NTNU
Norwegian University of
Science and Technology

Sensor Simulation and Environment Mapping using UWB Radar for Industrial Drone Inspection

David Chen Billdal

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Tor Arne Johansen, ITK
Co-supervisor: Kristian Klausen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MSC THESIS DESCRIPTION SHEET

Name:	David Chen Billdal
Department:	Engineering Cybernetics
Thesis title:	Sensor Simulation and Environment Mapping using UWB Radar for Industrial Drone Inspection

Thesis Description:

A system for automatic and safe inspection of tanks and other infrastructure with Unmanned Aerial Vehicles (UAVs) is under investigation at the department. Critical to the system is methods for collision and detection of obstacles, such as wires, pipes and walls.

The system is a part of an ongoing commercialization process in cooperation with NTNU TTO and Scout Drone Inspection AS, and the student will sign a “Standardavtale” with Scout Drone Inspection.

This project aims to investigate environment mapping with radar, and sensor simulation and visualization in Gazebo using real radar-data for comparisons.

The following items should be considered:

1. Investigate simulation of Radar operations in Gazebo
2. Investigate Gazebo as a tool for visualizing and simulating drone operations
3. Investigate and compare the different use-cases for ROS and DUNE
4. Implement a range-only radar as a sensor in Gazebo
5. Test the range-only radar in simple and cluttered environment
6. **Compare** simulated radar data with provided real radar data in a realistic environment.
7. Investigate algorithms for spatial environment mapping using radar data, and discuss different representation mechanisms (occupancy grid, etc)
8. Implement and test the most promising mapping algorithm in ROS. (python or C++)
9. Conclude findings in a report. Include Matlab/Python/C-code as digital appendices together with a user-guide.

Start date:	2018-01-08
Due date:	2018-06-11

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Thor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
Co-Supervisor:	Dr. Kristian Klausen, Dept. of Eng. Cybernetics, NTNU

Preface

This thesis is submitted in partial fulfillment of the requirements for the Master of Science degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU).

The master thesis builds upon the project thesis done during the autumn semester of 2017 (August to December). I would first and foremost like to thank my supervisor Tor Arne Johansen and co-supervisors Kristian Klausen and Morten Fyhn for their assistance. Further, I would like to thank the support of my friends and family through the completion of my grade at NTNU. Literature and research presented have been cited and viewed to be coming from reliable sources. Unless specifically mentioned, all code and figures are produced by me. NTNU UAVlab provided all hardware related to UAV testing. I would like to give special thanks to Kristian and Morten for their assistance in generating the required data sets to complement my thesis.

David Chen Billdal
Trondheim, June 2018

Abstract

The use of unmanned aerial drones for inspection of industrial environments have great potential in lowering both cost and risk. Development of such systems are going on at full speed, but many hurdles remain, especially that of creating robust sensors for object detection. The main obstacle is to find a sensor for object detection that is small, cheap, accurate and can fit on a small flying drone. Radar is proposed for this purpose, but little testing has been performed on smaller flying drones to either confirm or deny this prospect. New tools will need to be developed to increase the development process. In this thesis, a simulation framework has been implemented that allows for real-time testing of radar sensors within a 3D environment. The system is based on the software packages of ROS and Gazebo, using the ray-tracing technique to simulate the effects of electromagnetic waves on the environment. A radar sensor model was developed in Gazebo based on a 3D scanning lidar. The radar performed well through designed simulations, providing similar output to that of the radar sensor it simulates. In addition, a mapping technique called Occupancy Grid Mapping was implemented to complement the radar sensor, expanding its application area. The mapping technique showed good results, mapping clustered environments with high accuracy.

Sammendrag

Bruken av ubemannede luftfartøy for inspeksjon av industrielle miljø har et stort potensiale for redusering av kostnad og risiko. Utviklingen av slike systemer er i full gang, men mange utfordringer må løses. For objektdeteksjonssensor er det satt flere krav: den må være liten, billig, nøyaktig og kunne få plass på mindre luftfartøy. Radar har blitt foreslått som en sensorteknologi med stort potensial, men lite testing av en slik sensor har blitt utført på små autonome droner. For å kunne forbedre utviklingstiden av en slik sensor må nye verktøy utvikles. I denne oppgaven har et simuleringsrammeverk blitt utviklet for simulering av radarsensorer, og en radarsensor modell basert på raytracing er implementert i simulatoren. Systemet er basert på datapakkene Gazebo og ROS. Radaren har god ytelse gjennom forskjellige simuleringer og har en data produksjon som er temmelig lik radaren den er basert på. Occupancy Grid Mapping ble implementert som en kartleggingsmetode for å komplementere radarsensoren. Metoden viste gode resultater ved kartlegging av rom med flere objekter.

Contents

List of Tables

List of Figures

Nomenclature

EM	Electromagnetic
GO	Geometrical Optics
GUI	Graphical User Interface
ROS	Robotic Operating System
SAR	Synthetic Aperture Radar
SDF	Simulation Description Format
UAV	Unmanned Aerial Vehicle
URDF	Unified Robot Description Format
UWB	Ultra Wideband

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs) is a tool that is being utilized more and more in the modern world. They are entering both the consumer and the industrial arenas, playing a bigger and bigger part of our society. They can complete basic tasks like taking pictures to more demanding industrial or military functions as surveillance or target-tracking. As a result, there is a significant demand for sensors that can meet the specifications needed to be mounted on an autonomous drone.

Inspection of cargo holds and tanks on ships is a cumbersome and costly task. To do an inspection, scaffolds have to be set up and down multiple times. For this reason, routine inspections set units out of operation for long periods of time. Autonomous inspection using UAVs would drastically reduce the time needed to inspect the cargo holds and tanks, additionally reducing the workforce required.

UAVs needs precise and accurate position and object detection sensors to complete their tasks reliably. Either when used for anti-collision or merely following a given task, it needs robust sensors that can provide accurate information about the surroundings. Many different sensors are in use today, cameras, LIDARs, sonars and more. But there hasn't been many implementations with radar sensors, an interesting sensor as it can be both small and cheap. Consequently, it can be mounted on a smaller sized UAVs without reducing the mobility of the platform considerably. Research on this topic has been lacking, and there might be many opportunities in the implementation of radar for UAVs.

To develop new sensors and new technology inexpensive and fast, a simulation framework is preferred. Well designed simulators can reduce costs of testing and provide output data comparable to real-world data. For a systems engineer, this is more valuable than pure radar simulations, because these cannot give application specific results. Further, the need to test control algorithms in a suited environment is needed to reduce the risks related to real-world testing.

There are many challenges related to the creation of a simulation framework for radar sensors. For instance, the simulation of electromagnetic waves can be computationally costly, with most simulators choosing to use simplifications to have feasible simulations times. Also, the framework needs to be relevant enough for a plug and play attitude, where a module that works inside the simulator should have similar behavior in the real world.

1.1 Motivation

It is costly, monotone and potentially dangerous to do a routine inspection of cargo holds and tankers. Consequently, it is of great interest to be able to automatize the process. For instance, using UAVs would increase the efficiency and reduce the risks without the need for more workforce.

There are many problems related to inspection of enclosed environments as cargo holds. For instance, the reduced visibility and lack of signal from global position systems cause performance reduction in most commercial drones to date. Because the UAVs anti-collision system, navigation, and mapping functionality are dependent on accurate readings about its surroundings to perform satisfactorily, there is a need for a sensor capable of handling these environmental conditions.

Researchers at the NTNU UAVlab are developing a drone capable of high performance in enclosed and poorly lit environments. The radar sensor is proposed as a possible sensor for object detection since it has the ability to perform well in low light conditions. In addition, radar sensors can be both lightweight and small, enabling them to be easily mounted on smaller sized UAVs. Consequently making it possible to place multiple radar sensors on a single drone, allowing full coverage. Thus allowing for efficient object detection and anti-collision.

1.2 Previous Work

Automated inspection is a heavily researched topic, mainly because of the big cost savings that are possible. The research covers a wide field; Inspection of vessels and cargo holds using robotic crawlers that analyze the hull [1] [2] [3], inspection of wind turbines with UAVs [4] [5] [6] [7], inspection of buildings [8] [9] [10] [11]. Specifically for cargo hold inspection, the project MINOAS [12] has resulted in many interesting new methods for autonomous inspection, as given in [13] [14] [15] [16] [1]. INCASS is also another project that focuses on autonomous inspection of cargo holds [17].

Radar applications on smaller moving platforms have been researched for a while, both for military and civilian purposes. [18] presents a compact planar 24GHz quasi-Yagi antenna which uses a dipole driver, specialized for being mounted on UAVs. In paper [19], a miniaturized SAR system is developed and optimized for utilization in UAVs, including a motion compensating SAR-algorithm. [20] presents an FMCW-Ground penetrating radar mounted on a mini UAV, capable of generating underground images using SAR techniques. The Daredevil project [21], a military project, presents a small UGV mounted with a radar, enabling it to see through foliage and adverse weather conditions. [22] has also done a similar project.

Using Gazebo in conjunction with ROS is a common platform for simulation of UAV testing scenarios. [23] gives a thorough explanation of how to simulate a UAV in Gazebo and ROS. A package called the `hector_quadrotor` package developed by TU Darmstadt [24] is available as a ROS package, including implementations as described in [23].

Creation of different methods to simulate waves in Gazebo has been done by [25], which explores how rays in Gazebo can be used for simulating sonar waves. By using rays, different methods are implemented to represent how waves travel and react with objects physically. For instance, angle affection of the rays and using different reflection coefficients for the objects simulated.

There exist many different methods to simulate the data generated by radar sensors. [26] shows how the simple program of Blender is able to utilize the built-in ray-tracing methods to simulate SAR-images. [27] proposes a hybrid solution to SAR simulators, avoiding the problem of very high processing demands that follow when electromagnetic waves are simulated. [28] developed a SAR simulator that is capable of generating high precision SAR images of 3D objects using ray-tracing techniques.

1.3 Contributions

This Master Thesis presents a simulation framework that is capable of simulating radar sensors placed on a UAV, along with different scenarios and application experiments. Specifically, mapping with radar sensors using the Occupancy Grid Mapping method is implemented and tested.

This Thesis will specifically cover the following points:

- **Development of a simulation framework for radar.** The development of the simulation framework using Gazebo, ROS, and DUNE as a whole. Creation of interfaces between the different platforms to allow for a wide range of tools.
- **Development of a radar sensor model based on ray-tracing techniques.** A radar sensor model that can handle different angles and reflection values is presented. The workings of the model are thoroughly explained, along with its weaknesses and strengths.
- **Mapping with radar sensors.** Showcase the Occupancy Grid Mapping technique concerning multiple radar sensors.
- **Simulation Experiments.** Experiments that highlight the functionality of the radar sensor, mapping implementation and the simulator as a whole.

1.4 Outline

The Master Thesis is divided as follows:

- **Chapter 2: Radar Theory.** An introduction into how radars work, and the underlying physical properties that make it possible to use electromagnetic waves for object detection.
- **Chapter 3: Software Framework.** A presentation of the framework, including a description of Gazebo, ROS and DUNE, and how they interconnect together to create the software framework.
- **Chapter 4: Radar Sensor Model.** Development and implementation of the radar sensor model. The different methods used to simulate radar waves are presented.
- **Chapter 5: Mapping for Autonomous Vehicles using Radar Sensors.** Presents different sensors that are used for mapping, before highlighting the radar sensor. The most promising mapping method is chosen and implemented.
- **Chapter 6: Simulation Experiments and Results.** Design and execution of several simulations accompanied by the results. Showcasing different scenarios and application that are of interest.
- **Chapter 7: Discussion.** A discussion of the results obtained during testing of the radar sensor model and the mapping method, focusing on accuracy and applicability.
- **Chapter 8: Conclusion and Future Work.** Concludes the findings in this Thesis and suggests future work.

Chapter 2

Radar Theory

This chapter explains the principles of radar systems, including modern applications and how it might apply to flying platforms like UAVs. Different kinds of radar are described, from bare-bones beam scanning radars to advanced Synthetic Aperture Radar systems.

2.1 History

Radar is an object-detection system developed by several nations just before and during the second world war. It was coined as RADAR by the United States, the acronym meaning RAdio Detection And Ranging. During and after the war, the system was mainly used for detection of aircraft and missiles. The system has since been developed further and is in use both for military and civil applications today. Radars systems today can range from being able to fit inside a hand to covering hundreds of square meters. Typical use cases are air traffic control, weather surveillance, aircraft anti-collision systems, ground-penetrating radar for geological observations, and outer space surveillance.

2.2 Theory

Radar is an electromagnetic sensor used for detecting and locating objects, with the ability to find objects at great distances even in adverse conditions as darkness or rain. It uses the properties of electromagnetic waves to detect objects, transmitting electromagnetic energy and listening for echoes from potential objects of interest. These objects can be everything from aircraft, ships, humans, birds, insects or merely a wall. Different techniques are used to obtain location, velocity, size, and shape. Radar is mostly used to find the position and velocity of objects.

2.2.1 Radar Types

There are mainly two types of radar systems: Continuous Wave and Pulsed. Continuous wave radar systems illuminate an area with constant electromagnetic signals. To allow for range deduction using this technique, frequency modulation techniques has to be utilized. Pulsed radar system uses a transmitter to illuminate target areas with pulses of known size and duration. A type of pulsed radar is the Ultra Wideband (UWB) radar. It transmits signals at a wider frequency with a light power spectrum. Thus leading to a low interference of conventional transmissions operating in the same range. UWB signals are usually generated by transmitting pulses with short durations. As a consequence, there is little overlap with the signal reflections and the original pulse.

Pulsed radar systems are more intuitive and will be assumed for the rest of the chapter as a simple means to explain the workings of general radar systems.

2.2.2 Range Calculations

To calculate range, electromagnetic waves are sent toward the object of interest. By measuring the propagation time (t), the range to the object can be calculated as

$$R = \frac{1}{2}ct \tag{2.1}$$

Where c is the speed of light in vacuum. $\frac{1}{2}$ comes from the fact that the electromagnetic wave has to travel to the object and back again. When more advanced systems are used, for example SAR, the electric properties of the propagation medium has to be taken into account if it differs from that of vacuum [29].

2.2.3 Object Detection

The receiver of radar systems listens for echoes to determine object detection. When a signal is received that is of similar size and duration of the generated electromagnetic signal, the system determines that the signal originates from its transmitter and therefore has hit an object that has reflected the signal back.

A simple illustration of a radar operation is presented in figure 2.1. Here a beam-scanning radar is pulsing out electromagnetic waves and receiving an echo of the signal from a UAV. By using equation 2.1, it is possible to measure the distance to the object. The direction of the object has to be in the same direction that the antenna is pointing, but this direction is greatly affected by how big the radars azimuth is. As a result, many antennas have small azimuth openings to allow for higher accuracy of object location.

2.3 Synthetic Aperture Radar

SAR is a form of radar used to create 2D or 3D images of objects and landscapes. It differs from simple beam-scanning radars by utilizing the movement of the radar to provide finer spatial resolution than static beam-scanning radars.

SAR uses the principles of Doppler shift to give the system more information about the object. An object that moves towards the radar will shift the echo waves relative to the transmitted waves, making it possible for the radar system to detect the radial speed of the object. Moreover, it is possible to get information about the size and form of the object, as waves will be shifted differently dependent on the shape of the object. An example of this is given in figure 2.2, it should be noted that this is usually referred to as Inverted SAR, as the object to be detected is moving, and not necessarily the

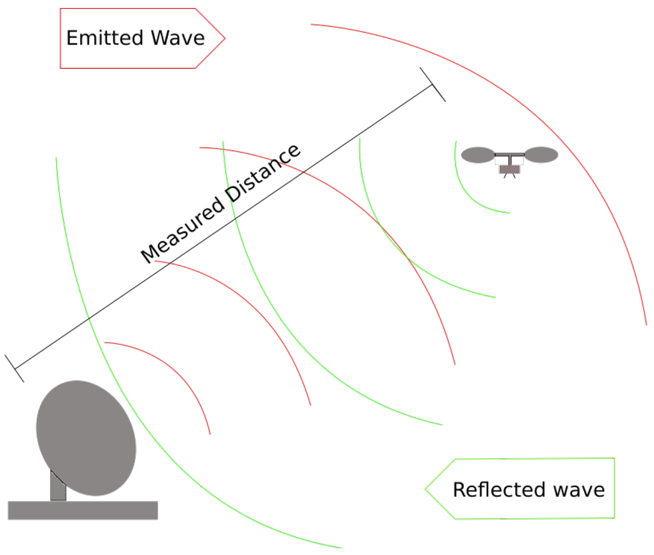


Figure 2.1: A beam-scanning radar used for object detection

radar sensor.

To create a bigger than physical aperture, SAR systems have to be on moving platforms. While moving, it continually illuminates the scene by sending out pulses, which are then received and recorded for processing to create SAR images. The processing combines the different recordings, in effect creating what is known as the synthetic antenna aperture. For an example of how the synthetic antenna aperture is imagined, see figure 2.3. Typically, the larger the size of the synthetic antenna aperture, a higher resolution on the SAR image is obtained.

Certain requirements must be in place for a SAR system to work effectively:

- Stable and robust transmitter
- SAR processor capable of generating images from the data
- Precise knowledge of the movement of the radar, including its velocity

Modern SAR systems can produce images with a resolution of a few centimeters

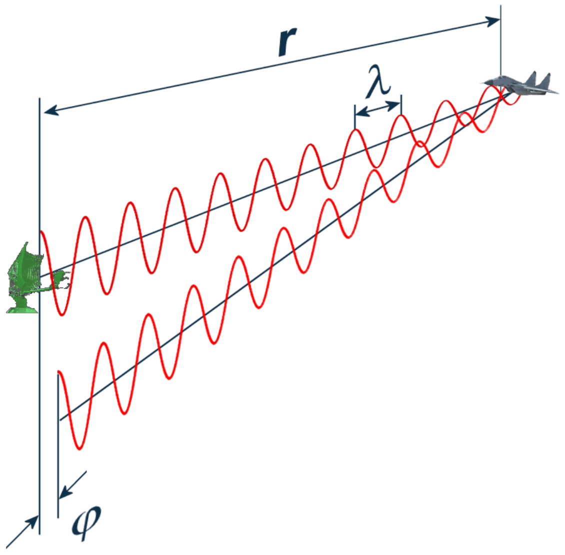


Figure 2.2: An airplane is hit by an electromagnetic wave. As the aircraft is flying towards the radar antenna, the echo signals wave length is reduced. Taken from [30]

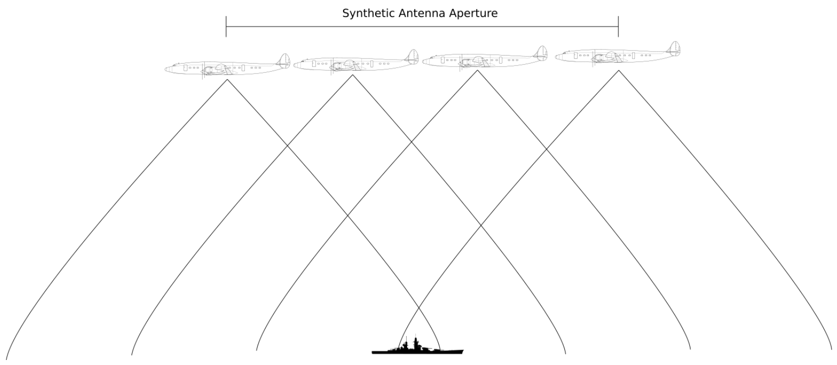


Figure 2.3: A synthetic antenna aperture is here created as multiple readings from different positions are combined together to provide a finer spatial resolution image of the ship

while providing consistent high performance when mounted on airplanes. A review of the SAR system has been done in [31].

2.3.1 Simple SAR Summation Algorithm

A simple algorithm is used in this thesis to showcase a SAR application. The algorithm was worked on in partial-cooperation with co-student Snorre Jablonski. This Thesis is concerned with simulated radar data, while Mr. Jablonskis Thesis [32] is concerned with real-world data sets. A comparison between the two applications is therefore natural and will be reviewed later.

The algorithm uses a summation method to generate SAR images. Using the position of the UAV, the reflected magnitude of the signal and the distance measured, the algorithm sums up all reflected signals and interpolates them onto a two-dimensional matrix. The values on the matrix are then given different color codes to match their value, generating a SAR image. In essence, the SAR algorithm sums up all the received reflected values at each sensor update and maps it in a one-dimensional straight line in front of the sensor. This method doesn't generate an image similar to what optical sensor would produce. Instead, hyperbolas are created around object locations, as seen in figure 2.4. The images are then processed for object position estimation. Namely, the top of the hyperbola is found and is then assumed to be the position of the object.

This algorithm does not consider the Doppler shift, and as such much information about the target is lost.

2.4 Error Sources

The sources of error in radar system has been explored for many years. [33] gives a good introduction into which sources affect the radar performance. [34] gives a more detailed description of how radar systems work, modeling methods, error sources, and algorithms for data processing.

Error sources that are typical to radar systems are receiver noise, clutter, and multi-path. These are explained in detail by [34].

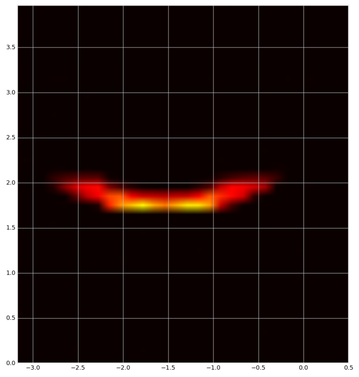


Figure 2.4: A typical hyperbola that is generated when the Simple Summation Algorithm is running. The flat part would here be assumed to be the location of the object

It is here given a quick introduction to the error sources and how they affect the system:

- Receiver noise originates from the random motion of electrons at the input of the receiver. This is best modeled by a random walk, as described in [34].
- Clutter is non-important objects within the environment that create an echo response which is received by the radar receiver. These objects can clutter the signal and make it hard to impossible to detect the objects of interest.
- Multi-path errors originate from the transmitted signal of the radar getting bounced around before entering the receiver. In essence, multiple paths for the wave causes the distance measurement of the radar to be miscalculated as seen from equation 2.1, as the time taken is much longer than the actual distance to the objects. An illustration of the multi-path problem can be seen in figure 2.5. [34] gives a more thorough description of error-models for multi-path.

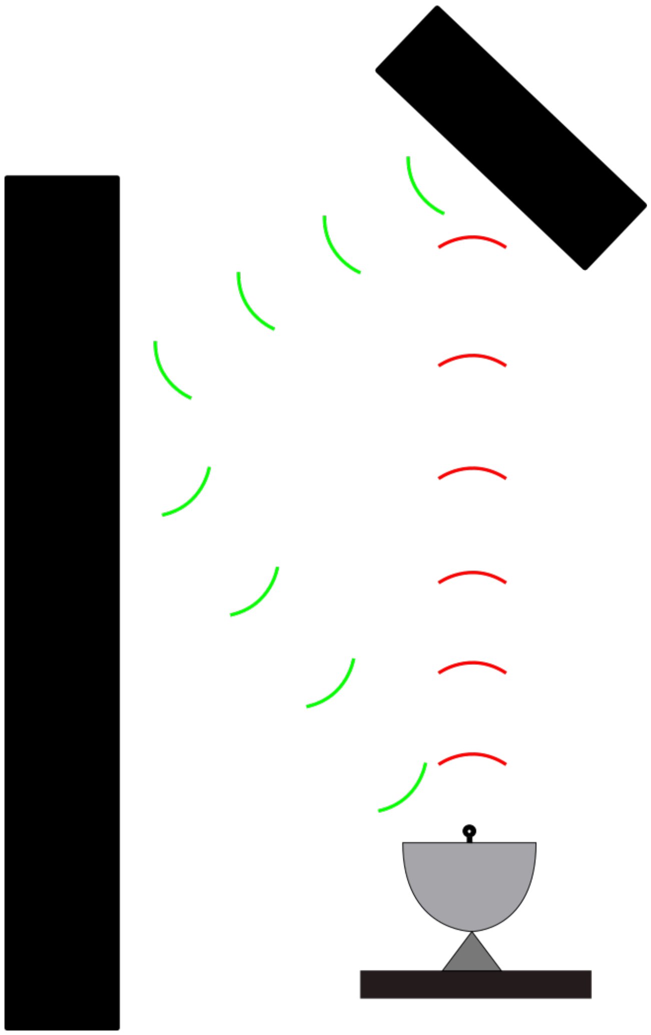


Figure 2.5: A multi-path error is created as the EM waves is bounced around the environment before arriving at the receiver, causing a miscalculation of the assumed distance to the object

Chapter 3

Software Framework

This chapter explains how the simulator framework is built up. The first part of the chapter goes into the different software that is chosen to create a complete framework for simulating a radar sensor on a quadrotor, a multi-rotor helicopter that is lifted and propelled by four rotors. The software features are explained and reasons given for why it is included in the framework. In the second part, an overview is given for how the system is interfacing between the different software platforms.

A complete overview of the system architecture can be seen in figure 3.1.

3.1 Goals of the Framework

The goal of the framework is to simulate the data received from a quadrotor mounted with a radar sensor.

To complete this goal, various software that covers different areas have to interact together. A powerful 3D physics engine and robust tools for robotic software development are needed. Further, to allow for a wide range of tools, the DUNE runtime environment for onboard system software is presented.

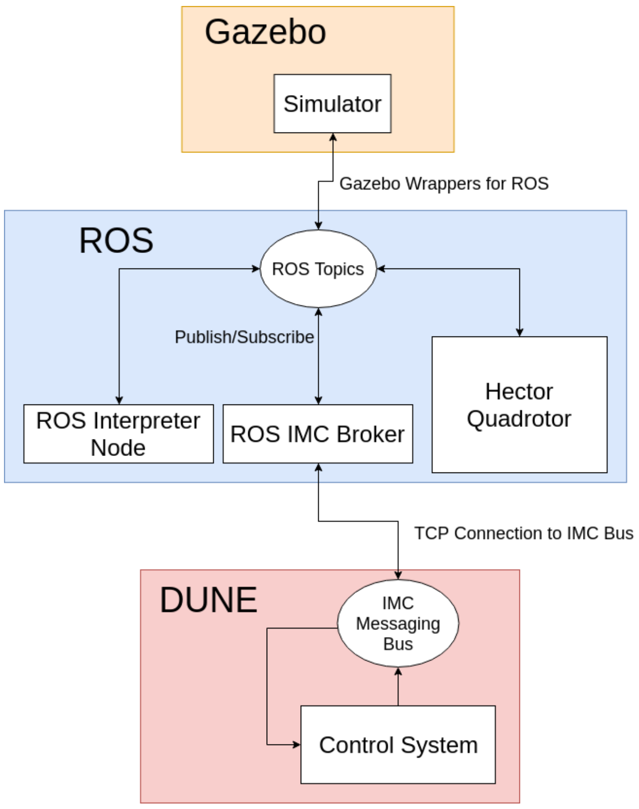


Figure 3.1: An overview of the complete framework and interface paths.

3.2 Gazebo

Gazebo is a simulator which was started on in the fall of 2002 at the University of Southern California. It was mainly developed by Dr. Andrew Howard and Nate Koenig, until 2009 when John Hsu, a Senior Research Engineer at Willow, integrated ROS into Gazebo. In 2012 Open Source Robotics Foundation became the steward of the Gazebo project, actively updating the software and reviewing pull-requests. Gazebo is an open-source project and is actively supported by a big robotics community.

The main purpose of Gazebo is to simulate robots in various scenarios, giving the developer a range of tools to develop and understand the ongoing simulations. As it has its roots from the ROS goal of robotics development, Gazebo is mainly focused around simulating robots. Consequently, it is not suitable for simulation scenarios that deviate much from this focus.

3.2.1 Simulator

Gazebo is a powerful and versatile simulator. It utilizes multiple physics engines, namely ODE, Bullet, Simbody, and DART. For rendering the ongoing simulation, Gazebo uses OGRE, providing the user with a detailed representation of the simulation including high-quality lighting, shadows, and textures. A view of the simulator can be seen in figure 3.2.

Gazebo is highly flexible with regards to development, enabling the creation of a wide array of different simulation types. Allowing high customizability for properties like inertia, collision geometry, visual geometry, lighting, and much more. Moreover, it is possible to include meshes that are created in other 3D creation software like AutoCAD and Blender.

The primary focus of Gazebo is to simulate robots, and this is made apparent in which tools the simulator offers. For instance, the tight implementation of sensors and the Gazebo environment allowing for simple data handling and visualization of sensor data. Further, the fact that Gazebo is based on the Robotic Operating System allows for easy integration with algorithms and models designed within that environment. These points will be further elaborated on in the following section.

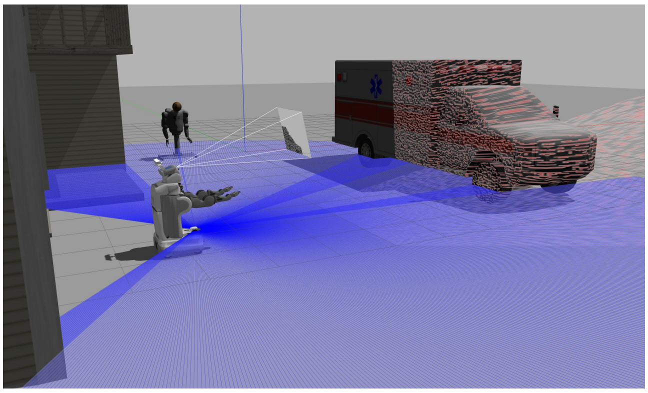


Figure 3.2: A view of the Gazebo simulator

3.2.2 Tools and Features

Gazebo gives the user many tools for environment creation and robot simulation. In this section, some of the most prominent tools and features are presented.

Sensor Integration

Sensors in Gazebo are tightly integrated with the simulator. There is a base set of sensors implemented in Gazebo, but each of these sensors can be changed through the use of plugins. For instance, a generic laser sensor can be changed to a short-range flashing laser through the use of a plugin. The sensors supported include cameras, contact sensor, force and torque sensor, GPS sensor, IMU sensor, ray sensor and sonar sensor. Each sensor is highly modifiable either through the SDF description file or the use of plugins, allowing for quick testing of different properties related to the sensor. This includes the range, noise, and FOV to mention some of the general properties.

Each sensor in Gazebo has a graphical window to show, in real-time, what the sensor is outputting. As a result, the user can understand the decisions made within the environment based on the data the robots are seeing.

Model Creation

Model or world creation in Gazebo is straightforward. Design and creation of environments are mainly through writing SDF files (see Figure 3.4) or through the GUI (see Figure 3.3). Gazebo also allows for models to be imported from meshes created in other software programs, for example, Blender and AutoCAD. In addition, Gazebo comes with a big library of models and worlds that can easily be added to any simulation. ROS also allows for easy conversion of their models into Gazebo models, increasing the scope of pre-made models.

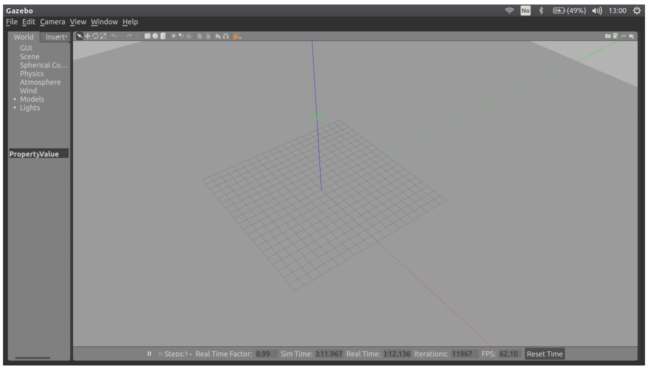
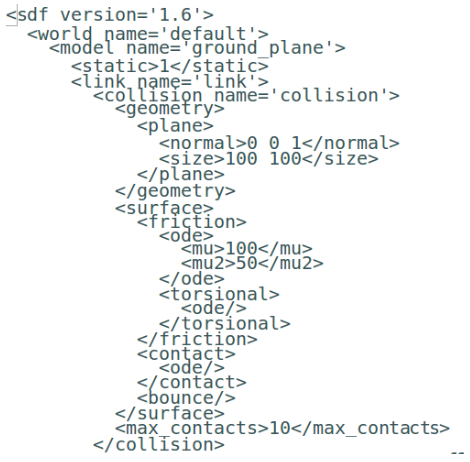

Gazebo
File Edit Camera View Window Help
World
Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps:1 Real Time Factor: 0.99 Sim Time: 3:11.967 Real Time: 3:12.136 Iterations: 11967 FPS: 62.10 Reset Time
13:00

Figure 3.3: A view from the GUI of Gazebo

Remote Simuation

Gazebo allows for remote simulation. The software platform is split into two components, the processing server called gzserver, and the graphical client called gzclient. These two components communicate through the IP protocol, usually communicating over the local network on the computer. Namely, it allows for communication through the Internet, making it possible to run high demanding simulations on a different platform.



```
<sdf version='1.6'>
  <world name='default'>
    <model name='ground_plane'>
      <static>1</static>
      <link name='link'>
        <collision name='collision'>
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>100 100</size>
            </plane>
          </geometry>
          <surface>
            <friction>
              <ode>
                <mu>100</mu>
                <mu2>50</mu2>
              </ode>
              <torsional>
                <ode/>
              </torsional>
            </friction>
            <contact>
              <ode/>
            </contact>
            <bounce/>
          </surface>
          <max_contacts>10</max_contacts>
        </collision>
```

Figure 3.4: A SDF file describing a world

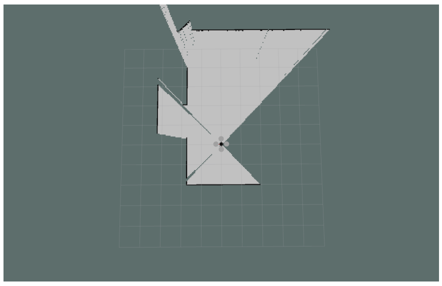
3.3 ROS

ROS was developed in 2007 by Stanford Artificial Laboratory under the name Switchyard, before changing the name to ROS in 2009 [35] [36]. The project aims to create a robust and good software framework for robot software development.

The Robot Operating System, ROS for short, is a framework providing libraries and tools to help software developers create and test robot applications. The framework includes hardware abstraction, device drivers, libraries, visualizers, message-passing, package management and more. It is not purely an operating system for robots but provides functions that are similar.

ROS is made with the aim of simplifying the creation of robust and complicated robot software. ROS code is modular and portable, as packages can be linked to new projects with minimal effort. C++, Python, and Lisp are supported languages and can be used interchangeably in a project.

As of this moment, ROS is only supported for Ubuntu and OS X. The community might develop versions for other platforms, although the project does not officially support these.


(a) Occupancy mapping within Rviz

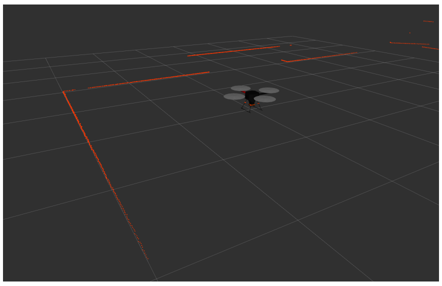

(b) LIDAR data presented in Rviz

Figure 3.5: Two different data representations within Rviz

3.3.1 Tools and Features

ROS includes many tools and features that make the development process manageable and structured.

Rviz

Rviz is a 3D visualizer for displaying sensor data and state information from ROS. The interface allows for a wide range of different data to be visualized, from LIDAR data to occupancy grids. A typical view of a LIDAR view can be seen in figure 3.5b and an occupancy grid view can be seen in figure 3.5a.

Modularity

The ROS system was developed to be modular and peer-to-peer. Systems based on ROS contain a network of nodes, each handling a specific function for the system. The nodes communicate through topics and services. Topics is an asynchronous communication method where one or more nodes publish messages to a topic, where the message is distributed to all nodes that subscribe to the given topic. Services are similar to regular function calls, where a node can call on specific functions from other nodes.

At the center of the communication is the ROS master. The master keeps track of all nodes in the system and sets up communication between nodes when needed.

No data is directly passed through the master. Instead, the master gives each of the communicating nodes the necessary information to set up TCP/UDP connections.

RQT

RQT is a Qt-based framework used as an alternative GUI for ROS. The framework contains many implemented and tested GUI developed by the team at ROS or independent developers. The strength of RQT is that it allows for costume designed GUIs. There is a multitude of different views available, the most relevant for this project is the `rqt_publisher` which makes publishing to topics possible without the use of console commands, `rqt_graph` which visualizes the ROS computation graph and `rqt_top` that shows the processing used by each node including CPU and memory usage. An illustration of these features can be seen in figure 3.6.

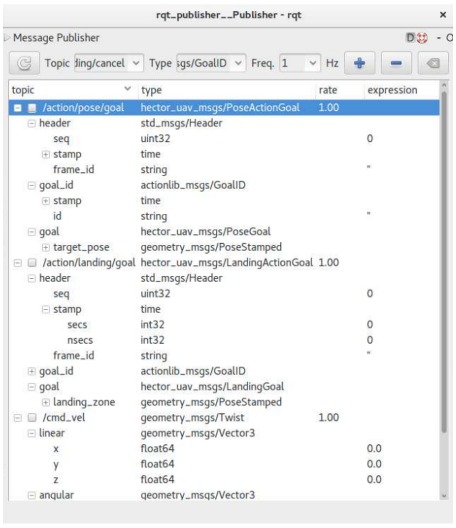
The RQT tool allows for a graphical alternative to the mainly console oriented system of ROS. Thus reducing the learning curve for new users of the system.

3.3.2 Hector Quadrotor - ROS Package

The significant support around the ROS platform has created a diverse and versatile library of ROS packages. Each package is easily implemented as a result of the modular architecture of ROS.

There is no need to recreate a model of a quadrotor in ROS. Thus a ROS package called Hector Quadrotor was included into the framework. Hector Quadrotor is part of a collaborated effort at Technische Universität Darmstadt to develop a realistic model of a quadrotor. The Hector team (Heterogeneous Cooperating Team of Robots) has worked with quadrotor research for a longer period, developing interesting new methods. For instance, their SLAM (Simultaneous Location and Mapping) called Hector-SLAM.

The model dynamics defined by the Hector team are, like all models of real-world objects, a simplification. But the model dynamics as thrust has been thoroughly tested in wind-tunnels to reflect real-world values. The model properties have also been confirmed with real-world testing of quadrotors, showing small discrepancies. A full

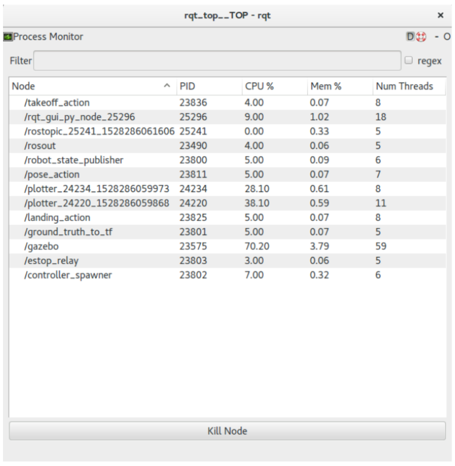

rqt_publisher__Publisher - rqt

Message Publisher

Topic ding/cancel | Type sgs/GoalID | Freq. 1 | Hz

topic	type	rate	expression
/action/pose/goal	hector_uav_msgs/PoseActionGoal	1.00	
header	std_msgs/Header		
seq	uint32		0
stamp	time		
frame_id	string		''
goal_id	actionlib_msgs/GoalID		
stamp	time		
id	string		''
goal	hector_uav_msgs/PoseGoal		
target_pose	geometry_msgs/PoseStamped		
/action/landing/goal	hector_uav_msgs/LandingActionGoal	1.00	
header	std_msgs/Header		
seq	uint32		0
stamp	time		
secs	int32		0
nsecs	int32		0
frame_id	string		''
goal_id	actionlib_msgs/GoalID		
goal	hector_uav_msgs/LandingGoal		
landing_zone	geometry_msgs/PoseStamped		
/cmd_vel	geometry_msgs/Twist	1.00	
linear	geometry_msgs/Vector3		
x	float64		0.0
y	float64		0.0
z	float64		0.0
angular	geometry_msgs/Vector3		

(a) The RQT Publisher GUI allowing for simple publishing of messages


rqt_top__TOP - rqt

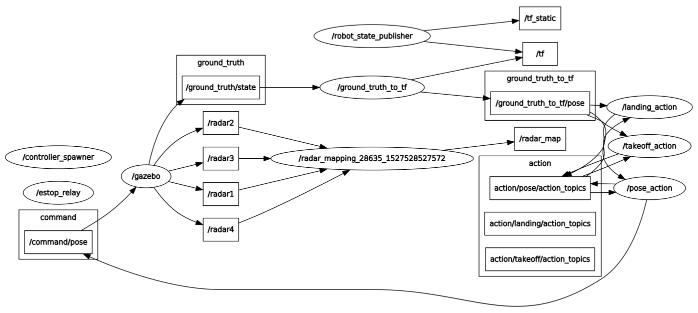
Process Monitor

Filter ☐ regex

Node	PID	CPU %	Mem %	Num Threads
/takeoff_action	23836	4.00	0.07	8
/rqt_gui_py_node_25296	25296	9.00	1.02	18
/rostopic_25241_1528286061606	25241	0.00	0.33	5
/rosout	23490	4.00	0.06	5
/robot_state_publisher	23800	5.00	0.09	6
/pose_action	23811	5.00	0.07	7
/plotter_24234_1528286059973	24234	28.10	0.61	8
/plotter_24220_1528286059868	24220	38.10	0.59	11
/landing_action	23825	5.00	0.07	8
/ground_truth_to_tf	23801	5.00	0.07	5
/gazebo	23575	70.20	3.79	59
/estop_relay	23803	3.00	0.06	5
/controller_spawner	23802	7.00	0.32	6

Kill Node

(b) The RQT TOP GUI allows for overview of resource usage for each ROS node



(c) The RQT graph GUI showing the ROS computation graph

Figure 3.6: Three different RQT GUIs

overview of the system and the work done to create the system can be found in [23].

3.4 DUNE

DUNE is an embedded software platform developed at Universidade do Porto [37]. It is an independent platform and can run on most platforms that can run a small Linux distribution. DUNE is part of the LSTS toolchain, which includes a minimal Linux distribution called Glue, DUNE, Ground Control Station called Neptus and Ripples for cloud services.

The primary goal of DUNE is to give the developer an efficient tool for development of software controllers used in vehicles, including quadrotors and submarines. Some widely used control algorithms are implemented into the DUNE eco-system, for example Ardupilot.

3.4.1 DUNE System

The building block of DUNE is the Task. Tasks are small subprograms completing a specific functionality that can run in parallel with each other. Communication between different tasks is done through a standardized messaging system called the IMC protocol. Each task may subscribe or broadcast to specific IMC messages. The broadcasted messages are sent on the IMC bus, allowing for anyone subscribing to a particular message to receive the contents. Tasks in DUNE are supported by the DUNE core library. The library consists of concurrency support, monitoring, filtering, control algorithms, modeling and the IMC protocol. Also, the library allows for shared memory including the standard TPC/UDP functionality found in ROS.

The system, containing the core library and tasks, is compiled into a single binary executable. All tasks and hardware drivers are contained within this executable. As a result, the executable can easily be made to run on a multitude of platforms. A UAV may use the same executable as a ground-based robot.

3.4.2 LSTS Toolchain

As mentioned in the introduction to this section, the LSTS was developed by Universidade do Porto, and contains the following tools: a minimal Linux distribution, DUNE, Ground Control Station Neptus and Ripples for cloud services. An overview of the system can be seen in figure 3.7.

The toolchain is made to make the whole process from embedded software development to control of the vehicle more streamlined. A small Linux distribution called Glue allows for a robust system that does not demand big memory space, DUNE allows for control algorithm development and testing, and the Ground Control Station enables the user to have full mission control of the vehicle either during simulation or when running in the real world.

3.5 System Interface

To gain the benefits from each system, there has to be a robust interface between the different software. Because each platform has its own set of communication protocols, handling and translation have to be implemented to create an understandable and efficient communication route.

The unification of the three systems presented here was part of the project thesis, and as such is included for completeness.

3.5.1 Gazebo-ROS Interface

ROS already supports an interface with Gazebo. It wraps the gzclient and gzserver that Gazebo exposes as its API. gzclient runs on the host computer and is a graphical user interface. gzserver runs the main computations needed for the simulations. ROS uses a package called `gazebo_ros` that exposes the topics which Gazebo publish on to the ROS environment. Because of the similarities between Gazebo and ROS, the connection is very stable and creates no need for additional translation between the two systems.

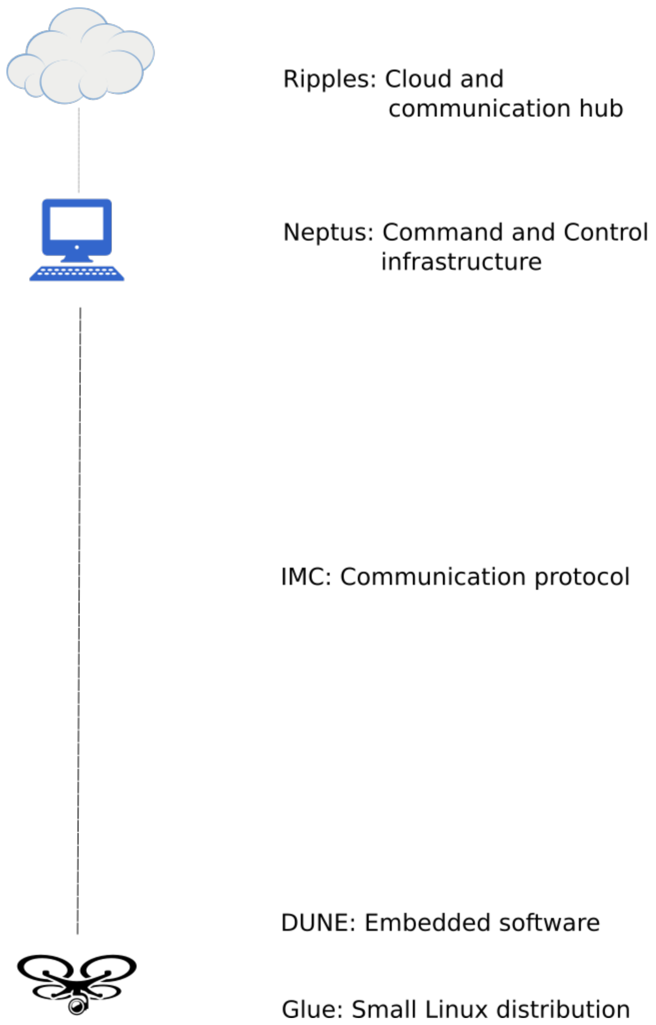

Ripples: Cloud and
communication hub
Neptus: Command and Control
infrastructure
IMC: Communication protocol
DUNE: Embedded software
Glue: Small Linux distribution

Figure 3.7: An overview of the LSTS toolchain.

The interface does add some levels of complexity to the system. There is a need to add wrappers to the output of Gazebo for ROS to work with the system. Furthermore, ROS and Gazebo have a different way of describing objects. Gazebo uses a description protocol called SDF while ROS uses its description protocol named URDF. This creates added complexity when trying to use models from ROS, as these have to be translated into SDF files manually.

3.5.2 ROS-DUNE Interface

DUNEs communication system is based on different tasks sending out messages on an IMC bus that other tasks are listening on. Similarly, ROS have nodes which send out messages to topics that other nodes are subscribing to. These two systems have their own set of protocols with their own set of message structures. Therefore, a translation between the messages has to be made.

DUNE backseat

DUNE has an option for other applications to connect to the TCP connection used by the IMC bus. Thus making it possible for other programs like ROS to be connected to the bus, for example, to receive messages referring to commanded speed.

ROS IMC broker

A package for an IMC broker has been created by UceanScan, a spin-off company from the people behind the LSTS toolchain [38]. The package contains a ROS node that is able to connect to the IMC bus, enabling it to be part of the communication system of the DUNE tasks. When a message is received that it listens for, it uses a translation module for IMC to ROS message and publishes it to the topic /IMC/In/Message. When a ROS node wants to publish to the DUNE system, it publishes to topics of similar structure, /IMC/Out/Message. The messages are read by the broker and published onto the IMC bus.

ROS Interpreter Node

The IMC Broker translates the messages in IMC as directly as it can when it publishes them into the ROS communication system. This creates a problem with understanding, as IMC message structure is not compatible with ROS message structure. For instance, the IMC message VehicleState has no equivalent message in ROS. Further, it is typical that software packages for ROS and Gazebo demand messages that are not part of the standard ROS message library.

To solve this problem, a node is developed to translate between IMC messages and their relevant counterparts in Gazebo or ROS. The interpreter node is called `interpreter_quadcopter` in the project thesis code. It contains a C++ class which handles the subscription and publishing between a simulated quadrotor in Gazebo and a DUNE controller. As it was mainly a proof of concept, only a basic interpreter was created, along with a simple controller in DUNE which sets the height position for the quadrotor.

3.6 Complete System Summary

ROS and DUNE seem to fulfill the same functionality, but there are some distinct use-cases where they differ. When a project is related to a widely researched area, there is a significant likelihood that many of the packages that are needed have been implemented in ROS. For example, developing software for autonomous cars would be easy to start working on, as many models have already been created within the ROS framework. Further, the number of tools to help visualize and support the development process reduces the complexity of the project. On the other hand, DUNE has fewer tools to aid in visualization except through the LSTS toolchain. The main strength of DUNE is related to the development of control software. Through the support of Neptus for mission control and the specialized Glue Linux distribution, the platform has many benefits compared to ROS. Moreover, the simplicity of having a single executable binary file compared to the complex ROS structure with a high dependency between modules is another strength of the DUNE system.

By unifying the system, many advantages are gained. The strength of the simulator is apparent, as it allows for testing of systems that would otherwise demand the risk of damaging property. ROS is a robust platform for robotics software development, while also boasting a big collection of packages that can be used for other projects. The DUNE system adds an alternative platform for control algorithm development, allowing for testing of its systems inside both the LSTS system and the Gazebo simulator. The communication between these three systems is clean and fast, capable of handling significant bandwidths.

Chapter 4

Radar Sensor Model

This chapter presents how the radar sensor model is implemented. First, an introduction into common methods to simulate radar sensors is given, then the implementation of the different aspects of the sensor is shown. Physical attributes will be considered, and the different design choices explained in relation to these. The last section goes into detail of how the system is implemented in Gazebo and ROS.

4.1 Radar Simulation Methods

The most accurate method to simulate EM waves is to use EM formulas, as this is a precise theory. The finite-difference-time-domain method [39] could for example be used for simulation of EM waves. Even so, the method is not very feasible for simulation of big areas, as the volume of the scene has to be divided into cells not larger than $\frac{1}{10}$ of the wavelength [39].

The simplified geometrical optics (GO) approach makes it possible to simulate radar with less computational demand than the direct EM method. It uses a set of small facets with known reflectivity and geometrical shape [40] [41]. The EM wave is simulated as an optical wave, making it possible to simulate effects as reflection and multi-path movement. Unfortunately, the method ignores the wave effects. A typical engine uses ray-tracing to allow for the simulation of optical waves. The rays are followed out from the sensor opening, and their interaction with the environment is calculated.

Efforts are made to combine the methods to complement each other [42] [43]. The idea is to let the GO methods handle the larger elements within the scene while the EM wave methods handle smaller elements that are within its resolution and the computing power restrictions.

The physical optics method is another conventional approach to radar simulations [44]. The approximation is based on estimating the field on a surface using ray optics, then integrating that field over the surface to calculate the transmitted or scattered field. This method is preferred when the phenomena of interest cannot be simulated by the GO method, specifically phenomena related to how waves interact. These include interference, diffraction, and polarization.

The geometrical optics approach, or ray-tracing method, is chosen to simulate radar signals. The reason is that it is less computationally demanding than full wave EM methods, and has already been implemented as the preferred technique to simulate radar signal in many simulators [45]. It has been shown to perform well when simulating, among others, ground penetrating radar [46] [47], finding main scattering points for vehicles [48] and recreation of high resolution SAR images [49] [28]. The technique has already been used for a sonar sensor in Gazebo with many of the same properties related to radar sensors [25].

4.2 Electromagnetic Wave Simulation using the Ray-tracing Method

The primary goal of the simulator is to output radar data that has an error that is acceptable for industrial application development and testing. To achieve this, the calculations are based on real-world physical properties to make the system more robust to changing environments.

4.2.1 Main Principle

Ray-tracing is a simple method that is intuitive while allowing for great complexity in what it simulates. The technique follows rays through the environment calculating the effects they have on the virtual objects. As a result, the method can simulate advanced effects such as reflection, scattering and dispersion phenomena. Moreover, the method allows for a high degree of scalability, as the number of rays is directly correlated with the resolution of the images generated.

4.2.2 Radar Sensor Model Property Estimations

Noise

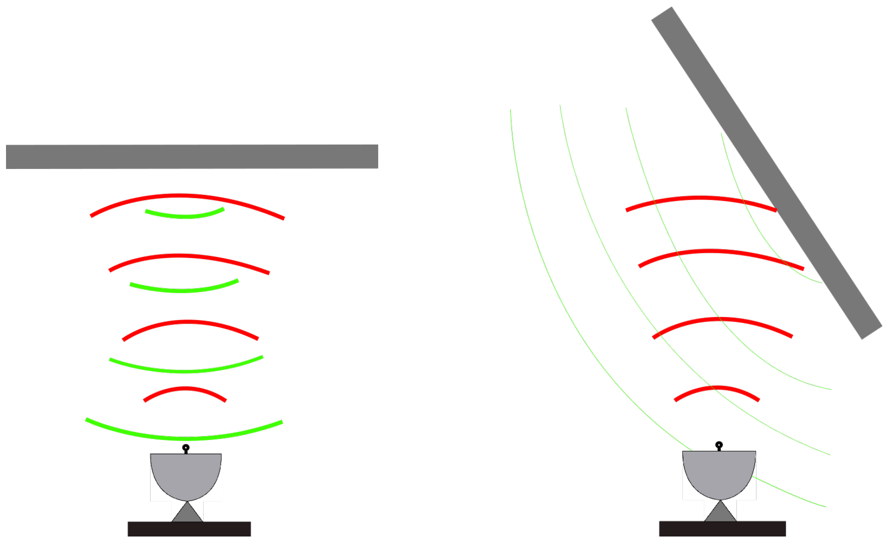
Noise is prevalent in radar sensors and can cause a considerable reduction of accuracy in specific scenarios. The effects of noise have already been discussed in chapter 2. Noise will be simulated as white noise because it has been shown to be the most accurate statistical description of the noise experienced at the receiver input [34].

Angle Calculations

An essential factor to consider when simulating electromagnetic waves is the fact that the angle the waves hit an object, greatly affects the returned echo [34]. A steel wall facing a radar sensor will reflect a more intense echo than the same steel wall hit by the same echo while facing away, see figure 4.1 for an illustration. The same effect is why stealth vehicles have sharp edges, as to reflect as little as possible when hit by an EM wave.

To simulate the same effect with the ray tracing method, every rays angle on their respective object has to be calculated and used to affect the reflected return value of the ray. A similar method has been done in Gazebo for a sonar sensor [25]. The mathematical equations are as follows:

$$\theta = \arccos(\frac{v_r * v_o}{|v_r||v_o|}) \tag{4.1}$$



(a) The radar wave hits a wall facing the radar sensor, generating a strong return echo signal

(b) The radar wave hits a wall facing away from the radar sensor, generating a weak echo signal

Figure 4.1: Steel wall reflecting radar waves

$$I_R = I_m * \cos(\theta)(\frac{|v_r|}{R_{max}})^2 \tag{4.2}$$

Here θ is the angle between the objects normal vector, v_o and the ray, v_r. I_R is the total reflected intensity and I_m is the reflection coefficient defined in the simulator as the materials reflection value. R_{max} is the maximum range of the sensor. An illustration of this can be seen in figure 4.2.

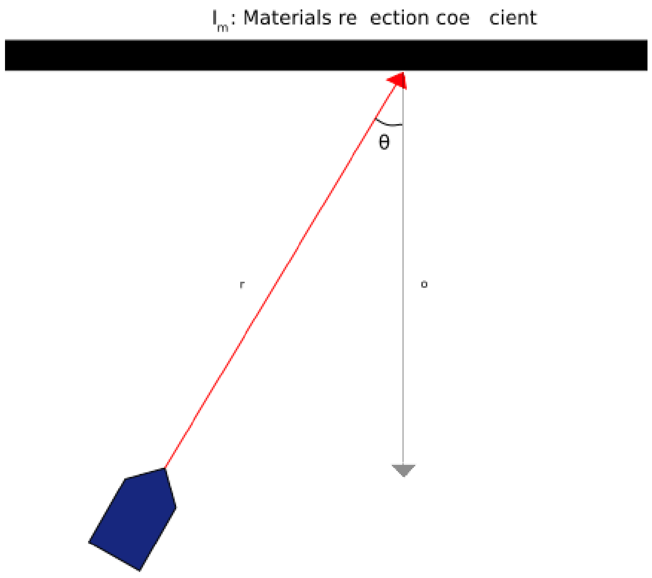


Figure 4.2: A single ray hitting a wall, with the symbols used in calculations for returned echo intensity

Surface Reflections

The reflection intensity from objects varies with their electric conductivity. A steel wall reflects an intense echo of the signal while for example, a similarly sized wooden wall would reflect a weak echo signal. There are different sources on this topic that handles the modeling of the backscattering coefficient [50] [51] [52]. The objects reflection coefficients are in this Thesis not defined based on typical values, but from a real-world sensor being used on the objects in question, where the relative reflection between objects is considered. As a result, the data generated by the radar sensor is more accurate as the objects reflection coefficient better follows the real world.

4.3 Implementation

The implementation of the radar sensor model is done within Gazebo and is split into sensor model design and data processing of the ray information.

An important goal when implementing the system is to create modules that use the raw simulated data as input, making it easy to plug the modules into real systems without having to change how the module handles the input.

4.3.1 The X4 Radar Sensor

The implementation of the sensor design follows the sensor that is used by the NTNU research UAV, namely the XeThru X4MO5 radar sensor produced by Novelda. It is a UWB impulse radar chip, providing sub-mm movement sensing accuracy at distances from 0-25 meters. The main attributes of the sensor to consider are that the effective range is 10 meters where the sensing can detect human presence and an azimuth opening of 65°. A more detailed description can be found at [53] and [54], while [55] gives a thorough description of the antenna. A picture of the sensor can be seen in figure 4.3.

4.3.2 Sensor Design

An important factor to consider is that EM waves are continuous while rays are not if we consider the area they cover. Consequently, the resolution of the radar data is affected by the number of rays being traced in the simulator. Unfortunately, the number of rays is positively correlated with the computing demand. A formula describing the

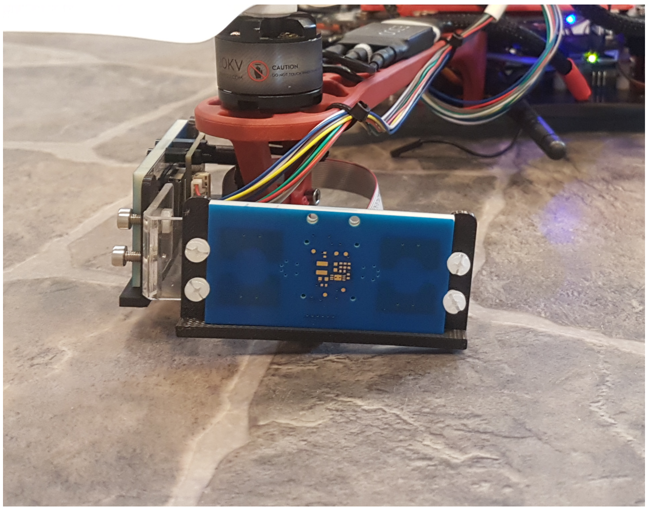


Figure 4.3: The XeThru X4M05 radar sensor mounted on a research drone from NTNU UAVlab

resolution calculations can be found in equation 4.3. Figure 4.4 for an illustration.

$$x_n = 2 * \tan(\frac{azi/2}{n-1}) * range \tag{4.3}$$

Here x_n is the resolution of the rays, azi is the angle opening of the sensor, $range$ is the range at which it operates, and n is the total number of rays being utilized in that plane.

To find an optimal resolution for the radar sensor model, we have to consider both the computational demand and the resolution wanted. At a range of 10 meters and with an azimuth opening of 65°, the resolution of the ray is calculated as:

$$n = \frac{azi/2}{\arctan(\frac{x_n}{2*range})} + 1 \tag{4.4}$$

Demanding a resolution of under 0.2 meters, we would get the following number of

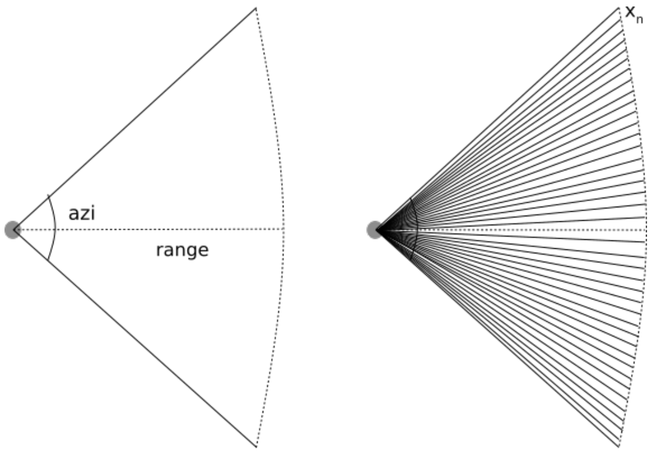


Figure 4.4: Illustration of ray resolution

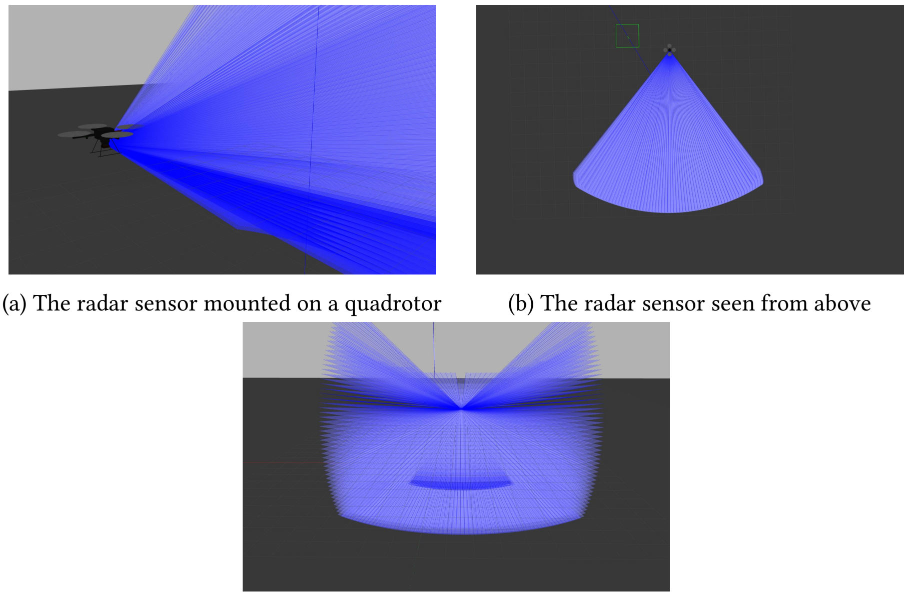
rays:

$$n = \frac{32.5}{\arctan(\frac{0.4}{2*10})} = 57.73 \tag{4.5}$$

Meaning that the radar sensor model needs $58 * 58 = 3364$ number of rays (for 3D coverage) to have a resolution of 0.2 m when used at a maximum range of 10 meters. The number of rays was tested in Gazebo but unfortunately crashed the simulator. A more modest number of 900 rays were chosen for the radar sensor, meaning a resolution of only 0.4 at 10 meters. Thus the sensor shouldn't be used for detection of smaller objects at the maximum range, as this does not accurately portray a real-world radar sensor. At a more typical range of 5 meters, the resolution is at a more acceptable 0.2 meters.

Given these specifications, a radar sensor model in Gazebo is created. It can be viewed in figure 4.5.

The shape of the radar beam is square and not circular as seen in figure 4.5b. Although a circular shape of the radar beam is a more accurate representation of the beam, the square shape was chosen because the handling in Gazebo is already integrated. This makes the scaling of the size of the beam and the number of rays



(a) The radar sensor mounted on a quadrotor

(b) The radar sensor seen from above

(c) The radar sensor seen from the front

Figure 4.5: Radar sensor model with rays visualized as blue lines

inside the beam simple, as the alternative with a cone would be not to register certain rays received from the LIDAR, demanding added work for every configuration of the sensor.

4.3.3 Data Processing

Data received from the Gazebo simulator has to be processed before it is published as radar data. The data obtained from the ray-tracing done in Gazebo is the range of each ray, the reflection coefficient of the object that was hit and the angle of the ray. The angle of the ray is removed from the data while angle and reflection calculations are performed as explained in section 4.2. White noise is then added to the signal. Finally, as the rays are of a certain amount and their range is highly concentrated around specific ranges, the graphs produced are jagged and does not have the same properties as continuous signals would have. An example of this phenomena can be seen in figure 4.7a, where no smoothing to the signal is done. A moving average is therefore used to smooth out the graph and create an illusion of a more continuous signal received. All the calculations are done as a plugin in Gazebo, which is updated following the frequency of the ray updates.

Gazebo - ROS for Ray-tracing

To solve the problem of dependencies between ROS and Gazebo, there is usually two version of each that has been tested and developed to work together with Long Term Support (LTS). The most recent to date is ROS Kinetic Kame with Gazebo version 7. A table of the different version combinations can be seen in Table 4.1. A problem with Gazebo 7 is that the ray-tracing done within the simulator can’t retrieve the objects that were hit for each ray. Consequently, it is not possible to calculate the angles between the rays and their respective objects. A newer version of Gazebo is therefore chosen, namely Gazebo Version 8.

ROS Lunar	Gazebo Version 7 (LTS)
ROS Kinetic (LTS)	Gazebo Version 7 (LTS)
ROS Jade	Gazebo Version 5
ROS Indigo (LTS)	Gazebo Version 2
ROS Hydro	Gazebo Version 1.9

Table 4.1: The combination of versions between ROS and Gazebo that are recommended for stability reasons

Multiple Sensors

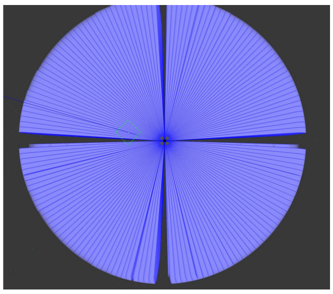
The research drone from NTNU UAVlab uses multiple radar sensor for complete coverage. To handle many simulated sensors in Gazebo, an unintuitive approach had to be used to circumvent a bug in the available software.

A radar sensor was placed at the bottom of the quadrotor with rays covering 360° in the horizontal plane and 65° in the vertical plane according to the sensor description. Then sections of the sensors ray output are used for each sensor, allowing up to four sensors to be mounted on the quadrotor. An illustration of this scenario is presented in figure 4.6.

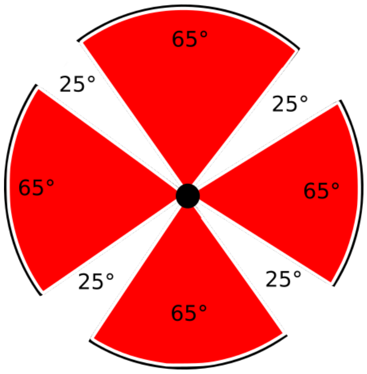
4.3.4 Complete Radar Sensor System

The implemented Gazebo-ROS radar model framework contains multiple ROS packages with each handling different aspects of the system. The packages will be presented, and its use cases explained.

- **`radar_data_visualization`:** Handles the data visualization, includes methods to visualize radar data amplitudes and SAR image generation. All nodes are written in python, using MatPlotLib [56] for visualization of data. All data visualization is happening in real time as data is outputted from the sensor.
- **`radar_quadrotor_description`:** Contains URDF files that describes the quadrotor model for which the radar sensor is mounted. Most of the models are based on the Hector Quadrotor package with slight modifications.



(a) A single radar sensor mounted on a quadrotor, with a complete coverage around the quadrotor and a 65° coverage in the vertical plane



(b) How each zone of the rays are mapped

Figure 4.6: The implementation done to allow for multiple radar sensor to appear using one actual sensor

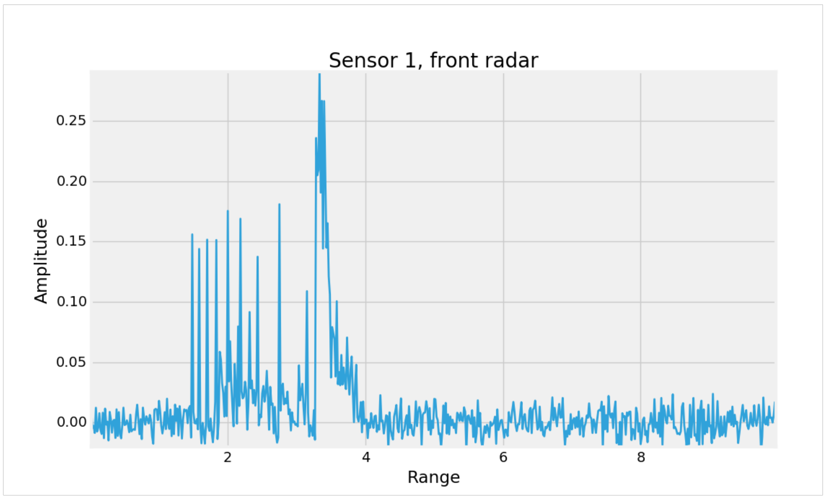
- **`radar_quadrotor_gazebo`:** Contains Gazebo specific files for launching ROS nodes and loading worlds for the simulator. Also contains all .world files that describe the simulation environments used.
- **`radar_sensor_description`:** Contains URDF files describing the sensor attributes. The files contain the description for sensor update rat, FOV, ray number, noise and more.
- **`radar_sensor_plugin`:** Gazebo plugin that handles the data processing of the rays to be presented as radar data. The code is written in C++, using callbacks monitored by the Gazebo system for handling ray updates.

4.4 Smoothing the Data Output

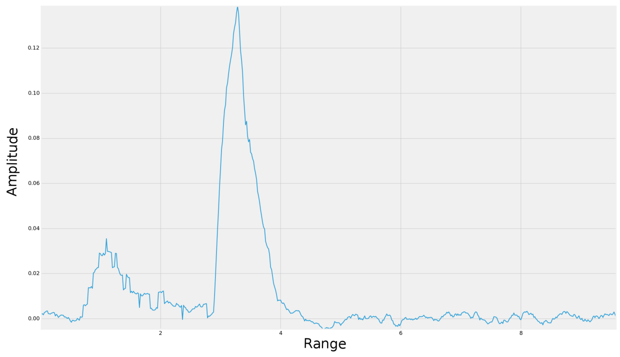
The data received from the sensor is significantly affected by the resolution of the rays. An illustration of the problem with the generated data set can be seen in figure 4.7a.

It is clear that the data generated by the sensor is returning amplitudes based on the spacing between the rays.

To reduce the discreteness of the data, a smoother is used, namely a moving average. The moving average is chosen to align the data from one object into a single peak, as would be expected if an infinite amount of rays were used. The result of using a moving average can be seen in figure 4.7b.



(a) Without the moving average implemented



(b) With the moving average implemented

Figure 4.7: A comparison between data that has been smoothed and not

Chapter 5

Mapping for Autonomous Vehicles using Radar Sensors

This chapter gives an introduction into mapping for autonomous vehicles, with a particular focus on mapping with radar sensors. Moreover, a presentation of the Occupancy Grid Mapping method is shown with implementation details.

Mapping of the environment is one of the most important aspects to consider when developing an autonomous drone platform. Development of autonomous drones involves many domains, such as path planning, control, modeling, and mapping. Both path planning and control heavily rely on an accurate understanding of the environment around the drone. Mapping is therefore of great importance.

5.1 Sensors for Mapping

Each sensor used in mapping provide different benefits and drawbacks with regards to mapping an environment. Most approaches today use stereo vision [57], 3D-LIDAR [58], radar or a fusion between the sensors. Stereo sensors extract 3D information from a digital picture while LIDAR uses pulsed laser light to measure the distance to objects. 3D-LIDARs can give out very high-resolution point clouds, which makes 3D mapping highly feasible as the distance and angle of each ray is known with high certainty. LIDARs are on the other hand not well suited for mapping in bad weather conditions [59] and in vegetation, as the laser rays will reflect when hitting leaves and smaller plants, which in reality is no obstacle for the vehicle in question [21]. Conversely, radar provide good results even in adverse weather conditions, thereby solving a big problem in both stereo vision and LIDAR sensor data [60], [61], [62], [63], [64]. Recent research that focuses on autonomous vehicles have taken a special interest in combining the best from LIDAR, stereo and radar sensors [65], [66], [67].

5.2 Mapping with Radar Sensor

The radar sensor has benefits and drawbacks when used for mapping an environment. The benefits are, as described above, the ability to work in adverse weather and low light conditions. The main drawback of the radar sensor is the uncertainty of its readings, as the data is affected by the angle of the reflecting object, multi-path, low resolution and more as mentioned in section 4.2.

5.2.1 Map Representations

There are many different mapping representations used for autonomous vehicles today. They can roughly be classified into two groups, namely topological and metric maps [68]. Topological maps use graph-like structure to represent places as nodes and paths as edges. Consequently, this representation can be used efficiently for path planning. On the other hand, metric maps store geometric information about the environment, enabling accurate features and fine-grained descriptions.

One of the most used geometric mapping techniques is the Occupancy Grid mode proposed by Elfes [69]. It represents the environment with probabilities of being occupied, free and not mapped, enabling probability calculations to be used when mapping an environment. Consequently, this representation is highly feasible when working with sensors that have a high degree of uncertainty, as radar sensors.

5.2.2 Occupancy Grid Mapping

To deal with the uncertainty of sensor data, a method to overlap multiple readings using probability analysis was proposed called Occupancy Grid Mapping [69]. Originally introduced to deal with the uncertainty related with sonar sensors tasked with mapping an environment, it is now widely used in the industry of autonomous mapping, as it allows for high integrity between multiple readings of sensor data.

The method is split into four parts:

1. **Preprocessing:** Rejects data from the sensor that is easily detected to be wrong, for example, a range reading that is too high or too low according to the specifications of the sensor.

2. **Occupancy probabilities:** Each cells probability of being empty or occupied is calculated based on two probability density functions, f_e and f_o, the probability of it being empty and it being occupied respectively. Details can be studied in [69].

3. **Occupancy map representation:** The map is a two-dimensional array of cells. Each cell has a value that represents the chance of it being occupied or empty, ranging from 0 to 100 (this range is what rviz operates with, in the paper, a range from -1 to 1 is used).

4. **Integrating information from multiple readings:** Every reading is combined with the pre-existing information of the map using a probabilistic addition formula.

An illustration of the effect of occupancy mapping can be seen in figure 5.1. As the sensor accumulates more information, the position of the object is more certain.

Memory Requirements

The main drawback of occupancy grids is the significant memory requirements. The number of cells is dependent on the map size. As a result, if a location on the map demands a high resolution, the rest of the map has to use the same grid size. Processing

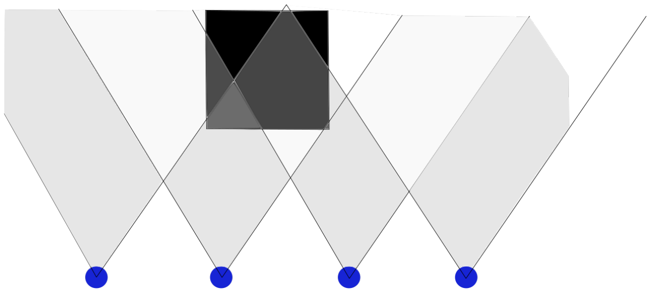


Figure 5.1: An illustration of the occupancy grid method. Overlapping readings reinforce the information about the environment. White represents empty and black represents occupied, blue circle represents the sensor

large-scale 3D environments in real-time are therefore practically impossible. Thus, 2D models are mainly used where Occupancy grids are concerned.

In the case of mapping an area of 50 square meters with a resolution of 0.05 meters per cell, with a cell representation size of 64 bits, the memory requirements are:

$$cells = (50/0.05)^2 = 1000000 \tag{5.1}$$

$$memorySize = \frac{50000 * 8}{1000000} = 8MB \tag{5.2}$$

Such big transmission sizes are not feasible in embedded system, it wouldn't even fit on the stack of most platforms. Some solutions to this problem is to reduce the bit size for each cell or decrease the resolution of the map.

The problem has been studied by the academic community, and many solutions have been suggested to reduce the memory requirements. [70] suggests using quadtrees, a type of data structure, to achieve a compact representation of large two-dimensional binary arrays. A featured based Occupancy Grid map is suggested by [71], where a sparse map can be created with implicit correlation between cells that represent an object.

Grid Traversal

Traversing the whole grid is computationally demanding. With the above example, each update of the algorithm would require to iterate through an array of 1000000 in size. In addition, an increase in the number of sensors producing their own data sets would increase the number of iterations even more.

5.3 Implementation of the Occupancy Grid Mapping in ROS

When implementing the Occupancy Grid Mapping for a quadrotor, it is important to consider hardware restrictions found on the platform. As an example, many of the UAVs at the NTNU uavlab are equipped with a Beaglebone Black due to its versatility and broad compatibility. Specifications can be found in Table 5.1. Embedded systems sets restrictions on the RAM usage and especially memory usage related to the method. Also, the CPU sets restrictions on the speed of processing available, as this is a sparse resource on a drone platform where multiple sensor data has to be processed, autopilot commands calculated and communication links held, among others.

CPU	AM335x 1GHz ARM Cortex-A8
RAM	512MB DDR3
Flash Storage	4GB 8-bit eMMC

Table 5.1: Specifications of the Beaglebone Black

5.3.1 Memory Usage

There is a need to reduce the memory size of the algorithm, as big mapping areas are relevant when considering the simulation of cargo holds. The ROS node was implemented in python and as such puts restrictions on the memory handling. With the python version used in ROS, python 2.7, an integer is represented with 24 bytes, demanding an extreme size if the resolution and scope of the map are big. To reduce this memory footprint, a python package called Numerical Python can be used (NumPy) [72]. The package allows for a smaller integer representation, similar to what can be expected for C programs. Using this package, a simple 8-bit representation of integer used would only demand a single byte per integer. If the calculations are done again the total memory requirements for one map is 1MB, an 87.5% reduction in memory size. A trade-off has to be made when choosing the integer representation, as an 8-bit integer does lose some of the precision compared to a 16-bit representation.

A decrease in resolution also dramatically affects the memory size used, but the resolution has to be determined from the application it is used for. As such, the resolution is set after initial testing with the algorithm has been performed.

5.3.2 Grid Traversal

To reduce the traversing computations required, a simple technique is utilized. A square is found around the emitted radar lobe, covering all the maximum and minimum grid values in the measurement. The traversing is then done inside this square instead of the whole grid. An illustration can be seen in figure 5.2 for how the square is found.

5.3.3 Object Detection

Object detection for use in mapping is done through a thresholding technique. The radar amplitude data received from the sensor is sent to the mapping node, where the data is processed for object detection. A threshold is set at a value that most accurately indicates that an object has been detected, done through testing and found to be 0.038. Multi-path and clutter can give false targets to a radar receiver. It is assumed in the simulator that none of these error sources exists, and as such, amplitudes received in the radar sensor above noise level will be considered to be an object.

The mapping node uses the first amplitude top that crosses the threshold as the range to the object. As a result, amplitude tops with a higher range is ignored, leading to a loss of information. The reason for not including this information is that the Occupancy Grid Mapping method is not able to handle the added data.

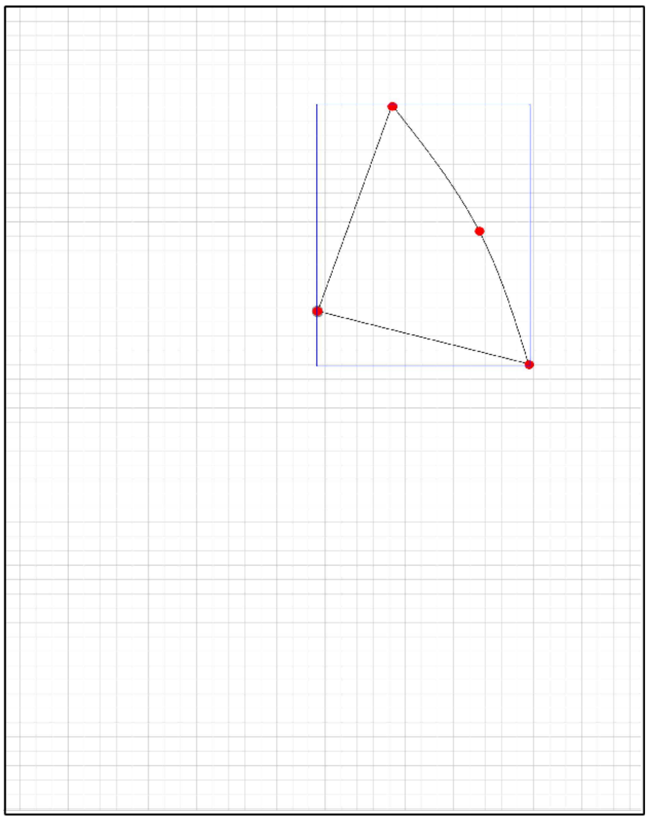


Figure 5.2: Square creation done for traversing in a smaller grid than the full map. Each red dot represents corners that are used for finding the maximum size of the rectangle

5.3.4 Occupancy Grid Mapping Algorithm

The main algorithm is implemented as described in the above section in a ROS python node. The inputs to the algorithm are the range to the object, grid to map, position and attitude of the sensor. The Occupancy Grid Mapping algorithm has been implemented into a ROS node for this Thesis called **`radar_mapping`**.

Chapter 6

Simulation Experiments and Results

In this chapter experiments showcasing the radar sensor and the simulator are presented. Each experiments purpose will be explained, along with methods used to run the experiment. Results will thereby follow, presented in a linear fashion. A discussion of the results will follow in the next chapter.

Information on how to replicate the experiments is given in Appendix B.

6.1 Radar Sensor Object Detection

This experiment will see how similar the radar data generated by the simulator is to the radar sensor X4 from Novelda.

Applications use the amplitude data generated by the sensor to determine the next action. Thus, if the generated data in the simulator is similar to a real sensor, applications tested in the simulator can be implemented into real systems with much of the calibrations already done.

A collaboration with Mr. Jablonski and his Thesis was done for this experiment, where the graphs produced from the real world are based on Mr. Jablonskis work and graphs produced from the simulator are based on the work from this Thesis.

SAR images are created from the amplitude data using the Simple Summation Algorithm, and the images produced from the simulator is then compared with the images produced from Mr. Jablonski's experiments on real world objects. The SAR images gives a different feedback about the amplitude generation because it sums up the amplitudes over a cross-range. For a detailed description of how the graphs were generated from the real world experiments, please see the Thesis produced by Mr. Jablonski [32].

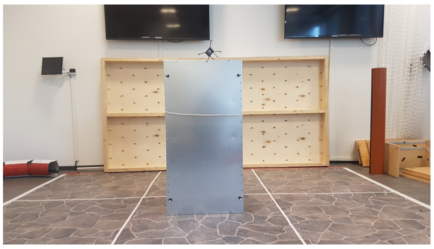
Setup

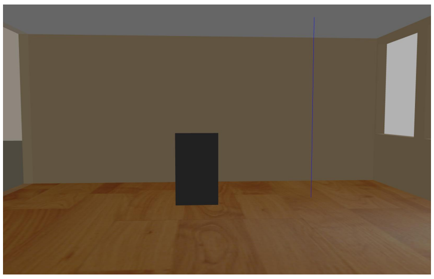
A testing room at NTNU was utilized. The room have, through the use of motion capture sensors, the ability to output high precision object location data. A similar setup is recreated within the simulator using the flight path of the quadrotor and known object positions for an accurate recreation of the scenario.

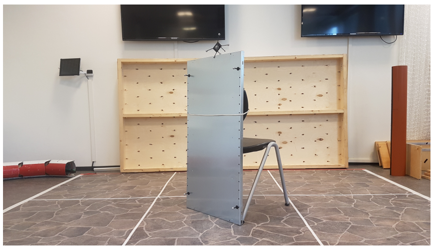
Different objects are placed at a center position in the room: A steel plate, a steel plate at 45°, a steel square pole and a steel wire. The same object are recreated in the simulator as closely as possible. The setups can be seen in figure 6.1.

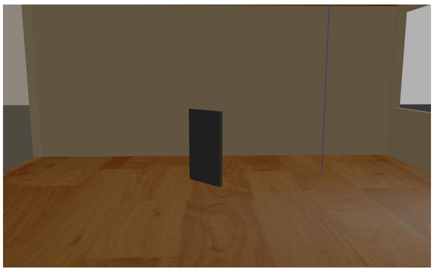
Method

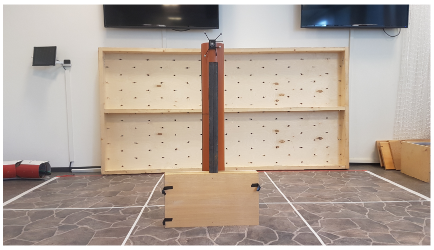
A quadrotor is flown inside the NTNU testing room with a handset controller, strafing the objects from side to side. The same movement is then recreated inside the simulator. A thing to note with this method is that the quadrotor is not very stable when controlled by a handset controller, and therefore have movement patterns that can be quite erratic. On the other hand, the simulated quadrotor moves with constant speed in a perfect straight line. The drone in action can be seen in figure 6.2.

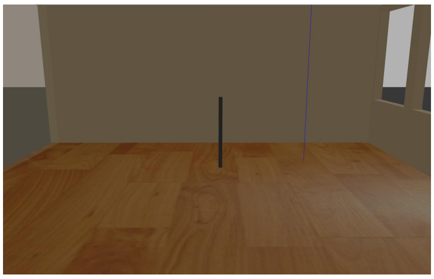

(a) Steel plate

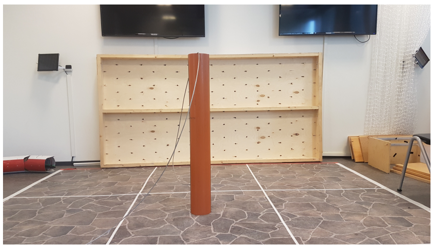

(b) Simulated steel plate


(c) Steel plate facing 45° to the side


(d) Simulated plate turned at 45°


(e) Square steel pole placed on wooden box, supported by plastic half-pipe


(f) Simulated square pole


(g) Steel wire placed against plastic half-pipe

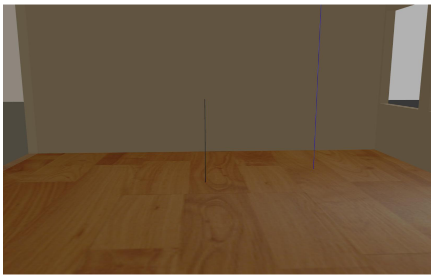

(h) Simulated wire

Figure 6.1: Different scenarios tested for the radar sensor

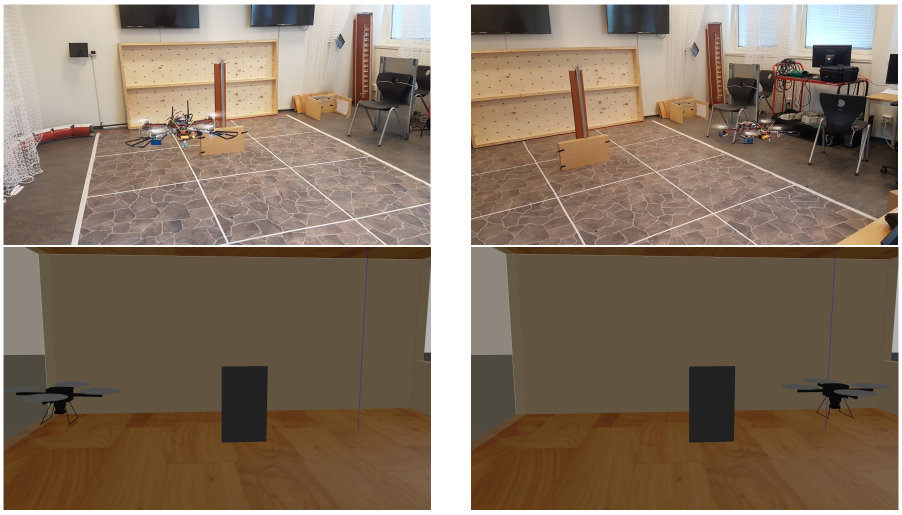


Figure 6.2: The drone strafing the object in the real world and simulated world

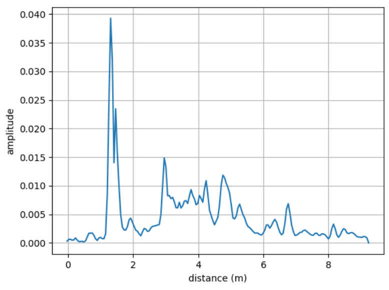
Results

The results accompanied by the different setups and objects can be seen in figures 6.3 - 6.9. Two types of graphs are presented from Mr. Jablonki's Thesis [32], namely snapshot of the amplitude shape of the graph and summation of all the amplitudes during the duration of the strafing. This is done to give a fuller picture of how the amplitude is outputted from the system. The simulator performance can be seen in table 6.1. It should be noted that these values are related to the computer used for the simulation. Specifications for the computer can be found in Appendix A.

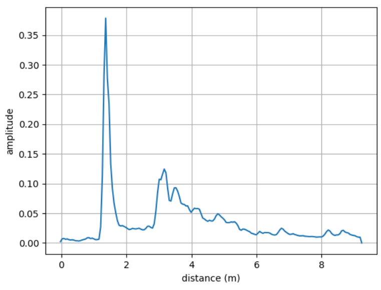
A video illustrating the simulator in real-time has been added as a digital appendix.

6.2 Mapping with Radar Sensors

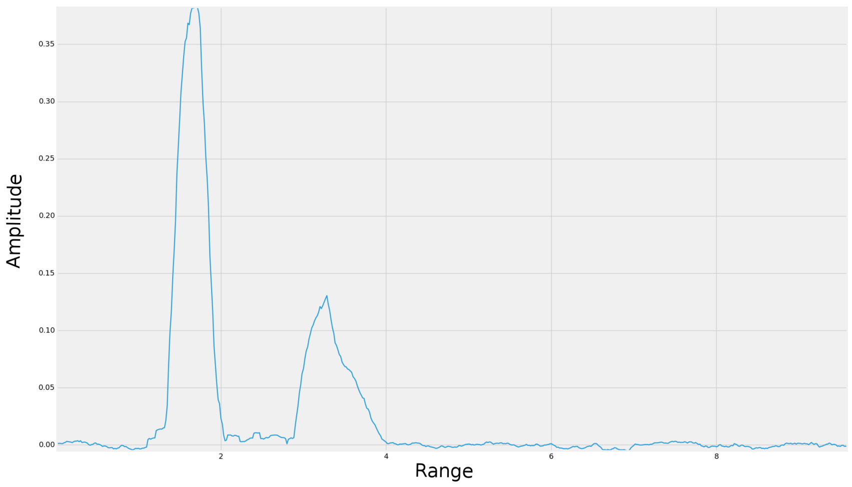
The following simulations are created to test the applicability of using the Occupancy Grid Mapping method in combination with radar sensors.



(a) Amplitude value snapshot during strafing of plate in NTNU testing room, from [32]

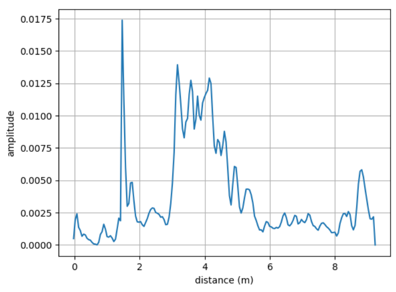


(b) Summation of amplitude values during strafing of plate in NTNU testing room, from [32]

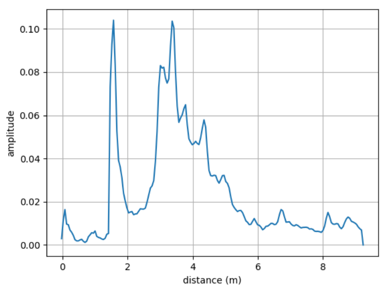


(c) Amplitude value snapshot during strafing of plate in simulator

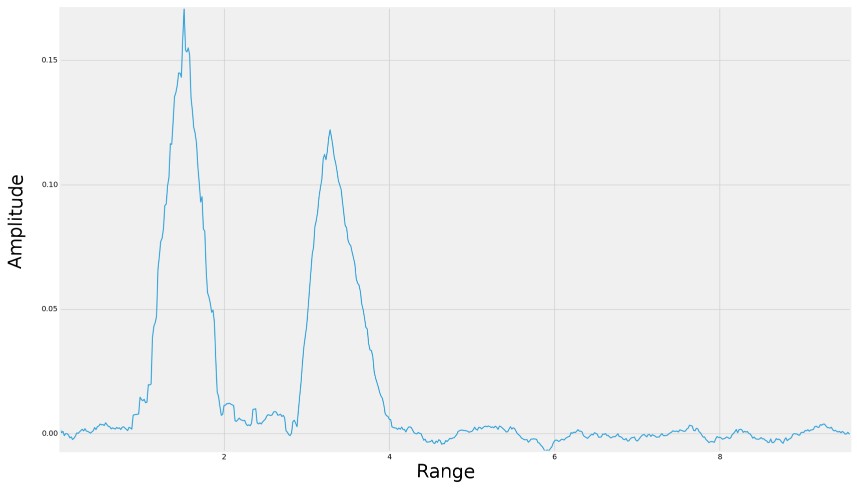
Figure 6.3: Amplitude values from the radar sensor during strafing of plate



(a) Amplitude value snapshot during strafing of plate at 45° in NTNU testing room, from [32]

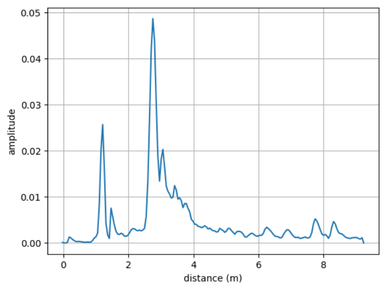


(b) Summation of amplitude values during strafing of plate at 45° in NTNU testing room, from [32]

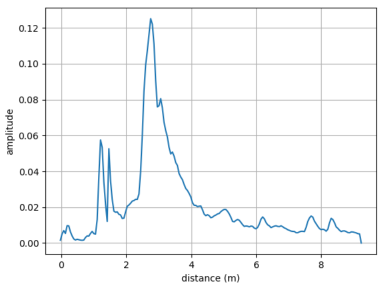


(c) Amplitude value snapshot during strafing of plate at 45° in simulator

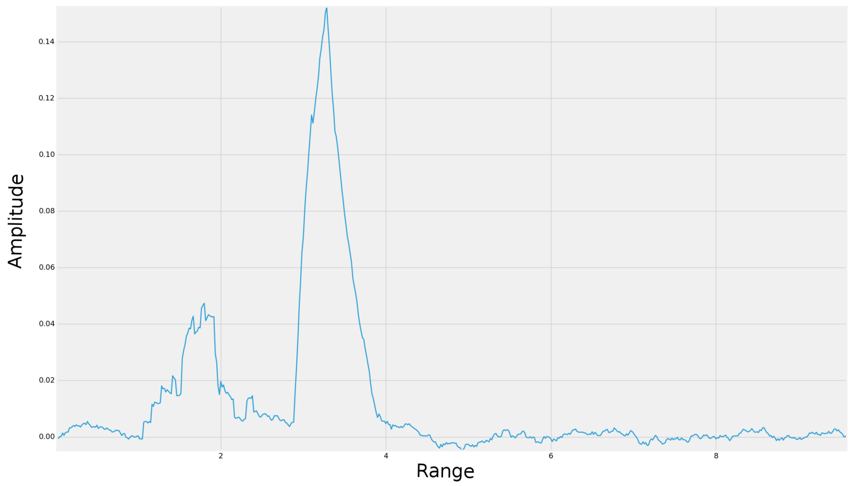
Figure 6.4: Amplitude values from the radar sensor during strafing of plate at 45°



(a) Amplitude value snapshot during strafing of square pole in NTNU testing room, from [32]

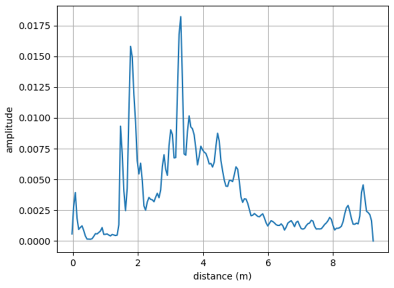


(b) Summation of amplitude values during strafing of square pole in NTNU testing room, from [32]

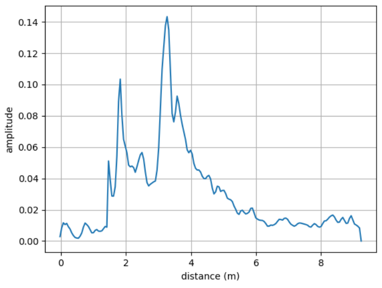


(c) Amplitude value snapshot during strafing of square pole in simulator

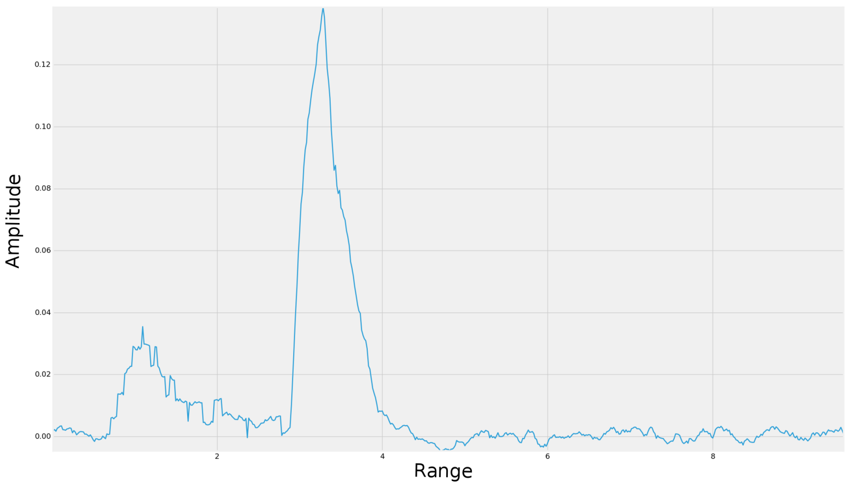
Figure 6.5: Amplitude values from the radar sensor during strafing of square pole



(a) Amplitude value snapshot during strafing of wire in NTNU testing room, from [32]

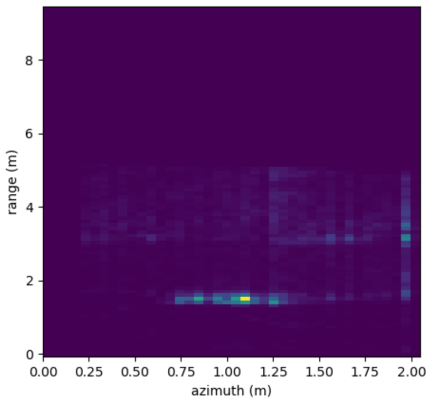


(b) Summation of amplitude values during strafing of wire in NTNU testing room, from [32]

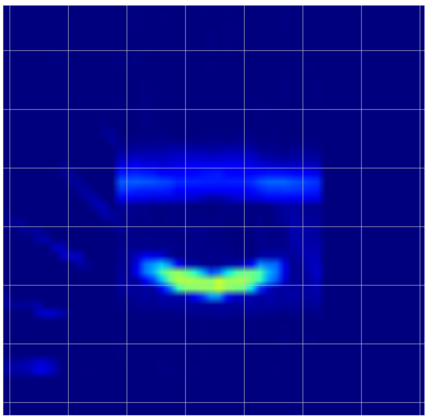


(c) Amplitude value snapshot during strafing of wire in simulator

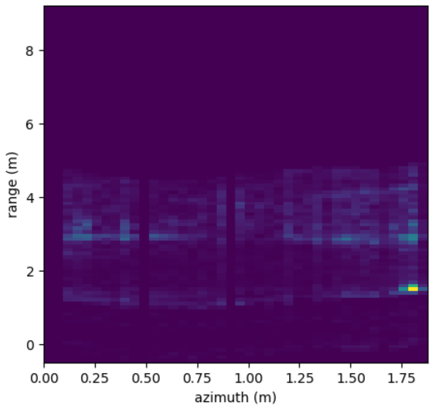
Figure 6.6: Amplitude values from the radar sensor during strafing of wire



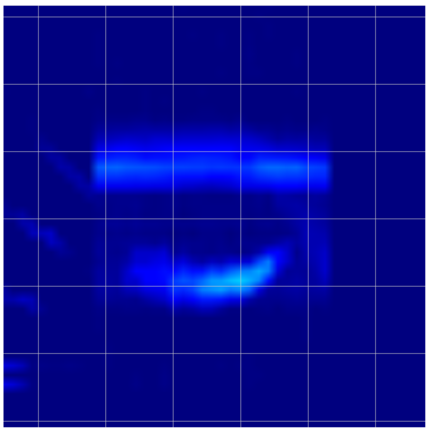
(a) SAR image from from real world run, plate, from [32]



(b) SAR image from simulator, plate

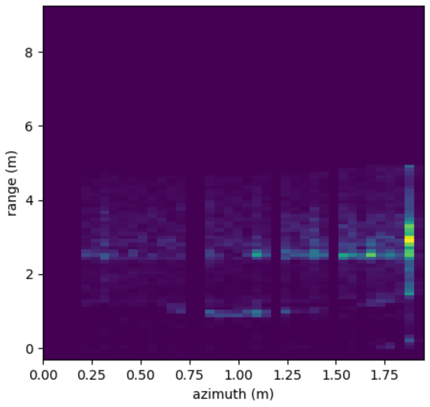


(c) SAR image from from real world run, plate at 45°, from [32]

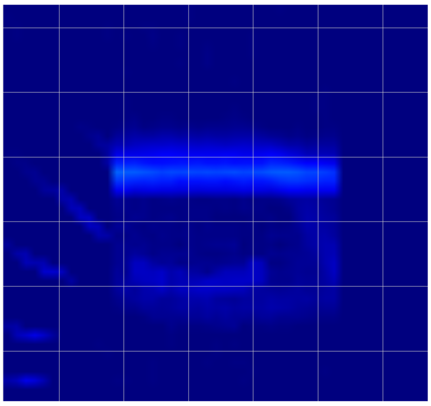


(d) SAR image from simulator, plate at 45°

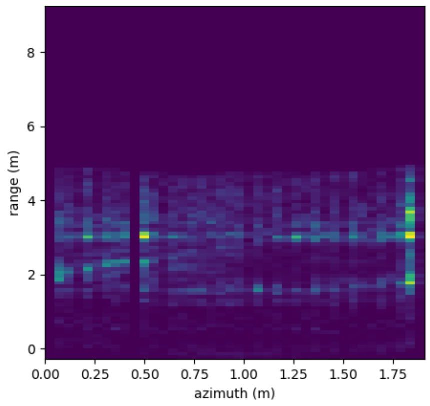
Figure 6.7: SAR images generated from strafing runs in the real world and trough the simulator



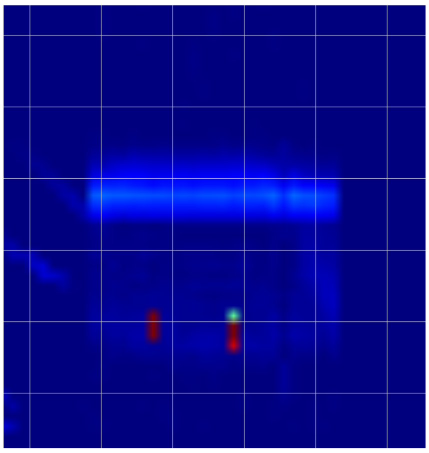
(a) SAR image from from real world run, square pole, from [32]



(b) SAR image from simulator, square pole



(c) SAR image from from real world run, wire, from [32]



(d) SAR image from simulator, wire

Figure 6.8: SAR images generated from strafing runs in the real world and trough the simulator

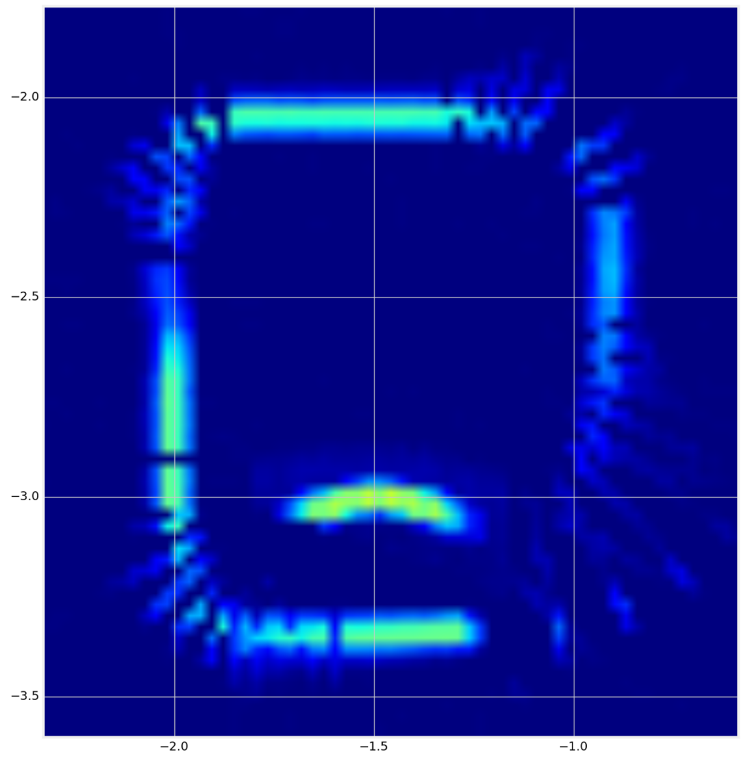


Figure 6.9: SAR image generated in simulator while traversing the NTNU room

Simulation Speed	1.0 Real-time speed
Gazebo CPU Usage	75%
Amplitude Visualizer Node CPU usage	25%
SAR Image Node CPU usage	35%

Table 6.1: Simulation Performance for the NTNU room amplitude generation and SAR image generation

Setup

Two distinct scenarios are created to test the capabilities of mapping with the radar sensors, the NTNU testing room is used and a bigger environment is created with a wide range of obstacles. The objects in the cluttered environment are a car, a postbox, a stop sign, a human, a robot, a control panel and a dumpster. The simulated NTNU room can be seen in figure 6.10 and the cluttered environment can be seen in figure 6.11.

Method

Different reflection coefficients are given the objects, but mostly around the same reflection coefficient used for steel objects. The quadrotor is flown in a given path around the room in a slow manner, giving it time to capture relevant features with the radar sensor.

For these simulations, the quadrotor is mounted with four radar sensors, covering 260° around the quadrotor. An illustration of the sensors can be seen in figure 4.6. Thus making the quadrotor setup capable of detecting at higher rates and covering all main angles around its location. The paths can be seen in 6.12.

Four different resolutions of the grid used by the Occupancy Mapping is chosen to test how the algorithm handles different memory restrictions.

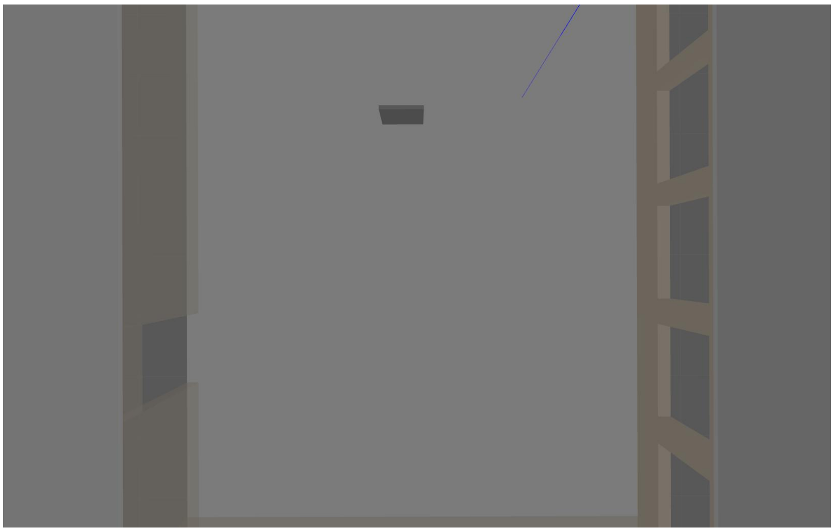


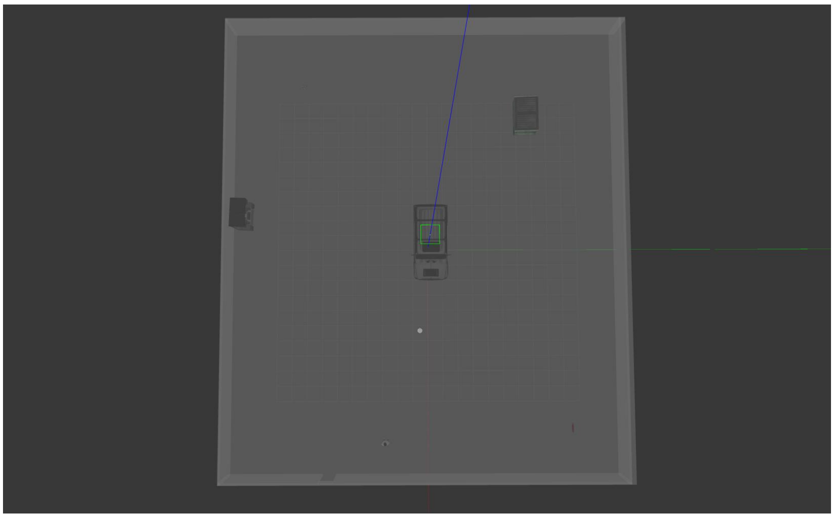
Figure 6.10: Overview of the simulated testing room at NTNU

Results

The results can be seen in figure 6.13 and 6.14. Simulation performance values can be seen in table 6.2 for NTNU room and table 6.3 for cluttered room. Should be noted that these values are related to the system used during simulation.

Simulation Speed	1.0 Real-time speed
Gazebo CPU Usage	80%
Mapping Node CPU usage	15%
Mapping Node Memory usage	0.43%

Table 6.2: Simulation Performance for the NTNU room mapping


(a) Overview of the simulated clustered environment, from the top

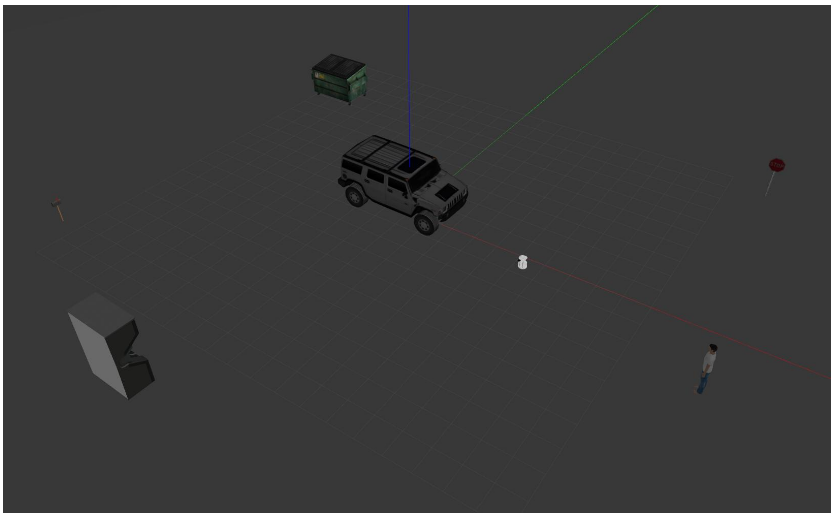
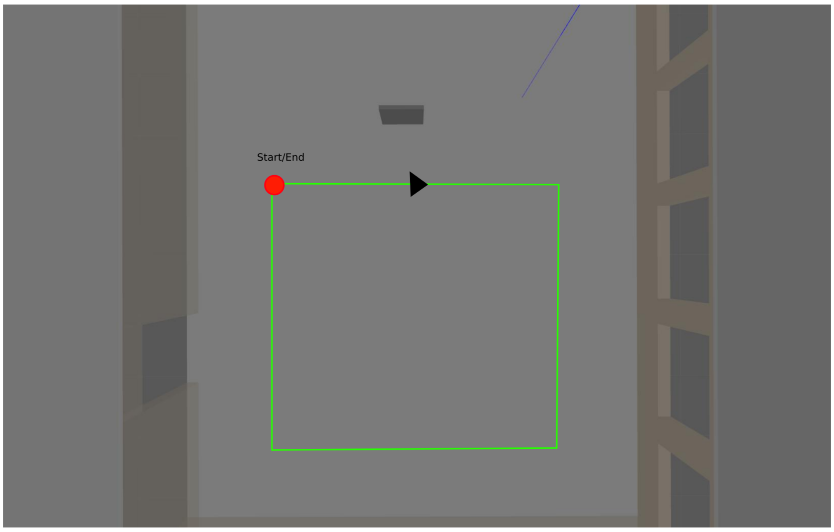
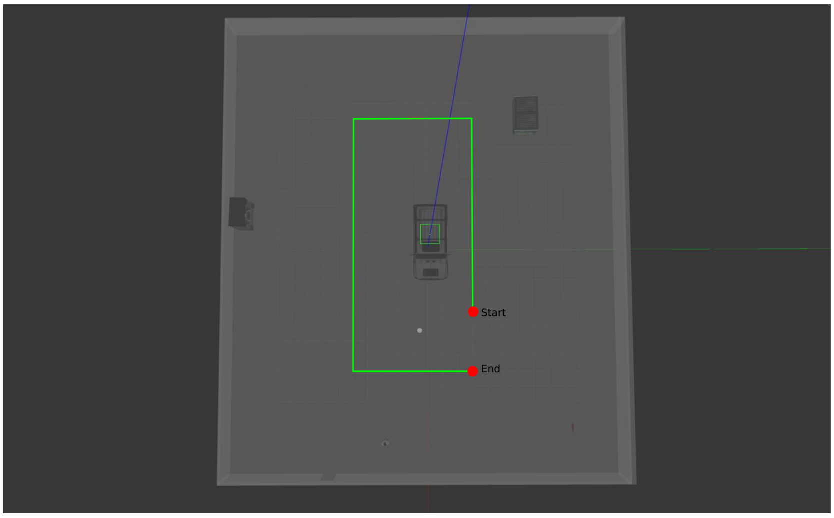

(b) Overview of the simulated clustered environment, wall and roof removed for a clearer view

Figure 6.11: Overview of the simulated clustered environment

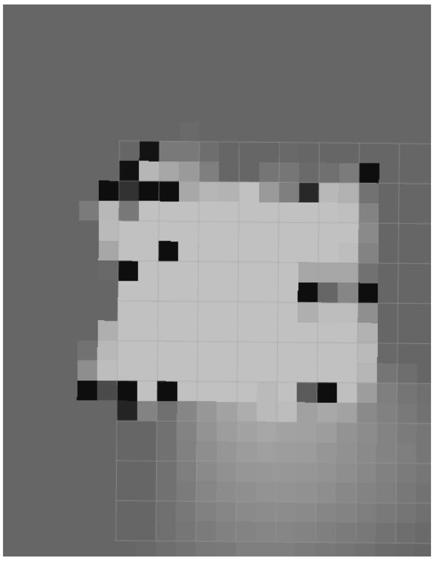


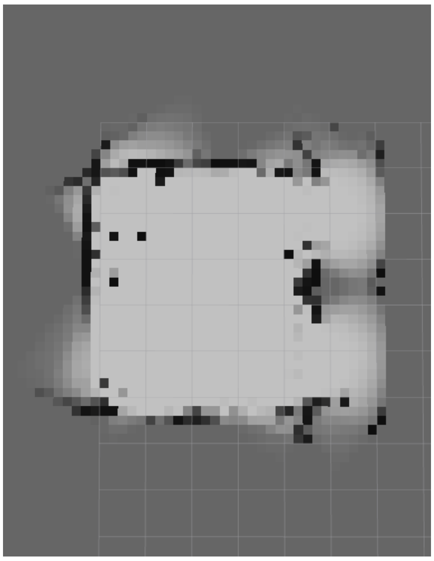
(a) Overview of the path taken by the quadrotor in the NTNU testing room

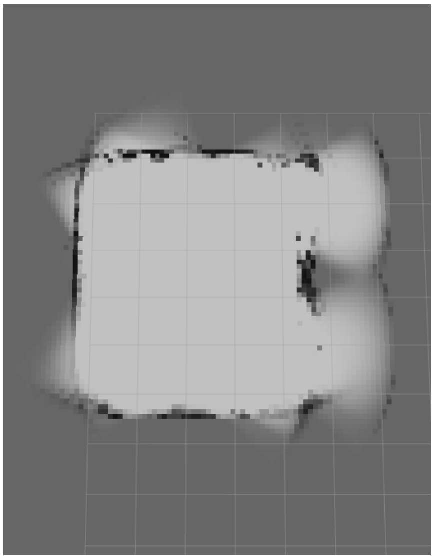


(b) Overview of the path taken by the quadrotor in the clustered environment

Figure 6.12: Overview of the different environments simulated


(a) Mapping of NTNU room with 0.5 meter resolution


(b) Mapping of NTNU room with 0.2 meter resolution


(c) Mapping of NTNU room with 0.1 meter resolution

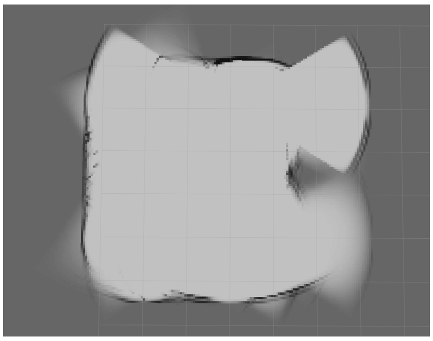
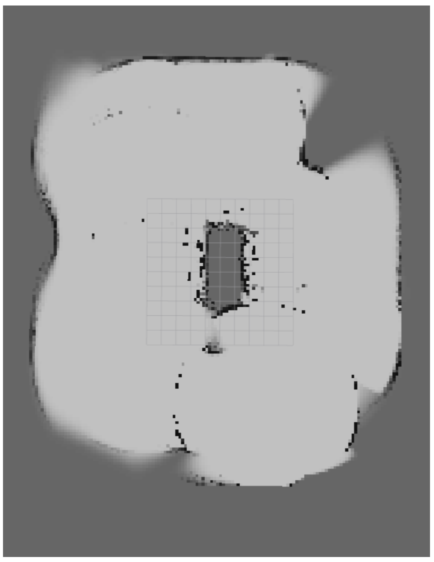

(d) Mapping of NTNU room with 0.05 meter resolution

Figure 6.13: Mapping of the NTNU room with different resolutions on the Occupancy Grid


(a) Simulation run number one of the cluttered environment

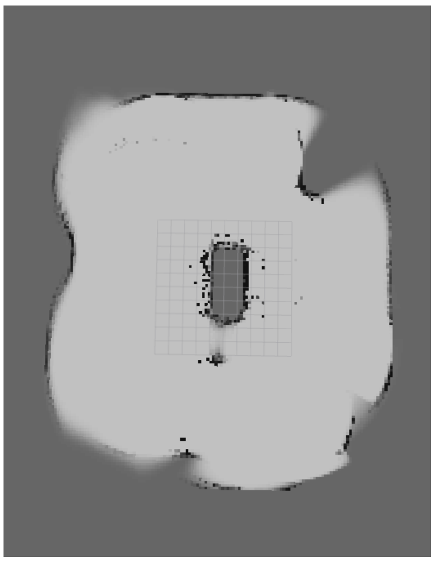

(b) Simulation run number two of the cluttered environment

Figure 6.14: Mapping of the cluttered environment, two different simulations with the same parameters set. 0.1 meter resolution on the Occupancy Grid is used.

Simulation Speed	0.5 Real-time speed
Gazebo CPU Usage	60%
Mapping Node CPU usage	55%
Mapping Node Memory usage	0.43%

Table 6.3: Simulation Performance for the cluttered environment mapping

6.3 Cargo Hold

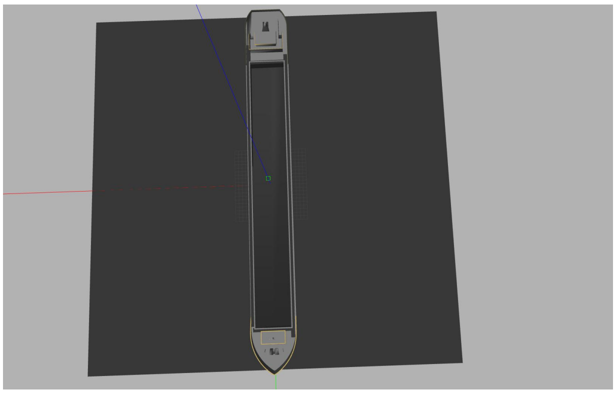
It is important to test how the sensor behaves in environments similar to its use-case. A cargo hold is recreated inside the simulator to test the mapping capabilities of the radar sensor.

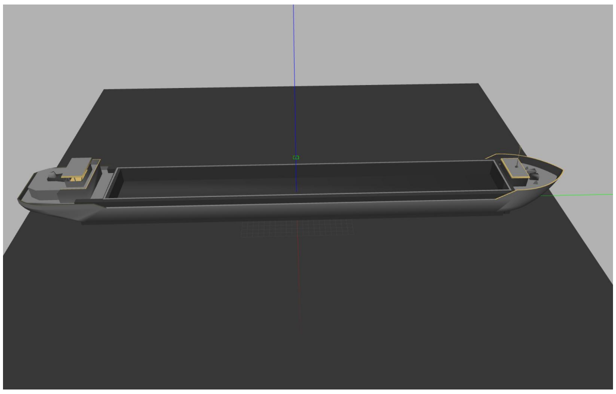
Setup

A cargo hold 3D sketch is taken from [73], containing a detailed mesh with dimensions matching that of the real ship. The mesh is inputed into the simulator and walls are created following the mesh of the ship. An overview of the ship can be seen in figure 6.15.

Method

The mesh does not contain any collision detection and as such the radar sensor can not be used directly against the mesh. To make the radar sensor able to detect the walls of the cargo hold, new walls are recreated inside the mesh that follows the shape and form of the ship, in essence copying the ships main structures. A view of the simplified cargo hold can be seen in figure 6.16. The quadrotor is flown in a pre-determined path along the walls of the cargo hold, as can be seen in figure 6.17, and is in this simulation also mounted with four radar sensors covering 260° around the quadrotor.


(a) Cargo hold seen from above


(b) Cargo hold seen from the side

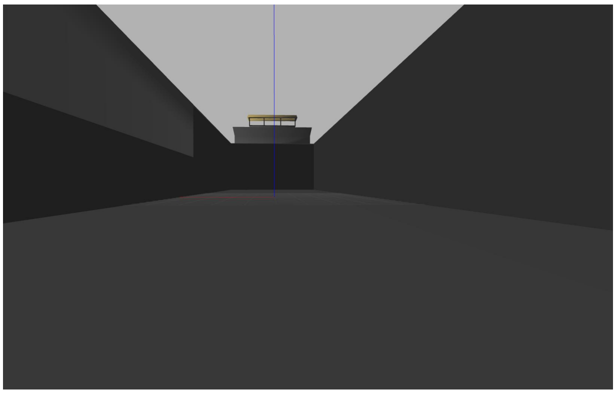

(c) Cargo hold seen from the inside

Figure 6.15: An overview of the cargo hold used for testing the radar sensor mounted on a quadrotor

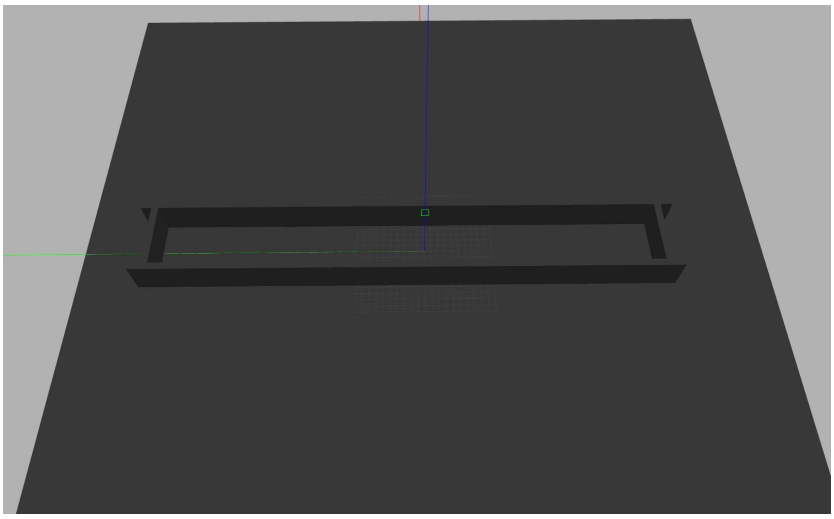


Figure 6.16: The cargo hold simplified

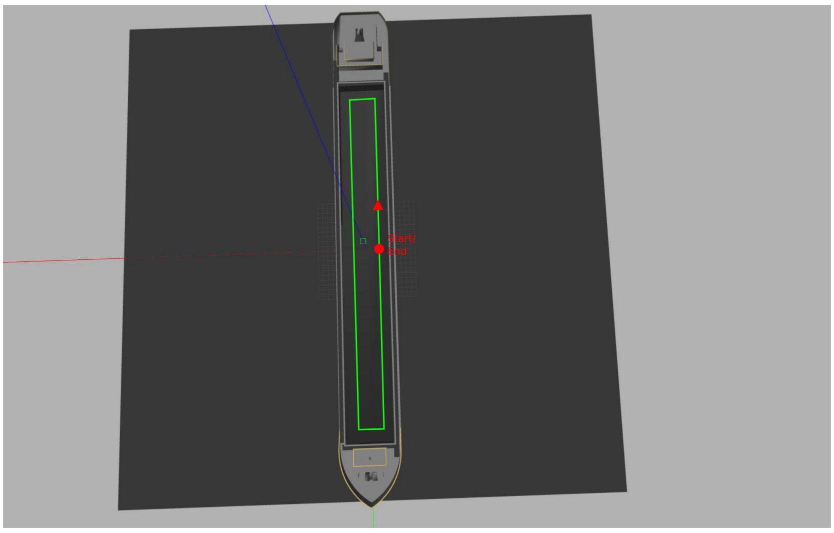


Figure 6.17: Path taken by the quadrotor when traversing the cargo hold

Results

The results from the simulation can be seen in figure 6.18. Simulation performance values can be seen in table 6.4. Should be noted that these values are related to the computer system used during simulation.

Simulation Speed	0.4 Real-time speed
Gazebo CPU Usage	38%
Mapping Node CPU usage	128%
Mapping Node Memory usage	0.43%

Table 6.4: Simulation Performance for the Cargo Hold experiment. Because the system uses eight CPU cores, the CPU usage can exceed 100%

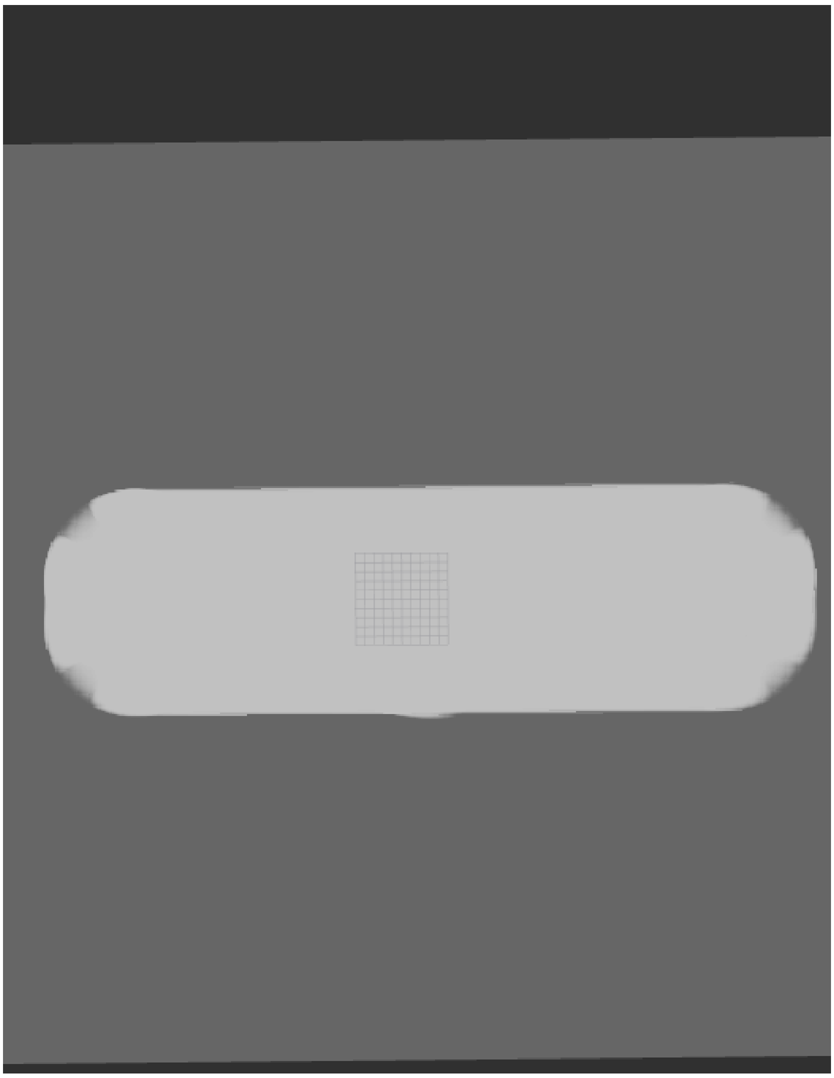


Figure 6.18: Map generated while traversing the cargo hold, 0.1 m resolution

Chapter 7

Discussion

This chapter discusses the results received from the simulation experiments. Each experiment is discussed modularly and a summary is given for each of the main experiments.

7.1 Radar Sensor Object Detection

Amplitude Graphs

The amplitudes generated from the simulator have similar characteristics to the amplitudes generated by the X4 radar sensor as seen in figures 6.3 - 6.6. Amplitude maximums are generated at the same ranges and the shape of the graphs are of similar size.

While the amplitudes are of similar size, their shapes do differ within the different scenarios. The simulated amplitude graphs are thicker and do not contain as many local maxima as the real radar sensor amplitudes. This indicates that the real world generates echoes and disturbances that are not captured by the simple white noise error used in the simulation. Moreover, the graphs amplitude is dragged behind the position of the wall, indicating that the real world receiver is receiving echoes beyond the wall or from multi-path propagation. These phenomena are not included in the simulator, and is therefore not apparent in the simulated radar amplitudes.

A particular case can be seen in figure 6.6, where the radar sensor strafes a wire. It is of interest that the real world radar is receiving a strong signal from the location of the wire, even though the wire itself is tiny. This might originate from the backscattering of the floor. The simulator is also generating significant amplitudes at the same range, originating from the backscattering of the floor. Thus indicating that they are the same phenomena. It would be beneficial to remove this clutter information, as it only masks other objects signature. An interesting method is to try to use the height information of the quadrotor and known opening and location of the sensor to reduce the echo, assuming a flat floor.

SAR Images

The SAR images generated in figures 6.7 - 6.9 present most of the same features. The plate seen in figure 6.7 is the clearest object in both the simulated and real-world experiments, as is expected of an object with a very high reflection coefficient. Nonetheless, it is of interest that no hyperbolas were created in the real world images, especially for the plate facing the sensor. This might stem from the fact that the speed held in the simulator is slower than in the real world experiment, giving the simulated radar sensor more time to create the hyperbola.

The angle calculations don't seem to capture the reduced reflection in the tilted plate simulation fully. The SAR image generated from the real world has a very weak image of the plate, only indicating an object at 1.75-meter azimuth. On the contrary, the tilted plate in the simulator is shown with high precision. Angle calculations done between the rays and objects should be changed to reflect this better. Future tests should be done to create a correlation between the angle of the plate and reduction in reflection.

While most of the images capture the object to some degree, the most significant difference is for the wire. The image generated from the real world radar sensor does, as expected, not produce any clear picture of the object. On the other hand, the simulator is detecting the wire at some points while strafing it. The main problem of the wire simulation stems from the fact that the wire is a tiny object with a high reflection coefficient. Because the rays are limited, a few rays hitting the thin wire gives the same output as if they hit a steel plate with the size as the distance between rays, in essence, the resolution of the rays. Accordingly, if one ray hits the wire at a range of 5 meters, where the resolution is 0.2 meters, the object detected is a steel plate of 0.2m^2.

There is a resolution difference in the SAR images generated. This mostly stems from the fact that the simulator allowed for more radar data samples than was done in the real world, where the quadrotor held a relatively high speed when strafing the object.

Summary

The amplitude generation in the simulator creates accurate amplitudes when presented with objects that are of size bigger than the resolution of the rays and have a high reflection coefficient. Some weaknesses with the implemented radar sensor are found, namely that it does not support multi-path propagation and propagation through objects inside the simulated world. Some tuning is also found to be necessary for the angle calculations. The simulator performed well for all scenarios, allowing for real-time data generation.

7.2 Mapping with Radar Sensors

NTNU Room Mapping

The NTNU room mapping results seen in figure 6.13 show that the main features of the room are accurately mapped with the Occupancy Grid Method. Namely, the plate and the placement of the walls are correctly portrayed in the map, including the fact that the space behind the plate might not be occupied even though it never flew behind it. Moreover, the method is able to synchronize data from four different radar sensors to create a map of the environment, showcasing the capabilities of the method to combine information from multiple readings.

Admittedly, the method does seem to map wrong in specific scenarios, namely the space behind the walls and no indication of where the windows and doors are located. The mapped space behind the walls originates from the way the Occupancy Grid Method works. During a turn, the radar sensor detects an object at the outer corner of the lobe. Using previous information, it assumes that the object is mainly located here and that it is empty to the side of the object. As a result, space behind the walls is mapped. The doors and windows are hard to detect as it uses information received only from radar sensors, which does not have a high enough resolution for detecting the smaller spaces that is the windows and the door.

The Occupancy Grid Method has a big memory demand, and it is therefore of interest to see which resolution is needed for use with the radar sensors. Resolution as high as 0.5 meters produced an unclear map with most of the features of the room removed. Mapping with the 0.2-meter resolution does provide a map where most of the features contained in the room are present, the plate has been detected and the walls are mapped correctly. It does, however, produce some occupied grids that are not featured in the environment. Resolution of 0.1 and 0.05 meters both produce a map of the room where the main features are mapped and with few falsely occupied grids. The memory for each of the resolutions are, when assuming 64 bit per grid in a map covering 50 square meters; 80kB for 0.5 meters, 500kB for 0.2 meters, 2MB for 0.1 meters and 8MB for 0.05 meters. As can be seen, the resolution dramatically affects the memory space required.

It is interesting to compare the map generated by the Occupancy Grid Method, and that generated by the SAR method 6.9. Comparing the two results, it is clear that the Occupancy Grid Mapping method produces a more detailed map of the room, containing a better representation of the plate and walls. However, the SAR image does give an alternative representation of the map that could be of use to reduce the uncertainty in the occupancy grid. The location of the plate and some parts of the wall is given with very high certainty in the SAR image, while the Occupancy Grid Mapping method produces some false occupied grids around these objects, these false occupied grids could be removed through an extra layer of calculations with the data from the two representations.

Cluttered Room Mapping

The radar sensors combined with the Occupancy Grid Method was able to capture most of the main features of the cluttered room, as can be seen in figure 6.14. Even though the radar sensors and the Occupancy Grid Mapping had to work with a longer distance to the objects in this scenario, most of the features in the room was still mapped accurately.

The most apparent miss mapping happens with the postbox and the stop sign, placed in the top left and bottom right of the room respectively. The Occupancy Grid Method is either smearing its values across the area where they are located, as with the stop sign, or having multiple possible locations of the object as with the postbox. This error might come from the fact that the Occupancy Grid Method is getting highly mixed signals from these two objects with the radar sensor, as the resolution of the radar sensor is very low at these ranges. Because of the big spacing between the rays at these ranges and the relatively small size of the object, the sensor will then switch between locating an object and not, in succession. This causes the probability calculations done in the Occupancy Grid Mapping method to be highly random, as it is never able to know for sure which area is empty and which is occupied. An increase in resolution would be able to fix this problem.

Another problem of the Occupancy Grid Mapping method appears when the quadrotor is standing still but objects are still detected. In some scenarios where the previous information is not very confident, the method might start mapping grids along the lobe to be occupied. This can be seen in figure 6.14, here the quadrotor waited for a little while at the end position, starting to map occupied grids to its right and left side. On the left side of the quadrotor, the lobe is hitting the human obstacle and starts mapping grids along the lobe as occupied. This problem might stem from the way the Occupancy Grid Mapping method calculates and combines the probabilities. At one stage, the method normalizes the probabilities of being occupied along the lobe of the radar beam. Because there is noise in the readings which were propagated to the map, the method could start normalizing the values along the lobe to increase the probability of being occupied.

The simulator speed was throttled to half speed in this simulation. Main reason was that the mapping node demanded a slower simulation speed to allow it time to complete all calculations between each sensor update. Comparing the two tables of 6.2 and 6.3, it is clear that the mapping node is having computation problems in the cluttered environment. The main difference between these two scenarios is that objects are getting detected at longer ranges in the cluttered environment compared to the NTNU room. Consequently the computation power is more demanding in environments where the range between the sensors and the objects are significant.

Summary

The Occupancy Grid Mapping method is able to efficiently use information from multiple sensors to generate a map that uses previous information to make assumptions about the environment. Even so, the method is highly reliant on the accuracy of the sensors used for mapping, as a high uncertainty in the sensors causes objects to be smeared across the occupancy grid or create falsely occupied grids. The method also has problems when many measurements are taken while the sensor is static, as the readings cause false occupied grids to be mapped. Moreover, the method is also dependent on the resolution used for the grids, as a too low resolution causes the mapping to be inaccurate. However, the resolution greatly affects the memory usage of the algorithm, creating a trade-off between accuracy and memory space required.

7.3 Cargo Hold

The cargo hold test shows how easy it is to implement a 3D model into the Gazebo world, and then use simple methods to make it interact with the radar sensor. In this case, a rectangle of walls was built following the mesh of the model. In a more relevant scenario, a mesh detailing the inside of the cargo hold with wires and potential obstacles would be used.

The problem of the mapping algorithm is made very clear in this experiment. Almost nothing is detected by the sensors, causing the algorithm to traverse a maximum amount of grids for every sensor update. The CPU demand is increased and the simulation time has to be throttled to allow for the mapping node to complete the calculations.

Chapter 8

Conclusion and Future Work

Based on the similarity of the real world sensor data and the simulated sensor output, it can be concluded that the physical interpretations done apply to the radar sensor. This includes the use of the ray-tracing method for EM waves, angle reflection, and noise. Nonetheless, the data generated does contain some errors, main contributions being the low resolution of the rays and not being able to simulate all the attributes that EM waves have, mainly multi-path propagation and penetration capabilities through objects.

The Occupancy Grid Mapping method has been proven to be an efficient mapping method when used in combination with radar sensors. Namely, the ability to incorporate error and uncertainty into the readings and using a probabilistic approach to combine measurements, significantly improving the mapping capabilities of the radar. Admittedly, the method does have some problems, namely a significant memory demand and a varying computation demand dependent on the object detection range.

As can be seen from the vast array of different simulation scenarios developed and tests done, the simulator framework fulfills its task to aid the user in generating relevant tests for their applications. With Gazebos powerful 3D physics simulator, the robust and diverse system for robot application development given by ROS and the alternative platform for embedded software development found in DUNE, the framework provides the user with multiple tools to work with.

The high portability of the code developed in this project shows that there is much to be gained when developing with the simulator framework. ROS gives the user the high modularization needed, while Gazebo provides data that is relevant to the real world. Together they allow for developed code to immediately be implemented into real-world systems able to run ROS.

8.1 Future Work

There are many possible steps to take for improving the radar sensor. One of the more significant errors that are not captured by the sensor is the multi-path propagation experienced by radar receivers in enclosed environments. A possible solution is to start tracing the rays for multiple bounces, generating a backscatter effect for every object that is hit. Allowing for penetration of object with the simulated rays should also be considered for objects with low reflection coefficients. More tests should be done with a real sensor in enclosed environments similar to the use cases of the sensor. This would allow for better tuning of the sensor and detection of errors that have not been detected in the experiments done at the NTNU testing room.

The code of the project also needs to be updated for future use. Angle calculations done for objects should be made more independent, enabling it to detect the angle between the object and the ray without running specific tests for each object. Further, the base code should be updated with the ROS and Gazebo to the next available stable version of the system. This might allow for fewer bugs when running the framework, perhaps solving the problem of adding multiple radar sensors into the simulator. Code that needs to have a high performance should be changed to C++, of particular note is the Occupancy Grid Mapping node written in python.

There are mainly three ways to improve the Occupancy Grid Mapping method, reducing the memory requirements, increasing the accuracy of the mapping and reducing the computation demand. [70] shows an efficient way of reducing the memory requirements by the use of quadtrees (data structure type). This method would be of interest to test, especially if the mapping is used for big cargo holds and other large sized environments. An interesting task, to increase the accuracy of the mapping, would be to look into if information generated by SAR images can be incorporated into the Occupancy Grids. The varying demand in computation proved to be a challenge when used in large sized environments, a way to reduce the traversal size for long-range object detection should be studied.

Appendix A

Simulation Framework Setup

This Appendix handles the problem of setting up the simulation framework with all the necessary files.

A.1 Setting up the System

The platform for setup should contain the following:

- Ubuntu Trusty or later installed
- Dedicated GPU (for Gazebo)
- A CPU that is at least an Intel I5, or equivalent
- At least 500MB of free disk space

The system specification of the computer used for this Thesis:

- Ubuntu Xenial 16.04
- Radeon HD 8570
- Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, octa core

A.2 Downloading the necessary softwares

To download the ROS Kinetic and Gazebo version 8, follow the following guides:

- **http://www.ros.org/install/** for installing the correct version of ROS
- **`http://gazebosim.org/tutorials?tut=ros_wrapper_versions`** for installing the correct version of Gazebo. Be aware that for Gazebo to work with ROS, something called the `gazebo_ros_pkgs` has to be installed. Installing Gazebo version 8 separately might cause problems related to this package.

Test throughly if both of the programs are working properly. Check if they are able to launch with the command **roslaunch `gazebo_ros` `empty_world.launch`**. For problems, look into the ROS and Gazebo homepage.

A.3 Downloading Companion Files

The Thesis files are available at the gitrepo hosted with UAVlab. A permission is needed from the team behind UAVlab to download the files. The name of the repository is `gazebo_ros_dune_simulator_framework` with URL:

- `http://uavlab.itk.ntnu.no:88/davidcb/gazebo_ros_dune_simulator_framework`.

Use the script located in the /scripts folder from the Thesis code called `script_hector.sh` to download the necessary files for Hector Quadrotor.

A.4 Compiling the Source Code

Every ROS package should now be placed in the /src folder in the Thesis package. Going to the catkin workspace folder called `/catkin_ws` and typing `catkin_make` should start compilation of the code. If it fails, try compiling once more, as there might be some dependencies that were compiled in the wrong order.

Appendix B

Running the Thesis Simulations

Only continue if system testing is complete. Of special interest is to check if the right Gazebo and ROS versions are installed. Further, running the **roslaunch `gazebo_ros` `empty_world.launch`** should generate an empty simulation environment inside Gazebo.

B.1 Running the Simulations

Before running the simulations, all environmental variables have to be set to the correct working space (ROS workspace has to be set to the downloaded Thesis `catkin_ws folder`). Further, the `GAZEBO_RESOURCE_PATH` has to include the path to the `catkin_ws/src/radar_quadrotor_gazebo` folder to let the launch files see the .world files needed.

To allow for the same speed on the quadrotor used in the simulations, change the `max_xy` to 0.5 in the controller.yaml file located in:

- `/catkin_ws/src/hector_quadrotor/hector_quadrotor_controllers/params`

All simulation which can be seen in the results chapter can be rerun through scripts that are placed in the /scripts folder. For example, to run the simulation for amplitude generation for the square pole, run: `./ntnu_room_strafing_square_pole.sh`. Press Ctrl+C in the terminal to make it strafe the object. All mapping simulations are done automatically and no interfacing is needed.

If the platform that the simulations are running on are slower than the given specifications above, it might be needed to reduce the simulation speed to allow for accurate mapping. To reduce the simulation speed, access the .world files located in the `/radar_quadrotor_gazebo` folder, change the `real_time_update` parameter to a lower value (1000 simulates at real time). Use the `rostopic hz radar_map` command during simulation to check if the output of the map is throttled, it should be 5Hz.

B.2 Changing Core Functionality

To change core functionality, a good understanding of the ROS architecture is needed. When changing a C++ file, a recompilation is needed through the use of `catkin_make`.

Changing the sensor data processing, look into the files locates at:

- `/catkin_ws/src/radar_sensor_plugin`

Changing the mapping functionality, look into the files locates at:

- `/catkin_ws/src/radar_mapping`

Changing the visualization functionality, look into the files locates at:

- `/catkin_ws/src/radar_visualization`

Changing world and environment are either done in the SDF files, or through the GUI of Gazebo. It is recommended to do most changes inside the GUI, unless the goal is to change the parameter values of objects.

References

[1] M. Eich and T. Vögele, “Design and control of a lightweight magnetic climbing robot for vessel inspection,” in *2011 19th Mediterranean Conference on Control Automation (MED)*, Jun. 2011, pp. 1200–1205.

[2] H. Huang, D. Li, Z. Xue, X. Chen, S. Liu, J. Leng, and Y. Wei, “Design and performance analysis of a tracked wall-climbing robot for ship inspection in shipbuilding,” *Ocean Engineering*, vol. 131, pp. 224–230, Feb. 1, 2017.

[3] A. Milella, R. Maglietta, M. Caccia, and G. Bruzzone, “Robotic inspection of ship hull surfaces using a magnetic crawler and a monocular camera,” *Sensor Review*, vol. 37, no. 4, pp. 425–435, Sep. 18, 2017.

[4] D. Zhang, K. Burnham, L. Mcdonald, C. Macleod, G. Dobie, R. Summan, and G. Pierce, “Remote inspection of wind turbine blades using UAV with photogrammetry payload,” in *56th Annual British Conference of Non-Destructive Testing - NDT 2017*, Sep. 4, 2017.

[5] M. Stokkeland, K. Klausen, and T. A. Johansen, “Autonomous visual navigation of unmanned aerial vehicle for wind turbine inspection,” in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2015, pp. 998–1007.

[6] C. Galleguillos, A. Zorrilla, A. Jimenez, L. Diaz, Á. L. Montiano, M. Barroso, A. Viguria, and F. Lasagni, “Thermographic non-destructive inspection of wind

turbine blades using unmanned aerial systems," *Plastics, Rubber and Composites*, vol. 44, no. 3, pp. 98–103, Apr. 1, 2015.

[7] B. E. Schäfer, D. Picchi, T. Engelhardt, and D. Abel, "Multicopter unmanned aerial vehicle for automated inspection of wind turbines," in *2016 24th Mediterranean Conference on Control and Automation (MED)*, Jun. 2016, pp. 244–249.

[8] A. Eudes, J. Marzat, M. Sanfourche, J. Moras, and S. Bertrand, "Autonomous and safe inspection of an industrial warehouse by a multi-rotor MAV," in *Field and Service Robotics*, ser. Springer Proceedings in Advanced Robotics, Springer, Cham, 2018, pp. 221–235.

[9] K. C. Peng, L. Feng, Y. C. Hsieh, T. H. Yang, S. H. Hsiung, Y. D. Tsai, and C. Kuo, "Unmanned aerial vehicle for infrastructure inspection with image processing for quantification of measurement and formation of facade map," in *2017 International Conference on Applied System Innovation (ICASI)*, May 2017, pp. 1969–1972.

[10] D. Mader, R. Blaskow, P. Westfeld, and C. Weller, "Potential of uav-based laser scanner and multispectral camera data in building inspection," in *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences; Gottingen*, vol. XLI-B1, Gottingen, Germany, Gottingen: Copernicus GmbH, 2016, pp. 1135–1142.

[11] C. Eschmann, C.-M. Kuo, C.-H. Kuo, and C. Boller, "Unmanned aircraft systems for remote building inspection and monitoring," Jan. 1, 2012.

[12] M. Caccia, R. Robino, W. Bateman, M. Eich, A. Ortiz, L. Drikos, A. Todorova, I. Gaviotis, F. Spadoni, and V. Apostolopoulou, "MINOAS a marine INspection rObotic assistant: System requirements and design," *IFAC Proceedings Volumes*, 7th IFAC Symposium on Intelligent Autonomous Vehicles, vol. 43, no. 16, pp. 479–484, Jan. 1, 2010.

[13] F. Bonnin-Pascual, A. Ortiz, E. Garcia-Fidalgo, and J. P. Company, “A micro-aerial platform for vessel visual inspection based on supervised autonomy,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 46–52.

[14] A. Ortiz, F. Bonnin-Pascual, E. Garcia-Fidalgo, and J. P. Company-Corcoles, “Vision-based corrosion detection assisted by a micro-aerial vehicle in a vessel inspection application,” *Sensors*, vol. 16, no. 12, p. 2118, Dec. 14, 2016.

[15] E. Garcia-Fidalgo, A. Ortiz, F. Bonnin-Pascual, and J. P. Company, “A mosaicing approach for vessel visual inspection using a micro-aerial vehicle,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 104–110.

[16] A. Ortiz, F. Bonnin, A. Gibbins, P. Apostolopoulou, W. Bateman, M. Eich, F. Spadoni, M. Caccia, and L. Drikos, “First steps towards a roboticized visual inspection system for vessels,” in *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*, Sep. 2010, pp. 1–6.

[17] I. Lazakis, K. Dikis, A. L. Michala, and G. Theotokatos, “Advanced ship systems condition monitoring for enhanced inspection, maintenance and decision making in ship operations,” *Transportation Research Procedia*, Transport Research Arena TRA2016, vol. 14, pp. 1679–1688, Jan. 1, 2016.

[18] Z. Gong, S. Ge, T. Guo, Q. Zhang, and Y. Chen, “A compact planar 24ghz quasi-yagi antenna for unmanned aerial vehicle radar applications,” in *2017 IEEE International Conference on Computational Electromagnetics (ICCEM)*, Mar. 2017, pp. 104–105.

[19] M. Caris, S. Stanko, S. Palm, R. Sommer, and N. Pohl, “Synthetic aperture radar at millimeter wavelength for UAV surveillance applications,” in *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Sep. 2015, pp. 349–352.

[20] M. A. Yarlequé, S. Alvarez, and H. J. Martínez, “FMCW GPR radar mounted in a mini-UAV for archaeological applications: First analytical and measurement results,” in *2017 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, Sep. 2017, pp. 1646–1648.

[21] B. Yamauchi, “Daredevil: Ultra-wideband radar sensing for small UGVs,” in *Unmanned Systems Technology IX*, vol. 6561, International Society for Optics and Photonics, May 2, 2007, 65610B.

[22] S. G. Sun, B. Cho, G. C. Park, Y. S. Kang, and S. H. Han, “UWB forward imaging radar for an unmanned ground vehicle,” in *2011 3rd International Asia-Pacific Conference on Synthetic Aperture Radar (APSAR)*, Sep. 2011, pp. 1–4.

[23] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, “Comprehensive simulation of quadrotor uavs using ros and gazebo,” in *Simulation, Modeling, and Programming for Autonomous Robots: Third International Conference, SIMPAR 2012, Tsukuba, Japan, November 5-8, 2012. Proceedings*, I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 400–411.

[24] J. Meyer. (Jan. 7, 2014). Hector quadrotor, [Online]. Available: `http://wiki.ros.org/hector_quadrotor` (visited on 06/04/2018).

[25] K. J. DeMarco, M. E. West, and A. M. Howard, “A computationally-efficient 2d imaging sonar model for underwater robotics simulations in gazebo,” in *OCEANS 2015 - MTS/IEEE Washington*, Oct. 2015, pp. 1–7.

[26] M. Ouza, M. Ulrich, and B. Yang, “A simple radar simulation tool for 3d objects based on blender,” in *2017 18th International Radar Symposium (IRS)*, Jun. 2017, pp. 1–10.

[27] K. S. Kulpa, P. Samczyński, M. Malanowski, A. Gromek, D. Gromek, W. Gwarek, B. Salski, and G. Tański, “An advanced sar simulator of three-dimensional

structures combining geometrical optics and full-wave electromagnetic methods," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 1, pp. 776–784, Jan. 2014.

[28] S. Auer, S. Hinz, and R. Bamler, "Ray-tracing simulation techniques for understanding high-resolution sar images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 3, pp. 1445–1456, Mar. 2010.

[29] J. v. Zyl and K. Yunjin, *Synthetic Aperture Radar Polarimetry*, English. Wiley, May 2011.

[30] C. Wolff. (). Doppler- effect, [Online]. Available: `http://www.radartutorial.eu/11.coherent/co06.en.html` (visited on 06/04/2018).

[31] A. Reigber, R. Scheiber, M. Jager, P. Prats-Iraola, I. Hajnsek, T. Jagdhuber, K. P. Papathanassiou, M. Nannini, E. Aguilera, S. Baumgartner, R. Horn, A. Nottensteiner, and A. Moreira, "Very-high-resolution airborne synthetic aperture radar imaging: Signal processing and applications," *Proceedings of the IEEE*, vol. 101, no. 3, pp. 759–783, Mar. 2013.

[32] S. Jablonski, "Application of radar sar for indoor visual inspection," Master's thesis, NTNU, 2018.

[33] *Radar | electronics*, en.

[34] J. Eaves and E. Reedy, *Principles of Modern Radar*. Springer Science & Business Media, Dec. 6, 2012, 709 pp., Google-Books-ID: NHXjBwAAQBAJ.

[35] M. Quigley, E. Berger, and A. Y Ng, "Stair: Hardware and software architecture," May 21, 2018.

[36] M. Quigley, K. Conley, B. P Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y Ng, "Ros: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, Jan. 1, 2009.

[37] (). Lsts, [Online]. Available: `https://lsts.fe.up.pt/` (visited on 06/07/2018).

[38] OceanScan. (). Ros imc broker, [Online]. Available: `https://github.com/oceanscan/ros-imc-broker` (visited on 06/05/2018).

[39] A. Taflove and S. C. Hagness, *Computational electrodynamics: The finite-difference time-domain method*, 3rd ed, ser. Artech House antennas and propagation library. Boston, MA: Artech House, 2005, 1006 pp.

[40] A. Mori and F. De Vita, "A time-domain raw signal simulator for interferometric SAR," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 42, pp. 1811–1817, Oct. 1, 2004.

[41] G. Franceschetti, A. Iodice, D. Riccio, and G. Ruello, "SAR raw signal simulation for urban structures," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 9, pp. 1986–1995, Sep. 2003.

[42] K. Kulpa, P. Samczynski, M. Malanowski, W. Gwarek, B. Salski, and G. Tański, "SAR raw radar simulator combining optical geometry and full-wave electromagnetic approaches," in *EUSAR 2012; 9th European Conference on Synthetic Aperture Radar*, Apr. 2012, pp. 24–27.

[43] K. S. Kulpa, P. Samczyński, M. Malanowski, A. Gromek, D. Gromek, W. Gwarek, B. Salski, and G. Tański, "An advanced SAR simulator of three-dimensional structures combining geometrical optics and full-wave electromagnetic methods," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 1, pp. 776–784, Jan. 2014.

[44] F. T. Ulaby, R. Moore, and A. Fung, "Microwave remote sensing: Active and passive, vol. III, from theory to applications," in, Feb. 1, 1986, pp. 1, 065–2, 165.

[45] T. Balz, H. Hammer, and S. Auer, "Potentials and limitations of sar image simulators – a comparative study of three simulation approaches," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 101, pp. 102–109, 2015.

[46] J. Cai and G. A. McMechan, “Ray-based synthesisog bistatic ground-penetrating radar profiles,” *GEOPHYSICS*, vol. 60, no. 1, pp. 87–96, 1995.

[47] D. Goodman, “Ground-penetrating radar simulation in engineering and archaeology,” *Geophysics*, vol. 59, pp. 224–232, Feb. 1, 1994.

[48] K. Schuler, D. Becker, and W. Wiesbeck, “Extraction of virtual scattering centers of vehicles by ray-tracing simulations,” *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 11, pp. 3543–3551, Nov. 2008.

[49] P. Wajer, E. Woźniak, W. Kofman, M. Rybicki, and S. Lewiński, “Simulation of SAR images of urban areas by using the ray tracing method with measured values of backscatter coefficients,” *International Journal of Remote Sensing*, vol. 39, no. 9, pp. 2671–2689, May 3, 2018.

[50] F. T. Ulaby and M. C. Dobson, *Handbook of Radar Scattering Statistics for Terrain.* Artech House, 1989, 357 pp., Google-Books-ID: XFN3QgAACAAJ.

[51] S. J. S. Sant’Anna, J. C. da S. Lacava, and D. Fernandes, “From maxwell’s equations to polarimetric SAR images: A simulation approach,” *Sensors (Basel, Switzerland)*, vol. 8, no. 11, pp. 7380–7409, Nov. 19, 2008.

[52] A. K. Fung, K.-S. Chen, and K. S. Chen, *Microwave Scattering and Emission Models for Users.* Artech House, 2010, 445 pp., Google-Books-ID: blMmAgAAQBAJ.

[53] (). Three sensor types drive autonomous vehicles | sensors magazine, [Online]. Available: `/components/three-sensor-types-drive-autonomous-vehicles` (visited on 05/26/2018).

[54] (). X4m05 radar sensor, X4M05 radar sensor, [Online]. Available: `https://www.xethru.com/shop/` (visited on 05/29/2018).

[55] (). X4a02 antenna board datasheet, Xethru Community, [Online]. Available: `https://www.xethru.com/community/resources/x4a02-antenna-board-datasheet.87/` (visited on 05/22/2018).

[56] (). Matplotlib: Python plotting — matplotlib 2.2.2 documentation, [Online]. Available: `https://matplotlib.org/` (visited on 06/06/2018).

[57] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey, "Outdoor mapping and navigation using stereo vision," in *Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, Springer, Berlin, Heidelberg, 2008, pp. 179–190.

[58] N. Haala, M. Peter, J. Kremer, and G. Hunter, "Mobile LIDAR mapping for 3d point cloud collection in urban areas - a performance test," *In: Proceedings of XXI ISPRS Congress, Beijing, China, July 3-11, 2008*, vol. 37, Jul. 1, 2008.

[59] I. I. Kim, B. McArthur, and E. J. Korevaar, "Comparison of laser beam propagation at 785 nm and 1550 nm in fog and haze for optical wireless communications," in *Optical Wireless Communications III*, vol. 4214, International Society for Optics and Photonics, Feb. 6, 2001, pp. 26–38.

[60] T. Peynot, S. Scheding, and S. Terho, "The marulan data sets: Multi-sensor perception in a natural environment with challenging conditions," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1602–1607, Nov. 1, 2010.

[61] A. Foessel, S. Chheda, and D. Apostolopoulos, "Short-range millimeter-wave radar perception in a polar environment," *Robotics Institute*, Jan. 1, 1999.

[62] P. L. Lowbridge. (Oct. 1, 1995). Low cost millimeter-wave radar systems for intelligent vehicle cruise control applications, Microwave Journal, [Online]. Available: `http://link.galegroup.com/apps/doc/A17762490/AONE?sid=googlescholar` (visited on 04/30/2018).

[63] F. Sadjadi, M. Helgeson, M. Radke, and G. Stein, "Radar synthetic vision system for adverse weather aircraft landing," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 1, pp. 2–14, Jan. 1999.

[64] A. Gern, U. Franke, and P. Levi, "Robust vehicle tracking fusing radar and vision," in *Conference Documentation International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI 2001 (Cat. No.01TH8590)*, 2001, pp. 323–328.

[65] J. A. Guerrero, M. Jaud, R. Lenain, R. Rouveure, and P. Faure, "Towards LIDAR-RADAR based terrain mapping," in *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, Jun. 2015, pp. 1–6.

[66] U. V. Gert Rudolph. (). Three sensor types drive autonomous vehicles | sensors magazine, [Online]. Available: `/components/three-sensor-types-drive-autonomous-vehicles` (visited on 05/26/2018).

[67] V. De Silva, J. Roche, and A. Kondoz, "Fusion of LiDAR and camera sensor data for environment sensing in driverless vehicles," *ARXIV:1710.06230 [cs]*, Oct. 17, 2017. arXiv: `1710.06230`.

[68] S. Thrun, "Robotic mapping: A survey," p. 31,

[69] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 3, pp. 249–265, Jun. 1987.

[70] G. K. Kraetzschmar, G. Pagès Gassull, and K. Uhl, "Probabilistic quadtrees for variable-resolution mapping of large environments," *IFAC Proceedings Volumes*, IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004, vol. 37, no. 8, pp. 675–680, Jul. 1, 2004.

[71] A. K. Pandey, "Feature based occupancy grid maps for sonar based safe-mapping," p. 6,

[72] (). NumPy — NumPy, [Online]. Available: `http://www.numpy.org/` (visited on 06/06/2018).

[73] P. V. (Mar. 21, 2014). River barge with open cargo hold, 3D sketch, [Online]. Available: `https://3dwarehouse.sketchup.com/model/13a9352a47bab8c98cfe472915a175bb/River-barge-with-open-cargo-hold` (visited on 06/03/2018).