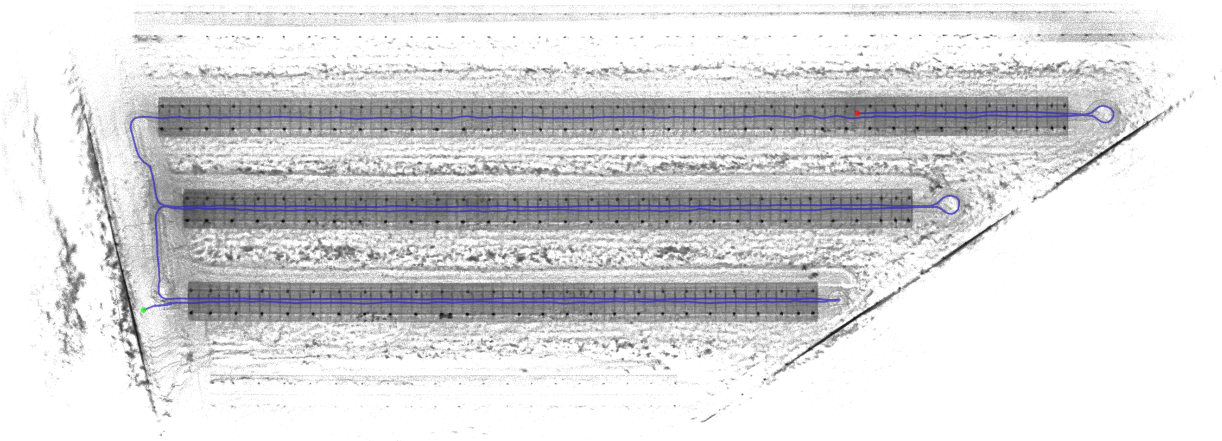
CHALMERS
UNIVERSITY OF TECHNOLOGY



LiDAR based Graph-SLAM for Autonomous operation in Solar Parks

Master's thesis in MPSYS

Nima Salmanpour

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

LiDAR based Graph-SLAM for Autonomous operation in Solar Parks

Nima Salmanpour

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

LiDAR based Graph-SLAM for Autonomous operation in Solar Parks

NIMA SALMANPOUR

Supervisor: Jonas Hejderup, Husqvarna Group AB
Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Map of three rows of solar panels created with a LiDAR-SLAM algorithm.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

LiDAR based Graph-SLAM for Autonomous operation in Solar Parks
Nima Salmanpour
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous vegetation control in solar parks is becoming an increasingly important and relevant field as the number of parks increase with the accelerating need to transition to renewable sources. These parks are characterized by long featureless structures, tall, dense vegetation and poor Global Navigation Satellite System (GNSS) reception. Extensive research has been made into autonomous control in vehicle and industry settings while the challenges involved in adapting these to solar parks remain mostly unknown. The purpose of this work is to utilize a Simultaneous Localization and Mapping (SLAM) algorithm to accurately localize a Husqvarna mower equipped with a LiDAR sensor in the solar park while also creating a map of said park. The implementation should be unaffected by tall vegetation, have minimal drift, create maps of high quality and should not depend on GNSS to achieve this. The SLAM of choice is the cartographer implementation which is a graph-SLAM variant. Furthermore a custom pre-filter, which mainly utilizes RANSAC to detect certain features in the LiDAR data, is designed to improve the localization and mapping results. The proposed solution manages to localize in both high and low vegetation but the accuracy is somewhat subpar with noticeable translational and rotational drift present. The pre-filter manages to improve the quality of the maps by successfully removing the majority of the vegetation, thus reducing the clutter in the generated maps, while the drift remains mostly unimproved. Currently the most efficient way to counter and correct this drift is with the use of GNSS, however this is not a perfect solution due to the aforementioned problems with reception and ideally an alternate solution should be developed. With improvements done in these areas this implementation shows great promise in achieving robust and reliable autonomous control in solar parks and environments of similar characteristics.

Keywords: SLAM, Cartographer, LiDAR, Renewable, Solar Park, Autonomous, Robot, RANSAC

Acknowledgements

I would like to express my gratitude to my supervisors at Husqvarna, Jonas Hejderup, Peter Svenningsson and Christian Vaugelade-Kilaas for their support and help during this project. I would also like to thank my examiner Lars Hammarstrand for his assistance and interesting discussions during this project.

Nima Salmanpour, Gothenburg, Mars 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

EKF	Extended Kalman Filter
FOV	Field of View
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LiDAR	Light Detection And Ranging
MRSE	Mean Root Squared Error
RANSAC	RANdom SAmple Consensus
ROS	Robot Operating System
RTK	Real Time Kinematic

Contents

List of Figures

1

Introduction

As the need for fossil free electricity production has increased solar energy has emerged as one of foremost alternatives with prices coming down and production increasing every year. According to [1] a yearly growth of 25% is needed until 2030 in order to reach net zero emissions by 2050. Husqvarna AB seeks to electrify and automate the lawn mowing process of these parks in order to replace fossil powered and manually operated machinery which are currently in use [2]. The proposed solution is to utilize an autonomous and fully electric mower which navigates between the panels and cuts the vegetation.

The mower uses an array of sensors, including a LiDAR (Light Detection and Ranging) range sensor, for localization and navigation through heavy vegetation without hitting or damaging the solar panels. The LiDAR is currently mainly used for localization by detecting different known parts of the solar panels such as the poles or panel edges. This is needed in order for the mower to follow a programmed path since Global Navigation Satellite System (GNSS) reception is periodically poor and must be complemented with other data. In order for the aforementioned path to be correctly determined so that the mower cuts all the vegetation without hitting any obstacles and for it to be able to localize correctly during GNSS outages, Husqvarna requires a decent ground truth map during the initialization of the site. These are rarely available and thus other alternatives must be considered.

One alternative way is to utilize high resolution areal photos or even blueprints from the construction of each solar park to determine the position of panels and obstacles at each site. This however is not an ideal process as it is limited by the quality and resolution of the images and the accuracy of the construction compared to the blueprints. Furthermore the process requires a significant amount of effort and manual work, especially for larger solar parks. An alternative way to achieve both a high quality map and localize the mower within the map is to utilize the LiDAR sensor already installed on the mower and a Simultaneous Localization and Mapping (SLAM) algorithm to localize and map in the solar park. The implementation is made more difficult by the very high level of vegetation present in solar parks. This vegetation can vary in density, height and even sway with the wind and thus it must also be dealt with.

1.1 Purpose

The Purpose of this project is implement a SLAM algorithm using data from the LiDAR sensor to localize a robot and map a solar park. Furthermore a RANSAC (RANdom SAmple Consensus) based algorithm should be implemented as a pre-filter with the purpose of removing any vegetation from the LiDAR data in order to further improve the accuracy and performance of the mapping and localization.

1.1.1 Objectives

To fulfill the purpose as defined above, the following research questions will be investigated:

- How well does a current SLAM solution manage to map and localize in this terrain using the LiDAR sensor on the Husqvarna mower? Are there any issues or best practices to consider when recording the data?
- Does the vegetation affect the map generation or the aforementioned accuracy of the map? Is it possible to compensate for any loss of accuracy caused by this? Is it possible to improve the results of by implementing a prefilter which filters out the vegetation. What drawbacks does this have?
- Do long GNSS blackouts affect the accuracy of the map and if so at what point does it become problematic with regards to accuracy? Is the algorithm robust enough to maintain accuracy even at prolonged blackouts?

2
Theory

In order to fully understand the methods and proposed solutions, some theory is necessary and will be presented and explained in the following section. The section will start with a general explanation of SLAM, the sensors used and different algorithms available. Then a section with a more detailed explanation of the inner workings of a specific SLAM and finally a section about the theory for the prefilters.

2.1 SLAM

Simultaneous Localization and Mapping (SLAM) is a family of algorithms used in robotics and autonomous systems with the purpose of building a map of an unknown environment while simultaneously keeping track of the robot's location within it, in particular in places where GNSS or external positioning systems are not available. While there are many algorithms they mostly fit the same conceptual description. According to [3] these steps are the following

- Generation of the Initial Environmental Map: The SLAM process begins with the initial data acquisition phase. During this stage, the robot's sensors actively collect data to construct a rudimentary map of the environment. This foundational map is not yet complete but offers a starting point for the robot by providing essential information about the immediate surroundings. Various sensors, such as LiDAR for precise distance measurement or cameras for visual cues, contribute to creating this initial spatial understanding.
- Establishment of the Robot's Starting Position and Landmark Computation: Once the initial environmental sketch is obtained, the robot must establish its position within this nascent map. Determining the starting position is crucial as it serves as a reference point for all subsequent navigation and mapping. At this juncture, the algorithm also computes and extracts identifiable landmarks from the initial data. These landmarks are features of the environment—distinctive and recognizable elements like edges, corners, or specific objects—that the robot uses to triangulate its position. The recognition and computation of these landmarks are derived from the data collected and the initial map, setting the stage for the robot to start its journey.
- Concurrent Mapping and Localization During Movement: As the robot moves through the environment, it doesn't merely passively traverse space but engages in an ongoing process of remapping and self-localization. The SLAM algorithm operates in a recursive manner, continually incorporating new sensory data to refine the map. This refinement is an iterative enhancement,

where each new piece of information contributes to a more detailed and comprehensive map. The updated map includes new environmental features and obstacles, thereby enriching the robot's "knowledge" and understanding of the space it inhabits.

- Map Creation and Update: As the robot moves, it continues to gather data and identify new features. It uses this information to update its map of the environment, adding new features and refining the positions of previously identified ones. This process is iterative and continues throughout the robot's operation, allowing the map to become more detailed and accurate over time.
- Association of Landmarks and Refinement of the Robot's Position: A critical aspect of SLAM is the ability to not only discover new landmarks but to also understand them in the context of those previously identified. The robot associates new landmarks with existing ones, a method that helps in correcting any positional drift that might have occurred. By drawing these associations and reconciling the new data with the old, the robot can update and correct its estimated location within the map, thereby enhancing its navigational accuracy.
- Navigation and Dynamic Planning: The robot navigates through the environment, dynamically adjusting its path planning in response to the continuous influx of updated information from its sensors.

A flow chart of this process can be seen in figure 2.1.

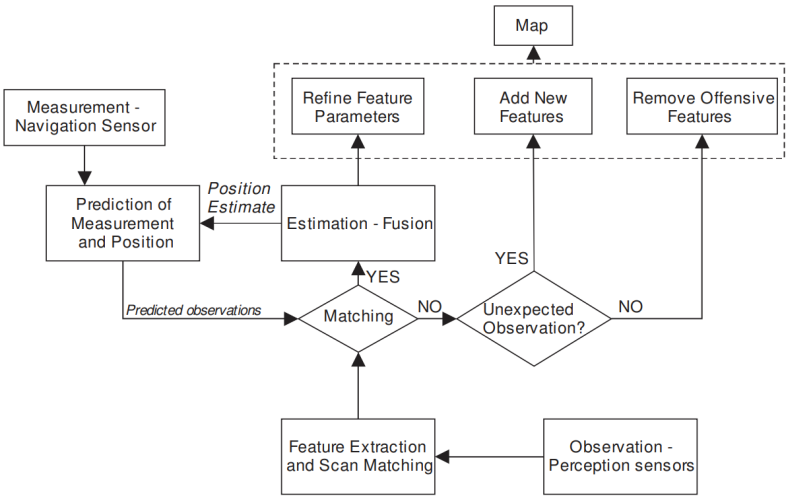


Figure 2.1: A flowchart from [4] which describes a general SLAM process.

