CHALMERS

Marine Autonomous Research Vehicle - Object classification

Radar based object classification between moving and stationary targets at sea

Bachelor's thesis in the Department of Electrical Engineering

ANTON CARLSSON
HUGO DRAKESKÄR
FILIP LARSSON
ERIK PUNT
GUSTAV SANDRÉN
GUSTAV WIKMAN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

CHALMERS

Radar based object classification between moving and stationary targets at sea for MARV - Marine Autonomous Research Vehicle

Bachelor Thesis Project EENX16-23-02
Department of Electrical Engineering
Chalmers University of Technology
412 96 Gothenburg, Sweden 2023

Authors:

Anton Carlsson
antonca@chalmers.se

Hugo Drakeskär
hugod@chalmers.se

Filip Larsson
filiplar@chalmers.se

Erik Punt
punt@chalmers.se

Gustav Sandrén
sandren@chalmers.se

Gustav Wikman
gustavwi@chalmers.se

Examiner:

Petter Falkman
petter.falkman@chalmers.se

Supervisor:

Viktor Lindström
viktor.lindstrom@chalmers.se

SSRS Representative:

Fredrik Falkman
fredrik.falkman@ssrs.se

Radar based object classification between moving and stationary targets at sea for MARV - Marine Autonomous Research Vehicle
ANTON CARLSSON
HUGO DRAKESKÄR
FILIP LARSSON
ERIK PUNT
GUSTAV SANDRÉN
GUSTAV WIKMAN

© ANTON CARLSSON, HUGO DRAKESKÄR, FILIP LARSSON, ERIK PUNT, GUSTAV SANDRÉN, GUSTAV WIKMAN 2023.

Supervisor: Viktor Lindström, Department of Electrical Engineering
Examiner: Petter Falkman, Department of Electrical Engineering

Bachelor's Thesis: EENX16-23-02
Department of Electrical Engineering
Division of System and Control Theory
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 (0) 31 772 1000

Cover: Picture of the MARV during testing

Abstract

This bachelor thesis conducted in collaboration with the Swedish Sea Rescue Society (SSRS) aims to enhance the autonomous capabilities of the Marine Autonomous Research Vehicle (MARV), a research platform developed by Chalmers University of Technology. This will be done by implementing a marine radar to differentiate between stationary and moving targets. The MARV is intended to be fully autonomous, capable of transporting itself to accident sites or to vessels in need of assistance. The project's scope involves mounting a commercial radar on the MARV, implementing it into the current Robot Operating System version 2 (ROS2) software environment and developing algorithms to distinguish between static and dynamic objects. A non-intrusive mounting system was also constructed to mount the radar system. The implementation of radar on the MARV is expected to improve the efficiency of rescue operations, reduce the workload of transporting the Rescuerunner (RR) manually, and bring the autonomous rescue vehicle one step closer to completion. The project realized many of its goals but did not manage to run the system in realtime.

Keywords: MARV, SSRS, Radar, ROS2, Target tracking, Clustering, DBSCAN

Acknowledgements

First and foremost we would like to express our deepest gratitude to our supervisor, Viktor Lindström. Without his invaluable guidance, time, and support throughout the research process, this project would not have been possible. We in the thesis group are also grateful for each other's contributions to the production and execution of the thesis. All the group members brought valuable feedback and insights whilst presenting a "can do"-attitude.

We would like to acknowledge this year's board members of the Chalmers Autonomous Systems and Electronics (CASE) lab committee. They not only provided us with an up-to-date mechatronics lab but also provided valuable advice and assistance in various machine courses. Chalmer's Department of Electrical Engineering also deserve a nod of appreciation, for offering financial support for this research.

We would also like to express our great gratitude to the developers and contributors of OpenCPN and especially the radar-pi plugin, without them this project would have taken significantly longer. Finally we also want to express our deepest gratitude to Ola Benderius and Krister Blanch at Chalmers Revere laboratory for helping us with the selection of the radar unit and radar interface code.

Gothenburg, May 2023

List of Acronyms

ABS Acrylonitrile butadiene styrene
ACU Autonomous control unit
CAD Computer-Aided Design
CASE Chalmers Autonomous Systems and Electronics
Chalmers Chalmers University of Technology
CSV Comma separated values data format
DBSCAN Density-based spatial clustering of applications with noise
DSP Digital Signal Processing
HBW Horizontal Beam Width
INS Inertial Navigation System
IP Ingress Protection
ISO International Organization for Standardization
MARV Marine Autonomous Research Vehicle
PMMA Polymethyl Methacrylate
PLA Polylactic acid
PoE Power over Ethernet
PPM Pixels per meter
PRF Pulse Repetition Frequency
ROS2 Robot Operating System version 2
RR Rescuerunner
SCU Sensor and Communication Unit
SSRS Swedish Sea Rescue Society
TCP Transmission Control Protocol
UDP User Datagram Protocol
VBW Vertical Beam Width

Contents

List of Figures

List of Tables

1
Introduction

The integration of advanced safety technologies in vehicles has become a key priority for ensuring safety. Technologies such as radar-based collision detection, blind-spot monitoring and adaptive cruise control in cars have been increasingly integrated into modern vehicles, significantly improving safety standards. As this technology continues to evolve for all kinds of vehicles, developments are likely to be acellerate in the marine industry, especially for watercrafts such as boats and ships. The integration of these safety technologies in the marine industry can help address safety concerns - collision with other vessels or objects in the water.

In addition, safety remains a critical concern in the marine industry, where accidents can result in significant damage to vessels, injuries, or even loss of life. This thesis aims to contribute to the ongoing research of autonomous systems by utilizing the MARV, Marine Autonomous Research Vehicle.

The MARV platform has been utilized for numerous projects[1] as it is a research collaboration between Chalmers University of Technology (Chalmers) and the SSRS. The objective of the work conducted in this thesis is to integrate a radar in the current platform and develop an algorithm that uses radar detections to classify moving and stationary targets at sea.

1.1 Background

The MARV testbed, based on a Yamaha VX Cruiser HO 2018, incorporates a wide variety of sensors, actuators, electronics, and computers. By utilizing the Yamaha watercraft as its foundation, the MARV testbed serves as a platform for testing and refining autonomous capabilities. Its primary goal is to assist SSRS in advancing the development of a reliable and efficient autonomous rescue craft.

Previous projects have implemented both the hardware and software, which have been designed for high-level guidance and control tasks. The platform has been successfully tested for successive waypoint following using two independent regulators for surge and heading [1].

Another project has been the implementation of a drive-by-wire system for the MARV [2]. This project aimed to develop the necessary hardware and software for

[1] see 1.1 for additional information

remote-controlling the platform. The realized system implementation consisted of a centralized bus system connecting several nodes responsible for controlling different functions in the MARV. Throttle control was achieved by emulating control signals, and nozzle thrust vectoring was realized using a stepper motor. In addition to these projects, one project focused on an alternative navigation method for the watercraft by developing a system where the MARV was supposed to follow a lead craft during transportation [2]. This system was entirely implemented in a simulator but not fully implemented in the physical platform.

In order to achieve full autonomy, the MARV platform requires the ability to sense its surroundings and accurately identify stationary and moving objects. This will necessitate the use of sensors, including radar, which is commonly used in marine navigation. Unlike automotive radar, which is solid-state, a marine radar rotates 360 degrees and gathers data from the vessel's surrounding environment. This data will be essential for path planning and collision avoidance algorithms on the MARV. Therefore, installing and integrating a radar sensor onto the platform is a crucial step toward achieving full autonomy.

1.2 Purpose

The purpose of this project is to utilize radar technology in order to differentiate between stationary and moving objects at sea for MARV.

1.3 Aim

- Design and construct a radar mount - For the radar to be placed on the MARV, a radar mount needs to be designed and constructed. The mount must be able to withstand torsion and bending forces and be designed in a way so that the MARV still could be driven by a human.

- System integration - In order to collect and use the radar data transmitted from the radar, an interface between radar and the already integrated system in the MARV needs to be developed.

- Create an algorithm - An initial version will be developed. Subsequently, it will undergo refinement and iterations to enhance accuracy, resulting in a more precise algorithm.
  After collecting and parsing the data, an initial version of the algorithm will be developed. Subsequently, the algorithm will undergo refinement and iterations to enhance its accuracy, resulting in a more precise algorithm.

1.4 Problem definition

To accomplish the specified purpose, the research will address the following inquiries:

- How can a commercial marine radar be used in order to discriminate between objects within 300 meters?
- How can an algorithm determine the heading and velocity for the detected objects?
- How can a radar with appurtenant hardware and tracker software be integrated with the MARV-platform?
- How can the radar physically be implemented without obstructing normal operation of the vehicle?

1.5 Delimitations

This project is confined by several aspects, which are listed below:

- Design of the radar system: The project is going to use a commercially available marine radar. This means that the project will be limited in terms of the functionality and features of the radar system. Therefore the system will not be tailored specifically to the needs or custom features that could be useful for the application will not be added.

- Testing environment: Testing and trials of the solution on the MARV will be conducted in the archipelago of Gothenburg utilizing the available surroundings. By conducting testing and trials of the system only in the specified areas, the radar system may not be fully tested under a wide range of conditions and environments. This will limit the ability to identify and address potential issues with the system.

- Hardware used: Apart from the hardware related to the radar, plotter, and mount, the existing hardware on the MARV will be used. By using only the existing hardware on the MARV aside from the components necessary for the radar system, the project will be limited in terms of the capabilities of the vessel. Any additional hardware or equipment that could enhance the performance of the radar system or the overall operation of the vessel will not be added.

- Design of the radar mount: The design of the radar mount can not be, in any way, intrusive on the MARV. For example, no holes can be drilled in the hull nor can parts be cut off et cetera. By not being able to make any permanent modifications to the vessel, the project will be limited in terms of the placement and installation of the radar system. Therefore the radar mount may not be optimized for the best radar performance due to restrictions on the physical location and orientation of the system.

2

Theory

In this part of the bachelor thesis, all relevant theories will be mentioned and explained.

2.1 Robot Operating System 2 (ROS2)

ROS2 is an open source collection of tools used for creating and controlling robot applications. One of the ways this is realized is through so called nodes and topics. A node is an executable program running inside the ROS2 enviroment. Communication is done over topics, pipelines where a user-specified type of messages are sent. One way to send information to a topic is through a publisher in the node, which publishes data to a specified topic. This data can then be fetched by a subscriber in the same node or another node, which listens to a specified topic and processes the data when a new message is sent. All data sent in the ROS2 environment are saved in so called ROS bags. The bagfiles can be used to extract the data after a test, or replay the data to be able to tweak the system without running the system live [3].

2.2 Radar

When speaking about how radars are constructed and how they work, it is vital to know the basics behind them. A marine radar works by using radio waves to detect and locate objects, such as other vessels, land masses, or navigational aids in the vicinity of the radar-equipped vessel. The radar antenna rotates 360 degrees, emitting a radio wave pulse that travels outward from the vessel at the speed of light. If the radio wave pulse encounters an object, a portion of the wave is reflected back to the radar antenna. The radar system then measures the time it takes for the reflected wave to return to the antenna, as well as the strength of the returned signal. This information is used to determine the distance to the object, its bearing (direction from the vessel), and its relative size. The data obtained from these measurements are then processed by the radar display unit and presented to the operator as a visual representation of the surrounding environment, commonly referred to as a radar screen or display. Modern marine radar systems may also have additional features such as automatic target tracking, collision avoidance, and integration with other navigation systems [4].

Table 2.1: Main categories for marine radars.

Type	Frequency [MHz]	Wavelength [cm]
X-band	9200-9500	3
S-band	2900-3100	10

2.2.1 Radar categories

One property that may differ between radars is their operating frequency for the radio waves, possibly limiting the radar to a certain work area. Marine radars are typically divided into two main categories, X-band and S-Band, see table 2.1. These are the standards for the operating frequencies for marine radars according to the International Electrotechnical Commission, more specifically the IEC 62388 [5]. For additional information about the various properties of the wavelengths, see table 2.2.

Table 2.2: Comparison between radar wavelengths of 3 cm and 10 cm [6].

3cm waves	10cm waves
Shorter pulse length	Longer pulse length
Smaller beam width	Greater beam width
Smaller minimum range	Larger minimum range
Shorter wavelength	Longer wavelength
Smaller range and bearing discrimination	Longer range and bearing discrimination
Small antenna length	Larger antenna length
More clutter due to shorter wavelength	Less clutter due to longer wavelength, better detection in extreme weather conditions
Better range detection	Better long-range detection, particularly on long pulse

2.2.2 Radio waves

When a pulse radar transmits electromagnetic energy, it does not send continuous waves, but in the form of pulses, short bursts of waves. The Pulse Repetition Frequency (PRF) is between 500 and 4000 pulses per second. It is of great importance that the radar sends radar waves in pulses and not in one continuous stream. This is to allow the time necessary for the radar wave to hit a potential object, and then return an echo. [6]

In addition to the frequency and the PRF, the length variation of the pulse can also bring advantages and disadvantages. The pulse length is the span that it takes for the leading edge to the trailing edge to pass a given point. Depending on the pulse length, the amount of power a pulse contains can be determined. Longer pulses equal more power but a less clear picture and the chance of detecting poor targets from a longer distance increases. Hence, the opposite regarding a short pulse wave. A short wavelength provides a clearer picture at a short distance, whilst receiving a blurrier picture from a long distance since the wave carries less energy. All this is controlled by the gain [6].

2.2.3 Doppler-effect

The Doppler effect is a fundamental principle that plays a crucial role in the operation of radar systems, including marine radar. In radar, the Doppler effect is used to measure the relative motion between the radar and the target. [7]

When a radar signal is transmitted and hits a moving target, the frequency of the signal reflected back to the radar is shifted due to the motion of the target. This shift in frequency is known as the Doppler shift, and it is proportional to the relative velocity between the target and the radar. When the object is moving away from the radar; it is referred to as red-shifted and when the object is moving towards the radars; it is referred to as blue-shifted.[7]

2.2.4 Horizontal Beam Width (HBW)

When the radar transmits a radio wave the leading edge of radio rays reflects from the object back toward the radar receiver. The echo will be transmitted continuously until the trailing edge passes over the object. As a consequence, the target will never be less in width than the bandwidth of the radar wave. This is also known as *Beam Width Distortion* and it affects the *Bearing Discrimination* [6].

Bearing discrimination, also known as *Azimuth Resolution*, is the ability of which the radar can distinguish and separate two different targets located within the same range but at different bearings. If the leading edge of the radio wave is located at target 1 and the trailing edge at target 2, the two targets will be displayed as one large echo [6].

The relation between bearing discrimination and the antenna length and wavelength is as follows:

$$\text{Bearing Discrimination} = \frac{\text{Antenna Length}}{\text{Wave Length}} \tag{2.1}$$

Bearing discrimination is proportional to the antenna length and inversely proportional to the wavelength.

2.2.5 Vertical Beam Width (VBW)

Although the Horizontal Beam Width (HBW) is critical to detect objects close together, it is necessary to detect objects close to the antenna. A large Vertical Beam Width (VBW) ensures that the boat (during this project the MARV) is allowed to roll and pitch during usage, and still retain the ability to detect objects.

2.2.6 Works of a radar sensor

To acquire detections a radar sensor operates by following these steps: [6]

1. Transmission: A radio wave is sent out in pulses from the radar antenna. The pulse amplitude is specified by the gain parameter.

2. Reflection and Detection: The pulses then travel through the air and either get consumed by non-reflective objects or bounce back toward the sensor.

3. Time Measurement: The time it takes for the pulses to get back to the sensor is calculated and the distance to the object is calculated using the speed of light. The distance is calculated by the radar sensor as follows:

$$D = \frac{c \cdot t}{2} \tag{2.2}$$

   Where D is the distance in meters, c the speed of light, and t the time between the pulse was sent out and received.
4. Direction Measurement: The direction of the object is determined by the position of the antenna at the time it receives the reflected wave.
5. Doppler Measurement: The response pulse will along with having a time delay, also have a shift in frequency. This shift is what is known as the doppler effect. As described above this effect is used to calculate the relative velocity of a detected object. The doppler speed is calculated by the radar sensor as follows:

$$v = c \cdot \frac{f' - f}{2 \cdot f} \tag{2.3}$$

   Where v is the relative velocity between the tracked object and the radar, c is the speed of light, f is the transmitted frequency and f' is the received frequency.

2.2.7 Radar Sensor Data Abstraction

The marine radar has physical limitations in the form of fixed VBW, HBW, maximum rotational speed, sampling rate, antenna width, and transmitting power. These limitations imply that a non-perfect picture of the environment is generated and used for tracking [6].

The radar sensor turns clockwise and is by software constrained to 2048 "spokes". Each spoke (also known as scan line) is correlated to a fixed angle from the radar forwards heading and due to the number of spokes has an approximate angular separation of 0.176°. A spoke is defined as the mean radar detection intensity from the minimum range to the set maximum range of the radar. In the data stream from the radar, a spoke consists of 512 mean intensities per spoke. See figure 2.2 for a simple visualization.

These spokes are what the radar "picture" is made out of (see figure 2.1 and figure 2.3). The "picture" or radar frame can be represented as a matrix consisting of 2048 spokes with 512 detections each. This gives an approximate resolution of 1 mega sample or 1 megapixel resolution. The Pixels per meter (PPM), which is an

important resolution measure can range from 10 PPM for the minimum range to 0.01 PPM for the maximum range.

