Autonomous object mapping using UAV-based remote sensing

Master of Science Thesis
University of Turku
Department of Future Technologies
Software Engineering
2020
Tero Yrjölä

UNIVERSITY OF TURKU
Department of Future Technologies

TERO YRJÖLÄ: Autonomous object mapping using UAV-based remote sensing

Master of Science Thesis, 50 p.
Software Engineering
May 2020

Safety plays a huge role in our world. As technology advances, we can harness it to negate risks in our everyday life. This thesis is made for a project by Työtehoseura, who are responsible for Finland's yearly warehouse pallet rack inspections. Their upcoming innovation is to automate this inspection process by introducing commercial quadrocopters capable of autonomous defect detection of these warehouse shelf structures. With University of Turku as a contractor for the project, in this thesis we build a digital playground for the team to experiment with. The simulation allows a fast, risk-free testing of different sensors, algorithms, drone models, warehouses and pallet racks. In this thesis we evaluate different existing methods in robot simulation, pathfinding, collision avoidance and simultaneous localization and mapping. As no suitable existing solutions for the problem are found, in this thesis a solution built on top of Microsoft AirSim drone navigation framework is offered. The proposed solution, and its components, are presented, explained and illustrated. The end result is a simulated quadrocopter equipped with a 3D-lidar sensor cabable of mapping shelf structures of various shapes and sizes. A demonstration video of this simulation will be used as a promotion material for the project.

Keywords: airsim, artificial intelligence, gazebo, lidar, mapping, pathfinding, quadrocopter, robot, simulation, slam

Contents

List of Figures

1 Introduction

Safety and risk management is a priority in almost everywhere. Those two points are the foundation of this thesis and the project this paper bases on. There are many businesses with warehouse storage units which do not meet the minimum safety regulations. Työtehoseura (TTS) is the company behind Finland's yearly inspections, and they have decided to ease the process of these inspections for both, the warehouse owners and TTS itself. Currently, the inspections are made by physically visiting each warehouse and examining every shelf individually. This is a burdensome process, even more so when taking into account the limited amount of TTS certified inspectors. That is why TTS is working on a solution of ensuring the robustness of these warehouse pallet racks using an autonomous UAV. Working together with the university of Turku, TTS plans to build a commercial quadrocopter capable of mapping, detecting defects and determining the sturdiness of a heavy-duty shelf. This thesis aims to aid in the project by building a simulation of the case described. This simulation will make the decision-making process more convenient as testing can happen much faster, cheaper and without the real life risks involved. In this thesis we will compare already existing methods of UAV mapping techniques and robot simulation frameworks.

1.1 Thesis structure

In this first chapter, we will briefly explain the problem and introduce the solution-to-be-built. We will also go through the whole thesis structure and announce the thesis research questions.

Autonomous drones and different mapping methods have been studied a lot before, yet a suitable and budgetary commercial product suitable for this case, is not available. These existing techniques and products, such as object mapping, pathfinding and robot simulators, are introduced in Chapter Two. The origin for the need for the product is introduced in Chapter Three, which presents the background details, goals and the affiliates of the whole project.

Chapter Four begins to address the simulation and discusses the different alternatives for the implementation of the simulation. Different simulation frameworks, sensors and configurations are discussed in this chapter. The next chapter, Chapter Five, digs deeper into the simulation world. In this chapter we begin to implement the simulation using the methods discovered in the previous chapter. We also present the result of the simulation.

After the simulation has been processed, the thesis advances to Chapter Six, which constains a conclusion for the project on my part. Finally Chapter Seven wraps up the thesis and shares some personal thoughts about the project, the simulation and the thesis.

1.2 Research questions

The research questions of the thesis are

1. What is a satisfactory simulation environment for a project of this kind?
2. What is the optimal setup for the simulated drone in terms of sensors?
3. How to efficiently map a shelf and detect possible deformations in it?
4. How applicable to the real world the solution in question is?

*RQ*1 (research question one) is a question about the tools for the project. We will be comparing different frameworks and techniques in order to find the optimal environment for the project.

For RQ2 we will be evaluating different sensors while keeping note of the budget. The sensors will be configured to match their real world counterparts.

The RQ3 will be thoroughly answered when we start planning and building the simulation itself. For the damage detection, we might need to resort to image recognition software.

The final research question, RQ4, will be discussed when evaluating different simulation frameworks, and hopefully validating the answer to this RQ by actually applying the solution to the real world.

These research questions will be explicitly answered in Chapter 6.

2 Autonomous drones and simulation techniques

2.1 Independent navigation

As technology advances, everything is becoming more autonomous, including vehicles. This goal of autonomous pathfinding brings many challenges and it has been researched a lot[1][2][3]. Successful path planning and navigation of autonomous robots requires a reliable estimation of the current location, and a sufficiently accurate presentation of the surroundings. When considering our challenge of navigating an unmanned aerial vehicle (UAV) in an indoors area, we can break the task down into three sections:

1. Self positioning and tracing the surroundings in an unknown environment.
2. Finding an efficient route from point A to point B.
3. Evading possible physical contacts with other objects.

These three points are discussed in the following subchapters.

2.1.1 Simultaneous localization and mapping (SLAM)

When initially dealing with an unknown environment, the autonomously navigating robot needs to have some awareness of its local position. This position awareness is even more important due to the fact that the drone uses an internal local coordinate system (discussed

more in 4.2.1) to navigate in front of the pallet rack. There has been studies of tracking the drone's position using only local data [4] but as we need more precision, an enhanced position tracking is required. This is where SLAM comes to play. It is a way for the robot to incrementally build a consistent map of its environment while simultaneously determining its location within this built map [5]. This fits well into the project as the pallet rack has to be mapped anyway for further usage.

The problem of mapping the surroundings while concurrently using this depiction to position itself sounds paradoxical, almost like the famous chicken-and-egg problem. The first mention of the SLAM problem was in 1986 IEEE Robotics and Automation Conference. During the next few years there were some key papers produced which proved the problem to be, at least to some extent, solvable. [5].

Graph SLAM

To demonstrate a SLAM algorithm, we will go through a simple graph SLAM method. In the figure 2.1 we can see an illustration of an autonomous drone moving from point 0 to point 1. The actual distance travelled parallel to the x-axis is z and therefore the drone's location at point 1 is (x_0 + z, y_0).

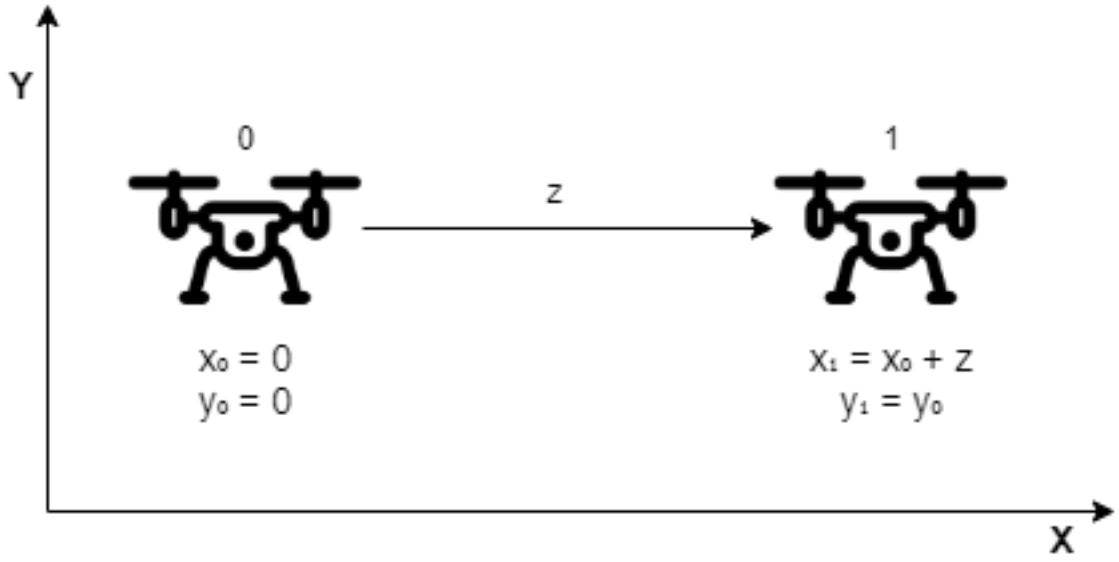
But what happens in reality is that the location tracking is not absolute, so it's better to say that the actual location after motion z would be a Gaussian centered around the predicted point 1, so x and y coordinates would, in order, be

$$\frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}\frac{(x_1 - x_0 - z)^2}{\sigma^2}) \tag{2.1}$$

$$\frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}\frac{(y_1 - y_0)^2}{\sigma^2}). \tag{2.2}$$

The product of these two Gaussian is now our constraint. Constraints in graph SLAM are the initial location, relative motion constraints and relative measurement constraints.

Figure 2.1: A non realistic example of perfect drone self-location tracking

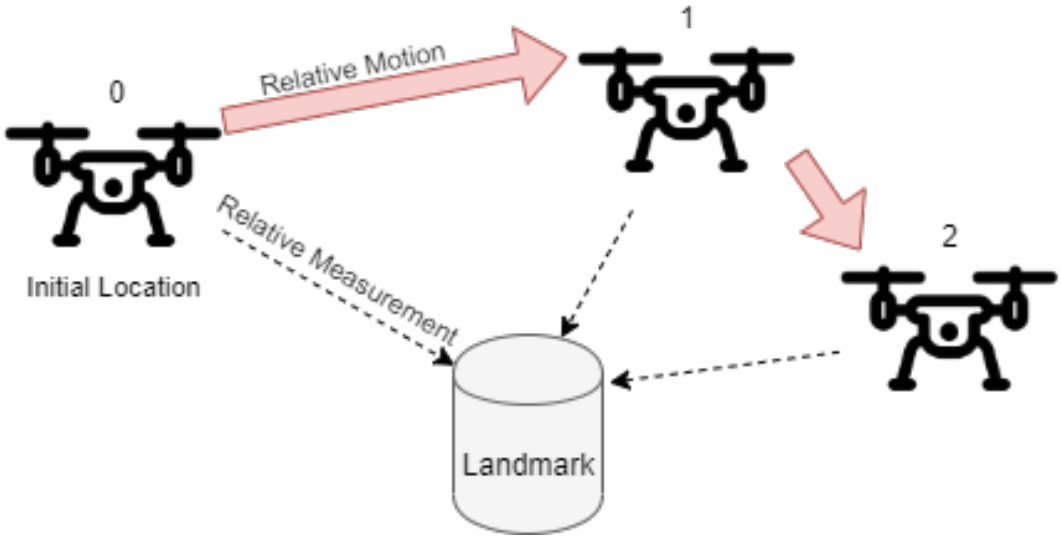


Relative motion constraints are constraints that compare the drone's position between previous and next positions, and the relative measurement constraint is a vector to a certain landmark visible from multiple position. Graph SLAM collects these constraints and finds the most likely path within them along with the location of the landmark. This how the drone is able to keep track of its location in a previously unknown environment using the SLAM technique. [6][7]

Now when the drone has a static landmark as a reference point, it can map its surroundings based on this known point and save this point cloud data. Every scanned point cloud point is some kind of a vector from this landmark. When advancing further, the drone can specify more landmarks (and calibrate these new points based on the previous landmarks). This way it will slowly build a representation of the surrounding environment.

The traditional SLAM algorithm works well when you are, for example, moving indoors and need to map your surroundings [8]. In our case (more information about the specified problem is introduced in Chapter 3), where the actual navigation from the dock

Figure 2.2: A figure showing the constraints of a graph SLAM process



to the pallet rack is not within the scope of this simulation, we have no need to map the warehouse itself. To fulfill our goals, it is necessary to only map the shelf in front of the drone and localize the drone relative to the shelf itself. This suggests that the drone should be able to keep track of its location based on what its camera or sensors detect in front of it. The detected data is then compared with the mapped information, containing multiple landmarks, to reassure that the drone is still in the same position it should be. This combined with the drone's inner estimated position should give us the required accuracy for the task. One aspect that needs to be taken into account is the quality of the mapped information. Accuracy is a big factor because if the drone locates itself mistakenly, then most of the mapped structure is unusable and the shelf cannot be mapped.

2.1.2 Pathfinding

Pathfinding is the act of finding the shortest route between two points while being blocked by different obstacles. There are many different pathfinding algorithms, each with their own good use cases [2][3]. Graph search algorithms, such as the A*, use a graph with a set of location nodes and a set of connection between these nodes (in a grid-map, a connection is between two adjacent nodes). The algorithm aims to find the shortest path between two given nodes using the connections. The result can vary depending on the algorithm used. There are many different variations of pathfinding algorithms, as an example we are going to cover breadth first search and greedy best first search algorithms.

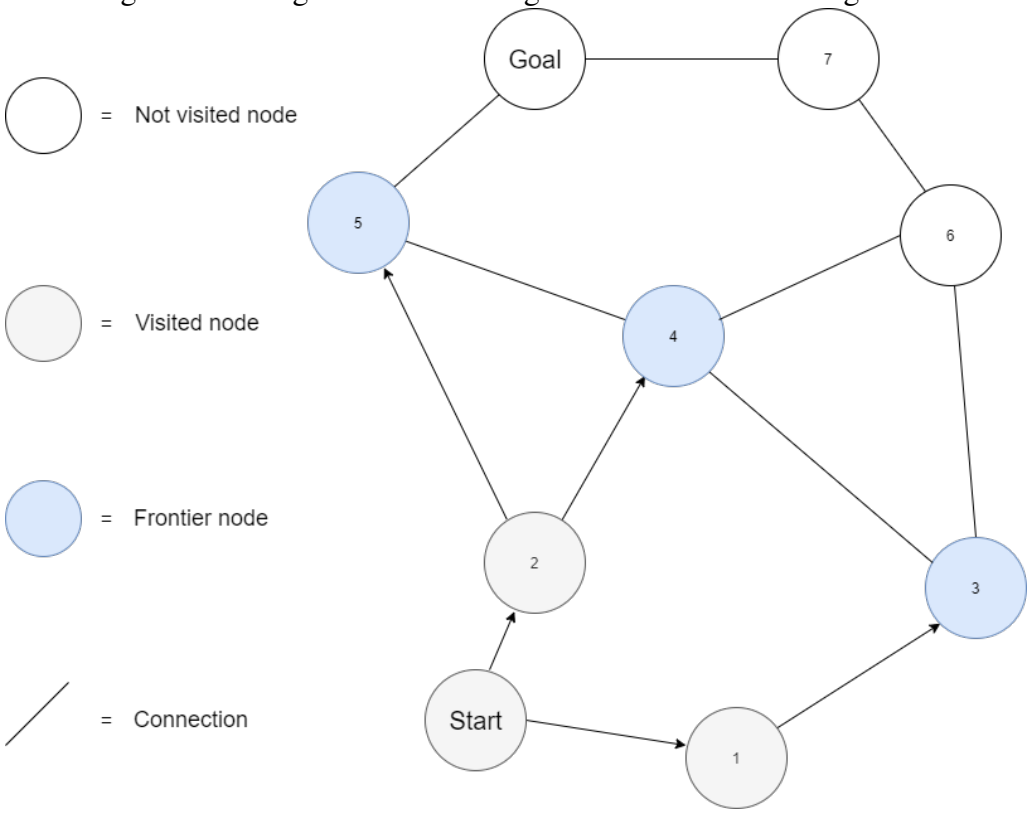
Breadth first search

As in all of these three algorithms, in a breadth first search, we will begin our search from one of the given nodes, called the start-node. Then the algorithm will start going through each of the neighbour-by-connection nodes beginning from the closest one from the start. Each of these locations are now added to a list called the Frontier (as well as to a list called Visited). After that, the process becomes iterable:

1. Pick a location from Frontier.
2. Remove the selected item from Frontier
3. Add all of its non-visited (not present in Visited) neighbours to the Frontier and the Visited lists.
4. If a neighbour is the goal-node, and we are using early exit, stop and return the path to the goal-node.
5. Repeat until all the nodes have been processed.

In the example figure 2.3, we have processed the nodes start, 1 and 2. Now the next step would be to take one of the Frontier nodes (3, 4 or 5) and process their neighbouring nodes

Figure 2.3: A figure demonstrating a breadth first search algorithm



in order. After reaching node 5, we can see that is has connections to nodes 2, 4 and Goal. We can see that nodes 2 and 4 have already been visited. This leaves us with the goal-node, which is our target. Now if we want to stop here, we have a somewhat optimized path from start to the goal-node: start, 2, 5, goal. If however we want to process the whole map and calculate paths to every node, we can continue until the last two nodes 6 and 7 are also processed.

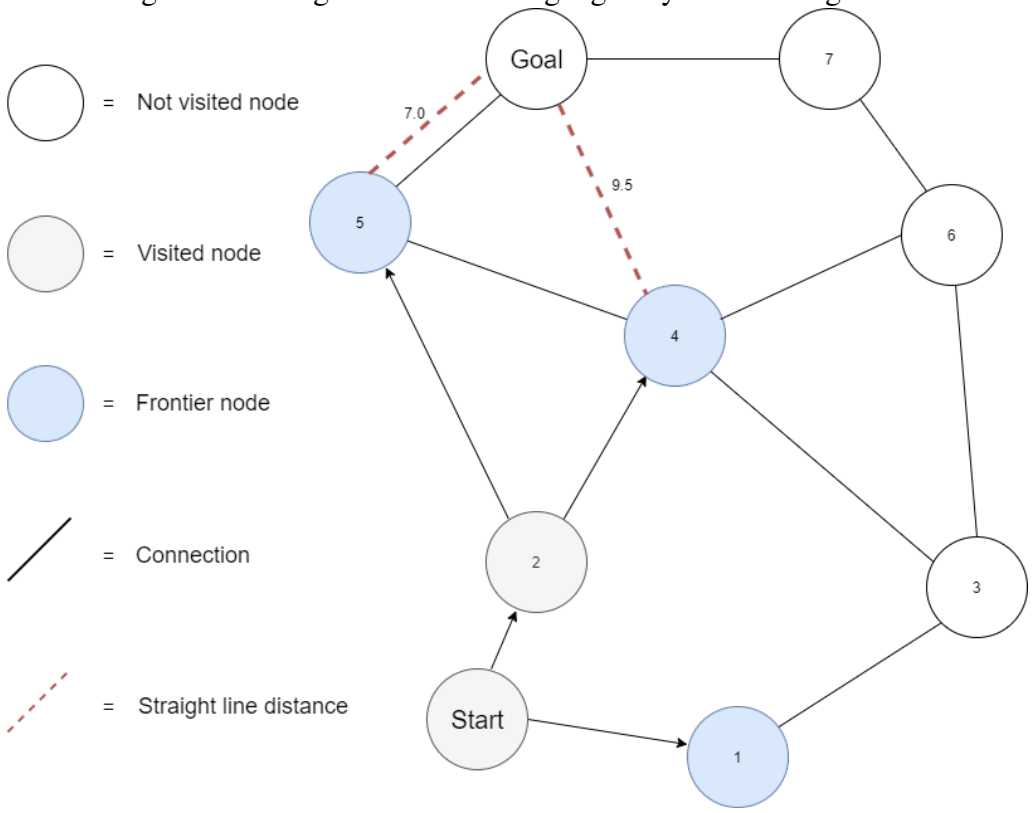
Greedy best first

In breadth first search algorithm, the Frontier expanded equally in all directions. This quickly becomes extremely inefficient in big open areas. This is where we can use a heuristic called greedy best first. The method resembles the Djikstra algorithm[9] and the previously mentioned breadth first search. In both algorithms we have a new variable called cost, which keeps track of the movement costs to a certain node from the start node.

In addition, the algorithms introduce a priority queue, which determines the direction where the Frontier expands to. In Djikstra, we simply expand to the node with the lowest cost next (this way we can have different costs to different neighbouring nodes too!). But in greedy best first, we can use a heuristic where we calculate the distance from the processed node to the goal-node. This distance calculation can be as simple as measuring the straight line distance to the target. Then we will expand the Frontier always to the node with the lowest distance.

The previous graph 2.3 processed with a greedy best first can be seen in the figure 2.4 on the next page. In the figure we have arrived at a situation where we need to determine whether to process node 1, 4 or 5 next. As we have already calculated each of their distance from the goal-node individually (node 1 distance not displayed in the figure), we will just pick the lowest distance, in this case it is the node 5. We will once again arrive at the goal-node using the same path start, 2, 5, goal. However, when compared to the breadth first search, we have processed only five nodes, whereas the breadth first search had to process eight nodes. Of course, this is situational but it shows the differences between these two algorithms.

Figure 2.4: A figure demonstrating a greedy best first algorithm



There are lots of different variations of these methods[1], shown in figures 2.5, 2.6, 2.7 and 2.8. These figures were built using an open-source pathfinding visualization tool[10]. In the figures the cyan color represents a visited node, light green a Frontier node, green is the start-node, red is the goal-node and yellow is the path travelled.

These algorithms, however, are aimed to finding the shortest path in a known (and usually static) environment. Our case differs from that because we are mapping the surroundings ourselves and the environment itself can be dynamic, so we have to be adaptive. As previously mentioned in 2.1.1, we are not going to require thorough indoor quadrocopter pathfinding [11]. A small pathfinding optimization will be required when we have mapped the outmost beams of the pallet rack in front of the drone (chapter 5.3).

Figure 2.5: A* algorithm. This algorithm uses a combination of best first's distance-to-goal estimation and breadth first's distance-from-start. Found the optimal path of 31 steps in 375 operations.

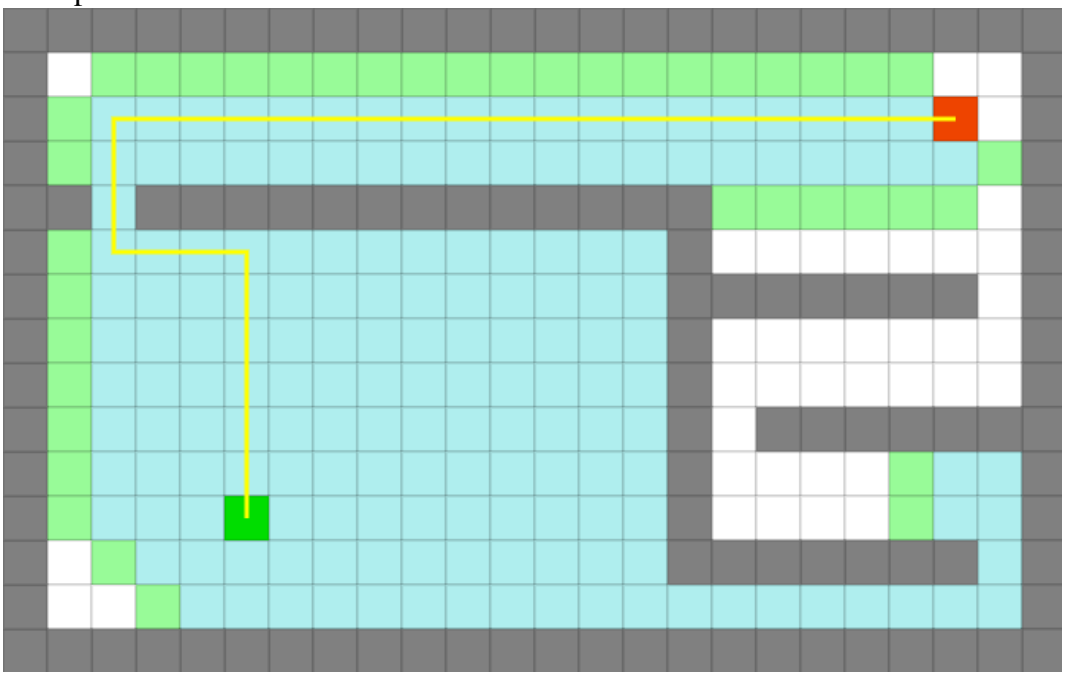


Figure 2.6: Best first search algorithm. Found a path of 55 steps in 210 operations.

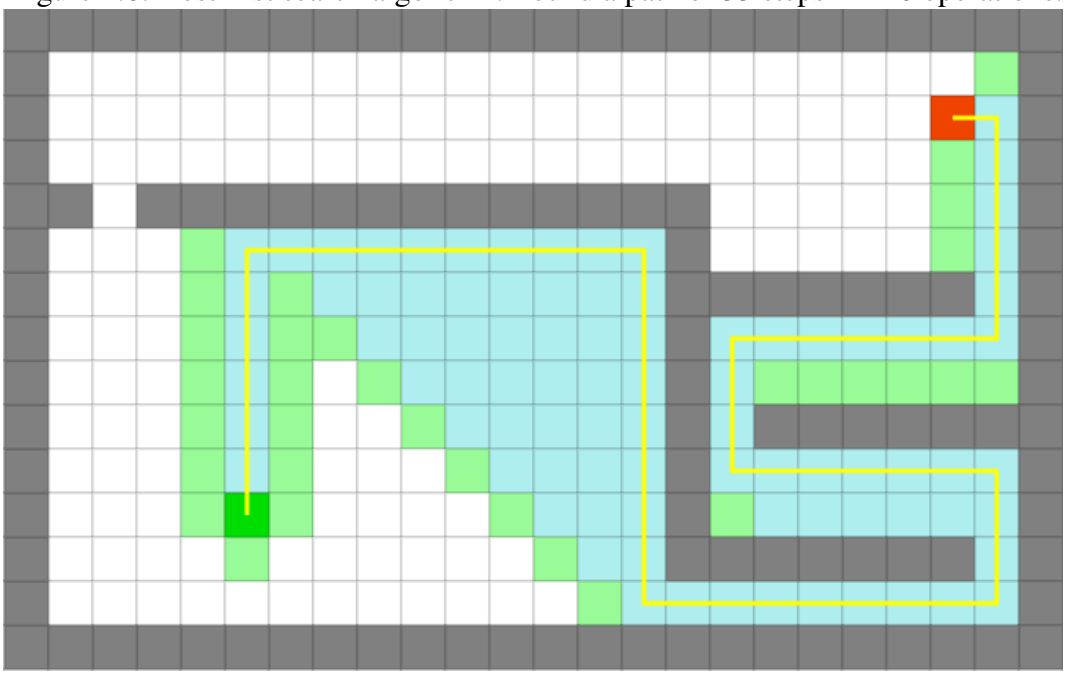


Figure 2.7: Breadth first search algorithm. Found the optimal path of 31 steps in 452 operations.

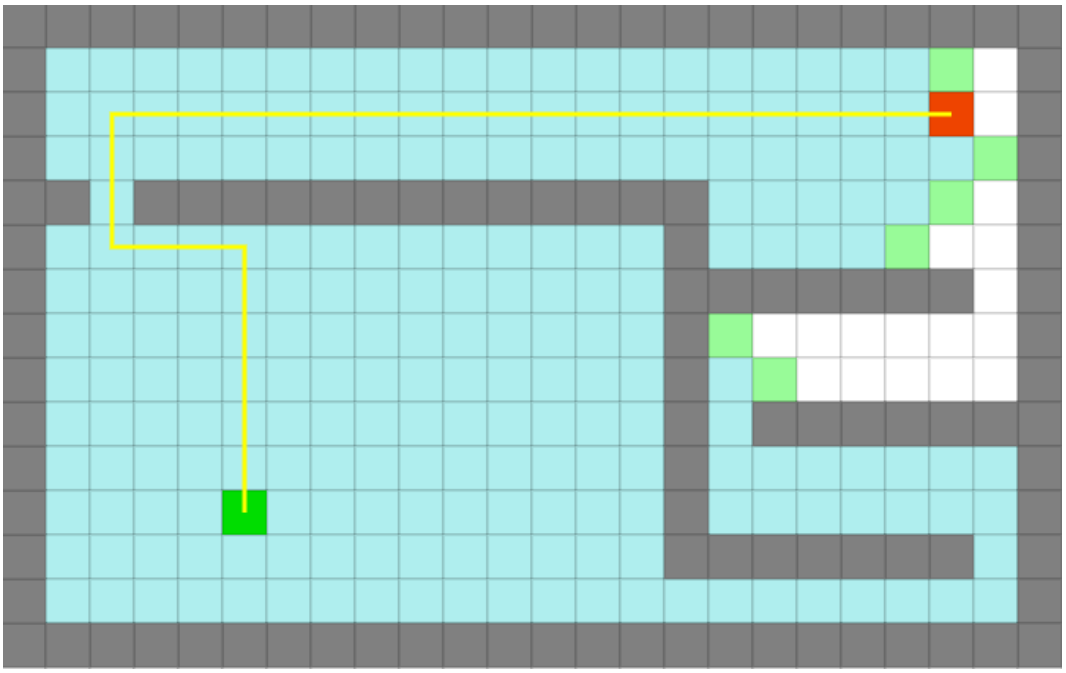


Figure 2.8: Djikstra algorithm. Found the optimal path of 31 steps in 459 operations.

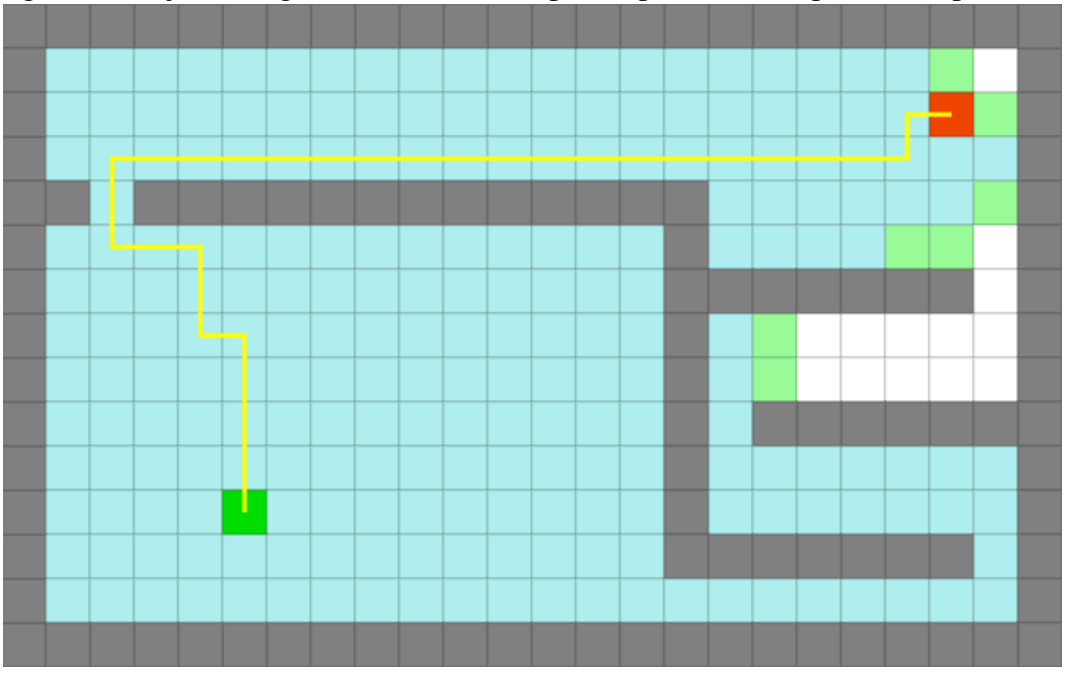
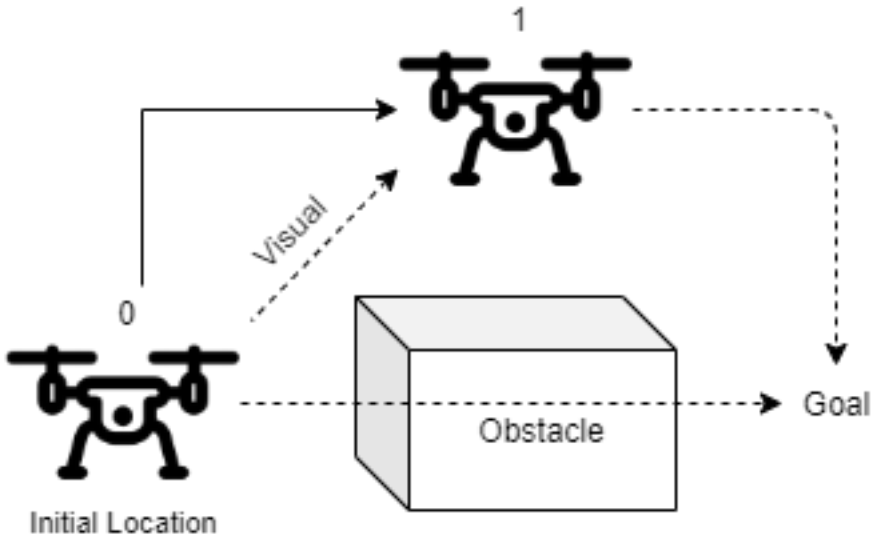


Figure 2.9: A simple demonstration of collision avoidance



2.1.3 Collision avoidance

As mentioned in the Introduction, safety is a big factor in our project. This is why the drone needs to be aware of the possibly moving objects around it. Since unmanned aerial vehicles (UAV) are becoming more popular, collision avoidance is becoming more relevant [12]. In the context of aerial collision avoidance, it is important to keep a safe distance to close-by objects.

When the drone is mapping the shelf, it is essential to detect any objects blocking the current path. This can be done using various different sensors like image or distance sensors. After this object is detected, the drone needs to create an alternative path to its destination, or abort the current route completely.[12]

In the simple example figure 2.9 we can see the drone in its initial location 0. It has been given a task of navigating to the Goal-area but it has been blocked by an obstacle. As the obstacle is stationary and the drone has a visual of another route past the obstacle, it can approach the Goal using a different route. After advancing to location 1, a new evaluation of whether to advance is required. When dealing with more complex routes or obstacles, the drone will need to have clear instructions on whether to try going around

the obstacle or just concede the route.

2.1.4 Combining the elements of independent navigation

The drone needs to be able to bring all three elements mentioned in the last three subsections together; SLAM, pathfinding and collision avoidance. When encountering an obstruction, first the quadrocopter has to determine whether the object should be bypassed (this depends on how the project advances and what are the requirements for the mapping) and if so, the drone needs to go find a good path around the object while maintaining safety, keeping track of the local position and being efficient in terms of computation time [13]. We can take advantage of the methods discussed in the previous subchapter 2.1.2 in order for the drone to navigate back to the desired route. If this seems impossible (whole path is blocked) then the drone should safely navigate back to the starting position and report this incident.

2.2 Virtualizing real life scenarios

A simulation is by definition an approximation or an abstraction of the real world. As mentioned before in the introduction (Chapter 1), in certain cases simulating the environment makes experimenting much more efficient in terms of time, budget and risks. Although the initial set up might be arduous, in our case it is worth the effort. The method of simulating robots has been researched a lot and the earliest studies date back in the 1980-century [14]. Since then, the industry sector has advanced in terms of performance, accuracy, complexity (while maintaining usability) and visual looks.

Today, there are countless different simulation frameworks for multiple different purposes. This thesis inspects two robot frameworks, Microsoft AirSim [15] and Gazebo [16]. Microsoft AirSim is a simulator that runs on Unreal Engine[17] or Unity[18] and it currently supports the simulation of drones and ground vehicles. It allows direct input control as well as controlling the vehicles programmatically. Gazebo on the other hand is a broader robot simulator which supports all kinds of robots, as well as designing own creations. It also runs on its own physics engine. Both of these simulators are more thoroughly discussed and compared in 4.1. In the end, the simulation was built using the Microsoft AirSim simulator and the Unreal Engine physics engine.

While it is fascinating to build appealing simulations, it is very important to be aware of the accuracy of the virtualization. The virtual elements must match the physical attributes of the same event occurring in the natural world. For example, when simulating a simple revolute joint in robotics, it is essential to take friction into account. Depending on the repetition count, wearing might be one of the things to consider. When dealing with large or fast-moving objects, wind resistance is a big aspect. In our case, the differents drone propellers might be spinning at different phases, there might be particles flying in the air or there could be seemingly random air currents in the warehouse. These, and many other aspects, all effect on how practical the simulated solution really is. This is why it is crucial to determine the degree of accuracy to which a model, or a simulation,

should be representing a real-world feature. [19]

In our case, the accuracy plays a big factor. If our controlling system differs from the simulated one, the results can be quite unfavourable. Also if our real life sensors behave in a different way, it might mess up the drone's navigation and mapping processes. The testing process must be iterable to ensure the best results. First we need to test the configurations in the simulation and then find a perfect match from the real world. And because the fine-tuning is easier to do in the simulation, we can adjust the simulated sensors, cameras etc. to completely match the real world counterparts.

3 Indoor autonomous object mapping - Case TTS

3.1 Työtehoseura

Työtehoseura (TTS) is a nationwide education and research organization who train approximately 8000 people every year for various lines of work. In addition, they involved in 80 different development or research projects. TTS is working on a project in order to advance relations in especially Europe. The main goal of the project is to create networks to the Storage Equipment Manufacturers Association (SEMA), the largest indoor logistics organization in the world. SEMA is committed to promoting and extending the safe design, installation and the use of storage equipment manufactured and supplied by its members.

SEMA is the most respected storage safety and training organizer in the industry. They certify those who pass their training, and the certified inspectors also share their own global network. TTS is eager to expand their contact list in this network in order to monitor and help the global advancement of the line of business. The current obstruction for the previous has been the lack of proper training in Finland. As the biggest indoor logistics company in the country, TTS has begun warehouse inspections after training one person in SEMA's official course during the year 2018. Unfortunately, the current lack of personnel makes it impossible to handle all the relations, training and storage room rack

examinations itself.

3.2 Warehouse heavy-duty shelf inspections

Currently there exists numerous warehouses in Finland in which the inspections have not been made. Many companies in Finland have warehouses and they are oblivious to the European standard, which also has the status of Finnish national standard, of yearly examinations [20], [21]. Another reason for the violation is that most companies are unsure of how to perform the inspections, or they feel challenging to do. Currently TTS is actively training more SEMA-certified inspectors for the job and to spread the word.

Racking accidents are extremely expensive and also serve a lethal threat. There are a number of different factors that could result in a racking collapse. The cause might not be a single factor but can be a number of combined factors which ultimately weaken the structure to a point where it collapses. The main causes of racking collapses are [22]:

- Inadequate design
- Incorrect installation
- Overloading
- Mechanical handling equipment impacts
- Supporting floor failure
- Environmental or chemical deterioration
- Change of configuration away from the original design
- Poor weight distribution on pallets

Most of these threats can be prevented by taking precautionary actions and following the standards.

On top of training more staff, TTS is working on a technical solution to make these heavy-duty shelf condition checks faster and less expensive. This new effortless and high quality solution would allow TTS to make themselves more known among the SEMA representatives and also in the country. The new autonomous method is based on an unmanned aerial vehicle equipped with various sensors. The UAV would recognize potential deformations in the shelf using different kinds of computer vision. Also, visual material received from the drone can be manually checked by the certified inspectors. This new operating model would remove the constant need of physically visiting the warehouses and it would hasten the process by a significant amount.

3.3 University of Turku and the simulation

A team from the university of Turku works as a contractor and is responsible for the hardware and the software of the drone. This thesis focuses on a simulation built for the project. The purpose of the simulation is to determine the drone's requirements regarding the sensors, cameras, algorithms and so on (regarding RQ1 and RQ2).

The simulation should be as lifelike as possible considering physics, sensors and the environment (RQ4). A 3D warehouse shelf model is created, and a drone will navigate and map the rack using only the data it receives by the sensors attached to it. The ultimate goal is the recognize any deformations, impact signs and possible dangerous situations (see the list in 3.2) and thus answer RQ3. This recognition will mostly likely happen with the use of an image recognition software (when receiving data from a camera mounted on the drone).

The tools for the simulation will be reviewed and compared in Chapter Four. We will try to find a suitable simulation framework that would allow a visually pleasing, but accurate, simulation environment. The idea is to use as much of the existing approaches as possible to reach the end goal as quick as possible. The idea is to work iteratively; first

building an MVP of the simulation, then increasing the configurability, the visuals and the overall usability of the program.

We assume that the drone will be placed in front of either vertical end beam of a pallet rack. A possible follow-up project would be to build a navigation system that would map the whole warehouse and could navigate to each of the racks autonomously. But for now, that is out of scope. The drone should be able to successfully map and photograph the whole structure, as long as it follows the standards mentioned in 5.3. In case of detecting any damages, they could be more closely documented but at least their position and visual information would be sent to the SEMA authorities for further inspection. The quadrocopter should be capable of simple collision avoidance. The end goal is to build a commercial quadrocopter that would take care of the yearly inspections (3.2) and would be easily available for the consumers.

Using a simulated environment will minimize the risks and costs of testing. It will allow the team to find the minimum viable sensors, quadrocopter-platform and algorithms in terms of necessary accuracy.

4 Simulation environment

4.1 Finding the correct framework

In this chapter we compare different possibilities and explain the problem thoroughly. Finding the correct framework for the job is important as we need to apply the learned concepts to the real world. The simulator has few requirements:

1. The drone has to not depend on any data received outside of its own simulated sensors (e.g. location).
2. It must have the possibility of programmatic control.
3. The controls has to be easy to transition from the simulated world to the real physical one.
4. There has to be a way of mounting sensors and cameras on the drone.

There are many different kinds of simulators but most of them are developed as a training tool for pilots. As our case is limited in terms of budget, only free to use software were considered. In the end, two different frameworks came up as a match for the requirements. Microsoft AirSim[15] and Gazebo[16]. Both simulators are capable of simulating several sensors and objects in a three-dimensional world. Gazebo is a whole physics engine whereas AirSim is based on either Unreal Engine[17] or Unity[18].

In the end both frameworks matched the requirements but Microsoft AirSim was used to do the simulation due to a more familiar software stack and system compatibility.

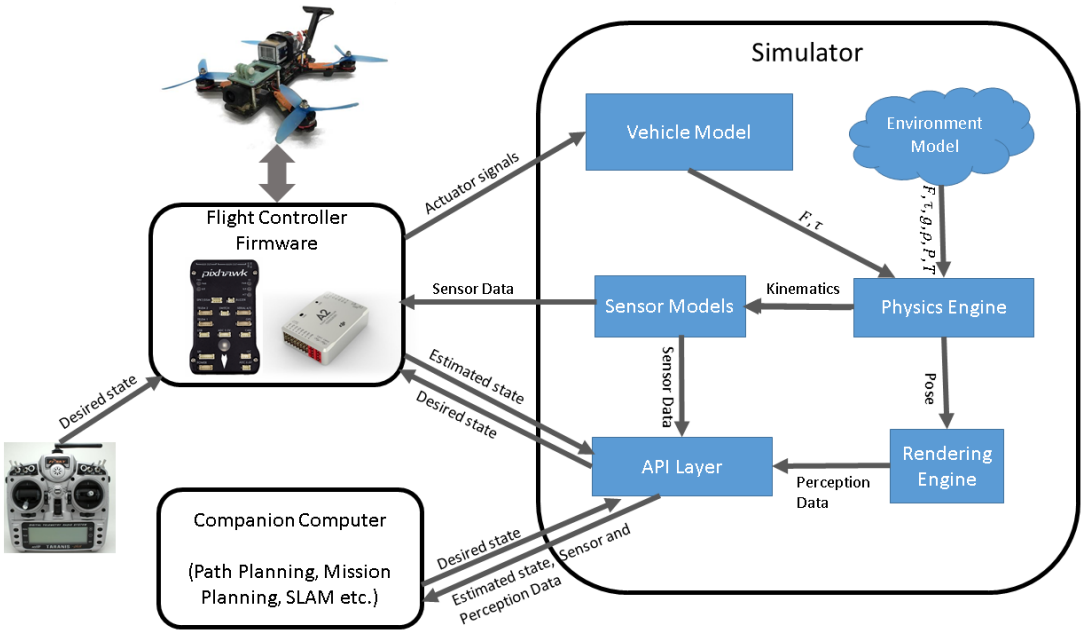
4.1.1 Microsoft AirSim

AirSim is an open source cross-platform simulator for autonomous vehicles built on Unreal Engine with an experimental release on Unity as well. It supports hardware-in-the-loop (HITL) and software-in-the-loop (SITL) with popular different flight controllers. The goal of AirSim is to experiment with artificial intelligence for autonomous vehicles. [23]

AirSim has a thorough documentation and it supports both Windows and Linux platforms. By default it uses a built-in flight controller called simple_flight but also has support for PX4[24] and future plans for ROSFlight and Hackflight. Advantages of using simple_flight is that it uses a steppable clock which means that the clock advances when the simulator tells it to advance (so it is not tied to real passage of time). This allows for the simulation to be paused, for example, on breakpoints in a program and there will be zero variance in the clock (unless the clock itself has variance). In addition, it is completely header-only dependency-free C++ code which allows users to step through from the simulator to inside the flight controller code within the same code base.

Microsoft's simulator exposes different application programming interfaces (API) for the user to interact with the vehicle programmatically using either Python or C++. These APIs can be used to control the vehicle, read sensor data, get sensor or drone states and so on. The interface can be given commands to via a companion computer mounted on the quadrocopter

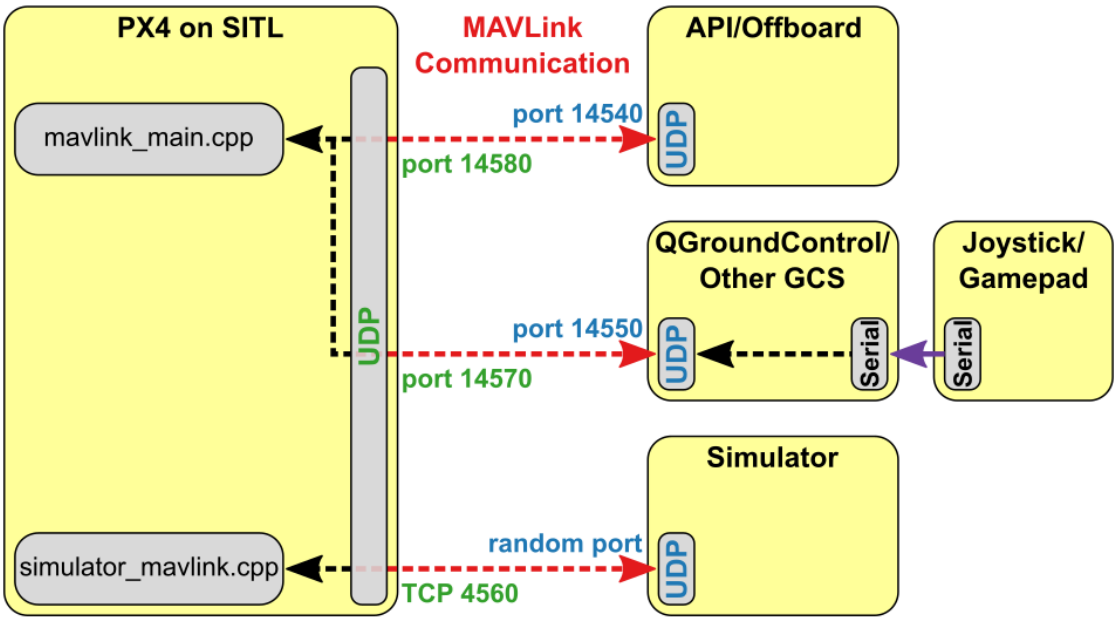
Figure 4.1: AirSim architecture [23]



4.1.2 Gazebo

Gazebo is an open-source standalone 3D robotics simulator. It supports all kinds of simulated robots, such as drones, and it uses the Robot Operating System[25] as the controlling interface for the robot. Gazebo and ROS can be used together with PX4 to provide flight control tools for aerial vehicles. This stack enables HITL and SITL using MAVLink as a messaging protocol for communication. It can be connected to a ground control station (QGroundControl preferred) which allows an external machine (an app, a companion computer etc.) to fly a PX4 based vehicle using an autopilot or manual control.

Figure 4.2: An example of a generic PX4 simulation environment [26]



4.2 The drone

The simulation uses a quadrocopter equipped with various sensors and cameras. We need to determine which is the lowest cost solution hardware-wise that can complete our goal. The possible sensors and cameras in our Microsoft AirSim simulated drone are

- inertial measurement unit (IMU)
- magnetometer
- global positioning system (GPS)
- barometer
- one dimensional distance sensor
- 2D-lidar
- 3D-lidar
- regular camera
- infrared camera
- simulation-only depth camera

AirSim allows multiple same kind of sensors and cameras mounted on the drone at once.

As we are working indoors, we can safely rule out the GPS sensor and the barometer. The magnetometer might get confused by nearby devices, so it won't be used either. That leaves us with the IMU, distance sensor and the lidars. IMU has to be used as the drone needs to keep track of its position relative to the starting point (discussed in 4.2.1).

This leaves us with the cameras, the distance sensor, the 2D-lidar and the 3D-lidar (explained in 4.2.2), from which we will need to decide which, or what, to use.

4.2.1 Local coordinate system

As we are working in a simulation, we of course have access to the absolute location of the drone at all times. We can use this absolute location as a accuracy indicator and see how close to the actual location we are estimating our position.

The simulated drone uses a local three-dimension coordinate system. The origin of the coordinate system is at (0, 0, 0) and it is the drone's location at the start of the simulation. Using the gyroscope and accelerometers of the IMU, we can track the movement of the drone quite accurately (more on the accuracy later). But the longer the drone is moving and tracking its location, the more small errors happen. In the long run they stack and after a while the self location estimate could be off by multiple meters.

AirSim uses the north-east-down (NED) coordinate system familiar to aircrafts, which is a system where its axes (x, y, z) are oriented along the geodetic directions defined by the Earth's surface. In the simulation the axes are aligned locally based on the drone's starting orientation. When looking at the drone from behind the x-coordinate is pointing forward of the drone (north), y is pointing right of the drone (east) and z is going down from the drone (down). Note that this means that gaining altitude corresponds to decreasing the z value.

All of AirSim's API use the NED local coordinate system format in output as well as in the input. This makes it easy to pass on data between the functions.

4.2.2 Distance sensors

AirSim allows sensors to be added or swapped easily by just modifying the configuration file [27].

Simple distance sensor

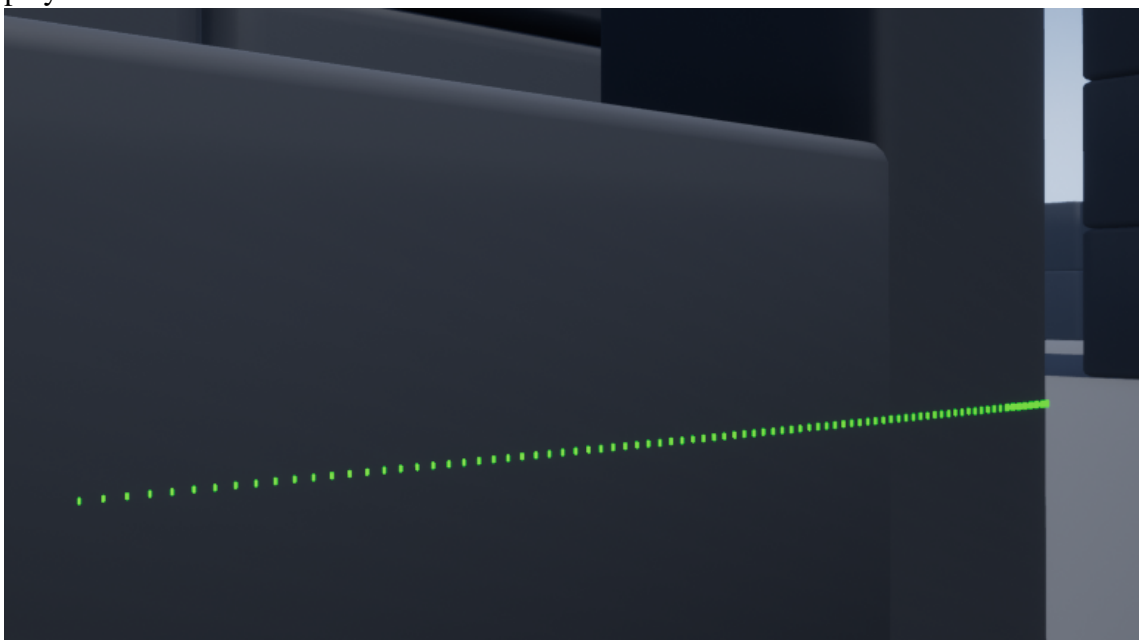
In the simulation, invoking the distance sensor API calls the C++ implementation in the AirLib folder which then again calls the game engine (Unreal or Unity) wrapper's own sensors. This sensor uses the engine's own implementation of ray tracing which calculates the distance between the sensor and the first object the ray collides with. This returns a value in SI-units (meters) which we can then use to detect objects in front of the mounted sensor on the drone. In real life a simple distance measurement can be achieved by using LED, VCSEL, ultrasonic, etc. sensors

While being extremely cheap, these simple distance sensors might be a little bit inefficient for the job of mapping and navigating a pallet rack. The drone needs to continuously move on the right-down (or east-down) axis to detect the shape of the object in front. This brings difficulties in collision avoidance as well. Multiple distance sensor could be an option but that means they would all need to be in sync and

LiDAR

When moving on to the LiDAR (light detection and ranging, later mentioned as 'lidar') world, the price goes up, but the complexity of the drone-control algorithm goes down. A lidar is an instrument that shoots numerous rays of light in a rapid succession and then calculates the time it takes for each beam to bounce back. As light moves at a constant and known speed, we can calculate the distance between the object and the sensor. By repeating this multiple times we lidar build a complex map of the surface. As our drone-mounted LiDAR is moving, we need to take the orientation and location into account when building this point-cloud of the object being mapped.

Figure 4.3: A closeup of a simulated single-channel lidar with laser contact points displayed.

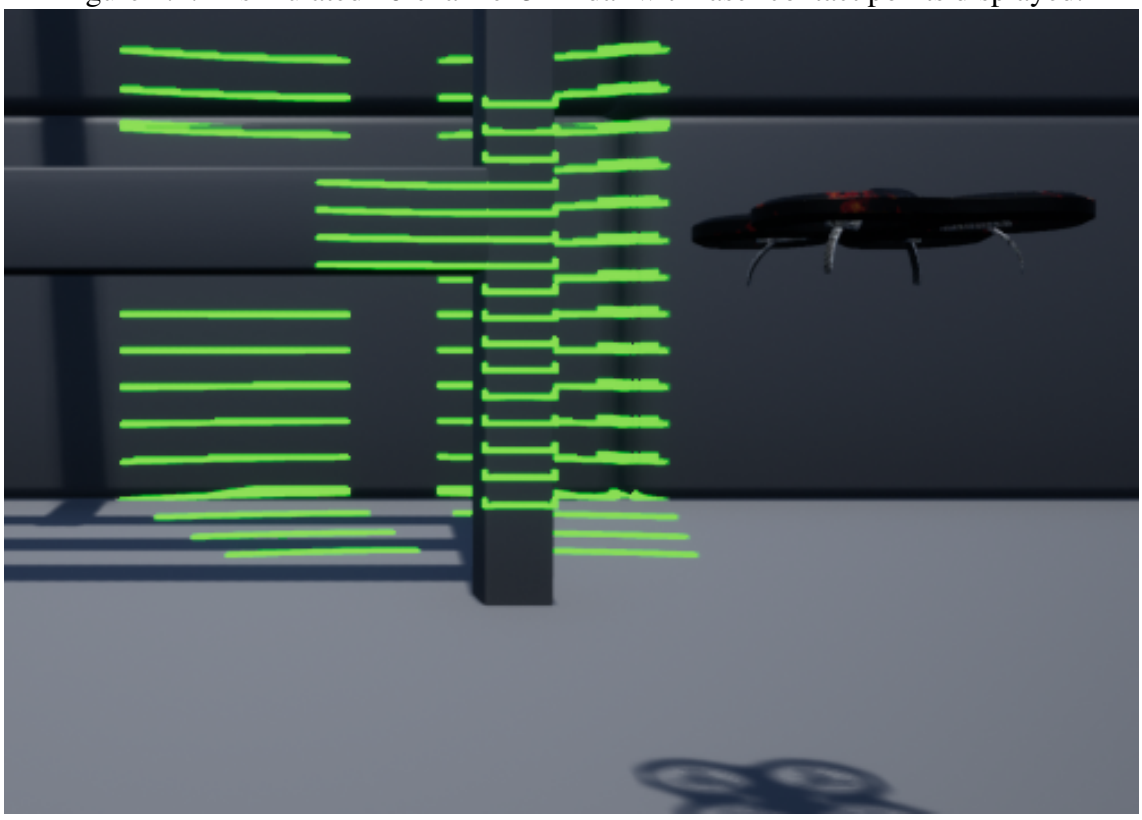


In the AirSim framework, the lidar sensor can be heavily configured [28] and in order to get a 2D-lidar we only need to set the number of channels to one in order to get a single laser instrument tracking the distances on a certain axis depending on the orientation of the device.

A 2D-lidar helps our drone to follow the vertical beams in the structure, as we can scan the horizontal plane. This allows the drone to detect any east-coordinate-wise abnormalities (such as horizontal beams to detect beam intersections) while only moving parallel to a vertical beam.

When moving up a dimension to the final 3D-lidar, the costs go up and fast. Using a three-dimensional lidar allows the drone to follow any sort of line in a east-down-pane without departing from the path. The instrument has multiple sensors, each firing at the same time, creating a 3D-map of the surroundings. The usage of a 3D-lidar will most likely be outside of the budget.

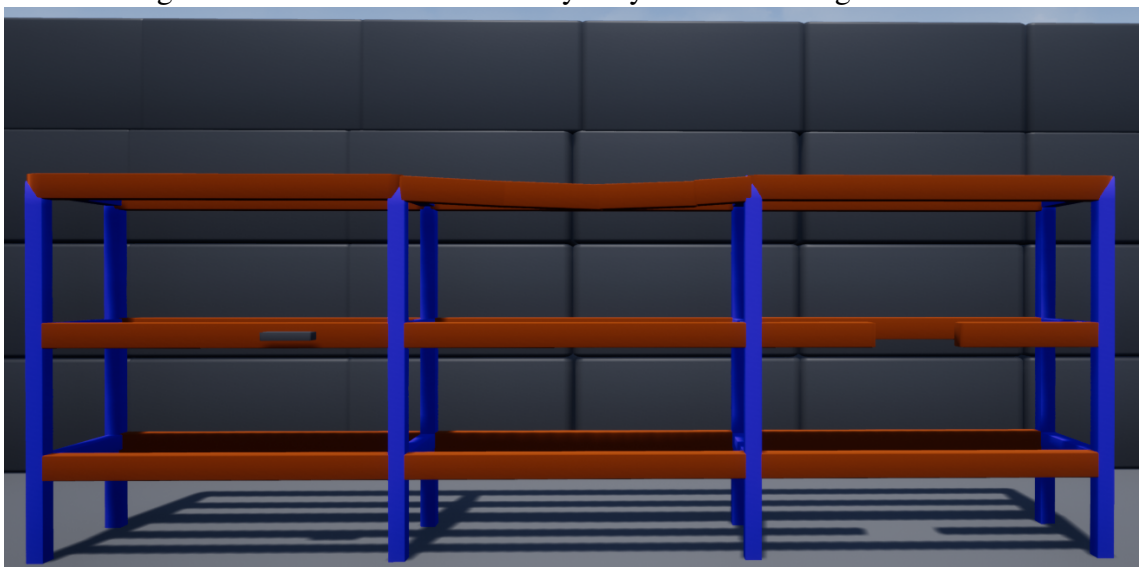
Figure 4.4: A simulated 16-channel 3D lidar with laser contact points displayed.



4.3 Modeling the pallet rack

In the first iteration of the simulation the pallet rack was built using simple rectangle-shaped structures as seen in figure 4.4. The point was to quickly test the functionality of different sensors. Later the structure was colored (fig 4.5) accordingly to one of the most common color scheme (red/orange horizontal beams and blue vertical beams). Also, some deformations were modeled. This model was used in the simulation built in the next chapter.

Figure 4.5: A 3D model of a heavy-duty rack containing some faults.



5 Building the simulation

5.1 Overview

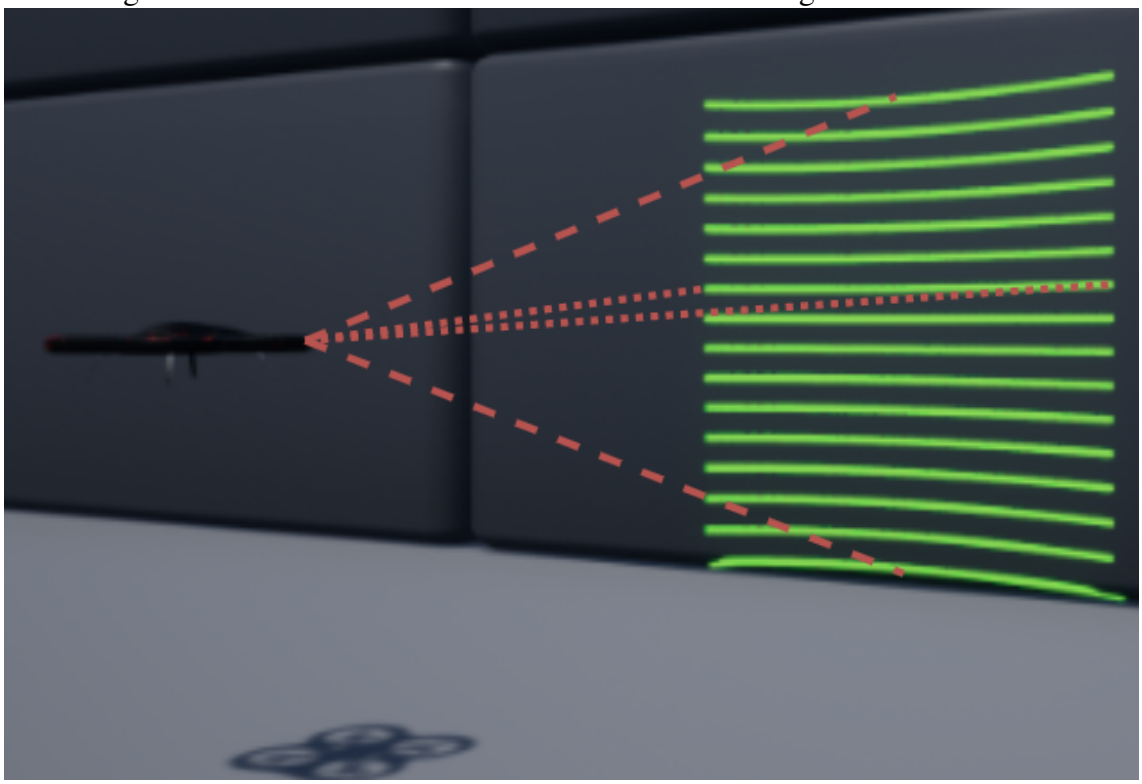
By creating a realistic simulation, we can validate our choices and configurations regarding the drone. The goal is to build a set of algorithms that allow the drone to autonomously map the structure. If we can find an efficient way to save the scan-path of a rack, the next examinations will be a lot faster. In this chapter we inspect the different approaches and solutions to the problem. We are going to use a lidar-sensor as a way of detecting where the beam is.

5.2 Using the lidar sensor

The simulated lidar sensor is highly configurable as we can modify the number of channels, maximum range, number of points captured per second, number of rotations per second, field of view (FOV) in each direction, position and orientation on the drone and finally the type of data returned.[29] We will be configuring the lidar sensor to have ten rotations per second and measuring 163500 points per second with a total field of view of 32 degrees horizontally. If we are using a 3D lidar (more than one channel) we will use a vertical FOV of 32 degrees as well.

Calling the function getLidarData in the client returns a point cloud in a flat array of points in float format. The points represent the north, east and down -values (NED-coordinates (4.2.1) in meters) for each laser beam that hit a surface within the specified

Figure 5.1: A simulated drone with a 16-channel 32-degree FOV 3D-lidar.



range. As the returned value is a flat array, we will need to reshape it into a nested array consisting of three-length arrays each containing a coordinate. Using this point cloud data we can acknowledge the environment and build algorithms that guide our autonomous drone to behave in the desired way in every situation.

5.3 Detecting the outermost beams of the pallet rack

In order of maximizing load bearing capacity, the outermost/bounding beams of the pallet rack create a rectangle-shape. This rule opens up easier possibilities to map the shape of the shelf-system because now the bounding area of the rack will be a simple shape every time (see figures 5.2 and 5.3). Now we can simply follow the outerior bounding beam and determine the size of the whole structure easily.

Figure 5.2: Front-facing examples of allowed pallet rack shapes.

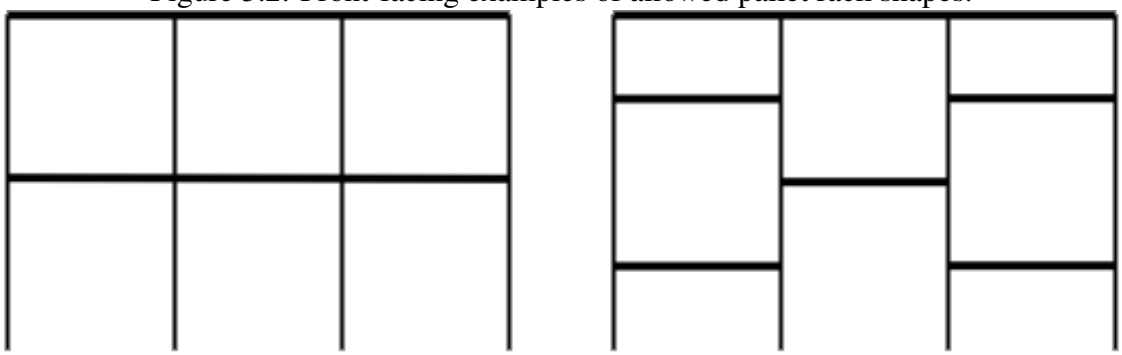
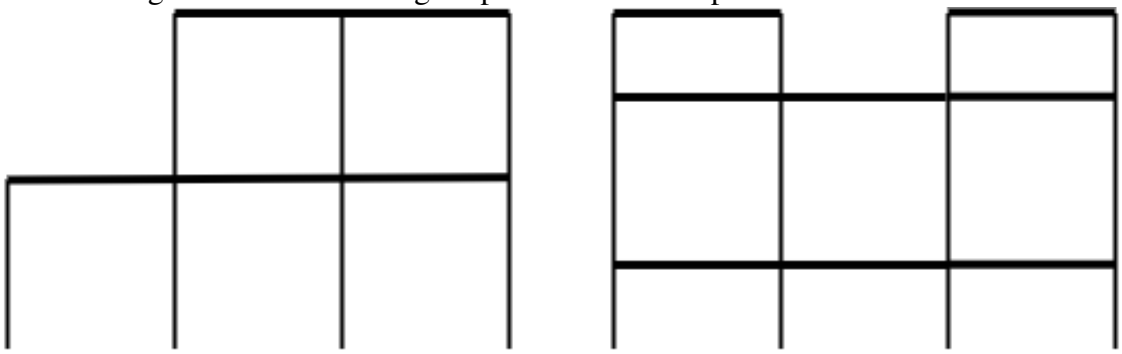
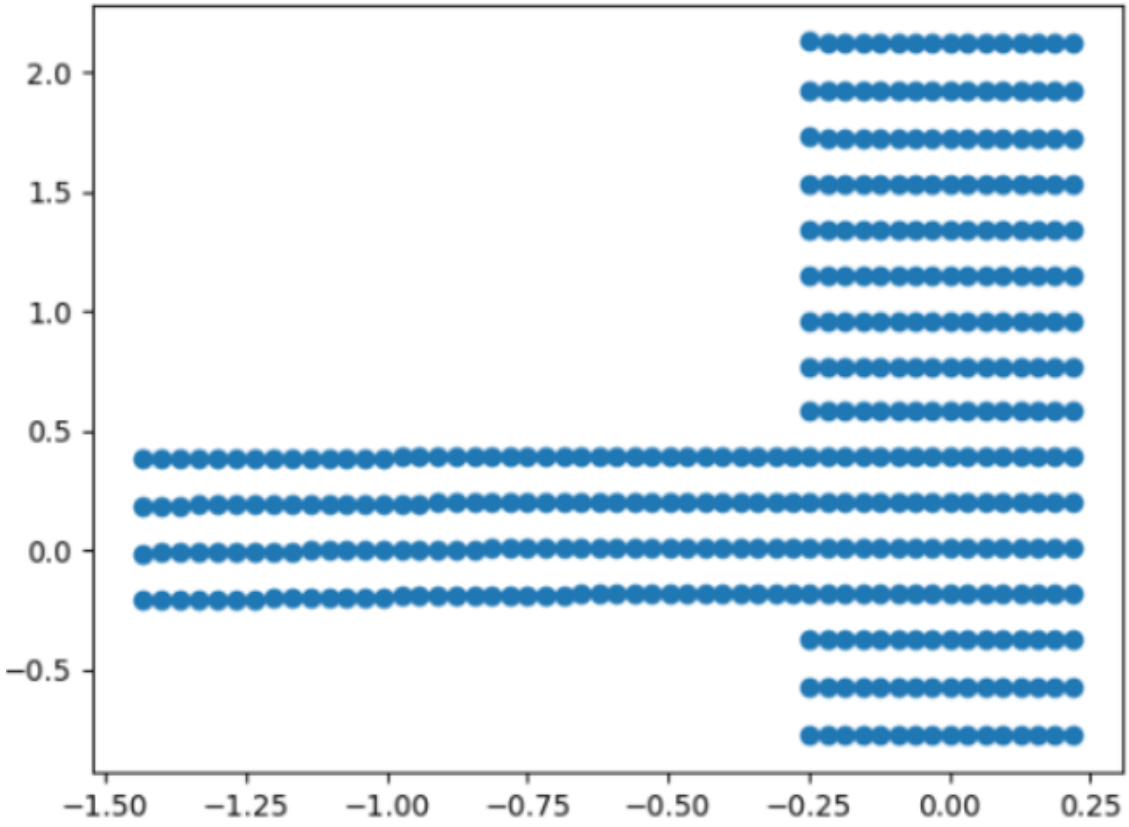


Figure 5.3: Front-facing shapes of disallowed pallet rack-formations.



When using only lidar for the mapping, we first need to determine the distance from the sensor to the beam. Then we can use this distance to follow the beam when going up the first time. In addition, it allows us to recalibrate if the drone happens to sway too much on the N-axis. When we read the lidar-data for the first time, we can filter out all the points that are not on the same east-down-plane. Then we are left with data shown in fig. 5.4 on the next page.

Figure 5.4: A raw scatter plot of a point cloud from image 4.4 where only the beam points are plotted. Note that the figure appears to be upside down due to the usage of NED-coordinates.



We can further narrow it by filtering out the points too far away from the drone on the east-axis. Then we are left with information about whether the object in front of the sensor goes continuously up or down. We can increase the drone's altitude and re-scan the area whenever reaching the top of the last scan. This way we have soon scanned the whole vertical beam in front of the starting position.

The same strategy can be applied on following the top horizontal beam:

1. Center the drone down-axis -wise on the horizontal beam. This is the beam start location on east-down-plane.

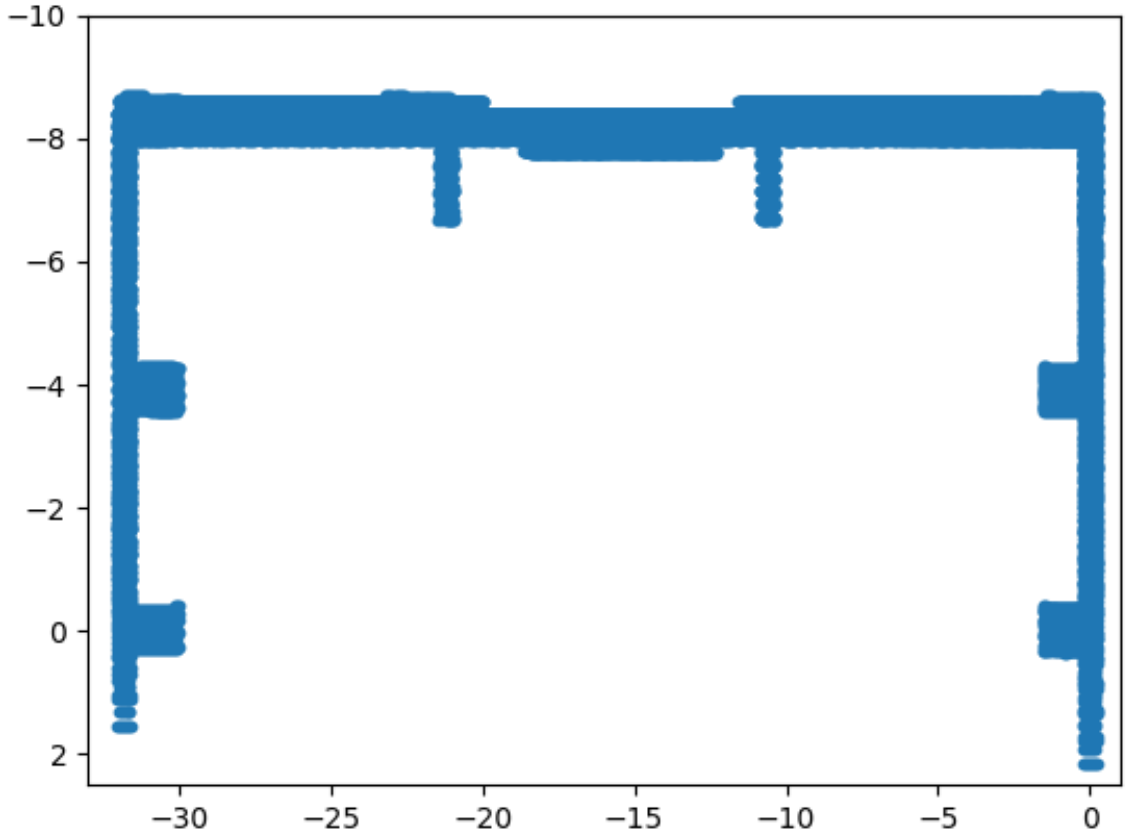
2. Read the lidar data and filter it to contain only points that hit a beam on a nearby altitude.

3. Determine the direction to follow by inspecting in which way the beam travels.

4. Move the drone in that direction until the lidar doesn't detect a beam anymore. This is the beam end location on east-down-plane.

After we have mapped the top beam, we already have knowledge of the width and the height of structure and we can just lower our altitude and use lidar to map the final vertical rack-bounding beam (fig. 5.5).

Figure 5.5: A scatter plot of all the point cloud points gathered while following the outer beams of the pallet rack. Note the flipped y-axis due to NED-coordinates



5.4 Mapping the pallet rack

5.4.1 Determining beam locations from a scatter plot

As figure 5.5 shows, at this point of the scan we have the bounding beams mapped. We can also find the "stumps" of the supporting vertical and horizontal beams. Using these stumps, it is possible to calculate all the beginning and end points of the beams. By beginning at either bottom corner, we can start looking for these beam-start coordinates by using the following algorithm (visualized in figure (5.6):

1. From a bounding beam's start coordinate, using a arbitrary small step size, loop until the beam's end coordinate (we call this the primary direction).

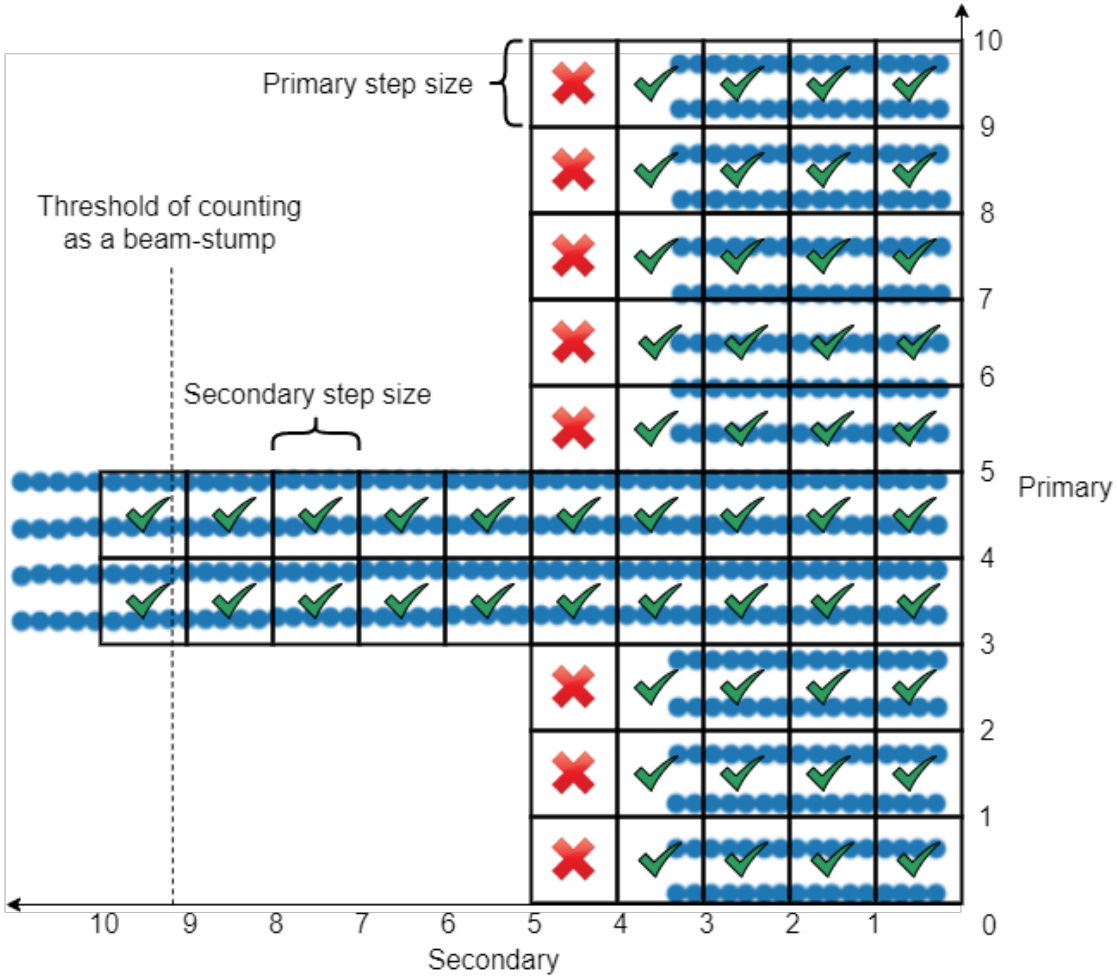
2. For each coordinate, iterate to the opposite direction (secondary) of the current iterable beam (right if currently on the left bounding beam, down if on the top bounding beam and vice versa).

3. Observe whether there lies at least a few mapped beam points on the plane determined by current primary and secondary coordinates and the previous primary and secondary coordinates.

4. If true, then we can assume that the structure is continuing to the opposite direction and we can keep iterating to that way until a certain threshold (of accepting that we are on a beam-stump) is met. If false, then we know there is not a beam at this position, and we can move on to the next primary-coordinate.

5. In case we find multiple neighbouring stumps, we can count them as one and use their average location. In the example 5.6 we can determine that there is one beam starting at the coordinate corresponding to the primary index 4.

Figure 5.6: A visualization of the algorithm used to find beam stumps (visualization done on the same location as the 2D-map 5.4). For each primary value we loop secondary values until we either reach the threshold or encounter an area without beam points.



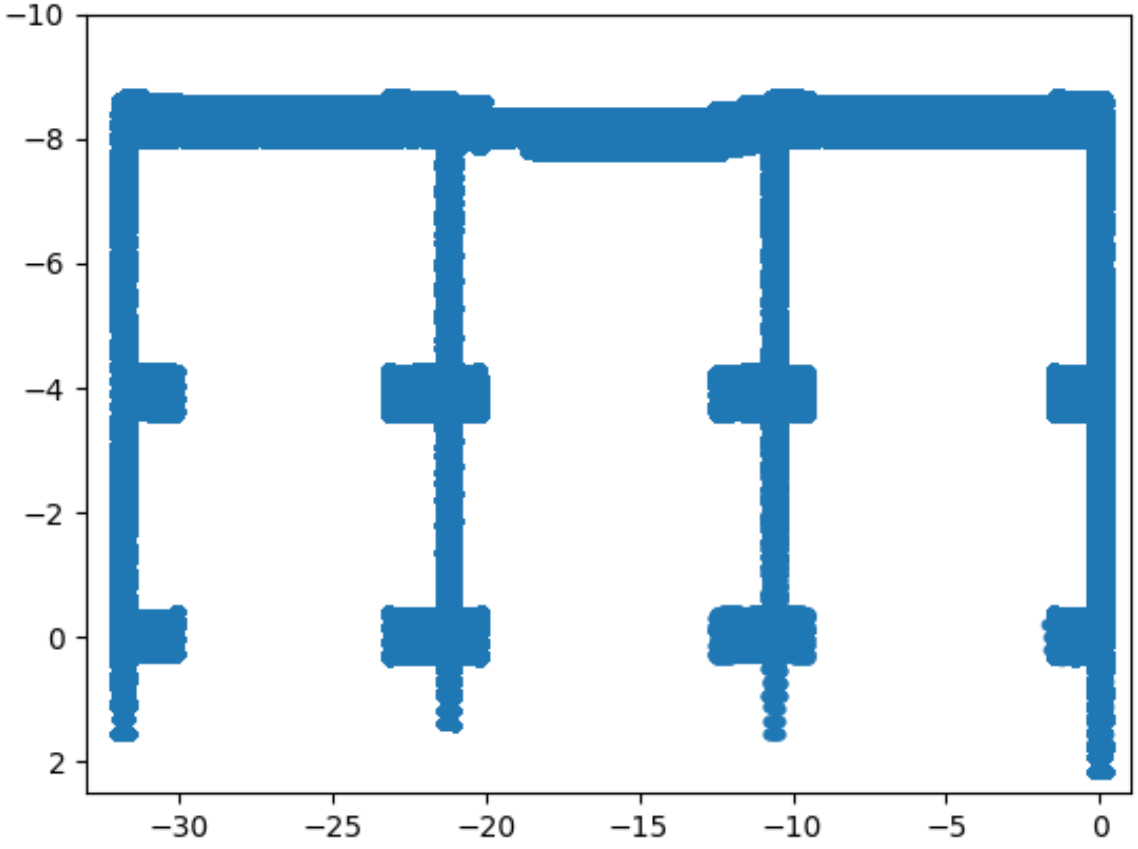
Using the previous method, we will avoid a lot of false positives made from noise. Depending on the point-density we can also set up a minimum amount of points needed per area for it to count as a beam. This improves the algorithm accuracy even more. It also works on both vertical and horizontal beams.

5.4.2 Mapping the beams

After utilizing the beam finding algorithm from 5.4.1 to all the surrounding beams we will know the start and end points of all the beams. Now we still need to navigate along these beams in order to map and photograph them. Because the horizontal beams are not continuously on the same elevation throughout the structure (see figure 5.2), we will first need to map the non-bounding vertical beams. This is a trivial task as we already have knowledge of their location on the east-axis. After the mapping of the vertical beams, we will arrive at a situation visualized in figure 5.7, missing only the non-bounding horizontal beams.

Now the final task before whole structure is processed, is to find the horizontal beams. This is done by first applying the stump finding algorithm (5.6) to the newly mapped vertical beams and then finding the positions where there are two adjacent stumps at the same height (such as positions (-32, 0) and (-23, 0) in figure 5.7). After all the horizontal beam stumps are known, the last thing left is to map each of the beams that are between adjacent same-height stumps.

Figure 5.7: 2D scatter plot of the scanned beam points after going through each of the vertical beams.

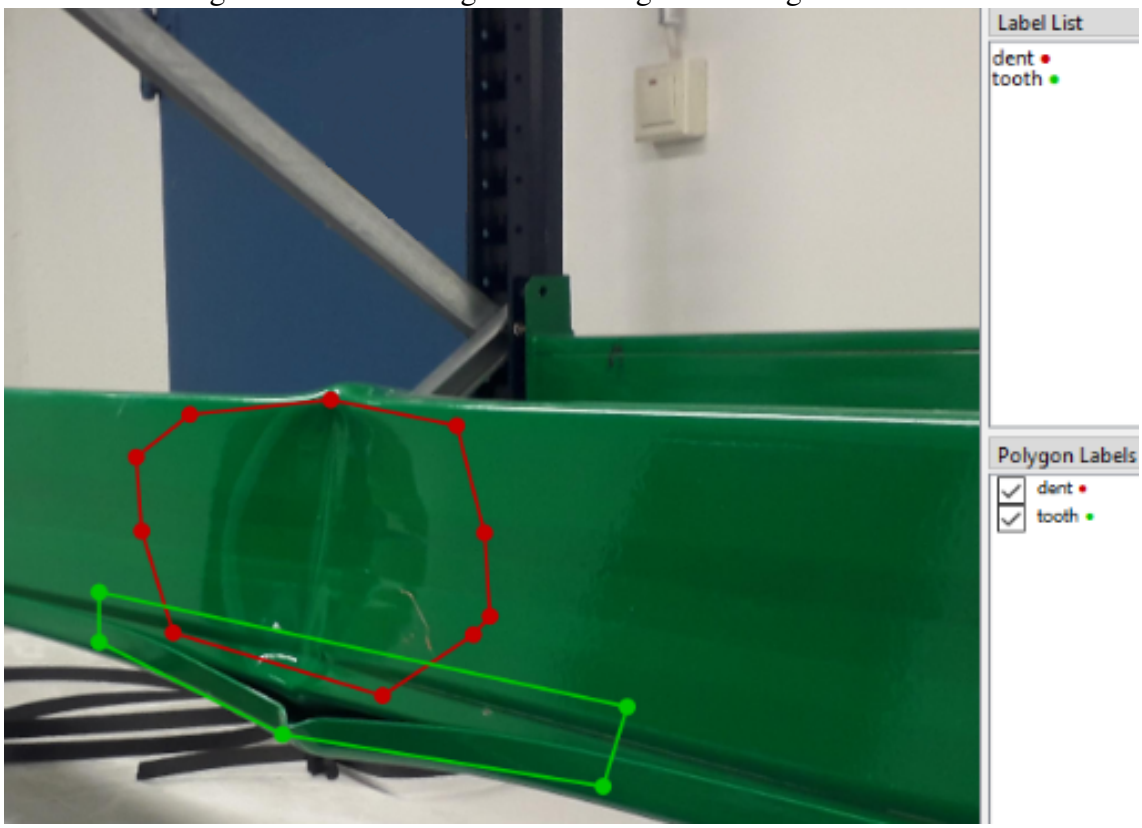


5.5 Damage detection

At the current phase of the project, the plan is to map and photograph each horizontal beam. The lidar-map (seen in figure 5.7) can show us some bending problems but when it comes to other structural health problems such as impact marks, it might be impossible to detect them. This is why it is necessary to have machine vision tools to help detecting these defects.

The method for recognizing these damaged parts of the beam is called Mask R-CNN [30]. By first manually gathering images of different sorts of faults in the beams, we can then annotate these faults visible in the pictures and then train our convolutional neural network with the data 5.8.

Figure 5.8: Annotating defects using VGG Image Annotator.



After training our model with the previously mentioned dataset, our model behaves at an accuracy suitable for our needs.

5.6 Avoiding collisions

Safety is a big point and as we will be moving a real object in a real world, collisions have to be avoided. As discussed in 2.1.3, we are flying indoors, the risk of collision is even higher.

We can use the already existing sensors and cameras to provide safety for us. There are many drones equipped with distance sensors to each direction just for this purpose but in our case we will be building our own drone on top of an existing platform, so there is no certainty of these collision detection sensors. Therefore we need to prepare for collision avoidance by either using a 360-degree sensor, having more sensors or turning the drone before moving to any direction. At the current phase of the project, collision avoidance was left to be implemented later on.

5.7 Demonstration

The end result[31] demonstrates the whole process of mapping a pallet rack. The drone is equipped with a regular camera, a non-realistic depth camera (for the sake of an example), a lidar sensor and IMU. In the demonstration the quadrocopter successfully maps a 33 meters wide and 10.5 meters tall heavy-duty shelf containing four vertical beams and three horizontal beams in around three minutes and thirty seconds. In the lidar map we can clearly see the three defects in the structure:

1. The top beam bent in the middle
2. An anomaly in the middle horizontal bar on the left
3. Part of the middle horizontal beam missing on the right

There are some unwanted effects visible as well. We can see some of the beams appearing larger on some areas than others. This is either because of the drone movement (horizontal vs vertical) or the beam being just barely outside of the lidar sensor's horizontal scan line (figure 5.1). The problem could be fixed by accounting the current travel direction or after the scan by modifying the data, but at the moment it functions well as it is.

In the case of an actual scan, the drone would've recorded the camera footage while traveling along the horizontal beams. This footage would then be analyzed by the image recognition program (5.5) and any possible defects would be then reported to the correct parties.

For future scans, this lidar map would be saved under a specified label. Then on the next scan, the drone would be able to straight away record the horizontal beams without the need of mapping the whole structure again. The program works on any allowed rack shape (figure 5.2) when placed on the ground in front of a bounding vertical beam.

In case of detecting an unknown shape or some other error occurring, the drone would return to the base and report this error.

At the moment the drone travels quite a bit "over" each of the beam's end points. This distance can be easily modified to suit the environment. If this distance is too short, then the drone could mistakenly identify the missing beam as a beam end point (the same problem would occur in case of a wrapping hanging in front of the beam).

Due to time restraints and uncertainty of the need, a functioning collision avoidance mechanism was not implemented for this demonstration.

6 Conclusion

To conclude the thesis, this chapter will go through each of the topics discussed and answer the research questions.

In the first chapter the thesis structure is presented, and the research questions of the thesis are introduced:

1. What is a satisfactory simulation environment for a project of this kind?
2. What is the optimal setup for the simulated drone in terms of sensors?
3. How to efficiently map a shelf and detect possible deformations in it
4. How applicable to the real world the solution in question is?

After introducing the thesis in the first chapter, we move on to the second chapter labeled *Autonomous drones and simulation techniques*. This chapter presented already existing methods and techniques for moving and simulating a flying robot (quadrocopter). We learned about independent simultaneous localization and mapping (SLAM), different pathfinding algorithms, collision avoidance and the virtualization of different robot environments. We figured out that there are no suitable already existing solutions for our specific need.

The following chapter *Indoor autonomous object mapping - Case TTS* gave insight into the actual project behind the need for this thesis. Työtehoseura was introduced, as well as their ambitious goal of automating the process of yearly pallet rack inspections. The responsibility of University of Turku is to design and implement the hardware and

software of the drone and the main goal of this thesis is to build a simulation as a tool for both Työtehoseura and the university of Turku. The grounds for this simulation and the implementation of it were introduced in the next two chapters.

The fourth chapter started the processing of the first two *Research Questions* (RQ1 and RQ2) and it evaluated different frameworks for the simulation and the different quadrocopter configurations in the chosen framework (Microsoft AirSim). The wide variety of sensors are showcased and the initial choice of sensors to use for the drone was to use lidar sensors.

In the fifth chapter the actual simulation where the drone behaves in the desired way, was built (RQ3). We successfully managed to map the pallet rack without any prior information about it. A way for accurate damage detection was found using an image recognition technique called Mask R-CNN.

And the final chapter after this one, *Discussion*, contains personal thoughts about the whole project, the simulation and this thesis.

6.1 Answering the research questions

When it comes to the research questions, RQ1 was thoroughly discussed and answered in chapter four. The environment depends on the project members. For this project, the main points of the end result are the overall functionality and visuals. In the end, the accuracy of the simulation was not as a huge deal as initally thought (as long as it was accurate enough for the drone to accomplish the mapping)

In the same chapter, RQ2 was discussed and I feel like the single lidar sensor provided an excellent basis for building the simulation. When it comes to adding more sensors, it is dependent on the future of the project. If need be, the simulated drone can be easily extended to support more sensors.

RQ3 was handled, explained and answered in chapter five. This question is the most

concrete and has a clearer answer, which is also visible in the simulation video. The mapping has to happen without any prior knowledge of the rack. This is where a combination of a custom SLAM and pathfinding algorithms come to play.

Personally it felt like the ongoing pandemic delayed the actual project, making the final research question (RQ4) very difficult to answer. It feels paradoxical to answer a question about the real life applicability using only simulated works and research papers. But the simulation helped the team to see the bigger picture and proved that the technique is actually possible. So the simulation worked as a validation for the whole project.

7 Reflections on the simulation project

During the time period of writing this thesis and creating the simulation, the following thoughts rose to the surface.

The project

First of all, this thesis and the simulation itself was written during the ongoing COVID-19 pandemic[32]. This affected the project in a negative manner, being one major factor of why the project advanced slower than expected. The initial plan was to have a working prototype by the end of April and at the time of writing this (20th of April), it does not seem possible. Also due to the social distancing guidelines, physical meetings were not an option which, in my opinion, hindered information sharing. This in return made creating the simulation a not so agile[33] process, which in return caused long periods of working time without communication.

Initially it seemed like the damage detection from pictures using image recognition was not a viable way. But then after labeling enough data and experimenting with different methods, a suitable way of defect detection was found. The data from the simulation (camera output, steadiness etc.) can be used when further developing the image recognition process.

The end result

The simulation is an actual working product as the video in. TTS can use this simulation as a demonstration for the project itself, which helps promoting the innovation and applying for further funding. Currently the program is built with focus on lidar sensors, while currently it appears that those sensors are out of scope for the project. At the moment the plan for the physical product is to use cameras and stereo vision[34] for the 3D mapping. But in the end the mapping process won't deviate a lot whether the drone is equipped with a stereo camera or a lidar.

Personal thoughts

Personally, I feel like the simulation could have been done in an even more thorough way using more different sensors, but given the current circumstances and the budget, I think the end result was satisfactory.

It would be interesting to continue on this project and work on the simulation in order to make it easier to use, more modular in terms of sensors and have multiple different environments in it. Collision avoidance is a big factor missing from the current implementation of the simulation. It would create an entirely different case for simulation (whether a block should stop the whole mapping process and report it, or continue by just dodging the obstacle) and therefore building it would have been too time-consuming when compared to the gains of it.

It would be fascinating to see the differences between different sensors and configurations in various conditions. Some other improvement ideas would be to use the mapped lidar data together with the image recognition technique in order to detect defects. When it comes to the absolute end goal, I believe the drone has to be able to autonomically navigate from the charging platform to a to-be-mapped pallet rack. This is something that requires a lot more research and work to accomplish.

It was disappointing not to see the physical product in action (and therefore answering

the final research question) but for me, personally, finishing this thesis was the most important factor. About the whole project itself, I think the goal is ambitious and technically very plausible. It would improve the overall safety of warehouses, and therefore prevent serious accidents, which would make the world a safer place. It was a pleasure working together on this project and to get a real problem to tackle for my thesis.

References

[1] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984, ISBN: 0201055945.

[2] K. Khantanapoka and K. Chinnasarn, "Pathfinding of 2d 3d game real-time strategy with depth direction a-star algorithm for multi-layer", in *2009 Eighth International Symposium on Natural Language Processing*, 2009, pp. 184–188.

[3] M. Gombosi, "Evolution of path finding", in *Proceedings of the 23rd International Conference on Information Technology Interfaces, 2001. ITI 2001.*, 2001, 133–138 vol.1.

[4] L. Zhang, J. Lai, P. Shi, S. Bao, and Liu Jian-Ye, "An improved mcs/ins integrated navigation algorithm for multi-rotor uav in indoor flight", in *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, 2016, pp. 2099–2104.

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

[6] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam", *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[7] M. P. Parsley and S. J. Julier, "Exploiting prior information in graphslam", in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2638–2643.

[8] Y. Chen, C. Jiang, L. Zhu, H. Kaartinen, J. Hyyppä, J. Tan, H. Hyyppä, H. Zhou, R. Chen, and L. Pei, "Slam based indoor mapping comparison:mobile or terrestrial ?", in *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, 2018, pp. 1–7.

[9] Bing Liu, Siew-Hwee Choo, Shee-Ling Lok, Sing-Meng Leong, Soo-Chee Lee, Foong-Ping Poon, and Hwee-Har Tan, "Finding the shortest route using cases, knowledge, and djikstra's algorithm", *IEEE Expert*, vol. 9, no. 5, pp. 7–11, 1994.

[10] *PathFinding visualization*, `https://qiao.github.io/PathFinding.js/visual/`, Accessed: 14-April-2020.

[11] S. Prophet, J. Atman, and G. F. Trommer, "A synergetic approach to indoor navigation and mapping for aerial reconnaissance and surveillance", in *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2017, pp. 1–8.

[12] L. Ling, Y. Niu, and H. Zhu, "Lyapunov method-based collision avoidance for uavs", in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, 2015, pp. 4716–4720.

[13] J. Dentler, S. Kannan, M. A. O. Mendez, and H. Voos, "A real-time model predictive position control with collision avoidance for commercial low-cost quadrotors", in *2016 IEEE Conference on Control Applications (CCA)*, 2016, pp. 519–525.

[14] D. Wloka, "Robsim - a robot simulation system", in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, 1986, pp. 1859–1864.

[15] *Microsoft AirSim*, `https://microsoft.github.io/AirSim/`, Accessed: 24-December-2019.

[16] *Gazebo*, `http://gazebosim.org/`, Accessed: 24-December-2019.

[17] *Unreal Engine*, `https://www.unrealengine.com/`, Accessed: 28-January-2020.

[18] *Unity*, `https://unity.com/`, Accessed: 28-January-2020.

[19] D. Thomas, A. Joiner, W. Lin, M. Lowry, and T. Pressburger, “The unique aspects of simulation verification and validation”, in *2010 IEEE Aerospace Conference*, 2010, pp. 1–7.

[20] CEN - European Committee for Standardization, *European Standard EN-15635 "Steel static storage systems. Application and maintenance of storage equipment"*. 2008.

[21] Työtehoseura, *Kuormalavahyllytarkastukset*, https://www.tts.fi/tutkimus_kehitys/hankkeet/muut/turvallinenvarasto/auditoinnit/kuormalavahyllytarkastukset, [Finnish; accessed 24-December-2019].

[22] G. Dobinson, *Warehouse racking collapses - cause and origin investigation*, https://www.hawkins.biz/insights/insight/warehouse-racking-collapses-cause-and-origin-investigation/, [Finnish; accessed 24-December-2019].

[23] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Aerial Informatics and Robotics platform”, Microsoft Research, Tech. Rep. MSR-TR-2017-9, 2017.

[24] *PX4*, https://px4.io/, Accessed: 28-January-2020.

[25] *Robot Operating System*, https://www.ros.org/about-ros/, Accessed: 28-January-2020.

[26] *PX4*, https://dev.px4.io/v1.9.0/en/simulation/, Accessed: 01-April-2020.

[27] *AirSim sensors*, https://microsoft.github.io/AirSim/docs/sensors/, Accessed: 28-January-2020.

[28] *AirSim LiDAR*, https://microsoft.github.io/AirSim/docs/lidar/, Accessed: 28-January-2020.

[29] AirSim, *AirSim lidar sensor documentation*, "https://microsoft.github.io/AirSim/docs/lidar/", [Accessed 25-February-2020].

[30] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask r-cnn*, 2017. arXiv: 1703.06870 [cs.CV].

[31] *Simulated drone mapping a pallet rack*, https://youtube.com/watch?v=WGsBYgwm2bk, Accessed:24-March-2020.

[32] WHO, *COVID-19*, https://www.who.int/health-topics/coronavirus, [Finnish; accessed 20-April-2020].

[33] D. Leffingwell, *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[34] D. Murray and C. Jennings, "Stereo vision based mapping and navigation for mobile robots", in *Proceedings of International Conference on Robotics and Automation*, vol. 2, 1997, 1694–1699 vol.2.