Next, a more mathematical formulation of the SLAM problem will be given. This will be based on the description in [3]. Consider a mobile robot moving through an environment taking relative observations of a number of unknown landmarks using a sensor located on the robot as shown in Figure 2.2. At time k we define

- x_k: The state vector of the robot.
- u_k: The odometry of the robot at time k.
- m_i: A vector of the coordinates of the ith landmark
- z_k: Measurements between the robot and landmarks.

- X_k: The path of the robot.
- U_k: Sequence of robot odometry.

After gathering and defining all of the necessary information, the subsequent stages involve constructing a map view of the environment and predicting the location. The SLAM technique employs a probabilistic approach that utilizes a probability distribution to estimate the positions of the robot and landmarks based on the map created. This probability distribution, denoted as P, is defined as

$$\mathrm{P}(x_k, m|Z_k, U_k)$$

This section is interpreted as calculating the probability of the robot's position at time k and the map, based on the historical measurements and odometry data. Furthermore, this calculation necessitates an additional component, known as the motion model. The motion model establishes the correlation between the robot's position, denoted as x_k , and the odometry data, u_k, and is defined accordingly.

$$\mathrm{P}(x_k|x_{k-1}, u_k)$$

Next we introduce the observation model. It is defined as the relationship between sensory measurements z_k, map environment m and robot position x_k which is defined as

$$\mathrm{P}(z_k|x_k, m)$$

In a typical environment, a robot is capable of discerning the distance to a landmark, the direction relative to itself, and the unique identity of that landmark. This capability forms the foundation for developing a measurement model. Essentially, this model leverages the robot's ability to identify and locate landmarks to understand its surroundings better. By incorporating a probability distribution into the measurement model, it becomes possible to account for uncertainties and variations in the robot's perceptions and measurements. This probabilistic approach allows for a more accurate representation of the environment by acknowledging the inherent inaccuracies in sensor data. Therefore, the measurement model, enriched with probabilistic reasoning, serves as a mathematical framework to predict the robot's location relative to identified landmarks with a defined degree of confidence. It is defined as

$$\mathrm{P}(z_k|x_k, m) \sim \mathcal{N}(h(x_k, m), Q_k)$$

where $h/(x_k, m)$ is a function that represents the expected measurement value given the robot's position x_k and the map m. $\mathcal{N}$ is the normal distribution and Q_k is the noise covariance.

The motion model employs a Gaussian (normal) distribution centered on the covariance of the kinematic motion model. The formulation of the motion model is encapsulated by the equation:

$$P(x_k|x_{k-1}, u_k) = \mathcal{N}(g(x_{k-1}, u_k), R_k)$$

where $g(x_{k-1}, u_k)$ represents the standard kinematic function, calculating the robot's new position x_k based on its previous position x_{k-1} and the motion information u_k

obtained from odometry. Here, R_k denotes the three-dimensional covariance matrix associated with motion noise. This function effectively merges the prior position with changes dictated by odometry, projecting the updated position.
As the robot progresses, it refines its position estimate and landmark measurements based on the preceding timestep $k-1$, harnessing improved accuracy through data assimilated from landmarks and its prior location at timestep k. This iterative procedure of position updating continues throughout the robot's exploration of the environment, enhancing the precision of robot measurements systematically.

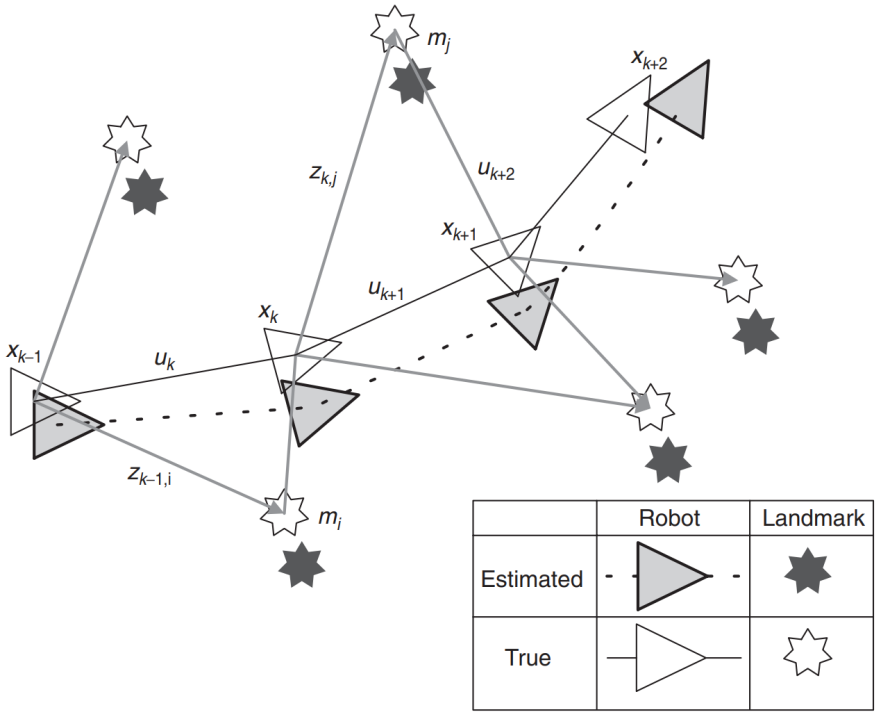


Figure 2.2: The essential SLAM problem. [5]

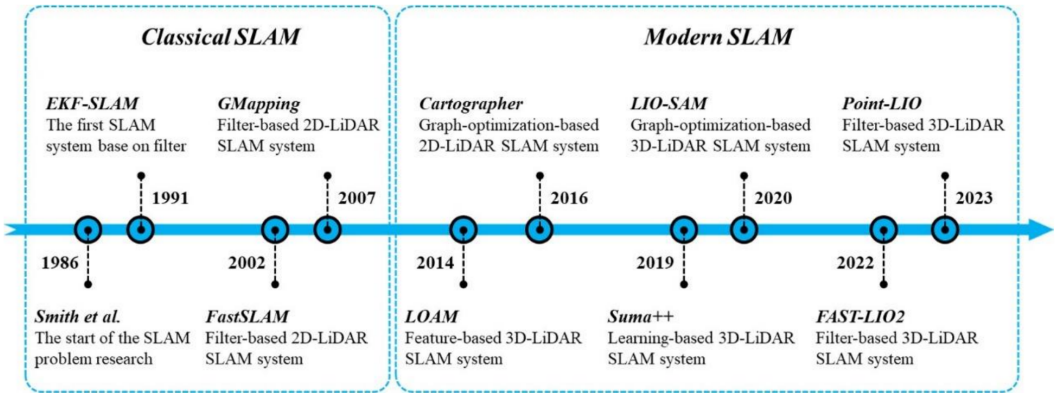


Figure 2.3: A timeline of the different SLAM algorithms by [6]. While Cartographer is classified as a 2D SLAM, which it was at the time of launch, it has since been updated with 3D capabilities.

2.2 ROS

According to [7] the Robot Operating System (ROS) is an open-source platform designed for developing robot software. It offers libraries and tools to facilitate the assembly and coordination of complex robotic systems. ROS is especially known for its modular design which makes it compatible on many different hardware setups and useful for a wide variety of tasks. This modularity further streamlines the debugging process for individual components. In ROS, software is organized into packages, which encompass software units like nodes or libraries. A typical robotic system operates through a network of nodes, with each node managing a specific subtask of the overall system.

2.3 LiDAR

According to [8] light detection and ranging (LiDAR) is an optical sensor technology that measures distance to a target by illuminating the target with laser light and measuring the reflection with a sensor. Differences in laser return times is then used to make digital 3D representations of the surrounding. LiDAR sensors are widely used in various applications, including autonomous vehicles, where they provide high-resolution maps of the vehicle's surroundings for navigation and obstacle avoidance. In these systems, LiDAR sensors can detect the shape, distance, and orientation of objects around them.

According to [9] LiDAR technologies are categorized into two primary types: mechanical (or spinning) LiDAR and solid-state LiDAR. Mechanical LiDAR operates with either a prism or a spinning mirror to scan across a horizontal field of view (FoV) of up to 360 degrees and a vertical FoV of up to 120 degrees. In contrast, solid-state LiDAR leverages electronic mechanisms for laser beam steering and data acquisition. Solid-state LiDAR itself encompasses three variants: flash-based LiDAR, microelectromechanical systems (MEMS)-based LiDAR, and optical phased array LiDAR. The horizontal FoV for solid-state LiDAR devices varies but it often it is under 120 degrees and certainly less then the 360 degrees of the spinning LiDAR.

2.4 Point Cloud

Point clouds are a pivotal data structure in three-dimensional (3D) spatial analysis, where a point cloud P is defined as a set of points $P = \{p_1, p_2, \ldots, p_n\}$, with each point p_i represented as a tuple (x_i, y_i, z_i, I_i) [10]. In this representation, (x_i, y_i, z_i) denote the spatial coordinates of a point within a Cartesian coordinate system, and I_i signifies the intensity of the returned signal, which reflects the reflectivity of the surface from which the laser light is bounced back. This intensity metric can yield insights into the material characteristics of the scanned object's surface.

Mathematically, the analysis of the points in a point cloud can be done through various algorithms aimed at surface reconstruction, shape identification, and fea-

ture detection and segmentation. Algorithms like the Random Sample Consensus (RANSAC) are commonly applied to derive meaningful geometric structures from the dispersed point data. Additionally, point clouds are subjected to a series of pre-processing steps including noise reduction, gap filling, and data enhancement to improve the fidelity of the resultant 3D models. In applications ranging from areal mapping and urban planning to autonomous vehicle navigation, point clouds serve as a direct digital representation of the physical world, capturing its geometry and surface properties with high precision.

2.5 Cartographer

Cartographer is a system that employs SLAM (Simultaneous Localization and Mapping) technology tailored for various sensor setups. It is a graph-based SLAM which was released in 2016, as can be seen in Figure 2.3. It is divided into two distinct subsystems. The first is local SLAM, where each new scan is aligned with a segment of the environment known as submap N. This submap is made up of only a few scans and gradually develops a bias error. To correct this drift, the second subsystem, global SLAM, is used. Global SLAM works with multiple stored submaps and searches for recurring patterns among them to link them effectively. Cartographer identifies structural features like wall shapes and attempts to align the current submap with the configurations of a previous one. By recognizing familiar landmarks, such as a corner of a wall, Cartographer can triangulate its position using past data and connect various submaps.

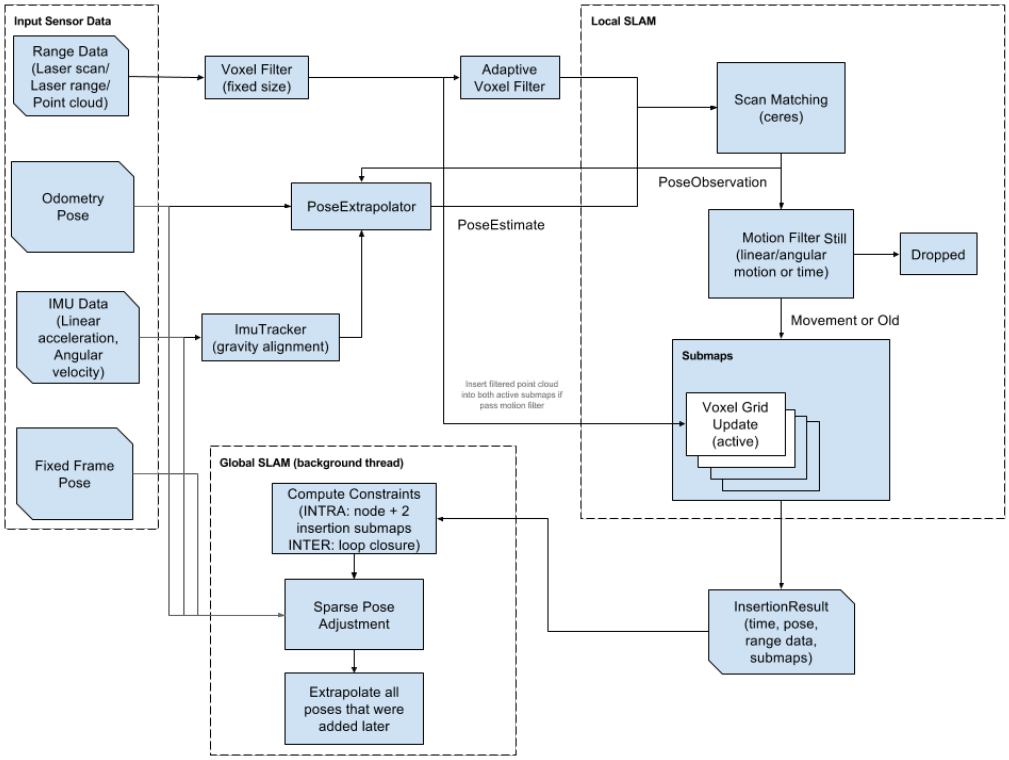


Figure 2.4: A high level overview of the Cartographer algorithm from [11].

Below a general explanation of the algorithm based on [[12], [13]] will be given. A high level overview of the algorithm can be viewed in figure 2.4. Here the input odometry pose and IMU are the relative robot odometry u and the range data z is the distance data from the LiDAR or depth cameras. The IMU measures the linear acceleration a and angular velocity ω which the pose extrapolator in Figure 2.4 integrates to obtain an estimation of the pose. The IMU acceleration data a is then transformed from local to global coordinates according to

$$\hat{a} = T\dot{a}$$
$$T = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

where θ is the rotation in the z-axis between the global and local coordinate frame. Thus it can be estimated using the IMU by integrating ω_z

$$\theta_t = \int_{t-1}^{t} \omega_z dt + \theta_{t-1}$$

where θ_{t-1} is the previous rotation estimated in the scan matching block in fFgure 2.4. Similarly the position p is given by

$$p_t = \int_{t-1}^{t} \hat{a}_t \dot{dt} + p_{t-1}$$

Thus the pose extrapolator block outputs the pose in the global frame.
The scan matching block in Figure 2.4 takes in LiDAR range data z and a pose

estimate from the previously described block and outputs a pose observation. This is done by comparing different LiDAR scans and finding similar geometry in them. If the scans do not align, the rotational and translational discrepancy between them can be calculated, which yields an estimation of how the robot moved between those two scans. The pose estimates from both the scan matcher and the pose extrapolator function block are compared. If the scan matcher confirms that it has recognized the same geometrical features in different scans we can be confident in the pose estimation.

2.6 RANSAC

This section on the Random Sample Consensus (RANSAC) is based on [14]. The RANSAC algorithm is a general parameter estimation approach designed to identify and model the underlying structure of data points that are contaminated with a significant amount of noise and outliers. RANSAC has it's roots within the computer vision community, which is unusual as most robust estimation techniques come from the statistics sciences. At its core, RANSAC operates by iteratively selecting a random subset of the original data points and using these points to fit a model. It then assesses this model by calculating how many of the remaining points fit well with it, where a point is considered a "fit" if its residual is below a predefined threshold. These well-fitting points are labeled as inliers, while those that do not fit the model are deemed outliers. This process is repeated for a specified number of iterations, and the model with the highest number of inliers is selected as the best representation of the underlying data structure. The following is a more structured description:

Algorithm 1 RANSAC Algorithm

1: Randomly select minimal points to fit a model.
2: Estimate model parameters.
3: Count inliers within tolerance ϵ.
4: **if** inliers ratio $> \tau$ **then**
5: Refit model using all inliers and stop.
6: **else**
7: Repeat up to N iterations.
8: **end if**

The number of iterations N, is determined to be sufficiently large to achieve a probability p that at least one sample set randomly chosen does not contain any outliers. The probability of selecting an inlier is denoted u while $v = 1 - u$ is the likelihood of picking an outlier. The formula to determine the required number of iterations N for the algorithm to achieve the desired confidence level p, iterating over a minimal set of points denoted by m, is given by:

$$1 - p = (1 - u^m)^N.$$

Through algebraic manipulation, this can be reformulated to express N as:

$$N = \frac{\log(1-p)}{\log(1-(1-v)^m)}.$$

This formulation calculates the necessary iterations to ensure the model's reliability by effectively excluding outliers and focusing on inliers within the dataset.

3
Implementation

This chapter will explain the hardware and software implementation of this thesis in more detail. All hardware was provided by Husqvarna and is based on a highly modified version of the Raymo electric mower. Furthermore some very critical details regarding the testing and evaluation will be explained.

3.1 Hardware

Figure 3.1: The basis of the robot used in the project from [15]. Note that this image does not include the many sensors and components that have been added to enable autonoumous operation.

The hardware used was a modified Raymo electric mower as seen in Figure 3.1, with a computer and various sensors added to enable autonomous operation. The robot is equipped with a wide range of sensors, but for this project the main sensors used were LiDAR and IMU. The LiDAR was a Oyster OS-1-64 which according to [16] have a vertical resolution of 64 channels and a horizontal resolution of 1024. Furthermore the accuracy is at most ± 3 cm at 90 m distance according to the manufacturer. The LiDAR is sampled at a frequency of 10 Hz and each scan is gathered as one `PointCloud2` message in ROS.

In addition to this the robot was equipped with the ublox ZED-F9P RTK-GNSS chipset which according to [17] supports all the current GNSS constallations (GPS,

GLONASS, BEIDOU, GALILEO) and is capable of a sub 1 cm accuracy when corrected with RTK. In reality, internal testing showed that this is closer to 3 cm with the real number varying depending on the distance to the RTK ground station, but it is still low enough in our case so that it can be omitted.

3.2 Software

3.2.1 SLAM

While there are many SLAM algorithms and implementations to chose from, for this thesis it was decided that Cartographer was the most suitable. This was done for several reasons. Primarily the graph based approach which according to [18] is suited for mapping large sites, the native ROS support, the native GNSS support and the somewhat decent documentation. Furthermore according to [[19],[20],[21]] it has performed well in other challenging scenarios. It is capable of running completely offline and offer flexibility in terms of configuration with wide support for various sensors. Cartographer's ability to perform offline is particularly advantageous for in scenarios where testing is done at difficult to reach locations such as in this thesis.

3.2.2 Pre-filter

In order to better deal with the unique setting and more specifically the dense vegetation which often varies greatly in one and the same park, different pre-filters were implemented and tested. They are of various complexity but principally all work by removing vegetation from the LiDAR data. Functionally they were also coded in a similar way, a python script would load the rosbag which contains the recorded data, iterate over each LiDAR scan, implement the filter algorithm and finally save the edited data to a new rosbag. The most important factor that needed to be considered was that the timestamps of the data was left absolutely identical in the original and edited data. The pseudocode which achieves this process can be seen in algorithm 2.

3.2.2.1 Minimum Height Filtering

The most trivial algorithm was simply a height restriction on the point clouds, meaning all points under a height $h = 0.5$m would be immediately omitted. This would inevitably remove some of the panel poles and any static structure even if it is not vegetation, due to the indiscriminate nature of the condition. It is understood that this is not an ideal implementation and it is only meant as a rudimentary, fast and easy to implement, alternative to use as comparison for the more complex and computationally intensive filters.

3.2.2.2 RANSAC Ground Segmentation

The second filter that was introduced aimed at implementing ground detection by conceptualizing the ground surface as a planar structure. The foundation of this

Algorithm 2 General structure of an algorithm for processing data from the rosbags and saving them with an unchanged timestamp

```
Open inputPath for reading and outputPath for writing
for each topic, msg, t in input bag messages do
  if topic matches specific condition then
    if message requires modification then
      modifiedMsg ← modify message based on condition
      Write modifiedMsg to output bag with topic and t
    else
      Write msg to output bag with topic and t
    end if
  else
    Write msg to output bag with topic and t
  end if
end for
Close input and output bags
```

approach was based on the RANSAC algorithm which attempts to fit a plane characterized by the equation $ax + by + cz + d = 0$ to a random subset of three points in the point cloud. The plane was then evaluated for all points and the number of inliers was calculated, i.e. the number of points which are within a set distance from the estimated plane. For each iteration a new subset of points was considered until finally the iteration with the highest number of inliers was selected as the best model, and the plane from this model was considered to represent the ground. The RANSAC implementation was based on the open3D function `segment_plane` which according to [22] utilizes the RANSAC algorithm and offers good Python integration. In order to prevent it from detecting the solar panels, since besides the ground they are the largest planar surfaces present in the data, a filter was set in place to ensure that only points below 0.7 m would be included.

3.2.2.3 RANSAC Panel and Pole Segmentation

Finally our last filer used RANSAC similar to the previous one, but this time was specifically configured to detect and segment the panel and poles, effectively removing all other points from the point cloud. This adaptation of the RANSAC algorithm focused on the unique shapes and sizes of the panel and poles, distinguishing them from other elements in the data. The process involves picking a RANSAC model which does not mislabel vegetation as poles and implementing sectioning the data to ensure that the most prominent shape was the panel or poles. This ensured that only relevant data is retained for further analysis. While the algorithm itself isn't overly complex, its effectiveness lies in the precise targeting of specific features within the point cloud and RANSACs robustness to outliers of which we predicted there would be many in the more challenging dataset. Below, an outline of the steps in the filter is presented.

- The panel detection is similar to the ground detection in that it used the same `segmant_plane` function as before. To prevent it from detecting the ground

plane all points below a height of $h = 1\,\text{m}$ were removed from the data before the RANSAC loop.
- When the panel has been detected, the coordinates of the points corresponding to the panels perimeter is considered and the area beneath them is segmented in 1x1 m blocks. We are not concerned about the height since there are no points above the panel.
- A custom RANSAC algorithm is run in each of these created blocks. The RANSAC attempts to find a line, defined by two points, which contains the most amount of inliers in each block. We assume the poles are the most prominent vertical lines and thus we add a constraint to our RANSAC line model which dictates that every randomly created line must be within a few degrees of vertical. If the randomly selected points do not fulfill this new points are chosen.
- Any resulting inliers will be considered a panel pole.
- The points detected by our RANSACs as panels or poles are saved while all other points are removed.

This filter was implemented in Python and while computationally heavy was, similarly to the previous filters, implemented in a way to run completely offline which negated the need for optimization. It should also be noted that this filter is not able to detect the diagonal bars that sometimes are present in the substructure of solar panels.

Algorithm 3 RANSAC Panel and Pole detecting filter

```
elevatedPoints ← filter_points_above_height(pointCloud, 1)
panel ← segment_plane(elevatedPoints)
blocks ← segment_area_beneath_panel(panel, 1)
for each block in blocks do
  while TRUE do
    line ← select_two_random_points(block)
    if is_almost_vertical(line) then
      inliers ← find_inliers(block, line)
      if inliers.count > threshold then
        poles.append(inliers)
      end if
    end if
  end while
end for
resultPoints ← union(poles)
pointCloud ← filter_pointCloud(pointCloud, resultPoints)
```

3.3 Evaluation

In order to properly test the implementation it is crucial to have an evaluation methodology which takes into consideration the limitations of this project. The

most important factor which will need to be defined is how the localization and map can and should be evaluated. One way would be to consider a ground truth point cloud or map of the environment which is tested. This would make it possible to take the generated point cloud of the world which is created by the SLAM, based on the LiDAR data of course, and utilize an algorithm such as ICP to match the points and then determine the Root Mean Square Error (RMSE). This however makes two rather large assumptions. Firstly that there exists a ground truth map or point cloud of the test site and secondly that this ground truth is of a significantly higher accuracy that the generated map will have. Since in our case none of these are fulfilled we can not use this method.

A different alternative would be to use the GNSS data which is recorded during testing and compare this to the estimated path of the SLAM. This assumes or GNSS is of sufficient accuracy so it can be used as an estimation of the ground truth path. Since in our case, the GNSS chip is capable of a low accuracy this was deemed to be an acceptable estimation of the SLAM trajectory. While this method will not specifically validate the *mapping* of the SLAM, in principle, it should not matter since they are closely linked, as cartographer uses the estimated trajectory to correctly place the LiDAR scans and create the map. The estimated trajectory from Cartographer was exported and under the assumption that the starting pose was known, it was compared to the GNSS trajectory. Both the total global error and translational error per second was computed.
Finally in the case that there is no or poor GNSS reception and thus the GNSS data can not be trusted, areal photos can be used to visually compare the generated map SLAM to those photos. This assumes that there exists areal photos of the solar park and that those are of high enough quality and accuracy to use as a reference. Unlike the previous method where the trajectory was compared to a GNSS reference, this method will also evaluate the mapping of the SLAM, but as previously mentioned, these are linked together and should not differ. Furthermore due to the way the photos are taken, only the edges of the panels and other features visible from above will be visible. This differs significantly from the map generated from the LiDAR point of view as it will manage to capture details such as the substructure and poles of the solar panels which is valuable information for example when planning a route. Therefore this method is mainly suitable as a quick and simple way to visually check the solution and make sure that it does not significantly differ from the areal photos, but in the context of this report

3.4 Work Area and Testing Methodology

The solar park setting presents some unique challenges that needed to be addressed, primarily the lack of robust and consistent GNSS connectivity. The internal testing, done by Husqvarna, has shown that the reception which is given to us as a status between 0 and 5 where 4 and above is good connection, is spotty to non existent when traversing underneath the solar panels due to them blocking the antennas view to the sky and thus the GNSS signals. However while the reception is poor underneath the panels, the reception is better in the corridors between two panel

rows and at the ends of the panels where the sky more visible. This will of course vary depending on the construction of each solar park and the exact behaviour will be determined by factors such as the distance between the panels, the width and height of the panels or the positioning of the satellites at a given time however an approximate visualization of the reception can be seen in Figure 3.3. Testing has shown that a GNSS status above 4 for at least 2 seconds can be considered good, thus it was deemed that periodically entering the areas in Figure 3.3 was not enough to obtain the aforementioned GNSS trajectory reference. Instead the robot stayed in those areas for the entire duration of the test run and the GNSS reception in the datasets were verified with a simple script which ensured the condition was fulfilled.

While most solar parks can be described as being built up by long, straight and monotone rows of panels, which traditionally is a especially challenging environment for SLAM, they do vary in certain aspects. The construction of the panels can vary significantly. Both in terms of the construction of the substructure, but also whether they are stationary or rotate to follow the position of the sun. Thus any solution must be either robust enough to handle these differences, or custom made for each park. In this case, due to limitations on the data, the thesis focused on creating a solution which was robust to changes in the *same* park.

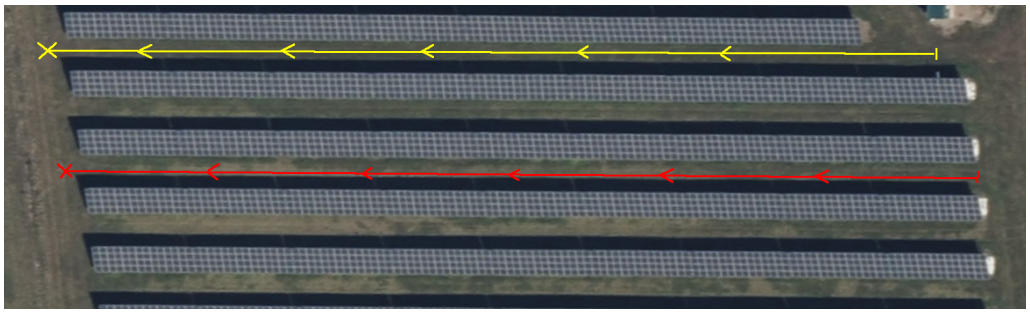


Figure 3.2: A satellite image from [23] showing the tested parts from the Ibstock solar park. The yellow line corresponds to the row which contained low vegetation while the red line corresponds to high vegetation. Note the short distance between these rows, which further implies the organic nature of solar plants.

At the time of testing Husqvarna was running tests at multiple sites across the world however in this project the data is recorded at a solar park in Ibstock, England. The park, which generates a peak power of nearly 5 MW, was commissioned in 2020 and according to [24] the panels are placed in rows going from west-east at a distance of 4-5 m and with a length of over 100m, which can be viewed in the satellite image in Figure 3.2. The data was recorded as the robot was manually driven at a velocity of slightly under 1 m/s between two rows of panels until the end was reached. Ideally in order to properly evaluate the effect of vegetation the same row should have been cut, followed by the recording of a second run with no vegetation present. Unfortunately due to time constraints, two different rows were chosen. The first featured very tall vegetation with certain parts reaching over one meter in height, which is substantially taller than the mower. The second row instead features very

low vegetation and certainly not taller than the mower. While ideally the same exact row for both test cases would have been used, it should be noted that the construction of the solar panels, length of the row, machine used for testing and time were all identical or similar in both cases. This should help mitigate any external factors, even if it is not ideal. The result of the data collection in the low vegetation case can be seen in Figure 3.4 which shows one of the LiDAR point clouds as the robot is driving between two panels. The point cloud when driving through high vegetation can be seen in Figure 3.5.

Figure 3.3: Satellite image from [23] where the areas within the red box have no GNSS reception.

It should also be noted that while there were more datasets from other solar parks available, these all lacked either good GNSS reception which would make the evaluation problematic, or variations in the vegetation. Thus rather then use them during the testing they mainly aided in verifying the SLAM and troubleshoot discrepancies during the tuning.

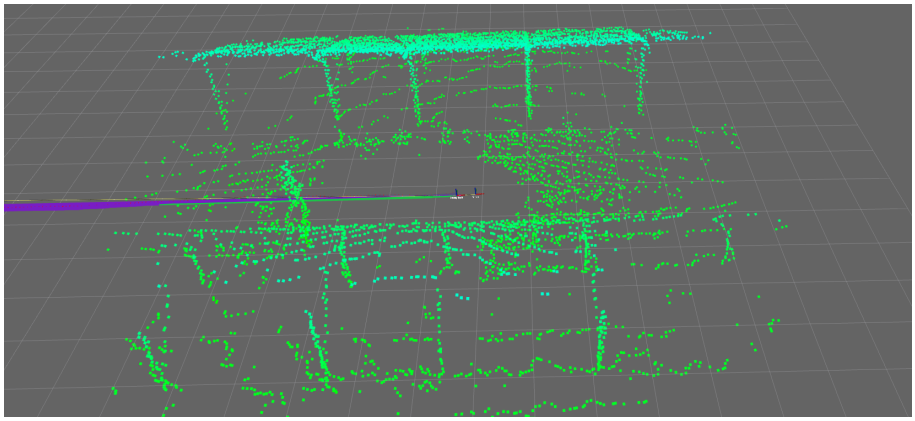


Figure 3.4: The point cloud that was recorded by the LiDAR between two panel rows where vegetation was low. We note the relative flatness of the ground and the that the panels on the far side is detected better due to it being nearly perpendicular to the light sent out by the LiDAR.

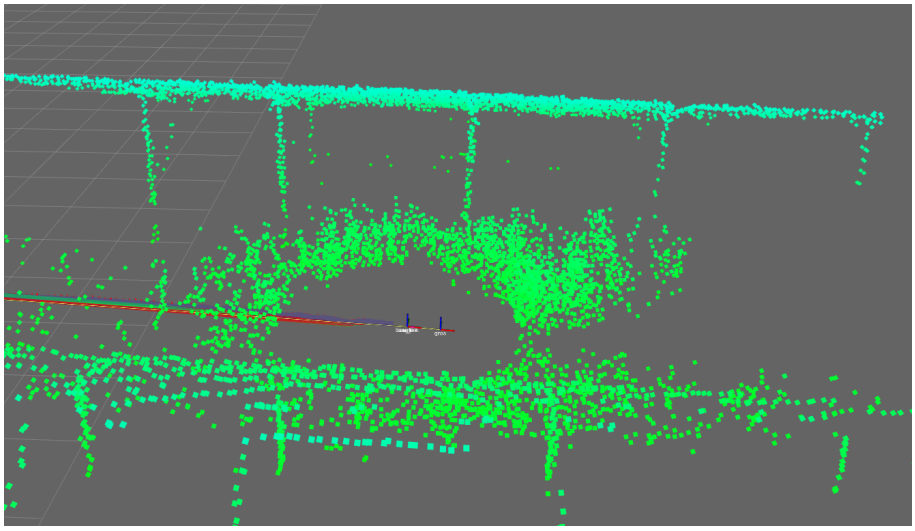


Figure 3.5: The point cloud that was recorded by the LiDAR between two panel rows where vegetation was high. We note the dense point cloud around the robot which is the vegetation, and the lack of points behind it. This is due to the vegetation blocking the view of the LiDAR.

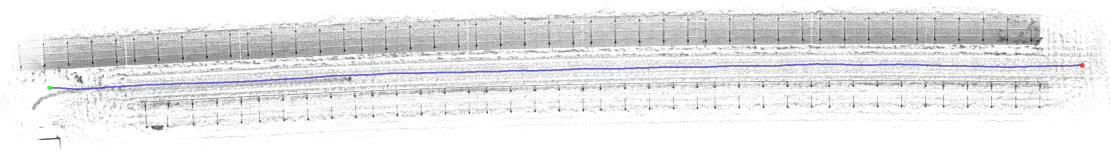
4

Results

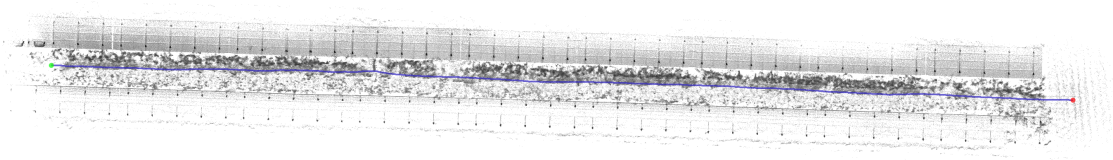
In this section the results will be presented both with and without the pre-filter. The generated maps from the low vegetation and high vegetation datasets will be compared and the accuracy of each respective solution will be presented. Furthermore the point clouds will be inspected and compared to evaluate the performance of the filters. Finally although it was not a focus for this thesis, some observation regarding the computational loads and performance will be presented.

4.1 SLAM without pre-filter

To get a baseline result, we first look at the maps generated by the raw input data. This data can be seen in Figure 3.4 3.5 and this will serve as a comparison to visually evaluate the different pre-filters. Thus they are omitted from this section as they will be identical since no filtering was done. From Figure 4.1, which shows the intensity map when the point cloud that is generated by the SLAM is projected onto the z-plane. This allows us to see a 'top-down' view of the panels while simultaneously observe any structure underneath them which would otherwise be hidden by the panels. We clearly see that there is an observable rotational drift in the map which was created with the low vegetation dataset. This drift is also present in the high vegetation case but to a much smaller degree. Furthermore the effects of vegetation in the input data is clearly visible and shows up as the darker areas close to the trajectory of the robot. The vegetation seems to clear up significantly under the panels, this could be for several reasons. Partly the fact that there is less sunlight reaching those areas which may affect the growth of the vegetation. A more likely scenario is that due to the LiDAR light not being able to penetrate surfaces, the vegetation closest to the sensor is blocking the view of the LiDAR causing it not to spot any from further behind.



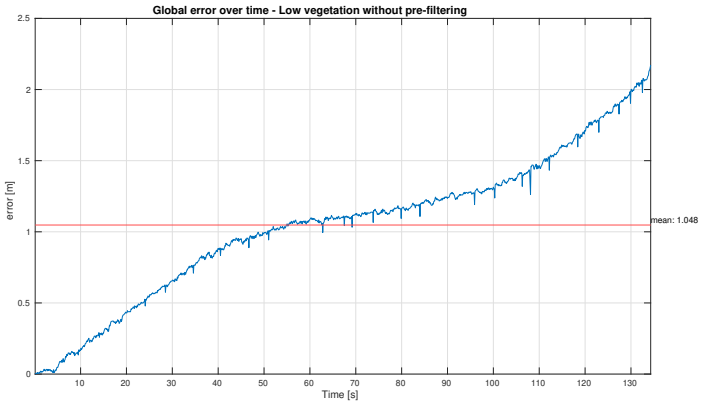
(a) In the low vegetation case we can clearly see the poles and panel outlines. While vegetation was low, it is still visible as a relatively constant gray noise. The darker line running parallel to the trajectory for the first quarter of the run is caused by an operator following the robot and then stopping.



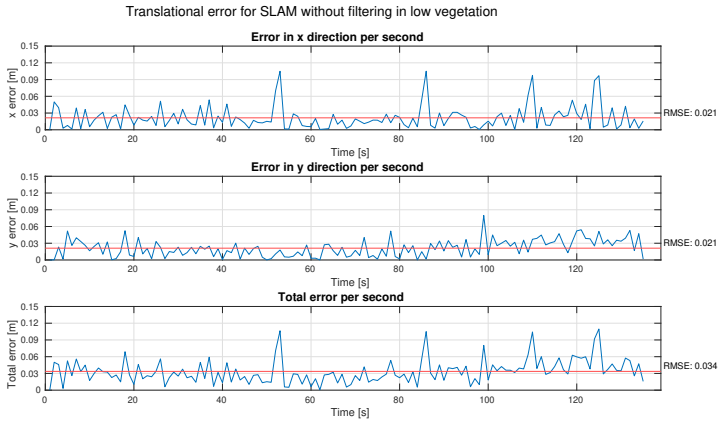
(b) In the high vegetation case, while the panels and poles are still visible, we also note a substantial amount of vegetation which in this case is represented by the dark clutter around the blue trajectory.

Figure 4.1: The generated intensity map when the point cloud is projected onto the z-plane without any pre-filtering for both low (a) and high (b) vegetation. The blue line is the trajectory as estimated by the SLAM where the green dot is the starting point and the red dot is the finish. We note that they both suffer from some level of rotational drift, although it is slightly less when vegetation is high.

When investigating the drift, or deviation from the reference GNSS trajectory, the estimated trajectory in both the low vegetation dataset, which can be seen in Figure 4.2, and the high vegetation dataset, which can be seen in Figure 4.3, it is evident that they are both increasing. This means that the SLAM is drifting away from the true trajectory throughout the run. Furthermore we can also see that, while very close, the dataset with high vegetation performs slightly better in this regard. This is unexpected as the LiDAR has a much better view of the static structures in the low vegetation datasets and not blocked by the stochastic and non-stationary vegetation. Thus we expect it to be more accurate but our numbers would indicate otherwise. From the previous Figure 4.1 we concluded that the rotational drift was larger in the dataset with low vegetation and this can also be seen in Figure 4.3 and 4.2 as the translational error per second in the y direction (perpendicular to the panel which is where the effect of rotational drift will be the greatest) is also 1 cm less. Finally it was noted that the computational load was greater with the high vegetation dataset, although both managed to run at real-time speeds on a commercial computer.

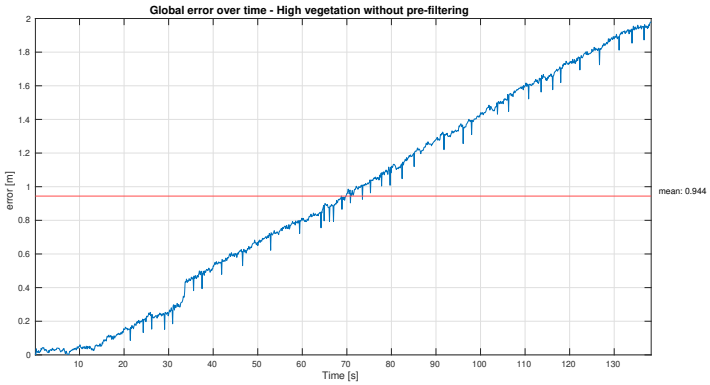


(a) We can see that we have a mean total error of 1.048 m and a total error at the end of the run at just over 2 m. This means that after traversing approximately 130 m the final position will deviate just over 2 m from the reference.

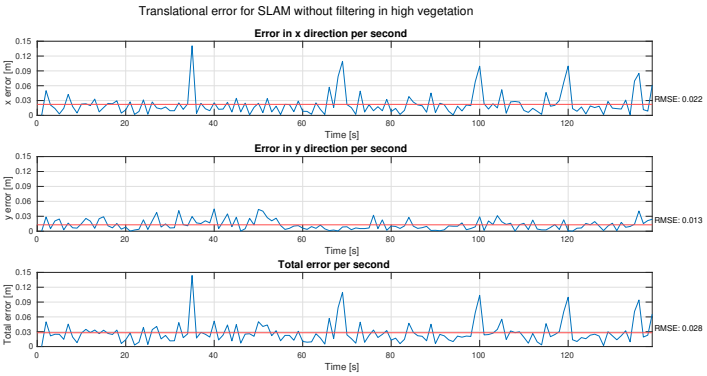


(b) We note a relatively constant error of similar size in the x and y direction and a total RMSE of 0.034 m per second.

Figure 4.2: The total global error (a) of the estimated trajectory when compared to the GNSS reference and the relative translational error per second (b) in the low vegetation dataset.



(a) We can see that we have a mean total error of 0.944 m and a total error at the end of the run at just under 2 m. This means that the final deviation will be just under 2 m after traversing approximately 130 m.



(b) We note a relatively constant error of similar size in the x and y direction and a total RMSE of 0.028 m per second.

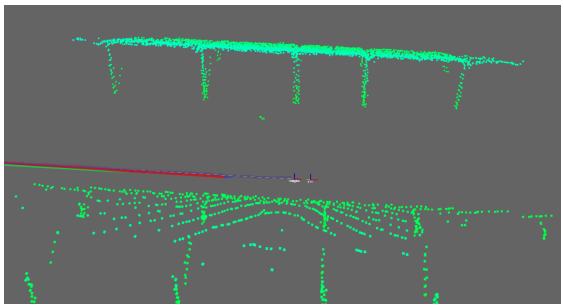
Figure 4.3: The total global error (a) of the estimated trajectory when compared to the GNSS reference and the relative translational error per second (b) in the high vegetation dataset.

4.2 SLAM with pre-filter

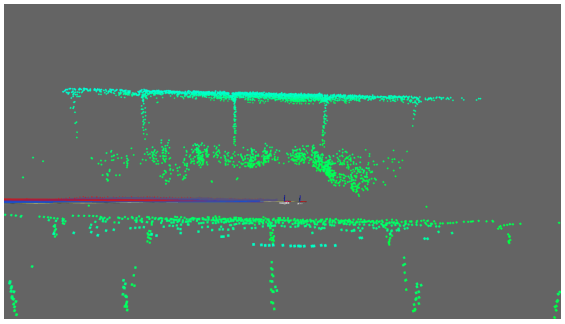
Now we will consider the results when adding a pre-filter to the input data. The filters will be evaluated based on how well they manage to remove vegetation from the datasets and this will be done mainly visually. Furthermore they will be evaluated based on how the accuracy of the trajectory is affected by the pre-filter. First we will consider the most trivial filter which was simply a minimum height restriction on the input data. Then we will look at the RANSAC ground filter and finally the RANSAC pole and panel filter. As previously mentioned the filtering step as well as the SLAM was run offline and thus the computational load will not affect our results in any way.

4.2.1 Minimum Height Filter

When we simply look at the point cloud after the filtering step which can be seen in Figure 4.4 and compare it to the unfiltered case in Figure **??** we can clearly see that while it manages to remove the ground in the low vegetation data set, due to the nondiscriminatory nature, it also remove a considerable part of the longer poles and nearly the entire shorter pole. Furthermore it fails at removing all the vegetation from the dataset with high vegetation due to some of it being taller than the threshold of 50 cm. In the generated maps in Figure 4.5 this is clearly visible with the low vegetation case having substantially less noise, even if it is also reduced for the high vegetation case.



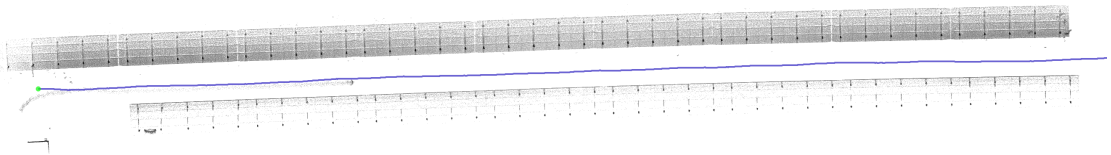
(a) In the low vegetation dataset while the ground and vegetation is successfully removed, so is a portion of panel substructure, especially the shorter poles.



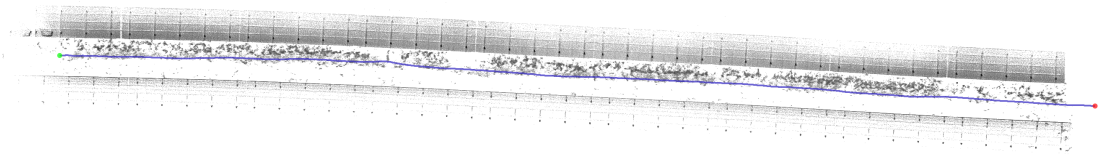
(b) In the high vegetation dataset we note that while the vegetation is removed, a substantial amount is left in place due to their height being over the threshold.

Figure 4.4: The point clouds that are inputed into the SLAM algorithm after filtering with a minimum height filter for low (a) and high (b) vegetation datasets.

When investigating the accuracy for this method we get some differing results. On the one hand, from Figure 4.6 which shows the drift and error from the low vegetation case, we can see that the error is reduced compared to the non-filtered case. While the difference is not particularly large, 0.886 m of mean global error compared to 1.048 m it is measurable and matches the reduced rotational drift we see in 4.5. On the other hand, comparing the results for the high vegetation case in plot 4.7 and comparing it to the non filtered approach we note that the mean error has instead increased from 0.944 m to 1.027 m. It was also noted that the computational load of this filter was quite low and that the performance of the SLAM improved when the input data was filtered.

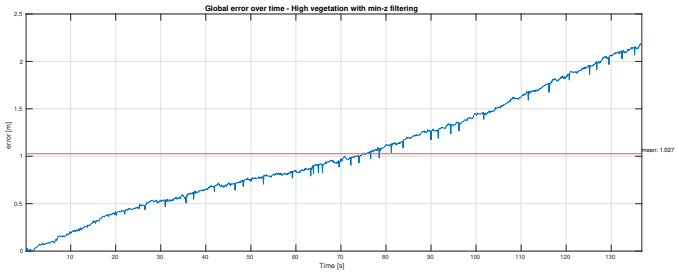


(a) In the low vegetation case we can clearly see the poles and panel outlines. Furthermore the filter manages to remove the ground from the unfiltered case. Furthermore a marginally reduced rotational drift is noted.

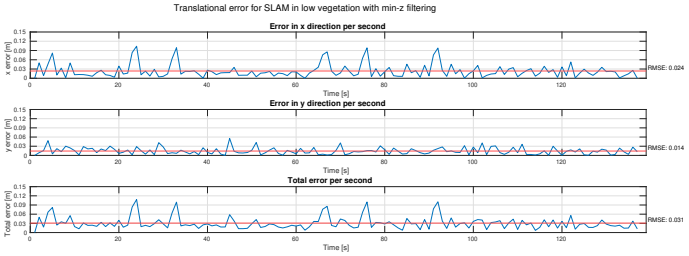


(b) In the high vegetation case, while the panels and poles are still visible, we still note that not all vegetation is successfully removed.

Figure 4.5: The generated intensity map when the point cloud is projected onto the z-plane with the minimum height filter for both low (a) and high (b) vegetation. The blue line is the trajectory as estimated by the SLAM where the green dot is the starting point and the red dot is the finish.

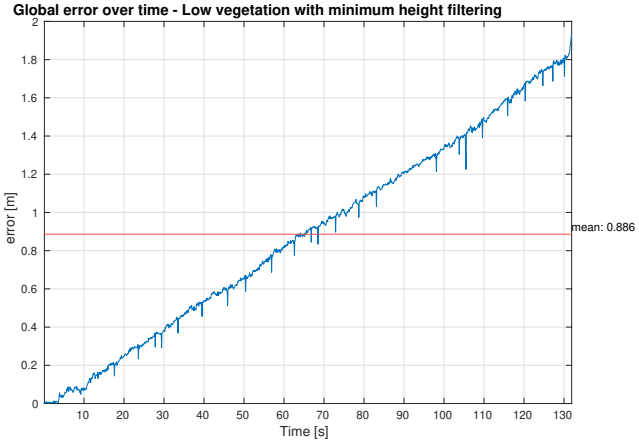


(a) We can see that we have a mean total error of 1.027 m and a total error at the end of the run at over 2 m. This is worse compared to the non filtered case.

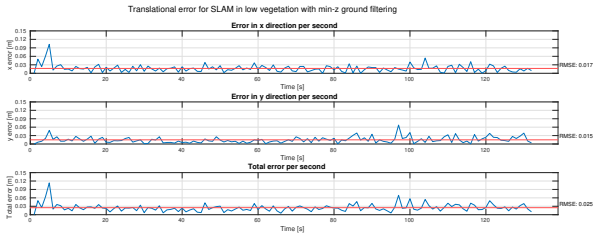


(b) We note a relatively constant error of similar size in the x and y direction and a total RMSE of 0.031 m per second which slightly worse compared to the no filter approach.

Figure 4.7: The total global error (a) of the estimated trajectory when compared to the GNSS reference and the relative translational error per second (b) in the high vegetation dataset with a minimum height filter applied.



(a) We can see that we have a mean total error of 0.886 m and a total error at the end of the run at just under 2 m. This is a slight improvement over the non filtered case.



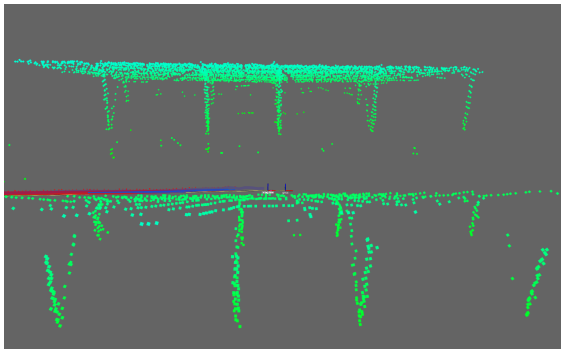
(b) We note a relatively constant error of similar size in the x and y direction and a total RMSE of 0.025 m per second which is an improvement over the no filter case.

Figure 4.6: The total global error (a) of the estimated trajectory when compared to the GNSS reference and the relative translational error per second (b) in the low vegetation dataset with a minimum height filter applied.

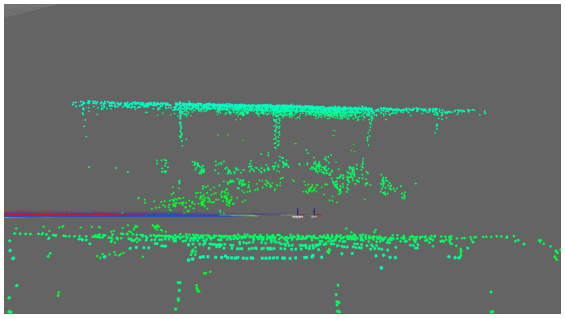
4.2.2 RANSAC Ground Filter

If we now consider the point clouds after the RANSAC ground filtering step, which can be seen in Figure 4.8, and compare it to the unfiltered case in Figure **??** we can clearly see that it manages to remove a substantial amount of the ground in the low vegetation data set while leaving most of the poles and substructure alone. Only very few outliers are present and overall the result when evaluated visually can be described as ideal. The high vegetation dataset proves harder for this filter as similarly to the previous filter, it struggles to deal with a ground plane that varies significantly in height due to for example vegetation and thus the results are nearly identical to the minimum height filter. Looking at the generated maps in Figure 4.9 we note results that are very similar to Figure 4.5, excellent map with little to no noise or clutter in the low vegetation dataset and some vegetation still present with the high vegetation dataset.

The accuracy also followed a similar pattern to the minimum height filter with the low vegetation dataset performing slightly better and the high vegetation slightly worse. This is expected as the only difference in the output point clouds that the two filters produced where poles that came slightly closer to the ground in the low vegetation dataset. The reader is thus referred to Figure 4.7 and 4.6 for accuracy numbers. Finally the computational load of this filter was quite high however similarly to the previous case the performance of the SLAM improved when the input data was filtered.

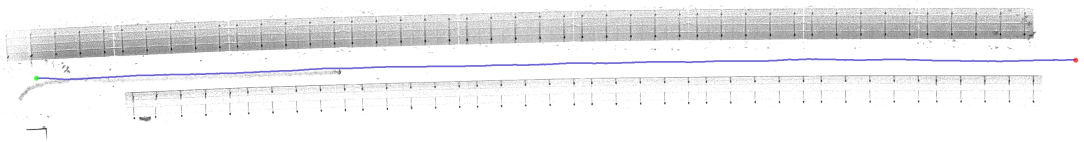


(a) In the low vegetation dataset the ground and vegetation is successfully removed with only a few outliers remaining. Furthermore the poles are left mostly intact with only the lower parts of them being removed with the ground.

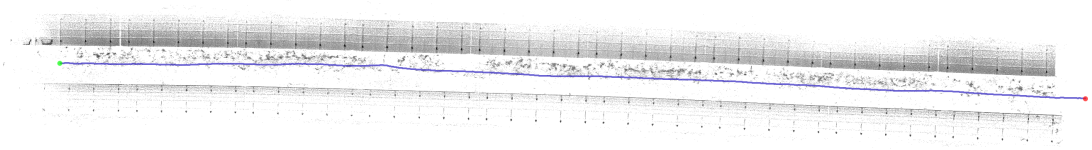


(b) In the high vegetation dataset we note a very similar result to Figure 4.4 where the RANSAC filter fails to remove a substantual part of the vegetation, and still removes a big part of the poles.

Figure 4.8: The point clouds that are used as input for the SLAM algorithm after filtering with a RANSAC ground detecting filter for low (a) and high (b) vegetation datasets.



(a) In the low vegetation case we can clearly see the poles and panel outlines. Furthermore the filter manages to remove the ground from the unfiltered case. Similarly to the minimum height filter, the results with this filter is ideal.



(b) In the high vegetation case, while the panels and poles are still visible, we still note that not all vegetation is successfully removed.

Figure 4.9: The generated intensity map when the point cloud is projected onto the z-plane with the RANSAC ground filter for both low (a) and high (b) vegetation. The blue line is the trajectory as estimated by the SLAM where the green dot is the starting point and the red dot is the finish.

4.2.3 RANSAC Panel and Pole Filter

Finally we look at the RANSAC panel and pole filter which is the first one to significantly differ from the previous filters in that we no longer try to detect the ground and vegetation but rather as the name implies the panel and poles. Since the low vegetation dataset already achieved an ideal filtering result with the RANSAC ground filter, this filter was implemented and tested only on the high vegetation dataset which both previous filters struggled with. Furthermore due to the limitations in using a RANSAC based filter, the right side row of solar panels could not be reliably detected with the current algorithm. In the cases that it did get detected, the majority of the vegetation would also be included. This is due to the panels being angled and thus the plane which is used by RANSAC to estimate the panels extends down into the ground which leads to a significant amount of vegetation being included. Therefore it was decided to only detect one of the panel rows so that a fair and realistic evaluation of the filter performance can be done. A point cloud representation of the filters performance can be seen in Figure 4.10. It is clear that aside from the issues mentioned, it manages to successfully detect both the panel and the poles. Some vegetation makes it through and the shorter poles are not always detected, possibly due to the view of the LiDAR being blocked by vegetation, however nearly all of the vegetation around the robot and between the panels is completely removed.

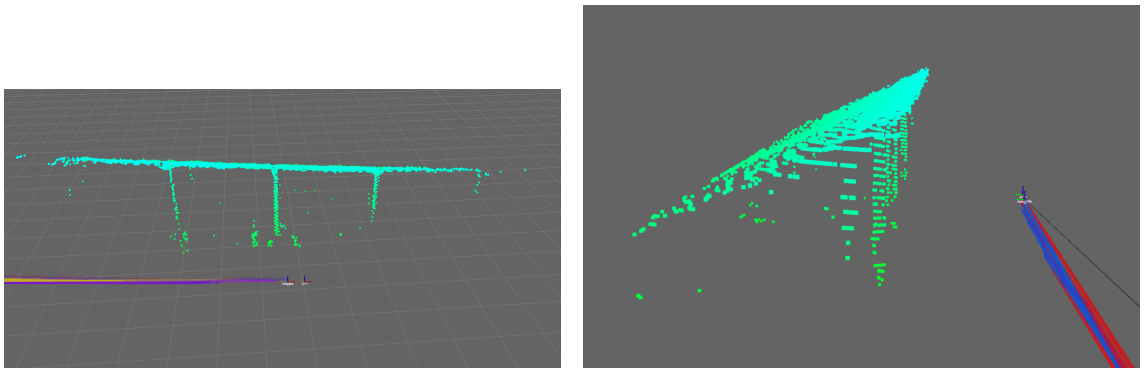


Figure 4.10: The point clouds that are used as input for the SLAM algorithm after filtering with a RANSAC based panel and pole detector. Both images represent the same dataset and filter, only from different angles. We note that the filter manages to capture both the panel and most of the poles correctly. Some vegetation is still present, however most has been removed. We also note that the filter only detects a single row of solar panels.

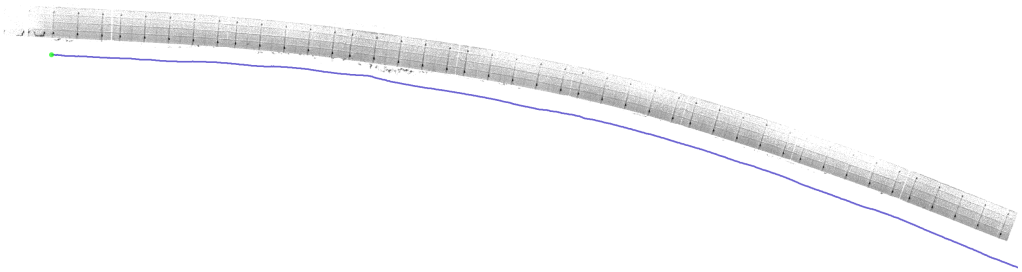
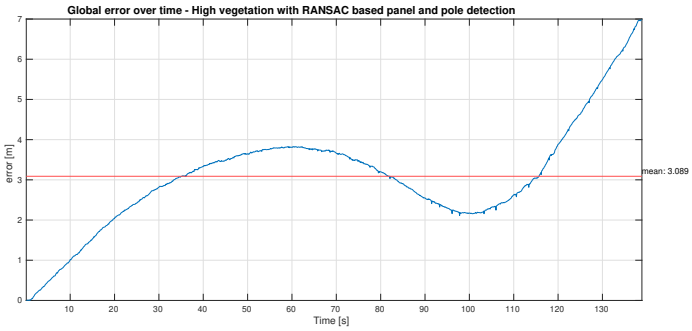
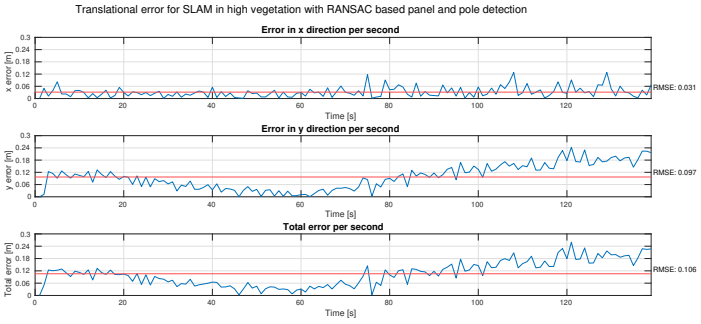


Figure 4.11: The generated intensity map when the point cloud is projected onto the z-plane with the RANSAC panel and pole filter. The blue line is the trajectory as estimated by the SLAM where the green dot is the starting point and the red dot is the finish. We note that while the vegetation is nearly completely gone, one panel row is missing and there is a substantial increase in rotational drift.

The accuracy of this filter follows the patterns seen in the previous filters when implemented on the high vegetation dataset. When removing the vegetation, we simultaneously note an increase in rotational and translational drift. In the case of this final filter, which not only removes nearly all vegetation but also one of the panel rows, it is clear that the drift increases substantially as a result of this. The rotational drift can be seen in Figure 4.11 which while being devoid of noise, also demonstrates a clear curve. The global and translational error also point to this. From Figure 4.12 we deduce that by every measurable metric removing the vegetation resulted in an increase in drift. In this case an average error of 3.089 m was noted which leads to the final point in the estimated trajectory to deviate by over 7 m.



(a) We can see that we have a mean error of 3.089 m and a total error at the end of the run at over 7 m. This is significantly worse compared to the other filters and the unfiltered case.



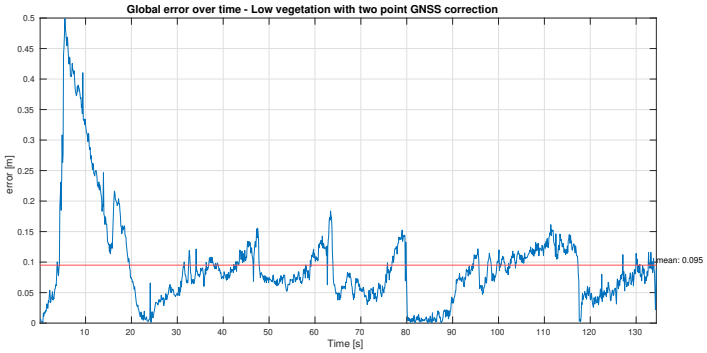
(b) We note a total RMSE of 0.106 m per second which significantly worse compared to the no filter approach. This would indicate that for every second passed, the estimated trajectory differs by 10 cm compared to the reference.

Figure 4.12: The total global error (a) of the estimated trajectory when compared to the GNSS reference and the relative translational error per second (b) in the high vegetation dataset with a RANSAC panel and pole filter applied.

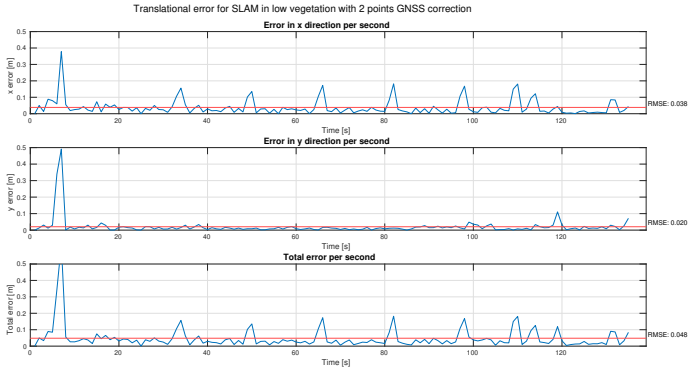
4.3 GNSS correction

Finally the results of simulating GNSS reception at the edges of the panel will be presented. Since the input data, which is the unfiltered variant, will also remain unfiltered the generated maps will not be posted. The implementation was tested on the low vegetation dataset as it is the dataset which previously had the largest deviation from the GNSS-reference and thus the highest drift. From Figure 4.13 we conclude that this method yielded in an average error of 0.095 m which is a considerable improvement over the previous approaches. The rotational drift and translational drift is thus reduced to manageable levels with very limited GNSS reception simulated. It should be noted that the large drift we previously encountered can only be corrected when the robot exits the GNSS blocked regions and regains connection at the end of the panels, at which point a optimization cycle will correct it. Thus, before this occurs the drift is still significant for over 100 s. Furthermore in

the translational error in Figure 4.13 it is noted that the error remains more or less unchanged, which is not necessarily unexpected since the error is no longer being continuously integrated but rather shows up as some noise, it also contains some peaks. These peaks corresponds well with the number of submaps created during the run and since Cartographer assumes there is no drift within a given submap and therefore only adjust the submaps as rigid units during the global optimization, these peaks likely stem from this operation.



(a) The global error after correcting the trajectory with GNSS at the edges of the solar panel rows yields a much lower mean error of 0.095 m. The error is mostly at or around 10 cm with a spike in the beginning which likely stems from cartographer trying to correct a large error over a fairly long distance and failing to perfectly align some of the submaps.



(b) We note a total RMSE of 0.048 m which is in line with previous results. We also note that there are some spikes which align with the edges of the submaps.

Figure 4.13: The total global error (a) of the estimated trajectory when compared to the GNSS reference and the relative translational error per second (b) in the low vegetation dataset and with GNSS correction.

5
Discussion

In this chapter, some of the findings in this project will be further explained and some conclusions regarding the filters and SLAM that were investigated in this report will be presented. Furthermore some suggestions for improvements and future works will be detailed.

It is clear that solar parks represents a challenging environment in regards to the SLAM problem and autonomous operation in general. While even the unfiltered managed to create maps of decent quality, and a rough localization in the solar parks, the assumption of GNSS-blackout does greatly impact the positional accuracy of said solution. While the average translational error matches other implementation, we suffered greatly from an continuously increasing drift, both translational and rotational. A reason for this may be due to the geometry of solar parks. In general, long and narrow corridors with few geometric features, which is a description that fits the solar park environment well, is challenging for any SLAM and considering the geometries of solar parks they fit this description. While the angled solar panels provides a good surface for the LiDAR to detect and the SLAM to maintain the position in the y-axis, or perpendicular to the solar panel rows, the x-axis parallel to the panels and also the direction of traverse is only constrained by the poles. This means that while the distance from the robot to the panels will be relatively accurate simply due to the shear amount of points that define the structure, any translation parallel to these must rely on the scan matcher to correctly, accurately and consistently align the different LiDAR scans based on only some thin metal poles.

Interestingly the hypothesis that accuracy could be improved by removing the vegetation from the LiDAR data turned out not to be correct. While it did marginally improve the accuracy of the low vegetation dataset, it did not significantly improve and in other datasets it even reduced accuracy and increased translational and rotational drift. One theory as to why this could be is that by removing the vegetation inadvertently we have also removed a large portion of the geometric features that the scan matcher used to create the trajectory. This would explain why the low vegetation dataset did not behave similar to the high vegetation dataset when removing the ground, since it did not contain much vegetation to start with and the ground could thus be described as close to an ideal plane which does not contain any geometric features. It would also explain why the high vegetation dataset provided a better accuracy even before the filtering was done.

It should also be noted that while the filters did not noticeably improve the localization, the custom RANSAC based panel and pole detecting algorithm was successful in filtering out nearly all vegetation from a very challenging dataset. This greatly increases the usability of the maps and makes further post processing much more feasible. For example, simple image processing could be implemented to obtain the position of the poles or panels. Furthermore while the vegetation in some cases increased accuracy, it does not mean that leaving them in place is the best action. In many scenarios the SLAM parameters that were required to obtain the best solution in one dataset, would fail completely in another. Considering the fact that the low and high vegetation datasets were recorded just a few rows apart, as seen in Figure 3.2, the surroundings in a solar park vary quickly and thus there is a need for homogeneous input data that is robust to differences in the surroundings. Furthermore if mapping is to be seriously considered, the maps created need to closely resemble the LiDAR scans at different times of the year and different levels of vegetation, although long-term SLAM algorithms may mitigate this somewhat [25]. Moreover while computational load was not a focus for this thesis, it was noted that while all filters improved the performance of the SLAM due to it having to process fewer points, both RANSAC based filters where running far slower than real time and much optimization will be needed before these can finish the computations within 100 ms (which is the limit if a 10 Hz lidar is used) and run online on a robot.

Finally while the filters did not improve the accuracy of the localization, adding GNSS correction did substantially improve it.This of course somewhat breaks our initial goal of creating a GNSS independent SLAM system, the integration of periodic GNSS corrections demonstrates a pragmatic approach to enhancing localization accuracy in challenging environments like solar parks. The addition of GNSS data, even if sporadic, provides anchor points that significantly reduce the cumulative drift inherent in pure SLAM systems, particularly over extended operations. Since it can only correct drift after exiting the areas of poor reception, which often means that the robot will travel very far before this happens, the robot can still deviate heavily from a planned trajectory and possibly crashing before it gets a chance to correct the drift. One suggestion would be to drive in a pattern that is perpendicular to the direction of the solar panels rather than parallel to them. This would both aid in making the largest geometric feature perpendicular to the direction of traverse which should improve the localization, and ensure that the robot avoids prolonged GNSS blackouts. It should be noted that this assumes the robot is lower than the lower edge of the panels at *all* times and that there is sufficient space between the panels for the GNSS to regain connection. If hardware solutions are available a compass would help give a noisy estimation of the heading which *does not* drift, but on the other hand is sensitive to electromagnetic disturbance which may be an issue on a robot containing several electric motors.

5.1 Conclusion

In summary, this thesis explores autonomous navigation in solar parks using a LiDAR-based Graph-SLAM approach with an added pre-filter for data processing

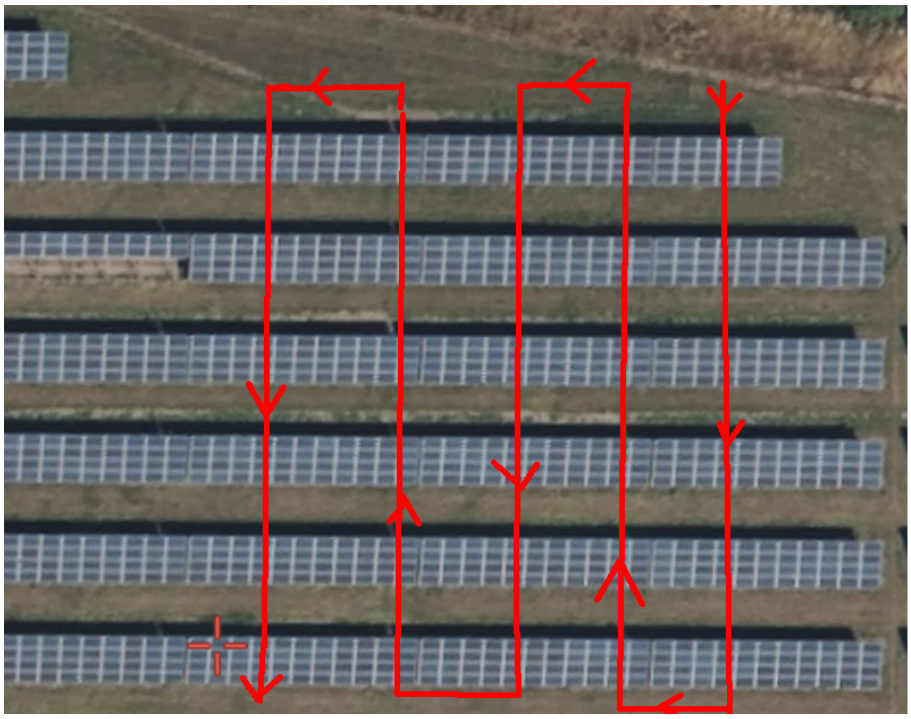


Figure 5.1: An alternative way of driving through the solar park to avoid prolonged GNSS blackouts.

and vegetation control. The method manages to map and track the autonomous mower's path relatively well however some drift is present. The pre-filter is successful in removing vegetation from the data, improving the quality of the maps while not noticeably improving accuracy. In this thesis, the sole method for correcting excessive drift relies on incorporating GNSS data. Future work could include refining the algorithm and incorporating additional sensors to enhance performance and reducing the need for satellite navigation. Overall, this study demonstrates the potential of using a LiDAR-based Graph-SLAM for navigation in solar parks, while also highlighting areas for further development.

Bibliography

[1] IEA. "Solar pv." (2022), [Online]. Available: https://www.iea.org/reports/solar-pv.

[2] H. AB. "Pilot project for electrified vegetation maintenance in solar parks receives government funding." (Feb. 2022), [Online]. Available: https://www.husqvarnagroup.com/en/press/pilot-project-electrified-vegetation-maintenance-solar-parks-receives-government-funding.

[3] A. Khairuddin, S. Talib, and H. Haron, "Review on simultaneous localization and mapping (slam)," Nov. 2015, pp. 85–90. DOI: 10.1109/ICCSCE.2015.7482163.

[4] F. Hidalgo and T. Bräunl, "Review of underwater slam techniques," in *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, 2015, pp. 306–311. DOI: 10.1109/ICARA.2015.7081165.

[5] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: 10.1109/MRA.2006.1638022.

[6] X. Yue, Y. Zhang, and M. He, *Lidar-based slam for robotic mapping: State of the art and new frontiers*, 2023. arXiv: 2311.00276 [cs.RO].

[7] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, Accessed: yyyy-mm-dd, Stanford University, 2009. [Online]. Available: http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf.

[8] "What is lidar?" [Online; accessed 11-March-2024]. (Accessed: 2024), [Online]. Available: https://www.synopsys.com/glossary/what-is-lidar.html.

[9] A. Name. "Why bulky spinning lidar sensors might be around for another decade." (2018), [Online]. Available: https://arstechnica.com/cars/2018/05/why-bulky-spinning-lidar-sensors-might-be-around-for-another-decade/.

[10] "What is point cloud?" (2024), [Online]. Available: https://hexagon.com/company/careers/capability-centre-india/intelligence-of-things/what-is-point-cloud (visited on 03/11/2024).

[11] Cartographer Project Contributors, *High level system overview*, Image in GitHub repository, Accessed: 2024-04-26, 2017. [Online]. Available: https://github.com/cartographer-project/cartographer/blob/master/docs/source/high_level_system_overview.png.

[12] Y. Liu, *Title of the document*, Master's thesis, Linköping University, Available at DiVA portal, 2020. [Online]. Available: https://liu.diva-portal.org/smash/get/diva2:1384360/FULLTEXT01.pdf.

[13] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1271–1278. [Online]. Available: https://research.google/pubs/pub45466/.

[14] K. Derpanis, *The ransac algorithm*, http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf, 2010.

[15] R. electric, *Raymo electric mower*, Online; accessed Feb 14, 2024, 2024. [Online]. Available: https://raymoelectric.se/.

[16] Ouster, *Os1 lidar sensor datasheet*, 2021.

[17] u-blox, *Zed-f9p high precision gnss module data sheet*, Accessed: yyyy-mm-dd, 2021. [Online]. Available: https://content.u-blox.com/sites/default/files/ZED-F9P-04B_DataSheet_UBX-21044850.pdf.

[18] Google. "Compiling cartographer ros." (2022), [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/compilation.html.

[19] A. Koval, C. Kanellakis, and G. Nikolakopoulos, "Evaluation of lidar-based 3d slam algorithms in subt environment," *IFAC-PapersOnLine*, vol. 55, no. 38, pp. 126–131, 2022, 13th IFAC Symposium on Robot Control SYROCO 2022, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2023.01.144. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896323001519.

[20] K. Krinkin, A. Filatov, A. Filatov, A. Huletski, and D. Kartashov, "Evaluation of modern laser based indoor slam algorithms," vol. 426, May 2018, pp. 101–106. DOI: 10.23919/FRUCT.2018.8468263.

[21] A. Pålsson and M. Smedberg, "Investigating simultaneous localization and mapping for agv systems," M.S. thesis, Chalmers university of Technology, 2017.

[22] Open3D Team, *Pointcloud – open3d 0.16.0 documentation*, Open3D Documentation, 2023. [Online]. Available: https://www.open3d.org/docs/release/python_api/open3d.geometry.PointCloud.html.

[23] A. Inc., *Satellite map*, Satellite image available through satellites.pro, Accessed on: 2023-03-25, via [https://satellites.pro/UK_map], 2023.

[24] L. BP, *Solar ppa lays foundation for future projects with ibstock brick*, online, available at: https://lightsourcebp.com/uk/news/lightsource-bp-breaks-ground-on-uks-first-brick-factory-powered-by-solar/, 2019.

[25] N. D. Carlevaris-Bianco, "Long-term simultaneous localization and mapping in dynamic environments," Ph.D. dissertation, The University of Michigan, 2015.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY