UiA University of Agder

Master

mmWave radar based contactless method for estimating parameters of swinging load: Implementation and Validation

TOR EGIL VOLL ØVERNES & DANIEL RÅMUNDSEN

SUPERVISORS
Ajit Jha, Lei Jiao

University of Agder, 2022
Faculty of Engineering and Science
Department of Engineering and Sciences
Department of Information and Communication Technology

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgement

We would like to express our gratitude to our supervisors Ajit Jha and Lei Jiao for their help and guidance throughout this project. Their support have been invaluable for us to finish this project.
We would also like to thank Jan Christian Bjerke Strandene and Philipp Thomas Pasolli for their help in the physical experiments.

Abstract

This thesis proposes a novel method to estimate the parameters of a suspended swinging load using a 77GHz millimeter wave (mmWave) radar. An experimental scale model setup was created in the lab to simulate a suspended load. The focus of this project is to achieve real-time 2D localization of the swinging load including estimation of displacement, velocity and angle. For this purpose, an AWR1843 mmWave radar was used to acquire two-dimensional point cloud data. This data was compared to two reference measurement devices. An inertial measurement unit (IMU) provides velocity and angle data and an IWR6843 mmWave radar programmed to accurately measure range which is interpreted as x-position in this thesis. A system was implemented using Robot Operating System (ROS) for data acquisition and visualization. The visualization features a simulation of the real world experiment which mirrors the physical system in real-time by using the processed point cloud data from the measurement radar, as a step toward a digital twin. The proposed method has a root mean square error for displacement of 3.0 [cm], 0.07 [rad] for angle and 0.127 $[\frac{m}{s}]$ for velocity compared to the reference sensors. On the basis of these results, it is concluded that mmWave radars may be applicable to determine parameters of physical systems.

Keywords: millimeter wave radar, point cloud processing, swinging load, robot operating system

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increased demand for automation in industrial environment calls for large variety of sensors, controllers and actuators to accomplish the given tasks. In many cases it is desirable to measure the related parameters e.g. distance, velocity, position and angular motion as accurately as possible with minimal error. There are various methods to measure these parameters. Typically piezoelectric, capacitive, null-balance, strain gauge, accelerometer sensors are used to detect the vibrations [1]. However they suffer from electrical interference, are contact based, cause loading effect, and cannot always be placed at the desired location. Further with the rise in Industry 4.0 and making digital twin representations of physical systems, it is desirable to measure the related parameters accurately. A digital twin of a physical system require that the measurement from the real world be integrated in the corresponding digital model, at the same time it can validate the real world measurement based on the model. This two way flow of data and information from model to real world thus help make the model synchronous to the physical system. This require novel sensing techniques that measure the physical parameters. Non-contact sensing makes it possible to acquire information about an object without having direct contact with that object. This can be applicable in a variety of different situations. Non-contact sensing is when a sensor emits its own energy to scan objects by measuring the energy that is reflected back to the sensor. Examples of this is radar (Radio detection and ranging) and LIDAR (Laser imaging, detection, and ranging) technology. At at fundamental level, these sensors measure the time delay between emitting and receiving a signal in order to determine position, velocity and angle of an object. In the context of this thesis, the focus will be on non-contact sensing with the use of millimeter wave radars (mmWave radar).

Radars allows for data gathering in dangerous or inaccessible environments. The area of application for radars is varied. Aerial traffic control, collection of meteorological data and advanced driver-assistance systems are a few examples of what applications radars are utilized for. Radar has found its application in distance measurement [2] and local positioning [3]. The added advantages that mmWave radar brings is the integration of transmitter and receiver on the same board, thus making the setup compact and robust, without the need to align the transmitter and receiver. In addition, the the electromagnetic radiation emitted by the radar are in millimeter wavelength range that can easily penetrate fog and can equally be used under limited conditions [4].

There are multiple cases where this technology could be applied to potentially create more efficient processes. An example of this is in offshore industries where crane vessels have to handle a load. Due to these ships operating in offshore conditions, where they are subjected to waves and wind, the load being handled will start swinging. In a case like this it could be advantageous to accurately acquire the load's parameters, which in turn can be used in developing a control system solution to stabilize the load during operation.

1.1 Research Questions

This thesis proposes using mmWave radar technology in order to accurately obtain parameters of a swinging load. This leads to the following research questions:

- Which parameters of a swinging load can be determined directly by a mmWave radar sensor and which can be further derived by processing the raw data?
- How accurately can these parameters be determined?
- How can these parameters be efficiently acquired and further digitally represent the physical system in real-time?

Experiments will be performed to see whether or not this technology is viable for operations where other sensors performance are less than optimal. In this case the sensors will measure a suspended object. When subjected to forces the object will start to move, and the goal is for the sensors to identify the position, velocity and angle of the object.

1.2 Objective and contribution

The objective of this thesis is to research the viability of using mmWave radar technology to accurately determining the parameters on a swinging load. In other words, using mmWave radars for acquiring point cloud data which provide measurements for displacement, velocity and angle estimation. Thereafter, displaying the data visually through the use of ROS and RViz.
The contributions made during the course of this project was as follows:

- Implementation of mmWave radars for accurate parameter estimation of a swinging load.
- Visualising the recorded data with the use of ROS.
- Validation of the parameter estimations by using other sensors as reference.

1.3 State-of-the-art

This section introduces the current state of related topics to this thesis. A broad overview is given of the field of radar technology and current techniques applied to swinging load handling. In addition to this, aspects of related work are inspected in which methods they have used and which results they have a achieved.

1.3.1 Radar

Radar detection has been around for decades. The technology applies electromagnetic waves in the radio-frequency spectrum to measure range, velocity and angle for objects. The radio-frequency spectrum ranges from about 3 [MHz] to 300 [GHz] [5], which means that the waves emitted from the radar has weak interactions with fog, rain and snow. Therefore, radar functions as an optimal sensor for detecting objects outside, even in extreme weather conditions. This makes radars widely applied in fields such as air and marine control, meteorology and automobile industry. Depending on the emitted signal form, radars can be divided into pulsed and continuous wave. Pulsed radars emit high frequency pulses within given time intervals, while continuous wave radars emit and detect continuous signals. If the emitted radar waves are periodically frequency modulated it is called a Frequency-Modulated Continuous-Wave (FMCW) radar. mmWave radar technology are radars that operate in the 76-81 [GHz] range. These radars have the advantage of being relatively small, therefore

having a small footprint. Since the introduction and hype of autonomous products both LIDAR and radar technology have made huge progress. Lapland UAS publications [6] wrote about radar and its importance to supporting autonomous driving. They state that the 76-81 [GHz] bandwidth has been established in the field. mmWave- technology is also used in other applications such as aviation and marine transport.

1.3.2 Swinging load

Swinging load handling has traditionally been reliant on human involvement for accuracy and avoiding unwanted motion of the payload [7]. The addition of remote sensing equipment could reduce the need for human decision making, allowing for a faster and more accurate process [8]. To reduce the pendular motions of a suspended payload, anti-swing control can be studied by controlling cranes and corresponding servomotors. Prior research has shown that the effect of turbulent wind on a payload's motion is limited, but the motion of the vessel is what dominantly affects the payload's motion [8]. The payload's shape, orientation and mass distribution also adds a layer of complexity when designing an automated anti-swing algorithm. This could potentially provide a safer and more reliable solution than the current state of the art approaches. Furthermore, this technology can potentially offer a much faster installation time when compared to current methods, thereby saving money and time for the manufacturers. The hypothesis is that by using radar sensors, swinging load handling will become a more efficient operation.

1.3.3 Related work

The emergence of high performance, low-cost mmWave radar and LiDAR technology has seen interest from researchers for the purpose of using them in industrial applications. This section addresses related work to this thesis within the fields of non-contact sensing and modelling of a physical system. This includes parameter estimation for similar experiments with different sensors and different approaches to processing the data. Common sensors for non-contact sensing are CCD cameras, LiDAR and radar. A notable related scenario is tracking a swinging load from a drone in flight, which multiple researchers have investigated.

Hu et al [9] proposes a method of estimating the sway angle of a swinging load in a trolley crane. This method uses an 24 GHz mmWave radar for measurement and a LiDAR as reference. The paper implements a Simulink model and a physical experiment. The physical experiment features no clear results, but the method is equivalent to using the IWR6843 reference radar as measurement device, as it only measures the range of the object.

Optical techniques include paper [10] which uses a single smart phone camera for object tracking and pose estimation. The authors achieve an average error of approximately ± 1.4 cm for position and ± 0.02 rad for angle. Camera based techniques has the advantage of cheap and available sensors but requires accurate calibration and markers.

Authors in [11] has published a feasibility study of using a LiDAR to track a load swinging from a drone in flight. This paper models the swinging load as a 3D pendulum. Their data is filtered by using k-means clustering and applied in an Extended Kalman filter alongside a nonlinear dynamic model to estimate the position of the payload. A angle estimation of $\theta = \pm 0.02$ rad was achieved using this technique.

In the field of simulating a swinging load, paper [12] implements a system in Gazebo to describe a suspended load from a quadcopter. It models the swinging load similarly to how it is implemented in this thesis by describing the system with an URDF model. This model consist of a fixed base connected to the load by a ball-and-socket joint. The paper applies a time-delayed feedback control Simulink model to the simulated system with a virtual IMU as sensor for the swinging load. Attaching an IMU to the swinging load is used in this project as a reference measurement, but in many applications this would not be an option.

Chapter 2

Theory

This chapter describes the theoretical background for the work performed in this project. The physical fundamentals of Radar technology, as well as basic principles of Robot Operating System is described.

2.1 Radar

Radar is an object-detection system capable of measuring the range, velocity and angle of an object with the use of radio waves. A radar measures these parameters by having transmitter antennas that emits electromagnetic waves, then receiver antennas capture the signals that are reflected back from objects. The radar will then process this information to determine the parameters of the objects that are detected.

There are different types of radars. These include Bistatic, Continuous-wave, Doppler, Monopulse, Passive and Instrumentation radars[13]. The differentiating properties between them is signal modulation and antenna functionality. The radar type that is utilized in this project is the Frequency-Modulated Continuous Wave (FMCW) radar. These radars emit continuous transmission signals, and can change their operating frequency while measuring. Frequency modulation provides a time reference that makes it possible to determine the distance of an object relative to the radar. The basic features of an FMCW is [14]:

- Ability to measure the distance to a target.
- Ability to simultaneously determine the velocity, angle and range of a target.
- High accuracy measurements

A complete mmWave radar system includes transmit (TX) and receive (RX) radio frequency (RF) components, analog components such as clocking, and digital components such as analog-to-digital converters (ADCs), microcontrollers (MCUs) and digital signal processors (DSPs).

2.1.1 Range Measurement

Radar systems functions by transmitting an electromagnetic signal that is reflected by objects in its path. In a FMCW radar the frequency is increased linearly with time as shown in figure 2.1. A radar transmits a signal called a "chirp", which is a sinusoid where the frequency is increased linearly with time. The radar has a synthesizer which generates the chirp, which is then transmitted from a TX-antenna. When this chirp interacts with an object, it will reflect off the object and the reflected chirp will be received by the RX-antenna. The RX and TX signals of a single object combined results in an Intermediate Frequency (IF) signal. A chirp signal is characterized by a start frequency (f_c), bandwidth (B) and duration (T_c) [15].

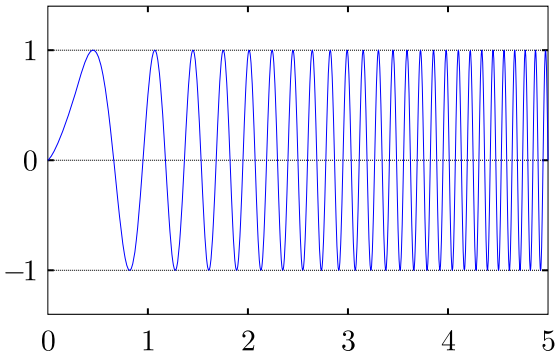

Figure 2.1: Linear chirp where frequency increases over time

The FMCW radar operates as follows:
A synthesizer generates a chirp (Figure 2.1 shows the chirp amplitude as a function of time, figure 2.3 (a) shows the chirp frequency as a function of time). The chirp is then transmitted by the TX-antenna. The chirp is reflected by an object which is captured by the RX-antenna. A "mixer" combines the TX and RX signals to produce an IF signal. This is illustrated in the block diagram in figure 2.2. The frequency mixer is an electronic component that combines two signals to create a new signal with a new frequency [15].

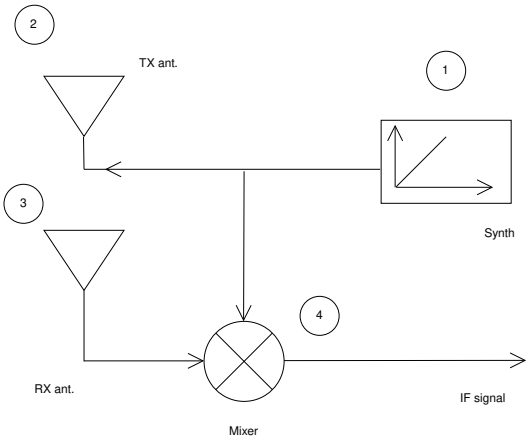

Figure 2.2: FMCW radar simplified block diagram

For two sinusoidal inputs x_1 and x_2:

$$x_1 = sin(\omega_1 t + \phi_1) \tag{2.1}$$

$$x_2 = sin(\omega_2 t + \phi_2) \tag{2.2}$$

The output x_{out} has an instantaneous frequency equal to the difference of the instantaneous frequencies of the two input sinusoids. The phase of the output x_{out} is equal to the difference of the phases of the two input signals:

$$x_{out} = sin[(\omega_1 - \omega_2)t + (\phi_1 - \phi_2)] \tag{2.3}$$

Figure 2.3 represents the TX and RX chirps as a function of time for a detected object. The RX chirp is a time-delayed version of the TX chirp. The time delay (τ_d) is defined as:

$$\tau_d = \frac{2d}{c} \tag{2.4}$$

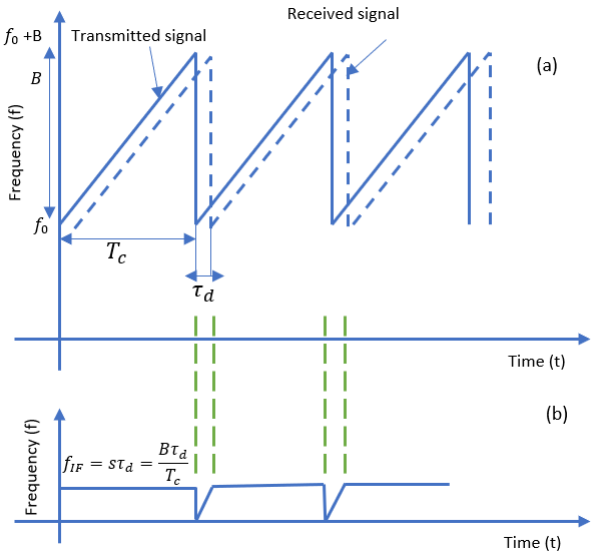

Figure 2.3: Transmitter and receiver chirps as a function of time

where:
d: distance of the detected object
c: speed of light

The distance between the two lines is fixed, meaning that the IF signal consists of a tone with a constant frequency. As shown in figure 2.3 this frequency is defined as $f_{IF} = S_\tau$. The IF signal is only valid in the time interval where the TX chirp and the RX chirp overlap.
The initial phase of the IF signal (Φ_0) is the difference between the phases of the TX and RX chirps at the time at the start of the IF signal [15].

$$\phi_0 = 2\pi f_c \tau \tag{2.5}$$

Further derived:

$$\phi_0 = \frac{4\pi d}{\lambda} \tag{2.6}$$

For an object at a distance d from the radar, the IF signal will be a sine wave:

$$Asin(2\pi f_o t + \phi_0) \tag{2.7}$$

where:
$f_0 = \frac{S2d}{c}$
$\phi_0 = \frac{4\pi d}{\lambda}$

At this point it has been assumed that the radar only detects one object. When the radar is detecting multiple objects, the receiver will receive multiple chirps, where each of the chirps are delayed by a different amount of time proportional to the distance of the corresponding object. The different RX chirps translate to multiple IF tones, which each have a constant frequency.
A Fourier transform is necessary for processing an IF-signal consisting of multiple tones in order to separate the different tones. The result of the Fourier transform processing is a frequency spectrum with separate peaks for each of the peaks that represents an object at a specific distance [15].

2.1.2 Range resolution

A radar's ability to distinguish between two or more objects is known as range resolution. When two objects are too close to each other, the radar will no longer be able to recognise them as separate objects. By increasing the length of the IF signal, the resolution can be increased. To increase the length of the IF signal, the bandwidth must also be increased proportionally, which results in an IF spectrum with two separate peaks. Fourier transform theory states that an observation window (T) can resolve frequency components that are separated by more than 1/THz. Therefore, two IF signal tones can be resolved in frequency as long as the difference in frequency satisfies the relationship given [15]:

$$\Delta f > \frac{1}{T_c} \tag{2.8}$$

where T_c is the observation interval. The range resolution (d_{res}) depends on the bandwidth swept by the chirp:

$$d_{res} = \frac{c}{2B} \tag{2.9}$$

With a chirp bandwidth of a few GHz, an FMCW radar will have a range resolution in the order of centimeters [15].

2.1.3 Velocity measurement

To measure velocity, two chirps have to be transmitted separated by a time interval T_c. The reflected chirps are then processed through FFT to detect the range of the object. The peaks of both chirps will be in the same location, but they will have different phases. The phase difference corresponds to a motion in the object. The phase difference is [15]:

$$\Delta\Phi = \frac{4\pi v T_c}{\lambda} \tag{2.10}$$

Rearranged to calculate velocity:

$$v = \frac{\lambda \Delta\Phi}{4\pi T_c} \tag{2.11}$$

Given that the phase difference determines the velocity, there will be ambiguity. The measurement is unambiguous if $|\Delta\Phi| < \pi$.
The maximum relative speed (v_{max}) measured by two chirps spaced T_c apart is:

$$v_{max} = \frac{\lambda}{4T_c} \tag{2.12}$$

If two objects are at the same distance from the radar at the time of measurement, the two-chirp velocity measurement does not work. Because the objects are at the same distance from the radar, they will generate chirps with identical IF-frequencies. Therefore the range-FFT will result in a single peak, which represents the combined signal of all the objects at equal distance [15].

2.1.4 Angle detection

An FMCW radar system can estimate the angle of a reflected signal with the horizontal plane. This angle is known as the Angle of Arrival (AoA). Angular estimation is based on the observation of a small change in the peak of the range-FFT or Doppler-FFT. At least two RX antennas is required to perform this estimation. The differential distance from an object to each of the antennas results in a change in the FFT peak. The phase change makes it possible to estimate the AoA [15].

The phase change is calculated as:

$$\Delta\Phi = \frac{2\pi\Delta d}{\lambda} \tag{2.13}$$

The AoA can therefore be expressed as:

$$\theta = sin^{-1}(\frac{\lambda\Delta\Phi}{2\pi l}) \tag{2.14}$$

where l is the distance between the antennas. The angle estimation accuracy is dependent on AoA and is more accurate when θ has a small value.
The maximum angular field of view is defined by the maximum AoA that the radar can estimate. Unambiguous measurement of angle requires $|\Delta\omega| < 180°$. By applying this to 2.14 [15]:

$$\frac{2\pi l sin(\theta)}{\lambda} > \pi \tag{2.15}$$

The maximum field of view that two antennas spaced l apart can detect is:

$$\theta_{max} = sin^{-1}(\frac{\lambda}{2l}) \tag{2.16}$$

2.2 Data Processing

2.2.1 Fast Fourier transform

Fast Fourier transform (FFT) is an algorithm for computing the discrete Fourier transform of a sequence [16]. This technique is used to convert time series data into frequency domain, which in the context of radar technology can be applied to received chirp signal to determine the parameters discussed in the previous sections. Zoom-FFT is a technique for signal processing used for analysing a portion of a FFT spectrum at high resolution [17].

2.2.2 Point cloud processing

The information acquired from 3D sensors is used to create point clouds. A point cloud is a collection of data points in a 3D space. Each point has an x-, y-, and z-coordinate, as well as intensity information. The intensity is the representation of the energy reflected by an item. A point cloud is limited to a certain resolution. In 3D space, the resolution is not fixed, and a nonuniform distribution of points may occur. A higher point density in a region means higher computing cost for processing that region. Radar sensors, as most other sensors, induce noise. The noise can cause inaccurate results in a point cloud. There are multiple processing methods to reduce unwanted noise in the point cloud. A simple solution is to filter the data points with regards to intensity or velocity [18].

2.3 ROS

Robot Operating System (ROS) enables communication between software components and hardware devices through an open source software framework. ROS can assist with interfacing hardware with advanced software features such as motion planning, computer vision, simulation and artificial intelligence. ROS mainly supports programming with C++ and Python through the libraries roscpp and rospy [19].

The way information is transmitted in ROS is through *nodes*. A node is a process that performs computations. Depending on whether information is sent or received, a node either *subscribes* or *publishes* to a *topic*. The data type that a topic is holding is called a *message*.

When a node is sending information, it publishes, and when it is receiving information, it subscribes.

Multiple different distributions of ROS exist. New distributions are released yearly and long term support versions (LTS) are released biannually. The ROS ecosystem is split into ROS 1 and ROS 2, where ROS 2 is redesigned in many aspects. Table 3.1 displays a list of current ROS distributions and their advantages as of April 2022. Older ROS distributions has the benefits of a well developed software library, while newer distributions has longer future support and newer features.

2.3.1 RViz visualization software

RViz(short for ROS Visualization) is a 3D visualization tool which is a part of the ROS environment. This tool can visualize sensor data, algorithms and robots. RViz has built in support for a wide range of data, but also supports plugins for specialized applications [20]. RViz features a graphical user interface which is displayed in figure 2.4.

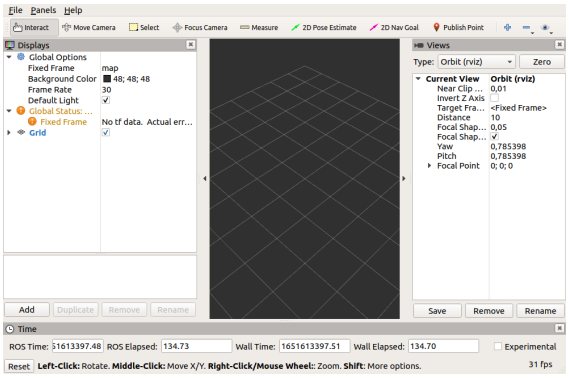

Figure 2.4: RViz main window

2.3.2 URDF Unified Robotics Description Format

Unified Robotics Description Format (URDF) is a specification for describing the mechanics of a multibody system [21]. The format uses Extensible Markup Language (XML) to describe how *links* and *joints* are connected relative to each other, and how these joints behave.

- Links define the geometry of a system, and can be basic shapes such as boxes and cylinders or use custom meshes modeled in 3D modelling software.
- Joints define how the links are allowed to move. There are five types of joints that restricts linear and rotational movement: continuous, revolute, prismatic, planar and floating(unconstrained) [22].

An URDF can also contain physical properties and collision mechanics.

2.4 Geometry, translation, rotation

In the context of describing the position, displacement and rotation of objects within a system, various techniques within the field of geometry can be applied. The concept *Six degrees of freedom* is a description of a rigid body movement in three-dimensional space. This movement can be determined by six parameters divided into two classes:

Translational:

1. Surge: movement along the X-axis
2. Sway: movement along the Y-axis
3. Heave: movement along the Z-axis

Rotational:

4. Roll α: rotation around the X-axis

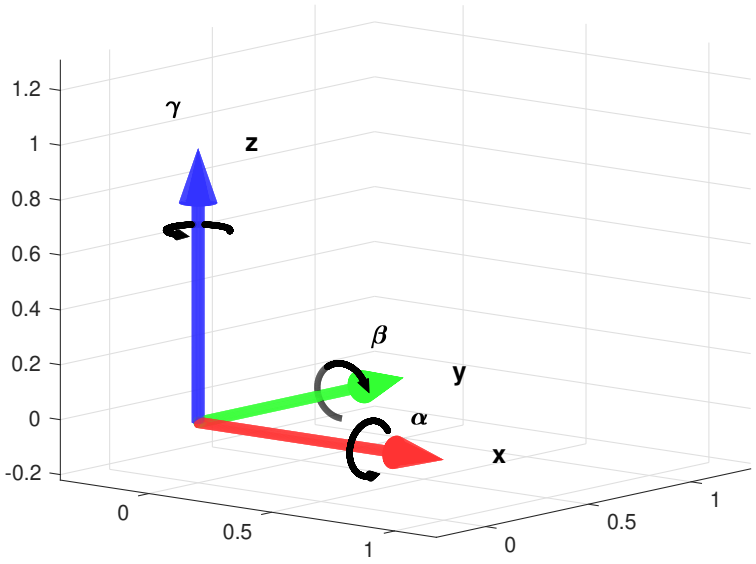

Figure 2.5: Coordinate system with rotation arrows

5. Pitch β: rotation around the Y-axis
6. Yaw γ: rotation around the Z-axis

The terms roll, pitch and yaw (RPY) is collectively known as *Euler angles* and are used to describe rotation of objects in 2D or 3D space. Different conventions for nomenclature and usage of Euler angles exist, therefore only the scheme described in this chapter is used in this report. The main advantage of Euler angles is that they are intuitive to understand, as all rotations are described as amount of degrees/radians of rotation around each axis [23].

Quaternions

A different approach to describing rotation in three dimensional space is by using *quaternions*. Quaternions are generally represented in the form:

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \tag{2.17}$$

where a, b, c, d are real numbers and $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ are the *basic quaternions*.

From a mathematical point of view, quaternions can be considered as an extension of complex numbers which is written in the form $a + b\mathbf{i}$. Quaternions extends this to include a $c\mathbf{j}$ and $d\mathbf{k}$ term [24]. In computer systems, quaternions are often described as a four dimensional vector where $xyzw$ is used instead of $abcd$:

$$q = [x, y, z, w] \tag{2.18}$$

Chapter 3

Method

3.1 Overview

The following sections describes in detail how the experiments was performed, which equipment was used and how the programming was conducted.

3.1.1 Hardware

The hardware used in this project consist of a three sensor devices connected to a common computer for data logging and analysis. Figure 3.1 describes how these devices are connected during experiments. The main hardware are the two FMCW radars AWR1843 and IWR6843 which are used in different configurations to improve precision and accuracy. These radars are connected to a computer running Ubuntu 18.04 with ROS Melodic Morenia installed. A phone is used as a reference IMU for comparison with the data provided by the radars.

Main equipment list:

- AWR1843BOOST development board, described in section 3.2
- IWR6843ISK development board, described in section 3.2
- An iPhone 8 as IMU, described in section 3.3.3
- Laptop running ROS

The AWR1843 radar was used as a starting point for this project. After performing preliminary experiments and testing, it was determined that the sensor had potential but needed configuration and its data needed further processing. The testing was done by following the guide which is supplied with the *ti_mmwave_rospkg* package.

An IMU was connected to provide reference measurements. Due to availability and features, an iPhone 8 was used. A wireless IMU was highly preferential due to the nature of a swinging load, as adding a cable connected device could interfere its movement. A cable could also have interfered with radar signals, however this was not tested and may not have been a significant issue. Due to the preference of having a wireless IMU, it is also implied that it needs to be battery powered. A phone fulfills all these requirements, along with advantages such as ease of connectivity and data quality.

As a another reference measurement device, the IWR6843 radar was used. Although it being similar in specification, it has the advantage of supporting TIs `High Accuracy Measurement` firmware which significantly increases the range resolution(x-axis) in its range measurement, at the cost of providing no data for bearing(y-axis), elevation(z-axis) and velocity. This made it suitable for reference measurements.

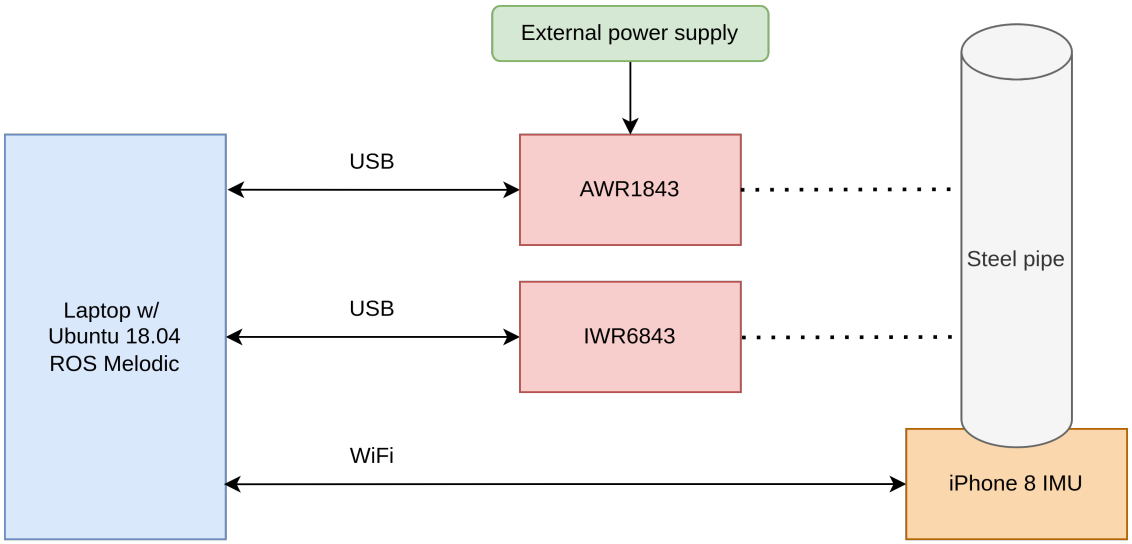

Figure 3.1: Hardware setup

3.1.2 Experiment

An experimental setup in the lab was created. This experimental setup is a steel pipe suspended by a strap to represent a swinging load. Figure 3.4 shows a picture from the real setup. Measures were taken to reduce external noise from other objects in the area. This was done by moving objects that could interfere away from the testing setup and only taking measurements when no other people were around the testing area.

A mount was made for the two radars in order to have repeatable and precise results. This mount were made from an aluminium plate and is displayed in figure 3.3. Figure 3.6 describes the physical dimensions of the experimental setup. The steel pipe was positioned at a 1.00 meter distance from the radar antenna, as this was determined to be a convenient location while giving acceptable results. The steel pipe had a diameter of 48mm and a length of 0.7 meters. A feature of the steel pipe was a crank to fine adjust its angle, which is displayed in figure 3.2. A larger plastic pipe (figure 3.5c) was tested in the setup, but it yielded poor results likely due to worse reflections for the radar. A box steel pipe (3.5b) was also tested, but it was ultimately decided that the cylindrical pipe (3.5a) was more convenient.

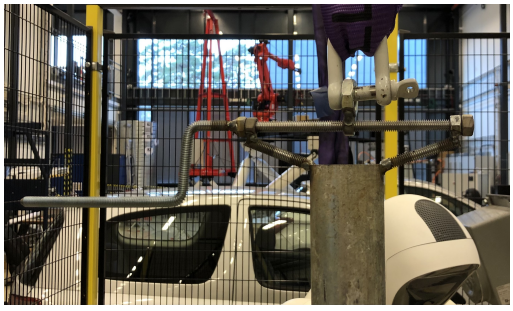

Figure 3.2: Steel pipe crank

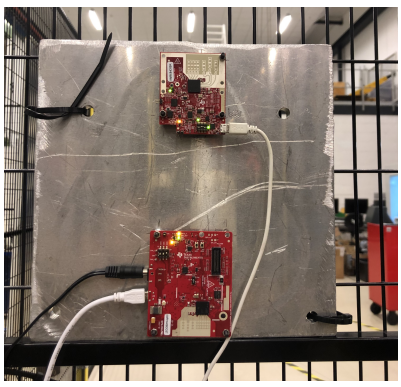

Figure 3.3: Aluminium radar mounting plate

Figure 3.4: Experimental lab setup. To the left is the aluminium mounting bracket for the two radars, and to the right is the cylindrical steel pipe that was used during experiments. The pipe is connected to a steel beam with a strap. The steel beam is overhead and is not in frame of this picture.

(a) Cylindrical steel pipe

(b) Box steel pipe

(c) Plastic tube

Figure 3.5: Three different objects were tested as swinging load.

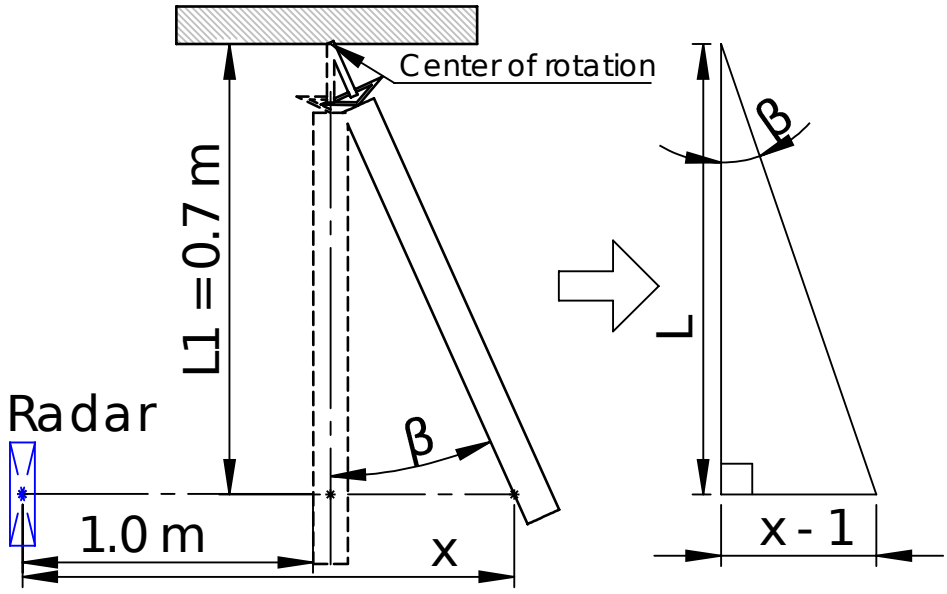

Figure 3.6: Geometric considerations of the experimental setup.

3.1.3 ROS setup and package configuration

For the purpose of both logging data and real time visualisation, ROS was determined to be a suitable platform. Selection of ROS over other systems is further discussed in section 5.2. The ROS distribution Melodic Morenia was determined to be well suited for this project, mainly due to it supporting TI's ROS package. Figure 3.1 displays and overview of currently active ROS distributions and their features. Multiple features of ROS were used including driver software for the AWR1843, rosbag, and RViz. Figure 3.7 displays a chart of the configuration of ROS that was used.

Picture	Distribution name	Year	ROS version	Features	EOL
	Melodic Morenia	2018	ROS 1	Old but extensive package support	May 2023
	Noetic Ninjemys	2020	ROS 1	Last ROS 1 release	May 2025
	Foxy Fitzroy	2020	ROS 2	Current ROS 2 LTS release	May 2023
	Galactic Geochelone	2021	ROS 2	Newest ROS developement version	Nov 2022

Table 3.1: ROS distribtions as of April 2022[25][26]
. The distribution used in this project is highlighted in gray.

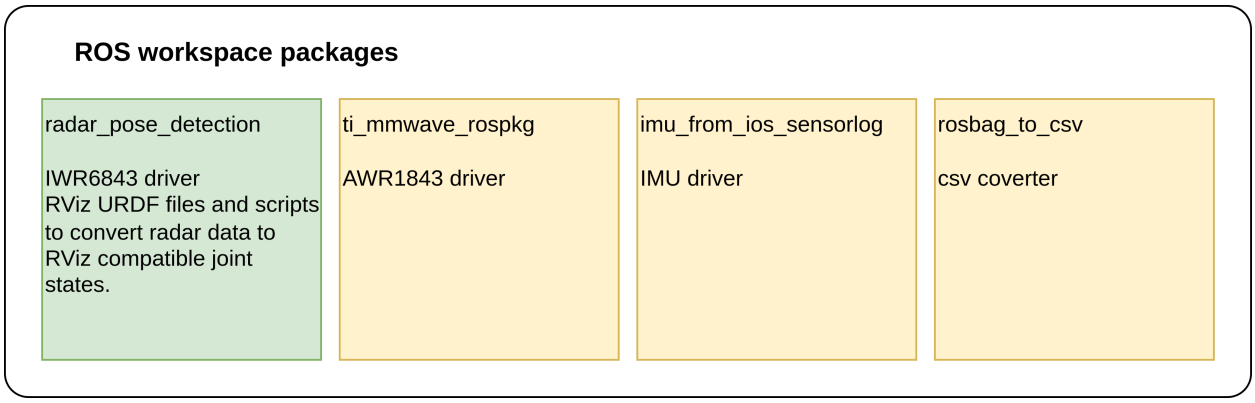

Figure 3.7: ROS workspace. The package created for this project is highlighted in green, while downloaded resources are highlighted in yellow.

3.2 Radar Hardware: AWR1843BOOST and IWR6843ISK

Two different FMCW radars were used for this project. Both are development boards made by Texas Instruments which feature a radar chip with supporting electronics for data processing and USB communication. The AWR1843 radar was used as measurement radar while the IWR6843 was used as a reference for range measurement.

AWR1843BOOST

AWR1843BOOST is a radar development board from Texas Instruments and uses a AWR1843 single-chip 76-81 GHz FMCW radar. It features 4 receivers and 3 transmitters and uses a standard micro-USB B cable to interface with a computer. This development board requires an external power supply which is specified to be rated at 5V minimum 2.5A. The version of the board that was used is an early revision of the chip, which is no longer supported by TI. A consequence of this were that only older firmware would work on it [27]. Figure 3.8 features a picture of the development board.

IWR6843ISK

IWR6843ISK is a radar development board from Texas Instrumentsand uses a IWR684 single-chip 60-64 GHz FMCW radar. It features 4 receivers and 3 transmitters and uses a standard micro-USB B cable to interface with a computer [28]. Figure 3.9 displays the IWR6843 development board.

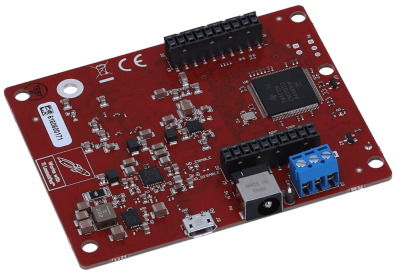

Figure 3.8: AWR1843BOOST Development Board

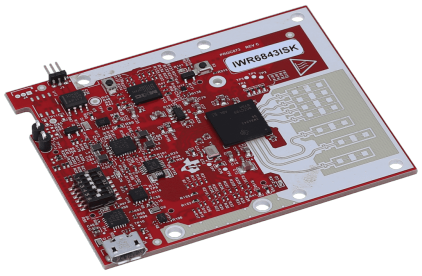

Figure 3.9: IWR6843ISK Development Board

3.2.1 Radar firmware

Two different radar firmware editions were used in this project. Both editions are intended to work on both the AWR1843 and the IWR6843 radars. However due to the AWR1843

being an early revision of the chip, Texas Instruments did not support it. These firmware are acquired from Texas Instruments mmWave Industrial Toolbox version 4.10.1[29]. Table 3.2 displays which firmware were used for the radars and the features of the different firmware versions.

Radar	Firmware	Features
AWR1843	"Out of box demo"	For general radar applications. Outputs point cloud data Highly configurable.
IWR6843	"High Accuracy Level Sensing"	For sub-millimeter precision measurements. Uses zoom-fft to increase resolution. Outputs range-FFT and the three strongest peaks.

Table 3.2: Radar firmware

3.2.2 Comparison between AWR1843BOOST and IWR6843ISK

A distinguishing difference between the two radars is that the model name starts with the letter A or I which correspondingly stands for Automotive and Industrial specification [30]. The practical meaning of this is that the devices are rated for different operating environments(mainly temperature) and does not have any impact of the experiments performed in this project. Regarding physical differences, AWR1843BOOST has a larger PCB footprint, has a port for CAN-bus communication and requires an external power supply while IWR6843ISK does not. Along several minor technical differences between the two devices, a major difference is that AWR1843BOOST operates at frequencies ranging from 76-81-GHz while IWR6843ISK operates at 60-64-GHz.

3.3 Interfacing ROS with radar hardware and IMU

Each hardware sensor required different approaches to be connected with ROS. To be connected with ROS essentially means that the sensors data is published by a node as a topic through ROS. This requires driver software that can either be sourced from a software repository if it already exist, or needs to be written if no compatible driver software is available. Table 3.3 displays an overview of the driver software that was used in this project and the topics they publish. Figure 3.10 displays a list of active topics while the hardware is connected and corresponding driver software is running.

```
(base) tor@tor-UX305CA:~/catkin_ws$ rostopic list
/clock
/imu_iphone
/joint_states
/rosout
/rosout_agg
/tf
/tf_static
/ti_mmwave/radar_scan
/ti_mmwave/radar_scan_pcl
```

Figure 3.10: The command line tool `rostopic` used to list active topics while the system is running.

Section 3.3.1, 3.3.2 and 3.3.3 describes implementation and usage of the driver software for each device.

Device	Driver package Node name	Published topics
AWR1843BOOST	ti_mmwave_rospkg ti_mmwave	/ti_mmwave/radar_scan /ti_mmwave/radar_scan_pcl
IWR6843ISK	radar_pose detection readData_IWR6843ISK	/joint_states
iPhone 8 IMU	imu_from_ios_sensorlog imu_from_ios_sensorlog_node	/imu_iphone

Table 3.3: Driver software

3.3.1 AWR1843: ti_mmwave_rospkg

To communicate with the AWR1843 board, the ROS package `ti_mmwave_rospkg` was downloaded. This package connects to the AWR1843 radar and publishes its data as the ROS topic `/ti_mmwave/radar_scan`. Table 3.4 displays a simplified overview of the data that is a part of this topic. First a version provided by radar-lab on GitHub was tested[31] which worked decently, however it did not publish the values of `range, doppler_bin, bearing, intensity` to the ROS system. This was fixed by switching to a forked version made by user Claud1234 on GitHub[32], which patched the `DataHandlerClass.cpp` file to provide full support for the AWR1843 radar. The `ti_mmwave_rospkg` ROS package is designed to work with the "Out of box demo" firmware for AWR1843.

Data type	Explanation
Position	Position of detected object in coordinates relative to the radar antenna [x, y, z] in meters
Range	Range to detected point in meters
Bearing	Angle of the detected object relative to the x-axis from the radar antenna in [rad]
Intensity	Intensity of detected point based on signal-to-noise ratio
Velocity	Speed of the detected object in $[\frac{m}{s}]$

Table 3.4: Output data from the `/ti_mmwave/radar_scan` ROS topic.

3.3.2 IWR6843: custom python parser script

Due to the IWR6843 running a specific firmware the only available way to read and display this data was through Texas Instruments own *High Accuracy Visualizer version 2.0.0*[33]. Although it does provide the functionality to log data this feature is limited and the data would be challenging to synchronise with the data from the AWR1843 and the IMU for comparison. It was therefore necessary to write a script to parse the data sent over serial USB. The procedure for this in broad steps was:

1. A script performing a similar task was found on GitHub[34]
2. Unnecessary parts of the script was removed or edited to fit the IWR6843.
3. The `C` source code of the IWR6843 High accuracy firmware was inspected to determine the data format.
4. The JavaScript source code of the *High Accuracy Visualizer* was inspected to determine how the data was parsed.
5. Raw data was logged to a spreadsheet and manually performed calculations on to verify if the format was interpreted correctly.
6. Code for parsing was implemented.

7. ROS node was created to publish the data to the ROS system.

Figure in appendix B.1 displays the starting section of a parsed message from the IWR6843ISK running high accuracy measurement firmware. The first column is a label and has the name of the data structure internally in the firmware source code. The second column contains the raw data read from the serial port where each row has two bytes of data in decimal format. The third column has partial interpretations and annotations of the data. The data structure `MmwDemo_detectedObj` contains three range estimates, x_1, x_2, x_3, separated into 9 fields. Table 3.5 describes the mapping of these fields to the corresponding bytes. Note that the mapping is not in the same order as the data received over serial, as the fields `peakVal` and `dopplerIdx` are switched. The labels `rangeIdx`, `peakVal` and `dopplerIdx` only contain information about range estimates, and are likely reused variable names from different firmware versions. The three range estimates are calculated according to the following formula:

$$x_{est} = \frac{(bytesA \cdot 2^0 + bytesB \cdot 2^8 + bytesC \cdot 2^{16}) \cdot 1.36}{1048576} \tag{3.1}$$

For the dataset provided in figure B.1, this would then give a range estimate value of:

$$x_1 = \frac{(53 \cdot 1 + 68 \cdot 2^8 + 25 \cdot 2^{16}) \cdot 1.36}{1048576} = 2.14764[meter] \tag{3.2}$$

Range estimate	**bytesA**	**bytesB**	**bytesC**
x_1	rangeIdx_1	rangeIdx_2	x1
x_2	peakVal_1	peakVal_2	x2
x_3	dopplerIdx_1	dopplerIdx_2	x3

Table 3.5: Mapping of bytes to range estimates, referencing the values in figure B.1.

3.3.3 IMU

For the lab experiments, an iPhone 8 was used as an IMU. The IMU hardware is a custom sensor from Bosch Sensortec[35]. Although there exist many options for logging motion data, this was determined to be a reasonable choice due to multiple reasons:

- The data is preprocessed to be unbiased, meaning environmental factors such as the effect of gravity is removed[36].
- The device has built in wireless communication(Wi-Fi) and a battery for wireless operation.
- Software was already available to enable communication between the iPhone 8 and ROS: imu_from_ios_sensorlog ROS package and the SensorLog app.

This data can be sampled at up to 100 Hz, but a rate of 60 Hz was used in this project to match the framerate that was used in RViz. For visualization purposes, 60 Hz is suitable as it is the standard refresh rate of computer monitors [37]. For an industrial application in a control system, a higher IMU refresh rate would be desired. The output data from the IMU is described in table 3.6.

Data type	Format
Orientation	Quaternion [x, y, z, w]
Angular Velocity	Vector in rad/s [x, y, z]
Linear Accelatration	Vector in m/s^2 [x, y, z]

Table 3.6: Output data from the /imu_iphone topic.

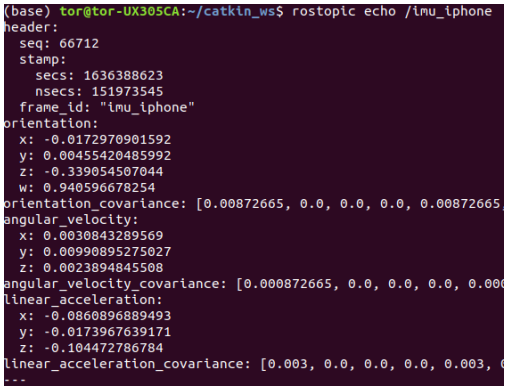

Figure 3.11: IMU message

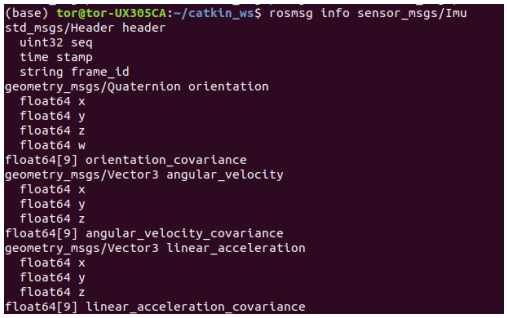

Figure 3.12: IMU message format

The app SensorLog[38] was used to transmit the motion data from the phone to ROS. This was done by configuring a local Wi-Fi hotspot on the host machine and connecting the phone to it. The SensorLog app was then configured to act as a TCP server to enable streaming of data from the phone to the host computer. The IP address of the phone was noted and inserted into the `imu_from_ios_sensorlog` launch file. When the node was launched with the command line tool `roslaunch`, then sensor data was available as a ROS topic. Figure 3.13 displays a screenshot from the SensorLog app interface, while table 3.6 has an overview of the data that is output by the `/imu_iphone` topic. The SensorLog app outputs device orientation in RPY format, which is converted to quaternions by the `imu_from_ios_sensorlog` package. The RPY values could have been used directly in RViz without conversion back and fourth through quaternions, but due to the standard ROS IMU message format using quaternions, it was performed this way. Figure 3.12 displays the data structure of a standard ROS IMU message, while figure 3.11 displays an IMU message with data.

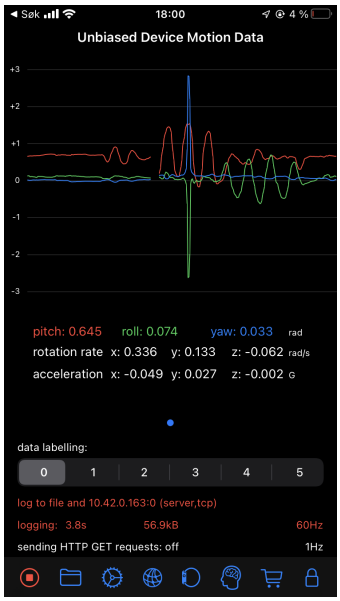

Figure 3.13: Screenshot from the SensorLog app interface.

3.4 Logging and Visualization

The command line tool `rosbag` [39] was used for logging purposes in this project. This tool enables all active ROS topics to be saved to a file, which can later be replayed or converted to a csv file. The main advantage of this is that the dataset from an experiment can be replayed and further investigated after the physical experiment has taken place. Furthermore, the package `rosbag_to_csv` was used to convert rosbag files to csv files for further data analysis.

An URDF robot model file was created to represent the physical experimental setup in the lab. The main advantage of this is to be able to verify visually in real time the quality of the data that is being logged. Figure 3.15 displays a simplified version of an URDF file that was used in this project. A ball-and-socket joint is modeled by combining three separate revolute joints. The URDF format requires each joint to connect to two links, therefore dummy links are inserted between the revolute joints. Each joint takes as input a value in radians that corresponds to a rotation displayed in RViz.

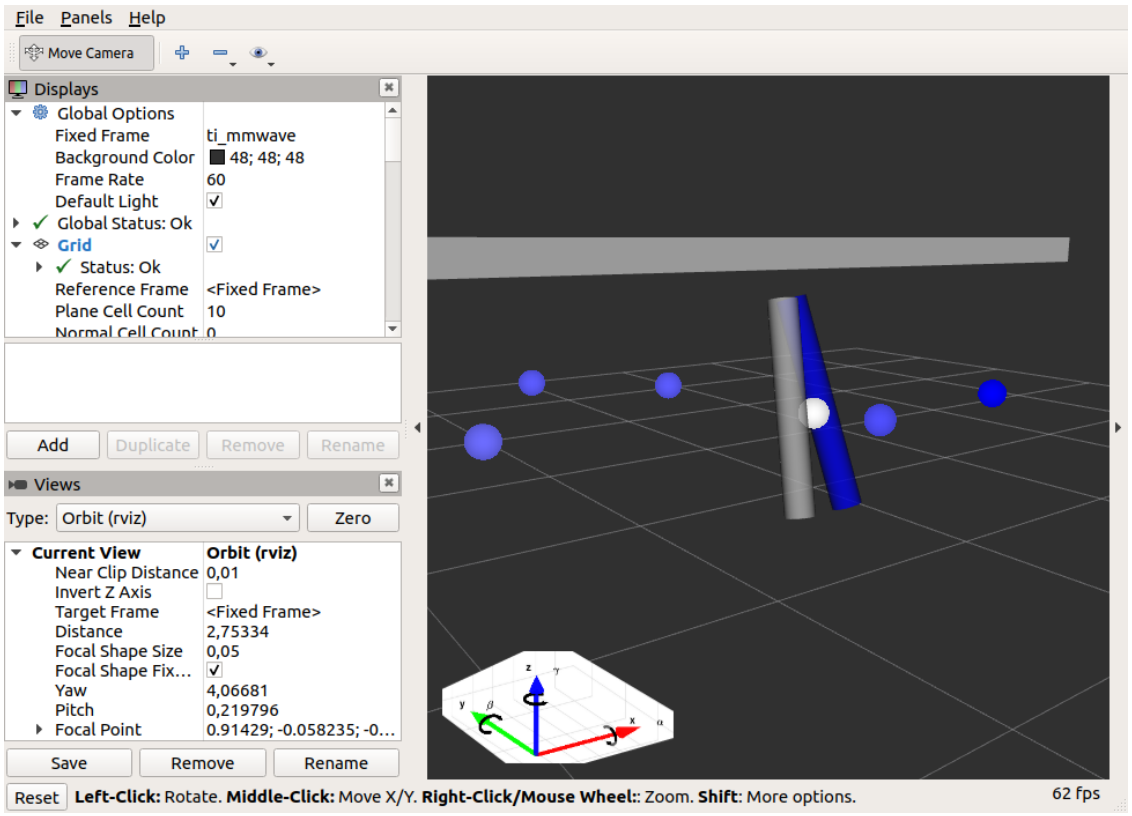

Figure 3.14: RViz

Figure 3.14 is a screengrab from RViz during real-time operation of the system.

- The spheres colored in the range from white to dark blue represents the raw point-cloud data from the radar. An advantage of including this data is that it makes sure that the radar sensor is located `(0, 0, 0)` in the coordinate system. The color is based on intensity of the signal(Signal-to-noise ratio) where dark blue is weak while white is the strongest. The scale is automatically set by RViz, therefore the white dot is always the strongest signal.

- The gray cylinder represents the data from the IMU. This uses essentially raw data, where the only processing that is done is converting the quaternions that the IMU outputs to euler angles that are accepted by the URDF model.

- The blue cylinder represents the filtered and processed data from either the AWR1843 or the IWR6843 radar.

- Additionally, a gray beam is included at the top to represent the geometry of the system. It has no technical purpose, and is only there for visualisation purposes.

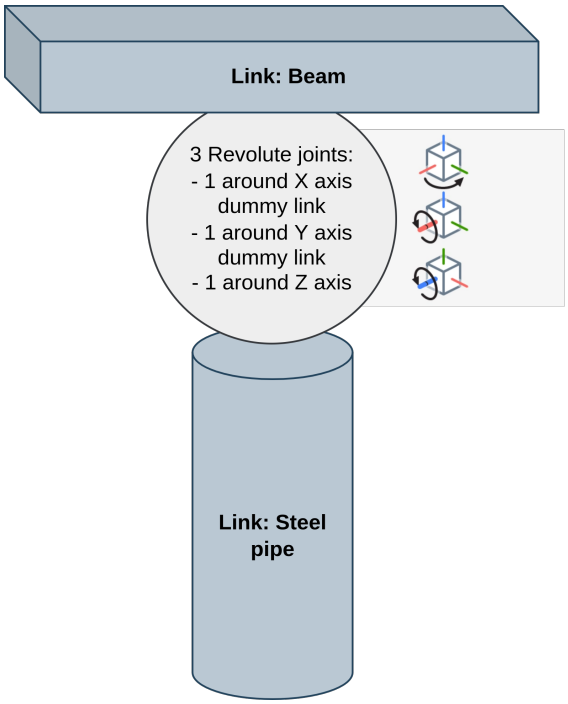

Figure 3.15: Schematic view of the URDF model. Dummy links are inserted between each joint as the URDF format has no ball- and socket joint predefined.

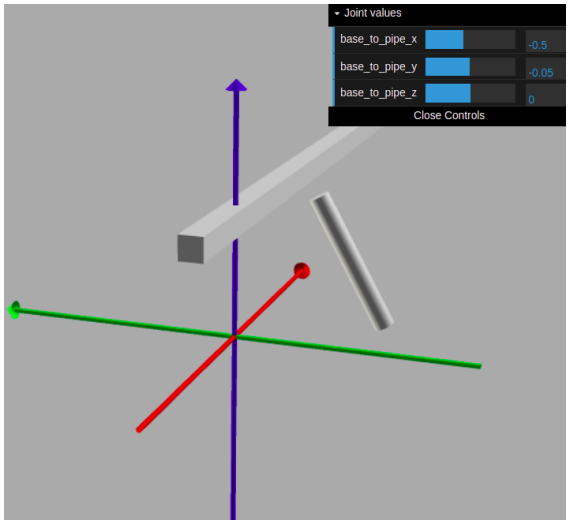

Figure 3.16: Screenshot from the online URDF viewer mymodelrobot. The joint base_to_pipe_x is rotated. This exact model can be interacted with at [40].

3.5 Data Processing

The following sections describe the data processing techniques and algorithms used to process the data from the experiments.

3.5.1 Selection of bestX

As the IWR6843 outputs the three strongest peaks from the range FFT, it was needed to select which of these to use in an automated manner. The three values x_1, x_2 and x_3 from the IWR6843 switches which object they represent continuously while data logging. An algorithm for determining which value of x_n was most likely to be the swinging load was therefore implemented. This algorithm is based on finding the x_n value is closest to 1 [m], due to the starting position of the swinging load being 1 [m]. Appendix A lines 285-300 contains the implementation of this in Python.

3.5.2 Region of Interest

As the radars captures data from a wide field of view beyond the measurement subject, a region of interest (ROI) was defined for the experiment. This excludes and discards data points that not within the potential range that the swinging load cloud move. The ROI that was used in general is defined in table 3.7, and is based on empirical values.

	Min	**Max**
x	0.4 [m]	1.5 [m]
y	-0.7 [m]	0.9 [m]

Table 3.7: Region of Interest

This ROI was used for both the plotted graphs and also during real-time operation in RViz. Additionally, velocity was also used as a metric in the ROI. Any points that had a velocity of zero was discarded during operation in RViz.

3.5.3 Conversion from range to angle

In order to measure the angle of the swinging load, the range measurements have to be used. When knowing the position of the load, trigonometry can be applied to calculate the angle β. The geometry of the system is described in figure 3.6. The value of β is estimated using the inverse tangent trigonometric function:

$$\beta = \tan^{-1} \frac{x-1}{L} \tag{3.3}$$

3.5.4 Data clustering

The data collected from the radar included noise. An algorithm (see algorithm 1) was therefore used to sort the swinging load data from the noise. This is a DBSCAN (Density Based Spatial Clustering of Applications with Noise) algorithm [41], which groups together points that are closely packed together. These groups of points are referred to as clusters. Multiple clusters can be identified, and points belonging to no cluster are identified as noise.

Algorithm 1 Displacement and velocity estimation using radar.

Input: data from radar, $\boldsymbol{p} = [\boldsymbol{p}_i]_{i=1}^{N}$, $\boldsymbol{p}_i = [p_i^x \quad p_i^y]$ ▷ radar point cloud consisting of x and y co-ordinate of all points in its field of view. $\boldsymbol{p}_i \in \mathbb{R}^{1\times 2}$ and $\boldsymbol{p} \in \mathbb{R}^{N\times 2}$

Output: displacement, $\boldsymbol{d}$ ▷ displacement of the swinging load. $\boldsymbol{d} \in \mathbb{R}^{N'\times 2}$ (where, $N' < N$). Each column gives displacement in x and y direction, $\boldsymbol{d} = [\boldsymbol{d}^x \quad \boldsymbol{d}^y]$, such that $\boldsymbol{d}^x = [d_i^x]_{i=1}^{N}$, $\boldsymbol{d}^y = [d_i^y]_{i=1}^{N}$

$\boldsymbol{z} \leftarrow \text{cluster}(\boldsymbol{p})$ ▷ apply density based clustering. $\boldsymbol{z} = [z_i]_{i=1}^{N}$, where z_i is radar point to cluster assignment, known as cluster label of p_i

$K \leftarrow \max(\boldsymbol{z})$ ▷ total number of cluster

for k=1 to K **do**

  $\boldsymbol{c}_k \leftarrow [\boldsymbol{p}_i]_{i:z_i=k}$ ▷ cluster k contains all the points ($\boldsymbol{p_i}$) whose cluster label (z_i) is k

  $\boldsymbol{o}_k \leftarrow \text{mean}(\boldsymbol{c}_k)$ ▷ mean of each cluster. $\boldsymbol{o}_k = \begin{bmatrix} o_k^x & o_k^y \end{bmatrix}$, where o_k^x, o_k^y is the x and y component.

  $\boldsymbol{\sigma}_k \leftarrow \text{var}(\boldsymbol{c}_k)$ ▷ variance of each cluster. $\boldsymbol{\sigma}_k = \begin{bmatrix} \sigma_k^x & \sigma_k^y \end{bmatrix}$, where σ_k^x, σ_k^y is the x and y component.

end for

$\boldsymbol{c}_k \leftarrow [\boldsymbol{c}_k]_{k:\boldsymbol{o}_k<1.5 \quad \& \quad \boldsymbol{\sigma}_k!=0}$ ▷ select the clusters that meet these conditions. First conditions is based on the resolution obtained experiments to acquire the relevant data and second conditions selects the points in cluster k that are dynamic (filtering the static points).

$\boldsymbol{c}_k \leftarrow [\boldsymbol{c}_k]_{k:\max(\sigma_k^x)}$ ▷ select the points in cluster k that has maximum variance in x direction. $\boldsymbol{c}_k$ contains all the points corresponding to swinging load. $\boldsymbol{c}_k = [\boldsymbol{p}_i]_{i=1}^{N\prime}$, where $N' < N$ and $\boldsymbol{p}_i = [p_i^x \quad p_i^y] \quad \forall i$

$\boldsymbol{d}^x \leftarrow \mathbf{c}_k[:, 1]$ ▷ first column in cluster $\boldsymbol{c}_k$ contains the x-displacement of swinging load

$\boldsymbol{d}^y \leftarrow \mathbf{c}_k[:, 2]$ ▷ second column in cluster $\boldsymbol{c}_k$ contains the y-displacement of swinging load

$\boldsymbol{d} \leftarrow [\mathbf{d}^x \quad \boldsymbol{d}^y]$ ▷ displacement in both x and y plane

return $\boldsymbol{d}$

Chapter 4

Results

In this chapter, the results of the work done will be presented.

4.1 Datasets

Dataset	Explanation
Dataset A	Dataset from AWR1843 and IWR6843 measuring the swinging load during a pendular motion[42].
Dataset B	Range and velocity measurements from AWR1843 and IMU during pendular motion[42].
Dataset C	Random data collected to cover of the of field of view of the AWR1843 radar[43].

Table 4.1: The three different datasets that are addressed in the results section. Dataset A contains multiple experiments.

4.2 Clustering

The point cloud captured from the measurement radar (AWR1843) using experimental setup is shown in figure 4.1. This figure contains detected points in the x-y plane in dataset A.

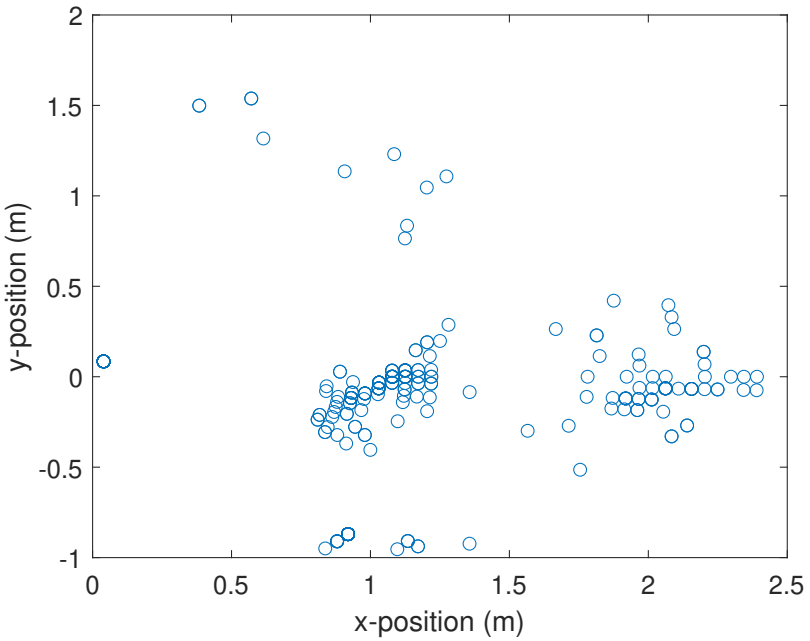

Figure 4.1: Data captured from measurement radar (AWR1843). Dataset A.

As shown in the previous figure, some of the data points that were recorded does not belong to the swinging load. Figure 4.2 shows the same data as figure 4.1, and specifies the cluster of points belonging to the swinging load. The acquired data contains multiple clusters related to the swinging load, sparse data points and static background. By filtering the data points based on algorithm 1, the cluster corresponding to swinging load is filtered from rest of the objects (shown in figure 4.1(up)) and its corresponding x and y position (down). This cluster contains the information related to the spatial location of swinging load in x-y plane.

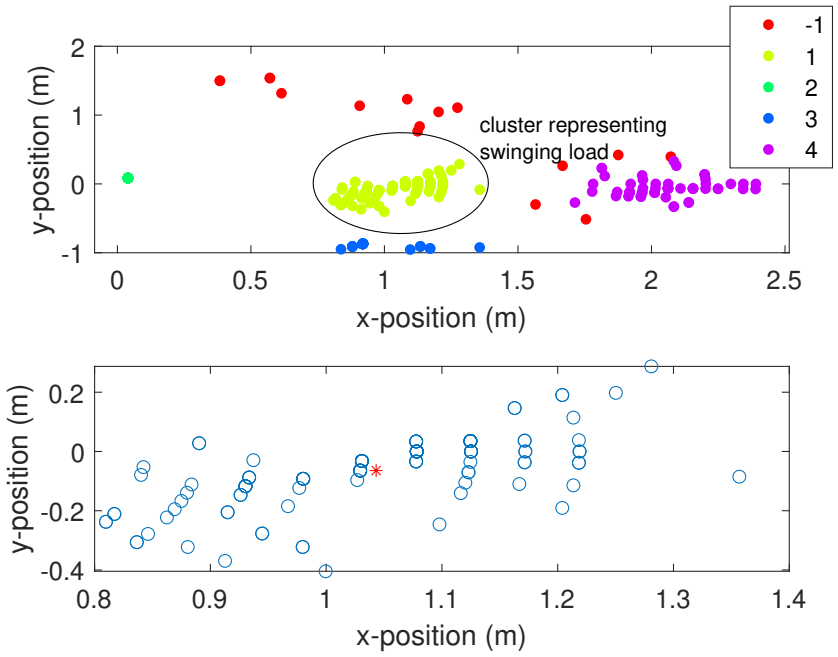

Figure 4.2: Clustering the point cloud related to swinging load. Dataset A.

Figure 4.3 shows that the point cloud data that contains the position of object in x-y plane. It is visible that the acquired data can be differentiated in three regions - (a) contains the information about the swinging load where the data are dynamic and densely distribute, (b) contains the sparsely data that contain very little information about the swinging load and (c) refers to the static background. Further, figure 4.4 gives the exploded view of the region (a) where the data points are distributed with the resolution of 0.03 [m] in x-direction.

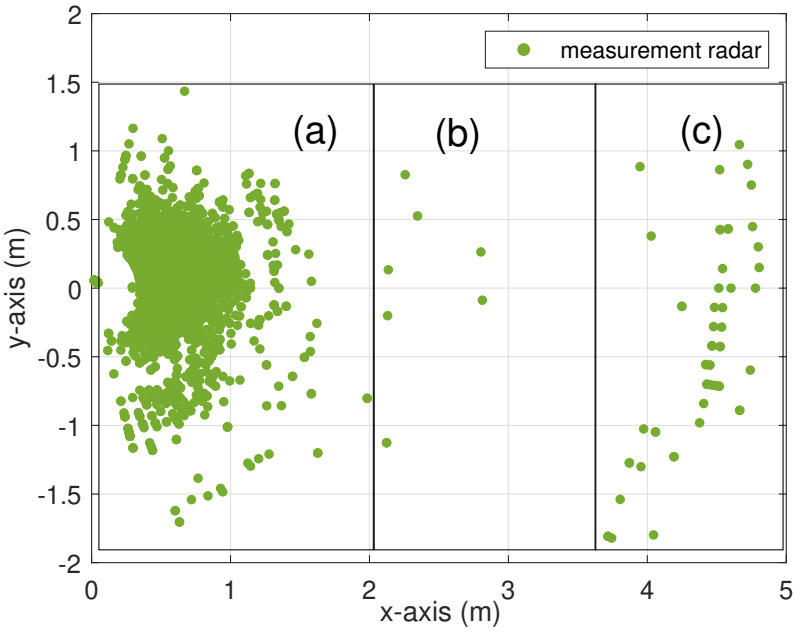

Figure 4.3: Point cloud data from measurement radar. Dataset C.

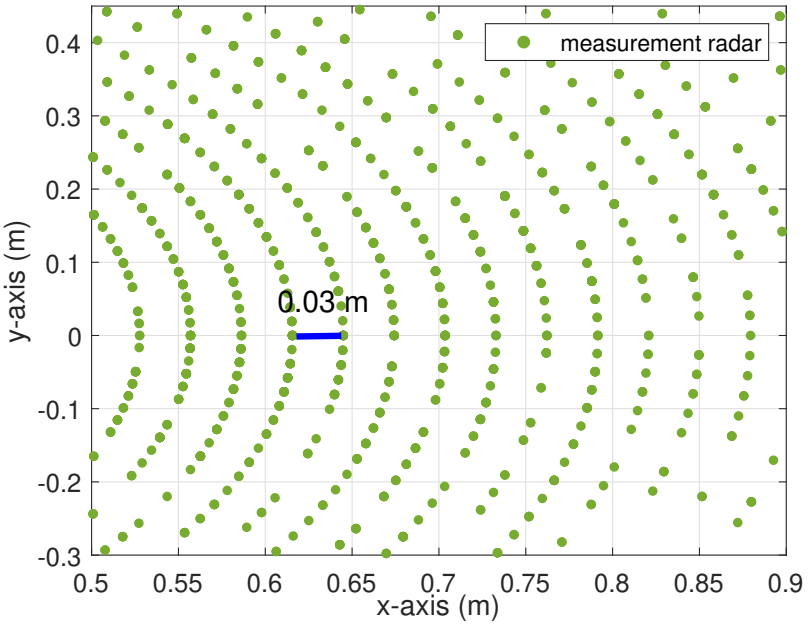

Figure 4.4: Demonstration of range resolution of region (a) in figure 4.3. Dataset C.

4.3 Displacement

Figure 4.5 shows the measurement and reference radar data for dataset A, when filtered by the ROI.

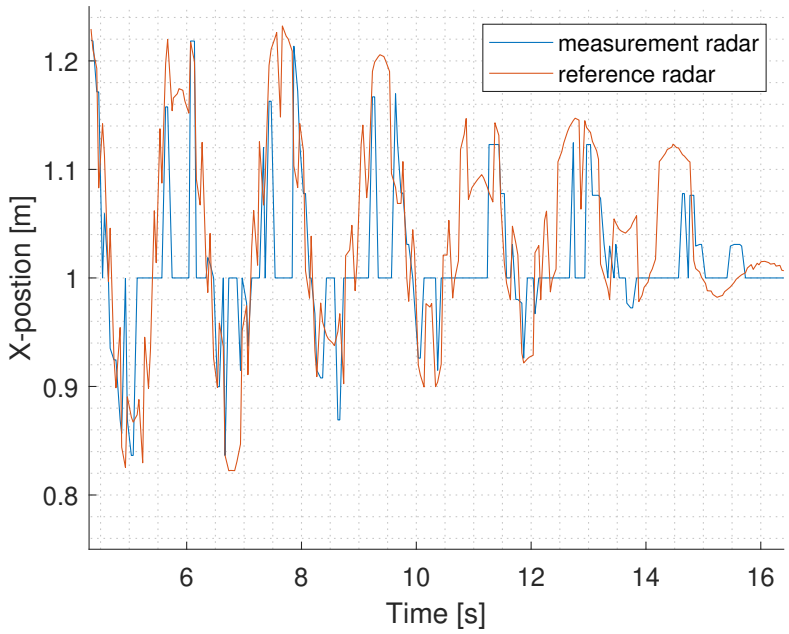

Figure 4.5: X-position filtered to only include points within the egion of interest. Dataset A.

The retrieved displacement in x-direction compared with the reference radar sensor (IWR6843) is shown in figure 4.6. Although the proposed method and algorithm is capable to estimate the displacement in x- and y-direction, the reference radar sensor is only capable to measure the displacement in x-direction (along its axis). So the validation is limited to x-direction only. The root mean square error (rmse) obtained from displacement estimated from the proposed method and the standard reference radar sensor is 3 [cm].

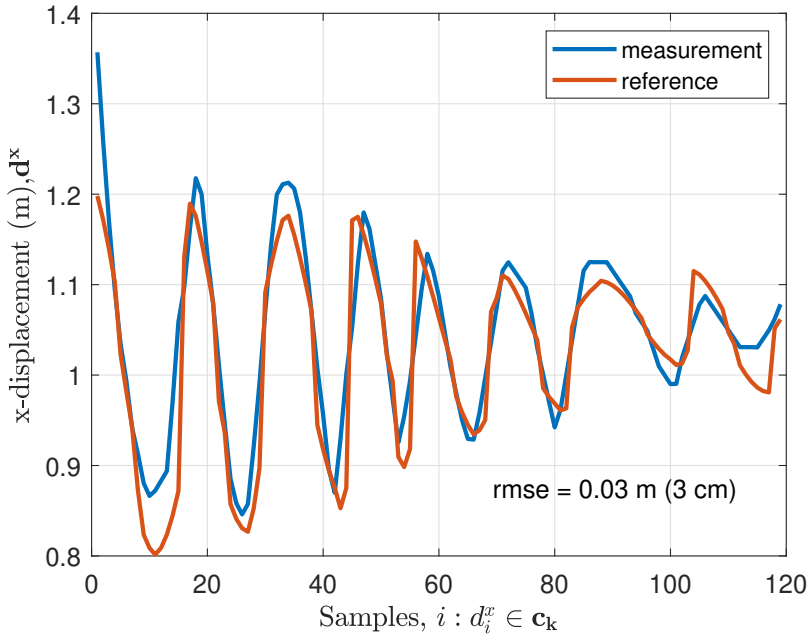

Figure 4.6: Estimated displacement in x-direction from comparing against reference sensor. Dataset A.

To validate the proposed method, four experiments were performed with different amplitudes. These are shown in figure 4.6 and figure 4.7. For each case the displacement is estimated using the proposed method and compared with the reference radar sensor. The rmse in each case is summarized in Fig 4.7.

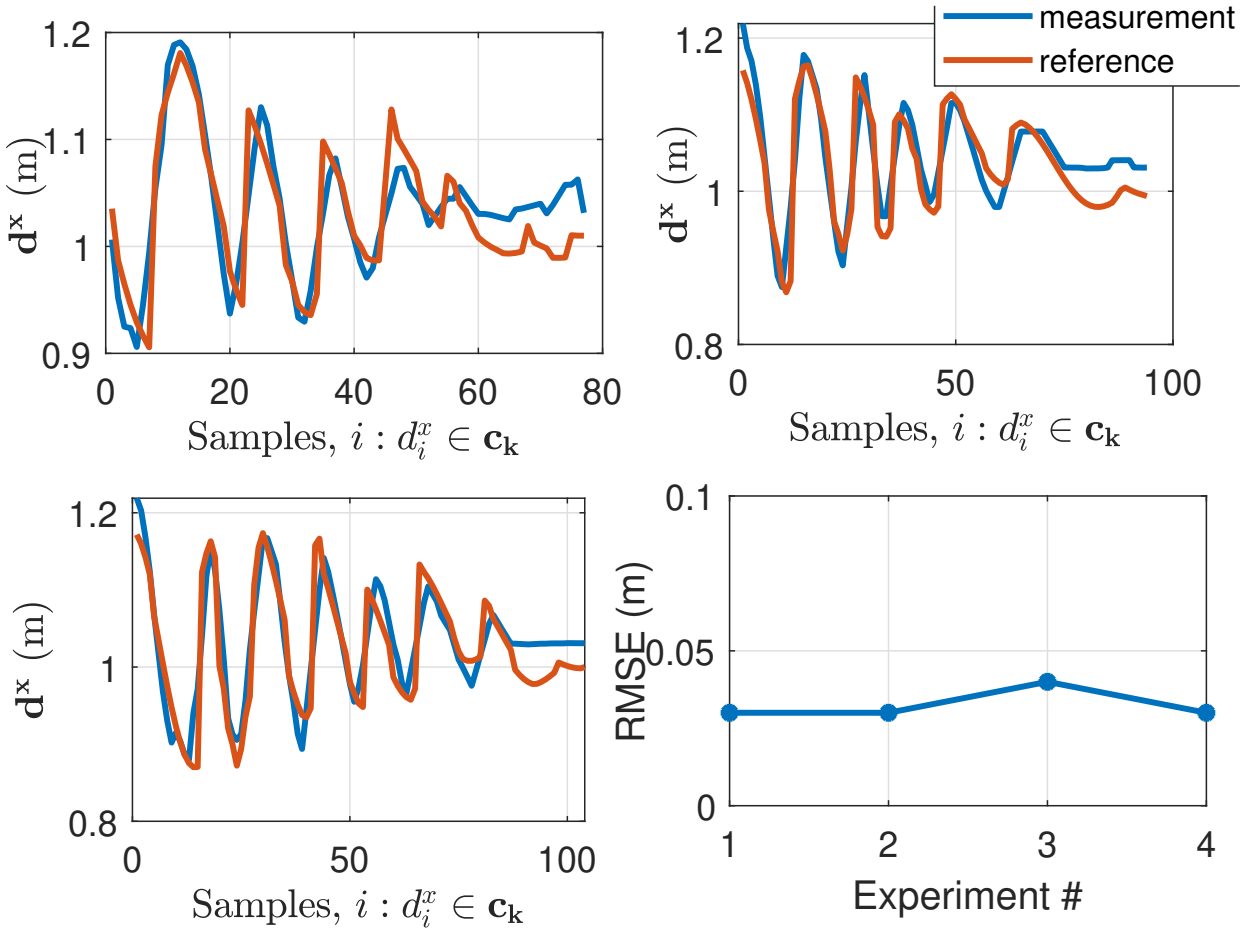

Figure 4.7: Experiments to validate proposed method for displacement estimation and rmse. Dataset A.

4.4 Velocity and angle

The IMU that was attached to the swinging load during operation provided very accurate measurements. Therefore, it was used as a reference for comparing and verifying the radar measurements. As shown in figure 4.8, the radar's measurements were fairly accurate during velocity estimation. However, the radar had a better capability of measuring the velocity when the load was swinging away from it, then it had when the load was swinging towards it. This deviation seemed to decrease as the swinging load slowed down. The rmse in this case was 0.1268 $[\frac{m}{s}]$.

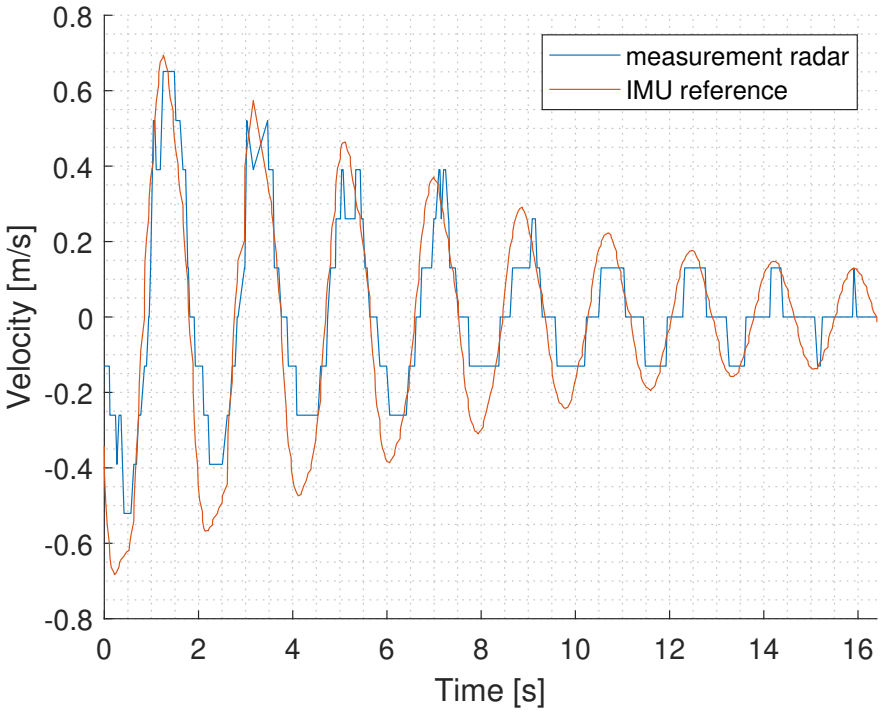

Figure 4.8: Velocity. Dataset B.

Figure 4.9 shows the angle measurements of the swinging load in motion. The AWR1843's measurements were somewhat accurate during the angle estimations. However, there were significant deviation in the measurements at certain points. The rmse in this case was 4.3821 [deg] or 0.07 [rad].

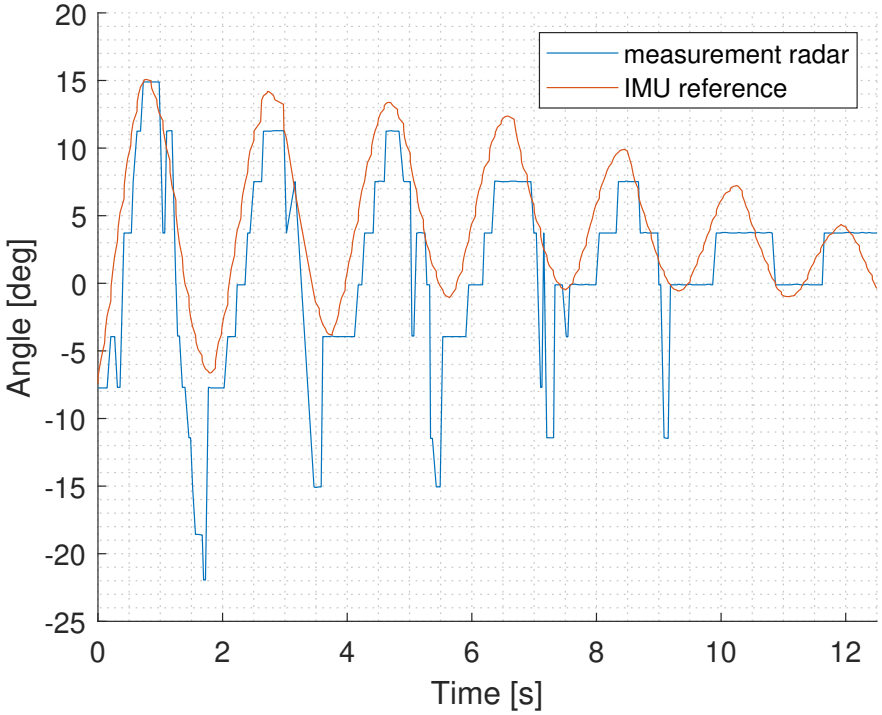

Figure 4.9: Angle. Dataset B.

Chapter 5

Discussion

5.1 Data quality

In this thesis it has been emphasized to get as high-quality data from the AWR1843 radar as possible. This has been done by investigating which configurations provides the highest range resolution, highest framerate, and lowest noise levels. In addition to this the physical setup was adjusted to increase data quality, by testing different objects as swinging load and removing interfering objects in the field of view of the radar. A source of noise was other people moving in the lab while experiments were conducted. The datasets which include noise from people moving were therefore discarded.

5.2 ROS as data acquisition and visualization platform

ROS was chosen as a platform to develop the system on, but other approaches could have been taken. Two alternative platforms to implement a system with would be only using Python or MATLAB:

- Only Python: In this project, python was used as a part of ROS but could also have been used alone without ROS. This would have the advantage of better platform portability of the code as Python can be used in any Windows/MacOS/Linux environment while ROS is dependent on specific Linux distributions.

- MATLAB: A lot of code would need to be written to connect and parse the data from the senors, but MATLAB also has the advantage of being easy to develop with and having good tools built-in for presenting and analyzing data. MATLAB would also have a portability advantage over ROS similar to Python.

- Another alternative for data capture would be using the sample tools provided by TI (mmwave demo vizualizer and High accuracy Vizualizer), however these are limited to very simple logs without timestamps. The source code of these tools are available and could be modified to behave as desired, but this would require significant development effort to get familiar with their frameworks. It was therefore determined that using more standard engineering tools would be better.

ROS is a layer on top of Python and C++ which adds complexity, but also provides useful tools to interact with the data from the sensors. MATLAB and Python has good capabilities for plotting data, but does not have a tool like Rviz which ROS has. The logging tool *rosbag* is also more powerful than logging that would be easily possible with MATLAB or Python, which would be limited to reading and writing `csv` files, without more extensive development.

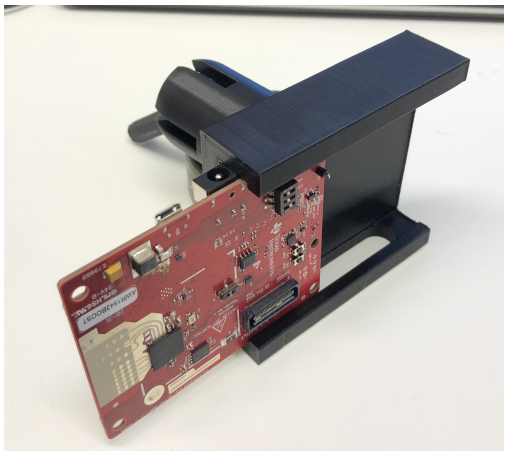

Figure 5.1: Initial bracket design

Figure 5.2: Ball-and-socket joint as a part of the bracket

5.3 Radar mounting bracket

Two different radar mounting brackets was made for the experiment:

1. An aluminium plate with drilled holes for experiment 1 displayed in figure 3.3
2. A 3D printed bracket with a ball-and-socket joint for experiment 2 displayed in figure 5.1 and 5.2

An aluminium plate was used for the experiment, which worked as intended. Figure 5.1 displays the first 3D printed bracket design, which features a slide in boltless slot and a ball-and-socket joint(figure 5.2) for adjustable angle. The socket joint worked as intended under light load, but it was determined that it was not stiff enough for reliable operation. The slide in slot was too small for the AWR1843 radar to fit, and two different brackets would be needed to made for both radars to be mounted simultaneously.

The design process for creating the bracket was:

1. Both AWR1843 and IWR6843ISK circuit boards were measured with calipers.
2. A sketch was made by hand on paper based on the measurements of the radars.
3. A 3D model was created in SolidWorks and exported as a 3D-printer compatible `.stl` file.
4. The brackets was then printed on the University's Ultimaker 2+ 3D-printers using PLA filament.
5. The final 3D printed part was cleaned for print artifacts using hand tools.

The bracket was printed using fine printer settings(0.15mm layer height, 30% infill) as the tolerances and strength of the socket joint was critical.

Chapter 6

Conclusion

The goal of this thesis was to use mmWave radars for non-contact parameter estimation of a swinging load. This was achieved by performing experiments in a lab environment while using an AWR1843 radar for data acquisition. ROS was used to log and visualize the data. An IWR6843 radar and an IMU was used as reference sensors in order to validate the measurements performed by the AWR1843. The IWR6843 was used to validate the position estimations, while the IMU was used to validate velocity and angle estimations.

The validation of the position estimations were performed only in one dimension, along the radars x-axis. This was due to the IWR6843 using the "High accuracy Level Sensing" firmware, which provided a higher range resolution than the AWR1843. However, this was at the cost of the IWR6843 providing no data for y- and z-direction or velocity. Therefore, the IWR6843 was used as a reference sensor during position estimations in x-direction. Using the proposed method gave a root mean square error of 3 [cm]. For velocity and angle estimations the IMU was used as reference. For velocity estimations this gave a rmse of 0.1268 $[\frac{m}{s}]$ and for angle estimations it gave an rmse of 4.3821 [deg].

6.1 Further work

There is room for improvement for the work done in this thesis. The methods are not fully developed, which causes the results to be less than optimal. Additional work can be performed on the project to improve the performance of the system. A step that could be taken to improve on the quality of the radar data is taking an embedded systems approach. By programming the radars instead of using firmware provided by the manufacturer TI, results can potential be improved. This could significantly increase the performance in terms of both resolution and sample rate.

Given that the methods utilized in this project were improved, the data acquired from the sensors could be used to develop anti-swing control systems. This could for example be used for load handling in offshore environments like monopile installations or ship-to-ship transfers. The measured radar data could be implemented in a crane controller in order to negate the wave-induced heave motion that affects a suspended load at sea.

Appendix A

Code

Key code files are included in this section. The full `radar_pose_detection` package along with other project files are available on GitHub[44].

1. listener_radar.py: coverts radar point cloud data to RViz compatible joint states
2. pile-setup-imu-radar.urdf: URDF file to represent the experimental lab setup
3. readData_IWR6843ISK.py: IWR6843ISK driver, parses serial data, logs to CSV and converts to RViz compatible joint states
4. display_radar_imu.launch: A ROS launch file to start multiple nodes with a specific configuration at the same time.

listener_radar.py

```
1  #!/usr/bin/env python2.7
2
3  ## Converts /imu_iphone quaternions to euler angles for "pile-setup.urdf"
4  ## Sets /joint_states
5
6  import rospy
7  from ti_mmwave_rospkg.msg import RadarScan
8  from std_msgs.msg import Float64
9  from sensor_msgs.msg import JointState
10 from sensor_msgs.msg import Imu
11 from tf.transformations import euler_from_quaternion
12 from math import pi
13 from math import atan
14
15
16 class CommandToJointState:
17     def __init__(self):
18         self.joint_state = JointState()
19         self.joint_state.name.append("base_to_pipe_x_radar")
20         self.joint_state.name.append("base_to_pipe_y_radar")
21         self.joint_state.name.append("base_to_pipe_z_radar")
22         rospy.loginfo("Publishing joint_states for " + ...
               str(self.joint_state.name))
23         self.joint_state.position.append([0.0, 0.0, 0.0])
24         self.joint_state.velocity.append(0.0)
25         self.joint_pub = rospy.Publisher("joint_states", JointState, ...
               queue_size=1)
26         self.command_sub = rospy.Subscriber("/ti_mmwave/radar_scan", ...
               RadarScan, self.command_callback, queue_size=1)
```

```
27
28
29
30      def command_callback(self, msg):
31          # Filter based on  x value and velocity
32          if msg.x < 1.3 and msg.y > -1 and msg.y < 1 and (msg.velocity > 0 ...
                or msg.velocity < 0):
33
34              # Calculate position with atan(x/ pipe length)
35              # msg.x-1 : 1 meter is subtracted as the pile is placed 1 ...
                    meter from the radar
36              self.joint_state.position = (atan(msg.y/0.5), ...
                    -atan((msg.x-1)/0.5), 0)
37              rospy.loginfo("thetaX:" + str(-atan((msg.x-1)/0.5)*(180/pi)) + ...
                    " \tthetaY: " + str(atan(msg.y/0.5)*(180/pi)) + " ...
                    \t\tIntensity: " + str(msg.intensity) + " point_id: " + ...
                    str(msg.point_id) + " velocity: " + str(msg.velocity))
38              #rospy.loginfo_throttle(1, "X: " + str(msg.x) + " Y: " + ...
                    str(msg.y))
39              self.joint_state.header.stamp = rospy.Time.now()
40              # Publish to topic
41              self.joint_pub.publish(self.joint_state)
42
43
44  if __name__ == '__main__':
45      rospy.init_node('command_to_joint_state_radar')
46      command_to_joint_stateX = CommandToJointState()
47      rospy.spin()
```

pile-setup-imu-radar.urdf

```
1   <?xml version="1.0"?>
2   <robot name="materials">
3
4     <material name="silver">
5       <color rgba="0.7 0.7 0.7 1"/>
6     </material>
7
8     <material name="blue">
9       <color rgba="0 0 0.9 1"/>
10    </material>
11
12
13    <material name="grey">
14      <color rgba="0.9 0.9 0.9 1"/>
15    </material>
16
17    <link name="ti_mmwave">
18      <visual>
19        <geometry>
20          <box size="3 0.1 0.1"/>
21        </geometry>
22        <material name="grey"/>
23        <origin xyz="1 0 0.5"/>
24      </visual>
25    </link>
26
27    <joint name="base_to_pipe_x" type="continuous">
28      <parent link="ti_mmwave"/>
29      <child link="steel_pipe_x_dummy"/>
30      <axis xyz="1 0 0"/>
```

```
    <origin xyz="1 0 0.5"/>
  </joint>

  <link name="steel_pipe_x_dummy" />

  <joint name="base_to_pipe_y" type="continuous">
    <parent link="steel_pipe_x_dummy"/>
    <child link="steel_pipe_y_dummy"/>
    <axis xyz="0 1 0"/>
    <origin xyz="0 0 0"/>
  </joint>

  <link name="steel_pipe_y_dummy" />

  <joint name="base_to_pipe_z" type="continuous">
    <parent link="steel_pipe_y_dummy"/>
    <child link="steel_pipe"/>
    <axis xyz="0 0 1"/>
    <origin xyz="0 0 0"/>
  </joint>

  <link name="steel_pipe">
    <visual>
      <geometry>
        <cylinder length="0.7" radius="0.05"/>
      </geometry>
      <origin xyz="0 0 -0.5"/>
      <material name="silver"/>
    </visual>
  </link>

  <joint name="base_to_pipe_x_radar" type="continuous">
    <parent link="ti_mmwave"/>
    <child link="steel_pipe_x_dummy_radar"/>
    <axis xyz="1 0 0"/>
    <origin xyz="1 0 0.5"/>
  </joint>

  <link name="steel_pipe_x_dummy_radar" />

  <joint name="base_to_pipe_y_radar" type="continuous">
    <parent link="steel_pipe_x_dummy_radar"/>
    <child link="steel_pipe_y_dummy_radar"/>
    <axis xyz="0 1 0"/>
    <origin xyz="0 0 0"/>
  </joint>

  <link name="steel_pipe_y_dummy_radar" />

  <joint name="base_to_pipe_z_radar" type="continuous">
    <parent link="steel_pipe_y_dummy_radar"/>
    <child link="steel_pipe_radar"/>
    <axis xyz="0 0 1"/>
    <origin xyz="0 0 0"/>
  </joint>

  <link name="steel_pipe_radar">
    <visual>
      <geometry>
        <cylinder length="0.7" radius="0.05"/>
      </geometry>
```

```
      <origin xyz="0 0 -0.5"/>
      <material name="blue"/>
    </visual>
  </link>

</robot>
```

readData_IWR6843ISK.py

```
import serial
import time
import numpy as np


# ROS imports
import rospy
from ti_mmwave_rospkg.msg import RadarScan
from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
from sensor_msgs.msg import Imu
from math import pi
from math import atan

# ROS joint state class
class CommandToJointState:
    def __init__(self):
        self.joint_state = JointState()
        self.joint_state.name.append("base_to_pipe_x_radar")
        self.joint_state.name.append("base_to_pipe_y_radar")
        self.joint_state.name.append("base_to_pipe_z_radar")
        rospy.loginfo("Publishing joint_states for " + ...
            str(self.joint_state.name))
        self.joint_state.position.append([0.0, 0.0, 0.0])
        self.joint_state.velocity.append(0.0)
        self.joint_pub = rospy.Publisher("joint_states", JointState, ...
            queue_size=1)









# Change the configuration file name
configFileName = 'high_accuracy_demo_68xx.cfg'

CLIport = {}
Dataport = {}
byteBuffer = np.zeros(2**15,dtype = 'uint8')
byteBufferLength = 0;


# ------------------------------------------------------------------

# Function to configure the serial ports and send the data from
# the configuration file to the radar
def serialConfig(configFileName):

    global CLIport
    global Dataport

    # Linux
```

```
    CLIport = serial.Serial('/dev/ttyUSB0', 115200)
    Dataport = serial.Serial('/dev/ttyUSB1', 921600)

    # Read the configuration file and send it to the board
    config = [line.rstrip('\r\n') for line in open(configFileName)]
    for i in config:
        CLIport.write((i+'\n').encode())
        print(i)
        time.sleep(0.01)

    return CLIport, Dataport

# ------------------------------------------------------------------

# Function to parse the data inside the configuration file
def parseConfigFile(configFileName):
    configParameters = {} # Initialize an empty dictionary to store the ...
       configuration parameters

    # Read the configuration file and send it to the board
    config = [line.rstrip('\r\n') for line in open(configFileName)]
    for i in config:

        # Split the line
        splitWords = i.split(" ")

        # Hard code the number of antennas, change if other configuration ...
           is used
        numRxAnt = 4
        numTxAnt = 3

        # Get the information about the profile configuration
        if "profileCfg" in splitWords[0]:
            startFreq = int(float(splitWords[2]))
            idleTime = int(splitWords[3])
            rampEndTime = float(splitWords[5])
            #freqSlopeConst = float(splitWords[8])
            numAdcSamples = int(splitWords[10])
            numAdcSamplesRoundTo2 = 1;

            while numAdcSamples > numAdcSamplesRoundTo2:
                numAdcSamplesRoundTo2 = numAdcSamplesRoundTo2 * 2;

            digOutSampleRate = int(splitWords[11]);

        # Get the information about the frame configuration
        elif "frameCfg" in splitWords[0]:

            chirpStartIdx = int(splitWords[1]);
            chirpEndIdx = int(splitWords[2]);
            numLoops = int(splitWords[3]);
            numFrames = int(splitWords[4]);
            framePeriodicity = int(splitWords[5]);


    # Combine the read data to obtain the configuration parameters
    numChirpsPerFrame = (chirpEndIdx - chirpStartIdx + 1) * numLoops
    configParameters["numDopplerBins"] = numChirpsPerFrame / numTxAnt
    configParameters["numRangeBins"] = numAdcSamplesRoundTo2
    #configParameters["rangeResolutionMeters"] = (3e8 * digOutSampleRate * ...
       1e3) / (2 * freqSlopeConst * 1e12 * numAdcSamples)
```

```
    configParameters["rangeResolutionMeters"] = 1
    #configParameters["rangeIdxToMeters"] = (3e8 * digOutSampleRate * 1e3) ...
        / (2 * freqSlopeConst * 1e12 * configParameters["numRangeBins"])
    configParameters["rangeIdxToMeters"] = 1
    configParameters["dopplerResolutionMps"] = 3e8 / (2 * startFreq * 1e9 ...
        * (idleTime + rampEndTime) * 1e-6 * ...
        configParameters["numDopplerBins"] * numTxAnt)
    #configParameters["maxRange"] = (300 * 0.9 * digOutSampleRate)/(2 * ...
        freqSlopeConst * 1e3)
    configParameters["maxRange"] = 1
    configParameters["maxVelocity"] = 3e8 / (4 * startFreq * 1e9 * ...
        (idleTime + rampEndTime) * 1e-6 * numTxAnt)

    return configParameters

# ------------------------------------------------------------------

# Funtion to read and parse the incoming data
def readAndParseData14xx(Dataport, configParameters):
    global byteBuffer, byteBufferLength

    # Constants
    OBJ_STRUCT_SIZE_BYTES = 12;
    BYTE_VEC_ACC_MAX_SIZE = 2**15;
    MMWDEMO_UART_MSG_DETECTED_POINTS = 1;
    MMWDEMO_UART_MSG_RANGE_PROFILE   = 2;
    maxBufferSize = 2**15;
    magicWord = [2, 1, 4, 3, 6, 5, 8, 7]

    # Initialize variables
    magicOK = 0 # Checks if magic number has been read
    dataOK = 0 # Checks if the data has been read correctly
    frameNumber = 0
    detObj = {}

    x1 = 0
    x2 = 0
    x3 = 0

    # Global for reusing old value if new is not good enough
    global bestX


    readBuffer = Dataport.read(Dataport.in_waiting)
    byteVec = np.frombuffer(readBuffer, dtype = 'uint8')
    byteCount = len(byteVec)

    # Check that the buffer is not full, and then add the data to the buffer
    if (byteBufferLength + byteCount) < maxBufferSize:
        byteBuffer[byteBufferLength:byteBufferLength + byteCount] = ...
            byteVec[:byteCount]
        byteBufferLength = byteBufferLength + byteCount

    # Check that the buffer has some data
    if byteBufferLength > 16:

        # Check for all possible locations of the magic word
        possibleLocs = np.where(byteBuffer == magicWord[0])[0]

        # Confirm that is the beginning of the magic word and store the ...
            index in startIdx
```

```
163         startIdx = []
164         for loc in possibleLocs:
165             check = byteBuffer[loc:loc+8]
166             if np.all(check == magicWord):
167                 startIdx.append(loc)
168 
169         # Check that startIdx is not empty
170         if startIdx:
171 
172             # Remove the data before the first start index
173             if startIdx[0] > 0 and startIdx[0] < byteBufferLength:
174                 byteBuffer[:byteBufferLength-startIdx[0]] = ...
                        byteBuffer[startIdx[0]:byteBufferLength]
175                 byteBuffer[byteBufferLength-startIdx[0]:] = ...
                        np.zeros(len(byteBuffer[byteBufferLength-startIdx[0]:]),dtype ...
                        = 'uint8')
176                 byteBufferLength = byteBufferLength - startIdx[0]
177 
178             # Check that there have no errors with the byte buffer length
179             if byteBufferLength < 0:
180                 byteBufferLength = 0
181 
182             # word array to convert 4 bytes to a 32 bit number
183             word = [1, 2**8, 2**16, 2**24]
184 
185             # Read the total packet length
186             totalPacketLen = np.matmul(byteBuffer[12:12+4],word)
187             # Check that all the packet has been read
188             if (byteBufferLength >= totalPacketLen) and (byteBufferLength ...
                    != 0):
189                 magicOK = 1
190     #print(f"magicOK = {magicOK}")
191 
192     # If magicOK is equal to 1 then process the message
193     if magicOK:
194         # word array to convert 4 bytes to a 32 bit number
195         word = [1, 2**8, 2**16, 2**24]
196 
197         # Initialize the pointer index
198         idX = 0
199 
200         # Read the header
201         magicNumber = byteBuffer[idX:idX+8]
202         idX += 8
203         version = format(np.matmul(byteBuffer[idX:idX+4],word),'x')
204         idX += 4
205         totalPacketLen = np.matmul(byteBuffer[idX:idX+4],word)
206         idX += 4
207         platform = format(np.matmul(byteBuffer[idX:idX+4],word),'x')
208         idX += 4
209         frameNumber = np.matmul(byteBuffer[idX:idX+4],word)
210         idX += 4
211         timeCpuCycles = np.matmul(byteBuffer[idX:idX+4],word)
212         idX += 4
213         numDetectedObj = np.matmul(byteBuffer[idX:idX+4],word)
214         idX += 4
215         numTLVs = np.matmul(byteBuffer[idX:idX+4],word)
216         idX += 4
217         #idX += 4
218         #print(f"magicNumber = {magicNumber} \t version = {version} \t ...
            totalPacketLen = {totalPacketLen} \t platform = {platform} \t ...
```

```
            frameNumber = {frameNumber} ")
219         #print(f"timeCpuCycles = {timeCpuCycles} \t\t numDetectedObj = ...
               {numDetectedObj} \t numTLVs = {numTLVs} \t\t idX = {idX}")
220         #np.savetxt("bytes.txt", byteBuffer)
221
222         # Read the TLV messages
223         for tlvIdx in range(numTLVs):
224
225             # word array to convert 4 bytes to a 32 bit number
226             word = [1, 2**8, 2**16, 2**24]
227
228             # Check the header of the TLV message
229             tlv_type = np.matmul(byteBuffer[idX:idX+4],word)
230             idX += 4
231             tlv_length = np.matmul(byteBuffer[idX:idX+4],word)
232             idX += 4
233             #print(f"tlv_type = {tlv_type} \t ...
                   MMWDEMO_UART_MSG_DETECTED_POINTS = ...
                   {MMWDEMO_UART_MSG_DETECTED_POINTS}")
234             # Read the data depending on the TLV message
235             if tlv_type == MMWDEMO_UART_MSG_DETECTED_POINTS:
236
237                 # word array to convert 4 bytes to a 16 bit number
238                 word = [1, 2**8]
239                 tlv_numObj = np.matmul(byteBuffer[idX:idX+2],word)
240                 idX += 2
241                 tlv_xyzQFormat = 2**np.matmul(byteBuffer[idX:idX+2],word)
242                 idX += 2
243
244                 # Initialize the arrays
245                 rangeIdx = np.zeros(numDetectedObj,dtype = 'int16')
246                 dopplerIdx = np.zeros(numDetectedObj,dtype = 'int16')
247                 peakVal = np.zeros(numDetectedObj,dtype = 'int16')
248                 x = np.zeros(numDetectedObj,dtype = 'int16')
249                 y = np.zeros(numDetectedObj,dtype = 'int16')
250                 z = np.zeros(numDetectedObj,dtype = 'int16')
251                 #print(f"tlv_numObj = {tlv_numObj}")
252                 for objectNum in range(numDetectedObj):
253
254                     # Read the data for each object
255                     rangeIdx[objectNum] =  ...
                           np.matmul(byteBuffer[idX:idX+2],word)
256                     idX += 2
257                     dopplerIdx[objectNum] = ...
                           np.matmul(byteBuffer[idX:idX+2],word)
258                     idX += 2
259                     peakVal[objectNum] = np.matmul(byteBuffer[idX:idX+2],word)
260                     idX += 2
261                     x[objectNum] = np.matmul(byteBuffer[idX:idX+2],word)
262                     idX += 2
263                     y[objectNum] = np.matmul(byteBuffer[idX:idX+2],word)
264                     idX += 2
265                     z[objectNum] = np.matmul(byteBuffer[idX:idX+2],word)
266                     idX += 2
267                     #print(f"rangeIdx[{objectNum}] = {rangeIdx[objectNum]} ...
                           \t dopplerIdx[{objectNum}] = ...
                           {dopplerIdx[objectNum]} \t peakVal[{objectNum}] = ...
                           {peakVal[objectNum]} \t x[{objectNum}] = ...
                           {x[objectNum]} \t y[{objectNum}] = {y[objectNum]} ...
                           \t z[{objectNum}] = {z[objectNum]} \t")
268
```

```
269                     # x1: 6 bytes: rangeIDX1, rangeIDX2*256, x*65536
270                     # x2: 6 bytes: peakval1, peakval2*256, y*65536
271                     # x3: 6 bytes: dopplerIdx1, dopplerIdx2*256, z*65536
272
273                     x1 = rangeIdx[objectNum] + x[objectNum]*65536
274                     x2 = peakVal[objectNum] + y[objectNum]*65536
275                     x3 = dopplerIdx[objectNum] + z[objectNum]*65536
276
277                     # Multiply by 1.36 and divide by 1048576 to get number ...
                            to meters
278
279                     x1 = round((x1*1.36)/1048576, 4)
280                     x2 = round((x2*1.36)/1048576, 4)
281                     x3 = round((x3*1.36)/1048576, 4)
282
283
284                     # Check which range value is closest to 1 meter
285                     strongestPeaks = [x1, x2, x3]
286                     myNumber = 1
287                     closestValue = min(strongestPeaks, key=lambda ...
                            x:abs(x-myNumber))
288
289                     # For average
290                     oldX = bestX
291
292
293                     # Check if new value is usable
294                     if closestValue > 1.5 or closestValue < 0.7:
295                         bestX = bestX
296                     else:
297                         bestX =  closestValue
298
299                     #Average with previous value
300                     avg = ((oldX + bestX)/2)
301
302                     #avg = -atan((avg-1)/1)
303
304
305                     print(f"x1: {x1} \t x2: {x2} \t x3: {x3} \t avg: ...
                            {oldX} Deg: {round(bestX*(180/pi),4)}")
306
307                     # Calculate position with atan(x/ pipe length)
308                     # msg.x-1 : 1 meter is subtracted as the pile is ...
                            placed 1 meter from the radar, length of pendulum ...
                            is 1 meter
309                     command_to_joint_stateX.joint_state.position = (0, ...
                            avg, 0)
310
311                     command_to_joint_stateX.joint_state.header.stamp = ...
                            rospy.Time.now()
312                     # Publish to topic
313                     command_to_joint_stateX.joint_pub.publish(command_to_joint_stateX.jo
314
315                 dataOK = 1
316
317
318         # Remove already processed data
319         if idX > 0 and byteBufferLength > idX:
320             shiftSize = totalPacketLen
321
322             byteBuffer[:byteBufferLength - shiftSize] = ...
```

```
                byteBuffer[shiftSize:byteBufferLength]
            byteBuffer[byteBufferLength - shiftSize:] = ...
                np.zeros(len(byteBuffer[byteBufferLength - ...
                shiftSize:]),dtype = 'uint8')
            byteBufferLength = byteBufferLength - shiftSize

            # Check that there are no errors with the buffer length
            if byteBufferLength < 0:
                byteBufferLength = 0


    return dataOK, frameNumber, x1

# --------------------------------------------------------------

# Funtion to update the data and display in the plot
def update():

    dataOk = 0
    global detObj
    x = []
    y = []



    # Read and parse the received data
    dataOk, frameNumber, x1 = readAndParseData14xx(Dataport, configParameters)
    #print(f"dataOK = {dataOk}")
    #if dataOk and x1:
    #    print(detObj)
    #    x = -detObj["x"]
    #    y = detObj["y"]
        #plt.clf()
        #plt.axis([0, 4, 0, 1])
        #plt.scatter(x1, 0.5)
        #plt.pause(0.01)
        #s.setData(x,y)
        #QtGui.QApplication.processEvents()

    #return dataOk


# -------------------------    MAIN   ...
   -----------------------------------------

# Configurate the serial port
CLIport, Dataport = serialConfig(configFileName)

# Get the configuration parameters from the configuration file
configParameters = parseConfigFile(configFileName)

# START QtAPPfor the plot
"""
app = QtGui.QApplication([])

# Set the plot
pg.setConfigOption('background','w')
win = pg.GraphicsWindow(title="2D scatter plot")
p = win.addPlot()
p.setXRange(-0.5,0.5)
p.setYRange(0,1.5)
```

```
p.setLabel('left',text = 'Y position (m)')
p.setLabel('bottom', text= 'X position (m)')
s = p.plot([],[],pen=None,symbol='o')

win.show()
app.exec_()
"""
#plt.show()

#plt.axis([0, 15, 0, 1])


# Main loop
detObj = {}
frameData = {}
currentIndex = 0

rospy.init_node('command_to_joint_state_radar')
command_to_joint_stateX = CommandToJointState()
#rospy.spin()


while True:
    try:
        # Update the data and check if the data is okay
        dataOk = update()


        if dataOk:
            # Store the current frame into frameData
            frameData[currentIndex] = detObj
            currentIndex += 1

        time.sleep(0.033) # Sampling frequency of 30 Hz

    # Stop the program and close everything if Ctrl + c is pressed
    except KeyboardInterrupt:
        CLIport.write(('sensorStop\n').encode())
        CLIport.close()
        Dataport.close()
        #win.close()
        break
```

display_radar_imu.launch

```
<launch>

  <arg name="model" default="$(find ...
      radar_pose_detection)/urdf/pile-setup-imu-radar.urdf"/>
  <arg name="gui" default="false" />
  <arg name="rvizconfig" default="$(find ...
      radar_pose_detection)/rviz/urdf.rviz" />

  <!-- Input arguments -->
  <arg name="device" value="1642" doc="TI mmWave sensor device type [1443, ...
      1642]"/>
  <arg name="config" value="2d" doc="TI mmWave sensor device configuration ...
      [3d_best_range_res (not supported by 1642 EVM), 2d_best_range_res]"/>
  <arg name="max_allowed_elevation_angle_deg" default="90" doc="Maximum ...
      allowed elevation angle in degrees for detected object data [0 > ...
      value >= 90]}"/>
```

```
11    <arg name="max_allowed_azimuth_angle_deg" default="90" doc="Maximum ...
          allowed azimuth angle in degrees for detected object data [0 > value ...
          >= 90]}"/>
12
13    <!-- Static transform from map to base_radar_link for visualization of ...
          stand-alone mmWave sensor using Rviz -->
14    <node pkg="tf" type="static_transform_publisher" ...
          name="static_tf_map_to_base_radar_link" args="0 0 0 0 0 0 ...
          ti_mmwave_pcl ti_mmwave 30"/>
15
16
17    <param name="robot_description" command="$(find xacro)/xacro $(arg ...
          model)" />
18
19    <node if="$(arg gui)" name="joint_state_publisher" ...
          pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" />
20    <node name="robot_state_publisher" pkg="robot_state_publisher" ...
          type="robot_state_publisher" />
21    <node name="listener_imu" pkg="radar_pose_detection" ...
          type="listener_imu.py" />
22    <node name="listener_radar" pkg="radar_pose_detection" ...
          type="listener_radar.py" />
23    <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" ...
          required="true" />
24
25  </launch>
```

Appendix B

Other

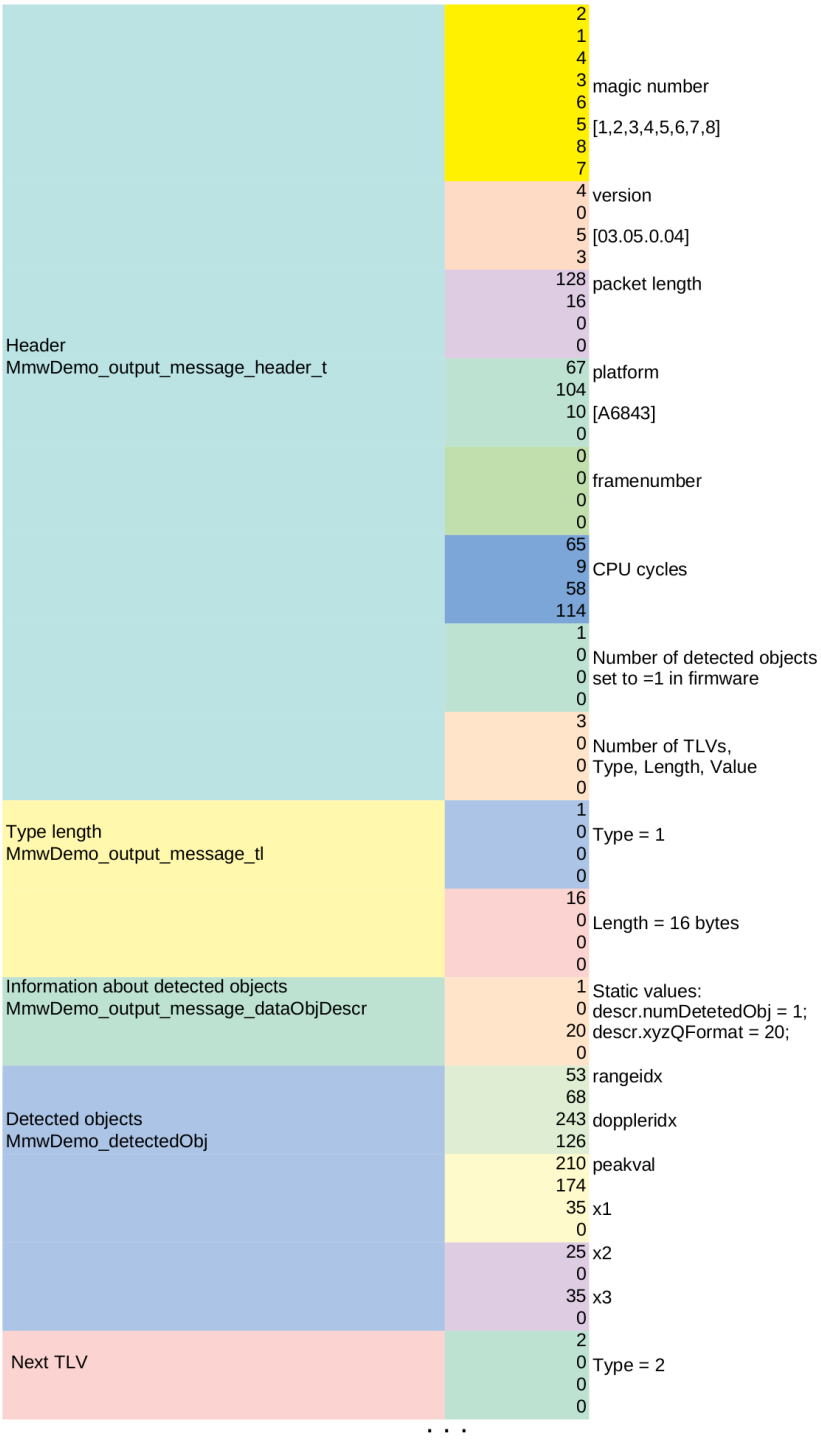

Figure B.1: Data format with partial parsing. The first column contains the data type from the C source code. The second column contains raw data in bytes, while the third column is annotations.

Bibliography

[1] Kok-Sing Lim et al. *Vibration Mode Analysis for a Suspension Bridge by Using Low-Frequency Cantilever-Based FBG Accelerometer Array*. 2021. DOI: 10.1109/TIM.2020.3018578.

[2] Werner Scheiblhofer et al. *A high-precision long range cooperative radar system for rail crane distance measurement*. 2014. DOI: 10.1109/EuRAD.2014.6991268.

[3] Reinhard Feger et al. "A 77-GHz Cooperative Radar System Based on Multi-Channel FMCW Stations for Local Positioning Applications." In: *IEEE Transactions on Microwave Theory and Techniques* 61.1 (2013), pp. 676–684. DOI: 10.1109/TMTT.2012.2227781.

[4] *Rain and Fog Challenge the Sensors - Sensible 4*. https://sensible4.fi/technology/rain-and-fog-challenge-the-sensors/. (Accessed on 05/24/2022).

[5] *IEEE Standard Letter Designations for Radar-Frequency Bands*. 2020. DOI: 10.1109/IEEESTD.2020.8999849.

[6] Matti Autioniemi Chris Händel Heikki Konttaniemi. *State-of-the-Art Review on Automotive Radars and Passive Radar Reflectors*. https://www.lapinamk.fi/loader.aspx?id=983f39aa-c97f-4f2d-bb39-abe006c6bad3. (Accessed on 05/04/2022). June 2018.

[7] Liyana Ramli et al. *Control strategies for crane systems: A comprehensive review*. 2017. DOI: https://doi.org/10.1016/j.ymssp.2017.03.015. URL: https://www.sciencedirect.com/science/article/pii/S0888327017301425.

[8] Zhiyu Jang. *Installation of offshore wind turbines: A technical review*. https://www.sciencedirect.com/science/article/pii/S1364032120308601. (Accessed on 05/04/2022). 2021.

[9] Yibin Hu et al. "Microwave Radar Based Estimation of the Sway Angle of Payload in Overhead Crane System." In: *2021 Telecoms Conference (ConfTELE)*. 2021, pp. 1–6. DOI: 10.1109/ConfTELE50222.2021.9435569.

[10] Manizhe Rahchamani et al. "Developing and Evaluating a Low-Cost Tracking Method based on a Single Camera and a Large Marker." In: *2018 25th National and 3rd International Iranian Conference on Biomedical Engineering (ICBME)*. 2018, pp. 1–5. DOI: 10.1109/ICBME.2018.8703592.

[11] Mitesh Patel and Philip Ferguson. "Tracking and Estimation of a Swaying Payload Using a LiDAR and an Extended Kalman Filter." In: *2021 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. 2021, pp. 1–7. DOI: 10.1109/ROSE52750.2021.9611771.

[12] Hanafy M. Omar. "Hardware-In-the-Loop Simulation of Time-Delayed Anti-Swing Controller for Quadrotor with Suspended Load." In: *Applied Sciences* 12.3 (2022). ISSN: 2076-3417. DOI: 10.3390/app12031706. URL: https://www.mdpi.com/2076-3417/12/3/1706.

[13] *Infineon-Radar FAQ-PI-v02_ 00-EN.pdf*. https://www.infineon.com/dgdl/Infineon-Radar%20FAQ-PI-v02_00-EN.pdf?fileId=5546d46266f85d6301671c76d2a00614. (Accessed on 05/26/2022).

[14] *Radartutorial*. https://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html. (Accessed on 05/20/2022).

[15] TI. *The fundamentals of millimeter wave radar sensors (Rev. A)*. https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf?ts=1651662278294&ref_url=https%253A%252F%252Fwww.google.com%252F. (Accessed on 05/04/2022).

[16] *Gauss_ History_ FFT.pdf*. https://www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf. (Accessed on 05/26/2022).

[17] *Zoom FFT*. https://www.arc.id.au/ZoomFFT.html. (Accessed on 05/26/2022).

[18] Joacim Dybedal, Atle Aalerud, and Geir Hovland. "Embedded Processing and Compression of 3D Sensor Data for Large Scale Industrial Environments." In: *Sensors* 19 (Feb. 2019), p. 636. DOI: 10.3390/s19030636.

[19] *melodic - ROS Wiki*. https://wiki.ros.org/melodic. (Accessed on 05/19/2022).

[20] *What is the Difference Between rviz and Gazebo? – Automatic Addison*. https://automaticaddison.com/what-is-the-difference-between-rviz-and-gazebo/. (Accessed on 05/03/2022).

[21] *URDF Primer - MATLAB & Simulink - MathWorks Nordic*. https://se.mathworks.com/help/physmod/sm/ug/urdf-model-import.html. (Accessed on 05/03/2022).

[22] *urdf/Tutorials/Building a Movable Robot Model with URDF - ROS Wiki*. http://wiki.ros.org/urdf/Tutorials/Building%20a%20Movable%20Robot%20Model%20with%20URDF. (Accessed on 05/04/2022).

[23] *Maths - Rotations - Martin Baker*. https://www.euclideanspace.com/maths/geometry/rotations/index.htm. (Accessed on 05/26/2022).

[24] Moti Ben-Ari. *A Tutorial on Euler Angles and Quaternions*. https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf. (Accessed on 05/02/2022). May 2022.

[25] *Distributions — ROS 2 Documentation: Galactic documentation*. http://docs.ros.org/en/galactic/Releases.html. (Accessed on 05/04/2022).

[26] *Distributions - ROS Wiki*. https://wiki.ros.org/Distributions. (Accessed on 05/04/2022).

[27] *AWR1843BOOST Evaluation board | TI.com*. https://www.ti.com/tool/AWR1843BOOST. (Accessed on 05/24/2022).

[28] *IWR6843ISK Evaluation board | TI.com*. https://www.ti.com/tool/IWR6843ISK. (Accessed on 05/24/2022).

[29] *Industrial Toolbox*. https://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg__VLyFKFf__LATEST. (Accessed on 03/18/2022).

[30] *mmWave Sensors - TI | Mouser*. https://no.mouser.com/new/texas-instruments/ti-mmwave-sensor/. (Accessed on 05/23/2022).

[31] *GitHub - radar-lab/ti_mmwave_rospkg: TI mmWave radar ROS driver (with sensor fusion and hybrid)*. https://github.com/radar-lab/ti_mmwave_rospkg. (Accessed on 03/15/2022).

[32] *GitHub - Claud1234/ti_mmwave_rospkg: TI mmWave radar ROS driver (with sensor fusion and hybrid)*. https://github.com/Claud1234/ti_mmwave_rospkg. (Accessed on 03/15/2022).

[33] *High Accuracy Visualizer*. https://dev.ti.com/gallery/view/4768107/High_Accuracy_Visualizer/ver/2.0.0/. (Accessed on 03/15/2022).

[34] *GitHub - ibaiGorordo/IWR1443-Read-Data-Python-MMWAVE-SDK-1: Python program to read and plot the position of the reflected points from the IWR1443 and the AWR1443. It works both with Windows and Raspberry Pi*. https://github.com/ibaiGorordo/IWR1443-Read-Data-Python-MMWAVE-SDK-1. (Accessed on 03/15/2022).

[35] *Bosch's 6-Axis IMU in the Apple iPhone X - System Plus Consulting;* https://www.systemplus.fr/wp-content/uploads/2018/01/SP18382-Bosch-IMU-in-iPhone-X_flyer-1.pdf. (Accessed on 05/09/2022).

[36] *Getting Processed Device-Motion Data | Apple Developer Documentation*. https://developer.apple.com/documentation/coremotion/getting_processed_device-motion_data. (Accessed on 05/03/2022).

[37] *How do refresh rates work for monitors?* https://insights.samsung.com/2022/03/07/how-does-refresh-rate-work-for-monitors/. (Accessed on 05/04/2022).

[38] *SensorLog on the App Store*. https://apps.apple.com/us/app/sensorlog/id388014573. (Accessed on 05/09/2022).

[39] *rosbag - ROS Wiki*. http://wiki.ros.org/rosbag. (Accessed on 05/26/2022).

[40] *mymodelrobot setup URDF*. http://www.mymodelrobot.appspot.com/6304284522053632. (Accessed on 03/16/2022).

[41] *How DBSCAN works and why should we use it? | by Kelvin Salton do Prado | Towards Data Science*. https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80. (Accessed on 05/26/2022).

[42] *mmwave-swinging-load/matlab_plots/logss/25.03-experiments at main · teover/mmwave-swinging-load*. https://github.com/teover/mmwave-swinging-load/tree/main/matlab_plots/logss/25.03-experiments. (Accessed on 05/26/2022).

[43] *mmwave-swinging-load/matlab_plots/logss/resolution_logs at main · teover/mmwave-swinging-load*. https://github.com/teover/mmwave-swinging-load/tree/main/matlab_plots/logss/resolution_logs. (Accessed on 05/26/2022).

[44] *teover/mmwave-swinging-load: Master thesis repo: ROS package mmWave radar tracking of swinging, log files and media for report*. https://github.com/teover/mmwave-swinging-load. (Accessed on 05/08/2022).