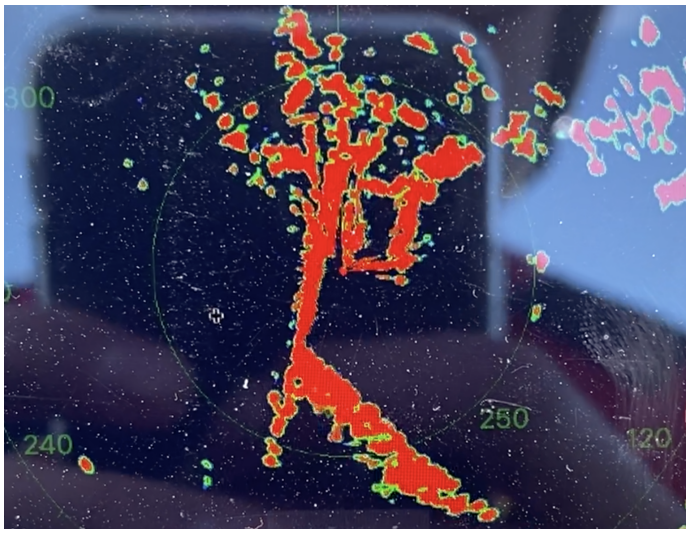


Figure 2.1: Example radar image displayed on a plotter

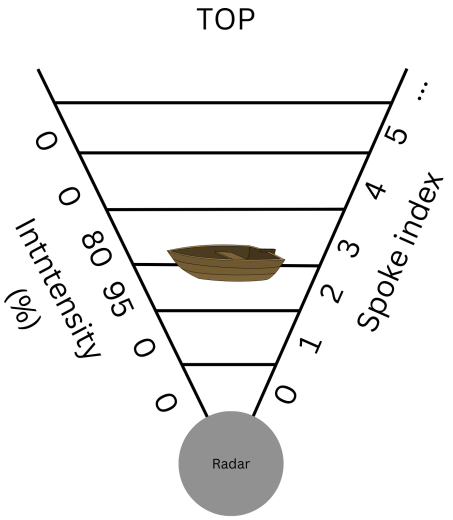


Figure 2.2: Visualisation of radar detection intensity

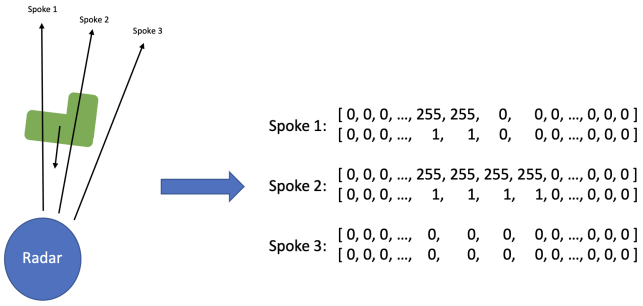


Figure 2.3: Visualisation of radar spokes

2.2.8 Radar sensor with multiple ranges

An important note is that the radar sensor used in this project has the ability to use the same radar antenna for two "virtual" sensors. These sensors are referred to as A and B, with the decision made to solely utilize sensor B in a more conventional manner. [8]

The potential use of two sensors is to have one set to look for far away objects but with low resolution, and one to look for near objects with high resolution. However, this was out of the scope of this project. [8]

2.3 Previous work on radar interfacing

Due to the proprietary nature of the marine radars available on the market, developing a radar interface is not an easy undertaking. To be able to create an interface, one must know all the bit orderings in the transmitted packages from the radar and how to send data to it. Fortunately enough the open-source project OpenCPN (a ChartPlotter/Navigator software) [9], and the plug-in radar-pi [10], already have developed a radar interface to use for the OpenCPN software. The code used in this project is highly inspired by it and has enabled this thesis to get hold of all the relevant information.

The main difference between radar-pi and what has been produced in this thesis is that radar-pi is built in C++ and for the specific software OpenCPN. The solution developed in this thesis is written in Python3 with assisting numpy calculations to increase speed. The purpose of writing it in Python3 is to easier implement it in the existing MARV ROS2 software suite.

2.4 UDP data protocol

User Datagram Protocol (UDP), which is a communication protocol used in computer networks. It is one of the core protocols in the Internet Protocol suite and

provides connectionless service to applications.

In difference to the other common communication protocol Transmission Control Protocol (TCP), UDP provides a fast but potentially unreliable service. The unreliability stems from the fact that UDP does not provide a virtual circuit to confirm the delivery of data, so it can send packets without waiting for acknowledgment or re-transmission of lost packets, whereas TCP utilizes confirmation of delivery. This makes UDP ideal for applications that require fast and efficient transmission of data.

UDP works by encapsulating data into packets, which are then sent to the destination address specified in the packet header. The header contains the source and destination port numbers, which are used to identify the sending and receiving processes on the host. Since UDP is connectionless, each packet is treated as an independent entity and the network does not maintain any state information about the packets. Therefore, packets may arrive out of order or be lost in transit, and it is up to the receiving application to manage the data and ensure its integrity [11].

2.4.1 Multicast contra Broadcast

Multicast and broadcast are two out of many technologies to send data from one source to multiple receivers. Broadcast is commonly used when many endpoints need the same information and can be seen as "simple" data transmitting where listeners select what transmitter IP address and port to listen to. Multicast works differently from broadcast and provides distinct advantages, making it a preferred choice in certain situations. There are two primary reasons for utilizing multicast instead of broadcast. The first one is when data should be selectively sent out and not be received by all nodes on the network. In the case of the radar sensor, it does by design only send radar packets to those who are listening to them. The second reason is that when listeners do not have information about who will send out the data, or the specific IP address of the sender, a multicast group can be used as common ground for the two parts. The sender sends packets to a specific group, and if there are listeners, those can, without knowing who will send, subscribe to that data through that group. The hypothesis of why these design choices were made is to save bandwidth in the network so that multiple listeners could collect data from the same radar without knowing its specific network address [12].

The communication between the sender and receivers is handled by the multicast group. The group is defined by an IP address and port. To be able to listen to multicast packets being sent, a receiving node must bind itself to the multicast group. On the other end, the transmitter must also bind to the same group. When this connection is made, a direct communication "lane" is established where the large data quantities can be sent. In the case of the specific radar unit used in this project, the address is 236.6.7.5 and the port 6878. In figure 2.5, the intersections of the lines can be thought of as the group.

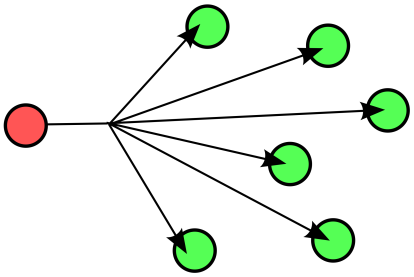


Figure 2.4: Broadcast

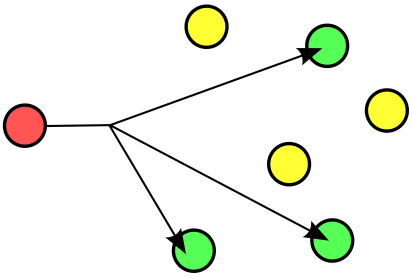


Figure 2.5: Multicast

2.5 Algorithm

In target tracking systems, various algorithms are employed to process data and make informed decisions about the existence and movement of targets. These algorithms utilize techniques such as M/N logic, Digital Signal Processing (DSP), dead reckoning, and clustering to effectively track and manage targets. Each algorithm plays a crucial role in different stages of target tracking, from initial detection to confirmation and analysis. By understanding these algorithms, valuable insights can be gained into the operations of target tracking systems and how they effectively use the data they receive.

2.5.1 M/N logic

M/N logic, commonly referred to as M-out-of-N logic, is an approach in target tracking systems [13]. It revolves around counting the number of times a track is detected within a specified number of updates. The two key parameters in this approach are M and N, representing the minimum required detections for confirmation and the total number of updates considered for confirmation or deletion.

As detections are assigned to a track, the detection count is incremented. If the count reaches or surpasses the value of M within the last N updates, the track is confirmed as a true target. On the other hand, if the track goes without any detection for N consecutive updates, it is flagged for deletion. By adjusting the values of M and N, the sensitivity and accuracy of the logic can be fine-tuned. Higher values of M mean stricter confirmation requirements, necessitating more detections for confirmation. Increasing N provides tracks with a greater window of opportunity to gather detections before potential deletion.

Tracks in the M/N logic can be confirmed and deleted using two main approaches, history-based logic and score-based logic:

- History based: Tracks are confirmed by counting the number of assigned detections within a recent time frame. If a track accumulates enough detections during this period, it is considered confirmed. On the other hand, if a track

goes without any assigned detections for a specific duration, it is deleted. This approach relies on the track's detection history to determine its confirmation and deletion status, providing a straightforward method for track management in target tracking systems.

- Score based: This approach calculates a score representing the likelihood of a track being a true target. A high positive score indicates a high probability of the track being valid, while a significantly negative score suggests it is likely false. Tracks can be confirmed if their scores surpass a threshold, and they may be deleted if scores fall below a certain level.

2.5.2 Digital Signal Processing

DSP is used to convert a raw signal into a form of data more suitable for the algorithm. A DSP can exist in many forms and is used as a term; and not a specific object or software.

2.5.3 Dead reckoning

Dead reckoning is a method of navigation used to estimate one's current position based on the direction, speed, and elapsed time from a known starting point. It involves using a previously known location and estimating the new position based on factors such as distance traveled and heading. While dead reckoning can provide a rough estimate of one's position, it is not always accurate and can be affected by factors such as wind, currents, and other external forces. As a result, it is often used in conjunction with other navigation methods, such as GPS or celestial navigation, to obtain a more precise location [14].

2.5.4 Clustering

Density-based spatial clustering of applications with noise (DBSCAN) (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm used to group together points in a dataset based on their proximity to each other. The algorithm starts by selecting a random point and finding all of the points within its epsilon radius. If there are at least minPts points within this radius, the algorithm creates a new cluster and adds all of these points to it. It then recursively expands the cluster by finding all of the points within the epsilon radius of each point in the cluster, and adding them to the cluster if they have at least minPts points within their epsilon radius [15].

If a point is found that does not have enough neighboring points to form a dense region, it is considered an outlier or noise point. The noise handler in DBSCAN is a feature that identifies and handles these noise points. This is important because in real-world datasets, there may be points that do not belong to any cluster, and it is important to identify and account for them in the analysis.

The number of clusters in DBSCAN is not specified beforehand, unlike in k-means clustering, where the number of clusters must be specified in advance. DBSCAN is capable of detecting clusters of arbitrary shape and can handle noisy data. The resulting clusters can be of different sizes and densities. The plots showcased in figure 2.6 illustrates how the epsilon parameter alters the number of clusters identified. The black points are regarded as noise.

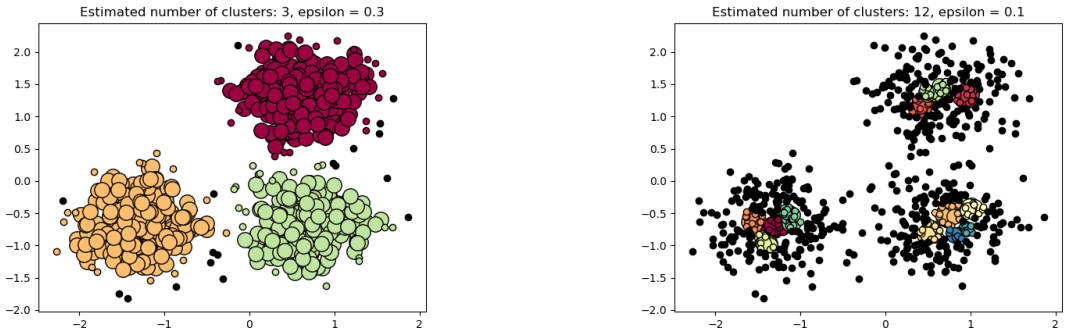


Figure 2.6: DBSCAN with epsilon = 0.3 to the left and 0.1 to the right, minPts = 10

The plots showcased in figure 2.7 illustrates how the minPts parameter alters the number of clusters identified. The black points are regarded as noise.

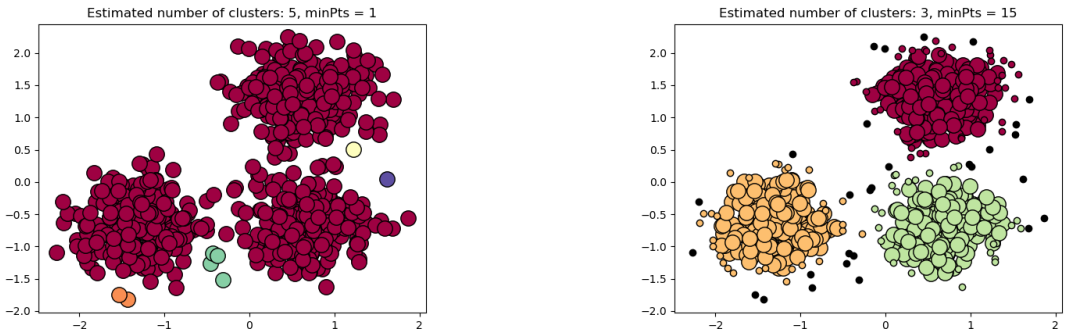


Figure 2.7: DBSCAN with epsilon = 0.3, minPts = 1 to the left, minPts = 15 to the right

2.5.4.1 Optimization of DBSCAN

scikit-learn offers a Python3 implementation of DBSCAN with a complexity of $O(n \cdot d)$ where d is the average number of neighbours [16]. In 2020 a group of researchers developed a theoretically-efficient and practical parallel DBSCAN providing a lower complexity of $O(log(n))$ for 2D data. This drastically improves the computational speed by up to 32 times [17]. The theoretically-efficient DBSCAN is used in Pyhton3; but the algorithm itself is called from a C-api. Figure 2.8 published by the developers illustrates the time it takes to cluster a 2D data set containing 50000 data points on a 6-core computer with 2-way hyperthreading

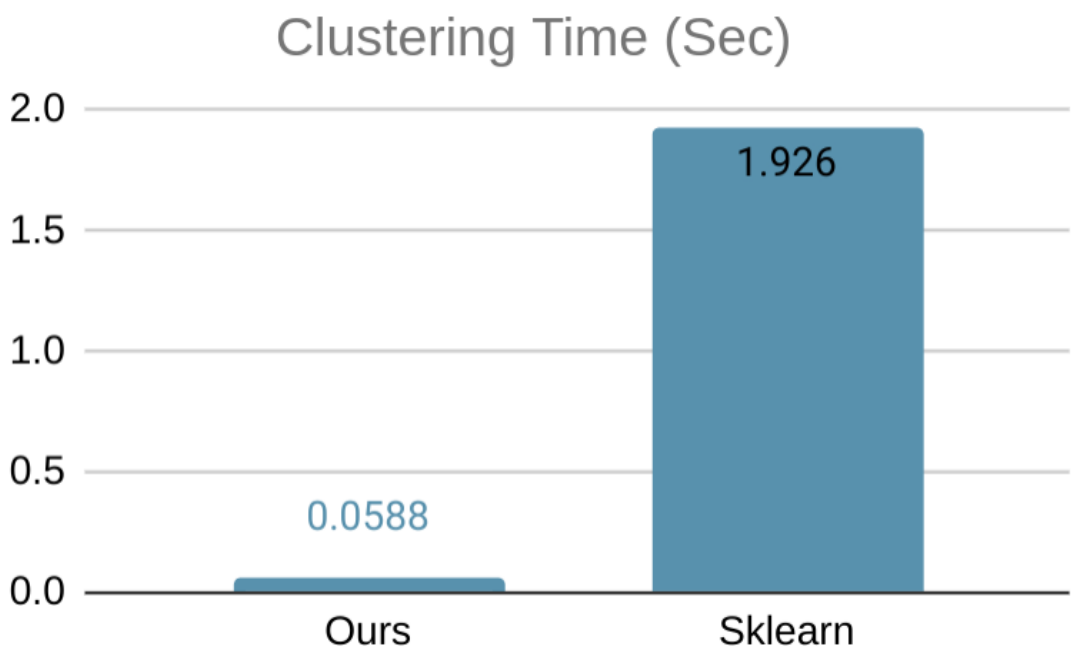


Figure 2.8: Comparison of theoretically-efficient DBSCAN and scikit-learn's DBSCAN.

2.6 Radar mount

The radar must be mounted on the MARV at a desired height to work correctly and not be blocked by the driver. A construction supporting the radar will be needed since the MARV, as its own, does not have any mounting brackets. Knowing the weight of the radar and the height it is supposed to be mounted, certain aspects regarding, for example, material selection and solid mechanics were considered. During this part of the thesis, a brief presentation about the theoretical background used to design and manufacture the mounting structure will be described.

2.6.1 Product development process

The product development process is a systematic approach that involves the identification of needs, the creation of a product concept, and developing and launching the product. In this project, the process was modified to fit the specific needs. The process was the foundation for the entire development of the structure [18].

Below, all significant steps in the process that has been executed are listed and explained.

Ideation
Gather ideas that could solve the problem from various sources. This part of the process includes, for example, internet searching, brainstorming, et cetera.

Requirement specification
Develop a detailed document outlining the product's functional, technical, and other requirements. This document is created before design work begins.

Concept development
Create an overall concept for the new product, including its features, characteristics and design.

Prototype development
Building a working model of the product to test and refine.

Testing and validation
Testing the prototype to ensure it meets the requirements and functions correctly. Based on the input, improvements are made.

2.6.2 Adaptive manufacturing

Adaptive manufacturing leverages technology and tools for greater flexibility and responsiveness. The goal is to create a manufacturing system that can quickly and efficiently produce customized products in small series while adapting rapidly to changes, for example, demand and material[19].

One of the technologies driving the development of adaptive manufacturing is 3D printing. Also known as additive manufacturing, 3D printing is a process whereby a digital model of a product is created using computer-aided design (CAD) software and then printed layer-by-layer using various materials, such as plastics and metals. The advantages of 3D-printing for adaptive manufacturing are numerous. It enables rapid prototyping of new products, allowing to iterate and refine designs quickly. It also allows for the creation of complex geometries that would be hard to produce using traditional manufacturing methods.

Waterjet cutting is a form of adaptive manufacturing. It is a flexible and versatile manufacturing process. Waterjet cutting uses a high-pressure stream of water mixed with abrasive materials (often some sort of sand) to cut through the desired material. It can cut intricate details, which is useful when working with complex geometries. It is also a digital manufacturing process that can handle digital files from Computer-Aided Design (CAD) tools.
Laser cutting is also an adaptive manufacturing tool due to its high flexibility and possibility to adapt to various materials. This tool uses a highly focused laser beam to cut through the specified material. Precisely like the waterjet, it has high accuracy on intricate details. It is also a digital manufacturing process, so the same files can be used here. While the waterjet can cut through almost anything, the laser that has been used is limited to Polymetylmetakrylat (PMMA) and other soft materials.

2.6.3 Material and components

The 3D printers' filaments have been Acrylonitrile butadiene styrene (ABS) and Polylactic acid (PLA) filaments. ABS is a petroleum-based thermoplastic that is strong, durable, and heat-resistant. It has a higher melting point than PLA, which makes it more suitable for printing objects that require greater strength and durability. PLA is a biodegradable material from renewable resources. It is easy and fast to print with and has a low melting point. Due to their melting points and other material characteristics, PLA is better suited for prototyping and ABS for the final parts [20].

In the project, aluminium is used in several applications. When exposed to saltwater, aluminium forms a natural oxide layer with corrosion resistance, making it an appropriate material choice for marine environments. Nevertheless, aluminium can still corrode in saltwater under certain conditions. A galvanic cell can form when it is in contact with other metals. This leads to accelerated corrosion of the less noble metal. Aluminium is, for example, less noble than steel, copper, and zinc [20].

Steel screws and fasteners have been used to attach the different components due to their capability to withstand high loads and stresses. As previously mentioned, this can cause issues with galvanic corrosion when in contact with other metals [20].

Polymethyl Methacrylate (PMMA), also known as acrylic- or plexiglass, has been used for prototyping. PMMA is a transparent thermoplastic that can be shaped by using laser cutting. It is also relatively light (1.18 g/cm^3), resistant to UV radiation and outdoor environment [20].

2.7 Cables and connectors

If a cable is to be used at sea or in a so-called hazardous environment, the cable and connectors should have a certain Ingress Protection (IP) rating, also known as IP-code. This is vital since a higher IP code results in greater protection for the cable connections from dust and water, keeping the connections safe [21]. The cables themselves are already water- and dust protected, hence them not needing any additional protection. Some connections are just fixed directly onto a surface, thus needing a silicone seal or a self-sealing cap to be able to achieve the correct IP class.

During this project a wide variety of different brands have been used, a selection of the most frequently used is listed below:

- Bulgin 400 series[1]
- Bulgin data connectors[2]
- Telegärtner coaxial cable[3]
- N-type antenna connector[4]

All connections used are certified to withstand the conditions the MARV might encounter during its testing and during its runs.

[1]See more on Bulgins own webpage [22]
[2]See more on Bulgins own webpage [22]
[3]See more on Telegärtner own webpage [23]
[4]See more on Amphenol RF own webpage [24]

3
Method

3.1 Choosing and purchasing radar

To choose a radar, different models were listed in a table, along with a specification of certain features. This was done to get an overview of the potential candidates. The features included in the table were:

- Diameter of housing [mm]
- Diameter of antenna [in]
- Maximum rotational speed [rpm]
- Vertical beam width [deg]
- Horizontal beam width [deg]
- Output power [W]
- Minimum detection range [m]

To make the selection, the models were then ranked from 1-4 on five different criteria, with 1 being best and 4 being worst. The criteria, with a brief explanation of why they were deemed relevant, can be seen below:

- Horizontal beam width. This is analogous with resolution.
- Ease of software implementation. Radars with already existing infrastructure for extracting data frees up more time for other parts of the project.
- Weight. A lighter radar will cause less stress on the mount.
- Size. A smaller radar will cause less drag and put lower demands on the mount.
- Rotational speed. A faster rotation means a quicker update of the radar picture, which is crucial when tracking objects.

The radars with the lowest aggregated score were discussed in the group and with the supervisor, and a final choice was made.

3.2 Construction of radar mount

To find the, based on the resources for the project, best solution to mount the radar on the MARV, a problem statement was formulated. Separate subsystems was also defined. Based on these formulations, a creative ideation process was started to gather many ideas on how to solve the problem. The brainstorming process was conducted together with the part of the group working with the radar mount. These formed a pool of ideas that was used as a base for refinement and the development

of combinations of ideas.

To narrow the number of options available, a requirement specification was created. The requirement specification serves as a detailed description of the requirements that the radar mount must meet to be considered acceptable and functional. The specification takes in different crucial aspects of the system and acts as a framework for the continued choices made. Below are the parameters of the specifications shown:

Requirements

- Be able to carry: Radar antenna, Two cameras, Electronics box, Power cable, CAN bus cable, and Ethernet cable
- Removable
- Maximum weight <10 kg regardless of its center of gravity
- Combination of arch and radar must not exceed a total weight of 20 kg
- Adjustable (up and down)
- Must withstand 20 work cycles (one work cycle consists of assembly, disassembly, and testing)
- Corrosion-resistant (must withstand marine environment)
- Must be mounted in the handle behind the third seat
- Non-intrusive mounting
- Radar must be parallel to the waterline of the MARV when mounted
- Radar mount must allow unobstructed view within 20 degrees downward
- Sufficiently torsion-resistant (challenging to specify)
- Radar signal can only be blocked directly aft (also applies in negative vertical direction, i.e. under the radar antenna, 10 cm)
- Radar antenna may not be blocked by more than an object equivalent to 4 cm in size
- Rear-facing camera, unobstructed
- The two cameras should cover at least 320°
- Must not block radar sensor LED lights
- The plotter comes with a mount that can be attached to the mounting device

Desires

- Aesthetically pleasing
- Streamlined design
- Low weight
- No obstruction of the radar antenna in any direction
- Cameras should cover 360°

When the pool of ideas had been consolidated into a smaller amount of concepts, their ability to be realised was investigated. Suppliers were contacted, and brief geometrical investigations were made. Finally, a concept where chosen to develop further.

The construction and building of the first functional prototype were developed by utilising available resources in CASE and CAD. The prototype was improved

through several iterations based on input from the supervisor, team members and testing of mounting geometry and stability.

A complete and working solution was launched to be tested during the first sea trials. The sea trials can be considered as the most complete test during the project. Other tests included applying high bending moments, rocking movements to simulate waves and geometry assurance and compatibility with other parts and hardware. Based on the input from the first sea trial, several improvements were implemented. This version is the final and is the base for the drawings and BOM (see appendix A).

3.3 Construction of the Sensor and Communication Unit

The purpose of the SCU is to have a durable box containing all components needed to communicate between the REACH computer, the radar, a camera, and an external computer through 5G Wi-Fi.

A durable box strong enough to withstand both water and fierce weather has been used and the rest of the components were either mounted on or inside the SCU. A list containing all the components related to the box is written below:

- Router - The router contains two SIM cards enabling the usage of Wi-Fi and 5G transmitting and also provides two Gigabit Ethernet (GE) ports and two Fast Ethernet (FE) ports for fast data communication. Besides that, the router has six coaxial connectors offering radio transmission.

- Antenna - Connected to all the coaxial cables are six antennas. Four allow 5G transmission and two Wi-Fi transmissions.

- Patch cables - Between the four ethernet ports located at the router and the box are four patch cables installed. These allow data to be transmitted between the router - the radar, REACH, and Power over Ethernet (PoE).

- Boost converter - Using a boost converter inside the SCU enables the use of up to 48V as output by PoE while the input is 12V.

- Switch - On the outside of the SCU a switch is located. This activates the awake command for the radar.

Below, in Figure 3.1, is a picture of the SCU:

Figure 3.1: The internals of the SCU

3.4 Development of a Radar Interface

The radar interface is a software stack that handles the incoming network data from the radar unit. The software stack is divided into three main classes that handles the different requirements for processing the radar data. The radar interface is written in Python3 and was inspired by two existing projects written in C++ [10] [25]. In order to ensure potential compatibility with other brands in the future, the modules have been labeled with the prefix "*Navico*," derived from the radar manufacturer of the *Lowrance HALO24* radar unit. This nomenclature was chosen to allow for future integration with additional brands in the pursuit of enhanced compatibility.

The radar communicates exclusively using the UDP communication protocol in multicast groups.

3.4.1 Python3 Class structure

The radar interface is divided into Python3 classes. Dividing the interface into three classes were done for the sake of having a clear structure of the different communication sequences and functionalities required by the radar unit. These classes are sequentially initiated by the /marv_radar_interface ROS2 node. Below is a description of how these classes interact and the functionalities they provide. A flowchart over how the logic works is provided in figure 3.3.

3.4.1.1 NavicoLocate

The location class has the responsibility of waking up the radar and acquiring a package that describes the IP addresses of which the radar uses for transmitting and receiving. The sequence is as follows.

1. The locating function **locate** inside the class NavicoLocate is called. In that function, two separate threads are initialized, where the first one points to the function **receive_udp_packet** that listens for a UDP packet with the length 222 and first two bytes with the value of $[0x01, 0xB2]$. This is the IP address configuration package. The latter function **wake_radar** sends out a packet of two bytes $[0x01, 0xB1]$ (hexadecimal) which tells the radar to send out the aforementioned packet of length 222.

   Both the send and receive functions communicate using multicast groups.

2. When the address configuration packet is received, it gets parsed and stored in nine variables as described in table 3.1.

   When done, the variable **addr_acquired** is set True and the next module takes over.

Table 3.1: Variables in wake-up packet

Variable name	**Usage**
id	Unique identifier for the radar unit
serialno	Serial number of the radar unit
addr0	The ip address the radar has to the router
addrDataA	Not used (see section 2.2.8)
addrSendA	Not used (see section 2.2.8)
addrReportA	Not used (see section 2.2.8)
addrDataB	Where spokes are received
addrSendB	Where config and heartbeat is sent to the radar
addrReportB	Where the config reports are received

3.4.1.2 NavicoControl

This part of the code is intended to send various commands to the radar, and thus set various parameters which in turn control the settings and adapt the radar to different environments. The code sequence is as follows:

1. Inside the class **NavicoControl** the **start** and **stop** definitions are initialized on command. When receiving the start command, the code confirms that the radar receives the correct IP address and wake-up call from **NavicoLocate**, sending the **self.start_stayalive_thread()** command keeping the radar on. It is essential that this step passes unhindered since the wrong IP address will lead to a loss of connection, and the radar will not be started. On the contrary, the stop command - **self.stop_stayalive_thread()**, will stop the radar.

2. When initialized, the radar needs to stay awake. This is done by repeatedly sending a sequence of four commands with three bytes $[0xa0, 0xc1]$, $[0x03, 0xc2]$, $[0x04, 0xc2]$, and $[0x05, xc2]$ to the address SendB every four-seconds.

   Furthermore, two functions **RadarTxOff** and **RadarTxOn**, enable the control of turning off and on the transmission from the radar, but still keeping the radar going. To turn on the radar transmission the packet $[0x01, 0xc1, 0x01]$ (hexadecimal) is sent. To turn off, $[0x01, 0xc1, 0x00]$ is sent.

3. Besides the start-up functions and stop functions mentioned above, the different control commands are listed below in Table 3.2.

Table 3.2: Controllable functions

Function Name	Function description
set_range	The maximum range the radar detects objects
set_gain	Adjusts the sensitivity of the radar, also option to set auto gain
set_sea	Adjusts the sensitivity of interference from sea and waves.
set_rain	Adjusts the sensitivity of interference from rain.
set_side_lobe_suppression	The amount of sidelobe allowed
set_interference_rejection	Limits the detection of unwanted reflections
set_target_expansion	Override and increase the default radar pulse length, providing larger target return
set_target_boost	Increasing the size of the targets
set_scan_speed	The speed of which the radar spins, 0->3
set_mode	Selects predefined scenario mode like Offshore, harbour.. mode 0=custom settings is used
set_noise_rejection	Sets thresholds below echoes are suppressed
set_doppler	Doppler mode to use. 0=off, 1=Objects both receding and approaching, 2=only approaching objects
set_antenna_height	The height of where the radar is placed above water level
set_halo_light	Turns on and off the light

3.4.1.3 NavicoReceive

The **NavicoReceive** class aims to process spokes and reports transmitted by the radar to the Autonomous control unit (ACU) The code sequence is as follows:

1. The first step is the part where all the data is received. Once the radar starts transmitting, the following two functions - **receive_spokes** and **receive_reports** starts to receive data and sending it to **process_spokes** and **process_report**. These functions start to parse the spokes (status, indexes, angle, and intensities) and reports (status of the radar, with settings such as gain, sea clutter, scan speed and more are set to).

2. The **def process_report** function contains conditional statements that depending on the input value enters five different radar settings where information about the current radar configuration will be parsed. The different potential packets are listed in Table 3.3.

Table 3.3: Radar sensor configurations reports

Report code	Code size	Information received
RadarReport_01C4_18	18 byte with 01 C4	Radar status: -Radar standby -Radar transmitting -Radar spinning up
RadarReport_02C4_99	99 byte with 02 C4	-Range - Gain - Rain clutter - Mode - Target expansion - Interference
RadarReport_04C4_66	66 byte with 04 C4	- Antenna height - Halo light - Bering alignment
RadarReport_08C4_21	21 byte with 08 C4	- Scan speed - Sea clutter - Noise rejection - Side lobe suppression - Doppler state

3. Once a packet is received by the multicast socket, it is sent to **process_spokes**. In this function the packet is split into 32 separate spokes packets of data length 536 (see byte structure in figure 3.2). This is due to that the radar sensor sends out a collection of 32 spokes, stacked back-to-back over the network every 15ms. The individual spokes are stripped from a heading consisting of 24 bytes, that contains the status of the radar sensor, the spoke index which the detections was taken from, among with more data that is currently not used. This ultimately results in 512 detections. The current angle of the detections are calculated as following: $\frac{\text{spoke index}}{2048} \cdot 360.0°$.

These detections is then put into a 2048x512 matrix that hold all the detections for a complete rotation. It is this matrix that is later published in the ROS2 network. If doppler is activated the detections are note valid without some more parsing. This it due to a limitation in the system design of the radar unit. It does not send separate information about doppler hits, but does instead send it "baked" into the detection intensities. Here is an explanation of how these doppler-intensities are parsed:

For each separate value of the 512 indexes:

1) Split the byte into a right and left nibble (4 bits).
2) If either nibble is 0x0f or 0x0e, then there is a doppler detection for that detection.
3) If both right and left is 0x0f or 0x0e then the detection at that index is approaching you (0xff) or receding from you (0xee). The corresponding detection intensity for this index is full (255 or 100%).
4) If the byte looks like 0xfe or 0xef then it corresponds to the border of an detected object and no conclusion of direction can be drawn. But the intensity is 100%.
5) If one of the nibbles has a value of 0x0f or 0x0e and the other a value between 0x00 and 0x0d, then a conclusion about both direction and intensity can be drawn. The direction corresponds to approaching if 0x0f and receding if 0x0e. The intensity if calculated as follows: $\frac{\text{nibble value}}{13} \cdot 255$
6) If neither 0x0f or 0x0e is present in the nibbles and doppler is activated, then no conclusions can be drawn about direction, and the intensity is calculated as follows. $\frac{\text{left nibble value+right nibble value}}{26} \cdot 255$

These intensities are store the same way in the big matrix, and the doppler directions are stored in a similar matrix with values: 0=doppler activated but no direction, 1=approaching, 2=receding, 255=doppler not activated.

When a rotation is completed the ROS2 publishing function places the spoke indexes, the angles, the intensities and doppler_directions in a custom ROS2 message and publishes it so that the /marv_radar_data_processing node can receive the data and process it (see figure 3.5). When the data processing is done, the tracking and clustering done in /marv_radar_tracking is initiated (see figure 3.7).

	-->																							<--	-->				<--
	HEADER																								Radar detections				
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	...	535
Data	header lenght	status	scan number		-	-	large range		angle		heading		small range		rotation		-	-	-	-	-	-	-	0xa0	Detecti on 0	Detecti on 1	Detecti on 2	...	Detecti on 511

Figure 3.2: Spoke packet byte structure

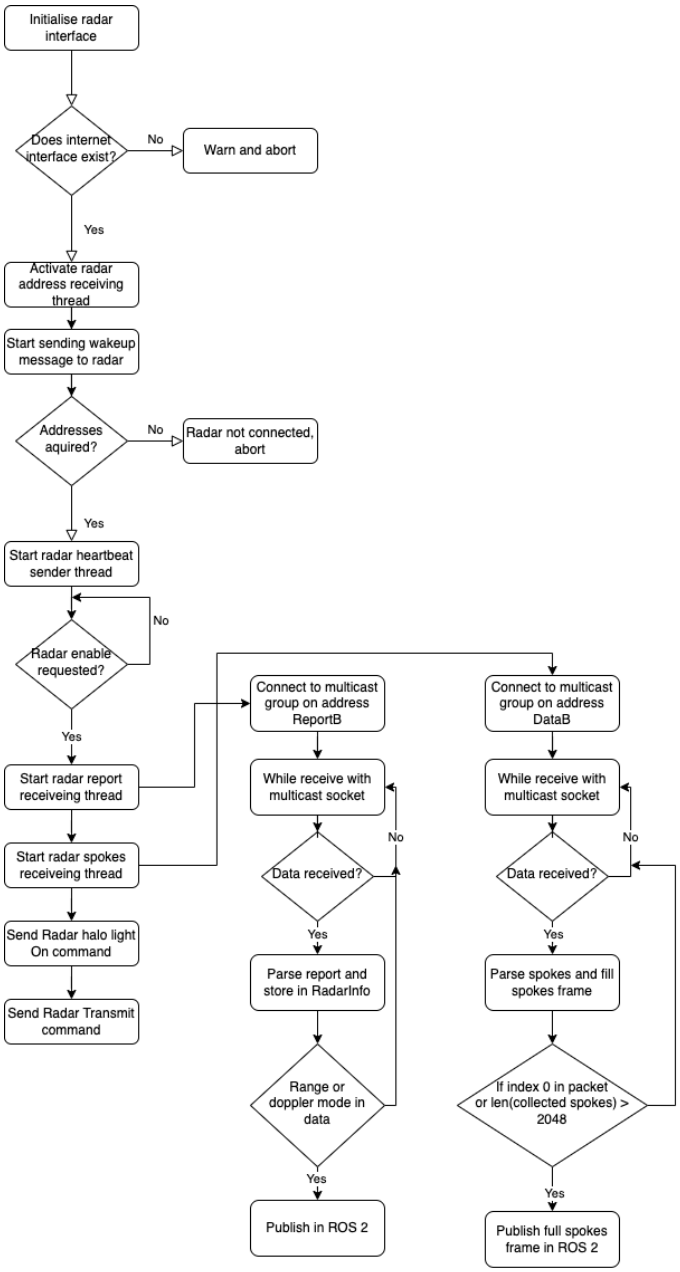


Figure 3.3: The flow of how radar reports and radar spokes gets processed

3.5 ROS2 Structure

The ACU hosts the ROS2 operating system in which multiple nodes run. These nodes provides all the functionality for sensor input and autonomous drive. All the nodes is can be seen in figure 3.4.

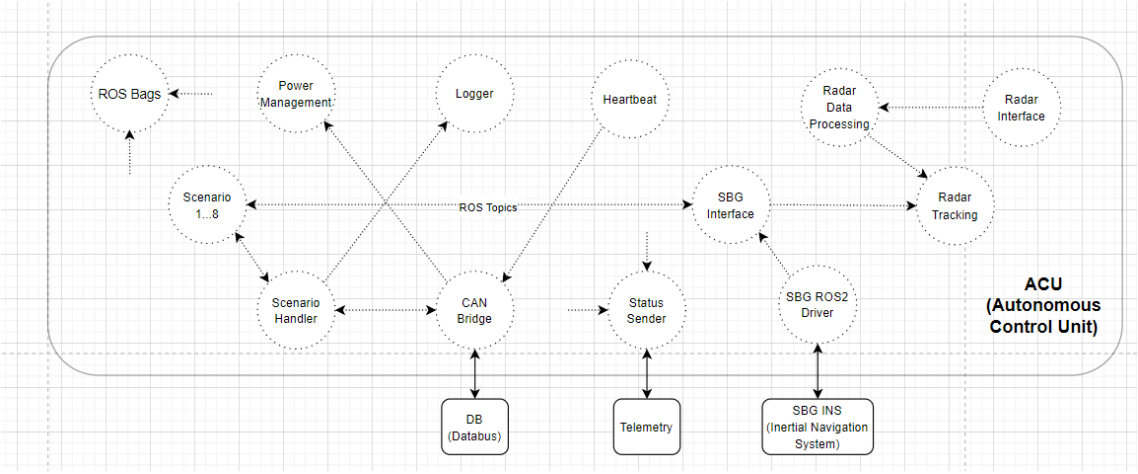


Figure 3.4: The ROS2 node structure

3.5.1 ROS2 topics

There are multiple nodes in the MARV ROS2 system that communicates with each other. All the ROS2 topics of which are published will not be listed, but instead describe those that are of essence for the radar and object detections. The complete structure is well described in the Master thesis by V. Lindström and N. Danielsson [1]. Below is a description of the topics being published from the three nodes added in this project to the MARV platform. Included is also what topics that these nodes depend on and subscribes to.

Table 3.4: ROS2 published topics

Node name	Published topic	Message type	Publish rate
/marv_radar_interface	/marv/nav/radar/spoke	RadarSpoke	1 Hz
/marv_radar_interface	/marv/sys/status/radar_state	Int8	2 Hz
/marv_radar_interface	/marv/sys/status/radar_doppler_state	Int8	2 Hz
/marv_radar_interface	/marv/sys/status/radar_range	UInt16	2 Hz
/marv_radar_data_processing	/marv/nav/radar/detections	RadarDetections	1 Hz
/marv_radar_tracking	/marv/nav/radar/radar_object	RadarObject	Multiple every second, 1 Hz
/marv_radar_tracking	marv/nav/radar/INS_data	INSData	1 Hz

Table 3.5: ROS2 subscribed topics

Node name	Subscribed topic	Message type
/marv_radar_interface	/marv/sys/ctrl/radar_enable	Bool
/marv_radar_interface	/marv/nav/radar/config	Int8
/marv_radar_data_processing	/marv/nav/radar/spoke	RadarSpoke
/marv_radar_data_processing	/marv/sys/status/radar_range	UInt16
/marv_radar_tracking	/marv/nav/radar/detections	RadarDetections
/marv_radar_tracking	/marv/nav/sys_time	UInt64
/marv_radar_tracking	/marv/nav/sbg_current_pos	Vector3
/marv_radar_tracking	/marv/nav/sbg_pose	PoseWithCovariance
/marv_radar_tracking	/marv/nav/radar/toggle_doppler	Int8
/marv_radar_tracking	/marv/sys/status/radar_range	UInt16

3.5.2 ROS2 Custom messages

ROS2 custom messages are a way to publish data that does not easily fit into the predefined message types. The structure of the custom message is defined in a MSG file, after which the custom message can be used the same way as any other message type. The custom messages used during the project are specified in table 3.6, 3.7, 3.8, and 3.9.

Table 3.6: RadarSpoke Custom Message

Data type	Name	Description
float32[2048]	angle	The angle 0.0° → 360.0° from "straight ahead" of the radar sensor.
uint16[2048]	spoke_index	0 → 2047. Index 0 = 0.0°
uint8[1048576]	spokes	2048 spokes * 512 detections flattened. Each detection has intensity 0 → 255
uint8[1048576]	doppler_spokes	2048 spokes * 512 doppler detections flattened. 0=no doppler detection, 1=approaching, 2=receding, 255=doppler off.

Table 3.7: RadarDetections Custom Message

Data type	Name	Description
float32[]	x	X position in meters from center of radar sensor (right - left).
float32[]	y	Y position in meters from center of radar sensor. (front - back)
uint8[]	intensity	0 → 255
uint8[]	doppler	0,1,2 or 255

Table 3.8: RadarObject Custom Message

Data type	Name	Description
uint32	time	epoch time in ms
uint16	object_id	0 → 65535 automatically increasing index for each tracked object.
uint32	score	Continually increasing score for object
float32	velocity	Velocity in m/s
float32	heading	From true north, not heading of vessel, in degrees
float32	latitude	Global coordinates. Ex 52.32131
float32	longitude	Global coordinates. Ex 11.3123123
float32[4]	bbox	{top_left_x, top_left_y, bottom_right_x, bottom_right_y} in meters from center of object

Table 3.9: INSData Custom Message

Data type	Name	Description
float32	x_diff	Distance traveled longitude wise
float32	y_diff	Distance traveled latitude wise
float32	heading	Continually increasing score for object
float32	velocity	Heading
float64	elapsed_time	Time elapsed since last tracking iteration
unint64	num_detections	Number of received and filtered radar detections

3.5.3 ROS2 Bags

The ROS2 bags used in this project are used to log all the messages sent between the nodes. These bags can be configured to log messages from all nodes or specified topics. Once the bags have been saved by turning off the system, the data can be analyzed in a software visualizing tool built in Python. This tool enables users to select specific time intervals from which data can be extracted and subsequently saved in a Comma separated values data format (CSV) file for further analysis [26].

3.6 Development of object tracking algorithm

To learn more about the subject of using sensor data to detect, classify, and track objects, a literature study of previous, similar projects, was conducted. The literature examined consisted mainly of scientific articles, conference papers, and previous thesis papers. The knowledge gathered was then used to separate the algorithm into smaller parts, each of which was then described using pseudo-code. The algorithm was then realized and tested on both simulated data and data gathered from the radar.

3.6.1 ROS 2 nodes flowchart descriptions

The data processing done in the ROS 2 node /marv_radar_data_processing can be seen in figure 3.5
The clustering done in the ROS 2 node /marv_radar_tracking can be seen in figure 3.6

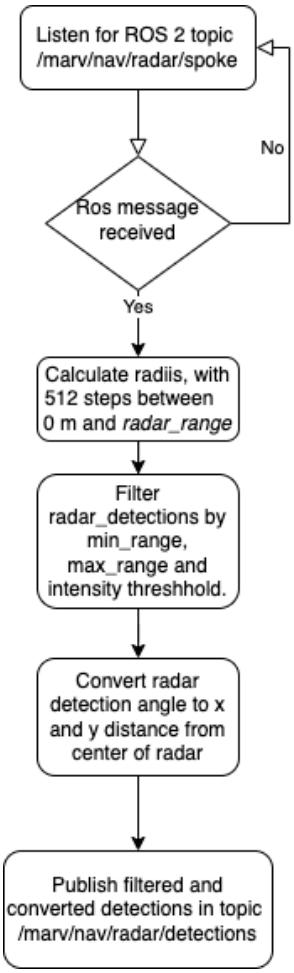


Figure 3.5: The flow of how radar detections are filtered and processed

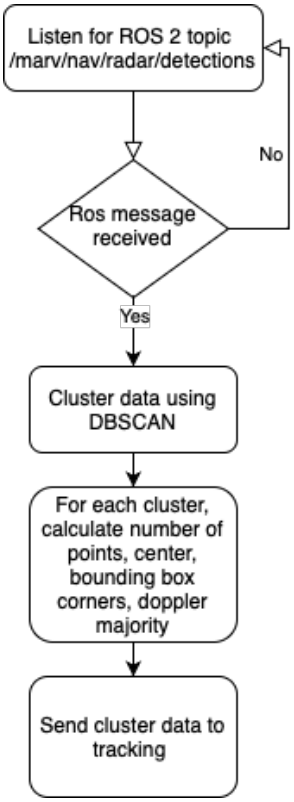


Figure 3.6: The flow of how processed radar detections are clustered and used to create new objects

The tracking done in the ROS 2 node /marv_radar_tracking can be seen in figure 3.7

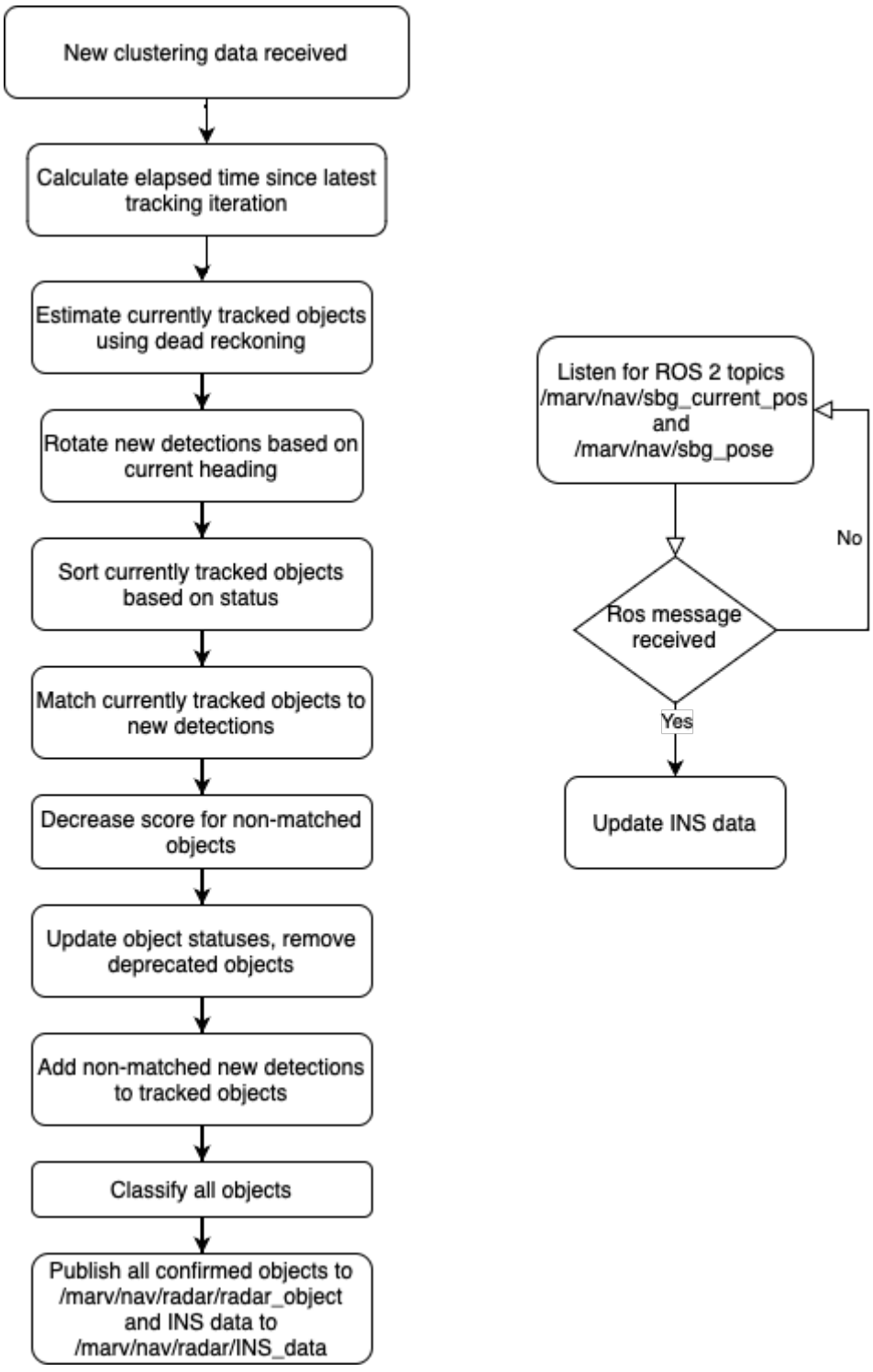


Figure 3.7: The flow of how the tracking module performs one tracking iteration

3.7 Verification of algorithmic performance

Both to tune the algorithm, and to verify the tracking and classification of objects, the algorithm output was compared to a ground truth. In the case of simulated data, the ground truth was trivially each objects position and velocity. When testing on MARV, a camera was used as ground truth. The start time of radar data collection was determined using GPS-time [27], allowing for accurate synchronization with the video. This synchronization enables visual confirmation of the objects in the collected data.

3.8 System overview

To get a better understanding of the system and its connections, below in Figure 3.8 is the system of the MARV with all its components and areas.

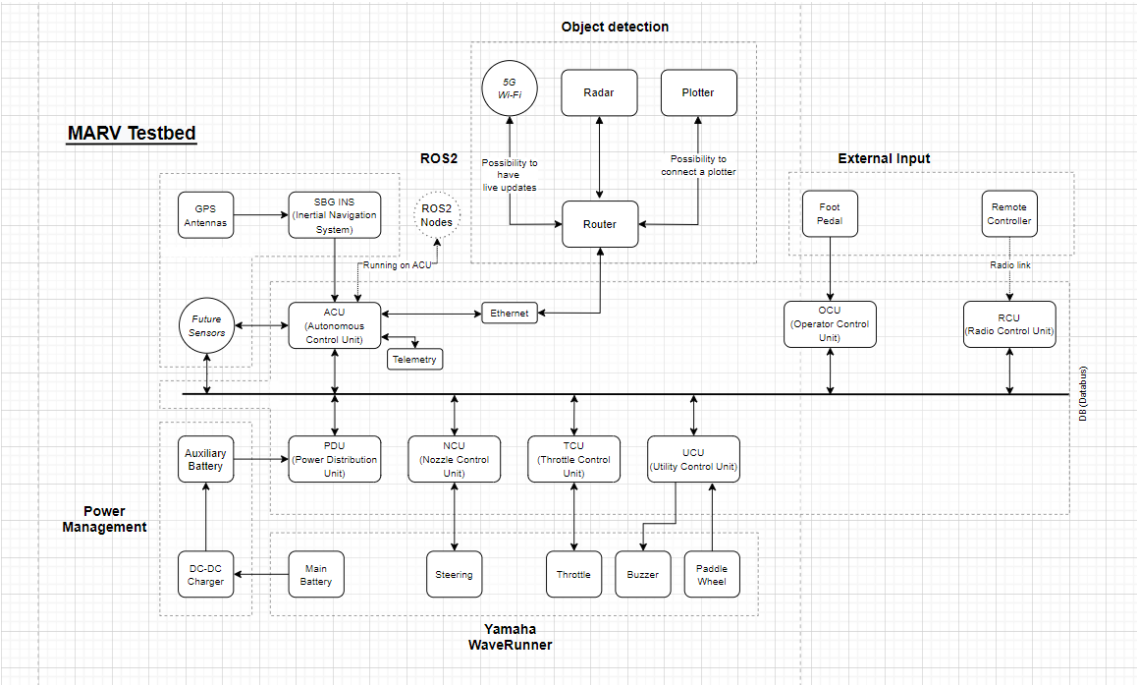


Figure 3.8: The MARV with all its components and their connections

4
Results

The following section presents the results of the radar system implementation and algorithm development. It includes the selection of radar unit, the mounting process, algorithm development, and how the algorithm performs according to the ground truth data. These results provide valuable insights into the technical solutions implemented and the effectiveness of the overall system.

4.1 Selection of radar

In table 4.1, the different radars considered along with a selction of specifications can be seen. The ranking with regards to specified critera is shown in table 4.2. The two models with lowest score, a total of 8, was B&G HALO20+ and B&G HALO24. B&G HALO20+ has a 20-inch antenna, and is thus slightly smaller and lighter. B&G HALO24 has a 24-inch antenna, which means better horizontal beam width and thus resolution. After discussing, B&G HALO24 was chosen, since a weight increase of 1 kg is negligible compared to a 20% reduction in horizontal beam width.

Table 4.1: Selected specifications of examined radars

Model	Diameter of housing [mm]	Diameter of antenna [in]	Maximum rotational speed [rpm]	Minimum range [m]	Maximum range [km]
Furuno DRS4DL+	488	19	24	25	66
Furuno DRS2D-NXT 19"	488	19	48	20	88
Furuno DRS4D-NXT	610	24	48	20	88
B&G HALO20	510	20	24	50	44
B&G Halo20+	510	20	60	50	66
B&G HALO24	610	24	60	6	88
Garmin Fantom 18"	508	17	48	6	88
Garmin xHD 18"	508	17	48	20	88
Garmin Fanton 24"	645	23	48	6	88
Garmin xHD 24"	645	23	48	20	88
Raymarine RD418D	521	18	24	-	88
Raymarine RD418HD	521	18	48	-	88
Raymarine RD424HD	652	24	48	-	88
Quantum Q24 C	541	18	24	7	44
Model	**Vertical beam width [deg]**	**Horizontal beam width [deg]**	**Power supply voltage [V]**	**Output power [W]**	**Weight [kg]**
Furuno DRS4dl+	25	5.2	12-24	-	5.7
Furuno DRS2D-NXT 19"	25	5.2	12-24	25	6.5
Furuno DRS4D-NXT	25	3.9	12-24	25	7.3
B&G HALO20	25	4.9	12-24	20	5.0
B&G Halo20+	25	4.9	12-24	25	5.9
B&G HALO24	22	3.9	12-24	25	6.9
Garmin Fantom 18"	25	5.2	12-24	25	7.7
Garmin xHD 18"	25	5.2	12-24	-	7.7
Garmin Fanton 24"	25	3.7	12-24	40	7.9
Garmin xHD 24"	25	3.7	12-24	33.5	9.5
Raymarine RD418D	25	4.9	12-24	40	9.5
Raymarine RD418HD	25	4.9	12-24	60	9.5
Raymarine RD424HD	25	3.9	12-24	60	10
Quantum Q24 C	20	4.9	12-24	20	5.6

Table 4.2: Ranking of examined radars

Model	Horizontal beam width	Ease of software implementation	Weight	Size	Rotational speed	Total points
Furuno DRS4DL+	4	2	1	1	3	11
Furuno DRS2D-NXT 19"	4	3	2	1	2	11
Furuno DRS4D-NXT	1	2	2	3	2	10
B&G HALO20	3	1	1	2	3	10
B&G HALO20+	3	1	1	2	1	8
B&G HALO24	1	1	2	3	1	8
Garmin Fantom 18"	4	2	3	2	2	13
Garmin xHD 18"	4	2	3	2	2	13
Garmin Fanton 24"	1	2	3	3	2	11
Garmin xHD 24"	1	2	4	3	2	12
Raymarine RD418D	3	2	4	2	3	14
Raymarine RD418HD	3	2	4	2	2	13
Raymarine RD424HD	1	2	4	3	2	12
Quantum Q24 C	3	2	1	2	3	11

4.2 Radar mount

The chosen approach for the design and integration of the radar mount resulted in a functional and reliable mount that effectively met the requirements of the radar system. The mount provided the necessary support and stability for the radar components, ensuring their proper functioning and accurate data collection.

4.2.1 Preparatory work

The following problem definition was formulated: design a radar mount that can withstand the harsh marine environment and be securely attached to MARV. The following subsystems where defined:
Base attachment The bracket will be mounted in the handle behind the rear/third seat (see image below). All forces will thus be brought down into the water scooter through the plastic handle and then via the four bolts attached to the fibreglass hull.
Strut (get the radar higher) For the radar to rise to the required height, some brace, pillar or similar is required to run from the base mount/mounts up to the mount for the radar antenna, cameras and electronics box.
Height adjustment A technical solution for height adjustment must be implemented for the radar antenna, cameras and electronics box to be raised and lowered within the given range.
Bracket for specific equipment Equipment specified to be placed on the mount must have its own mount that connects to the stage via the height adjustment solution.
Based on the above and the requirement specification (shown in 3.2 Construction of radar mount), the options had been narrowed down to three major concepts.
1. To use commercially available components that could be compatible with the handle of MARV. At this stage, roof racks for cars were a theoretical option.
2. Design and order a carbon fibre or fibreglass bow from a subcontractor.
3.Designing and manufacturing a solution utilizing the resources available in the CASE and at Chalmers. This included 3D printers, and laserjet, among others.

After consultations with suppliers, it was concluded that an entirely in-house developed solution best suited the project's requirements, resources and schedule. The third alternative was therefore chosen.

4.2.2 Choice of technical solution

The technical solution for constructing the radar mount involved the utilization of aluminium profiles, 3D printed ABS brackets, an acrylic mounting plate, and an aluminium mounting plate. The choice of these materials and techniques were made based on several factors, mainly due to their availability, cost-effectiveness, and adaptability to the desired specifications of the radar mount. As mentioned under the 4.2.1 Preparatory work, the mounting was initially required to be adjustable.

During the development, it was found through calculating the beam-propagating over the watercraft that a fixed height of the radar on 2.4 meters above the water level was satisfactory. So the requirement of adjustable height was dropped.

The aluminium profiles offered the required structural strength and stability, while the 3D printed ABS brackets provided flexibility in customization. All different parts were assembled with steel bolts, T-slot nuts, and angle brackets adapted for the aluminium profiles, et cetera. For the full Bill of Materials and the drawings, see appendix A.

4.2.3 Fabrication of components

The 3D printed ABS brackets were designed using CAD software, profile gauge, and then manufactured using a 3D printer. These brackets were positioned to support and secure the radar components to the aluminium frame. The process resulted in accurately formed components that allowed for efficient integration.

For the final version, an acrylic mounting plate was laser cut to securely hold the SCU of the radar system, while an aluminium mounting plate was cut with a water jet to accommodate the radar itself. These fabrication techniques resulted in accurately formed components that met the necessary specifications for successful assembly.

4.2.4 Results from testing

The whole structure remained functional and without structural failure during the final sea trial. Speeds up to 65 km/h were reached in calm to moderate sea conditions. It also handled launching and reloading from/to the trailer without issues. However, moderate oscillations of the whole structure occurred, especially when being reloaded to the trailer and transported on rough surface conditions. All 3D printed mounts managed two sea trials without structural failure. However, minor cracks occurred in several of the parts. On the first sea trial, the PMMA mounting plate suffered a structural failure in the front left corner close to the mounting holes. In the first sea trial, the PMMA plate had been changed to an identical but in aluminium which made it through the second trial without any failure tendencies.

4.3 Radar interface

A radar interface was coded and implemented in the current ROS2 structure. It has been tested rigorously both by peer readings of the code and live test with the platform on the sea. The data produced and published is verified to be accurate, but the main limiting factor has been performance. It was later realised that Python3 might not be the best language to use when implementing a high data throughput ROS2 node. With hard time limits caused by the rotation speed of the radar, it had to perform in a certain time window. With that said, the collection of data was concluded for the other nodes to utilise, but not in a stable and consistent manner.

The radar parameter configurator was also an issue. Occasionally when the gain was adjusted for example to 50% the radar interpreted that as 0.5 %. The correct way of sending configuration updated was hard to determine due to the non open-source nature of the radar sensor.

4.4 Algorithm Construction

In the following section, the algorithm constructed to distinguish, track, and classify objects from raw radar detections is outlined. The algorithm can be divided into three main modules: processing of the raw radar data, clustering, and object tracking. This schematic is clarified in 4.1, and a more in-depth explanation of the different modules follows below.

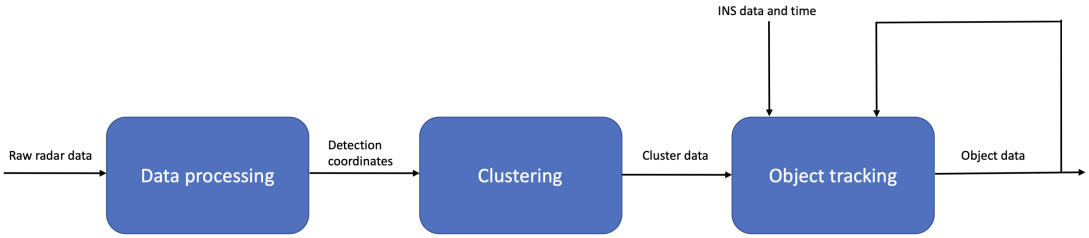


Figure 4.1: Algorithm schematic

4.4.1 Data processing

The algorithm takes input from one frame at a time, meaning that the size of the input is a 2048x512 matrix, or just above one million numbers. This is way to many data points to efficiently cluster. The data points are represented in spherical coordinates, with the angle being $\frac{\text{spoke number}}{360°}$, and the radius being $\frac{\text{number index}}{\text{radar range}}$, while the clustering algorithm needs the data in Cartesian coordinates. Furthermore, only detections within the range of 50-300m are of interest. The goal of the data processing is thus:

1. Filter out all non-detections, and any detections that lie outside the 50-300m range
2. Convert the remaining points to Cartesian coordinates

The pseudocode for the data processing module can be seen in listing 4.1. The output of the dataprocessing is then fed directly into the clustering part of the algorithm.

```
INPUT: spoke_list
OUTPUT: output
ALGORITHM: data_processing

    for spoke in spoke_list
        for point in spoke
            if point intensity > intensity_threshold
            and point in range(50,300)
                convert point to Cartesian coordinates
                save point to output
```

Listing 4.1: Pseudocode for data processing

4.4.2 Clustering

The clustering module receives the filtered list of detections and clusters them using the DBSCAN algorithm. It then calculates the number of points, x- and y-position of the center of the cluster, and a bounding box constraining the cluster. It also calculates the doppler majority of all points in the cluster. The x- and y-position of the cluster center is calculated as the average of all points of the cluster. The doppler majority is set to the doppler value that is most common among the points in the cluster. This information about the newly detected objects from the radar data is forwarded to the tracking module.

4.4.3 Object tracking

In the following section, old objects refer to objects that the tracking module is already tracking and thus has information about. New objects refer to the objects that the clustering module has found in the last frame. The new objects are the input to the tracking module, along with the tracked objects from the previous iteration.

To make tracking easier, a map of all tracked objects is maintained. This map has a constant orientation, with the y-axis always aligned with true north. This means that longitudinal travel is along the x-axis, and latitudinal travel is along the y-axis. The vessel is always located at the origin of the map. To properly maintain the map, the tracking module takes data from the Inertial Navigation System (INS) as input, more specifically a triple containing the change in longitude since last iteration, the change in latitude since last iteration, and the vessel heading. Lastly, the elapsed time between the end of the last tracking iteration and the beginning of the current iteration is input to the algorithm to be able to accurately calculate object velocities.

The main idea of the tracking module is to match previously detected objects with newly detected objects sent from the clustering module. This is done by using a score implementation of M/N-logic. Every tracked object starts out with a score

of zero and status "Tentative". When the clustering module sends the information about the new objects, the algorithm tries to match the old objects to the new objects. If a match is found, the old object is updated using the new objects information, and its score is increased. If no match is found for an object, its score is decreased. If an objects score surpasses a constant called "detection_threshold", the status is updated from "Tentative" to "Confirmed". Inversely, if an object is not matched and the score decreases more than a constant called "score_threshold" from its maximum score, the object is deemed lost and is removed from the tracked objects.

To properly match old objects to new ones, the positions of the old objects need to be estimated. This is done using the estimated velocity and heading of each object. The velocity and heading are estimated when a match is made. The object velocity is estimated as

(Distance traveled since last match) / (time since last match)

The heading is simply the angle of the vector between the last seen position and the new position. The heading and velocity of an object are set as the average of the last five "true" velocities and headings, since the "true" values are very noisy. Using the averaged values, the estimated position for each object is calculated using dead reckoning. Since objects with no previous match have no estimated velocity and heading, their estimated position is set to the last seen position. The estimated positions are shifted according to the INS data received, to always keep the vessel in the center of the internal map.

When matching new objects to old objects, three criteria are used: Size, position, and, if enabled, doppler. Objects match in size if their number of detections differs by at most 30%. They match in position if the new objects center point falls inside the estimated bounding box of the old object or, if the object is new and has no recorded velocity, the object is within 25 meters of the bounding box. To check the doppler criteria, the angle between two vectors is calculated, more specifically the angle between the vector from the origin and the new object's position, and the heading vector of the old object. The doppler criteria are true if any of the following statements are true:

- the velocity of the old object is under a minimum velocity threshold and the new objects doppler says it is standing still
- the angle is less than 75° and the new objects doppler says it is approaching
- the angle is in the range [75°,105°] and the new objects doppler says it is standing still
- the angle is greater than 105° and the new objects doppler says it is moving away

For an old object to be matched with a new object, all three criteria need to be true. Before matching, the old objects are sorted according to their status, with confirmed objects coming before tentative objects. This is to make sure that new tentative objects do not "steal a match" from any confirmed objects.

After matching, updating statuses, and removing any deprecated objects, each object is classified. Three classes are distinguished: 0 (undefined), 1 (stationary), and 2 (moving). The objects are assigned a class by the following logic:

- Class 2: Objects with a velocity greater than 1 m/s and where the historical headings kept do not differ more than 180° from the one before are considered moving.
- Class 1: Objects with a velocity lower than 1 m/s, or with a higher velocity but inconsistent headings, are considered stationary.
- Class 0: Any object with status "Tentative" is considered undefined.

At the end of each iteration, new objects that did not match with any old objects are added to the tracked objects. Pseudocode for the tracking module can be seen in listing 4.2.

```
INPUT: tracked_objects, new_objects, INS_data, time_diff
OUTPUT: tracked_objects
ALGORITHM: track

    update estimated positions for all tracked_objects
    rotate new_objects to align y-axis and true north
    sort tracked_objects on status

    for each old_object in tracked_objects:
        for each new_object in new_objects:
            if match(old_object, new_object):
                update old_object
                delete new_object
                break

    decrease score for all non-matched objects

    for each object in tracked_objects:
        if (score - max_score) < score_threshold:
            delete object
        else if score > detection_threshold:
            set status as confirmed

    Classify all objects

    for each object in new_objects:
        add object to tracked_objects
```

Listing 4.2: Pseudocode for object tracking

4.5 Verifying performance

The following section describes the results when verifying the algorithm both on real data acquired during testing, and simulated data that was manufactured. To properly understand the the output from the algorithm, some explanation of the visualization tool built into the algorithm needs explanation. In figure 4.2 the visualization tool is taken from the test data. The visualization plots a snapshot of the internal map that the algorithm maintains, meaning that true north is always aligned with the y-axis and MARV is located in origo, and marked by a small red dot. The small blue dots are the radar detections. Each object discerned by the clustering module is marked with a bigger point on the cluster center. A red marker means that the objects status is tentative, and thus its' classification is undefined. A yellow or green marker corresponds to an object with status confirmed, with yellow corresponding to confirmed objects classified as stationary, and green with confirmed objects classified as moving. For confirmed objects, the bounding box is plotted. For moving objects, the heading is indicated by a small line, where the velocity is shown at the end.

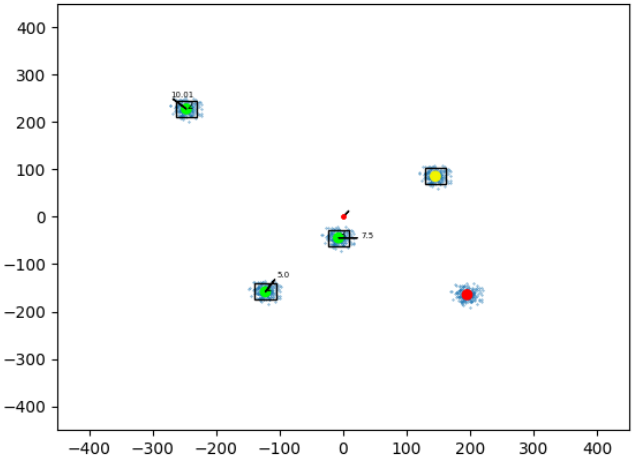


Figure 4.2: A sample of the visualization, taken from the simulated data

4.5.1 Real data

Real data was collected in the surroundings of Öckerö Båtsällskaps harbour for small boats, in the Gothenburg archipelago. An outline of the route driven can be seen in 4.3. The drive took just under 12 minutes, during which 265 iterations of the tracking algorithm was performed. Under optimal circumstances, the algorithm should be executed once every second, which during this run would mean about 700 times. This means that etiher the radar data is not parsed fast enough and the algorithm is starved from input, or that the algorithm is running too slow.

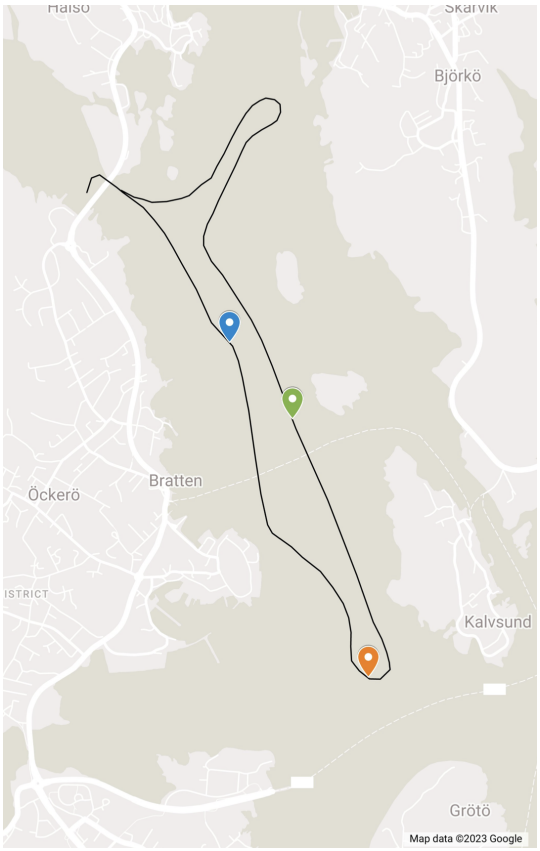


Figure 4.3: Test route driven in Öckerö

Figure 4.3 also contains different colored markers, each corresponding a visualization from the algorithm. Figure 4.4 shows that even though both Öckerö to the left and the two smaller islands to the right are discernible from the radar detections, the algorithm fails to identify them as stationary objects. The algorithm visualization from the green marker in 4.5 show that when driving further from bigger landmass of Öckerö the radar gives a more coherent picture which results in better output, but there are still a lot of smaller tentative objects present among the three smaller islands.

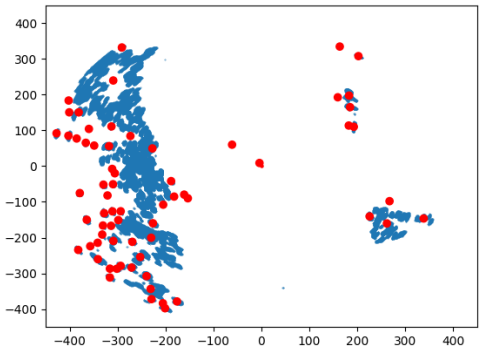


Figure 4.4: Algorithm visualization from blue marker

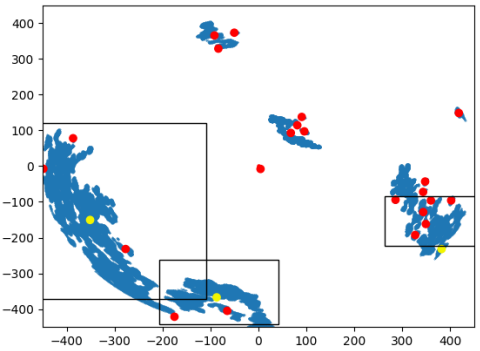


Figure 4.5: Algorithm visualization from green marker

Figure 4.6 shows the algorithm visualization from the orange marker. None of the nearby landmasses are identified as stationary objects, but one is classified as a confirmed object moving away from MARV with a velocity of 18 m/s. A tentative cluster is also clearly present about 100 meters directly east of MARV, which does not seem to correspond to any landmasses and thus could be a boat. When compared to the ground truth from the camera in figure 4.7, it is apparent that the algorithm has failed to identify the ferry from Öckerö to the mainland as a moving object.

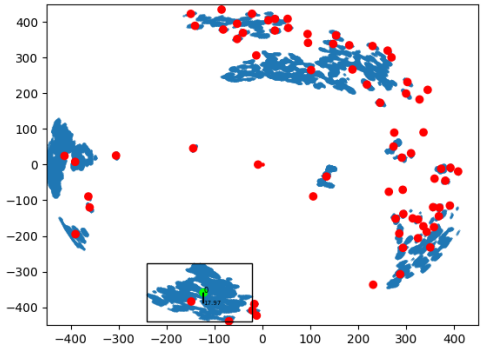


Figure 4.6: Algorithm visualization from orange marker

Figure 4.7: Ground truth from GoPro mounted on MARV. There seems to be a ferry ahead

4.5.2 Simulated data

In order to verify the algorithms performance; it was tested on a simulated scenario shown below.

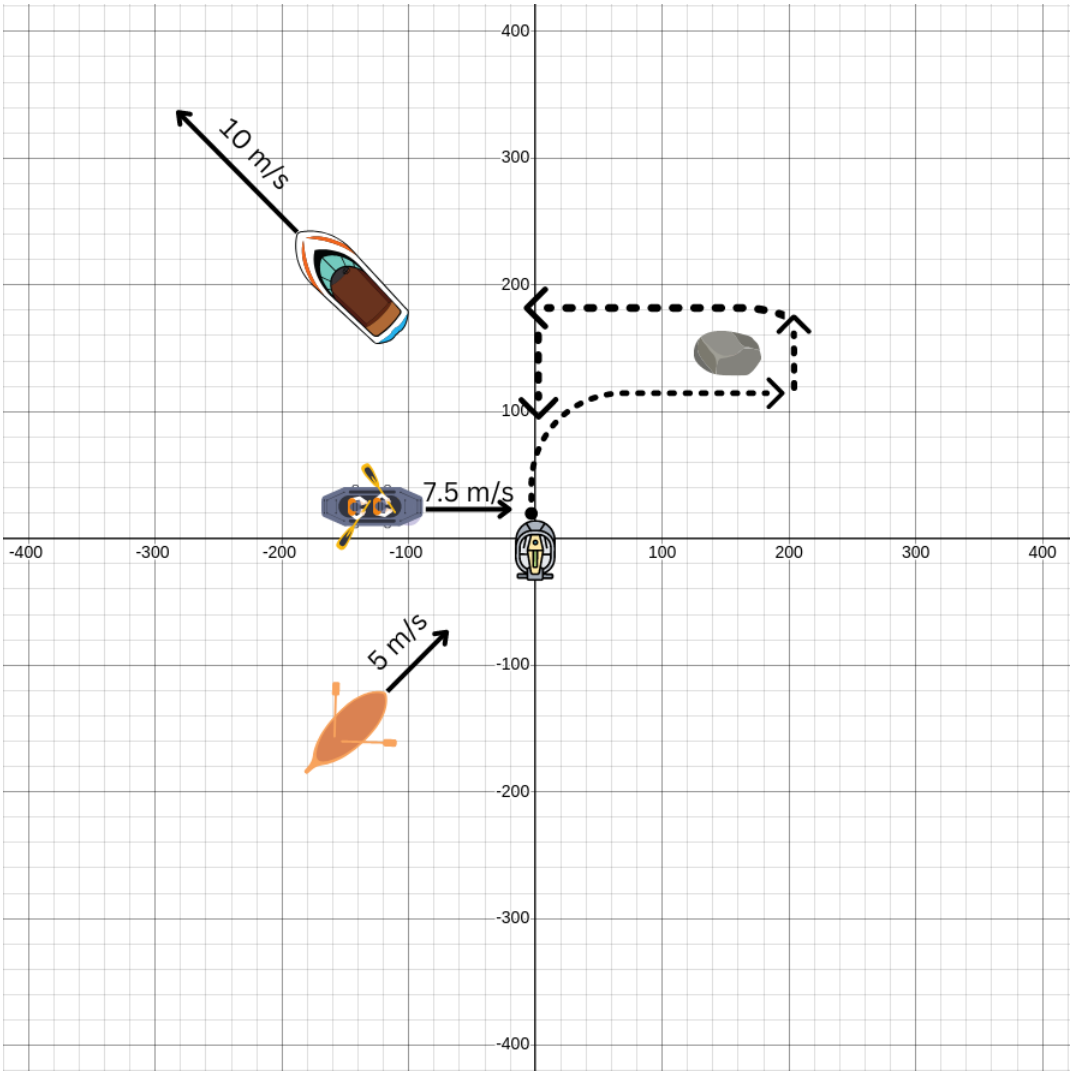


Figure 4.8: Simulated scenario

Figure 4.8 describes the initial state of the simulated scenario, with MARV moving along the dotted arrows. Each algorithm visualization in figure 4.9 and corresponding radar output in figure 4.10 corresponds to the end of an arrow. As can be seen, the algorithm performs perfectly in the simulated data, with both correct tracking, classification, and velocity/heading for every object.

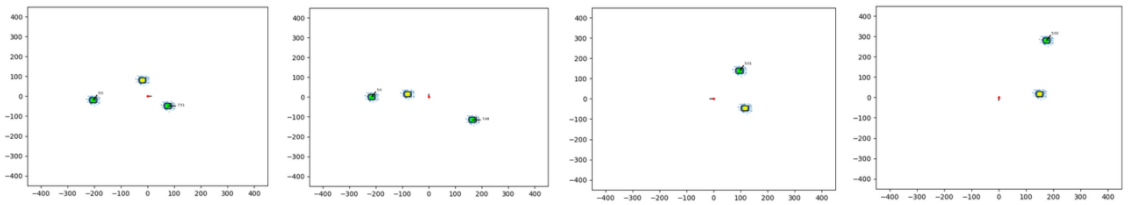


Figure 4.9: Algorithm Output

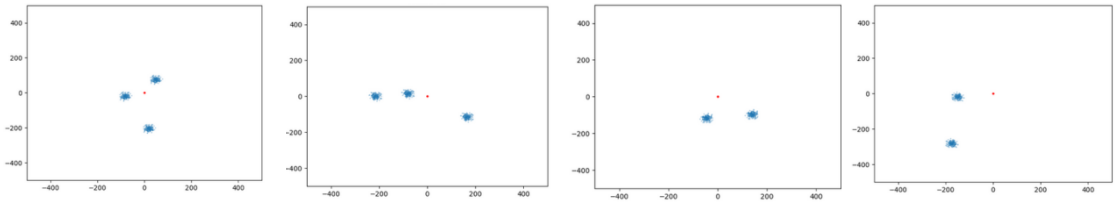


Figure 4.10: Radar output

5
Discussion

In the following section, our results, their implications, and potential improvements will be discussed. Further development of MARV will also be brought up.

5.1 Data protocol

The sole reason the UDP data protocol was used is simply due to the radar not being able to publish data on any other protocol. Using another protocol would ensure more reliability in terms of packages being lost.

5.2 Radar interface improvements

As can be concluded from the results, the performance of the radar interface was varying. Several different approaches was tried when it came to packet the data that was going to published in the /marv_radar_interface node. Publishing 1, 32, or all 2048 spokes at a time was tried but the main performance problem was still not solved. The conclusion is that a ROS 2 node running Python3 is too slow to being able to process more than 1Mb of data per second. It is therefore recommended in upcoming works to write the radar interface in C++ or a similar performance oriented language. This will most probably solve the performance issues experienced with a Python3 node.

In addition further verification of radar parameter setting should be done to ensure that the correct values of gain, rain and so on are sent to the radar sensor.

Another interesting usage of the radar is the ability to use monitor two ranges at once. In an autonomous vehicle like the MARV aspires to be, this kind of multiple monitoring could help in future path planning where both near and far objects are of interest. Monitoring two ranges at once can give you detections in the long range setting without sacrificing the good resolution of the short range detections.

5.3 Algorithm improvements

As the results suggests, several improvements of the algorithm are needed to achieve the original goal of continuous object tracking on MARV. The primary ones are discussed here.

Since the radar used has a frequency of 1 Hz, the algorithm must be able to perform one iteration in under one second, to not lag behind the input. Longer time between iterations means more movement of both the MARV and the tracked objects, both of which are potential pitfalls. Longer iteration time also increases the risk that the radar data that is used for tracking has become outdated when it reaches the tracking part of the algorithm, since the INS data supplied to the algorithm updates much faster than 1 Hz. The speed problem could be solved in three ways: upgrading the hardware, further optimizing the algorithm, or rewriting the algorithm in a faster programming language. The algorithm is currently implemented in Python, except for the DBSCAN clustering which is called from a C API. Python is notoriously slow, a better choice may be to use C++, since it is also possible to write ROS2 nodes in C++. While the algorithm certainly could further optimized, a lot of it is already vectorized using NumPy, and the risk is that simply optimizing further would not be enough. The best choice for improving speed is probably to both upgrade the hardware and to implement the algorithm in C++.

After improving the algorithm speed, the first thing that should be done is to run further tests. A lot of time was spent on optimizing the speed of the algorithm and ROS2 parts of the system, which resulted in that the algorithm only was tested live on MARV in the archipelago on two occasions. Due to software related bugs, only the last run of the second occasion resulted in any meaningful data to test on. To understand more in depth how the algorithm performs, more testing is needed. The doppler aspects of the algorithm is also not tested at all, since it was deemed more important to get the the radar detections working properly before introducing more complicated input data.

The algorithm also has a lot of parameters built into it. The DBSCAN algorithm uses both an epsilon and a min_pts parameter. The tracking module has detection_threshold for deciding when an object is confirmed, a score_threshold for deciding when to delete deprecated objects, and several implicit parameters used when matching, just to mention a few. Little work has been done to choose optimal values for these parameters, which means that there could be a lot of accuracy to be gained by further tuning the parameters. To properly tune the parameters, more and better data also needs to be gathered.

Several parts of the algorithm could also be improved. The classification of objects currently only looks at an objects current velocity and heading history. There is certainly room to also look at other parameters when classifying objects. One inherent flaw in the algorithm is the placement of the center of each object, which is just the average x- and y-position. Since the radar data is highly variable, the center of the objects tend to jump around from iteration to iteration, which in turn causes the object headings and velocities to vary more than desired. An attempt to solve this was to use a historical average for heading and velocity of objects, but better would probably be to look at the root of the problem and use a more sophisticated method of placing object centers.

5.4 Camera

The camera originally considered; and also tested is a camera of model AXIS M1124-E. This camera provides phenomenal video quality as well as IP66 rating [28]. This is theoretically a great camera to use. One big problem which was learnt the hard way was its compatibility issues with the MARV-platform and industry standard streaming protocols. It utilizes a proprietary streaming protocol called AXRTSP not available on Linux without emulating AXIS software only available for windows. Installing this would even further put strain on the computer already operating on high capacity.

5.5 Further development of MARV

The MARV platform and its auxiliary components can possibly be further improved in potential new renditions of the project.

5.5.1 Docker

Docker is platform for software containerization widely used within software development. Docker ensures consistency, robustness, portability, and reproducibility unlike anything else.

This ensures software running on one machine; runs everywhere. Situations where software will not run on a certain machine due to a lack of unknown dependencies, will not arise.

5.5.2 Ground truth data

In order to properly verify and continuously improve algorithm performance; a proper time-synced ground truth sensor is needed. The current GoPro-oriented solution works with a bit of post-time synchronizing but is far away from optimal. Better off would be to incorporate multiple cameras (ensuring 360 degrees field of view) in ROS2, eliminating the potential loss of information. This would allow for perfectly synchronized sensors.

5.5.3 Radar mount development and future possibilities

The bracket components of the radar mount underwent iterative design and were fabricated using 3D printing technology. The testing process of the brackets involved assessing their strength based on human perception and sea trials. Although this approach provided valuable insights, a more formal and rigorous evaluation is necessary to ensure optimal performance over long periods of time and resistance against fatigue.

While the designed and integrated radar mount serves as a temporary solution,

further development is possible and necessary to enhance the accuracy and functionality of the system. The current solution has demonstrated sufficient stability and effectiveness in the data collection and gathering process. However, improvements are required to establish a permanent, more robust solution and possible industrialization and implementation in the Rescue Runner fleet.

Considering the functionality and reliability of the current Rescue Runner (RR) mounts [29], they serve as a suitable model for the permanent solution. Nevertheless, modifications are required to accommodate the weight and size of the radar system on the RR mount. Additionally, the permanent mount must adhere to International Organization for Standardization (ISO) standards, necessitating professional installation, quality during production and especially the ability to withstand the durability and environmental requirements that the installation requires.

6
Conclusion

The various aims of the project where all addressed and fulfilled; although with varying degree of perfection. The radar mount developed in this project enables testing of the radar system under actual circumstances without structural failure. The mount is a modular system, and many parts are made with adaptive manufacturing techniques, making it easy to upgrade individual parts and provide replacement parts. Due to this, the radar mount can be used in future research projects before an industrialised solution is developed. In the industrialisation of a solution, it is crucial to consider that the rigidity of the fibreglass chassis has to be utilised.

The radar was integrated on the existing MARV-platform. This allows for control of the power settings, collection of radar data, as well as controlling various parameters such as gain, range, sensitivity to rain, and toggling doppler mode to list a a few. An algorithm capable of determining whether or not an object is static or dynamic was developed. In the case of a dynamic object, the algorithm calculates the velocity and heading of the object.

Although coming a long way; a functioning real-time system was not achieved. This is due to multiple reason, but it can be boiled down to: Not powerful enough hardware, A sub optimized algorithm, and lastly usage of Python3 instead of a faster programming language like C++. Furthermore, in order to verify what exactly needs to be improved, more testing is required. The setup is far from perfect yet not miles away from operational. The theory is sound and the system in its entirety works given a batch processing approach (not real-time). In other words, the radar can be mounted, the radar can collect data, process and parse the data to later feed it to the algorithm. The algorithm can in turn process the data to later output information.

Bibliography

[1] N. Danielsson and V. Lindström. "Development and Verification of an Autonomous Personal Watercraft Testbed". M.S. thesis, Dept. Elec. Eng. Chalmers Univ. of Technology, Gothenburg, Sweden, 2021.

[2] E. Ingemarsson et al. "Autonomous WaveRunner Follow the Leader". B.S thesis, Dept. Elec. Eng. Chalmers University of Technology, Gothenburg, Sweden, 2019.

[3] Open Robotics. *ROS 2 Documentation: Galactic*. https://docs.ros.org/en/galactic/. 2021.

[4] A.Bole, A.Norris, and A.Wall. "The Radar System - Technical Principles". In: *Radar and Arpa Manual: Radar, AIS and Target Tracking for marine radar users*. 3rd ed. UK: Elsevier, 2014, ch. 2, pp. 29–138.

[5] *Maritime navigation and radiocommunication equipment and systems – Shipborne radar – Performance requirements, methods of testing and required test results*. IEC 62388, International Electrotechnical Commision. 2013.

[6] Abdul. Khalique. *Navbasics : watchkeeping & electronic navigation. Vol. 3*. 2nd revised edition. Livingston: Witherby Seamanship Inter, 2011. ISBN: 185609491X.

[7] Merrill I. Skolnik. "Radar Handbook". In: 2nd ed. Section: Doppler Effect. McGraw-Hill, 1990. Chap. 14.

[8] Simrad Yachting. *HALO Dome Radar*. https://www.simrad-yachting.com/en-gb/halo-dome-radar. Section: Dual Range, Accessed: 24-May-2023.

[9] Dave Register. *OpenCPN*. https://github.com/OpenCPN/OpenCPN. 2023.

[10] Kees Verruijt. *opencpn-radar-pi*. https://github.com/opencpn-radar-pi/radar_pi. 2023.

[11] *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

[12] L. Harte. *Introduction to Data Multicasting*. Althos Publishing, 2008. ISBN: 9781932813555. URL: https://books.google.se/books?id=EUUqAAAACAAJ.

[13] The MathWorks Inc. *"Introduction to Track Logic"*. mathworks.com. Accessed: Mars 3, 2023. [Online]. Available: https://www.mathworks.com/help/fusion/ug/introduction-to-track-logic.html.

[14] *Dead Reckoning (DR)*. https://www.skybrary.aero/articles/dead-reckoning-dr. Accessed: April 2, 2023.

[15] K. Salton do Prado. *How DBSCAN works and why should we use it?* https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80. 2017.

[16] scikit-learn developers. *sklearn.cluster.DBSCAN*. https://tinyurl.com/scikitlearnDBSCAN. 2023.

[17] Yiqiu Wang, Yan Gu, and Julian Shun. "Theoretically-Efficient and Practical Parallel DBSCAN". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 2555–2571. ISBN: 9781450367356. DOI: 10.1145/3318464.3380582. URL: https://doi.org/10.1145/3318464.3380582.

[18] P. Lindstedt and J. Burenius. *The Va lue Model: How to Master Product Development and Create Unrivalled Customer Value*. Ödesborg: Nimba, 2003. ISBN: 9163063492.

[19] Lennart Hågeryd, Stefan Björklund, and Matz Lenner. *Modern produktionsteknik*. Liber, 2018. ISBN: 9789147113439.

[20] Michael F. Ashby. *Materials: Engineering, Science, Processing and Design*. 4th edition. Waltham, MA: Elsevier, 2019.

[21] International Electrotechnical Commission (IEC). *IP Ratings*. IEC website. 2021. URL: https://www.iec.ch/ip-ratings (visited on 04/11/2023).

[22] Bulgin. *Circular Data Connectors*. https://www.bulgin.com/us/products/range/circular-data-connectors.html?option_text_number_of_contacts.untouched[0]=6%20Contacts. Accessed: April 14, 2023.

[23] *Telegärtner - DataVoice - Network solutions*. https://www.telegaertner.com/en/. Accessed: April 18, 2023.

[24] Amphenol RF. *N-Type Connectors*. https://www.amphenolrf.com/connectors/n-type-connectors.html. Accessed: May 15, 2023.

[25] O. Benderius and C. Berger. *Open DLV*. https://opendlv.org/. 2023.

[26] N. Danielsson and V. Lindström. *Marv test data*. https://github.com/CASE-Lab/MARV-Test-Data. 2021.

[27] *What is GPS Time?* https://timetoolsltd.com/gps/what-is-gps-time/. 2019.

[28] *IP ratings*. https://www.iec.ch/ip-ratings. 2023.

[29] Swedish Sea Rescue Society. *Rescuerunner*. https://www.sjoraddning.se/rescuerunner. Accessed: May 15, 2023.

A

Appendix 1

Table A.1: Bill of Material for radar mount

	Part name:	**Part description:**	**Material:**	**Qty:**
1	Radar mounting plate	Radar mounting plate	Alu.	1
2	Box mounting plate	Box mounting plate	PMMA	1
3	Top left bracket	3D-printed top left bracket	ABS	1
4	Top right bracket	3D-printed top right bracket	ABS	1
5	Top middle bracket	3D-printed top middle bracket	ABS	1
6	Aluminium profile connector	3D-printed connectors	ABS	4
7	Low left bracket	3D-printed main bracket	ABS	1
8	Low left bracket plate	3D-printed bracket plate	ABS	1
9	Low right bracket	3D-printed main bracket	ABS	1
10	Low right bracket plate	3D-printed bracket plate	ABS	1
11	Low middle bracket inside	3D-printed main bracket	ABS	1
12	Low middle bracket outside	3D-printed main bracket	ABS	1
13	Right support	3D-printed support	ABS	1
14	Left support	3D-printed support	ABS	1
15	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	2x1440 mm
16	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	1x1410 mm
17	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	1x485 mm
18	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	1x480 mm
19	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	1x265 mm
20	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	2x47 mm
21	Rexroth 20x20 mm profile, 6mm groove	Aluminium profile	Alu.	1x43 mm
22	Rexroth 20x20 mm angle bracket	Angle bracket	Alu.	12
23	M4 x 8 mm	Hex screw, cylindrical head	Stainless steel	36
24	M4 x 10 mm	Hex screw, cylindrical head	Stainless steel	9
25	M4 x 12 mm	Hex screw, cylindrical head	Stainless steel	3
26	M4 x 14 mm	Hex screw, cylindrical head	Stainless steel	15
27	M4 x 25 mm	Hex screw, cylindrical head	Stainless steel	2
28	M4 x 30 mm	Hex screw, cylindrical head	Stainless steel	4
29	M4 x 40 mm	Hex screw, cylindrical head	Stainless steel	6
30	M6 x 50mm	Hex screw, cylindrical head	Stainless steel	2
31	M6 x 65 mm	Hex screw, cylindrical head	Stainless steel	8
32	M6 x 70 mm	Hex screw, cylindrical head	Stainless steel	2
33	M8 x 20 mm	Hex screw, cylindrical head	Stainless steel	4
34	T-nut	Nut	Stainless steel	64
35	M4 lock nut	Nut	Stainless steel	4
36	M4 nut	Nut	Stainless steel	7
37	M6 nut	Nut	Stainless steel	12
38	Round washer M4	Washer	Stainless steel	44
39	Round washer M6	Washer	Stainless steel	12
40	Spacers	Spacers	PMMA	4

Figure A.1: Drawing: Top left bracket

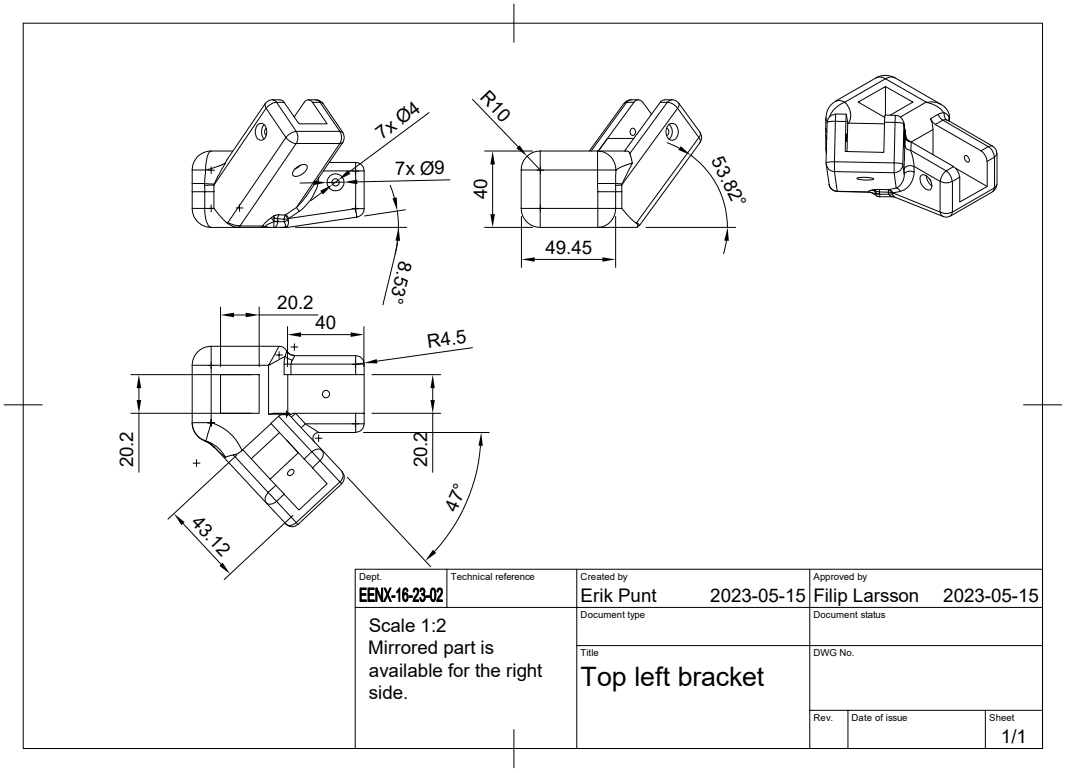


Figure A.2: Drawing: Top middle bracket

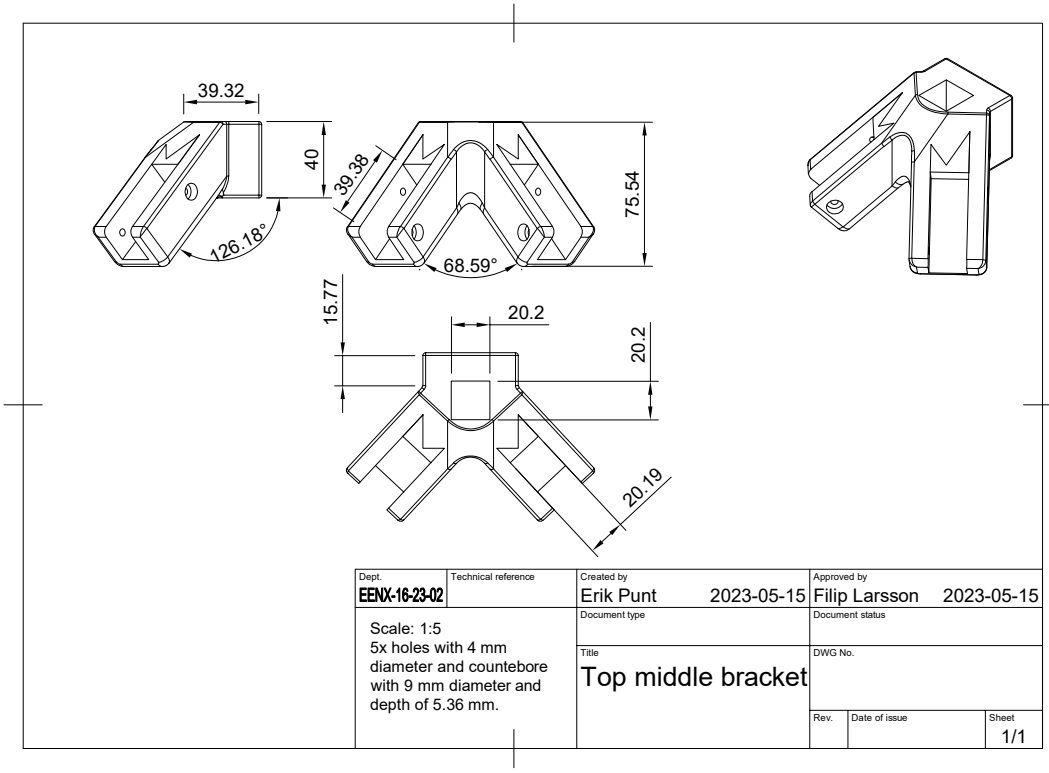


Figure A.3: Drawing: Low left bracket

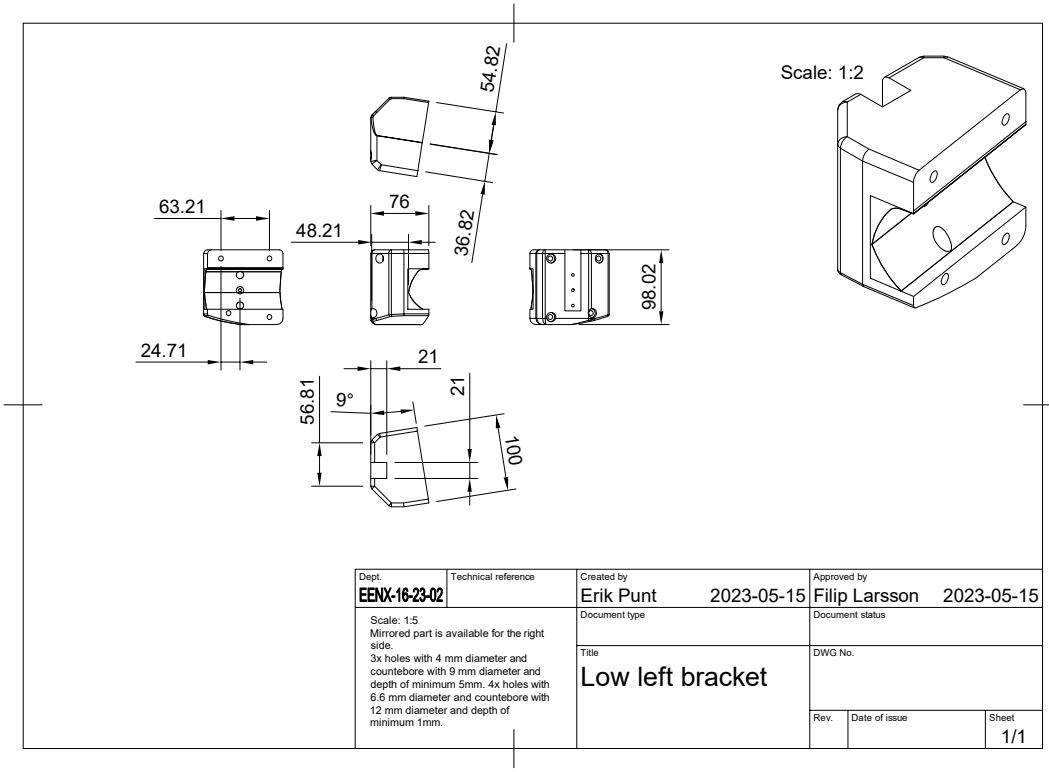


Figure A.4: Drawing: Low bracket right plate

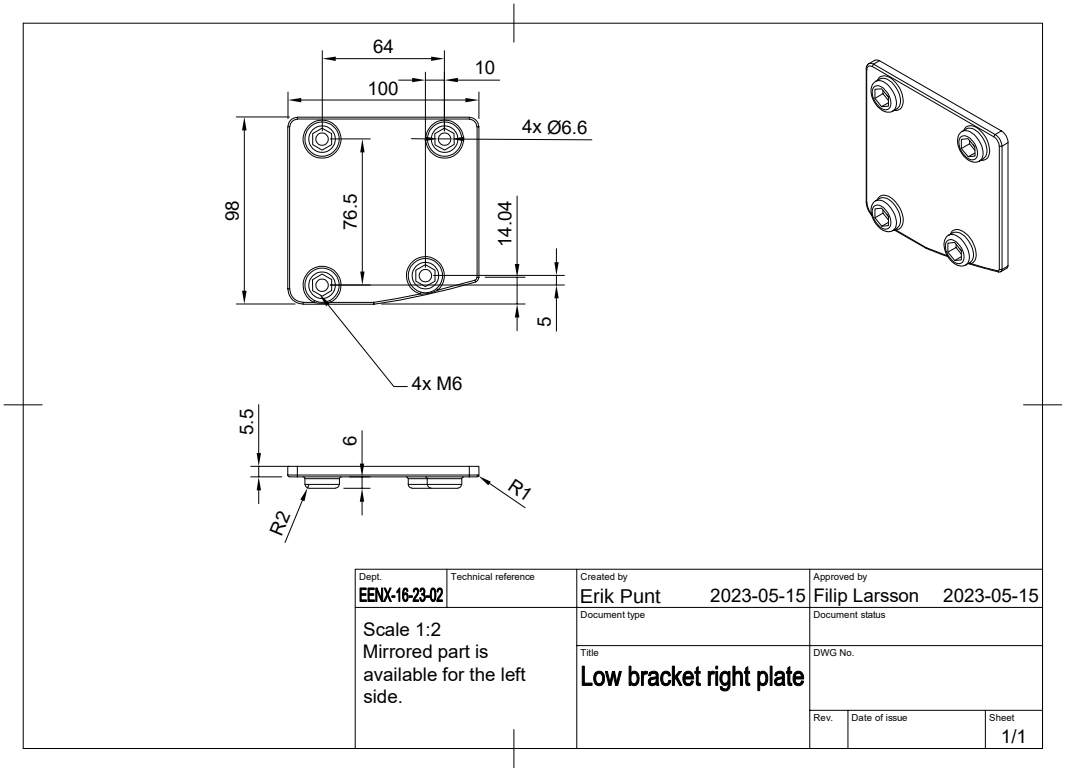


Figure A.5: Drawing: Low middle bracket (inside & outside)

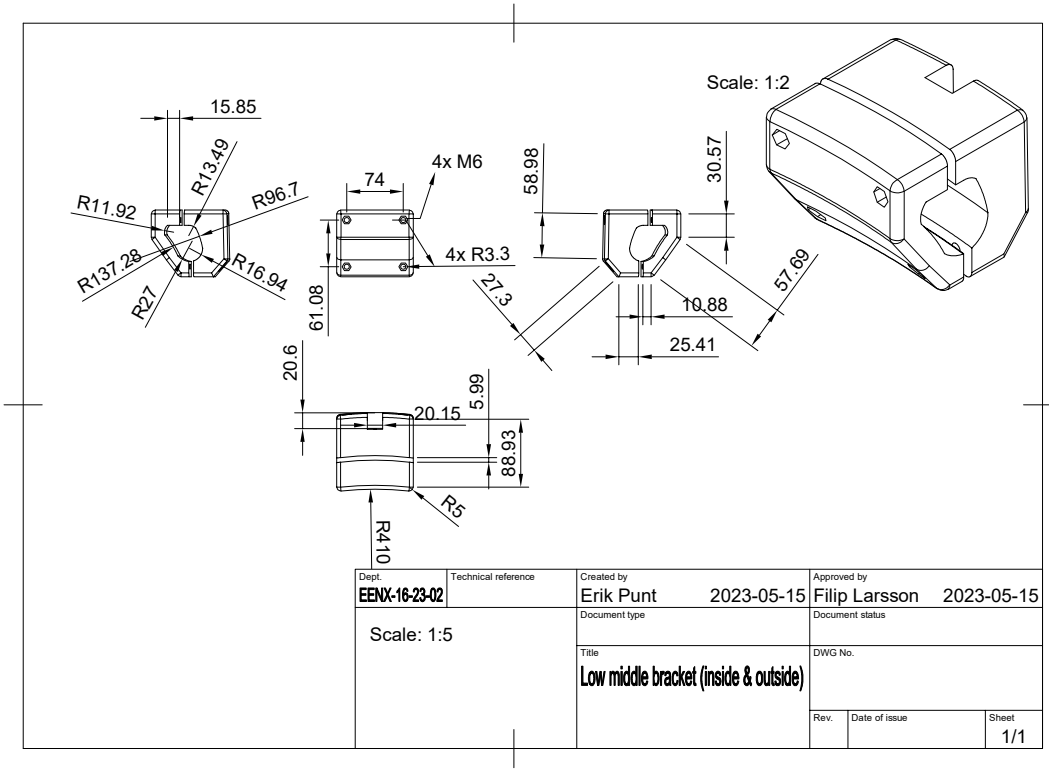


Figure A.6: Drawing: Box mounting plate

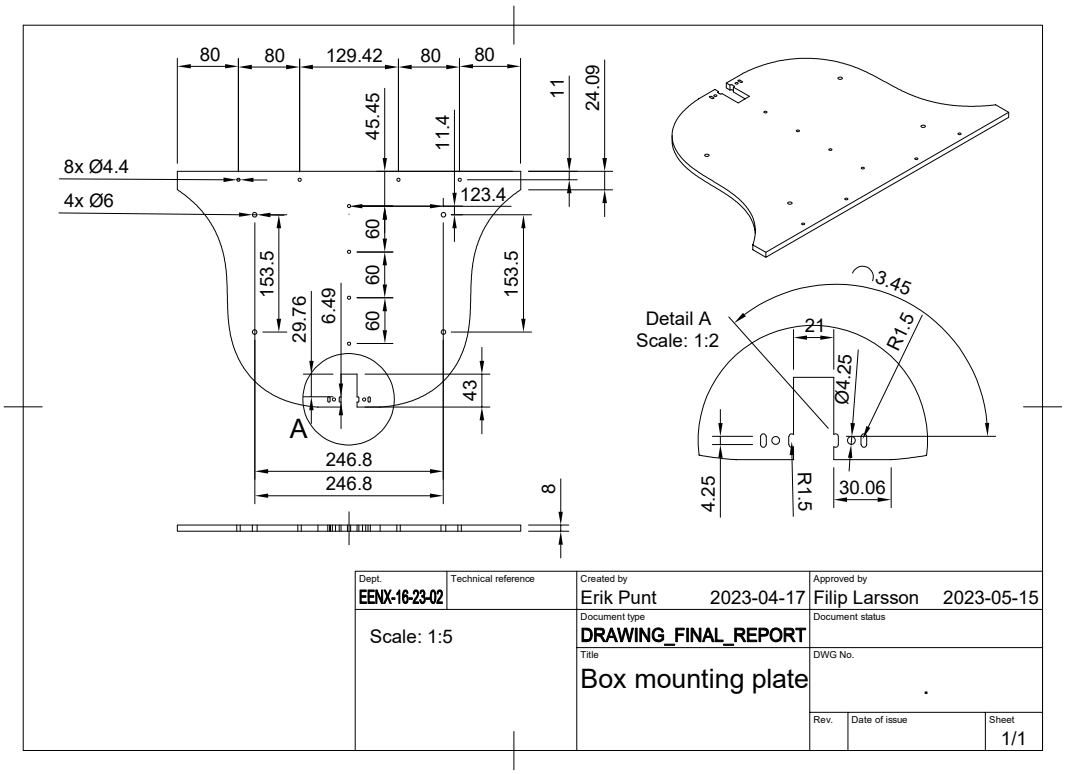


Figure A.7: Drawing: Radar mounting plate

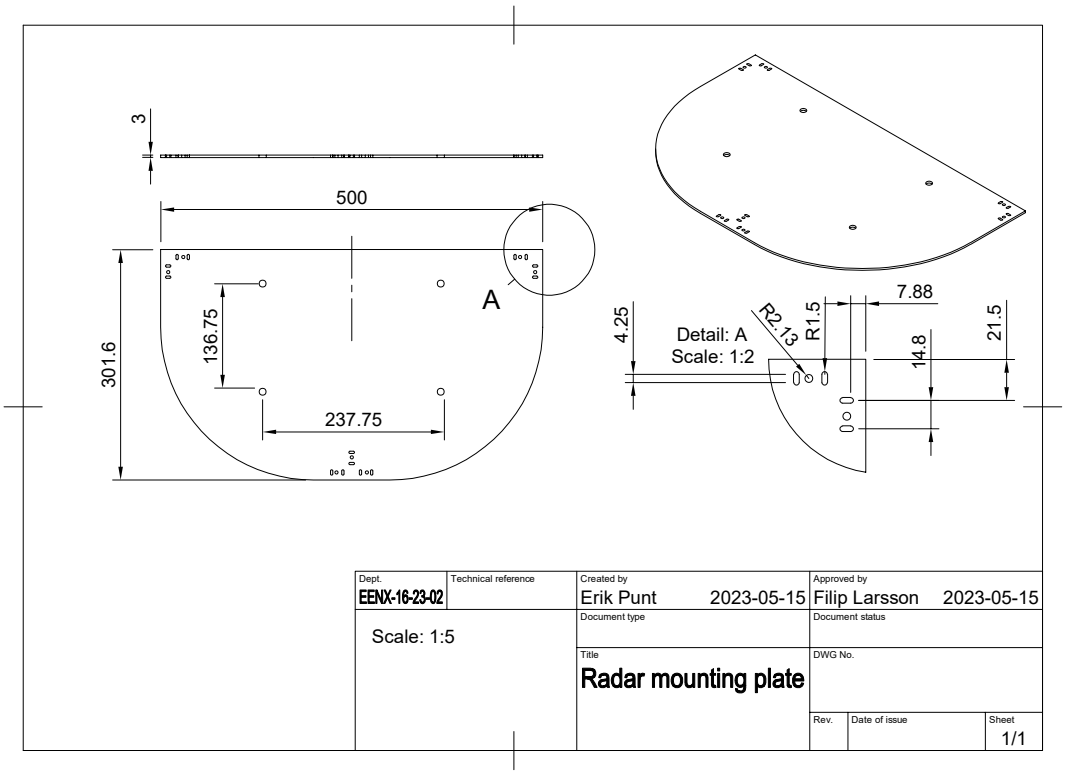


Figure A.8: Drawing: Spacers

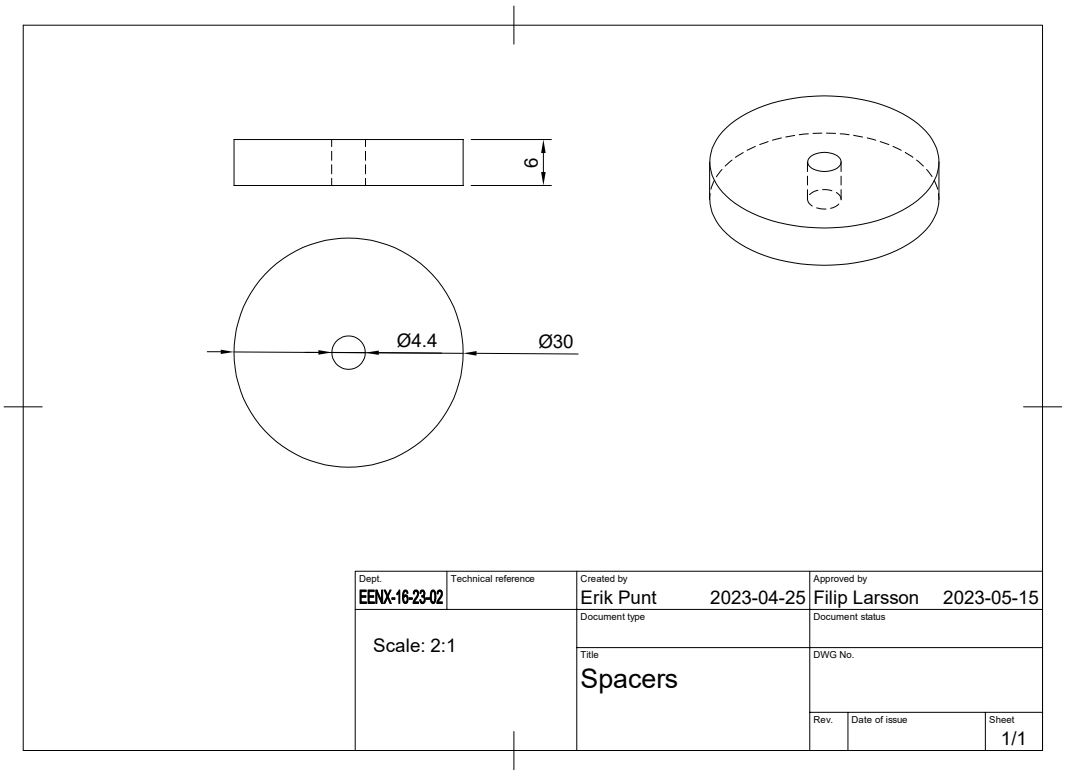


Figure A.9: Drawing: Aluminium profile connector

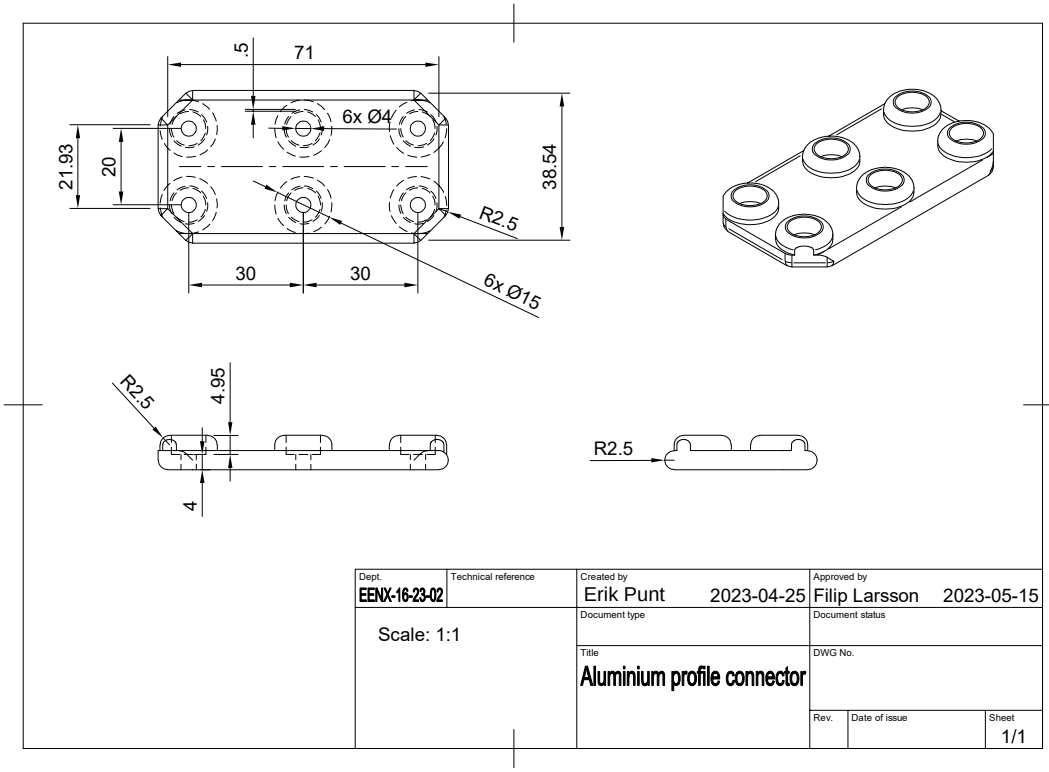
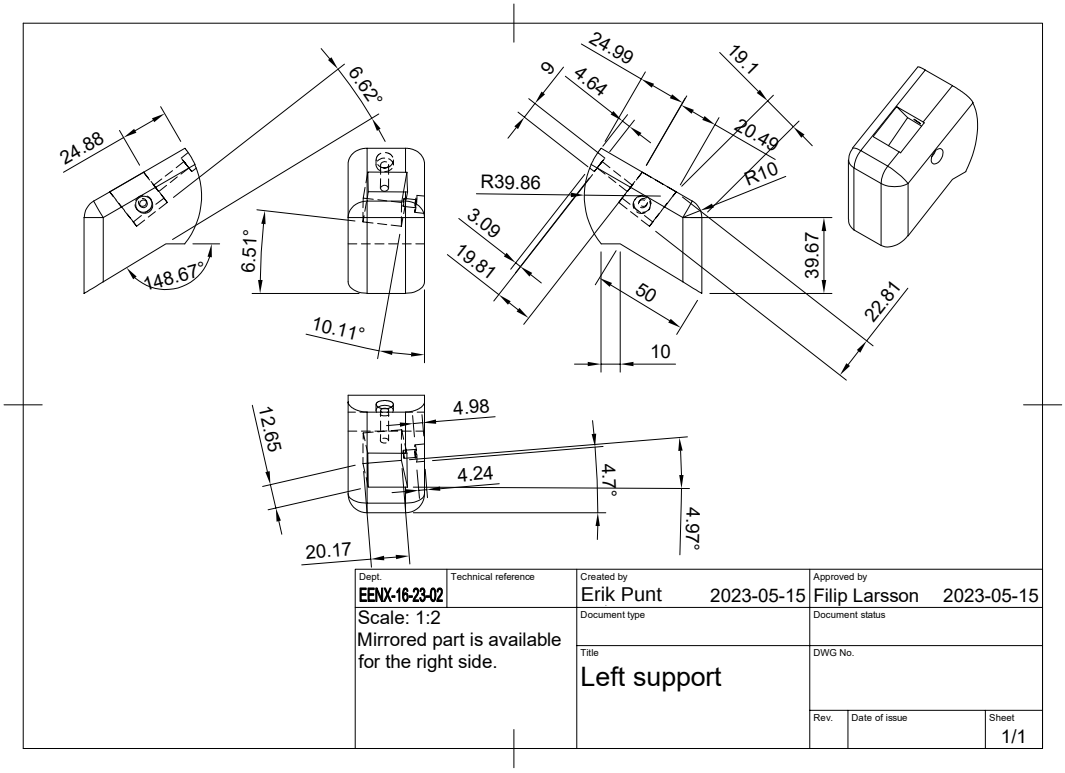


Figure A.10: Drawing: Left support



DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS