Tampere University

Miikka Mäki

LIDAR SLAM-MAPPING AS A POTENTIAL POWERLINE MAINTENANCE TOOL

Faculty of Information Technology and Communication Sciences
M. Sc. Thesis
March 2022

ABSTRACT

Miikka Mäki: Lidar SLAM-mapping as a potential powerline maintenance tool
M.Sc. Thesis
Tampere University
Master's Degree Programme in
March 2022

The purpose of this master's thesis is to determine whether it is possible to utilize non-conventional sensory systems for power line inspection during times that more conventional methods are severely restricted. To achieve this, we will utilize lidar-generated point clouds in conjunction with measurement data from inertial measurement units to create a geo-referenced set of point clouds to generate a map of the point-of-interest area. This will be achieved by conjoining raw lidar-data from a lidar sensor and using the data provided from the inertial measurement unit to merge millions of points into a one cohesive map. We will use Robot Operating System to achieve the evaluation, fusion and integration of the different streams of data. We will also use Google Cartographer to aid us in the SLAM-mapping of the different sensory data sources. Once a SLAM-mapped point cloud is generated, we can evaluate the accuracy of the data and the possibilities of the generated data to be used as a maintenance tool to assist in detecting and solving various problems that many electrical companies in rural Finland face during their daily business. Such problems are snapped power lines or excess object blocking the power lines. We will want to determine if a system like this could be used when it is impossible for cameras or the naked eye of a human to detect this kind of faults, for example during the night or during a storm.

Key words: LiDAR, IMU, INS, ROS, SLAM

Contents

List of keywords

LiDAR – Light Detection and Ranging
IMU – Inertial Measurement Unit
INS – Inertial Navigation System
GPS – Global Positioning System
ROS – Robot Operating System
FOV – Field of View
SLAM – Simultaneous Localization and Mapping
PLC – Point Cloud Library
MEMS – Microelectromechanical System
UAV – Unmanned Aerial Vehicle
AI – Artificial Intelligence
ToF – Time of Flight
VLP-16 – Velodyne Lidar Puck, 16-channel 3D-lidar
EKF – Extended Kalman filter
AHRS – Attitude and Heading Reference System

1. Introduction

Modern people have grown increasingly dependent on electricity. The flow of our daily lives is greatly impacted by the functioning of various electrical systems more than ever before in our history: our homes are covered with a myriad of different electrical devices that we have grown to take for granted. In addition, amidst the increasing trend of work-from-home-mentality, our livelihood and ability to work can be a great detriment if the access to electricity is severely limited. The climate of this electricity-dependent lifestyle has set increasingly demanding requirements to electricity provider companies that provide us the power for our growing electrical needs. The increasingly digitalized population requires reliable upkeep of power to properly function.

Power outages can be a significant setback for electricity providers that operate in harsh and sparsely populated rural areas and potentially cost millions in reparation costs. Customer satisfaction is a highly valued asset in the competitive energy industry, so it is critical to efficiently manage the fault-tolerance of the power line infrastructure, and in case of a failure, quickly determine the specific location of the problem and take swift reparative action. Damage assessment is also important so that the duration of the power outage can be minimized and that the expected power outage duration can be properly informed to the affected and inconvenienced customer base. As a countermeasure to failure situations, electrical companies utilize manned aerial reconnaissance to tackle the possible large-scale power outage events that usually are triggered by excessively stormy or snowy weather. Rough weather can cause unexpectedly large amount of damage to the power line infrastructure. Most common failure events are usually snapped power lines, trees that have fallen on power lines or medium-to large-sized tree branches that the storm has hurled on delicate instruments, like high-voltage transformers. A problem arises with the fact that manned aerial surveillance can only be done during suitable weather. As the aerial reconnaissance is based on visual detection, the capability for a human eye to detect problem areas can be severely restricted by a multitude of environmental factors: fog, dense rain, snow or darkness are all limiting factors for humans to effectively detect potential faulty areas, even with the aid of advanced camera technology. Therefore, interest to find a cost-effective solution has risen among multiple energy provider companies to provide a solution that could both be operated independently and detect and evaluate the affected environment without weather-factoring limitations through accurate three-dimensional representations of the environment.

One potential candidate to solve this problem is a Light Detection and Ranging (LiDAR) sensor in combination with 3D-environment mapping algorithms. A LiDAR sensor sends channels of lasers in an arc and receives laser bounces, which contain point

position and distance data. The lidar can generate a three-dimensional point-cloud from the collected data points to build a 3D-representation of the observed area [Murphy 2004]. These attributes make lidar a suitable mapping alternative for environments in which visibility is severely restricted. A lidar can also be attached and operated with a UAV-drone attached with a suitable single-board computer that handles the sensory output data collection [Khan 2017]. A drone enables unmanned surveillance missions, that can be operated within a short response-time, can be partly or completely automated, and provide enough operating range to effectively scout wide problem areas [UAVS 2010]. This would also decrease or completely nullify the need for expensive and laborious manned aerial surveillance missions.

Recently, small-scale technology has taken major advancements forward that have enabled alternative sensing platforms that provide cost-effective and high-resolution data to be more widely used in 3D-data capture and environment surveying. Lidar-attached drones have already been operational in answering this specific problem, provided by companies that focus on lidar and drone technology [Hepta 2021]. Usually, this kind of service is based on automated flight paths and lidar-data collection that is observed by a lidar-operator/observer in real time. By continuously observing the data stream sent by the drone, the operator can determine by the point-cloud data whether the powerlines under observation are damaged or not. By automating this process with 3D-model detection methods, the solution grows increasingly scalable by eliminating the need for a drone operator.

The purpose of this thesis is to determine whether it would be possible to use cost-effective lidar sensors to generate and combine lidar-generated point clouds into a one, coherent map representation during the flight, and possibly use the generated map to achieve AI-based object detection. To achieve this, we need to know the exact position of our lidar sensor to effectively merge point clouds that are being generated during flight. An Inertial Measurement Unit (IMU) will be used to achieve this. The IMU provides us with acceleration, roll, pitch and yaw data of the drone [Murphy 2004]. The lidar and IMU data can be merged by using Simultaneous Localization and Mapping (SLAM) algorithms [Qian et al. 2017, Droeschel and Behnke 2018, Tang et al. 2015].

Using the SLAM-mapped point clouds, we can study their density, cohesiveness, odometry accuracy, spatial correctness, metric representation and overall potential for automated object detection. We can also evaluate the computational requirements of the process and assess the possible hardware requirements. These parameters are important, because our goal is to provide a virtual, three-dimensional representation of the world as accurately as possible. The accuracy is necessary for knowing which kind of sensory systems are needed to achieve accurate-enough representations of objects that are narrow, small, thin, or otherwise hard to detect using lidar-sensory systems (e.g., power lines)

[Qian et al. 2017]. The study can reveal the requirements and the generated output usefulness of the system. If the system shows promise and shows credibility as a proof-of-concept-project, we can evaluate the possible next steps to continue developing a fully operational, production-grade product.

Chapter 2 will mainly handle the required sensory technology and the theory behind their operation. Chapter 3 introduces and explains the theory and practical applications of simultaneous localization and mapping. Chapter 4 deals with the integration between our sensory hardware, and localization and mapping software. Chapter 5 will be dedicated to the evaluation of the research findings. Finally, chapter 6 presents the research conclusions.

2. Sensory technology

In order to get accurate representations of the world, we must use suitable sensory technology to suit our needs. There are many options available, which are narrowed down by the specific use case and restrictions of the environment to be recorded. When we want to achieve range sensing, there is generally two technologies to choose from: *triangulation* and *time-of-flight.* These two main technological categories have underlying variations, each with their own strengths and weaknesses, some of which will be shortly presented next.

Triangulation uses two sensors that operate as the two segments of a triangle, with the detected point being the third [Konolige and Nüchter 2016]. The strength of a sensor like this is the added accuracy it provides. By having two sensors that are separated by a baseline of length, we can achieve precise depth perception of the detected area that can be beneficial when high accuracy of the detected surface is needed. One good example of a sensor such as this is a stereo camera that consists of two separate cameras with slightly different viewpoints. These cameras use texture matching to determine corresponding points that are used in generating a 3D-depth image. The underlying weakness is the usual lack of texture that can be used to make reliable matches. In our case, visibility is also a critical factor: we must be able to get accurate sensory data regardless of possible visibility restrictions described above. Stereo cameras rely on good visibility for texture matching. Due to these limitations, especially when considering the rather small area-of -interest that narrow power lines draw, a suitable sensor had to be found elsewhere.

Time-of-flight (ToF)-sensors are called LIDARs (*light detection and ranging)*. The general principle of operation behind time-of-flight measurement is the same as radar, but instead of radio waves, we use light: by measuring the two-way time between transmission of a pulse and detection of the signal that is reflected from the surface of the measured object, we can measure the depth of surrounding objects within the measured environment. Due to light traveling at about 0.3 meters per nanosecond, direct ToF-measurement requires very precise timers. The advantage of ToF is its theoretical ability to have constant depth measurement regardless of how far the measured object is. But unlike triangulation sensors such as stereo cameras, ToF sensors cannot achieve the same close-range precision [Konolige and Nüchter 2016]. Our use case requires a safe distance between the sensor and measured area, so we do not need to worry about this disadvantage. Another significant advantage of a lidar laser sensor is the capability of providing data regardless of visibility restrictions [Murphy 2004]. ToF operation and theory will be further explained in Section 2.1.1.

In order to accurately combine multiple iterations of sensory data, inertial data of the used sensor is required to use. When mounting a sensor on a mobile platform such as hand-held device or a UAV, it’s absolute position and the orientation of the sensor must

be determined to gather useable data and keep track of the sensory device position in relation to the environment for accurate merging of multiple point cloud iterations during environment mapping [Murphy 2004]. To give positional reference of the sensor, inertial measurement unit is used to provide the roll, pitch, yaw and acceleration of the sensor to provide the method for translating sensor data into static points. This information is essential for accurate projection of the collected data. Inertial navigation will be further explained in Section 2.2.

2.1. LiDAR

A lidar device functions by sending pulses of light to measure distances of objects and surfaces. A lidar usually functions with a high density, providing data with high density and high accuracy [Murphy 2004]. These factors depend on the model of lidar we choose to operate with. Lidars vary by their number of light channels emitted, their rotational speed, and their range. Single channel lidars are usually used like line cameras: to provide a simple 2D-presentation of an object or environment. Multiple channels provide denser 3D- point clouds with the channel number being as high as 128 channels. Lidar technology is widely deployed on many different navigation and mapping systems due to its accuracy and ease of integration [Venugopal and Suresh 2013]. The range of most lidars is typically between 10-100 meters with accuracy of 5-10 millimetres. Lidars vary in size but are usually compact, being able to be easily mounted on vehicles.

2.1.1. Operation

As stated above, the method of ranging using light is called *direct time of flight*, which uses lasers and a chronometer to measure the travel time of the laser. According to Konolige and Nüchter (2016), by receiving the bounce of the laser, we can use the following equation to measure the travelled distance:

$$2d = ct \tag{1}$$

Where *d* represents the distance of the object surface that reflects the laser, *c* represents the speed of light, and *t* the time of flight.

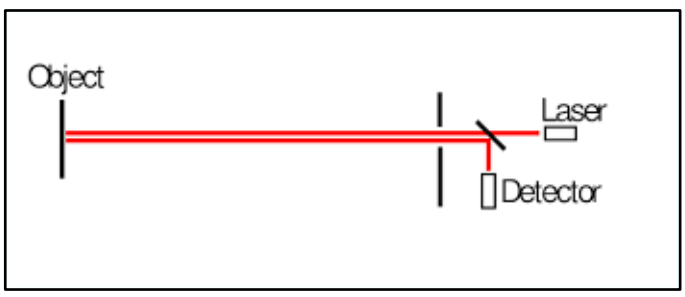

Figure 1: time of flight-operation

With this method, we get an accurate distance between the sensor and the measured surface. As shown in Figure 1, in its simplest form, ToF sensors use a single beam to send and receive a reflection from the environment. Thus, the resulting measurement is only obtained from a single point. In order to generate denser surface reading, the data is supplied as a vector containing distance values from multiple points. This is achieved by sweeping the beam across the scene. Rather than moving the laser itself, the sweeping motion is achieved by using rotating mirrors. Receiving multiple scanning beams causes an error probability: as the scan repetition rises, ambiguity about which pulse is received rises as well. This is called *ambiguity interval*, which can be calculated as

$$1/2c\delta t \tag{2}$$

Where δt is the time between pulses. Error correction is done by the laser pulse intensity to adjust to the conditions and circumstances of the environment. The returned pulse is measured in relation to the coordinate system fixed to the scanner, not earth. With multi-beamed measurement, the sensor uses ambiguity interval to reduce the uncertainty with mixing multiple bouncing beams with each other [Konolige and Nüchter 2016]. More advanced lidars use multiple, up to hundreds of channels per sensor. Rotating lidar-sensors sweep an area to get a denser representation of the world in correlation with the number of channels emitting laser pulses that bounce back from the environment. The number of channels also increase the amount of data generated.

2.1.2. Point Clouds

The ability to digitally represent the world with computer graphics has undergone vast improvement during the last decades, especially in the field of surface recognition. In essence, surface recognition is the process of reconstructing a 3D object from a collection of discrete points that represents the objects shape [Berger 2017]. With proper sensory systems, surface recognition can give us a dense and accurate 3D-representation of the surveyed area. Using a lidar-sensor, the received laser bounces are constructed as a set of points as a point cloud. Each point has x, y and z-coordinates, and usually contains other supporting information, such as a timestamp and distance with colorization. Point cloud visualization can be achieved with appropriate software, which enables the visual estimation and further manipulation. Point clouds can be stored into many different formats, we use Point Cloud Library standard format (.pcl) since it is open-source and supported by software provided by the sensory equipment manufacturers.

Point clouds have seen use in various fields, from solving problems relating autonomous vehicles, to ground surface surveying in the building industry. In relation to the advancement of our sensory systems, point clouds can grow large: powerful computer hardware is necessary to both handle the initial collection of point cloud data and the

analysing of said data. With each rotation of a LiDAR sensor generating an individual point cloud consisting of up to 15,000 points, combined iterations of point clouds into a coherent map can grow very large in size and computationally laborious to analyse.

If the used scanner is unable to autonomously generate accurate models on its own from the gathered data, external algorithms must be used to combine the data into a useful form. Some sophisticated scanners provide complete solutions for the end-user for analytic purposes with integrated inertial sensory systems and computational capability, while others only provide raw data output that must be further processed to generate an accurate model. In this study we will be using a scanner that represents the latter group; Velodyne VLP-16 LiDAR. The operation of this sensor will be further explained in the next chapter.

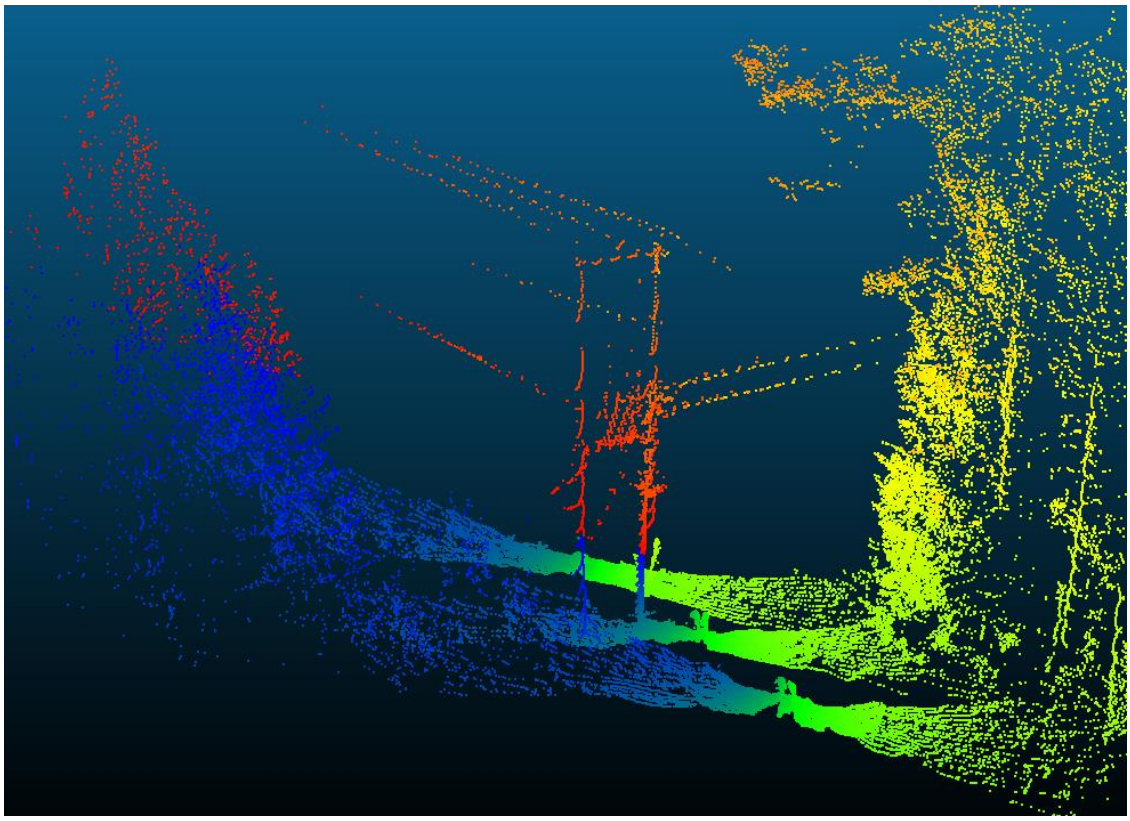

Figure 2: A point cloud representation of a power line transformer using VLP-16 lidar

With the point cloud density being the driving factor while evaluating the plausibility of our proposed solution, some further elaboration is needed on the topic. In order to differentiate a power line from the vast amount of 3D points generated by lidar sensors the generated points need to be close enough to each other to be considered as a cohesive line. When considering power lines, it is beneficial that we can generally assume that no excess clutter is expected around the lines during normal operation. If something unexpected is detected near the power line, it could usually be assumed as a fault. Finnish

power lines have a standardized safety distance between active power lines and the surrounding environment, such as growing foliage in forested areas. The distance grows larger as the voltage rises, starting from a couple of meters and being up to ten meters around power lines carrying more than 100 kilovolts. This makes the detection of power lines plausible from a large point cloud where small details can be lost due to density limits, since they can be easily observed and stand out from other points of data, as seen in Figure 2. In other words, in order to automatically detect faults around a power line, we must first find a way to classify a power line as such and find its position from all the gathered data. Questions about point cloud density arise when answering this problem: the generated point clouds must be dense enough in order to find, recognize and classify unique objects from. Higher density of objects reduces the probability of objects being labelled as noise. In our case, a single line is a very simple object. Therefore, this classification process condenses to the process of finding straight lines that have no excess clutter around them. This thesis does not delve deeper into point cloud object classification or automatic fault detection but tries to validate the sensors and data gathering methods that strives to this goal. Research done by Awrangjeb et al. [2018] suggest that both point cloud density and lack of noise play a critical role for accurate power line detection and classification. In our case, the generated point cloud density increases as more individual sets of 3D points (in other words, point clouds) are fused together to form a larger, cohesive map consisting of multiple sensor rotations, as each rotation generates an individual point cloud representation of the environment. In other words, accuracy increases as more rotations are gathered and fused together from a power line segment.

2.1.3. Velodyne VLP-16 LiDAR

Velodyne VLP-16 is a LiDAR-sensor released and manufactured by Velodyne Inc. [Velodyne 2021], a United States of America based company that evolved to its current form from the DARPA Grand Challenge of 2005. Velodyne has been acclaimed to be the leading developer in lidar technologies, manufacturing its real-time lidar scanners used in a variety of commercial applications. Interest towards the sensor in the surveying and mapping industry has been high due to the sensors small size, relatively low power requirements and competitive cost efficiency. The sensor was initially manufactured as a cost-effective collision detection sensor for autonomous vehicles [Glennie et al. 2016] and has since also seen use in 3D mobile and aerial mapping and security. As of 2021, the VLP-16 model costs approximately 4,000$. For comparison, Velodyne HDL-32E, consisting of 32 channels instead of 16 and with a mean precision of +-2cm instead of +-3cm, costs around 30,000$. Lastly, the most high-end Velodyne lidar Velodyne HDL-64E, having 64 laser channels and +-2cm mean precision, sets the customer back with about 75,000$.

Figure 3: Velodyne VLP-16 lidar sensor

From its release, VLP-16 has seen extensive use in laser-based sensory research. The emitted laser is classified as class 1, which means it is completely eye safe. These attributes make the sensor a solid choice for our detection tool: VLP-16 rotates with a full 360-degree horizontal scan and a 30-degree vertical FOV (-15° to +15° with 2° increments) with depth precision at approximately 2 centimeters. VLP-16 consists of 16 laser emitter/detector pairs (also called channels) operating at 905 nm wavelength. Thousands of laser pulses are shot and received from each laser channel in a sequential manner. From the gathered depth measurements, it is possible to generate a 3D point cloud in real-time [Krishnaswamy 2016]. The 16 laser channels operate by horizontally sweeping an area and gathering a series of ToF result from the observed area. As stated above, vertical increments between each sweep of laser channels are 2°. Due to beam divergence, this vertical angle increases exponentially as the measurement distance increases. In other words, as measurement distance increases, the further away from each other every laser channel sweep will appear. This relates to one of the central research questions of this thesis: how much of an effect this increasing distance between laser channel sweeps has on the density of the generated point cloud when measuring from a distance expected for a drone? An example of a laser sweep with every laser channel of VLP-16 is presented in Figure5.

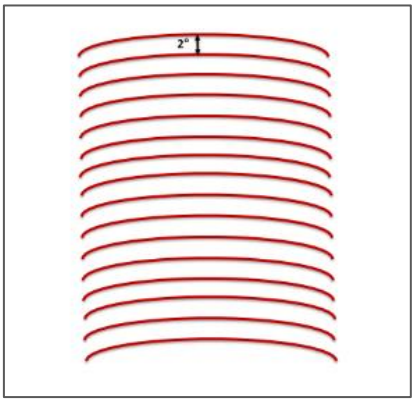

Figure 4: An Example laser scan using VLP-16 lidar [Krishnaswamy 2016]

In addition to distance measurement, VLP-16 provides reflectivity of a surface with 256-bit resolution. Dual return-mode enables two separate measurements of a reflection from the transmitted laser beam (strongest and last measurement). Measurement data also consists of the elevation and azimuth angles of the measured data and timestamps based either on the devices own internal clock or to an externally attached GPS device that VLP-16 supports via integration to its interface box that also houses power and ethernet connections between the device and user computer. VLP-16 also provides a handy web-based user interface from which device calibration can be managed. Values like rotation frequency, FOV, output ports etc. can be changed from the user interface.

According to Glennie et al. [2016], the scanner calculates the x, y and z-coordinates in the scanners own coordinate system as:

$$r^s \begin{bmatrix} \left(s^i * R_i + D_o^i\right) * \cos(\delta_i) * [\sin(\varepsilon) * \cos(\beta_i) - \cos(\varepsilon) * \sin(\beta_i)] \\ \quad -H_o^i * [\cos(\varepsilon) * \cos(\beta_i) + \sin(\varepsilon) * \sin(\beta_i)] \\ (s^i * R_i + D_o^i) * \cos(\delta_i) * [\cos(\varepsilon) * \cos(\beta_i) + \sin(\varepsilon) * \sin(\beta_i)] \\ \quad +H_o^i * [\sin(\varepsilon) * \cos(\beta_i) - \cos(\varepsilon) * \sin(\beta_i)] \\ (s^i * R_i + D_o^i) * \sin(\delta_i) + V_o^i \end{bmatrix} \quad (2)$$

where: s^i is the distance scale factor for laser i; D_o^i is the distance offset for laser i; δ_i is the vertical rotation correction for laser i; β_i is the horizontal rotation correction for laser i; H_o^i is the horizontal offset from scanner frame origin for laser i; V_o^i is the vertical offset from scanner frame origin for laser i; R^i is the raw distance measurement from laser i; ε is the encoder angle measurement. The first six parameters are interior instrument specific calibration values (interior calibration) and are supplied by the manufacturer in an xml document delivered with each scanner. The final two values (R^i and ε) are the observations returned by the scanner assembly for each individual laser [Glennie et al. 2016, Glennie and Lichti 2010].

The VLP-16 sends distance measurements through an Ethernet port as *User Datagram Protocol* (UDP) packets. Each data packet is 1248 bytes long and contains a 42-byte header and a 1206-byte payload. The device sends two kinds of data packets: Data Packets and Position Packets [Krishnaswamy 2016]. Since we are not going to utilize an external GPS, only data packets will be used. Explained by Krishnaswamy [2016], the 1206-byte payload includes 12 separate data blocks, a 4-byte timestamp and a 2-byte factory field. Every data block is made of 100 bytes and includes:

- two-byte start identifier (Flag xFFEE),
- two-byte azimuth value (Azimuth N, the laser beams horizontal firing angle)

- 32 x 3-byte data points (data from Channel N) from two firing sequences for each of the 16 laser channels. Each data point consists of two bytes of data representing the object distance, and one byte of data representing the calibrated reflectivity.

The laser channel elevation angles are fixed and unique to each channel, so the laser ID is used to determine the correct angle for each block. The azimuth values are reported once per data block, and the intermediate values that are missing are obtainable with the use of interpolation. For straight-forward data packet acquisition, VLP-16 is relatively easy to setup for basic use with default settings. Basic VLP-16 operation setup is illustrated in Figure 5.

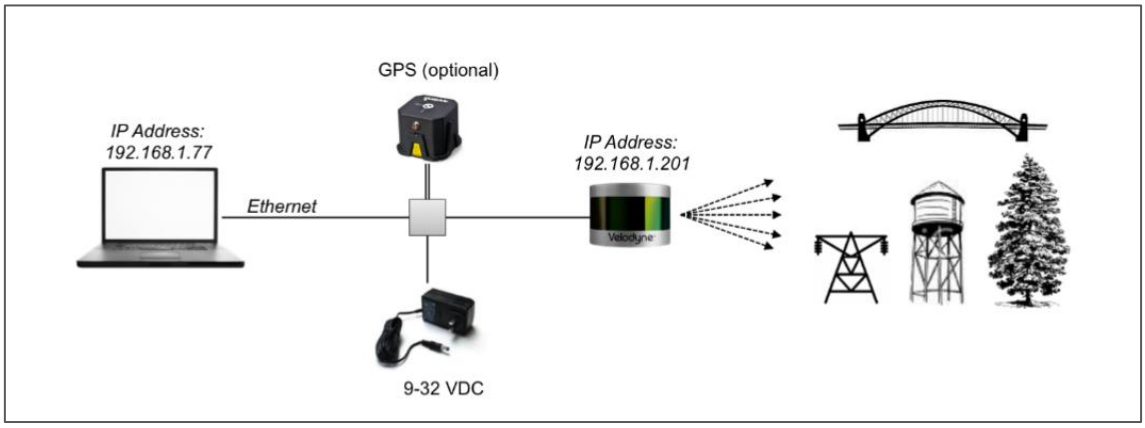

Figure 5: Velodyne VLP-16 operation setup

2.2. Inertial navigation

Orientational and positional data is needed to correctly establish the position of the scanner in the real world. As the scanner traverses through space, the coordinate system is under constant change. In order to fix raw data from the scanner into a correct relation with the world, it needs to be attached to a fixed coordinate system generated by data fusion with inertial data. To fuse subsequent iterations of data from a moving object, object position and orientation in relation to a known starting point can be used to match incoming data iterations over time with each other. Without inertial data from the vehicle a sensor is attached to, it is difficult to determine the exact position of observed landmarks represented as 3D points [Deilamsalehy and Havens 2016].

Inertial navigation is a technique in which measurements from a combination of sensors are used to track the relative position of an object to a chosen starting point. An Inertial Navigation System (INS) uses accelerometers, gyroscopes and magnetometers to determine the orientation and velocity of a moving vehicle. By processing these signals it is possible to track the position and orientation of a device moving in a three-dimensional space [Woodman 2007]. An INS can detect a change in its geographic position

(moving from east to west, for example), a change of its orientation (rotation in an axis), or a change in its velocity (speed, direction of movement). This is achieved by measuring the angular velocity and linear acceleration applied to the system. Inertial navigation is most commonly used in ships navigating at sea or in the air where the ships position and orientation can be hard to estimate, and more recently in much more compact and smaller vehicles with the rise of *Microelectromechanical Systems* (MEMS) [Grewal et al. 2006, Chatfield 1997].

2.2.1. Inertial measurement unit

To successfully determine the travelled distance and the orientation of our device in relation to our gathered data, an inertial measurement unit (IMU) will be used. An IMU is a combination of inertial navigation sensors that generate accurate data representing the orientation and position of a device, estimating relative acceleration, velocity and device position. According to Dudek and Jenkin [2016], here are two categories of IMUs: gimbaled systems and strap-down systems. Gimbaled IMU's are mounted within complex gimball structures that function as a stable platform from which accurate measurements can be gathered from. The gimball needs to be aligned with the initial frame of reference during start-up. A gyroscope is used to achieve this. Dudek and Jenkin explain that "the orientation of the gimbaled platform relative to the vehicle is used to map measurements taken within the IMU to the reference frame of the vehicle". On the other hand, strap down IMUs are, hence the name, strapped down and rigidly connected to the vehicle, thus eliminating the need for the same transformation method that the gimball approach uses. Nevertheless, both categories require integrating the motion relative to the initial frame, which requires integrating information from the IMU sensors. This is computationally more laborious to achieve, which explain why the gimball method was prominent prior to the 1970's whenever inertial measurements were done. Compared to today, these computational requirements are insignificant, so strap down IMU's are much more common [Dudek and Jenkin 2016].

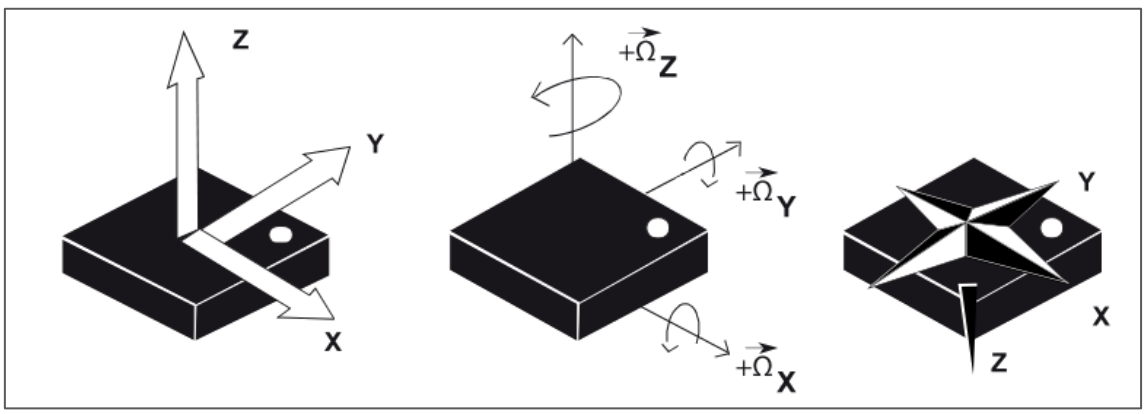

Figure 6: acceleration, angular rotation and magnetic force

Research advancements in robotics have given rise to IMUs as vital sensors in solving the navigational challenges of unmanned, autonomous devices. Thus, the usage if IMUs in robotics have reduced their relative size to better fit in smaller and more technologically advanced autonomous devices. Furthermore, decreasing prices among manufacturers have prompted IMUs to be more available for both commercial and non-commercial use and opened the possibility for cost-effectively solving more specific use-cases. IMUs vary with their features from offering simple inertial data to full six degrees of freedom (6-DOF) that contains the orientation and position of a vehicle (x, y, z & roll, pitch, yaw). These kind of IMUs are called *Attitude and Heading Reference Systems* (AHRS).

The IMU consists of three different sensors: an accelerometer, a gyroscope and a magnetometer, as illustrated in Figure 6. The accelerometer provides a measure of acceleration that is used to estimate the vehicles travelled distance. Accelerometers can be either mechanic or piezoelectric. A mechanic accelerometer uses a mass-spring system where the movement of the spring gives a reading in proportion to the displaced mass. The spring and mass are situated in a vacuum. The forces affecting the system are in parallel with the spring axis, and thus, can be measured. Piezoelectric accelerometers are based on the use of certain crystals which generate a voltage when under stress. A small mass is positioned next to the crystal that, upon under force, acts upon the crystal that induces a measurable voltage. Accelerometers are sensitive to external forces acting upon them, including gravity [Dudek and Jenkin 2016]. Capacitive sensing can also be used, that follows the same principles as a mechanical spring accelerometer, but instead of measuring mass displacement, a capacitance is measured [Andrejašic 2008].

The gyroscope provides values on the roll, pitch and yaw of the vehicle. According to Dudek and Jenkin [2016], “gyroscopes and gyrocompasses rely on the principle of conservation of angular momentum, which is the tendency of a rotating object to keep rotating at the same angular speed about the same axis of rotation in the absence of external torque”. Same as the accelerometer, the gyroscope uses mass displacement measurement with an output signal that is proportional to the mass movement perpendicular to its axis of oscillation. With movement and rotation, the mass inside the gyroscope experiences force that is greater when the mass is further away from the centre of rotation. When we compare the signal readout from the sides of oscillating movement, we can determine the rate of turn of the device. Dudek and Jenkin [2016] further elaborate that an individual gyroscope only measure rotation about a single axis. Therefore, in order to measure 3D rotations, it is common to combine multiple gyroscopes together with orthogonal axes of sensitivity.

The magnetometer assists the gyroscope with determining the heading of the device. A magnetometer is a device that measures magnetic field or magnetic dipole moment. The earth’s magnetic field resembles that of a simple bar magnet. This magnetic

dipole has its field lines originating at a point near the south pole and terminating at a point near the north pole, which are referred as the magnetic poles. The strength and direction of these field lines vary across the earth's surface. The direction and strength of the earth's magnetic field can be represented by three axis values. These values can be used to determine compass headings in reference to the magnetic poles. The AHRS system uses sensory fusion to merge measurements from these three sensors to provide inertial data.

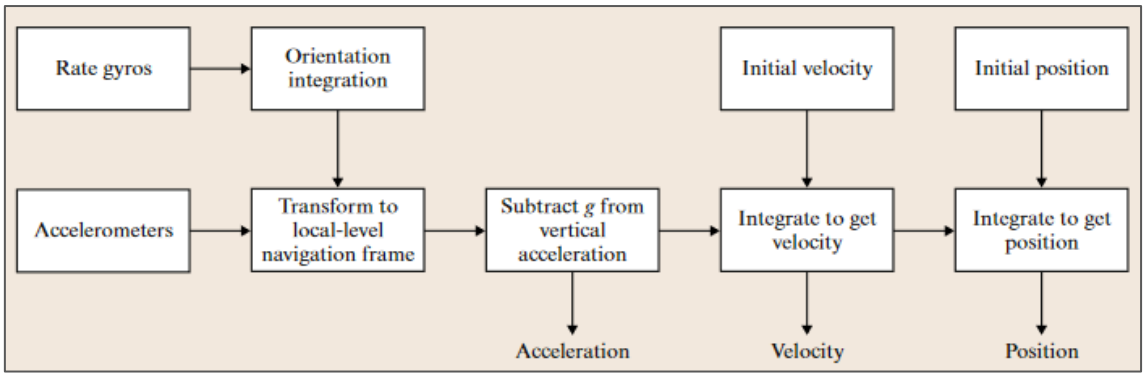

Figure 7: AHRS-system overview [Dudek and Jenkin 2016]

The IMU used in this research is an Xsens MTI-630 AHRS-system, seen in Figure 8. The frequency of data gathered is 200 hertz, which is an advantage during sensory fusion since its vital to get as accurate reading as possible that best represents the time a reading from another sensor have been received. Thus, there are multiple measurements gathered from each lidar scanline window, which can be used to provide a more reliable prediction of the attitude and position of the device [Karam et al. 2019]. IMU measurement has its challenges: some sensor components, especially the gyroscope, are subject to some error margin and can have a detrimental effect to the navigational output [Woodman 2007, Dissanayake et al. 2001]. Slight errors in acceleration data can have a gathering effect in relation to the gravitational acceleration that has a direct impact on the sensory fusion output. To combat this accumulating drift, IMU manufacturers have implemented computational margin-of-error-algorithms that try to combat the always-present, although small, rate of error. [Des Bouvrie 2011]

Figure 8: An Xsens Mti-630 AHRS Inertial Measurement Unit

2.2.2. Position and pose estimation

As stated above, when using a device moving through three-dimensional space, methods of gathering and representing the position and orientation of the device must be used. Correct position and pose estimation are essential for point cloud merging. Orientation and position can be determined by using sensory fusion methods with the raw IMU data: with the gathered data, we can appropriately determine the position and orientation of our device through inertial navigation where we compute the position and orientation of the measured object in relation to a known starting point. The position of an object can be represented with the use of "coordinate reference frames": A coordinate reference frame *i* consists of an origin and three vectors, *x*, *y* and *z,* that are all fixed within a particular body. With these measurements we can express the relation between two different coordinate frames. This relation between frames *i* and *j* can be denoted by the 3 × 1 vector: $({}^{j}p_i^x,\ {}^{j}p_i^y,\ {}^{j}p_i^z)$, which represent the coordinates x, y and z [Waldron and Schmiedeler 2016]. In navigation, this vector is represented as coordinate frame with latitude, longitude and altitude. In order to track the position of the device we will project the signal received from the IMU's accelerometer into the initial frame of reference. We can derive these three position parameters by double integrating the acceleration data [Waldron and Schmiedeler 2016]. The tracked acceleration signal a_b with *t* as time is:

$$a_b(t) = (a_{bx}(t), a_{by}(t), a_{bz}(t))^T \tag{3}$$

which is projected to the frame of reference:

$$a_g(t) = C(t)\, a_b(t) \tag{4}$$

Gravity acts as a constant force to the accelerometer. For velocity obtainment we subtract the force of gravity affecting the accelerometer and integrate the remaining velocity:

$$v_g(t) = v_g(0) + \int_0^t a_g(t) - g_g dt \tag{5}$$

Where $v_g(0)$ is the initial velocity if the device during the initial frame gathering, and g_g is the subtracted velocity due to gravity. To obtain the displacement between the two frames of reference, we use the obtained velocity and integrate again:

$$s_g(t) = s_g(0) + \int_0^t v_g(t)dt \tag{6}$$

where $\boldsymbol{s_g(0)}$ is the initial displacement of the device in the initial reference frame.

As the new samples are constantly arriving from the accelerometer, we will use an integration scheme to weave the new readings in and update the values [Dudek and Jenkin 2016]:

$$v_g(t+\delta t) = v_g(t) + \delta t \cdot \left(a_g(t+\delta t) - g_g\right) \tag{7}$$

$$s_g(t+\delta t) = s_g(t) + \delta t \cdot v_g(t+\delta t). \tag{8}$$

The pose estimation of the device can be derived from three attitude parameters: roll, pitch, and yaw. These parameters can be determined by integrating angular velocity with time between the initial frame of reference and new measurement frames [Waldron and Schmiedeler 2016]. The 3D orientation output format can be determined for Xsens MTI-630 to be either Euler angles, quaternions or as a rotation matrix. For our usage we will be representing the pose estimation algorithm as a direction-cosine-based system. The measurement body will be compared to, in a same sense as the position estimation, to the initial frame of reference gathered at the beginning of measurement. The body frame reference is represented as a 3 × 3 rotation matrix $C(t)$. With the tracked angular rate signal $\omega^s = (\omega_x^s, \omega_y^s, \omega_z^s)$ gathered from gyroscope readings, we can accumulate the change of orientation:

$$\dot{C} = \lim_{\delta t \to 0} \frac{C(t+\delta) - C(t)}{\delta t}. \tag{9}$$

Here $C(t+\delta t)$ can be written as a product of two matrices

$$C(t+\delta t) = C(t)A(t) \tag{10}$$

With $A(t)$ being the rotation matrix that relates the time *t* and $t+\delta t$ in the sensor measurement frame. When the device experiences small rotation changes between high-frequency measurement, we can approximate the small angle changes

$$A(t) = I + \delta\psi \tag{11}$$

with rotation matrix $\delta\psi$ representing the small changes of rotation during measurement. In the limit this matrix is equivalent to

$$\lim_{\delta t \to 0} \frac{\delta\psi}{\delta t} = \Omega(t) \tag{12}$$

with $\Omega(t)$ representing the skew symmetric form of the angular rate vector ω^s. Thus, the change in orientation yields

$$\dot{C} = C(t)\Omega(t) \tag{13}$$

Last, we integrate both parts of the equation

$$C(t) = C(0) \cdot exp(\int_0^t \Omega(\tau)d\tau) \tag{14}$$

with $C(0)$ being the orientation of the device in the initial frame of reference [Woodman 2007, Diaz et al. 2015]. The Xsens IMU uses calibration models to filter and smooth out the measurements, since the sensors are prone to error and can cause inaccurate position and orientation estimations the further the measurements continue and cause position and pose orientation estimation drifting.

3. Simultaneous localization and mapping

As previously described, the main goal of this system is to generate a high-resolution representation of an area of interest environment for analysis. In our case, the primary areas-of-interest are the power lines and the surrounding area since one of the goals is to analyse, in addition to the power lines themselves, the area around the power lines for occurrence of excess foliage and disrupting objects. To generate a high-resolution environment map with an autonomous device we use the simultaneous localization and mapping process. Simultaneous localization and mapping (SLAM) is a process where a device builds a map of its surroundings and simultaneously uses the generated map to determine its location. SLAM is used and often required by systems that, by either implicitly or explicitly, require a globally consistent map. Applications like structural inspection in the construction and environment surveying fields have successfully implemented SLAM where a globally consistent 3D reconstruction is a requirement for successful operation [Cadena et al. 2016].

The SLAM-problem has existed within the realm of robotics for decades and is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robots [Thrun 2007]. While our approach does not try to benefit from completely autonomous pathfinding due to the use of drone path-planning, SLAM-mapping provides the necessary theory and tools to generate an accurate visualization from a laser-based sensory device attached to a vehicle that's position and orientation are under constant rapid change. The SLAM-problem asks if it would be possible to place a device into an unknown environment and with the use of appropriate sensory technology, generate a coherent map of said environment and use it to determine the location of the device while it traverses the environment.

There are mainly two reasons why a map is needed: first, to support other tasks related to SLAM, a map is often required. The generated map provides an intuitive map for a possible human operator of the vehicle if the system is not completely automated. In the case of automated vehicle movement, a map is required to inform path planning and reduce the error committed in estimating the state of the vehicle [Cadena et al. 2016]. If a map is not used, drift in vehicle and landmark position estimations would quickly start growing due to dead reckoning. In recent years major advancements have been made in solving the SLAM-problem, and as a theoretical problem, a solution has been formed in several different forms [Durrant-Whyte and Bailey 2006]. SLAM-algorithms have been successfully used in various use cases, such as indoor-and outdoor robotics and airborne systems. However, a more general solution has yet to be discovered as usage of the process revolves around specific use-cases and have great variation depending on the application SLAM is used for. This can be seen by the rise of three different algorithmic paradigms that have been utilized in answering the problem.

Recent advancements of computational power in small single-board computers have enabled the practical use of SLAM-algorithms in simple low-cost robotics: computational complexity, data acquisition and environment representation have all been key research targets that are all demanding processes for small computers. As research has enabled new ways to make SLAM more accurate and powerful, this correlates into new, more demanding power requirements to run SLAM on a moving device while maintaining compact size and reasonable price per unit. With these advancements we can answer the need for raw computational power to efficiently manage big data flow and algorithmic sub-processes.

3.1. Preliminaries

When handling the SLAM-problem, it is beneficial to establish some preliminaries first. SLAM is formally best described in probabilistic terminology [Thrun 2007]. According to Durrant-Whyte and Bailey [2006], while a SLAM-mapping device traverses through an environment and gathers data of surrounding landmarks, the following quantities are defined, represented in Figure 9:

- x_k – location and orientation of the vehicle represented as a state vector,
- u_k – a control vector to drive the vehicle to a state x_k at time k, applied at time $k-1$,
- m_i – a vector that describes a landmark found at ith location whose true location is assumed time invariant,
- z_{ik} – an observation of a location of the ith landmark at time k, and
- z_k - multiple landmark observations at any one time or when a specific landmark is not relevant.

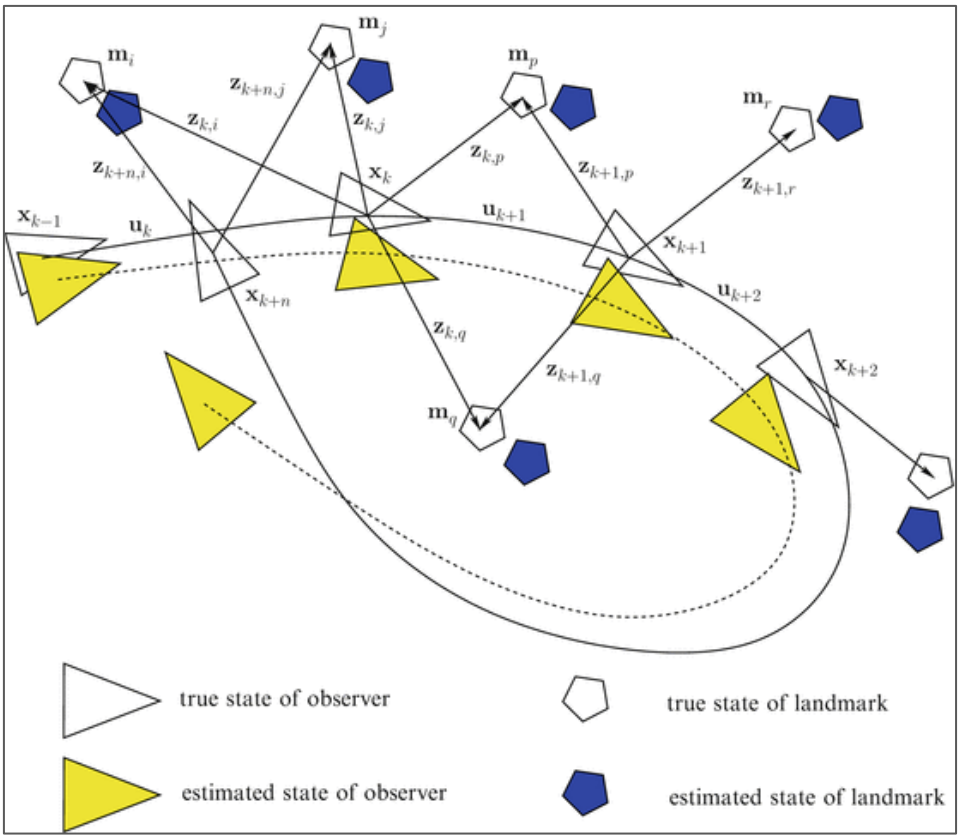

Figure 9: The SLAM-problem [Hess et al. 2016]

With these known preliminaries we can define the following sets that describe the history of vehicle locations, control inputs and landmark observations

- $X_{0:k} = \{x_0, x_1, \dots, x_k\} = \{X_{0:k-1}, X_k\}$: history of vehicle locations
- $U_{0:k} = \{u_1, u_2, \dots, u_k\} = \{U_{0:k-1}, U_k\}$: history of control inputs
- $Z_{0:k} = \{z_1, z_2, \dots, z_k\} = \{Z_{0:k-1}, Z_k\}$: set of all landmark observations
- $m = \{m_1, m_2, \dots, m_k\}$: set of all landmarks

To answer the SLAM-problem we will calculate a probability distribution to describe the concerted posterior density of the landmark locations and vehicle state for all times *k*. This includes all landmark observations and control inputs given to the vehicle up until and during *k* from the initial starting state of the vehicle. The probability distribution can be described as

$$P(x_k, m | Z_{0:k}, U_{0:k}, x_0). \tag{15}$$

To efficiently address the SLAM-problem, a recursive solution is generally considered as the default approach. When we start with an estimate for the probability distribution

$$P(x_{k-1}, m | Z_{0:k-1}, U_{0:k-1}) \tag{16}$$

at time $k-1$, the joint posterior is computed using Bayes theorem after receiving control inputs and landmark observations while the vehicle advances forward in space. Bayes'

theorem describes the probability of an event when prior knowledge of conditions that are known that are or might be related to the event [Swinburne 2004]. To utilize the Bayes' theorem with this computation we will use an observation model and a motion model for the computation process: the observation model

$$P(z_k|x_k, m) \tag{17}$$

describes the probability of seeing when we know the locations of prior vehicle and landmark observations gathered before *k*. The motion model

$$P(x_k|x_{k-1}, u_k) \tag{18}$$

describes the probability distribution on vehicle state transitions. Here the transition is assumed as dependent only on the preceding vehicle state and applied control input and does not take the known observations or the generated map into account in the model.
With these preliminaries we can implement the two forms of recursive prediction that are calculated in sequential order as a two-step process. These forms are the time-update form:

$$P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k|x_{k-1}, u_k)P(x_{k-1}, m|Z_{0:k-1}, U_{0:k}, x_0)dx_{k-1},$$

and the measurement update form:

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(Z_k|X_k, m)P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})}. \tag{20}$$

These equations recursively calculate the joint posterior for the vehicle state and map at time *k* that are based on all the control inputs and landmark observations before *k* [Durrant-Whyte and Bailey 2006].

3.2. SLAM paradigms

As SLAM-research progressed through history, different theories evolved to answer the problem by either adding to the existing suggested solution or by innovating new approaches for more efficient and accurate solutions. These different approaches can be generalised into three main SLAM paradigms. Knowledge of these different paradigms is useful for understanding the SLAM-process as the chosen method greatly defines the possible outcomes that can be reached: measurement accuracy, computational efficiency and scalability are all greatly affected by the chosen method, as we can expect different levels of robustness across the different algorithms [Durrant-Whyte and Bailey 2006]. I

will briefly introduce the three main paradigms considering SLAM-research and explain some of their strengths and weaknesses. Research done by Stachniss et al. [2016] about the different SLAM paradigms and the algorithms that fall within each category is used as the main source for this chapter, as they explain the functionality behind each paradigm in much detail.

The earliest, while also the most influential approach, was the *Extended Kalman Filter* (EKF) SLAM. This SLAM-method was the first one to introduce the approach to store the gathered robot locations and environment features into a single state vector. This vector has an associated covariance matrix that determines the uncertainty of gathered data points, including estimates on how the vehicle and feature states correlate between each other. When the state vector is updated with new robot positions and environment features, an extended Kalman filter [Welch and Bishop 1995] is used on the new acquired measurements. By default, by implementing linearized models to non-linear motion, EKF-SLAM can only perform single linearization. As new data points are introduced into the state vector, the system covariance matrix grows quadratically with the number of observed landmarks [Thrun and Wolfram 1998]. This is a key concern of the EKF SLAM as it severely limits scalability. Many optimizations have been suggested and successfully implemented to EKF-SLAM, allowing it to process large amounts of landmarks more efficiently. Nevertheless, due to its limiting computational properties, EKF SLAM has fallen out of style as more efficient approaches rose to address the scalability issue [Stachniss et al. 2016].

The second paradigm is the *particle filter* SLAM. This approach uses a set of particles to represent the robot posterior. A single particle could be best described as a concrete guess as to what the true value of the robot state is at the given time. When a set of guesses like this are gathered into a set of guesses as described above, a particle filter is used to generate an approximation of the robot posterior distribution within a map. What makes particle filter SLAM challenging is the vast space of potential maps and robot paths. As the general principle behind particle filter SLAM is to populate an area with particles containing the state guesses, the number of particles grow very large as the number of features within a map grows. This causes the particle sets to scale exponentially in relation to the underlying dimensions of the state space. In addition, when using particle filter SLAM algorithms, the number of required sample size for map generation is usually given manually to the algorithm as an educated guess. The larger the uncertainty of mapping is via filtering, the larger the significance of this parameter is to the correct functioning of the algorithm. When compared to EKF-SLAM, particle filter considerably rises both accuracy and computational efficiency, but still lacks in effectiveness when we want to SLAM-map big areas over a longer span of time, rather than mapping smaller in-door environments [Stachniss et al. 2016].

The third paradigm is the *graph-based* SLAM approach. Unique compared to the previous groups of algorithms, graph-SLAM functions by forming a graphical representation of the SLAM-problem: environment landmarks and robot positions are treated as nodes in a graph that are tied together by using the robot odometry readings. According to Stachniss et al. [2016], every edge between two nodes represents a data-dependent spatial constraint between the nodes. When we get consecutive location readings x_{t-1} and x_t, we can tie their edges together conveyed by odometry reading u_t. As the robot moves forward, further edges form between edges of nodes that correspond to locations x_i and landmarks m_i, when we assume that during time t the robot was able to sense landmark i. Next, lets illustrate the graph generation with figures. For example, in Figure 10, a robot in position x_1 senses a landmark m_1. An arc is formed in a graph between the robot position and landmark reading. When the edges are cached in a matrix, a value is added to the elements between robot location x_1 and landmark reading m_1.

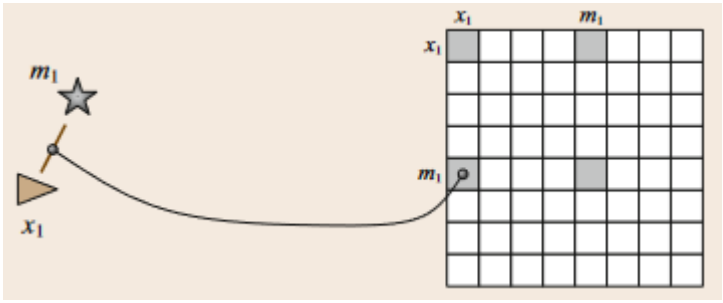

Figure 10: Adding initial values to the graph [Stachniss et al. 2016]

As the robot moves forward, using odometry reading u_2 from a new position, an arc between nodes x_1 and x_2 is formed and added to the graph next to previous measurements. This formulation is represented in Figure 11.

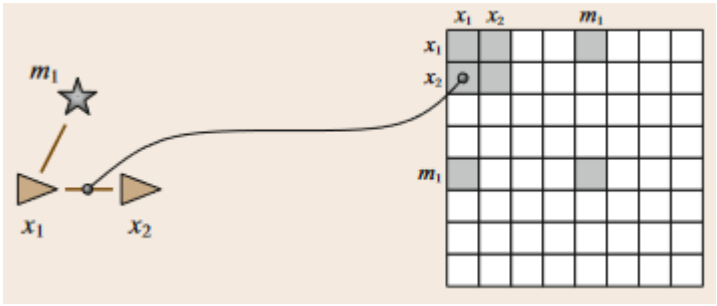

Figure 11: robot moving from point x_1 to x_2 [Stachniss et al. 2016]

As the robot moves and discovers new environment landmarks, with the use of these two basic steps we can fill out the graph over multiple sensory data iterations. The graph increases in size as the robot traverses forward and more nodes are added. This is represented in Figure 12.

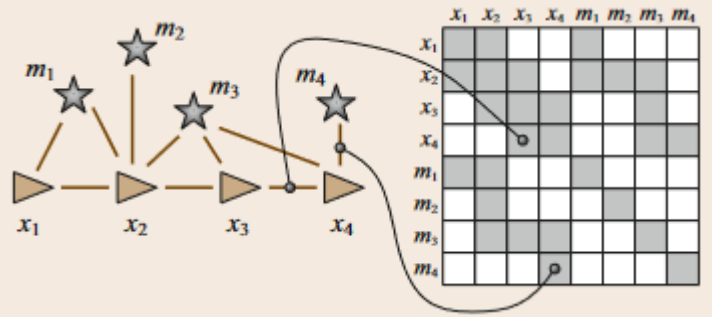

Figure 12: Filling the graph with consecutive steps [Stachniss et al. 2016]

The advantage of graph-SLAM is its efficient optimization techniques that can be applied to the SLAM process. Graph-SLAM scales to much higher-dimensional maps than, for example, EKF-SLAM due to having to covariance matrix and having nonlinear sparse optimization techniques applied to it. When reconstructing highly accurate environments, optimizing each individual measurement often results in better results than just optimizing robot poses. The graph updating time is constant and the amount of memory required is linear [Pierzchała et al. 2018, Montemerlo 2002].

These three SLAM-paradigms explained above cover most of the research and work done in the field of SLAM. With this knowledge we have a better understanding of the different approaches to the SLAM-problem and can use this information to determine which approach would suit the needs of specific use-cases. In our case, graph-SLAM is showing most promise with its scalability, optimization options, and reduced computational requirements.

3.3. Loop closure

During the SLAM-process, the device can deviate from its actual position from either the inevitable error rate of used sensors or from algorithmic filtering necessary for generalizing the large amount of sensory data. To combat drift accumulating in position estimation, a process called loop closure is used. Explained by Newman and Ho [2005], within the context of SLAM, “loop closing is the task of deciding whether or not a vehicle has, after an excursion of arbitrary length, returned to a previously visited area”. Loop closure uses landmarks, or in other words, already known areas of the environment generated by SLAM to re-visit and re-evaluate its position. Loop closure can be outlined as scan-to-map-matching where previously known locations are merged into a new set of detections containing known locations from previous scanning iterations. Known points are merged as a new point cloud and new locations are added to the map. Loop closure is an effective method in eliminating accumulating errors and increasing accuracy in SLAM.

Scan-to-scan-matching has also been utilized in laser-based SLAM approaches, where subsequent scans are merged without the aid of a map. This approach was discovered to quickly accumulate error. Scan-to-map-matching limits the accumulation of such error during scan-matching [Wolfgang et al. 2016]. Inertial data can be used for gravity orientation estimation to correctly project the laser sweep to a horizontal plane, and thus improve the accuracy of loop closure detection. In Figure 13 we can see how errors and uncertainties quickly grows in the robot's location and pose estimation while moving forward. When the robot returns near a position it has previously been and re-observes known landmarks, loop closure enables the rapid decrease of error and uncertainty with scan-to-map-matching [Zhang et al. 2015].

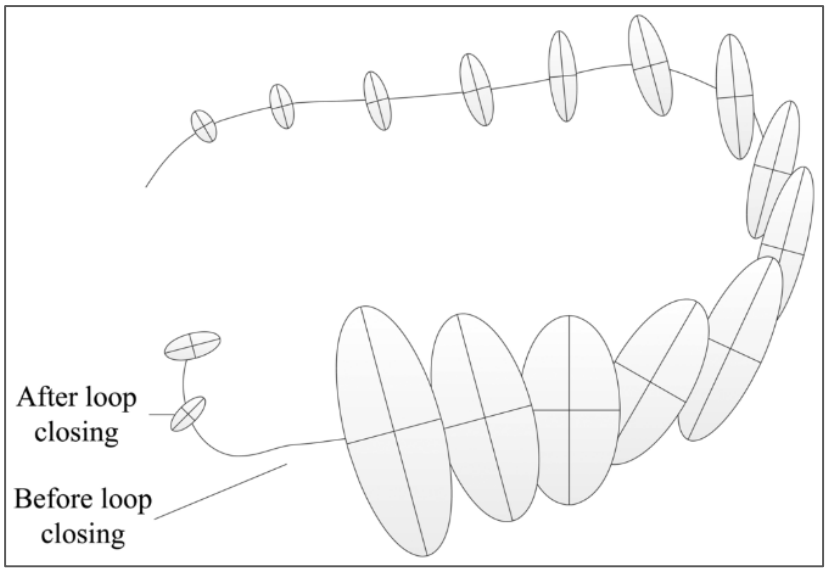

Figure 13: Accumulating error rate before and after loop closing [Zhang et al. 2015]

3.4. Google Cartographer

Google Cartographer [Cartographer 2021] is an open-source library of SLAM-algorithms developed by Google. Released in 2016, Cartographer provides both 2D and 3D-SLAM-maps and supports both offline-and online SLAM methods. By the term "offline" we refer to SLAM-mapping done to a previously gathered dataset. In online SLAM, data is processed in real-time and therefore is computationally more challenging. Explained by Nüchter et al. [2017], Google Cartographer delivers high-resolution SLAM results both offline and online, which is impressive since Cartographer can be run on commodity hardware and has somewhat modest system requirements. Cartographer requires a 3rd generation i7 or later, 16GB of RAM and can be operated on Ubuntu versions Bionic (18.04) or Focal (20.04). Cartographer is written in C++ and uses gcc (GNU Compiler Collection) versions 6.3.0, 7.5.0 or 9.3.0. Due to these requirements Cartographer can be

operated efficiently with single board computers that usually have less computational capacity than traditional computers but are easier to utilize in robotics for their compact size. Cartographer integrates with other systems with the aid of a middleware software called *Robot Operating System* (ROS), that is the de facto standard software in the robotics community [Nüchter et al. 2017]. ROS is used to create heterogeneous connections between software packages via a standardized inter-process communication system (IPC). ROS will be further explained in the next chapter.

According to Nüchter et al. [2017], the Google Cartographer library achieves its results by "grouping scans into probability grids that they call submaps and by using a brand and bound approach for loop closure optimization". The algorithmic function can be interpreted as working in two layers: the foreground and the background. In the foreground new scans are matched and added into the current submap. Meanwhile in the background, the scans are matched to a nearby submaps to create loop closure constraints. In 3D, the background process also tries to find the direction of gravity. The submap-and scan pose constraint graphs are continually optimized during scan acquisition. The local matching inserts new scans into the current submap, which accumulates error over time, and global matching that provides loop closing that removes the errors that accumulate in each submap that are part of the loop. These processes are outlined in Figure 14.

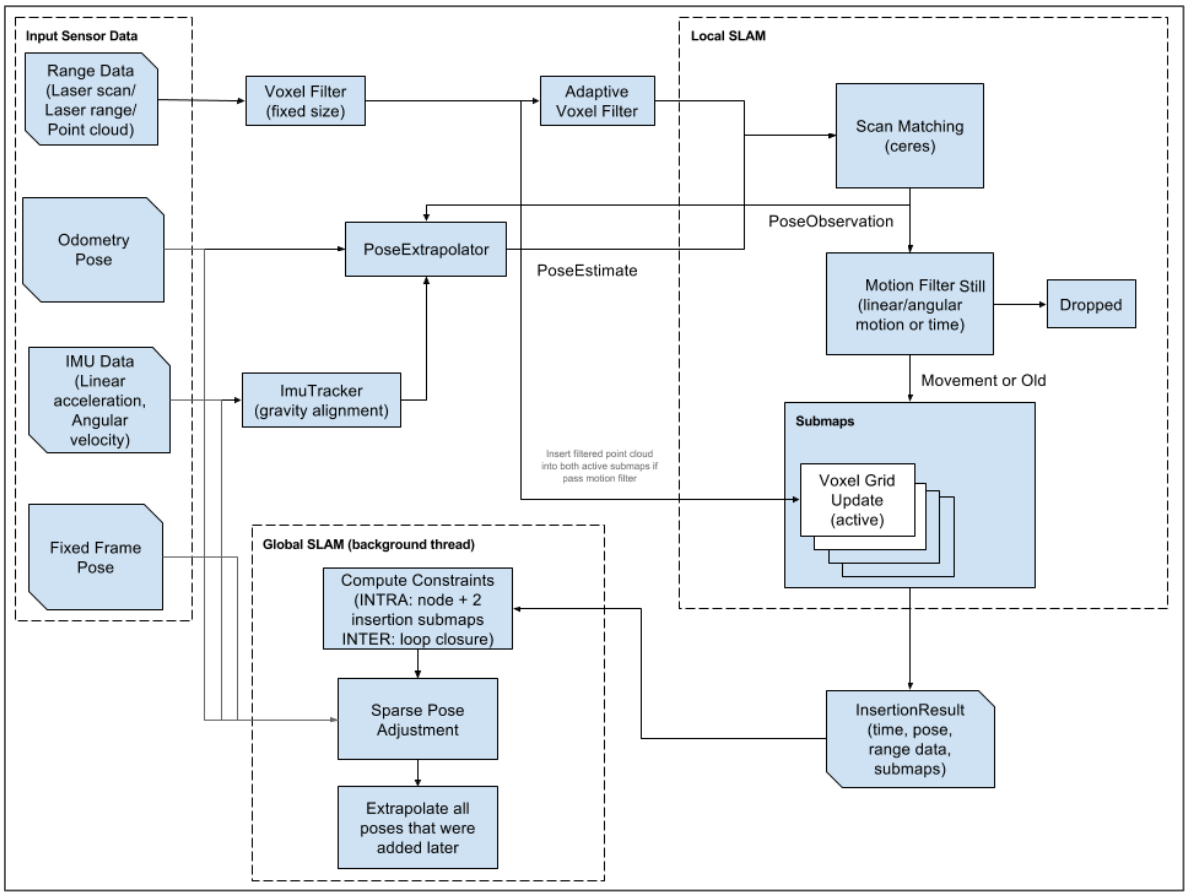

Figure 14: System overview of Google Cartographer [Hess et al. 2016]

Multi-channel lidars that rotate over 360 degrees gather depth information data from multiple directions at the same time. In SLAM-algorithms, some of this data can be irrelevant for mapping purposes: lidar-sensors can provide measurements from undesired sources (reflection, sensor noise) if measurement distance grows too far. To combat this, Cartographer SLAM applies a bandpass filter that disregards values outside of certain minimum and maximum ranges. These minimum and maximum values are always case-specific depending on used sensors and affecting environmental attributes. For example, during in-door scanning, these values would naturally be much smaller than out-door scanning [Hess et al. 2016].

3.5. Robot Operating System

As the scale and scope for robotics continues to grow, it has become exceedingly difficult to write software for the vast amount of different hardware configurations that have been utilized in the field. Code re-use is almost impossible due to the wildly varying robotic setups both in research and commercial use. The amount of required code can also be daunting, as robotics require a deep stack of software from driver-level all the way through perception, pathfinding and decision making. Beyond the capabilities of any single researcher or developer, integration of large-scale software must be supported in robotic software architectures [Quigley et al. 2009]. SLAM-algorithms require a multitude of data streams to function and requires many processes to be run concurrently and strictly synchronized. This makes ROS a go-to tool for SLAM-mapping, since it provides all the necessary tools and integration systems for developing, testing and visualizing these sorts of algorithms.

The *Robot Operating System* (ROS) is a robust open-source software framework developed for complexity management and rapid prototyping of software for experiments in the field of robotics. ROS is not an operating system from the perspective of process management and scheduling, but rather a structured layer of communications above the host operating systems. Originating from Stanford university during the mid-2000's, ROS was picked up for further development in 2007 and saw its first release in 2010. ROS provides a distributed network of processes (known as Nodes) that function as message passing processes, device controllers, package managers and state visualizers et cetera. Quigley et al. [2015] lists parts that ROS consists of:

- A set of drivers for sensor data reading, motor control and other actuators in an abstracted and defined format.
- A collection of fundamental robotics algorithms allowing map-building and traversing, sensory data simulation and acquisition, motion planning and object manipulation etc.

- The computational infrastructure to send and receive data, connecting various components of a complex system to incorporate the users own code into. ROS functions as a distributed system, allowing workload splitting across multiple systems. The ROS-architecture is primarily based on the "publish-subscribe" network model where nodes register with a master node that handles communications. The mechanism uses "topics" where nodes publish and subscribe to. This is implemented using Transmission Control Protocol/Internet Protocol (TCP/IP) or User Datagram Protocol (UDP). This enables concurrent message transaction during runtime [Krishnaswamy 2016].
- A set of tools to visualize and interpret a robot's state and record data from multiple sensory sources at the same time. Robot software debugging can be a daunting task, so ROS provides a set of tools to detect and correct faulty behavior.

As partly stated above, the fundamental concepts of the ROS implementation are nodes, topics, messages and services. Nodes are denoted as processes that perform computation. Nodes can be also interpreted as individual "software modules" that function independently. One of the design goals of ROS is to be modular: a system usually consists of multiple nodes working in parallel with each other and communicating through the network. As described in Figure 15, nodes communicate with each other by sending and receiving messages, which is a strictly typed data structure in ROS, supporting standard primitive types like integers, floats, booleans etc. Messages are sent from nodes to a topic, being a simple string describing the message content or its origin, like "odometry" or "map". A single topic can be published to or be subscribed by multiple nodes.

Although being flexible and robust, the topic-based publish-subscribe model is not sufficient for synchronous transactions. In ROS, a *service* is used instead. A service has a string name and a pair of strictly typed messages, one for the request and one for the response. Unlike topics, only one node can advertise a particular service name [Quigley et al. 2009].

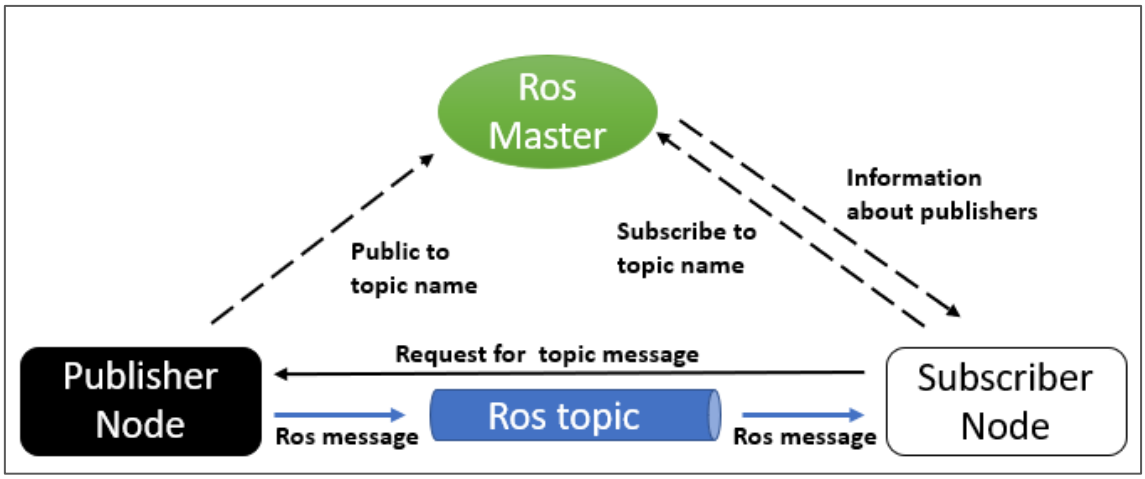

Figure 15: Robot Operating System communication principles

Through the collaboration efforts between many institutions and individuals in the open-source scene, ROS has grown to be one of the most prominent libraries in robotics, claiming its position as a *de facto* standard for robot programming. As an open-source project, ROS relies on the contributions from volunteering developers around the globe [Estefo et al. 2019]. Integration efforts have also proven fruitful, as ROS also supports integration with the most popular computer vision libraries. ROS is available as multiple different distribution versions that are akin to Linux distributions, like Ubuntu for example. Support of ROS-packages varies between distributions, which must be taken into consideration when choosing the right distribution for development. This thesis will use the version Melodic Morenia, released in 2018, since it supports all the packages that are used for our research.

4. Implementation

As mentioned in the introduction, the goal of this thesis is to validate the point cloud density and viability for power line condition inspection with the use of Velodyne VLP-16 lidar that is relatively low-cost and less accurate than its more expensive counterparts with considerably more laser channels (32, 64 or 128 channels compared to 16), enabling the generation of denser point clouds. Another goal of this thesis is to determine the viability of SLAM-mapping in generating structurally cohesive representation of the area of interest, in our case, a power line strip. Cohesion will be evaluated by point cloud density and feature extraction: are the lines dense enough to be detected as unique objects? ROS will be used in data acquisition to gather a dataset of the environment containing lidar-scans and IMU measurements of the lidar odometry and inertial state. In other words, offline-SLAM will be performed for point cloud generation. Opposed to online SLAM, the evaluated data is pre-recorded before the actual SLAM-mapping. Online SLAM is the goal of a possible complete application, but falls out of the scope of this thesis, as its implementation is substantially more challenging.

Google Cartographer will be used as a framework and algorithm collection for performing SLAM. Performing SLAM accurately depends on a variety of system calibrations and parameters, both in Robotic Operating System and Google Cartographer. Computational capacity of the host computer also influences the results [Biström 2019].
The target environment used for testing consists of a large 400 Kilovolt power line running from Tammisto, Vantaa to Anttila, Sipoo and a smaller, 20 kilovolt power line that is the main target for data gathering and analysis. The main grid's felling area is 40 meters wide, with safety distances of 10 meters. This is the closest range that we can expect our measuring system to gather data from. The measuring range does not affect system configuration. According to Fingrid, the company that owns and maintains the main power line grid of Finland, the power poles of a 400 kilovolt power lines are 31-35 meters high. 20 kilovolt line poles are 5-10 meters tall [Fingrid 2021]. Possible scenarios that inflict damage to the lines are fallen trees or foreign objects becoming violently in contact with the lines during storms.

4.1. System Setup

To test the SLAM-mapping, a dataset containing the measurements of the used scanners must be generated. After dataset generation, it can be SLAM-mapped using Cartographer. We will run ROS and Cartographer with MacBook Pro 2018 with 2,6 GHz Intel Core i7 CPU, Radeon Pro 555X GPU and 16Gb 2400 MHz DDR4 RAM memory. Due to the absence of ROS-support in MacOS Big Sur, ROS and Cartographer will be installed and run in a virtual machine (VirtualBox 6.1.18). A maximum amount of available computational power of the host machine will be directed for the virtual machines use. VirtualBox is known to be slow with some platforms and hardware combinations, and thus not being

able to utilize all the computational power the host machine provides, but my tests confirmed that the provided computational power was sufficient for both the dataset generation and offline SLAM-mapping process [Li, 2010].

The Velodyne VLP-16 lidar will be connected to the system via the ethernet port. The Xsens MTI-630 IMU can be directly connected to the system through a USB connection. The IMU will be directly strapped on the lidar to gather as accurate inertial data as possible. The lidar will be moved through the environment hand-held, attached to a standard tripod. The lidar can be expected to move quite much while being hand-held, but this can be negated with the aid of inertial data the IMU provides.

4.1.1. Velodyne ROS

The Velodyne ROS package [Velodyne ROS 2021] is an open-source ROS package that will be used for data acquisition from the UDP Ethernet packets sent by Velodyne VLP-16 lidar. This package was initially developed for the 2005 Grand DARPA Challenge by Austin Robot Technology Inc. Although initially developed to support the HDL-64E, another lidar by Velodyne, support for VLP-16 has since also been added [Krishnaswamy 2016].

The Velodyne ROS package consists of three ROS applications: "velodyne_driver", "velodyne_msgs" and "velodyne_pointcloud". Velodyne_driver is used to capture the raw UDP-packets and store them into a custom ROS message format "*velodyne_msgs"*. The messages are published into a topic called "*velodyne_packets*", so subscribing to this topic enables a node to receive lidar data. "*velodyne_pointcloud"* is an application used to extract the distance evaluation and azimuth values from the data coming from velodyne_driver application. The values are then converted from spherical coordinates to cartesian coordinates and published into a topic called "velodyne_pointcloud" [Krishnaswamy 2016].

4.1.2. Xsens ROS

The Xsens ROS package, called "*xsens_mti_driver*", provides a driver for the 1, 10, 100, and 600-series of Xsens IMU devices. The package is developed and maintained by Xsens, providing access for their customers to the driver on top of other configuration and calibration software included in their "MT Software Suite"-package. With the existence of dedicated calibration software provided by Xsens, the xsens_mti_driver node only receives data from an MTI-device and publishes it to ROS. If not properly configured, the node only receives data packets filled with zero-values until a specific data type is set for the device.

The Xsens MTI driver uses Public Xsens Device API for scanning, connecting, packet parsing and log file handling. There are 16 topics in total where the node publishes data to, most notably *"imu/data", "imu/acceleration", "imu/angular_velocity"* and

"imu/mag". Data sent to these topics will be accessible for the Cartographer nodes to subscribe to and receive data from for the SLAM-process.

4.2. Process

Here I will explain the process steps necessary to run SLAM and generate a map for evaluation. First, it's beneficial to do some initial tests with a smaller test-dataset to validate the IMU-lidar calibration in the ROS-configuration settings. This is extremely important, as even slight misalignment in the calibration will greatly affect the results as data is impossible to successfully fuse together if the IMU and lidar are not perfectly aligned with each other. We will monitor and configure the algorithm if needed for better results. Here I will better explain how to use Cartographer in practice. The next step is to decide a suitable target environment for analysis and generate a dataset of sensory data for the SLAM-algorithm. This part will be presented in the Subsection 4.2.2. After we have generated a suitable dataset, we can use that dataset as an input for the algorithm. Lastly, when we get an adequate result, we can start the evaluation process.

4.2.1. Initial test

First, let's start with the introduction of Cartographer. To start off we will first generate a small test-dataset to validate the lidar-IMU calibration of our setup. As mentioned before in Section 4.1, our system setup consists of a lidar and an IMU connected to a laptop: lidar via ethernet, and IMU through USB. We can verify the proper data flow by launching the ROS nodes and subscribing to a topic via "rostopic". In Figure 16, we see incoming lidar data, published by the xsens_mti_driver-ROS package.

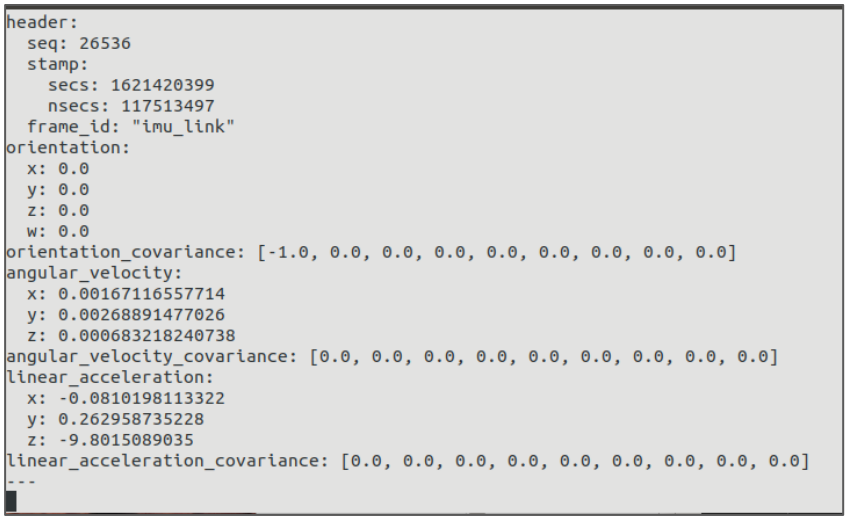

Figure 16: Example of IMU data published by the ROS node

When proper data flow is validated, we redirect this data to a topic expected by the Cartographer ROS-node. The node expects a pre-determined data transformation architecture

to function properly. This transformation architecture can be modified through Cartographer's configuration files, as different sensors publish data in their unique form and must be manually re-fitted to be accepted by Cartographer. In Figure 17, we can see the transformation architecture of our setup where the sensor nodes are linked together with link joints to form a coherent frame. This represents the simulated vehicle in ROS. Individual links are calibrated in relation to each other that resembles the true structure of our setup. IMU-data is received by the *imu_link*-joint that is connected to the Xsens MTi-specific *world*-frame. The lidar-data is published to the *velodyne*-joint and connected with the IMU-data in the *base-link*-frame. Odometry-frame is calculated based on this data and represented in the *odom*-frame. This frame is eventually linked into the *map*-frame where the data fusion is handled.

In our setup the IMU is strapped under the lidar. Therefore, we determine the xyz-coordinate difference between the sensors in the calibration file so proper sensory fusion can be achieved during offline simulation. Roll, pitch and yaw between the sensors and the vehicle can also be determined if necessary.

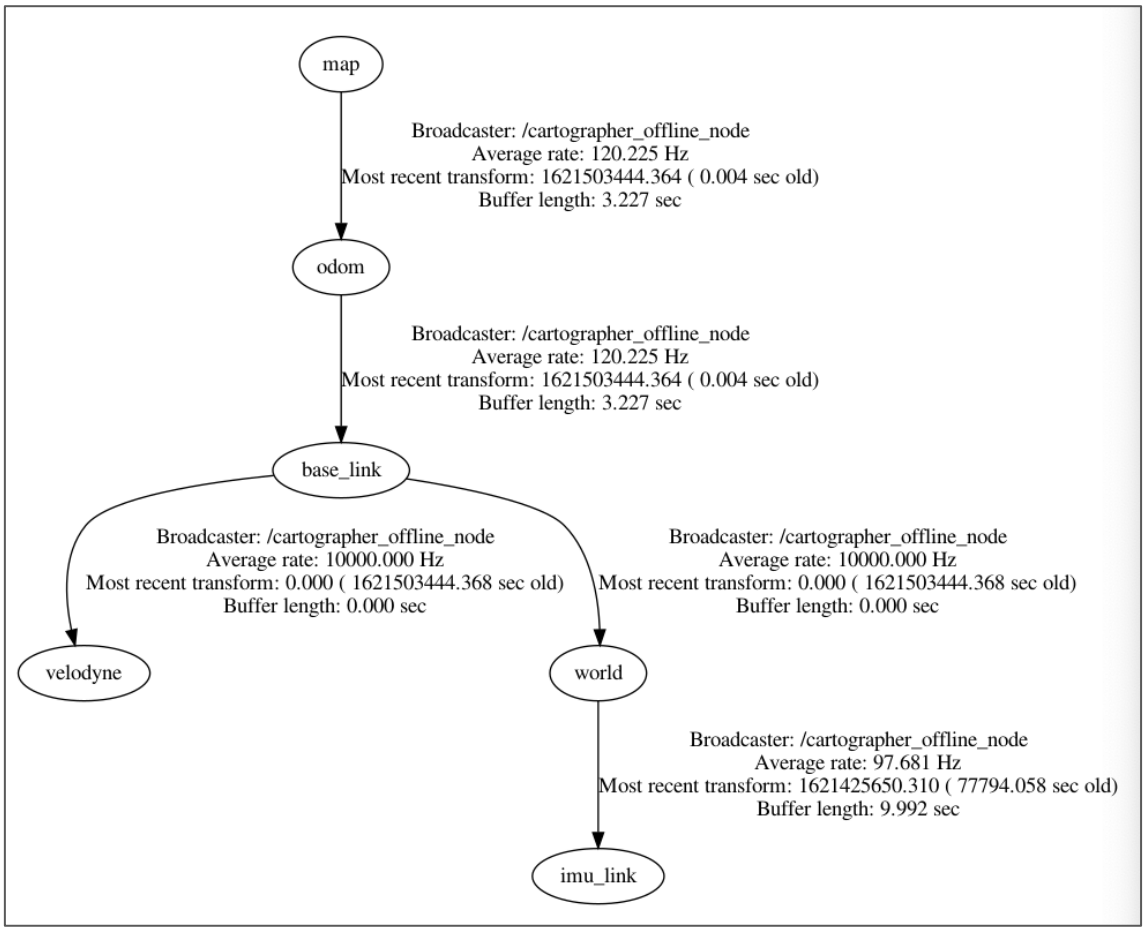

Figure 17: ROS transformation architecture

For generating a dataset, we will use a ROS-library application called Rosbag. Rosbag is a set of tools that can be used for recording active ROS topics where data is sent to from active ROS nodes. The recorded data can be later played back, simulating active sensor input. Rosbag is a high-performance tool that avoids deserialization and reserialization of messages, even during handling high frequencies of data coming from multiple sources. Rosbag is a convenient tool for calibration and testing, ridding us the need for continuous calibration and data acquisition cycles, while also enabling slower or faster data input feeds when it is necessary to carefully observe the algorithms progress.

The initial test was run in a 40-square meter apartment. In Figure 18, we can see the trajectory of our setup and the generated map. The IMU trajectory seemed to function as expected, with some minor inconsistencies that were successfully filtered to match the same starting and ending points together. Slight inconsistencies can be observed in the matching of walls between each iteration. Windows had the effect of forming “shadows” in the lidar data where two measurements would be considered in equal value and drawn into the map. These inconsistencies can be properly filtered out if needed and can be disregarded as our main dataset will be generated outdoors. Apart from these interferences, the calibration can be deemed as functional with some minor tweaks to achieve full lidar-IMU rotation. With a proper setup configuration, we can continue with the proper dataset acquisition.

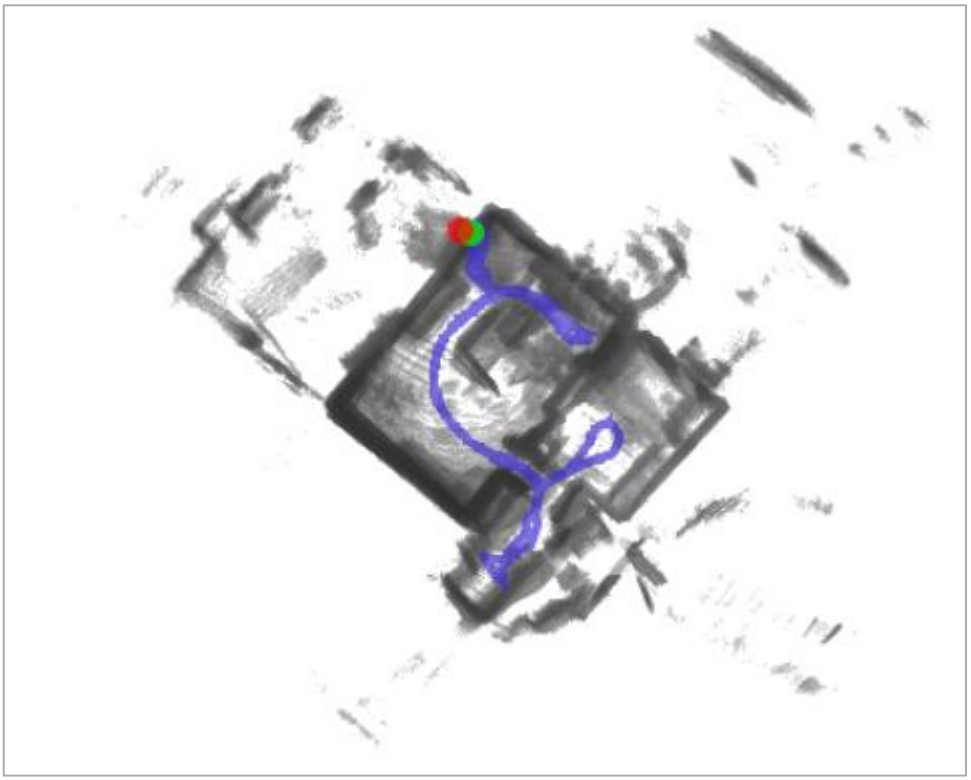

Figure 18: Generated map during initial testing

4.2.2. Dataset generation

The dataset will be generated from a medium voltage line (20 kilovolts), observable in Figure 19, as it is more beneficial to move from the smaller scale upwards to the bigger power lines with the point cloud density validation (>100Kv). We want to determine the viability to do maintenance check-ups on all the power line infrastructure, but the results from the smaller lines are most telling whether the point clouds are dense enough for this purpose.

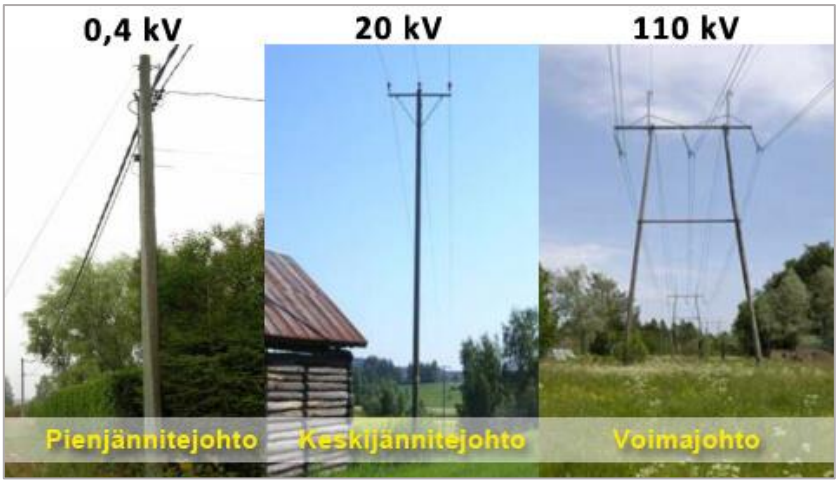

Figure 19: Examples of low, middle, and high-voltage power lines in Finland

The data gathering was done on-foot while following the direction of a power line for about 100 meters from approximately 10 meters to simulate the minimum safe distance that a drone could approach a power line while in operation. The 20 kilovolt line travels alongside of a bigger 400 kilovolt line. While not the main target, it would also be under evaluation if the lidar manages to detect it.

As previously seen in Subsection 2.1.3 (Figure 3), the lidar laser sweeps have increments of 2° between each channel. This causes the laser channel sweeps to be further apart from each other the longer the distance is to the measured object. This means that while we could efficiently detect the line closer to us, the bigger line would be too far from the sensor to be detected as a line, only receiving random bounces with too low density. To reduce the chance of power lines getting lost between the gaps of laser channels, the laser will be held in a 45° tilt to maximize the chance of laser bounces from the power line. The effect of this tilt will be taken into consideration in the lidar-IMU calibration.

Figure 20: Taken travel path while generating dataset

For loop-closure testing, the pre-planned path was travelled as a loop to allow the SLAM-algorithm to detect as many pre-known landmarks as possible, as seen in Figure 20. While loop-closing may not be viable during drone operation, it's amount of impact to the cohesion of our results may provide valuable information about the viability of the general approach to use SLAM-mapping in geo-referencing where loop-closing can't be continuously maintained (unless drones are set to fly pre-determined routes as a loop to achieve this). If a drone simply follows a line and never returns to the pre-seen landmarks, we can't achieve proper loop closure.

4.3. Running Cartographer

Now that we have generated a proper dataset for our test, we can use it as input for Cartographer. A ROS-node called *Cartographer_offline_node* is launched with our Rosbag containing the sensory data used as a parameter. Range data is measured from the vehicles point in multiple angles. This means that closer objects provide a lot of points, while far objects provide far less points. Subsampling is done to reduce the computational load of point handling. This cannot be done at random, since it would remove point from areas that have a low density of measurements. This is vital for us, since our point-of-interest can be expected to be low in density when compared to the vast amount of data generated outside of the area close to the power lines. To address this density problem, we can use a voxel filter that down samples raw points into cubes of constant size and only keep the centroid of each cube [Hess et al. 2016].

The node processes the Rosbag-data as fast as possible, creating submaps from the detected environment landmarks and links them together while the vehicle traverses the environment. This process can be observed in Figure 21: Each colour represents individual submaps that are generated in relation to an initial point. Each subsequent point is measured in relation to this first initial point that is determined when a new submap generation starts. While observing new features from the environment, the system closes a

submap and starts generating a new submap with a new colour. Each submap has its own coordinate frame origin, representing the first range scan data points position that was accumulated and inserted into it. Different colours help to illustrate this. The submap generation in relation to the initial coordinate frame origin can be observed from the points coloured in blue, where every point of the trajectory path is drawn in relation to the initial point at the bottom of the image. The submap coloured in red also uses some of these same positions in its generation. in Figure 21, the trajectory starts from the top of the image, travels to the bottom of the image and does a loop and returns to the initial starting position at the top and does another, smaller loop to increase submap fusion. The travelled trajectory can be better observed from Figure 22, with a green dot representing the starting point and a red dot representing the ending point of the trajectory. Finally, these submaps are fused together to form the completed map. We can also observe the amount of submaps generated from Figure 21. A submap is considered as complete when a given amount of range data is received. According to Cartographer documentation [Cartographer 2021], "submaps must be small enough so that the drift inside them is below the resolution, so that they are locally correct. On the other hand, they should be large enough to be distinct for loop closure to work properly". Resolution relates to the density of the point cloud: a high-resolution point cloud indicates a denser number of points.

Once a scan has been filtered and properly assembled from multiple range data, it is ready for the first algorithm step: the local SLAM. Local SLAM inserts a new scan into the current submap by using the calculation from the Cartographer-library's *pose extrapolator* that handles IMU data to predict where the next scan should be inserted into the submap. To avoid inserting too many scans into each submap, a motion filter is used to drop scans that haven't had enough movement between them. A submap can be considered complete when the amount of received data from the lidar exceeds a certain amount. The submaps store the data into a structure called probability grid. According to Hess et al. [2016], "scan matching starts by aligning far points of the low-resolution point cloud with the low-resolution hybrid grid and then refines the pose by aligning the close high-resolution points with the high-resolution hybrid grids". This way a map is slowly forming as the device traverses the environment.

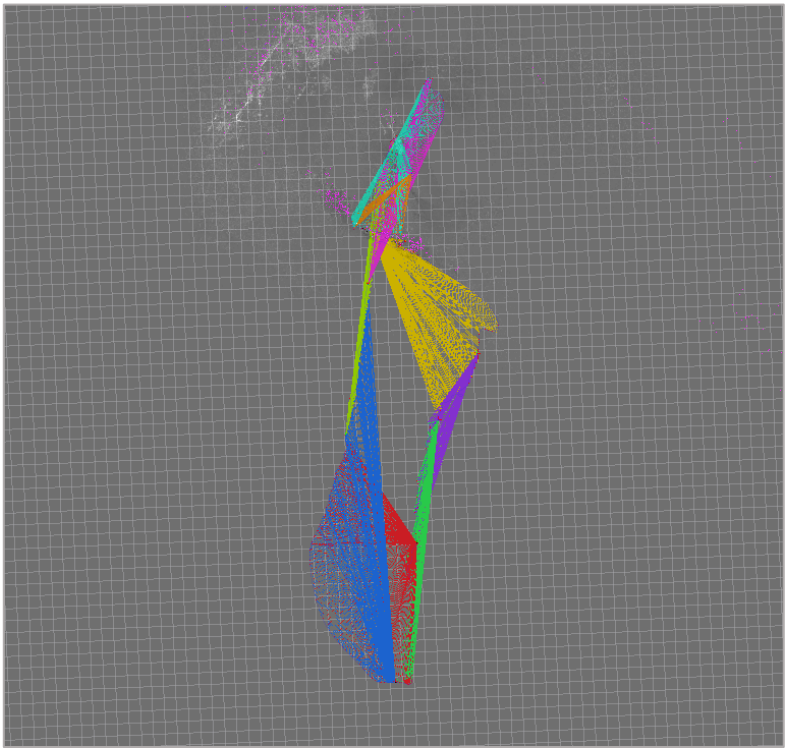

Figure 21: The vehicle trajectory with sub-map merging

Next step is the conduction of global SLAM. While local SLAM generates a succession of submaps, global SLAM runs optimization tasks in the background. The global SLAM's role is to re-arrange submaps as they are generated, eventually forming a coherent global map. The optimization can be triggered to run after a set number of batches are received. This is set as a suitable value for our dataset. Hess et al. [2016] describes that the global optimizer can be imagined as a set of little ropes tying the submaps together after constraints between submaps have been established, as previously mentioned in Section 3.2 describing different SLAM paradigms. This method of fusion falls within the category of graph SLAM algorithms, since it follows the same principles. These constraints are called nodes.

To keep the number of computations reaching too high, only a subsampled set of all close nodes are considered for constraints building. This is done to achieve the speed and robustness to achieve loop-closures in real time with online SLAM-mapping. These limiting factors can be also tweaked to benefit our use case, since we are running offline-slam and do not need to worry about computational speed as of now. This problem arises when online SLAM is possibly introduced at a later stage when a proper prototype is conducted after a successful proof-of-concept. As described before, the IMU has an inherent trait to always have some small error rate in its data. The global SLAM gives some flexibility to the IMU pose by running optimization steps between every submap. This can only correct some of the pose errors, as even 1-degree difference between the lidar and IMU pose causes accumulating drift as the trajectory extends.

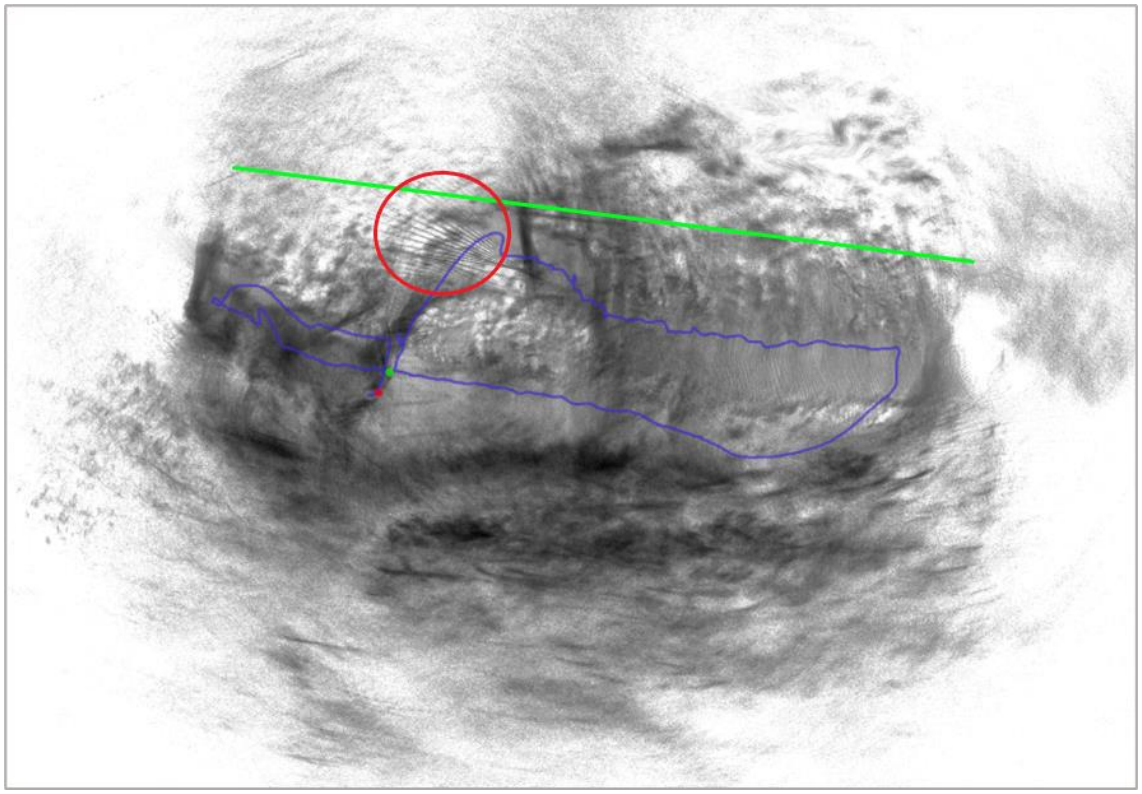

Figure 22: Vehicle trajectory and generated point cloud as an x-ray image

Once the trajectory is finished, Cartographer runs a new global optimization with more iterations than previous iterations. Especially during offline-SLAM as speed is not the main concern, this is beneficial in order to get a final polish to the result by running a large amount of global optimization iterations. In Figure 22, we can observe a point cloud generated from the fusion of separate sub-maps that were generated during the initial states of the algorithm. The blue line represents the travelled trajectory during data gathering, the green line represents the approximate position of the measured point cloud. In the red circle we can observe multiple power line representations that are mis-aligned by the algorithm. Evaluation of these findings is done in more detail in the next chapter.

5. Evaluation

The results and findings of the SLAM-analysis will be summarized and discussed in this chapter. Based on the generated dataset and trajectory, we can make some assumptions on the viability of the system to function as a maintenance tool for power lines. The main emphasis is the research question that asks whether it would be possible to use SLAM-mapping to generate dense and cohesive enough point clouds where we can detect the power line and, based on the generated point cloud, make possible maintenance analysis.

The second research question was whether the used scanners in this study could provide accurate and dense enough results for environment mapping. As the power lines are quite thin and need to be completely mapped in order to make any assumptions on their condition or possible damage, inconsistencies during mapping could lead to a false conclusion of the true state of the power lines. Although bigger objects obstructing the power lines could be effectively detected, smaller objects that still do damage could fall unnoticed. Also, when searching for snapped power lines, all the lines must be able to be mapped. Small inconsistencies can be filtered out, but bigger gaps between lines that are not damaged could compromise the credibility of the system.

The results will be evaluated by reviewing the accuracy of generated data, success of the device trajectory, success of generated submaps of point cloud data, and the success of global scan matching of many iterations of subsequent submap iterations. The search for inconsistencies is simple, as we know the pre-planned device trajectory and have a clear view of a sufficient, geo-referenced map. If a power line can be detected from the whole trajectory and be consistently fused together, we can then continue to evaluate the needed density to automatically detect a power line from the generated map. If this is not the case, we can further evaluate the viability of the system for this kind of use and the possibility of future improvements.

5.1. Result evaluation

Here, we will evaluate the results of our second georeferencing test that was done with data gathered from the outdoors area presented in the previous chapter. The results of the SLAM-mapping test were promising, but puzzling. The generated point clouds were dense, but cohesion was seriously lacking. Significant distortion between multiple sub-map iterations were clearly visible in the generated point cloud. This can be further observed in Figure 22. Inside the red circle, we can observe multiple different representations of the power line that we are trying to measure. Due to the mis-fused submaps, these point cloud representations of the power line are out of sync, since they are not properly aligned with each other.

In other words, the algorithm was unable to reference the device pose when fusing point clouds together. This can be further validated by the evaluation tools provided by the Cartographer package. These tools are used to assess SLAM results when no dedicated

ground truth is available, which is true in our case. We do not have a stationary reference point, so the system calculates its position dedicated by its previous known positions. This process comprises of two steps: auto-generation of "ground truth" relations, and test data evaluation against the generated ground truth. However, self-evaluation of this type does not comprise of the full accuracy of the SLAM-system, only an evaluation with real ground truth states can provide this, which we lack in this evaluation. Therefore, these metrics must be considered as directive rather than absolute. The transitional error observed between sub-map generation was more than 10 centimetres, and the rotational error more than a degree. This was clearly visible from the point cloud as two sequential submaps had the same landmarks drawn in multiple different positions near each other. These errors caused by accumulating drift can be observed inside the red circle in Figure 22. One cause to this may be the lack of distinguishable landmarks SLAM-mapping was run in open and forested area, since the distances between landmarks can grow unsustainably large and prevent Cartographer from effectively conducting scan-matching without any complex environments to be assumed as their own separate, unique landmarks. Cartographer is known to work well outdoors, especially when scanning complex environments that are much more favourable for submap generation and scan-matching than simple and repetitive environments [Hess et al. 2016]. In addition to the original dataset, additional datasets were recorded as separate Rosbags to better pinpoint the error causing overlap of landmarks during scan matching, repeating the same findings as the first dataset. As compared to the point clouds reached by Eriksson [2018], the point clouds significantly lacked cohesion.

Minor errors in Cartographer configuration and optimization are assumed to be playing a decisive role for the misaligned point clouds, as the IMU trajectory was far from ideal, despite showing promise in the initial testing. Although the device pitch that was held when conducting the dataset generations was accounted for in the configuration, the IMU trajectory suffered from continuous ascend in the y-axis, as seen in Figure 23. The rise of height is observed while traveling a straight path, so this constant elevation should not be present. This resulted in the submaps to lose the previous acquired landmarks and start generating new landmarks from scratch, without properly achieving global SLAM-matching.

Due to the challenging nature of Cartographer configuration, I couldn't find a perfect set of configurations to get a better result. The bad results were most likely caused by insufficient calibration of the relative orientation between the IMU and lidar. In other words, the lidar and IMU alignment wasn't marked accurately enough, which caused Cartographer to either assume a wrong position for the device, or a wrong pose. This was also apparent from the faulty IMU trajectory that should stay cohesive enough even when considering the small accumulating drift. A proper trajectory can be used to determine

successful calibration, which was done during the initial tests when mapping indoors. Cartographer is also included with its own pose optimizer that tries to correct these errors as they accumulate. The pose optimizer can also be adjusted to show information about the process, where correctional values can be seen. During our initial test, this correction was reduced to under one degree. As we mapped the second dataset, this correction rate rose to over five degrees. Therefore, it can be concluded that by increasing the calibration accuracy we could expect much more accurate results from the algorithm. Time restrictions and lack of skill played a part in hindering this process during calibration.

When evaluating the results, loop closures were not detected due to the continuous climb experienced by the sensors. More fine tuning of the optimizers may also provide better results but are difficult and laborious to perform since Cartographer has a myriad of different minor parameters and values to tweak. This means that Cartographer requires a good knowledge of the inner workings of the algorithm and the cause and effect between every parameter. This calls for more future work in finding a better set of configurations to achieve more cohesive results with more testing.

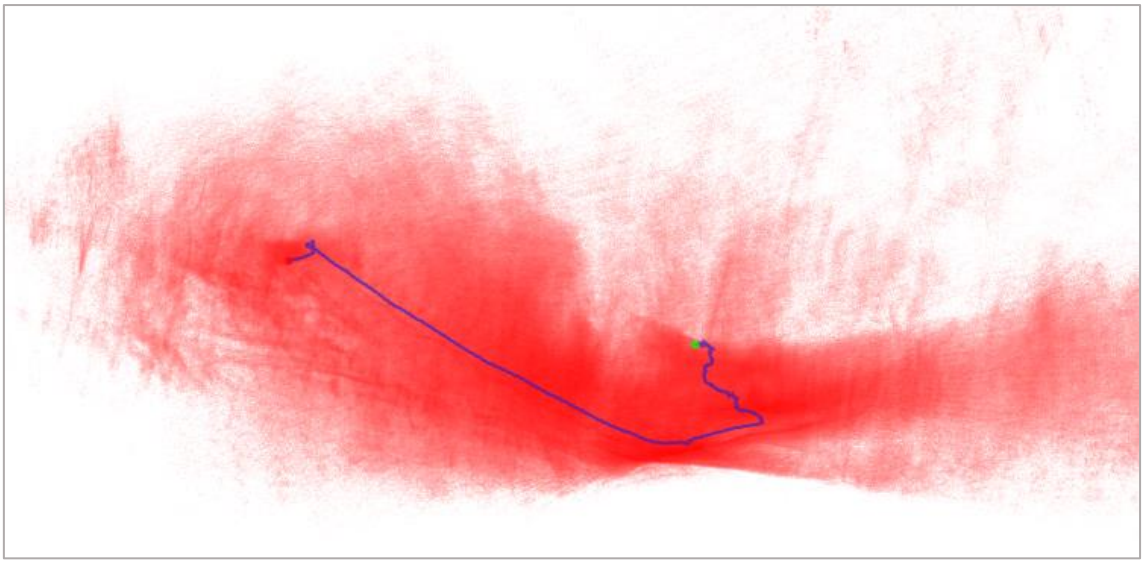

Figure 23: Example of accumulating drift in the y-axis

Improvements in the dataset acquisition could also be done, as holding the sensor by hand subjects the sensors to sudden and unpredictable movement that makes optimization and scan matching more challenging compared to a more rigid setup.

Point filtering also proved challenging. Without proper filtering, the point cloud grew too dense to distinguish any clear landmarks, for example, individual trees. This was due to the poor global SLAM results, as the submaps saw much overlap and lost the main reference to the previous landmarks. But if too much filtering was used, the power lines were undistinguishable from the less dense point cloud. More efficient point filtering

methods must be also applied in the future to better make sure the actual point-of-interest is clearly visible in the generated map.

We can assume from the generated dataset that additional sensors may be needed to generate a cohesive point cloud map representation in this kind of circumstances. Additional IMUs and lidars have proven to significantly enhance the map cohesion [Hess et al. 2016]. A GPS could also be introduced to the system to give additional reference to the sensors location, and thus aid in the scan matching, as used by Eriksson [2018]. Nevertheless, the system showed promise as the generated submaps showed somewhat adequate cohesion, as seen after the initial testing. When provided with an environment with enough complexity, Cartographer can be expected to work well with the used sensors.

5.2. Proof-of-concept result

In the past, successful geo-referencing has been utilized using lidar and IMU sensors with the aid of a SLAM-mapping algorithm [Karam et al. 2019]. Many commercial solutions exist that promise to reach the kind of results we were trying to reach. One big hurdle is that having a solution based in a closed system without the possibility to get to have your own impact to the solution may drive some operators away from these products. On top of that, they are usually much more costly. Therefore, one of our main research questions was to determine the possibility to use more cost-effective sensory hardware. Our question was that could this kind of sensory system be applied as a tool to monitor power lines. The system showed promise but requires more precise and throughout calibration and testing to provide sufficient results. Additional sensors may also be considered. However, a functioning groundwork has been established for future study to tackle the issues faced when conducting this research. Also, the affordable VLP-16 lidar and Xsens MTI-630 MEMS seemed to be good tools for the job, as the lidar density wasn't the main issue, and neither was the accuracy of the IMU. Both sensors are very cost-effective, making them desirable in the realm of proof-of-concept tools as they are not as a big of an investment and are also much more expendable compared to the sensors that cost many tens of thousands of dollars. Nevertheless, merging data from these sensors need more future work in order to generate a cohesive enough point cloud to conduct proper analysis.

6. Conclusions

In this thesis we successfully implemented a geo-referencing system with the aid of a lidar and an IMU sensor. To answer the two research questions presented in this thesis, two geo-referencing tests were arranged with the aid of a SLAM-mapper algorithm called Google Cartographer. Using the inertial data from the IMU and distance data from the lidar, we were able to generate two sets of point cloud maps from the environment. The first test was done to validate the system and the second test was done to gather results to be evaluated based on our research questions. Based on our results, we are able to come to the conclusion that SLAM-mapping needs more precise and careful calibration than used in this thesis. Although showing potential, we were not able to generate cohesive point cloud representations of the environment with the set of tools and parameters used in this thesis.

The submap-generation and global optimization in the Cartographer SLAM-algorithm were unable to be successfully utilized due to the poor calibration between the sensors. Efforts to improve the calibration were made, but in the end proved unfruitful in achieving better results. The algorithms in Cartographer require a better understanding from the user to be completely implemented. Nevertheless, we were able to successfully implement the algorithm in a smaller test environment, as shown by the initial test. The initial test results showed that, when configured correctly, the system has much potential in accurate geo-referencing. The viability for automated 3D point cloud detection is unable to be deemed as potential or not from these findings alone.

The system showed promise by validating the viability of used sensors to this kind of geo-referencing and evaluation process, but the means of SLAM-mapping need more knowledge and skill to be properly performed. Accurate 3D SLAM, both the offline and online methods, have many challenges if the goal is to generate point cloud maps that are as accurate and dense as possible. Especially with our goal, which was to find a very small area of interest amongst vast amounts of sensory data, the results of this thesis are inadequate to deem this approach as viable for commercial use. Nevertheless, the built system showed much promise and requires future work to be properly utilized. Especially the sensory calibration and algorithmic configuration require more research to be effectively ran for this kind of goal in mind. Researching the sensory systems and the SLAM-algorithm function have given me a solid understanding of the founding principles of geo-referencing and an introduction to robotics with the usage of Robot Operating System. This enables for a more throughout research in the future of the subject if the research project takes its next step of moving towards a more commercialized use. But as for now, this seems unlikely as the findings do not yet support it.

The work proved to be a good set of challenges and forced me to learn many new things from implementing new systems, conducting tests and evaluating test findings.

The most time-consuming part was the actual implementation. I found the Google Cartographer to be very complex and hard to properly implement, requiring tons of research in conjunction with the ROS-framework to even get it running and exchanging data between the sensors, even without considering any kind of calibration or configuration tweaking. Information about the implementation steps were sparse and far in between, having to be found from different Internet forums from people who have experienced the same trial-and-error steps as I did, and gradually pushed towards a working setup. Robotics is a challenging branch of computer science and engineering, but the extensive research and hard work of robotics engineers and software developers has made it more accessible than ever before. Thanks to these people and their hard work, this thesis has been able to be created.

My work has enabled me to do more precise research in the future regarding geo-referencing, SLAM-mapping and the overall automation process development of critical infrastructure maintenance. Other uses for geo-referencing via SLAM-mapping can also be potentially implemented in the future as more use-cases are found for the technology.

7. Future work

The most important future work step would be to adequately complete the entire sensory calibration in Cartographer. The inaccurate calibration was found to be the most influential part to the results seen in this research. Future work could also consist of adding additional sensors to the system. By adding another lidar or an IMU we could expect better results for the geo-referencing process. Also, a GPS-system could aid the system to have a better understanding of its trajectory. Another future work step would be a more careful and wide study of the used algorithms to truly untap the potential they have. Overall, the power lines were found to be too small to be adequately detected by our system. The system could be better tested when attached to an actual drone rather than generating the dataset on-foot with a hand-held system. This way we could more efficiently filter unnecessary data as noise and get to record data from the air, above the power line. This could also reduce the sensory noise affecting the IMU, although flight could pose its own set of challenges from the perspective of inertial measurement.

In terms of software, many geo-referencing applications and SLAM-mapping algorithms exist that could provide better results than Cartographer. By performing testing with other SLAM-algorithms, such as HectorSLAM, FastSLAM or LOAM, we could have a better understanding of the different options within the SLAM research field. ROS could also be superseded for a more stand-alone solution.

References

Andrejašic, Matej. "Mems accelerometers." *University of Ljubljana. Faculty for mathematics and physics, Department of physics, Seminar*. 2008.

Awrangjeb, M., Gao, Y., & Lu, G. (2018). Classifier-Free Extraction of Power Line Wires from Point Cloud Data. 2018 Digital Image Computing: Techniques and Applications (DICTA), 1-7.

Bailey, Tim, and Hugh Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II." *IEEE robotics & automation magazine* 13.3 (2006): 108-117.

Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., ... & Silva, C. T. (2017, January). A survey of surface reconstruction from point clouds. In *Computer Graphics Forum* (Vol. 36, No. 1, pp. 301-329).

Biström, Benjamin. "Comparative analysis of properties of LiDAR-based point clouds versus camera-based point clouds for 3D reconstruction using SLAM algorithms." (2019) (Master's Thesis, Åbo Akademi, Turku, Finland).

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., ... & Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, *32*(6), 1309-1332.

Chatfield, A.B., 1997. *Fundamentals of high accuracy inertial navigation*. American Institute of Aeronautics and Astronautics.

Deilamsalehy, H. and Havens, T.C., 2016, October. Sensor fused three-dimensional localization using IMU, camera and LiDAR. In *2016 IEEE SENSORS* (pp. 1-3). IEEE.

Des Bouvrie, S.L., 2011. Improving rgbd indoor mapping with imu data.

Diaz, E. M., de Ponte Müller, F., Jiménez, A. R., & Zampella, F. (2015, March). Evaluation of AHRS algorithms for inertial personal localization in industrial environments. In *2015 IEEE International Conference on Industrial Technology (ICIT)* (pp. 3412-3417). IEEE.

Dissanayake, G., Sukkarieh, S., Nebot, E. and Durrant-Whyte, H., 2001. The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications. *IEEE transactions on robotics and automation*, *17*(5), pp.731-747.

Droeschel, D. and Behnke, S., 2018, May. Efficient continuous-time SLAM for 3D lidar-based online mapping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1-9). IEEE.

Dudek, G., and Jenkin, M., "Inertial Sensing, GPS and Odometry", in Springer Handbook of Robotics, B. Siciliano, O. Khatib, Eds. Heidelberg, Berlin: Springer-Verlag, 2016, 737 –751.

Durrant-Whyte, H. and Bailey, T., 2006. Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine*, *13*(2), pp.99-110.

Eriksson, Peter. "Evaluation of Low-Cost INS and Forming a Georeferenced 3D Point Cloud." (2018) (Master's Thesis, Åbo Akademi, Turku, Finland).

Estefo, Pablo, et al. "The robot operating system: Package reuse and community dynamics." *Journal of Systems and Software* 151 (2019): 226-242.

Fingrid Maintenance [Online] (2021) Available: *https://www.fingrid.fi/en/grid/maintenance/*

Glennie, C. L., Kusari, A., & Facchin, A. (2016). CALIBRATION AND STABILITY ANALYSIS OF THE VLP-16 LASER SCANNER. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, *9*.

Glennie, C., Lichti, D.D., 2010. Static Calibration and Analysis of the Velodyne HDL-64E S2 for High Accuracy Mobile Scanning. Remote Sensing 2, 1610-1624.

Google Cartographer [Online] (2021) Available: *https://google-cartographer.readthedocs.io*

Grewal, M.S., Weill, L.R. and Andrews, A.P., 2007. *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons.

Hepta Airborne ltd, [Online] (2021) Available: *https://heptaairborne.com/case-study/elektrilevi/*

Hess, W., Kohler, D., Rapp, H. and Andor, D., 2016, May. Real-time loop closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1271-1278). IEEE.

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016, May). Real-time loop closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1271-1278). IEEE.

Swinburne, R. (2004). Bayes' Theorem. *Revue Philosophique de la France Et de l*, *194*(2).

Kannan, Krishnaswamy. "Development of a reference software platform for the Velodyne VLP-16 LiDARS." (2016) (Master's Thesis, KTH Royal Instutute of Technology, Stockholm, Sweden).

Karam, S., V. Lehtola, and G. Vosselman. "INTEGRATING A LOW-COST MEMS IMU INTO A LASER-BASED SLAM FOR INDOOR MOBILE MAPPING." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* (2019).

Khan, Aragão. "A UAV–lidar System to Map Amazonian Rainforest and Its Ancient Landscape Transformations." International journal of remote sensing 38.8-10 (2017): 2313–2330. Web.

Konolige, Kurt, and Andreas Nüchter. "Range Sensing." *Springer Handbook of Robotics*. Springer, Cham, 2016. 783-810.

Li, P., 2010. Selecting and using virtualization solutions: our experiences with VMware and VirtualBox. *Journal of Computing Sciences in Colleges*, *25*(3), pp.11-17.

Li, R., Liu, J., Zhang, L. and Hang, Y., 2014, September. LIDAR/MEMS IMU integrated navigation (SLAM) method for a small UAV in indoor environments. In *2014 DGON Inertial Sensors and Systems (ISS)* (pp. 1-15). IEEE.

Murphy, K.E., Johns Hopkins University, 2004. *Light detection and ranging (LIDAR) mapping system*. U.S. Patent 6,711,475.

Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B., 2002. FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, *593598*.

Newman, Paul, and Kin Ho. "SLAM-loop closing with visually salient features." *proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005.

Nüchter, A., Bleier, M., Schauer, J., & Janotta, P. (2017). Improving Google's Cartographer 3D mapping by continuous-time slam. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*, *42*, 543.

Pierzchała, M., Giguère, P. and Astrup, R., 2018. Mapping forests using an unmanned ground vehicle with 3D LiDAR and graph-SLAM. *Computers and Electronics in Agriculture*, *145*, pp.217-225.

Qian, C., Liu, H., Tang, J., Chen, Y., Kaartinen, H., Kukko, A., Zhu, L., Liang, X., Chen, L. and Hyyppä, J., 2017. An integrated GNSS/INS/LiDAR-SLAM positioning method for highly accurate forest stem mapping. *Remote Sensing*, *9*(1), p.3.

Quigley, Morgan, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).

Stachniss, Cyrill, John J. Leonard, and Sebastian Thrun. "Simultaneous localization and mapping." *Springer Handbook of Robotics*. Springer, Cham, 2016. 1153-1176.

Tang, J., Chen, Y., Kukko, A., Kaartinen, H., Jaakkola, A., Khoramshahi, E., Hakala, T., Hyyppä, J., Holopainen, M. and Hyyppä, H., 2015. SLAM-aided stem mapping for forest inventory with small-footprint mobile LiDAR. *Forests*, *6*(12), pp.4588-4606

Thrun S. (2007) Simultaneous Localization and Mapping. In: Jefferies M.E., Yeap WK. (eds) Robotics and Cognitive Approaches to Spatial Mapping. Springer Tracts in Advanced Robotics, vol 38. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-75388-9_3

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. "A probabilistic approach to concurrent mapping and localization for mobile robots." *Autonomous Robots* 5.3 (1998): 253-271.

Velodyne Lidar, Inc. [Online] (2021) Available: *https://velodynelidar.com*

Velodyne ROS Package [Online] (2021) Available: *http://wiki.ros.org/velodyne*.

Venugopal, Vivek, and Suresh Kannan. "Accelerating real-time LiDAR data processing using GPUs." *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2013.

Waldron, Kenneth J., and James Schmiedeler. "Kinematics." *Springer handbook of robotics*. Springer, Cham, 2016. 11-36.

Woodman, O.J., 2007. *An introduction to inertial navigation* (No. UCAM-CL-TR-696). University of Cambridge, Computer Laboratory.

Welch, Greg, and Gary Bishop. "An introduction to the Kalman filter." 1995. 127-132.

Zhang, H.; Liu, Y.; Tan, J. Loop Closing Detection in RGB-D SLAM Combining Appearance and Geometric Constraints. *Sensors* 2015, *15*, 14639-14660. *https://doi.org/10.3390/s150614639*