encounter in various fields. The simulated gardens vary in number of rows, plants' sizes and distance between rows. To ensure real life applicability, a test garden within Bosch Research Campus in Renningen will be used. The testing site resembles fields the robot is meant to work on. This is a tree nursery in a smaller scale, built with artificial plants arranged in rows.

Results show that combining methods that use LiDAR to detect plants and plant rows with graph-based localization offers a better performance in simulation, compared to methods that use just perception. The improvement is evaluated as upwards of 15 percent in terms of distance driven before the robot deviates from its desired path. The results of real life tests show that a robotic platform equipped with LiDAR, Inertial Measurement Unit (IMU) and wheel encoders is capable of navigating with accuracy in plant fields, if prior knowledge about the field's dimensions is provided.

Outline

Chapter 2 introduces the notions related to the topic of this Master thesis and how they take effect in the related work. The next section, Chapter 3, offers an explanation of the way this project was designed, going into details about the platform and the developed algorithms. Following this, the setup and results of the evaluation are detailed in Chapter 4. Finally, the findings and discussion over the choices that have been made in the implementation are presented in Chapter 5.

2 Background

This section introduces some concepts that will be useful in the following chapters, focusing also on how they relate with the work of robotics in agriculture. The chapter starts by introducing the field of agricultural robotics and presenting some of the research done in this field. Next, the tools that are essential to what this Master thesis proposes are presented. LiDAR data is used and processed with the help of Point Cloud Library (PCL). The information given by these methods that use RANdom SAmple Consensus (RANSAC) at their core is later used to localize a robot inside a field of plants. The accurate navigation inside the field is handled by one of two types of controllers: Proportional Integral Derivative (PID) and Input/Output linearization (I/O lin). These parts come together with the help of software tools like Robot Operating System (ROS) and Gazebo paired with PCL and A General Framework for Graph Optimization (g2o).

2.1 Agricultural Robots

The majority of autonomous agricultural robots developed so far are comprised of big platforms equipped with a multitude of sensors that drive above the crops ([JNJ+12]). Most robots navigate inside the rows using visual data from cameras. This information is processed by using Hough Transform ([AAIF10], [LD06], [HMT00]) or pixel histogram ([TGZZ13]) methods to detect the row line. Field navigation is typically done using a GPS sensor and a prerecorded map ([BJ04], [WB11]). Some projects work towards GPS-free navigation ([WB10]), using 3D LiDAR to detect and classify gaps and end of rows in unfamiliar terrain. These projects are tested in fields with small plants, allowing for the platform to drive above the rows. These large machines cause significant soil compaction damage. Moreover, they are inherently not made for driving inside tree nurseries in which plants could reach up to 4 meters high.

Small platform robots that can easily drive through such fields are being developed for different applications. The Trakür ([Mas14]) follows a 1 mm copper cable, installed on top of or just below the soil's surface, that carries a mild electric charge of less than one volt to navigate in gardens. The Oz Robot ([GBM+]) is deployed in small plant gardens and uses LiDAR technology to navigate between rows of small plants. To detect the end of rows, the Oz Robot requires red rods to be planted at the end of rows. Some vineyards robots use GPS and laser scanners to navigate through very structured vine rows ([LPB+10], [DT+15]). When steep terrain makes it hard for the robot to localize itself, some robots may use Radio-Frequency IDentification (RFID) tags with location information of the plants as landmarks for Simultaneous Localization and Mapping (SLAM) methods ([DSC+16]).

This thesis addresses the need for a small robot, that can work in different types of field with trees of varying sizes, even up to 3-4 meters. The robot should be able to reliably navigate through the detected plant rows, correctly detecting the end of rows and overcoming gaps in between the plants. This approach should also be capable of individual plant detection which is necessary for inter row weeding.

2.2 LiDAR

LiDAR (also called LIDAR, Lidar, and LADAR) is a surveying method that measures distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor. Differences in laser return times and wavelengths can then be used to make digital 3D representations of the target. There are two types of LiDAR sensors: 2D, which have a single laser beam that spins around, giving distance information in a single plane and 3D. In order to have 3D information, the LiDARs can either have more beams spinning in different planes, offering more information about the environment or by using a single beam that changes direction with a small mirror. Using just one beam to get 3D information restricts the field of view of these kinds of sensors to just a small window to the world.

Agricultural robots equipped with LiDARs have been used for a variety of purposes in agriculture, ranging from seed and fertilizer dispersions, sensing techniques as well as crop scouting for the task of weed control. Controlling weeds requires identifying plant species. A preprocessing step needs to analyses the visual data and classify a plant as a weed or as a crop plant. This can be accomplished by combining 3D LiDAR data and machine learning algorithms, as presented in [WBL+10]. LiDAR produces plant contours as *point clouds* with range and reflectance values. This data is transformed, allowing plant specific features to be extracted, thus having a basis for labeling the plant species. This method is efficient because it uses a low-resolution LiDAR and machine learning approaches. It includes an easy to compute feature set with common statistical features which are independent of the plant size.

In robotics, LiDAR is often used to detect the obstacles' position and velocity, in order to safely navigate through environments ([PQZ+15]). Another application is crop mapping in orchards and vineyards, to detect foliage growth and the need for pruning or other maintenance, detect variations in fruit production, or count plants. LiDAR is useful in GPS-denied situations, such as nut and fruit orchards, where foliage blocks GPS signals necessary for precision agriculture equipment or a driverless tractor. LiDAR sensors can detect the edges of rows, so that farming equipment can continue moving until GPS signal is reestablished.

The works of [BMIN07] and [SBA06] show successful results of equipping large tractors with 3D LiDAR and cameras for vision that allow the vehicles to drive autonomously between tree rows in orchards, but do not address the problem of gaps or individual tree detection.

2.3 RANSAC

RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, which should have no influence on the values of the estimates ([FB87]). The algorithm considers a small initial set that is the minimum number of points that can describe the given model and creates a model from that set. This is called a consensus set. Then the algorithm tests all other points for fitting the proposed model with a certain threshold, adding the points that fit into the consensus set. At the end of one iteration, it randomly selects a new initial set and repeats the procedure. This is done a given number of times. The model with the biggest

Input:
$data$ – a set of observed data points
$model$ – a model that can be fitted to data points
n – minimum number of data points required to fit the model
k – maximum number of iterations allowed in the algorithm
t – threshold value to determine when a data point fits a model
d – number of close data points required to assert that a model fits well to data
Output:
$bestfit$ – model parameters which best fit the data (or null if no good model is found)

```
1  iterations = 0
2  bestfit = NULL
3  besterr = something really large
4  while iterations < k do
5      maybeinliers ← n randomly selected values from data
6      maybemodel ← model parameters fitted to maybeinliers
7      alsoinliers = ∅
8      for point ∈ data ∧ point ∉ maybeinliers do
9          if point fits maybemodel with an error smaller than t then
10             add point to alsoinliers
11     if size(alsoinliers) > d then
12         bettermodel = model parameters fitted to all points in maybeinliers and alsoinliers
13         thiserr = a measure of how well model fits these points
14         if thiserr ≤ besterr then
15             bestfit ← bettermodel
16             besterr ← thiserr
17     iterations + +
18 return bestfit
```

Algorithm 2.1: RANSAC algorithm

number of consensus points is considered the correct one. At the end, the RANSAC method could also employ a smoothing technique, such as least squares, to compute an improved estimate for the model's parameters.

The reasoning behind this concept can be better understood in algorithm form as presented in Algorithm 2.1.

Given a data set and a model, for example when a line needs to be detected (graphical example in Figure 2.1), the minimum number of points required to fit a model (n) would be 2. In every one of the maximum k iterations, $n = 2$ points are chosen randomly from the data set and a line is parameterized using those values. Then, for every point in the data set, it is verified weather the distance from the point to the proposed line is below the threshold t. All the points that are close enough are considered inliers and the others, outliers. If this line parametrization has enough inliers

(a) Data set **(b)** Conventional fitting **(c)** RANSAC fitting

Figure 2.1: RANSAC line fitting. Given a data set like in (a), conventional fitting methods would be greatly affected by the ouliers resulting in an erroneous line model (b). The RANSAC algorithm's robustness to outliers offers a much better estimates of the line given such a dataset (c).

(more than minimum d) and is better that the previous best, in terms of how the model fits the points, this parametrization is considered as the best. At the end of k iterations, the best parametrization for our data set is obtained.

The RANSAC algorithm has been shown to provide good results when trying to detect basic shapes in unorganized point clouds ([SWK07]). Agricultural robots have used this algorithm to efficiently navigate between plants rows by detecting lines in data provided by horizontal LiDAR ([RMA+16]) and to detect the ground plane ([WB11]).

2.4 Graph-Based Localization

For any mobile robot that needs to navigate, the problem of localization is posed. In order to avoid obstacles and to drive from point A to point B, the robot needs an accurate estimation of its position inside the environment, usually represented through a map. In case the map is also unknown, it is said that the robot deals with SLAM ([DB06]). For the case of this thesis, where the map is considered to be already known, the only task that remains is localization, i.e. determining the pose, position and orientation of the robot in the map, based on the information received from the sensors installed on the robot.

Using information from the sensors that track the robot's movement, one can calculate the current position. This can be done by using a previously determined position and advancing it, based upon known or estimated speeds, over elapsed time and course. This process is called dead reckoning and is subject to cumulative errors.

A better way to estimate one's position in a given map is to incorporate other information, such as visual or GPS, as presented in [AK06]. The information gathered from the different sensors usually gives divergent results for the robot's pose or its movements. Obtaining a unified result that is consistent with all the data is basically an optimization problem. There are two types of information describing where the robot is located in the world. The first type comes from visual sensors, which provide measurements relative to the robot's surroundings. The second information is obtained from motion sensors, which measure how the robot moves. All these measurements impose constraints on the the actual position and orientation of the robot in its environment. A way to solve this optimization problem is to formulate these constraints as a graph.

In the graph-based SLAM paradigm (explained in the works of [LM97], [FC04], [TM06], [DK06], [GKSB10]), each pose of the robot is represented as a *node* in a graph (x_i depicted in Figure 2.2 as a blue triangle). Most cases of robot localization and mapping deal with motion in 2D space, where the *pose* of the robot is described by $\phi = (x, y, \theta)$, with x and y being Cartesian coordinates and θ the angle made between the robot's facing direction and the world frame. In general, the nodes may also represent features in the world, such as interest points extracted from images or laser point clouds ([RKGB11]). They may also represent physical landmarks (depicted in Figure 2.2 as green plants) such as trees or structures in the world. A factor between two nodes is represented by an *edge* in the graph (denoted by (2) in Figure 2.2). The factor may represent rigid body transformations for pose-to-pose constraints or bearing measurements for features.

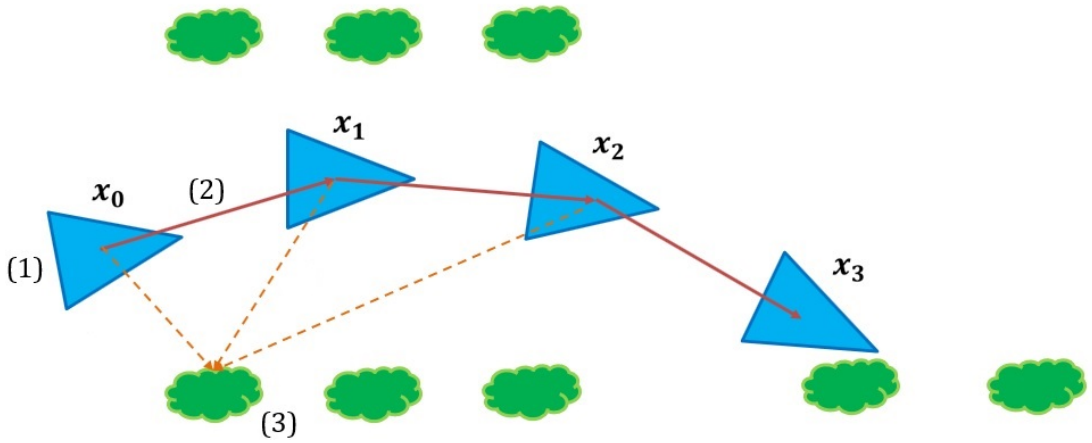


Figure 2.2: Graph-based localization.
The elements present in graph based localization: (1) - nodes, (2) - edges, (3) - landmarks.

The first part of the overall problem is to create a graph (factor-graph), by identifying nodes and factors between the nodes based on sensor data. Such a system is often referred to as the *front-end*. The primary function of a graph-based SLAM front-end is to identify the geometrical relationships between multiple nodes in the graph, by interpreting sensor data from perception sensors.

The second part entails finding the configuration of the nodes that best explains the factors. This step corresponds to computing the maximum-likelihood map and a system solving it is typically referred to as a *back-end*. The back-end aims to find the configuration of the nodes that minimize the error induced by factors from the front-end.

2.5 Controllers

One subfield of control systems engineering is control theory. This area deals with the control (regulation) of continuously operating dynamical systems in engineered processes and machines. The objective is to develop a control model for regulating such systems. This purpose is achieved by using an adequate action in an optimum manner without delay or overshoot. The taken action should also ensure control stability.

To do this, a *controller* with the necessary corrective behavior is required. This regulator monitors the process variable (PV) that needs to be controlled and compares it with the reference or set point (SP). The difference between actual and desired value of the process variable, called the error signal, or SP-PV error, is applied as feedback to generate a control action to bring the desired process variable to the same value as the set point. A graphical representation of the relation between the different parts of a control system can be observed in Figure 2.3.

Controllers range from the most simple, where the command is either maximum or none, basically having the process be on or off, to controllers that take into account the mathematical model of the process and can even adapt to variation over time of this model. In the middle of this spectrum we can find controllers such as the PID, presented in Section 2.5.1, and I/O lin, detailed in Section 2.5.2.

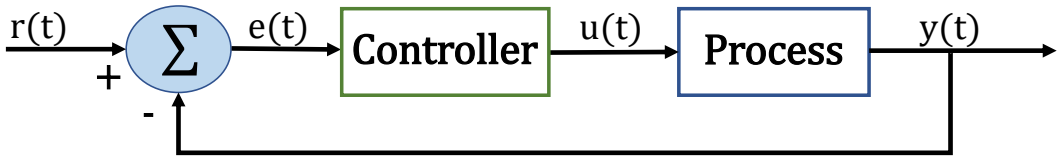


Figure 2.3: Block diagram of a controller in a feedback loop.
r(t) is the desired process value or setpoint (SP),
y(t) is the measured process value (PV),
e(t) is error between the setpoint and the process value, SP-PV,
u(t) is the command generated by the controller to get the process variable to the setpoint.

2.5.1 PID

A proportional–integral–derivative controller (PID controller or three term controller) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value $e(t)$ as the difference between a desired setpoint $(SP) = r(t)$ and a measured process variable $(PV) = y(t)$

$$e(t) = r(t) - y(t) \tag{2.1}$$

and applies a correction $u(t)$ based on proportional, integral, and derivative terms (denoted P, I, and D respectively), which give the controller its name. The overall control function can be expressed mathematically as

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.2}$$

where K_p, K_i, and K_d, all non-negative, denote the coefficients for the proportional, integral, and derivative terms respectively (sometimes denoted P, I, and D).

The block diagram in Figure 2.4 shows the principles of how these terms are generated and applied. It shows a PID controller, which continuously calculates an error value $e(t)$ and applies a correction based on proportional, integral, and derivative terms. The controller attempts to minimize the error over time by adjusting the control variable $u(t)$ to a new value determined by a weighted sum of the control terms.

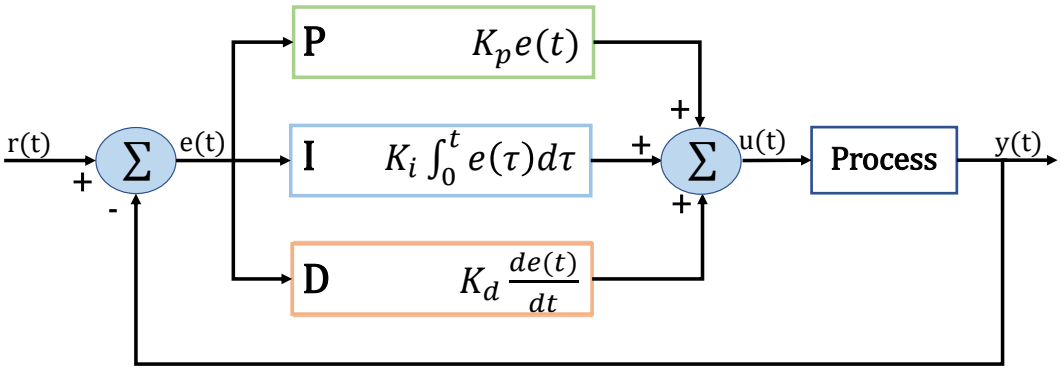


Figure 2.4: Block diagram of a PID controller in a feedback loop.
r(t) is the desired process value or setpoint (SP),
y(t) is the measured process value (PV),
e(t) is error between the setpoint and the process value SP-PV,
u(t) is the command generated by the controller to get the process variable to the setpoint.

Several projects that deal with navigating between tree rows show good results when using PID controllers to keep a steady trajectory in orchards ([SBA06], [LRWH14]).

2.5.2 Input/Output Linearization

A well-known systematic approach to the design of trajectory tracking controllers is based on Input/Output linearization via feedback. The works of [DOV01] show that this controller has good results for differentially-driven mobile robots, where the kinematic model can be written as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega. \tag{2.3}$$

where v and ω are the linear and respectively the angular velocities of the robot.

Given the Cartesian coordinates of a point B located along the roll axis of the robot at a distance $|b|$ from the base footprint:

$$\begin{aligned} y_1 &= x + b\cos\theta, \\ y_2 &= y + b\sin\theta, \end{aligned} \tag{2.4}$$

where x and y are the coordinates of the robot's base footprint, the time derivatives of y_1 and y_2 are

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \mathbf{T}(\theta) \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{2.5}$$

Matrix $\mathbf{T}(\theta)$ has determinant b and is therefore invertible under the assumption that $b \neq 0$. It is then sufficient to use the following input transformation

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \mathbf{T}^{-1}(\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\frac{\theta}{b} & \cos\frac{\theta}{b} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.6}$$

to put the equations of the robot's movement in the form

$$\begin{aligned} \dot{y_1} &= u_1 \\ \dot{y_2} &= u_2 \\ \dot{\theta} &= \frac{u_2\cos\theta - u_1\sin\theta}{b}. \end{aligned} \tag{2.7}$$

An input/output linearization via feedback has therefore been obtained. At this point, a simple linear controller of the form

$$\begin{aligned} u_1 &= \dot{y}_{1d} + k_1(y_{1d} - y_1) \\ u_2 &= \dot{y}_{2d} + k_2(y_{2d} - y_2) \end{aligned} \tag{2.8}$$

with $k1 > 0$, $k2 > 0$, guarantees exponential convergence to zero of the Cartesian tracking error, with decoupled dynamics on its two components. y_{1d} and y_{2d} are the Cartesian coordinates of points along the desired trajectory, which should be followed by point B.

Figure 2.5 shows the relation between the robot and point B.

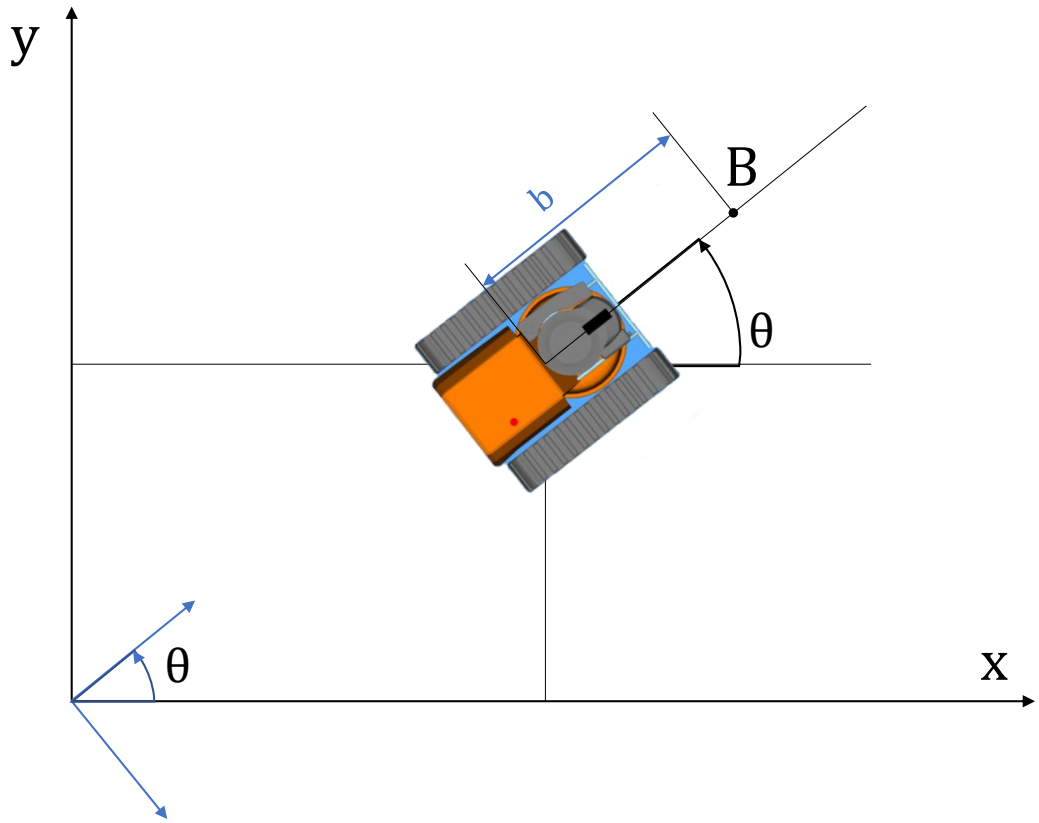


Figure 2.5: Graphical representation of input/output linearization controller variables. The picture depicts the robot's pose, with x and y being Cartesian coordinates and angle θ the robot's facing angle. The point B at distance b in front of the robot will follow a given trajectory.

2.6 Software Frameworks

In this section, several software frameworks are presented. These are often adopted in robotic applications. ROS ([QCG+09]) is especially popular in the robotics community, providing a versatile framework for robot software. ROS is often paired with Gazebo, which allows accurate 3D simulations that are essential before deploying algorithms on real life applications. PCL offers a wide range of methods and algorithms designed for point cloud processing. Finally, g2o ([KGS+11]) is presented as an optimization framework for solving nonlinear function minimization needed in graph-based localization (presented in Section 2.4).

2.6.1 ROS

ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. The main task of the framework is the data exchange between

different processes. The processes in ROS are called *nodes* and the data for exchange is called *messages*. A node sends a message by publishing it to a given *topic*, to which other nodes can subscribe to get the message information. The management of these data exchanges is being done in the background by the *roscore*. One helpful and important tool for the development of the navigation stack is Rviz. This tool is used for 3D visualization of different data types in ROS. Typical data for visualization are 2D laser scans, 3D point clouds, camera images or position data. Further on, additional data can be made visible by markers.

2.6.2 Gazebo

Gazebo is an open source 3D simulation environment for robots, which is built on top of the Open source Dynamics Engine (ODE) and the Open source 3D Graphical Rendering Engine (OGRE). ODE is a very fast, powerful, robust and flexible physical engine especially for simulating vehicles and objects in virtual reality environments. It also has integrated collision detection with friction. The vehicle and other objects are described with the Unified Robot Description Format (URDF), which uses the XML-Format. In this description, friction of the wheels or effort and limits of a joint can be defined. This is important to design a model as close as possible to reality. ROS supports this simulator very well by having a bridge for sending and receiving messages between a ROS-node and the simulation environment. There is also a plugin system applied, where new sensors and actuators can be integrated. By using a standard message format, data from the simulated sensors and the position of the robot can be visualized by Rviz.

ROS and Gazebo are often used together to simulate and test robot configurations before using it in the real world ([RMA+16]).

2.6.3 PCL

PCL is a free licensed library for handling n-D Point Clouds and 3D geometry processing ([RC11]). PCL is fully integrated with ROS. PCL is a fully templated, modern C++ library for 3D point cloud processing used in modern CPUs. It is based on efficiency and performance-oriented algorithms. From an algorithmic perspective, PCL is meant to incorporate a multitude of 3D processing algorithms that operate on point cloud data, including: filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration and more. Each set of algorithms is defined via base classes. These attempt to integrate all the common functionality used throughout the entire pipeline. Using such classes keeps the implementations of the actual algorithms compact and clean.

2.6.4 g2o

g2o is an open-source C++ framework for optimizing graph-based nonlinear error functions. It has been designed to be easily extended to a wide range of problems and a new problem can be specified in a few lines of code. The current implementation provides solutions to several variants of SLAM and Bundle Adjustment. This library handles the back-end part of graph-based optimization problems, such as the one introduced in Section 2.4.

3 Implementation

This chapter describes the robot setup and how the different parts work together. Four methods are developed for detecting the rows of plants. Each new method is built upon the weaknesses of previous ones.

3.1 Robot Platform

The robot that will be used as driving platform is a RoboFlail Mini, seen in Figure 3.1. This is a small mower with hybrid drive that is able to drive on small hills, up to a 45 degree angle. Its dimensions are: 50 cm high, 90 cm wide and 115 cm long. Its wheels are equipped with continuous tracks, also called tank treads, which makes it highly robust to drive in a variety of gardens. This mower serves as a basis upon different sensors will be added to provide visual and motion information.

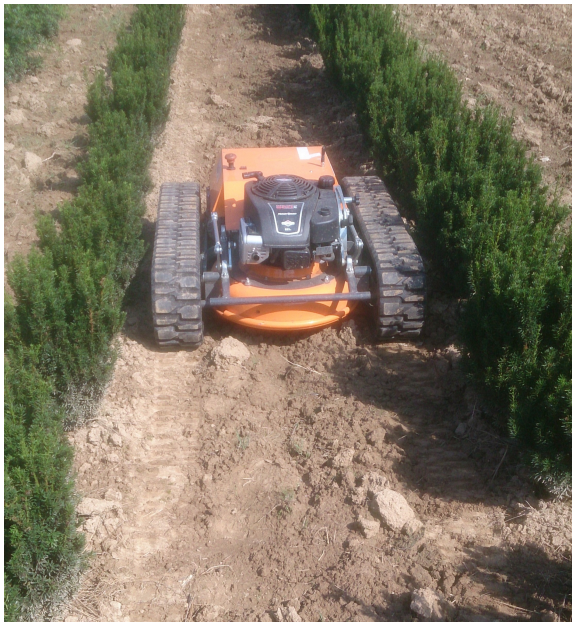


Figure 3.1: Robo Flail Mini lawn mower inside a tree nursery.

For sensing movement, the robot is equipped with wheel encoders. Counting how much each wheel turns gives information about the distance and the direction the robot moves. This way of obtaining movement information is prone to errors caused by the wheels slipping. In addition, an IMU gives acceleration information for each of the 3 coordinates of space. Combining these two movement information sources in a better estimate of the robot's true states.

A 3D LiDAR is used for visual information. The FX10 is a medium cost sensor that is usually used for obstacle detection. It functions using a single laser beam that is focused on a grid of points by a small moving mirror, in a Lissajous pattern depicted in Figure 3.2. It has a detecting range of 60 by 50 degrees, from 30 cm up to 15 m. The sensor has three settings. It can function at 16, 10 and 4 frames per second with a resolution of 53 by 33, 65 by 40 and 100 by 60 points receptively. For the tests, the sensor is set at a resolution of 65 by 40 points, received every tenth of a second. These values are chosen because it offers a good balance between number of points per scan and frequency of scans obtained by the sensor.

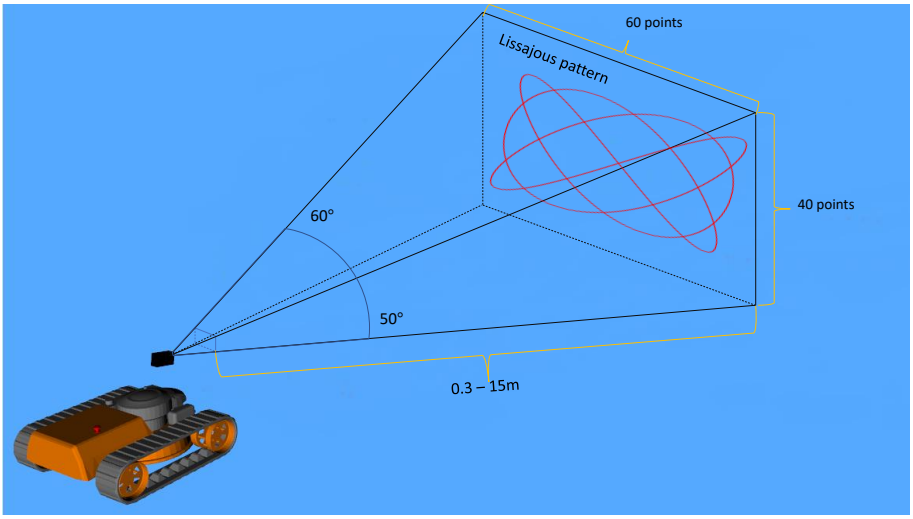


Figure 3.2: FX10 LiDAR Parameters. The range of the sensor is between 0.3 and 15 meters. The range is 60 degrees horizontal and 50 vertical with a resolution of 65 by 40 points. The laser creates a Lissajous pattern when obtaining the 3D and intensity information for the 65x40 points.

3.2 Software Modules

The software implementation consists of three main software modules: amubot perception, amubot navigation and amubot localization. These modules process the information that the robot receives from the sensors and allow it to drive within the fields. The simulation and testing also use amubot gazebo, amubot tools and amubot odometry. These modules provide simulated information that allow testing in simulated gardens. The connection between the modules and the information flow within the framework can be seen in Figure 3.3. The modules that are highlighted are created from scratch for the purpose of this Master thesis. The orange module and lines represent the real robot, that bears the name of the project: AMU-Bot. The blue modules are used only in simulation. Green modules are meant to work both in simulation and on the real robot.

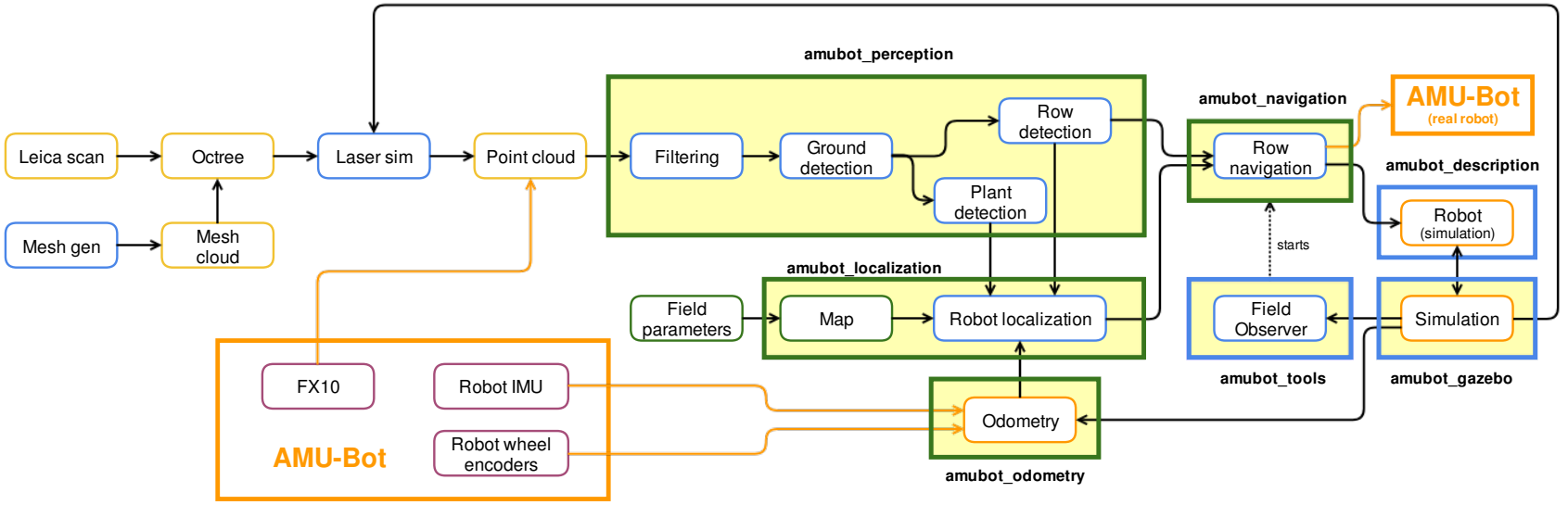


Figure 3.3: Software modules. The core modules that deal with perceiving and moving inside the world, be it real or simulated are: *amubot_perception*, *amubot_localization*, *amubot_navigation*. Modules used for testing are: *amubot_description*, *amubot_gazebo*, *amubot_tools*, *amubot_odometry*. The modules highlighted with yellow are developed for the purpose of this thesis.

3.2.1 Perception Module

This module deals with the point cloud information received from the FX10 sensor.

Most sensor data is affected by *noise*, which results in incorrect values. A way to eliminate this unwanted false information from sensor data is by applying a filtering method. The visual information is preprocessed by removing noisy data with a statistical and a radial filter.

The 'SOR filter' (Statistical Outlier Removal) of the PCL library was used. It computes first the average distance of each point to its neighbors (considering k nearest neighbors for each point - k is the first parameter). Then it rejects the points that are farther than the average distance plus a number of times the standard deviation (second parameter).

The 'ROR filter' (Radius Outlier Removal) of PCL removes all indices in its input cloud that do not have at least some number of neighbors (k is the first parameter) within a certain range (r is the second parameter).

We use a RANSAC method from PCL to detect the ground plane. RANSAC utilizes two parameters to run: the maximum number of iterations run before choosing the best fit and the distance threshold for a point to be considered belonging to the plane. The method is applied to the point cloud provided by the FX10 sensor at each time step. The output are the inliers and the outliers of the plane, with coefficients given in Hessian form:.

$$ax + by + cz + d = 0 \tag{3.1}$$

To avoid the effect of wrongly detected planes, the plane coefficients are checked. The detected plane should not make an angle with the robot's frame larger that a set threshold, in the experiments set at 45 degrees. Furthermore, the final plane coefficients are obtained by averaging over the last N good values (for this work, N has a numerical value of 10).

From the averaged coefficients, a 3-axis frame is created to describe the plane. The X-axis is set by projecting the robot's center and a point in front of the robot (a point on the robot's X-axis) on the plane and connecting them. The Z-axis is computed directly as the normal of the plane. The last axis, Y, is computed as the cross product between the X and Z -axis.

After the plane detection, the outliers, points not belonging to the plane are considered from now on belonging to potential plants. This is done by transforming the current scan in the plane's frame and eliminating the points that have a value of Z lower than the distance threshold used for detecting the plane i.e. the points belonging to the ground. The resulting point cloud is used as input for the developed methods to detect the plant rows. The output of these methods are line coefficients describing the rows of plants. These line coefficients represent a point on the line and the line direction. From these coefficients, poses are created in the frame of the robot and sent as ROS messages to the navigation and localization modules.

3.2.2 Localization Module

This module implements graph-based localization. It uses information about the position of the plant rows given by the perception module, together with the odometry data, in order to estimate the position of the robot inside the fields. The map used for localization is generated from the parameters that describe each field and are known beforehand. The majority of fields, and the ones considered in this thesis, are rectangular shaped fields in which plants are planted at a fix distance between each other and the distance between the rows is always the same. Thus, a simple map comprised of the position of the planted trees can be created knowing these parameters: distance between plants, distance between rows, number of rows and length of field/number of plants per row.

Additional information is required for the localization process and it is assumed to be known. This is the start position of the robot in the field. For the localization process it is necessary to know its initial conditions, i.e. the initial position of the robot. Given that these fields are symmetrical, there are only two possible start positions for the robot: bottom left or bottom right of the field (starting from the top right would give the same result as starting from bottom left and starting from the top left would give the same result as starting from bottom right). The localization module details are further explained in Section 3.4.

3.2.3 Navigation Module

The navigation module receives the coordinates of two lines in the frame of the robot. These lines will be used to navigate between the plant rows using a controller that maintains the trajectory of the robot on the middle of the field rows. If the localization module is in use, when no lines are provided to the navigation module, the robot can navigate based on the field map and the robot's estimated pose in the field provided by the localization module. A more detailed explanation of this module is presented in Section 3.5.

3.2.4 Simulation Modules

These modules allow the developed algorithms to be tested safely before deploying them with the robot in a real garden. The simulated environments consist of gardens resembling real tree nurseries and a model of the robot to drive through them. They can be obtained either by scanning real gardens using a Leica scanner and creating an octree ([Mea82]) from the resulting point cloud or by generating them using different plant models. The sensor data also needs to be simulated.

Gazebo Module

For simulating the robot a 3D model of the robot is created. This will be driven inside the simulated fields. Robot dynamics and sensor information need to also be simulated.

The robot is comprised of a mesh provided by the same company that created the robot platform, depicted in Figure 3.1. For controlling the dynamics, a skid differential driver plugin offered by ROS is used. This plugin allows a control comprised of angular and linear velocities to move the robot to a desired location.

Simulated gardens are created as point clouds and saved in memory as an octree. Meshes of trees in different growth stages are arranged in rows and weeds are placed randomly in the field. These plants are transformed into point clouds from meshes and placed in the gardens.

This module is also responsible for simulating the point cloud information given by the FX10 sensor. For this purpose, a plugin has been created in order to match the sensors specifications. A ray casting technique is used to find point information in the garden octrees. The rays are sent from the center of the sensor into the field's point cloud. The number of cast rays is given by the FX10's resolution. Each ray cast generates one point in the window corresponding to the LiDAR's perception. The rays return intensity and 3D information of the points hit in the octree. Each such point belongs to the surface of the ground or to an object in front of the sensor. In order to realistically simulate this information, a small noise is added to the returned 3D information. This noise increases exponentially with the distance between the sensed point and the sensor's origin.

Odometry Module

A noisy odometry is necessary for a correct simulation in Gazebo. This is done by adding noise to the motion model of the robot. For a robot that is moving in 2D space, the pose is described by the Cartesian coordinates x and y, and the angle of the robot's facing direction with respect to the world frame θ. Thus the robots pose can be written as $\phi = (x, y, \theta)$.

The robot is considered to be moving as described by Equation (3.2), taking inspiration from the works of [EP04], with $\phi' = (x', y', \theta')$ being the robot's pose one time step in the future. D is the actual distance traveled by the robot, T is the actual turn, and C is the lateral translation performed in that time step. The term C is present to model shift in the orthogonal direction to the major axis (the robots driving direction), i.e. when the robot is slipping laterally.

$$\begin{aligned} x' &= x + D\cos(\theta + \frac{T}{2}) + C\cos(\theta + \frac{T+\pi}{2}), \\ y' &= y + D\cos(\theta + \frac{T}{2}) + C\cos(\theta + \frac{T+\pi}{2}), \\ \theta' &= \theta + T \mod 2\pi. \end{aligned} \tag{3.2}$$

To take into account inaccurate measurements, the noise model from Equation (3.3) is used, assuming a normal distribution of the terms C, D and T.

$$\begin{aligned} C &\sim \mathcal{N}(\mu_c, \sigma_c^2), \\ D &\sim \mathcal{N}(\mu_d, \sigma_d^2), \\ T &\sim \mathcal{N}(\mu_t, \sigma_t^2). \end{aligned} \tag{3.3}$$

Tools Module

This software module is comprised of tools useful for testing the implemented algorithms in a simulated environment. The field generation tool is used to generate simulated gardens with different parameters e.g. field size, number of plant rows, probability of gaps in the rows, plant size and type. The field observation tool computes the metrics necessary to evaluate a robot's run through the field.

Field Generator A tool for generating diverse plant fields uses different parameters in order to create gardens like those which the real robot might encounter. Each field can differ from another by number of plant rows, number of plants per row, distance between rows and distance between individual plants. The type of plant and also its dimensions can be changed. Because in real gardens trees are sometimes dug out to be planted elsewhere or to be sold, a parameter for missing plants can be set and a hole in the soil is added in its place. Weeds of different types and sizes can also be added to the field. They are placed randomly and may vary in number.

Some of the files generated for simulating a garden will also be used for navigating the real fields, including the map of the field. An example of a simulated field can be seen in Figure 3.4. The field has small trees arranged in rows. Furthermore, the gaps representing missing trees are also depicted by the same figure. The weeds are spread uniformly in random places inside the field.

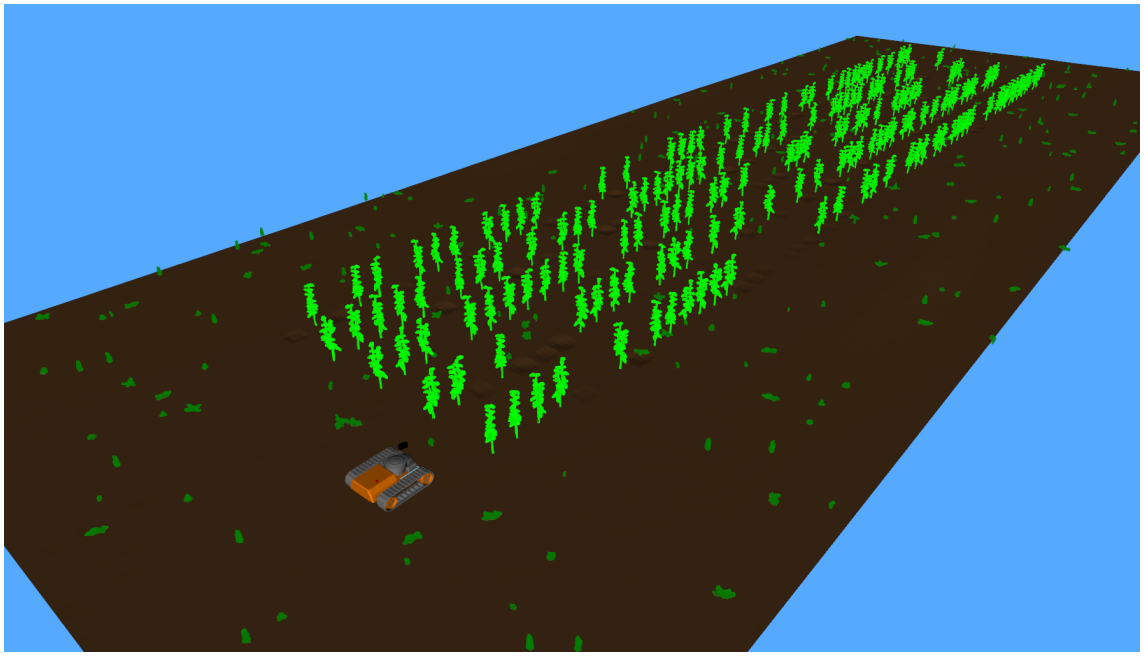


Figure 3.4: AMU-Bot inside a simulated field. The figure shows a simulated field with plants arranged in rows. The gaps between plants are distributed randomly, as well as the weeds.

Field Observer This tool starts the simulation and offers an overview of the algorithms performance. It also tracks the robot's movement and how it behaves within the field. Using the information from the robot's behavior, several metrics were developed to compare the row detection methods between each other. For this purpose, the module uses the ground truth location of the robot to keep track of its movements and the interaction with the plants and plant rows. When it detects that the run needs to stop, either because the run is completed or because it encountered an interaction that is set to stop the run, the field observer reports the metrics it has kept track up until that point.

In order to calculate the metrics, the robot's frame is reduced to a bounding box and plants are approximated by circles. The metrics are computed based on the interaction between the robot's bounding box, the plants' circles and the lines obtained by connecting the centers of the plants.

3.3 Row Detection Methods

Because most tree nurseries and gardens have plants arranged in rows, the main problem in navigating such environments is to detect the rows and to navigate between them. Thus, this thesis focuses mostly on perceptions algorithms that aim to detect lines, corresponding to the plants rows, from the information received from visual sensor, in our case a 3D LiDAR. After processing the input point cloud from the sensor, filtering it, and removing the points belonging to the ground, the algorithm starts to search for plant rows in the remaining point cloud with different approaches. Starting from a simple approach and adding upon it, several line detection methods are developed.

3.3.1 PCL Lines

PCL functions are used to extract the line with most points from the obstacle point cloud - the scanned point cloud minus the ground points - using the RANSAC method. It is assumed that this first cloud is the row of plants "seen" best by the sensor. The inliers are eliminated and the outliers are assumed to be the points belonging to the second plant row. Using the same PCL method to extract the first line, the second line, belonging to the other plant row, is also extracted. RANSAC requires two parameters to be set i.e. the maximum number of iterations run and the maximum distance a point can be from the detected line such that it will be considered to be in the inliers set.

This method fails when the LiDAR senses a small part of one row. The information about this plant row is not enough to create a line representing correctly this plant row. Erroneous detection of rows in such cases make the robot drive on a wrong path. Figure 3.5 shows the result of this method. The white points belong to one detected line and the colored ones belong to the other. The figure shows that there is not a clear separation between the points belonging to the left and the right detected line. This problem may result in detecting lines that do not coincide with the plant rows.

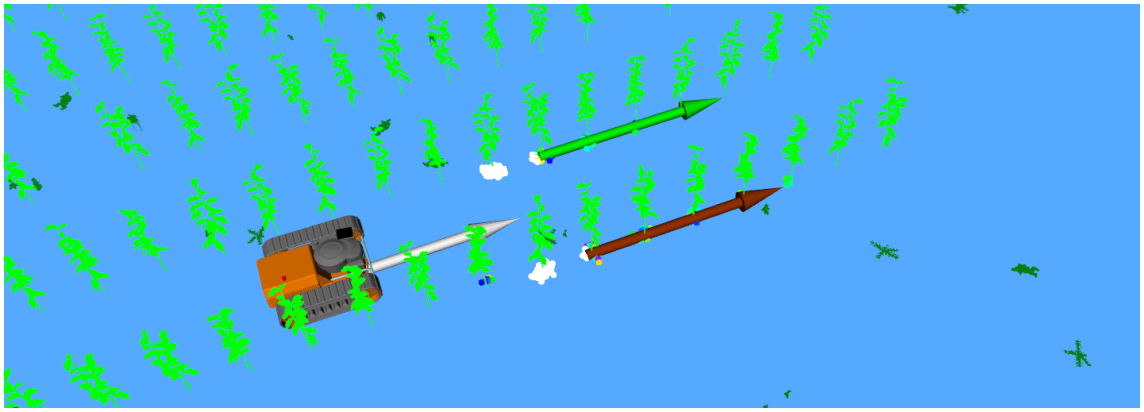


Figure 3.5: PCL Lines method. The green and red arrows represent the results of the line detection algorithm. The white arrow is the middle line computed such that it bisects the red and green arrows.

3.3.2 Parallel Lines RANSAC

This method uses a RANSAC-based algorithm that detects two parallel lines situated at distance D one from the other, developed within this thesis. This implementation eliminates the downside of the previous method by requiring less information about one of the lines if the other one is well defined in the point cloud. For this RANSAC approach, two points from the data set are chosen, a line is created through the selected points and another line, parallel to the first one, placed at a distance equal to the distance between the rows is also created. These two lines should resemble the two rows of plants the robot should see when navigating the field, inside the rows. This is the model that is then fit to the entire data set. After a predefined number of iterations, the algorithm returns the two lines that best fit our model i.e. two parallel lines placed at a distance equal to the distance between the rows from each other and the inliers, i.e. the points belonging to these two lines. The RANSAC algorithm needs the same two parameters to be set as for the *PCL Lines* method i.e. maximum iterations and distance threshold.

Compared with the previous method, the *Parallel Lines RANSAC* outputs a clear separation between the point belonging to the left row and left row of plant, as can be seen in Figure 3.6. The shortcomings of this method appear when too little information is available for both lines, making the fitting of two parallel lines impossible.

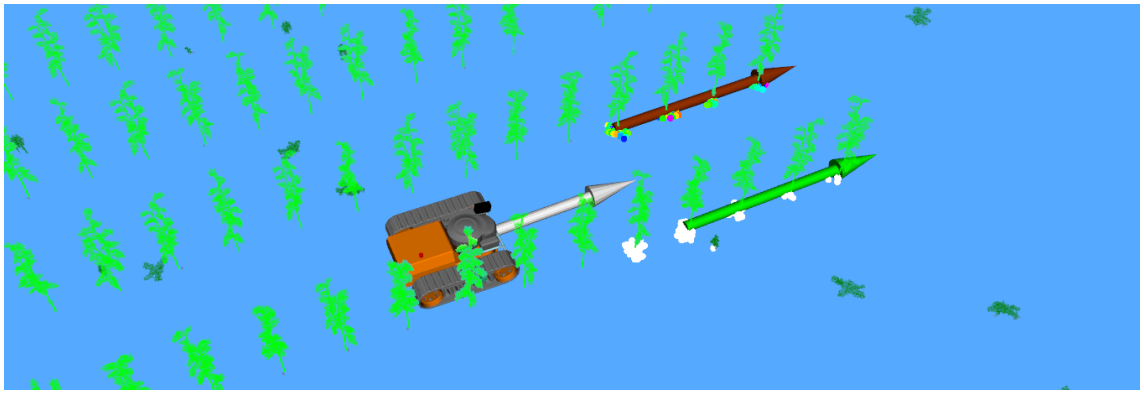


Figure 3.6: Parallel Lines RANSAC method. The greed and red arrows represent the results of the line detection algorithm. The white arrow is the middle line computed such that it bisects the red and green arrows.

3.3.3 Centroid Lines

Both previous methods suffer from the same downfall. The methods are too reactive to change, using information from just one frame of the laser scanner.

The *Centroid Lines* method tackles this problem by keeping information about the plants from one frame to the other. It starts by creating clusters in each scan and computing the centroids of each detected cluster. The provided PCL methods are used to create a kd-tree, a well-known space-partitioning data structure for organizing points in k-dimensional space, from the input point cloud.

An organized structure that speeds up searching inside the set of data points is needed when applying the next PCL method to the point cloud. The following part of the algorithm creates clusters of points. The clusters represent individual plants. The clustering algorithm needs two parameters to be set: the minimum number of points a set can have to be considered a cluster and the maximum distance between two points belonging to the same cluster. Then, these points are stored in a queue, eliminating older centroids based on the queue limit (the third parameter of this method).

Having the history of centroids as an input, the clustering algorithm is deployed again, having a larger maximum distance between the points belonging to the same clusters. Now the clusters differentiate between plants belonging to one of two rows: the row on the left of the robot and the row on the right of the robot respectively. Having two new groups of points, they are separated by their position in relation to the robot: on the right side and on the left side. The points in the queue, now separated into the sides, each belonging to the plant row on one side of the robot or the other, are used to detect two lines. The centroids and the lines created from them can be seen in Figure 3.7.

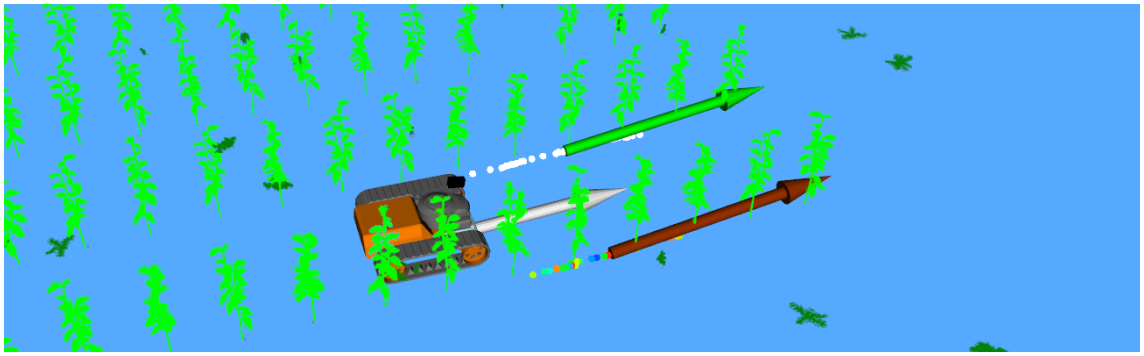


Figure 3.7: Centroid Lines method. The greed and red arrows represent the results of the line detection algorithm. The white arrow is the middle line computed such that it bisects the red and green arrows.

3.3.4 Map Lines

This method extends the *Centroid Lines* method by creating a sliding window around the robot in which the information about the surrounding plants is kept. Each scan received by the method is transformed into the odometry frame. The resulting frame is integrated in a local map of the plants seen by the robot.

When each new scan is added to the map, each point's distance to the robot is checked. The points that have fallen out of the sliding window around the robot are eliminated. Furthermore, the map is down-sampled to just a point per voxel (3D equivalent of a pixel - picture element). The down-sampling is done to avoid redundant information when the same point is added by different scans, burdening the algorithm with greater computational effort.

Using the points found in the local map, clusters are created, setting the parameter $distance\_threshold$ such as each cluster represents a plant. By creating the centroid of each cluster, the center position of each plant should be obtained in the row located in the sliding window around the robot. To differentiate between plants and weeds, the information about the field layout is used. Knowing at what distance the plants should be placed in this field and the distance between the plant rows, a grid of the plants that are expected to be detect can be created. For this purpose, a RANSAC method that uses a grid as model to fit the plant centroids to the grid is developed. Selecting a small distance threshold for the fitting, a set of inlier points is obtained. These points represent the position of plants belonging to the field. The outliers are the centroids of the plants that should not be there i.e. the weeds.

Figure 3.8 shows the local map and the resulting lines obtained by applying the clustering method to this local map.

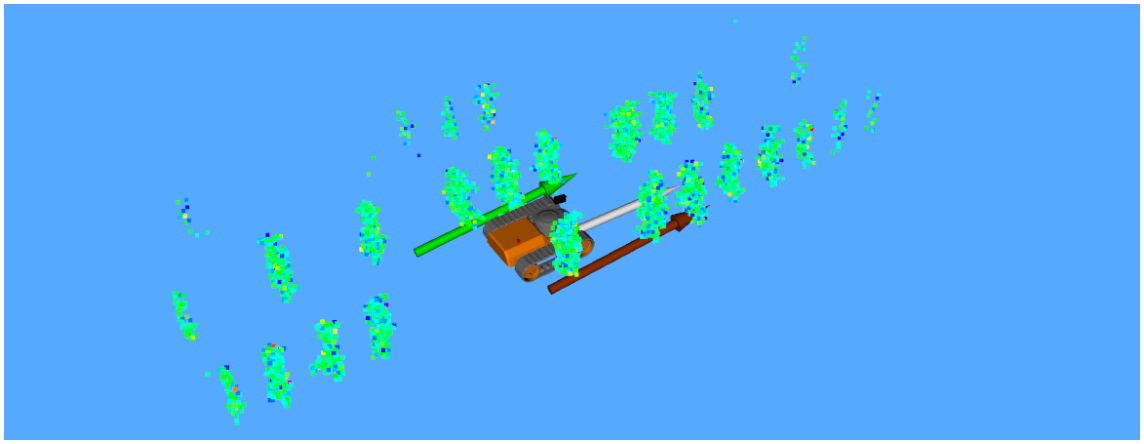


Figure 3.8: Map Lines method. The greed and red arrows represent the results of the line detection algorithm. The white arrow is the middle line computed such that it bisects the red and green arrows.

The local map relies on the robot's movement to keep track of the plants passed by the robot. If the movement is inferred solely using odometry, which is prone to errors, the local map becomes skewed. Creating lines from this deformed map, in which the plants' positions are incorrect, results in inaccurate depictions of the plant rows. This immediately leads to a wrong detection of the plant rows. To overcome this problem, a localization step needs to be present, which ensures that the robot creates an accurate representation of its surroundings.

3.4 Localization

In graph-based localization, each node of the graph represents the robot's position and orientation (x_i in Figure 3.9). The first node of the graph is the robot's starting position in the map. In order to simulate initial uncertainty, a small Gaussian noise is added to the start position in the first node. A new node is added to the graph when the uncertainty of the estimated position grows beyond a certain threshold. In practice we add a new node when the robot has either traveled a certain distance, or when it turned a certain amount of degrees. The edges between the nodes represent the relative poses, meaning how much the robot traveled and how much did it turn between the moments of time the two nodes were added. This information is given by the odometry (the relative movement registered by combining the information from the wheel encoders and the IMU) and encoded in the edges ((2) in Figure 3.9).

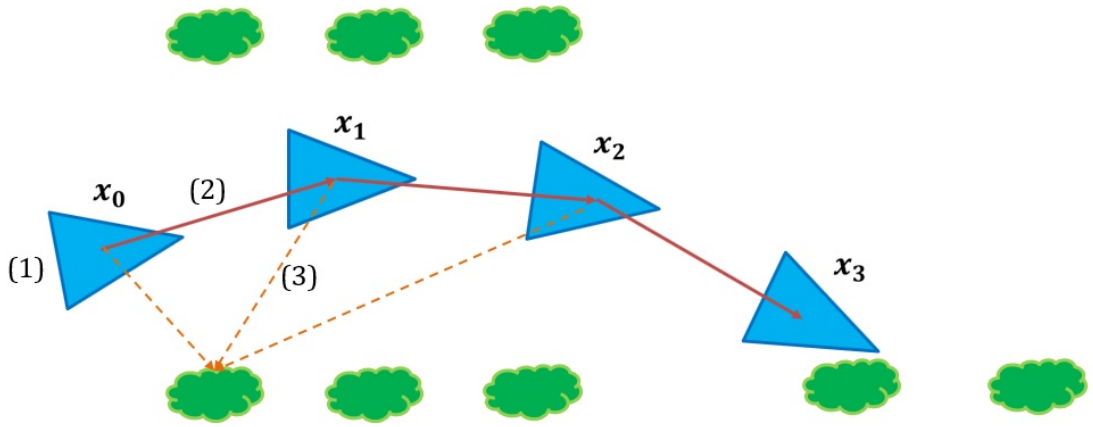


Figure 3.9: Graph-based localization.
(1) - Initial motion constraint,
(2) - Relative motion constraints,
(3) - Relative measurement constraints

Based only on relative motion constraints, the estimated pose of the robot would coincide with its odometry. Because errors accumulate when integrating information from movement sensors, additional sensory data can be added to obtain a better pose estimation. Using LiDAR information, one can add additional relative measurement constraints ((3) in Figure 3.9) to the graph to improve the estimation. This new data acts like a spring to pull the graph in such a way that it fits all the information and offers a better representation of the robot's real movement.

In the current implementation, the points belonging to the lines detected by the algorithms are used as matches for the lines created by the plants in the map. This kind of match allows for the graph to estimate correctly that the robot is driving in the middle, between the rows, because this is enforced by the navigation controllers. This kind of matching corrects the errors caused by slipping wheels that will count more ticks on one wheel than on the other when the robot is driving on a straight line. These errors might be translated as the robot turning at an angle than differs from the ground truth and making the robot believe that it's moving to the right or to the left when in fact it is just moving straight. Because the robot is driving in the middle of the row and perceives this with the LiDAR, this constraint is fed to the graph based optimization, resulting in a position estimation that places the robot in the middle of the row.

Other errors in interpreting the data from the motion sensors can lead to measurements that wrongly describe the distance the robot traveled, resulting in either too short or too long paths. To overcome this, the information about the detected plants is used. This is done by adding relative measurement constraints to the graph each time a plant is correctly detected (using the grid fitting algorithm Section 3.3.4). Because the planting distance is knows from the map, it provides enough information about the distance traveled by the robot to correct erroneous motion sensor data.

Up until this point, the front-end of graph-based localization has been described i.e. how the graph is built. For optimizing the graph and computing a pose estimation that fits all the information provided to the graph, a previously build localization pipeline that uses g2o is utilized. This localization pipeline had to be adapted for using FX10 sensor data because it was created to use Velodyne data.

Velodyne is type of 3D LiDAR that uses 16 laser beams that turn 360 degrees offering a greater amount of information about the environment. The difference in field of view between these two LiDAR sensors can be observed in Figure 3.10. The Velodyne is a common sensor used in localization applications, while the FX10 in a sensor created with the purpose of obstacle detection.

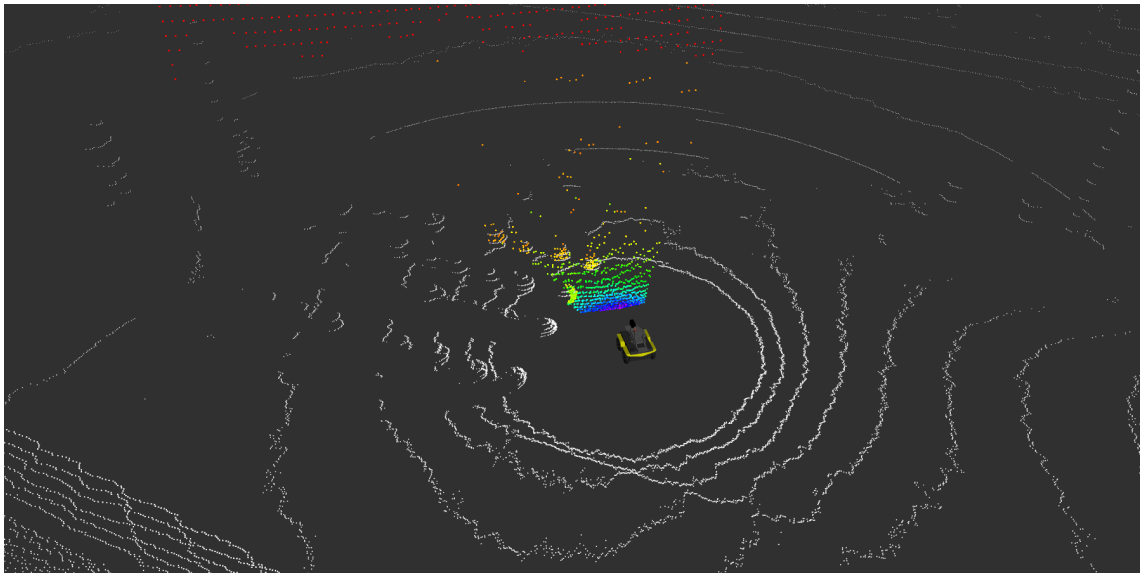


Figure 3.10: LiDAR comparison: FX10 vs Velodyne. The colored points represent the point cloud captured by the FX10 sensor. The white points represent the point cloud captured by the Velodyne sensor.

3.5 Navigation

The navigation module is responsible for driving the robot from the start of the field to its end as accurate as possible. This translates to a smooth drive, keeping equal distance between the rows and not damaging any plants along the way.

The navigation is realized using a state machine that describes the different states in which the robot can exist, seen in Figure 3.11.

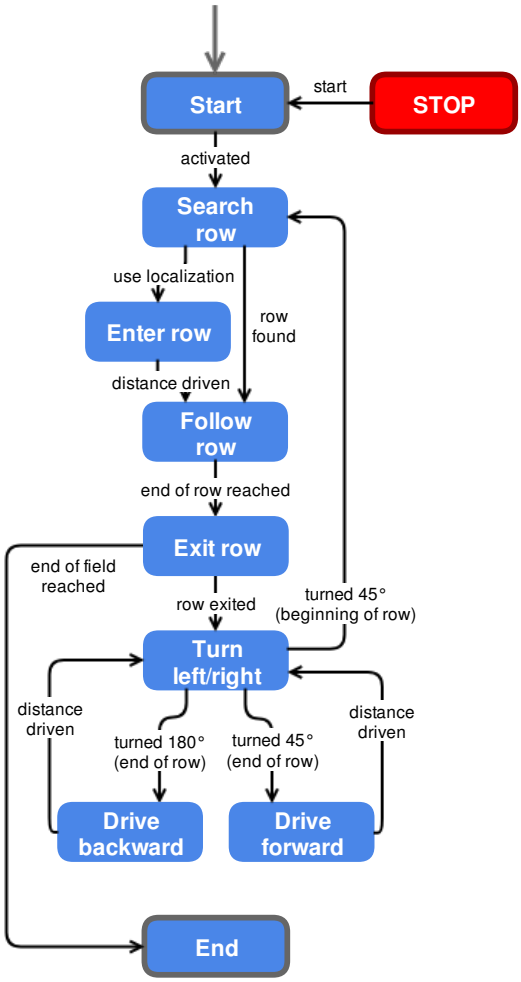


Figure 3.11: State machine of the navigation logic. The figure illustrates the flow between each state and the event that triggers the change of state.

Start Once the robot is turned on, it is in *Start* state. Assuming it is placed at the start of a row, in one corner of the field, it can be activated and directly switch to the *Follow row* state.

Search row If the localization is used, the robot transitions to the *Enter row* state, where it follows the rows for a few meters using the estimated position and the provided map. When navigating using only perception, the robot moves slowly in front until two lines are detected. After driving using localization or detecting two lines (the two plant rows), the robot enters *Follow row* state.

Follow row In this state the robot uses a controller to stay in the middle of the rows detected by perception. To do this, an auxiliary line is created: the Middle line between two 2D lines (Mid Line). When two lines are detected, the Mid Line is the bisecting line between them. When only one line is detected, this Mid Line is created by translating that line half of the distance between the rows towards the robot, parallel to the detected line. If no lines are observed by the perception module,

the robot will stop moving or it can use the localization module to create a Mid Line between the plant rows of the map and navigate based on the estimated position. The end of the row can be signaled by the perception module, when not enough points belonging to the plants are detected, or using the localization module to estimate when the row has ended.

While in *Follow row* state, if the localization module is used, lines provided by the perception are checked. Knowing the estimate location of the robot in the map, lines that do not match the expectations are discarded. It is checked if the lines given by the perception module are at a reasonable distance and form a reasonable angle from the map rows.

Exit row When the robot has finished one row, it drives a certain distance to be completely out of the row. In case it was the last row, the robot has finished its job and enters *End* state. If not, it transitions into *Turn* state.

Turn Based on the previous state, the robot can turn in three different ways. If it just exited a row, the robot turn 180 degrees to face the field again and switches to *Drive backward*, as shown in Figure 3.12. It does that to have the most amount of information for localization while driving blindly. If it just drove backward, the robot turns 45 degrees in the direction of the next row, then transitions to *Drive forward*. When it finishes driving forward and should be at the start of the new row, the robot turn 45 degrees to be aligned with the new row. After the third type of turning, the state machine transitions into *Search backward* and start from the beginning.

Drive backward After the 180 degrees turn, the robot starts its blind drive into the next row, creating an imaginary isosceles right triangle, that can be seen in Figure 3.12. It drives backward a distance equal to the distance between the plant rows, creating the first side of the imaginary triangle.

Drive forward After the first 45 degrees turn, the robot drives forward a distance equal to the square root of two multiplied with the distance between the rows ($row\_distance\sqrt{2}$), creating the hypotenuse.

End At the end of the field, this state is reached. Here the robot is awaiting to be stopped or to be given further instructions.

Stop This state is reached when the robot encounters any unpredictable event or when it is stopped remotely by the user. This is an emergency state created to prevent any unwanted actions. From this state the robot cannot return to its previous state, it can only transition to the *Start* state.

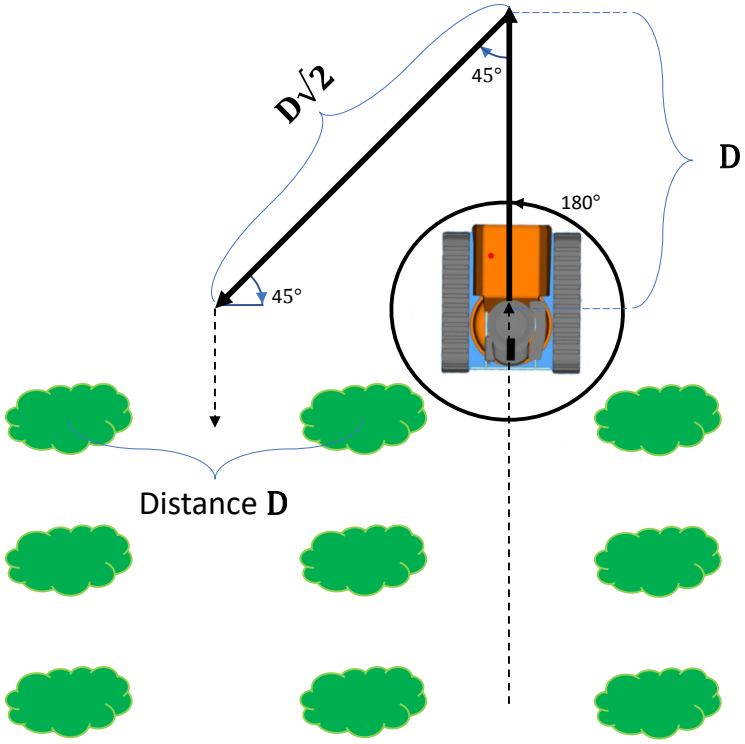


Figure 3.12: Blind turning. The distance between the plants rows is denoted by D. The blind turn maneuver consists of 5 actions: turning 180 deg at the end of one row, driving backwards distance D, turning 45 deg in the direction of the next row, driving forward distance $D\sqrt{2}$ and finally turning 45 deg to face the new row.

3.5.1 Control

When the robot is in the *Enter row* or *Follow row* states, it needs a line to guide it. This line is created as the Mid Line between the rows detected by perception or as the Mid Line between two plant rows of the map. The bisecting line is created from the lines' equations in the form of $ax + by + c = 0$.

Because the detected lines are provided to the navigation module as pose messages, each line coefficients are created with two points on the X-axis of each pose, $A(x_A, y_A)$ and $B(x_B, y_B)$, using:

$$\begin{aligned} a &= y_B - y_A \\ b &= x_A - x_B \\ c &= x_B y_A - x_A y_B \end{aligned} \tag{3.4}$$

Having the equations of the two lines, auxiliary value Equation (3.5) is computed:

$$\phi = \pm\sqrt{\frac{a_1^2 + b_1^2}{a_2^2 + b_2^2}}. \tag{3.5}$$

With ϕ calculated, the equation of the bisecting line can be computed.

$$\begin{aligned} a_m &= a_1 - \phi a_2 \\ b_m &= b_1 - \phi b_2 \\ c_m &= c_1 - \phi c_2 \end{aligned} \tag{3.6}$$

Using this equation, the Mid Line is computed in such a way that it is always placed in front of the robot, facing its desired driving direction. This is done by setting the x value to 0 for the one point, 1 for the second point and computing the associated y values. Having two points computed using Equation (3.6), a ROS pose message can be created and used by the controllers.

PID Controller

Using PID control to keep the robot on the middle of the plant rows requires two regulators: one PID for keeping a reference of 0 meter distance between the Mid Line and the robot's center (base footprint). The second PID is used for keeping a reference of 0 degrees angle between the Mid Line and the driving direction of the robot (the X axis of the robot's frame).

Input/Output Linearization Controller

This controller needs a point belonging to a desired trajectory for the robot to try and reach. At each time step, a point on the Mid Line is given. The Mid Line is constructed to always be in front of the robot. This allows for a continuous trajectory to be followed by the robot. This trajectory is either the bisecting line between the detected lines, or the middle of the plant rows from the map.

4 Evaluation

The aim of this project is to robustly navigate through various types of fields. These environments distinguish themselves by being populated with a diversity of plants with different sizes and shapes. The implementation is tested on 100 different generated fields, resembling a variety of tree nurseries. These environments are used to assess the performance of the developed algorithms. In assessing the performance of each developed method, a series of metrics are computed and averaged over the 100 fields. After testing that the proposed methods work on the simulated field, the algorithms are deployed on a real robot that drives in a real garden.

4.1 Testing Scenario

The demonstration and testing scenario is chosen to cover a wide range of plants in tree nurseries. For this purpose, a tree (the Beech shown in Figure 4.1) and a shrub (the Cherry laurel shown in Figure 4.2) are selected. Both types of plants display different stages of growth. The shrub in the first year is exemplary for almost all plants in this growth stage. The selected shrub shows a similar growth behavior as a multitude of other shrubs. The tree from the 2nd year stands as an example for all trees with a clear stem and, if possible, no near-bottom wild drives. The Beech and Cherry laurel are delicate trees and shrubs, which are very different in size from weeds and do not have a solid stem. In the case of the Beech, starting from the second year of growth, bamboo rods are used to ensure a straight growth of the stem.

A typical field for beeches or cherry laurels has a row distance of 1.3 meters and a planting distance of 0.4 meters. After a few years the trees get so big that they grow together. Then, the trees are either sold or replanted into a new field. Bigger trees like avenue trees need a planting distance of up to 1.8 meters, determined by the size of the tree crown. An overview of the arrangement that these plants have in the fields and of the sizes that these plants can reach in each type of field can be seen in Table 4.1.

	Row distance [m]	Planting distance [m]	Maximum height [m]
Beech			
First field (up to 4-5 years)	1.30	0.40	2.00
Second field (from 5 years)	1.50 - 1.60	1.20 - 1.60	3.00 - 4.00
Cherry laurel			
First field (up to 3 years)	1.30	0.40	1.25
Second field (from 3 years)	2.00	2.00	2.00 - 3.00

Table 4.1: Row distance, planting distance and maximum height of Beech and Cherry laurel in the two types of fields.

(a) Second year **(b)** Second year

(c) Third year **(d)** Seventh year

Figure 4.1: First plant culture: Beech in the 2nd year, in the 4th year and in the 7th / 8th year. Plant grows as a tree with a single clearly identifiable stem. In addition, in the second year, bamboo rods are used to ensure straight growth.

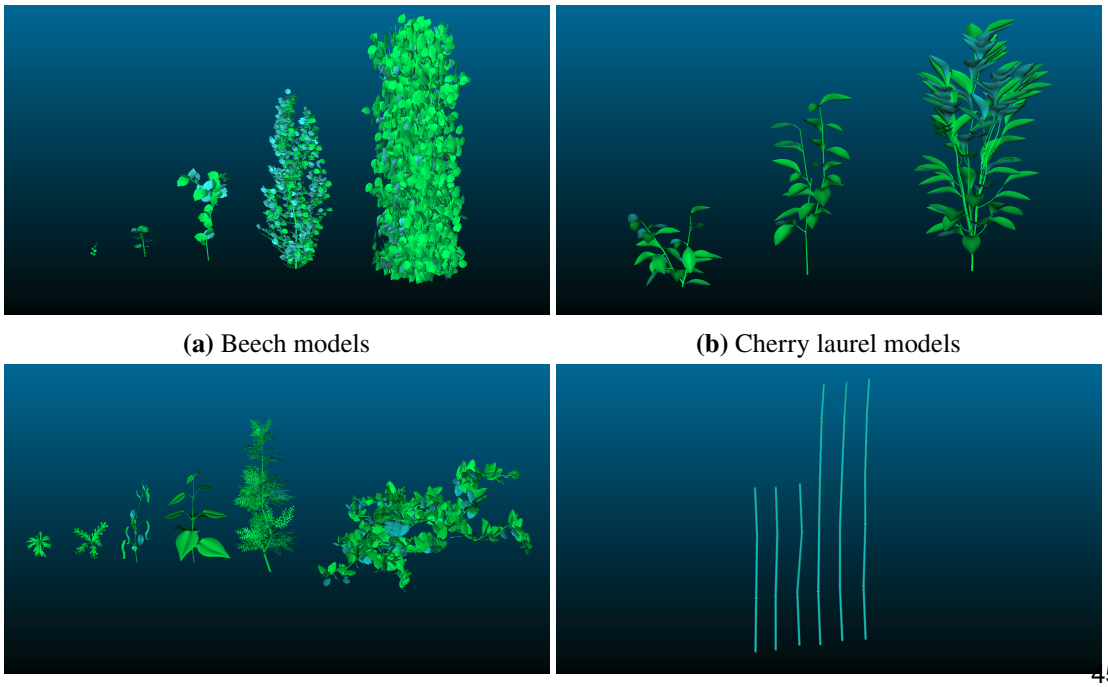
4.1.1 Simulation

For simulation, the Gazebo ROS environment is used to test the developed methods in 100 generated gardens. The fields cover a diverse configuration which resembles the different growth stages of the Beech and Cherry laurel chosen for testing. Weeds are also added to the field to emulate real gardens. The simulated models can be seen in Figure 4.3.

(a) First year **(b)** First year

(c) First year **(d)** Third/4th year

Figure 4.2: Second plant culture: Cherry laurel in the first year and in the 3rd / 4th year. Plant grows as a shrub, without a single clearly identifiable strain. There are a variety of wild shoots also just over the ground.



(a) Beech models **(b)** Cherry laurel models

(c) Weed models **(d)** Bamboo sticks

Figure 4.3: Database of 3D plant models.

The 100 simulated fields are generated to resemble real gardens, but also include extreme scenarios that offer a better understanding of the performance and limits of the implemented algorithms. The field specifications used in generating these fields are presented in Table 4.2. The values in the table represent the mean. For each generated plant, a Gaussian distribution is assumed. The actual values differ from the mean with a standard deviation. The standard deviation for the plants' height and radius is computed by randomly choosing a value between 0.05 and 0.3 and multiplying it with the mean i.e. $stddev = s * plant\_height$, with $s = rand(0.05, 0.3)$. The position of each plant can also differ with a standard deviation between 0.01 and 0.08 meters.

	Row distance [m]	Planting distance [m]	Plant height [m]	Plant radius [m]
Beech	1.3, 1.5 and 1.6	0.4, 1.2, 1.4, 1.5 and 1.6	between 0.1 and 4	between 0.04 and 0.81
Cherry laurel	1.3 and 2	0.4 and 2	between 0.1 and 3	between 0.06 and 0.76

Table 4.2: Parameters used for generating the 100 fields. Row distances and plant distances have fixed values. The plants dimensions: height and radius increase incrementally from the minimum value to the maximum value.

For every field, the number of rows is picked at random between 5 and 11. The number of plants per row are picked at random between 20 and 40. Each field has a probability of having between 0.01 and 0.25 percent of plants missing, chosen at random.

The robot is set up to run through the fields in order to evaluate the performance of the developed algorithms. The field is traversed one row after the other in an attempt to cover the whole field and be as gentile as possible with the plants while doing so. The test stops when the robot completes driving through all plant rows successfully or when the simulation is aborted. Specifically, the tests end when the robot does one of the following unauthorized actions:

- WRONG ROW the robot enters a different row from the natural flow
- CROSSED ROW the robot crosses into the neighboring rows while driving in a specific row
- SAME ROW the robot enters a previously visited row
- TOO FAR the robot drives too far away from the field; defined as being farther than distance D to the nearest plant of the field (for the tests, a distance of $D = 4.0$ meters is set)
- STUCK the robot doesn't move anymore; defined as not driving more than distance d in t seconds (for the tests, $d = 10$ centimeters and $t = 30$ seconds are used)

In order to differentiate between the overall performance of the different methods developed, during the run, the behavior of the robot is monitored. Several metrics are computed in order to evaluate individual method performances over the whole batch of 100 fields. Each metric is computed from the start of the simulation up until the test stops. The metrics computed in the simulations are:

- *covered_rows* the percentage of rows covered, computed as fraction from the total number of rows in that field;
- *trajectory_error* the distance between the actual trajectory of the robot and the desired one;
- *damaged_plants* the percent of damaged plants, computed as fraction from the number of plants passed by the robot;

• *computation_time* the time elapsed for each method in order to find the rows in each frame, measured in milliseconds.

covered_rows The goal of this project is to create a robot that can drive autonomously through a diversity of gardens of different shapes and sizes. This means that the algorithms should assist the robot in driving from start to end without any human intervention. The goal here is to obtain full coverage of any encountered type of fields. The *covered_rows* metric indicates how far towards this goal the robot drives. The metric is reported in percentage.

damaged_plants Driving in an environment populated by small, delicate trees, the algorithm must be able to navigate without causing any harm to the plants. Achieving this goal can be seen by decreasing the *damaged_plants* to zero. A plant is considered damaged when the robot drives over half of the plant's radius. An illustration of this action can be seen in Figure 4.4.

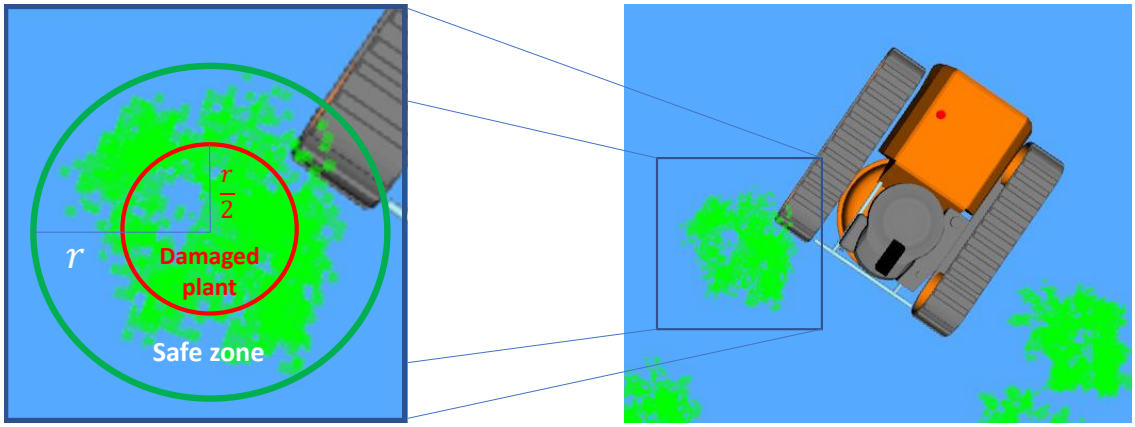


Figure 4.4: Damaged plant metric. If the robot reaches beyond the safe zone, into the red circle of radius equal to half of the whole plant, the plant is considered damaged.

trajectory_error Achieving a smooth run through the fields with the least amount of casualties translates into the robot following a certain perfect trajectory. The robot should drive exactly in the middle between two plant rows. The difference in distance measured between our ideal trajectory and robots actual path is named *trajectory_error*. This error is computed at a frequency of 10 Hz, only when the robot is inside the plant rows. The blind turning is not subjected to trajectory error computations. We characterize this metric by: maximum value, 99 percent error, mean value and standard deviation. The 99 percent error is the maximum value among the first 99 percent of values. This is a metric that shows the maximum of most values, excluding the maximum 1 percent from the data set, usually consisting of outliers.

computation_time In order to ensure a consistent frequency at which the robot can operate, the time it takes for each method to find the rows in each frame needs to be under a certain threshold. For this purpose each line detection method is timed in milliseconds to check if it fits the criteria.

At the end of a simulation, the *Field observer* - described in Section 3.2.4 - outputs an image showing the trajectory of the robot and the damaged plants. Figure 4.5 shows one such image output where the robot drives through all the plant rows, but damages some plants - marked with red circles - along the way. Based on this trajectory information, the metric *trajectory_error* can be computed.

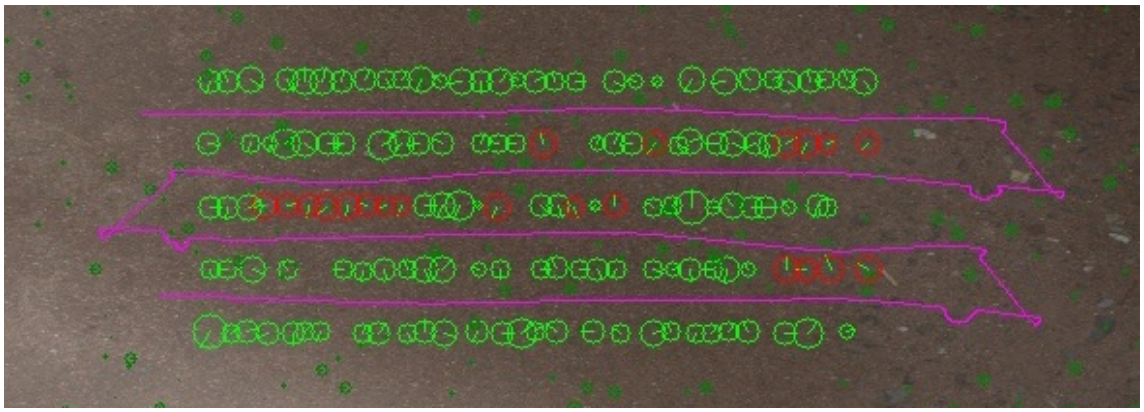


Figure 4.5: Robot trajectory and damaged plants. The pink line draws the trajectory of the robot inside the field. The green circles represent plants. The red circles represent damaged plants.

4.1.2 Test Garden

To test the approach proposed in this thesis on a real robot, a small artificial garden was created within the Bosch Research Campus in Renningen using plastic plants. The testing garden can be observed in Figure 4.6.

Figure 4.6: Test garden in Bosch Research Campus, Renningen

In order to run the algorithm in real life, a simulated field must be created. The simulated fields is generated using the parameters of the real field. When generating the field, the *amubot_tools* module, described in Section 3.2.4, also generates all the files needed to run the robot on a real

field. These files include the plant map and the launch file with parameters specific to that field. The plants are arranged in 4 rows, 9 plants per each row. The plants were arranged at a distance of 0.7 meters between each other in every row and a space of 1 meters was left between each plant row. Gaps were left in some rows to mimic the real life scenario when a tree is dug out and sold, or moved to another field.

Because the AMU-Bot has not been built yet, a small robot platform, named Jackal, was used to deploy the final algorithm.

Jackal Jackal (Figure 4.7) is a small, fast, entry-level field robotics research platform. It has an onboard computer, GPS and IMU fully integrated with ROS for out-of-the-box autonomous capability. As with all Clearpath robots, Jackal is plug-and-play compatible with numerous robot accessories.

Figure 4.7: Jackal equipped with FX10 LiDAR
https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

4.2 Parameters Selection

Most methods have different parameters that impact their performance to a certain degree. Testing one method with one set of parameters spans between 2.5 and 6 hours, depending of the overall performance. To reduce the time needed to test each individual method with several sets of parameters, a small subset of the gardens is selected. From the 100 fields, only 10 were selected at random. Every methods was tested on this subset of fields. Several experiments were performed applying different values for the parameters. After performing the experiments, the optimal values for the parameters are obtained. This is done through a study of different combination of parameters that delivers the best overall performance for each method. Using these optimal parameters, the methods were compared among each another.

For the *PCL Lines* method, presented in Section 3.3.1, the parameters with the most impact are the distance threshold (dis_threshold measured in meters) for a point to be classified as belonging to the line and the maximum number of iterations (max_iteration) that the RANSAC algorithm could perform. The results presented in Table 4.3 show that values of 30 centimeters for the dist_threshold and between 10 and 20 iterations for the RANSAC algorithm yield the best results for this method.

dist_threshold [m]	0.3	0.3	0.3	0.1	0.1	0.1	0.5	0.5	0.5
max_iteration	10	20	40	20	40	10	10	40	20
covered rows [%]	28.7	27.5	26.3	23.5	21.8	20.4	17.9	17.1	16.5

dist_threshold [m]	0.3	0.3	0.3	0.1	0.1	0.1	0.5	0.5	0.5
max_iteration	20	40	10	10	20	40	20	40	10
damaged plants [%]	24.2	25.1	25.1	29.1	30.3	30.8	33.4	35.4	36.3

Table 4.3: *PCL Lines* parameters. The table shows the percent of *covered rows* and *damaged plants* for different sets of parameters. The best values for the metrics are highlighted in red. The parameters values are color coded from lower to higher intensity of color representing lower and higher values of the parameters.

The *Parallel Lines RANSAC* method, discussed in Section 3.3.2, being closely related to the previous method, both relying on iterations of the RANSAC algorithm to match one or two lines given a point cloud, shows best results also for a value of 30 centimeters for the dist_threshold and around 20 iterations for the max_iteration parameter (Table 4.4).

dist_threshold [m]	0.3	0.5	0.5	0.3	0.3	0.5	0.1	0.1	0.1
max_iteration	20	40	20	40	10	10	40	10	20
covered rows [%]	46.6	44.7	44.6	43.6	42.1	41.4	35.6	31.1	30.7

dist_threshold [m]	0.3	0.5	0.5	0.3	0.3	0.5	0.1	0.1	0.1
max_iteration	40	40	20	20	10	10	40	20	10
damaged plants [%]	30.6	30.6	31.1	31.9	32.3	33.6	34.8	37.0	37.7

Table 4.4: *Parallel Lines RANSAC* parameters. The table shows the percent of *covered rows* and *damaged plants* for different sets of parameters. The best values for the metrics are highlighted in red. The parameters values are color coded from lower to higher intensity of color representing lower and higher values of the parameters.

Centroid Lines, described in Section 3.3.3, have shown to be more affected by other parameters. *cluster_tolerance* is the maximum distance two points can be from each other to still be classified in the same cluster - measured in meters, which has a minimum of *min_cluster_size* points. After grouping the LiDAR scan data, the centroids from the clusters are kept in a queue with a maximum size of *cluser_limit*. These three parameters influence the performance of the methods as seen in Table 4.6 and Table 4.5. From these tables we can observe a better overall performance, both in percent of covered rows and damaged plants, for a smaller cluster tolerance of 0.1 meters. A smaller number of points in the cluster seem to also influence positively the detection of rows. The best

result is obtained, also for both metrics, for a value of 10 points. The maximum queue size looks to have different results for the two metrics, but as a general tendency, a smaller queue offers better performance.

cluster_limit	20	40	40	20	20	20	40	40	40	80	80	20	40	80
cluster_tolerance [m]	0.1	0.4	0.2	0.3	0.3	0.2	0.4	0.2	0.1	0.2	0.4	0.2	0.2	0.4
min_cluster_size	10	40	40	10	40	10	20	10	10	20	20	40	40	40
covered rows [%]	73.5	70.7	69.9	69.9	69.4	69.1	69.0	68.9	67.9	65.2	64.4	64.4	61.3	59.6

cluster_limit	40	80	20	40	40	40	40	80	80	80	80	40	80
cluster_tolerance [m]	0.1	0.2	0.1	0.6	0.6	0.4	0.2	0.6	0.2	0.6	0.4	0.6	0.6
min_cluster_size	40	40	40	20	40	80	80	20	80	40	80	80	80
covered rows [%]	59.5	58.4	56.8	54.6	53.2	51.6	51.3	45.9	44.5	44.2	43.5	39.4	29.3

Table 4.5: *Centroid Lines*: Parameters for *covered rows*. The table shows the percent of *covered rows* for different sets of parameters. The best value for the metric is highlighted in red. The parameters values are color coded from lower to higher intensity of color representing lower and higher values of the parameters.

cluster_limit	40	40	20	20	20	20	40	40	20	40	40	20	80	40
cluster_tolerance [m]	0.1	0.2	0.3	0.3	0.2	0.1	0.2	0.4	0.2	0.4	0.2	0.1	0.2	0.1
min_cluster_size	10	40	40	10	10	10	10	20	40	40	40	40	20	40
damaged plants [%]	28.2	28.7	29.3	29.5	29.9	29.9	30.1	30.4	31.0	32.3	32.8	32.9	33.9	34.6

cluster_limit	40	80	40	40	80	80	80	40	80	40	80	80	80
cluster_tolerance [m]	0.6	0.4	0.4	0.6	0.4	0.6	0.2	0.2	0.4	0.6	0.6	0.6	0.2
min_cluster_size	20	20	80	40	80	20	40	80	40	80	80	40	80
damaged plants [%]	34.7	36.1	37.6	38.1	38.1	39.1	39.4	39.9	40.2	40.8	41.1	41.1	43.3

Table 4.6: *Centroid Lines*: Parameters for *damaged plants*. The table shows the percent of *damaged plants* for different sets of parameters. The best value for the metric is highlighted in red. The parameters values are color coded from lower to higher intensity of color representing lower and higher values of the parameters.

The *Map lines* method, introduced in Section 3.3.4, extends the *Cluster Lines* method. Therefore, the same parameters influences both methods, with the exception of one parameter. *cluster_limit* is not present in the Map lines method because the map is not limited in terms of number of points, but by spatial constraints. Looking at Table 4.7, it can be observed that a smaller number of points needed to define a cluster and a small distance between points has better results that greater values for both parameters.

cluster_tolerance [m]	0.06	0.06	0.06	0.1	0.06	0.1	0.1	0.2	0.06	0.2	0.2
min_cluster_size	15	10	20	20	30	30	40	20	40	30	40
covered rows [%]	86.3	86.1	85.7	84.2	82.3	77.5	73.4	72.7	70.4	65.8	58.3

cluster_tolerance [m]	0.1	0.06	0.06	0.1	0.06	0.06	0.1	0.06	0.2	0.2	0.2
min_cluster_size	20	15	10	40	20	30	30	40	30	40	20
damaged plants [%]	26.6	27.3	27.5	27.85	28.2	29.1	29.8	29.8	29.9	31.1	31.4

Table 4.7: *Map Lines* parameters. The table shows the percent of *covered rows* and *damaged plants* for different sets of parameters. The best values for the metrics are highlighted in red. The parameters values are color coded from lower to higher intensity of color representing lower and higher values of the parameters.

4.3 Methods Comparison

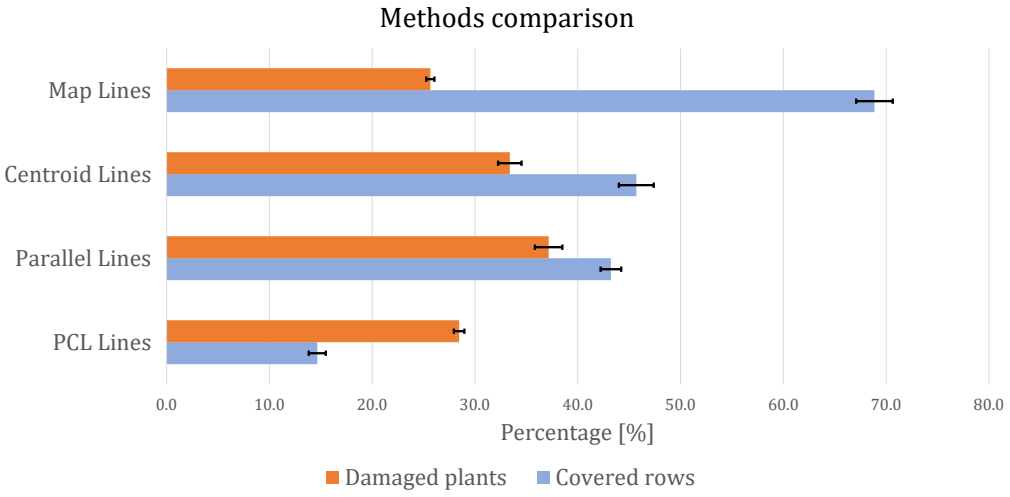


Figure 4.8: Metric values for perception algorithms. *Covered rows* and *damaged plants* metrics obtained by testing on the 100 simulated fields.

Having found the optimal set of parameters for each method, the algorithms can be tested on the 100 simulated fields. Using the field observer, presented in Section 3.2.4, the values for the three metrics – *covered_rows*, *damaged_plants* and *navigation_error* – were recorded. Each metric represents the mean over the 100 fields. The metrics are averaged once more, over 5 runs of the 100 fields coming to a total of 500 fields for each method. Figure 4.8 shows the evolution in performance achieved by each method. Starting with the simple *PCL Lines*, each newly developed method brings an increase in performance over the previous ones. The Figure 4.9 shows the same trend for the accuracy of driving through the simulated fields.

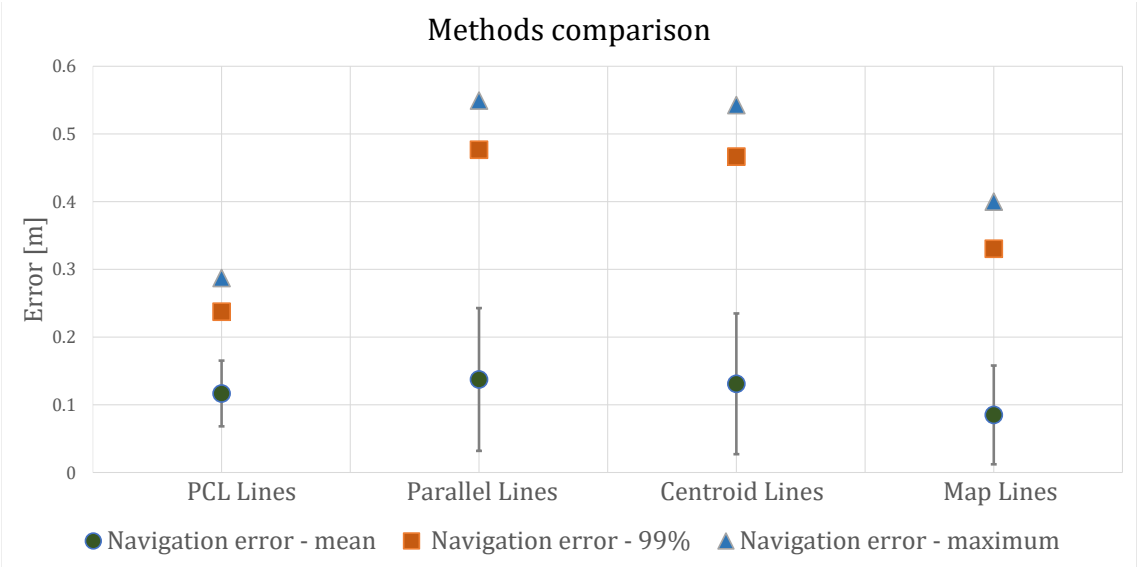


Figure 4.9: Metric values for perception algorithms. Navigation error characteristics obtained by testing on the 100 simulated fields

PCL Lines This first implemented method suffers from many flaws. Relying on a single frame to detect lines in a field of plant rows often results in erroneous or even absent detections. This makes the robot highly instable and unable to navigate past a full row in most runs, as can be seen in Figure 4.10.

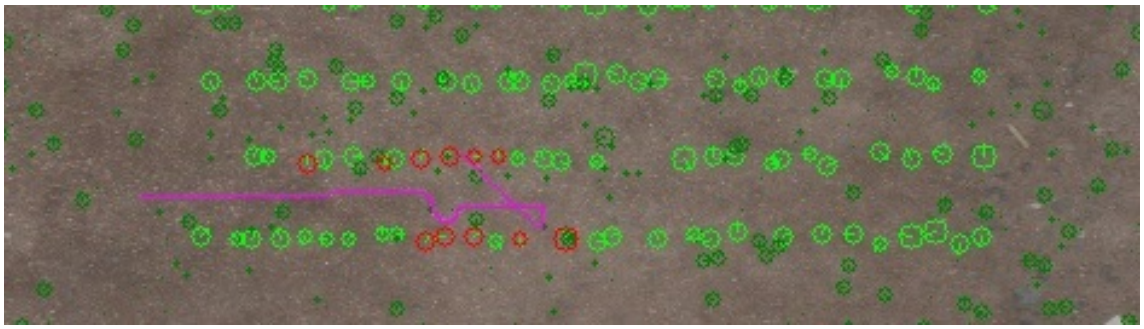


Figure 4.10: *PCL Lines* - observation result. The robot detected the end of row too early and started the blind turn maneuver. The simulation stopped because the robot crossed the second plant row.

Parallel Lines RANSAC This methods adds upon the previous one by searching for two parallel lines at a distance equal to that between two plant rows. Adding this additional information proves to have a big impact over the percent of row covered compared the *PCL Lines* method. Tests show that this method is still susceptible to gaps in the field, making the robot drive off its desired path. The result shown in Figure 4.11 depicts such a case. The gap on the left plant row could not be overcome by this method.

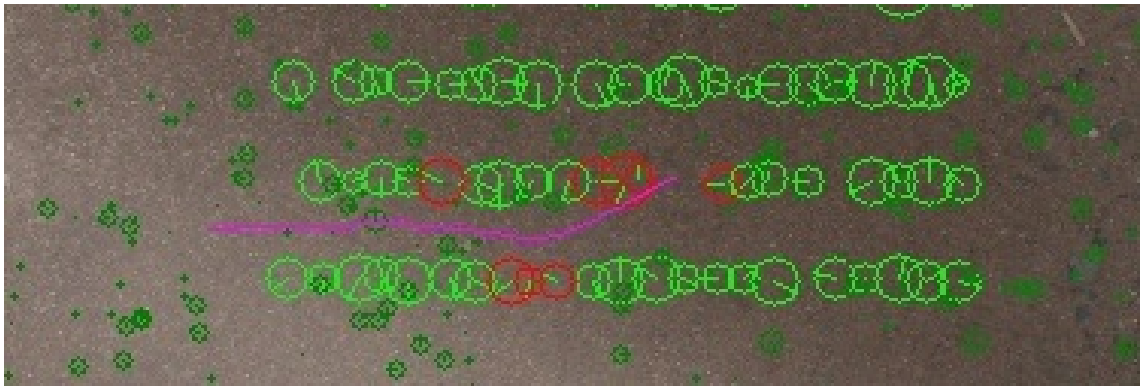


Figure 4.11: Parallel Lines RANSAC - observation result. The robot turned left and crossed the plant row.

Centroid Lines The *Centroid Lines* method tries to approach the problem differently. Instead of using a priori information about the field and try to fit it to what the robot sense - like the previous mentioned method - it instead gathers more LiDAR data to detect lines more robustly. This approach has similar results as the parallel lines method, both in row coverage and in driving accuracy - pointed out by the damaged plant metric and the navigation error. More information about the field helps it overcome small gaps in the plant row, but it is still prone to false detections of row ending when the gaps are too large. One example is presented in Figure 4.12.

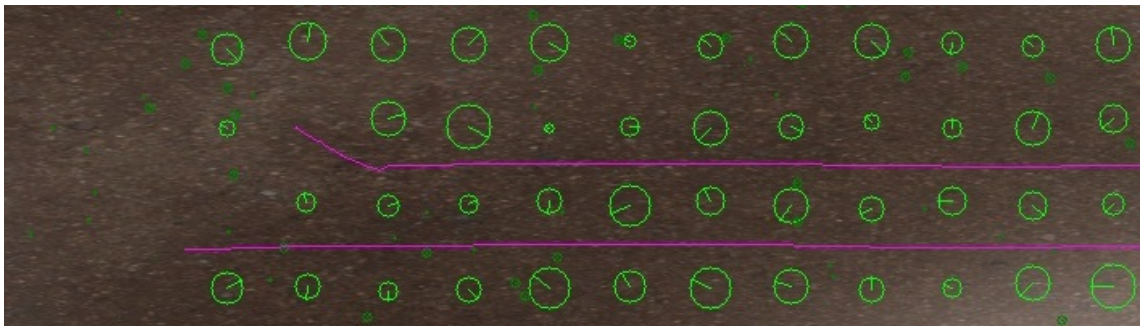


Figure 4.12: *Centroid Lines* - observation result. The robot turned right and crossed the plant row.

Map Lines This final method combines the advantages of the previous best methods. Integrating scans makes it less prone to reacting in a wrong manner because of sudden changes in landscape. Additionally, using known field information - the distance between rows and between plants - with the help of the grid fitting RANSAC algorithm proves to advance the performance even more. Both row coverage and driving accuracy show an increase in value when using this method.

As can be observed in Figure 4.9, the best method overall is *Map Lines*. It offers the best performance over all measured metrics. This is the reason why all following tests were conducted using just the *Map Lines* method.

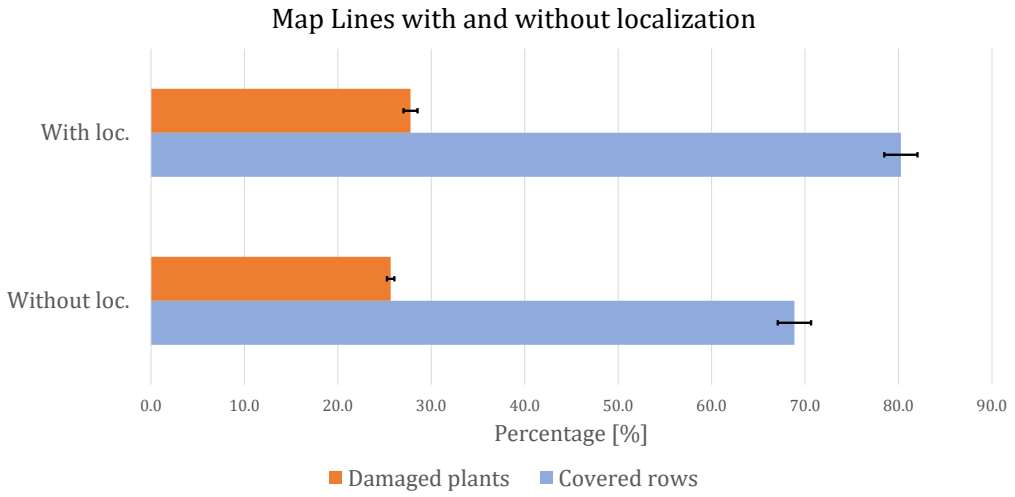


Figure 4.13: Localization results: *covered rows* and *damaged plants* metrics. The algorithm was run using the *Map Lines* method 10 times. Half of the test were run using localization and the other half without using localization.

Figure 4.13 shows the results of applying graph-based localization together with the *Map Lines* method. The effect of adding localization in the algorithm improved performance in terms of covered rows by 13 percent. This comes at the cost of increasing the number of damaged plants by 2 percent. The difference in accuracy while driving through the fields, between driving with localization and without, is unnoticeable. This can be observed from the results of the trajectory_error metric, illustrated in Figure 4.14.

Adding localization to the equation does not improve accuracy, but helps the robot to overcome some of the problems encountered in the previous methods. Having a good estimate of the robot's position in the field helps it navigate past big gaps to the end of the plant rows. Entering a new row when wrongly exiting the previous seems to be a common problem encountered by both *Parallel Lines RANSAC* and *Centroid Lines* algorithms. Localization now helps finding the new row when previously the robot had problems because it had to little information about newly started rows.

The missing 20 percent in row coverage from reaching the goal of navigating through all encountered fields seems to be caused by shortcomings in the perception part of the algorithms. When using the *Map Lines* method paired with localization, the robot has problems navigating fields with plants having shapes and sized towards the extremes. Fields with very small plants make it almost impossible for the robot to distinguish between data points belonging to the ground and the ones of plants. Not having information about the plants prevents the algorithm both from finding lines to navigate between and finding matches for the localization, making the robot unaware when it crosses a plant row, or drives too far. On the other extreme of the spectrum, when the tree crowns are so big that they start touching from one plant row to the other the robot encounters other problems. The algorithm is not able to differentiate between individual plants and sometimes not event between

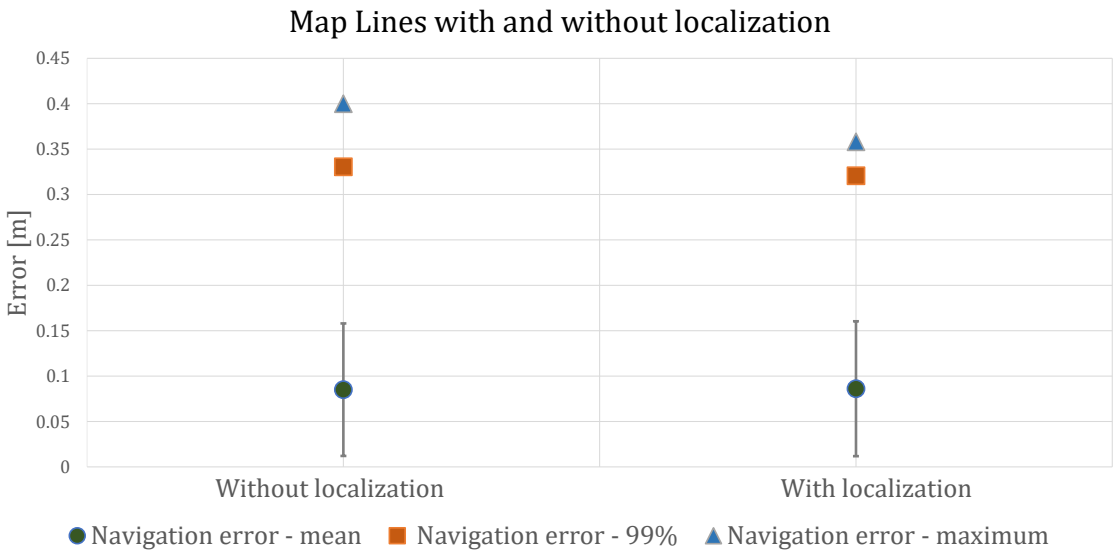
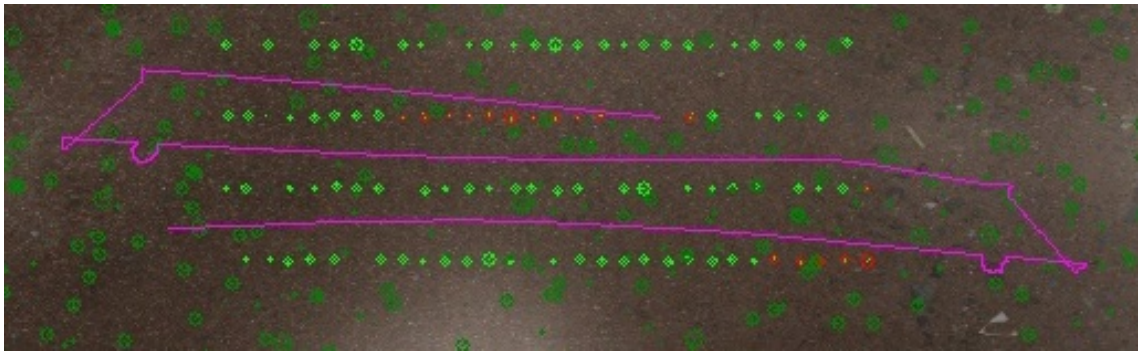


Figure 4.14: Localization results: *trajectory errors*. The algorithm was run using the *Map Lines* method 10 times. Half of the test were run using localization and the other half without using localization.

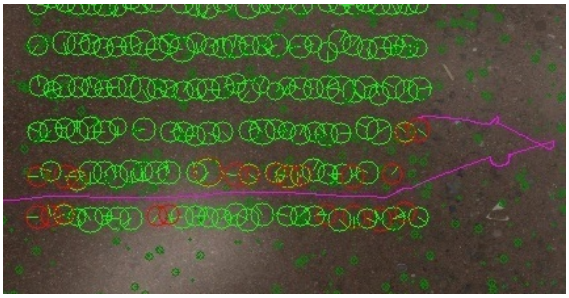
plant rows.In such fields the robot often sees obstacles in its way and no path to drive through. This is also a problem for the localization because it can not match all the points seen by the robot in a correct manner to the provided map.

Computation Time In terms of computational time, all methods perform in under 100 ms. This is the time constraint needed to achieve the 10 frames per second goal. The results of measuring the time elapsed for different part of code can be observed in Table 4.8. The *PCL Lines*, *Parallel Lines* and *Centroid Lines* methods compute the line detection result in negligible times with a mean of 0.5, 2.2 and 5.4 ms, respectively. The standard deviation of these times are 2.1, 4.3 and 5.2 ms, respectively. The *Map Lines* method is the most computationally expensive one, needing in average 83 ms, with a standard deviation of up to 50 ms. The localization part of the algorithm performs in just 0.8 ms, being unnoticed in the overall time consumption of the algorithm. The rest of the task, like filtering and detecting the ground sum up to less 10 ms. The first three methods allow for the FX10 sensor to run on a faster frame rate. Unfortunately, in this configuration, the *Map Lines* method is the bottleneck of the overall algorithm. This latter method can run with LiDAR sensor set only to a frame rate of 10 scans per second, or lower.

Controllers The two controllers were tested with the *Map Lines* method and using localization. The results show that the PID aids the algorithm in driving further in the simulated fields than the I/O lin. The results in Figure 4.16 shoes that the PID outperformed its counterpart by around 8 percent in terms of covered rows. But in terms of accuracy, as can be seed in Figure 4.16 and Figure 4.17, the I/O lin offers better performance. The percent of damaged plants and the trajectory error indicate that the I/O lin controller is better at keeping the robot in the middle of the row while driving.



(a) Field with small plants



(b) Field with overlapping plants

(c) Field with big plants

Figure 4.15: Figure (a) depicts a field where the plants are too small for the robot to detect.
Figure (b) depicts plants that are overlapping, resulting in plants to be clustered together.
Figure (c) depicts a field where the plants from different rows touch, creating a barrier for the robot.

	Mean [ms]	StdDev [ms]
Row detection methods		
PCL Lines	0.5	2.1
Parallel Lines RANSAC	2.2	4.3
Centroid Lines	5.4	5.2
Map Lines	83	58
Other modules		
Filtering	6.4	3.2
Ground detection	1.1	3.1
Localization	0.8	0.12

Table 4.8: Computation times. The results of running the algorithm in fields and timing the computational effort of module or pieces of code, measured in milliseconds.

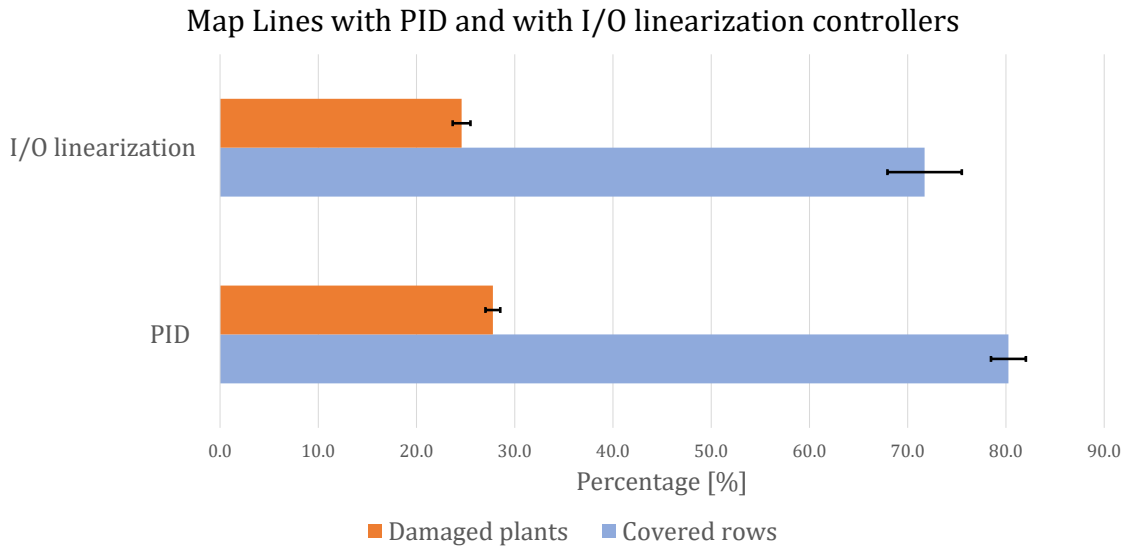


Figure 4.16: Controller comparison results: *covered rows* and *damaged plants* metrics. The algorithm was run using the *Map Lines* method 10 times. Half of the tests were run using the PID controller and the other half without using the I/O lin controller. Both type of tests were run using localization.

The difference between them is that the I/O lin controller is more reactive to changes, but is capable of following with more accuracy a desired trajectory. The PID controller follow a given path well and reacts slower than the I/O lin for sudden changes in line detections, which may occurs more often for the earlier developed methods like *PCL Lines*, *Parallel Lines RANSAC* and *Centroid Lines*. The test show that the *Map lines* method also suffers from instable line detection. This shows that more work needs to be done in order to better stabilize the detected lines.

4.4 Results

The developed algorithms were tested in simulated environment, comprised of 100 fields of various sizes and with plants of different dimensions. Afterwards an artificial garden made with plastic plants was built in a small garden in Renningen. Testing the row detection algorithm showed to be successful in both simulated and real scenarios.

4.4.1 Simulated Gardens

The simulations have shown that point cloud data provided by a LiDAR sensor like the FX10 is capable of detecting objects and correctly identifying them as plants in plant rows. Using this information it can accurately detect lines from plant rows for the purpose of navigating between them to the end of the fields.

Figure 4.18 shows the result of the algorithm with the *Map Lines* method and using localization deployed on a simulated field. The robot preformed a full drive through the field creating a graph with the estimated positions during its run. The green dots represent the plants belonging to the

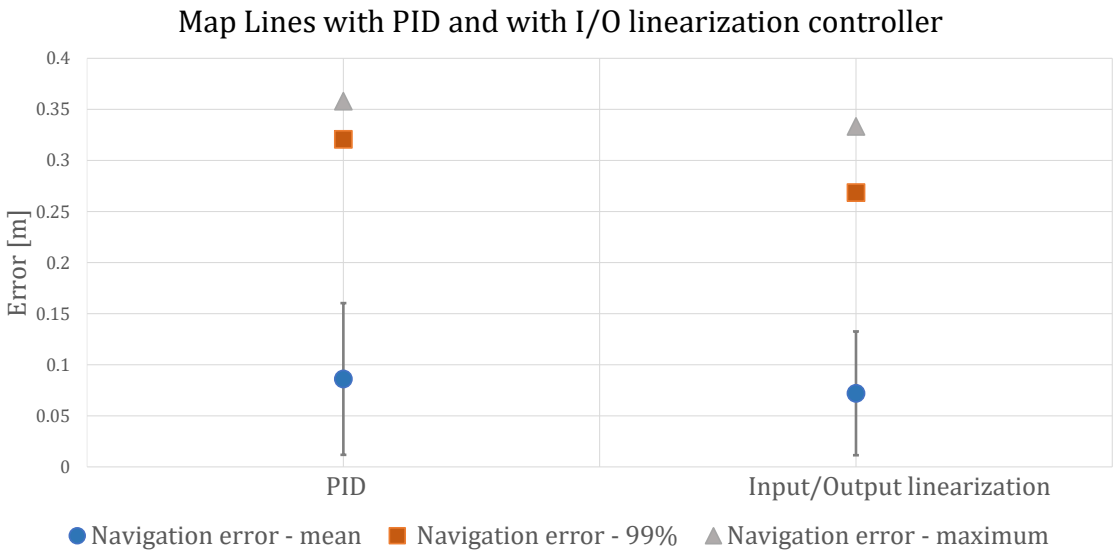


Figure 4.17: Controller comparison results: *trajectory errors*. The algorithm was run using the *Map Lines* method 10 times. Half of the tests were run using the PID controller and the other half without using the I/O lin controller. Both type of tests were run using localization.

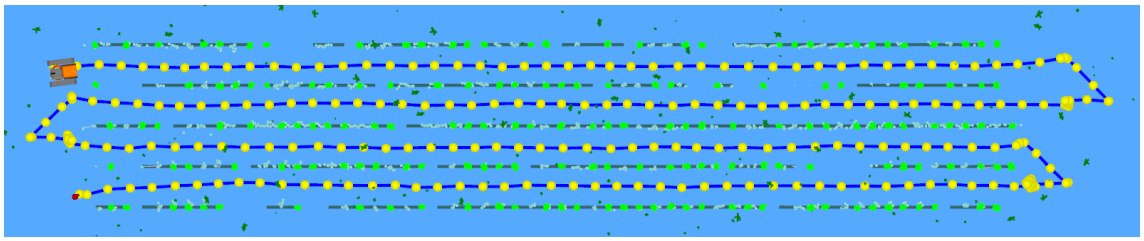


Figure 4.18: Field localization. The graph is drawn with yellow circles for nodes and blue lines connecting them representing the edges.

rows. The lines drawn between the green plants represent lines matched by the localization to the field map. These matches allows for an accurate pose estimations. As seen in the figure, the pose graph localization approximates correctly the path that the robot has driven inside the field, in the middle of the plant rows.

Using a graph-based localization algorithm enhances the robot's performance by up to 15 percent in term of field coverage when compared with methods that rely only on perception to navigate.

4.4.2 Real Garden

The final aim of this thesis is to have a functioning robot capable of autonomous navigation on a variety of tree nurseries. Although the simulated fields serve as accurate replica of real field, applying the concepts to a real life scenario usually poses different problems than the simulated world. In simulation it is often difficult to accurately simulate every aspect of the real world. The friction coefficient between the robot wheels and the ground are the biggest discrepancy between

real tests and simulation. This causes the odometry to behave differently and produce different results than those expected from analyzing the simulations. The blind driving at the end of the row needed to be adjusted to accommodate the slippage encountered by the robot. Another problem that the algorithm had to deal with was the uneven terrain in the real garden. The simulation did not cover large differences in soil height. The algorithm managed to overcome this problem without any changes.

The tests were run using the *Map lines* method using localization, combined with PID controllers. This setup was chosen based on the results from the simulation. Time and resources constraints did not permit to test every method on the real garden. Thus, only the best configuration of the algorithm, that yielded from the simulated results was chosen for testing.

The robot managed to cover 100 percent of the field in every run. All runs even yielded a *damaged_plants* metric of 0 percent. In some runs the robot touched between 1 and 3 plants, but not to the degree of being classified as damaged. The navigation error could not be assessed because it required the ground truth trajectory of the robot to be known. A lack of appropriate tools to do this hindered the calculation of this metric.

5 Discussion

For the purpose of this thesis several perception methods were developed with the aim of aiding a small robot to navigate autonomously through tree nurseries. The robot is equipped with a 3D LiDAR for vision and wheel encoders paired with an IMU for registering how the robot moves. The developed algorithms start from a simple implementation that makes use of a single LiDAR scan. They are further enhanced by integrating several scans. These scans enable creating a local map, which is used to localize the robot inside the field. The following section presents the results of testing these methods, both in simulated and real life scenarios. After results, the next two sections deliver the challenges faced while implementing and testing the methods and alternative steps that could be done to better improve the developed algorithms. The final statements of this Master thesis are expressed briefly at the end of this chapter.

Results

The goal of this Master thesis was to develop an algorithm that, deployed on a real robot, would allow it to navigate autonomously through tree nurseries in a robust fashion.

Several software modules were developed in this sense. A perception module was created with the purpose of handling point cloud information from a LiDAR, in order to detect the ground and plant rows. The perception module can receive information either from a FX10 sensor or from simulated data. In order to simulate LiDAR data, a ROS plug-in that searches inside an octree needed to be created. A localization pipeline was modified to suit the data coming from the perception module. In simulation, the odometry of a robot needs to also be produced. A ROS node that uses a motion model affected by noise solves the odometry simulation. Using a map generated from field characteristics, graph-based localization methods that make use of g2o optimization predict the estimate pose of a robot. The navigation module was developed to use information from perception and localization in order to drive a robot inside various gardens. Finally, an observer that monitors the robot's activity in simulation was developed to compute several metrics needed to assess how the robot performs.

The developed algorithms showed that in simulation the robot was able to overcome the majority of issues posed when driving through gardens with plants of different sizes and shapes. Making use of a localization algorithm the robot was able to navigate past gaps, i.e. not considering them end of rows. When adding localization, the performance of the algorithm that relied initially only on perception improved by 15 percent in terms of correctly driving inside fields. Moreover, the robot could correctly detect when rows truly ended. The result in this sense can be compared with other agricultural robots that accomplish the task of driving through field autonomously. However, most times these robots are equipped with bigger and better sensors than the ones used in the setup considered in this thesis.

Insights

Tests on the simulated fields show that the developed methods could be further improved. Even though the real test was successful, the simulations show that different types of gardens can still pose problems to the robot.

Problems such as fields with small plants would need better row detection methods. This can be dealt with either by improving the ground detection or finding a better way to distinguish between plant and soil. An additional sensor such as a RGB (Red Green Blue) camera could solve this problem by differentiating between them using color information. Another way this could be done is by processing more of the LiDAR information and making use of intensity and surface normals to detect between different types of surfaces.

Further work can be done also when dealing with large plants that are touching, situation that makes the current algorithm unable to detect accurately individual plants when creating clusters. Information about the approximate size of the plants in a certain field could help the clustering process to correctly distinguish between a number of plants touching and creating a large mass.

The result from the 100 fields showed that the robot can encounter problems when entering a new row. This problem often comes if the robot exits the previous row at a different angle than intended. Even though the robot might recover, in most cases when trying to enter the row at a bad angle results in crossing the plant row and stopping the simulation. Working towards a better way to handle driving at the end of the field, now executed blindly, should eliminate this kind of problems.

Working with the FX10 LiDAR and achieving good results for graph-based localization show that this type of sensor could also be used for mapping. Driving inside fields with remote control, or even autonomously, could allow the robot to also gather valuable information about the plants. This kind of data could refer to plant size or location of gaps, e.g. where the plants were dug out and moved.

Conclusion

The algorithms developed during this thesis allow a robot of small proportions to drive inside a variety of fields. A good performance was achieved when driving through both simulated and real scenarios. Using graph-based localization techniques coupled with perception algorithms showed an improvement in performance over approaches that just use perception. The graph-based optimization offers a framework for adding more information about the robot's movement and the environment, which helps the robot to localize itself even better.

The real life tests showed that the robot can handle the navigation through a small artificial garden, created with plastic plants. The robot faced challenges such as uneven terrain and slippage of the wheels at a degree that was not modeled in simulation. The row detection methods and localization deployed in the real garden managed to overcome these problems and perform full coverage. Running the algorithm successfully several times proved that the mission set at the start of the Master thesis has been achieved. A small robot platform equipped with an FX10 LiDAR sensor is

able to detect the artificial plants placed on the grass outside the robotics laboratory in Renningen. Making use of perception and localization, the robot is able to drive through the designed garden, even when gaps are present in the plant rows.

Bibliography

[AAIF10] M. Asif, S. Amir, A. Israr, M. Faraz. "A vision system for autonomous weed detection robot." In: *International Journal of Computer and Electrical Engineering* 2.3 (2010), p. 486 (cit. on p. 15).

[ÅB02] B. Åstrand, A.-J. Baerveldt. "An agricultural mobile robot with vision-based perception for mechanical weed control." In: *Autonomous robots* 13.1 (2002), pp. 21–35 (cit. on p. 13).

[AK06] M. Agrawal, K. Konolige. "Real-time localization in outdoor environments using stereo vision and inexpensive gps." In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 3. IEEE. 2006, pp. 1063–1068 (cit. on p. 18).

[BBM+10] T. Bakker, J. Bontsema, J. Müller, et al. "Systematic design of an autonomous platform for robotic weeding." In: *Journal of Terramechanics* 47.2 (2010), pp. 63–73 (cit. on p. 13).

[BGN+04] B. Blackmore, H. W. Griepentrog, H. Nielsen, M. Nørremark, J. Resting-Jeppesen. "Development of a deterministic autonomous tractor." In: *Proceedings CIGR*. Vol. 11. 2004, p. 2004 (cit. on p. 13).

[BJ04] T. Bak, H. Jakobsen. "Agricultural robotic platform with four wheel steering for weed detection." In: *Biosystems Engineering* 87.2 (2004), pp. 125–136 (cit. on pp. 13, 15).

[BMIN07] O. C. Barawid Jr, A. Mizushima, K. Ishii, N. Noguchi. "Development of an autonomous navigation system using a two-dimensional laser scanner in an orchard application." In: *Biosystems Engineering* 96.2 (2007), pp. 139–149 (cit. on p. 16).

[DB06] H. Durrant-Whyte, T. Bailey. "Simultaneous localization and mapping: part I." In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110 (cit. on p. 18).

[DK06] F. Dellaert, M. Kaess. "Square Root SAM: Simultaneous localization and mapping via square root information smoothing." In: *The International Journal of Robotics Research* 25.12 (2006), pp. 1181–1203 (cit. on p. 19).

[DOV01] A. De Luca, G. Oriolo, M. Vendittelli. "Control of wheeled mobile robots: An experimental overview." In: *Ramsete*. Springer, 2001, pp. 181–226 (cit. on p. 21).

[DSC+16] F. N. Dos Santos, H. Sobreira, D. Campos, R. Morais, A. P. Moreira, O. Contente. "Towards a reliable robot for steep slope vineyards monitoring." In: *Journal of Intelligent & Robotic Systems* 83.3-4 (2016), pp. 429–444 (cit. on p. 15).

[DT+15] M. P. Diago, J. Tardaguila, et al. "A new robot for vineyard monitoring." In: *Wine & Viticulture Journal* 30.3 (2015), p. 38 (cit. on p. 15).

[EP04] A. I. Eliazar, R. Parr. "Learning probabilistic motion models for mobile robots." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 32 (cit. on p. 30).

[FB87] M. A. Fischler, R. C. Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.” In: *Readings in computer vision*. Elsevier, 1987, pp. 726–740 (cit. on p. 16).

[FC04] J. Folkesson, H. Christensen. “Graphical SLAM-a self-correcting map.” In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 1. IEEE. 2004, pp. 383–390 (cit. on p. 19).

[FFM+15] C. Frese, C. Frey, F. Meßmer, K. Pfeiffer, S. Sander, D. Di Marco, M. Wenger, A. Albert, M. Wopfner, A. Burkhardt, et al. “AgriApps–An App-based Solution for Field-Robot-Based Agriculture.” In: *Land-Technik AgEng* (2015) (cit. on p. 13).

[GBM+] R. GUYONNEAU, E. BELIN, F. MERCIER, A. AHMAD, F. MALAVAZI. “Autonomous Robot for Weeding.” In: () (cit. on p. 15).

[GKSB10] G. Grisetti, R. Kummerle, C. Stachniss, W. Burgard. “A tutorial on graph-based SLAM.” In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43 (cit. on p. 19).

[HMT00] T. Hague, J. Marchant, N. Tillett. “Ground based sensing systems for autonomous agricultural vehicles.” In: *Computers and Electronics in Agriculture* 25.1-2 (2000), pp. 11–28 (cit. on p. 15).

[JNJ+12] K. Jensen, S. H. Nielsen, R. Joergensen, A. Boegild, N. Jacobsen, O. Joergensen, C. Jaeger-Hansen. “A low cost, modular robotics tool carrier for precision agriculture research.” In: *Proceedings 11th International Conference on Precision Agriculture Indianapolis, USA*. 2012 (cit. on p. 15).

[KGS+11] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard. “g 2 o: A general framework for graph optimization.” In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 3607–3613 (cit. on p. 23).

[LD06] V. Leemans, M.-F. Destain. “Application of the Hough transform for seed row localisation using machine vision.” In: *Biosystems Engineering* 94.3 (2006), pp. 325–336 (cit. on p. 15).

[LM97] F. Lu, E. Milios. “Globally consistent range scan alignment for environment mapping.” In: *Autonomous robots* 4.4 (1997), pp. 333–349 (cit. on p. 19).

[LPB+10] D. Longo, A. Pennisi, R. Bonsignore, G. Muscato, G. Schillaci, et al. “A multifunctional tracked vehicle able to operate in vineyards using gps and laser range-finder technology.” In: *International Conference Ragusa SHWA2010-September 16-18 2010 Ragusa Ibla Campus-Italy" Work safety and risk prevention in agro-food and forest systems*. 2010 (cit. on p. 15).

[LPK+16] F. Liebisch, J. Pfeifer, R. Khanna, P. Lottes, C. Stachniss, T. Falck, S. Sander, R. Siegwart, A. Walter, E. Galceran. “Flourish–A robotic approach for automation in crop management.” In: *Workshop Computer-Bildanalyse und Unbemannte autonom fliegende Systeme in der Landwirtschaft*. Vol. 21. 2016, p. 2016 (cit. on p. 13).

[LRWH14] A. Linz, A. Ruckelshausen, E. Wunder, J. Hertzberg. “Autonomous service robots for orchards and vineyards: 3D simulation environment of multi sensor-based navigation and applications.” In: *12th International Conference on Precison Agriculture, ISPA International Society of Precision Agriculture, Ed., Sacramento, CA, USA*. 2014 (cit. on p. 21).

[Mas14] G. Masiá. “Trakür. Plataforma autoguiada para aplicaciones de fitosanitarios.” In: *Simposio Internacional de Tecnologıas de Aplicación de Fitosanitarios. 3. Jornada Nacional de Agroelectrónica. 1. 2014 08 06-08, 6 al 8 de agosto de 2014. Concepción del Uruguay, Entre Rıos. AR.* 2014 (cit. on p. 15).

[Mea82] D. Meagher. “Geometric modeling using octree encoding.” In: *Computer graphics and image processing* 19.2 (1982), pp. 129–147 (cit. on p. 29).

[NGNS08] M. Nørremark, H. W. Griepentrog, J. Nielsen, H. T. Søgaard. “The development and assessment of the accuracy of an autonomous GPS-based system for intra-row mechanical weed control in row crops.” In: *Biosystems Engineering* 101.4 (2008), pp. 396–410 (cit. on p. 13).

[NRZ+98] N. Noguchi, J. F. Reid, Q. Zhang, J. D. Will, K. Ishii, et al. “Development of robot tractor based on RTK-GPS and gyroscope.” In: *2001 ASAE Annual Meeting*. American Society of Agricultural and Biological Engineers. 1998, p. 1 (cit. on p. 13).

[NUK+04] Y. Nagasaka, N. Umeda, Y. Kanetai, K. Taniwaki, Y. Sasaki. “Autonomous guidance for rice transplanting using global positioning and gyroscopes.” In: *Computers and electronics in agriculture* 43.3 (2004), pp. 223–234 (cit. on p. 13).

[PQZ+15] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, J. Gu. “The obstacle detection and obstacle avoidance algorithm based on 2-d lidar.” In: *Information and Automation, 2015 IEEE International Conference on*. IEEE. 2015, pp. 1648–1653 (cit. on p. 16).

[QCG+09] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng. “ROS: an open-source Robot Operating System.” In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. on p. 23).

[RBD+09] A. Ruckelshausen, P. Biber, M. Dorna, H. Gremmes, R. Klose, A. Linz, F. Rahe, R. Resch, M. Thiel, D. Trautz, et al. “BoniRob–an autonomous field robot platform for individual plant phenotyping.” In: *Precision agriculture* 9.841 (2009), p. 1 (cit. on p. 13).

[RC11] R. B. Rusu, S. Cousins. “3d is here: Point cloud library (pcl).” In: *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1–4 (cit. on p. 24).

[RKGB11] M. Ruhnke, R. Kümmerle, G. Grisetti, W. Burgard. “Highly accurate maximum likelihood laser mapping by jointly optimizing laser points and robot poses.” In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2812–2817 (cit. on p. 19).

[RKL+06] A. Ruckelshausen, R. Klose, A. Linz, J. Marquering, M. Thiel, S. Tölke. “Autonomous robots for weed control.” In: *Journal of Plant Diseases and Protection* (2006), pp. 173–180 (cit. on p. 13).

[RMA+16] D. Reiser, G. Miguel, M. V. Arellano, H. W. Griepentrog, D. S. Paraforos. “Crop row detection in maize for developing navigation algorithms under changing plant growth stages.” In: *Robot 2015: Second Iberian Robotics Conference*. Springer. 2016, pp. 371–382 (cit. on pp. 18, 24).

[RSM+07] R. RN Jorgensen, C. Sorensen, J. Maagaard, I. Havn, K. Jensen, H. Sogaard, L. Sorensen. “Hortibot: A system design of a robotic tool carrier for high-tech plant nursing.” In: (2007) (cit. on p. 13).

[SBA06] V. Subramanian, T. F. Burks, A. Arroyo. “Development of machine vision and laser radar based autonomous vehicle guidance systems for citrus grove navigation.” In: *Computers and electronics in agriculture* 53.2 (2006), pp. 130–143 (cit. on pp. 16, 21).

[SGD08] D. Slaughter, D. Giles, D. Downey. “Autonomous robotic weed control systems: A review.” In: *Computers and electronics in agriculture* 61.1 (2008), pp. 63–78 (cit. on p. 13).

[SWK07] R. Schnabel, R. Wahl, R. Klein. “Efficient RANSAC for point-cloud shape detection.” In: *Computer graphics forum*. Vol. 26. 2. Wiley Online Library. 2007, pp. 214–226 (cit. on p. 18).

[TGZZ13] J. Tang, N. Geng, Z. Zhang, Z. Zhu. “A Vision-Based Method of Wheat Row Detection for Agricultural Robot.” In: *International Journal of Digital Content Technology and its Applications* 7.5 (2013), p. 129 (cit. on p. 15).

[THM02] N. Tillett, T. Hague, S. Miles. “Inter-row vision guidance for mechanical weed control in sugar beet.” In: *Computers and Electronics in Agriculture* 33.3 (2002), pp. 163–177 (cit. on p. 13).

[THM98] N. Tillett, T. Hague, J. Marchant. “A robotic system for plant-scale husbandry.” In: *Journal of Agricultural Engineering Research* 69.2 (1998), pp. 169–178 (cit. on p. 13).

[TM06] S. Thrun, M. Montemerlo. “The graph SLAM algorithm with applications to large-scale mapping of urban structures.” In: *The International Journal of Robotics Research* 25.5-6 (2006), pp. 403–429 (cit. on p. 19).

[VBA+08] R. Van der Weide, P. Bleeker, V. Achten, L. Lotz, F. Fogelberg, B. Melander. “Innovation in mechanical weed control in crop rows.” In: *Weed research* 48.3 (2008), pp. 215–224 (cit. on p. 13).

[VLBG02] R. Van der Weide, L. Lotz, P. Bleeker, R. Groeneveld. “Het spanningsveld tussen beheren en beheersen van onkruiden op biologische bedrijven.” In: *Themaboek* 303 (2002), pp. 129–138 (cit. on p. 13).

[WB10] U. Weiss, P. Biber. “Semantic place classification and mapping for autonomous agricultural robots.” In: *Proceeding of IROS Workshop on Semantic Mapping and Autonomous, Knowledge Acquisition, In*. Citeseer. 2010 (cit. on p. 15).

[WB11] U. Weiss, P. Biber. “Plant detection and mapping for agricultural robots using a 3D LIDAR sensor.” In: *Robotics and autonomous systems* 59.5 (2011), pp. 265–273 (cit. on pp. 15, 18).

[WBL+10] U. Weiss, P. Biber, S. Laible, K. Bohlmann, A. Zell. “Plant species classification using a 3D LIDAR sensor and machine learning.” In: *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*. IEEE. 2010, pp. 339–345 (cit. on p. 16).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature