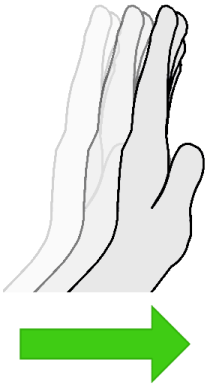
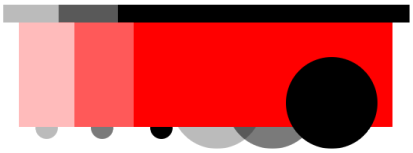
CHALMERS
UNIVERSITY OF TECHNOLOGY





Human Control of Mobile Robots Using Hand Gestures

A Proof of Concept Implementation

Bachelor's thesis in Electrical Engineering

CHRISTIAN BERKIUS
MAX BUCK
JONATAN GUSTAFSSON
MARTIN KAUPPINEN

Supervisor: Mohammad Ali Nazari
Co-supervisor: Canan Aydogdu
Examiner: Henk Wymeersch

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Bachelor's thesis 2018:EENX15-18-33

Human Control of Mobile Robots Using Hand Gestures

A Proof of Concept Implementation

Christian Berkius
Max Buck
Jonatan Gustafsson
Martin Kauppinen

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Human Gesture Control of Mobile Robots Using Hand Gestures
A Proof of Concept Implementation

CHRISTIAN BERKIUS
MAX BUCK
JONATAN GUSTAFSSON
MARTIN KAUPPINEN

Supervisor: Mohammad Ali Nazari, Department of Electrical Engineering
Co-supervisor: Canan Aydogdu, Department of Electrical Engineering
Examiner: Henk Wymeersch, Department of Electrical Engineering

Bachelor's Thesis 2018:EENX15-18-33
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Abstract

The field of autonomous systems is receiving a great attention from both academia and industry in the recent years. Together with the advances in artificial intelligence (AI) and signal processing, this has led to dreams of motion controlled autonomous vehicles and robots. This bachelor's thesis aims at the implementation of a prototype for human control of mobile robots using gesture recognition via radar. The project paves the way toward more complex projects in facilitating applications such as autonomous warehouses and delivery systems. In this project, we use a radar for recognizing hand gestures, and for localizing a mobile robot. A machine learning algorithm is used to classify the gestures. The robot is supposed to follow the control commands manually issued by the operator, and drive autonomously from a starting point to a destination. Although this project considers a basic implementation as a proof of concept, it is scalable and flexible to accommodate more robots and more complex tasks.

Keywords: Autonomous Systems, Machine Learning, Gesture Recognition, Radar, Localization.

Sammandrag

Forskningsområdet autonoma system har under senare år varit en fokuspunkt för utveckling, både i akademiska världen och industrin. Detta, tillsammans med framsteg inom artificiell intelligens (AI) och signal behandling, har lett till förhoppningar om geststyrda och autonoma fordon samt robotar. Denna kandidatuppsats kommer försöka att implementera en prototyp av ett system där en människa kan kontrollera mobila robotar med gestigenkänning via radar. Detta kan föranleda och förenkla för komplexa projekt som applicerar tekniken, som till exempel i ett autonomt varuhus eller utlämningstjänst. Detta projekt kommer använda en radar för att kunna känna igen en gest med handen men också för att lokalisera roboten. Maskininlärning används för att klassificera gesterna. Roboten skall kunna följa kommandon som manuellt ges av användaren, och kunna köra autonomt mellan en start och slut position. Trots att detta projekt endast avser en grund implementation i syfte att bevisa konceptet, så är det mycket flexibelt finns det stor möjlighet att införa fler robotar och mer komplexa uppgifter.

Nyckelord: Gestigenkänning, Maskininlärning, Autonoma system, Radar, Lokalisering.

Acknowledgements

We would like to thank Mohammad Ali Nazari and Canan Aydogdu for their supervision and guidance in this project; without their help this would not have been possible. We would also like to thank Henk Wymeersch for the guidance he provided and his part as examiner.

Christian Berkius, Max Buck, Jonatan Gustafsson, Martin Kauppinen
Gothenburg, May 2018

Contents

List of Figures

List of Tables

Glossary

ADC
Analog-to-Digital Converter. Samples an analog signal, producing a digital signal to be used in digital signal processing. 5, 13, 14

API
Application Programming Interface. 13, 16

chirp
Signal emitted by radar module. 3

FFT
Fast Fourier Transform, an algorithm that samples a signal over time and separates it into its frequency components. 11

FMCW
Frequency Modulated Continuous Wave. 3, 11, 37

frame
A sequence of chirps. 4

Q-format
A fixed point number format where the number of fractional bits can be specified. 10, 21, 22, 33

ROS
Robot Operating System, which controls the robot in the project. 15, 16, 25, 27

ROSARIA
A node in the ROS network through which the robot can communicate with the rest of the computers. 16, 25, 27

SDK
Software Development Kit. 16

TI
Texas Instruments. 11, 20, 33

TLV
Tag-Length-Value. A data structure containing an identifying tag, the length of the structure, and the value of the data it represents. 21

1

Introduction

In today's modern society, there is a growing desire for more advanced and efficient technologies. Autonomy in systems and power plants is one area that has seen ongoing research for many years [1], and is today at the forefront of autonomous tasks and applications [2] [3].

In addition, with the advancements in signal processing and perception techniques, several technologies and algorithms have been proposed for sensing the environment, increasing the awareness, and localizing objects in complex systems. With this, the systems would be aware of the objects in their immediate vicinity and can take appropriate actions depending on the specific tasks they want to perform.

There are several devices which can be used for sensing the environment. We consider radar as a technology which can be used for short-range perception of its surroundings, relying on low-power communication [4] [5]. Moreover, the radar is able to detect moving objects, which can be a good choice for recognizing gestures to be issued by an operator commanding the whole control system.

This could also be used to facilitate tasks in systems where it is difficult to gain physical access the system itself, but that still require people to operate them.

A use case of such a human-controlled system is for users with impaired physiology. If the user has impaired motor skills and as such may have trouble using traditional keyboard and mouse setups or other controllers relying on fine motor skills, it could be used to steer a system with other types of movement or with other body parts. If it would be cumbersome to implement a conventional control system, then one based on gesture recognition could be beneficial for all kinds of users.

Applications within warehouse industries are also possible. For example, a person from an office or on the warehouse floor, can remotely direct robots to loading stations to retrieve items.

This can of course be done without gestures, but it increases the facility of the system to send commands only by gestures.

This project aims to implement a radar gesture recognition system. It should be able to steer one or a small group of mobile robots by pairing the different gestures

to specific tasks, such as recalling all robots or moving them to a predetermined location. To be able to successfully implement these functions, the following problems need to be solved: the gesture recognition needs to have good accuracy, the robots need a control system to be able to act on gestures, the robots need to be connected to the the radar system, and a localization system has to be built.

1.1 Scope

The project's scope is to implement a system with two major parts. The first part is recognizing hand gestures via machine learning fed with data from a radar. The second part is getting robots to perform predefined tasks. When this is achieved further parts will be added, such as localization of the robots using radar.

To make the system feasible to implement in an acceptable time, a number of predetermined limitations were set. The system is able to recognize four gestures. These primary gestures are simple in nature. The complexity of the tasks of the robots will be kept simple as well. Movement according to gestures is the first goal. Lastly, the system will focus on a single robot, but it is easily scalable and can be used with more robots. This scope will greatly increase the chances that the project will stay focused on the system as a whole and a finished product.

2

Theory

This project incorporates three different areas of research: communications and localization, perception and learning, as well as robotics and software engineering. This chapter explains the necessary information to understand the method and results of this study. For the purpose of communications, we use the radar technology which is explained first, followed by the learning algorithm for gesture recognition, and the theory for control of mobile robots.

2.1 Radar

When talking about radar technologies there are a few terms one should know. One such term is Doppler, which in radar terms denote frequency shift due to velocity in radial direction. Likewise when one talks of range it is the distance from the target.

2.1.1 Frequency Modulated Continuous Waves

FMCW radar uses electromagnetic signals with high frequency to detect objects in front of the radar. The frequency of each signal increases linearly over time, hence the name frequency modulated. These electromagnetic waves are called *chirps* [6]. The total frequency span of a chirp is called the bandwidth. When the chirp ends, the frequency fluctuates in an unregulated manner.

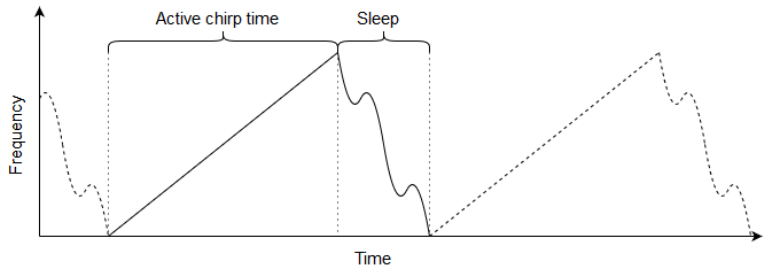


Figure 2.1: Visualization of a chirp wave

A sequence of chirps form a *frame*. When the AWR1642 sends out a signal, it is either in the form of a frame or one continuous signal.

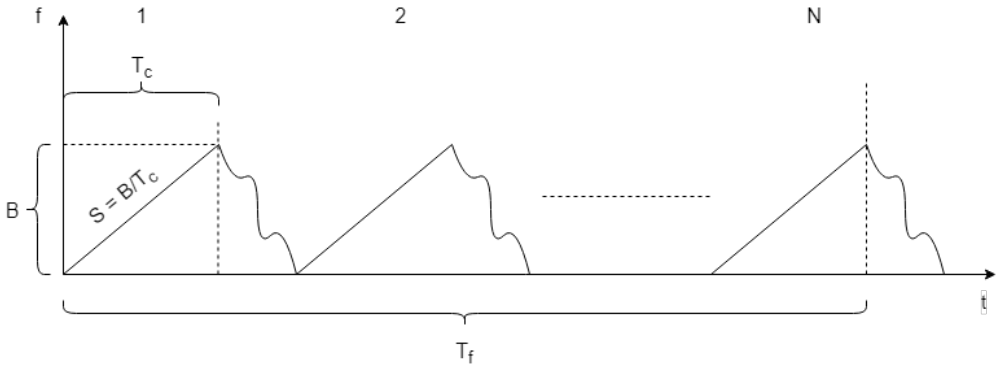


Figure 2.2: A frame with the variables marked.

2.1.2 General Radar Equations

This section provides an insight into four basic but essential equations that determine some of the theoretical capabilities of the radar. These are: its maximum range, the minimum distance required to resolve two objects as separate entities, the maximum velocity of an object in front of the radar, and the minimum velocity difference required to resolve two objects as separate entities. These properties have been illustrated in Figures 2.3, 2.4, 2.5, and 2.6. For a visual representation of some of the variables used in the equations, see Figure 2.2. In the following sub-sections these properties will be explained further.

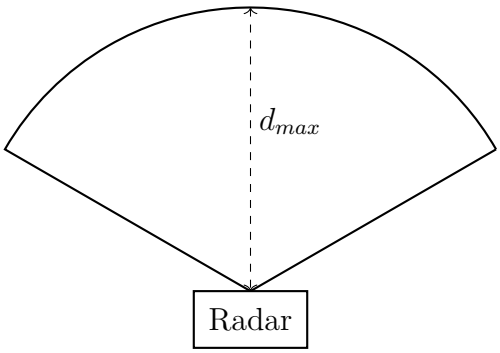


Figure 2.3: The longest possible distance between the radar and a detected object.

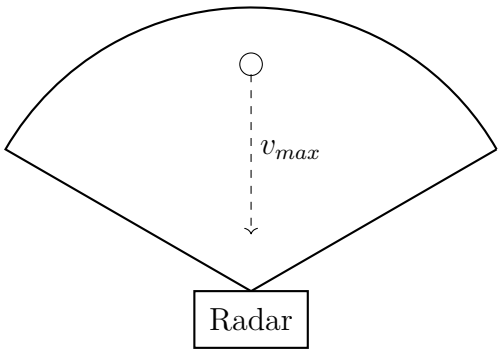


Figure 2.4: The highest possible velocity of a detected object.

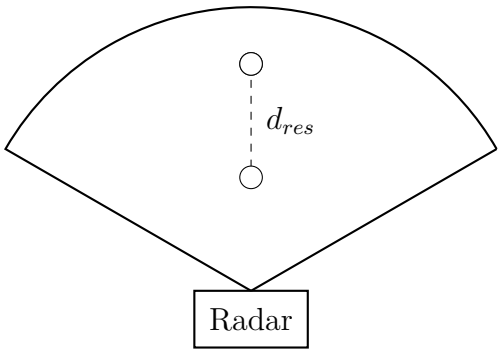


Figure 2.5: The shortest possible distance between two objects for them to be resolved as two different reflections by the radar.

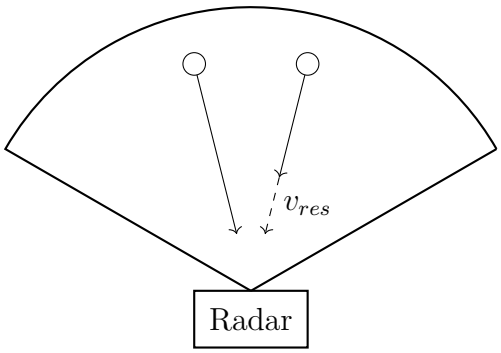


Figure 2.6: The lowest possible difference between two objects' speeds for them to be resolved as two different reflections by the radar.

2.1.3 Maximum Radar Range

The maximum detection range (d_{max}) of the radar is shown as the radius of the circle sector in Figure 2.3. It is determined by the sampling rate of the radars ADC (F_s), the speed of light (c), and the frequency slope of the chirp (S) [6]. The equation is:

$$d_{max} = \frac{F_s \cdot c}{2S} \tag{2.1}$$

The AWR1642 model is limited to an ADC sample rate $F_s = 6500$ due to the nature of the architecture [6].

2.1.4 Maximum Velocity

The maximum velocity v_{max} that can be detected by two chirps is illustrated in Figure 2.4. The determining factors of the v_{max} are the carrier wavelength λ and the chirp duration T_c. Carrier wavelength can be calculated from the starting frequency of a chirp and the duration is simply the time it takes the ADC from start to finish of a chirp. The equation is:

$$v_{max} = \frac{\lambda}{4T_c} \tag{2.2}$$

This means that as the chirps get shorter, higher velocities can be measured. As the carrier wavelength is basically constant at 77 GHz, the only parameter that can be influenced is the chirp duration [6].

2.1.5 Range Resolution

The shortest possible distance between two objects such that the radar could resolve them as separate points is called range resolution. This distance is measured only in the range difference of objects in relation to the radar. In Figure 2.5 this has been visualized with the dashed line between the two objects. The range resolution, d_{res}, of the radar is determined by the speed of light (c), the frequency slope of the chirp (S), and the chirp duration (T_c). Because $S \cdot T_c = B$ the range resolution is only dependent on the bandwidth of the radar [6]. The equation is:

$$d_{res} = \frac{c}{2ST_c} = \frac{c}{2B} \tag{2.3}$$

2.1.6 Velocity Resolution

The velocity resolution, v_{res}, is the smallest difference in velocity two objects equidistant from the radar have to have in order for the radar to be able to estimate their separate velocities. This can be seen in Figure 2.6 as the magnitude of the dashed vector. The velocity resolution is determined by the wavelength (λ) of the chirp and the frame duration (T_f) [6]. The equation is:

$$v_{res} = \frac{\lambda}{2T_f} \tag{2.4}$$

This means that the longer the frames are, the better the resolution gets.

2.1.7 Trade-offs Between Resolutions and Maxima

There are trade-offs when designing chirps for FMCW radars. For this radar, the range resolution will be treated as constant, but the other three parameters are dependent on chirp duration in the following way:

$$d_{max} = \frac{F_s \cdot c}{2S} \propto \frac{1}{S} = \frac{1}{\frac{B}{T_c}} = \frac{T_c}{B} \propto T_c \tag{2.5}$$

$$v_{max} = \frac{\lambda}{4T_c} \propto \frac{1}{T_c} \tag{2.6}$$

$$v_{res} = \frac{\lambda}{2T_f} \propto \frac{1}{T_f} \propto \frac{1}{N \cdot T_c} \tag{2.7}$$

d_{max} is proportional to the chirp time, but v_{max} and v_{res} are inversely proportional to it. Furthermore, v_{res} is also inversely proportional to the number of chirps in

a frame. Thus to get a better (smaller) velocity resolution, either more chirps per frame or longer chirps are needed. Longer chirps will increase the maximum range of the radar, but decrease the maximum measurable velocity.

2.2 Localization

The robot used in this project has advanced positioning memory, but it does not have a way to know where it is. Localization can be used to relate the position of the robot to each other and also to predict and correct movements of the robot. With such a system, localization in the environment and collision avoidance are possible.

The localization in this project is done through the fingerprint provided by the radar in five different sectors of the radar field. Through these fingerprints, the relative position of the objects/robot compared to the radar can be found.

2.3 Machine Learning

This section discusses machine learning algorithms used for gesture recognition with a focus on decision trees, as was used in the project.

2.3.1 Decision Tree

A decision tree is an algorithm containing control statements which are displayed as a tree. Depending on the outcome of a control statement, the algorithm chooses one of several branches and is then faced by a new control statement. This continues until the algorithm reaches a decision on which it bases its predictions [7].

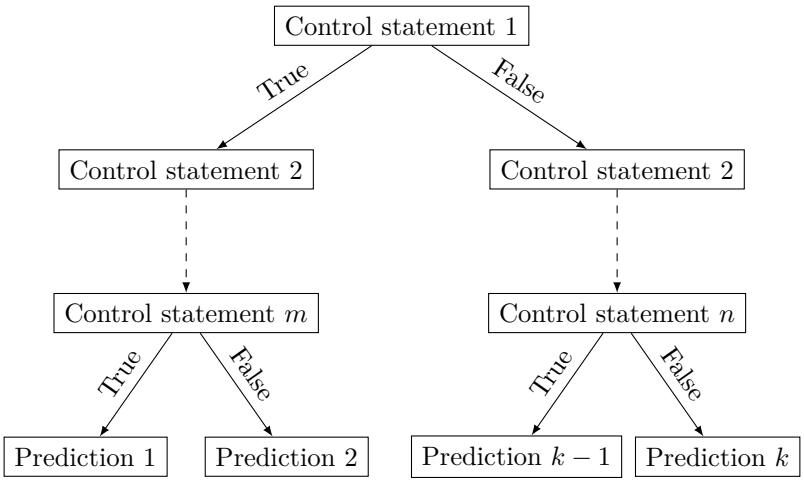


Figure 2.7: Flowchart for a decision tree

2.3.2 Bagged Tree

Bagged tree, also known as bootstrap aggregating, is a technique to stabilize classifications for unstable procedures with big variations in the training data. It does this by producing several new training sets by sampling the incoming data. To populate the newly created sets, it duplicates some of the observations made. This leads to a couple observations occurring twice in the new training sets and some points existing in all new sets [8].

When the new sets are divided from the incoming data, the different learning methods, for example decision tree, are applied on all different sets. After evaluating the outcome of the learning methods, a voting by the program is made to classify the best method and make a decision [8].

2.3.3 Classification Tree

When creating different decision tree models, the Classification Learner creates a ClassificationTree object with binary splits for classification [9]. This object is then exported and can be used to give predictions on incoming data. The possible input arguments to a ClassificationTree object is predictor data, response variable names, tables and formulas which contain response variable names, and predictor variables for model training. Depending on what prediction is desired and on what kind of data the prediction is going to be made, different input arguments are needed.

Predictor data is made up of matrices which are created from the sample data, where each row corresponds to one variable of the data and each column corresponds to a new data sample. Response variable names are variables in the table that denote the response of a specific task. Formulas are used when a subset in the table with several variables is used as predictors [9].

The table consists of the sample data, where each column is a new observation. The table could include both predictors and response variables, depending on how the tables are constructed. MATLAB uses different input arguments to optimize the model [9].

2.4 Control System Theory

For controlling a system to function in a desirable way, we use controllers. Controllers are functions on the current error $e(t)$ of a system, where the error is the difference between its setpoint and current state. For example, if the cruise control of a car is set to 90 km/h and the car at one instant t_0 in time is travelling at 70 km/h, the error at that instant is $e(t_0) = 90 - 70 = 20$ km/h. The controller tries

to minimize this error and ideally bring it to zero. This section will briefly describe different kinds of controllers and their functions in a control system.

2.4.1 Proportional Controller – P-controller

A proportional controller is the most basic controller. The control signal it sends out is a constant multiplied by the error. Thus, the larger the error, the stronger the corrective control signal.

$$u(t) = K_p e(t) \tag{2.8}$$

Proportional controllers are appropriate for systems where one simply wants to eliminate rough errors. Since the control signal is proportional to the error, it may not be able to send control signals strong enough to affect the system if the error is small enough [10].

2.4.2 Integral Controller – I-controller

While the proportional controller operates on the error at one instant in time, the integral controller operates on the accumulated error – i.e., it gives a stronger control signal the longer the error has persisted. It does this by multiplying a constant with the time integral of the error.

$$u(t) = K_i \int_0^t e(\tau)\ d\tau \tag{2.9}$$

Integral controllers can correct smaller persistent errors, which a pure P-controller cannot, which makes them ideal for processes no persistent error is desired [10].

2.4.3 Derivative Controller – D-controller

The derivative controller does not work directly on the error, but rather tries to bring the rate of change of the error to zero. This is done by multiplying the time derivative of the error with a constant.

$$u(t) = K_d \frac{de(t)}{dt} \tag{2.10}$$

This is useful if one wants to stabilize a system where the error changes rapidly [10].

2.4.4 PID-controllers

These controllers are rarely used in practice on their own (with the exception of the pure P-controller, which has its uses) and are often combined into PI, PD, or

PID-controllers. The control signal of these controllers is simply the sum of the control signals of their components [10]. So a PID-controller would have the control signal

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\ d\tau + K_d \frac{de(t)}{dt} \tag{2.11}$$

The combination of controllers in this way can utilize the strengths and compensate for the weaknesses of the component controllers [10].

2.5 Fixed-point Numbers and Q-format

Fixed-point numbers are a way to store fractional values in a computer, where the word *fixed* refers to the fact that a fixed number of bits in the number are designated to be after the decimal point. One notation for fixed point numbers is the Q-format. This notation is on the form Q`a.b` where `a` represents the number of integer bits and `b` the number of fractional bits. If the total number of bits in a number is specified (and whether the number is signed or unsigned), `a` can be omitted and the notation becomes Q`b`. Converting a binary Q-format number to a decimal number is done by interpreting the binary number as a (signed or unsigned) integer and diving it by $2^{\texttt{b}}$ [11].

Fixed-point numbers are useful because they are easy to implement in a small memory space and operations performed on them are fast, making them well suited for real-time applications such as radar [11].

3

Equipment

In this chapter, the software and devices used in the project will be presented and described. Two hardware solutions were used: the radar AWR1642, and the mobile robot Pioneer 3-DX. This hardware was used together with several software solutions for programming. millimeter Wave Software Defined Kit (mmWave-SDK) and Robot Operating System (ROS) were used to configure the radar and drive the robot, respectively. We also used MATLAB for learning and classification, as well as Python as an object oriented programming language to perform the robotic tasks.

3.1 Hardware

3.1.1 AWR1642 Radar

The radar used in this project was the AWR1642 model from Texas Instruments (TI). It is a Frequency Modulated Continuous Wave (FMCW) radar that utilizes waves in the frequency span of 76-81 GHz, with a maximum bandwidth of 4 GHz [12]. It has 2 transmitters and 4 receivers, as can be seen in Figure 3.1. Communication is done through a micro-USB port. There is a signal processing unit on the radar which computes the Fast Fourier Transform (FFT) on the received signals, which means that the radar can output a cloud of reflection points directly, instead of received signals.

3.1.2 Pioneer 3-DX

The robot used in this project was the Pioneer 3-DX from Adept MobileRobots. It is a small but sturdy robot that has differential drive, which means one axis for each wheel [13]. This allows it to steer with only two wheels, with one small additional wheel at the back for balance. It has a linear velocity of 1.2 m/s and a rotational velocity of 300 degrees/s. Inside the robot there is an embedded computer keeping track of data such as the distance traveled and position relative to the starting position. To access the embedded computer a laptop can be placed on top of the

robot, connected through the serial port on the side.

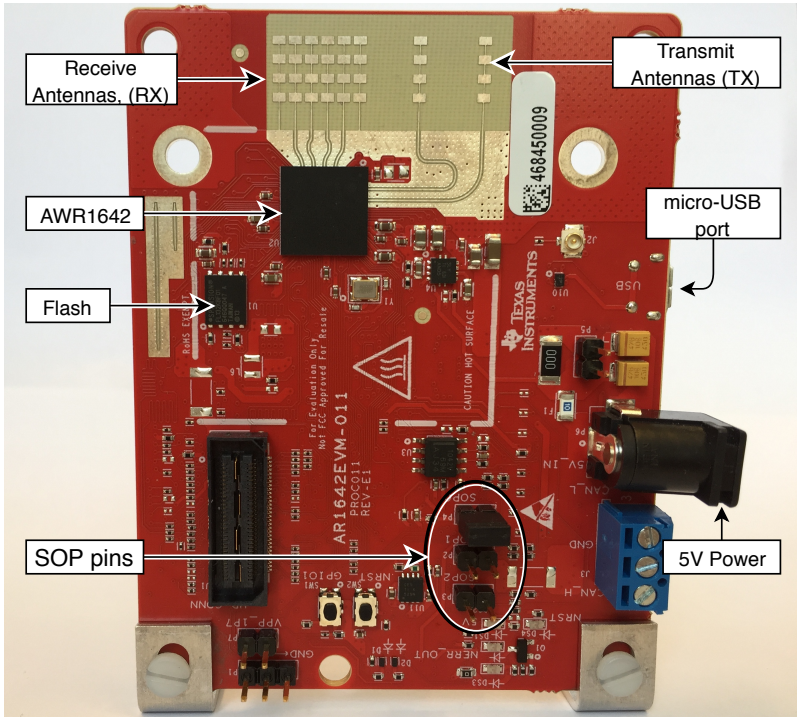


Figure 3.1: The AWR1642 module

Figure 3.2: The Pioneer 3-DX. Tinfoil added to increase strength of reflected signals.

3.2 Software

This section will focus on the software tools used in the project. Firstly we explain the tools required to interact with the hardware, and secondly, we discuss the programming languages for gesture recognition and control of the system.

3.2.1 The mmWave SDK

TI has developed a front-end API to program their mmWave series radars, which AWR1642 is a part of. The programming is done by a sequence of commands, briefly outlined in Table A.1, with a deeper explanation available in [14].

The radar provides a programmable interface for chirp and frame construction. It allows for up to four profile chirps' `profileCfg`, which acts as the base building blocks for specific chirp variants. These variants are defined using the `chirpCfg` command which can be repeated to get up to 512 unique chirp variants. These can then be composed into one frame using the `frameCfg` command and used as a signal. A more detailed explanation of the profileCfg, chirpCfg and frameCfg are as follows.

3.2.1.1 Base Profile of a Chirp

In Figure 3.3, the values of the `profileCfg` command are explained. Especially noteworthy values are: start frequency, which determines the carrier wavelength λ; the combined total time of idle time and ramp end time, which determines the total time of a chirp T_c; frequency slope, the rate at which frequency of the chirps ramps, which is S; and lastly the ADC sample rate, F_s, which is part of the maximum signal distance equation.

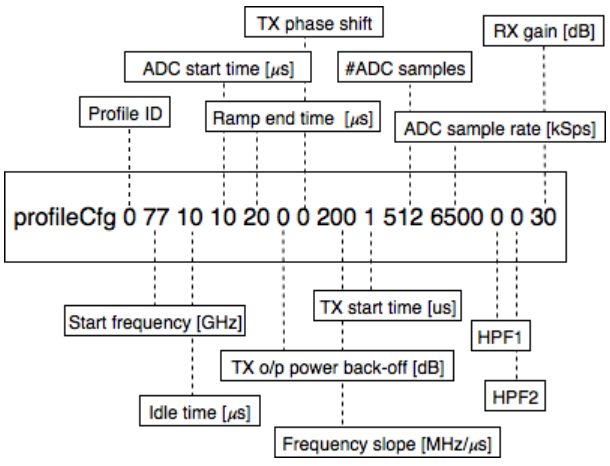


Figure 3.3: An example of a `profileCfg` command.

3.2.1.2 Individual Chirp Variation

Figure 3.4 explains the meaning of the values used by the `chirpCfg` command. The values denoted "Var." in the figure are added to the base values decided by the specified profile ID. Values that can be modified are start frequency, frequency slope, ADC idle and start time. Also, the transmitter(s) that should be used are determined.

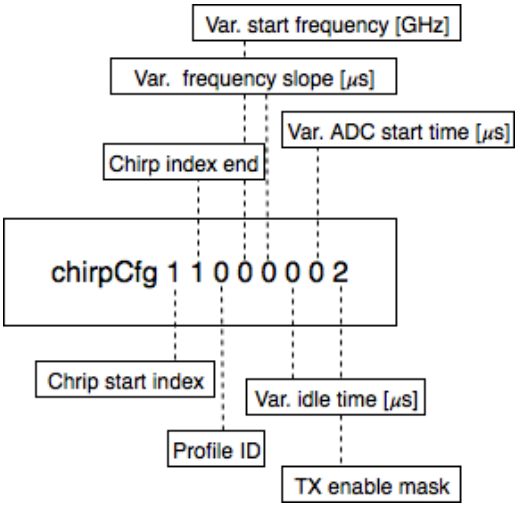


Figure 3.4: An example of a `chirpCfg` command without any variation.

3.2.1.3 Frame Construction Parameters

For a brief explanation of `frameCfg`, see Figure 3.5. The most important interaction of the command is between the total time of the chirp-loops and the frame periodicity. The loop time should be shorter than the periodicity to give the radar enough time to process the reflections. A longer break gives the radar more time to do computations between frames, which enhances the quality of the data output.

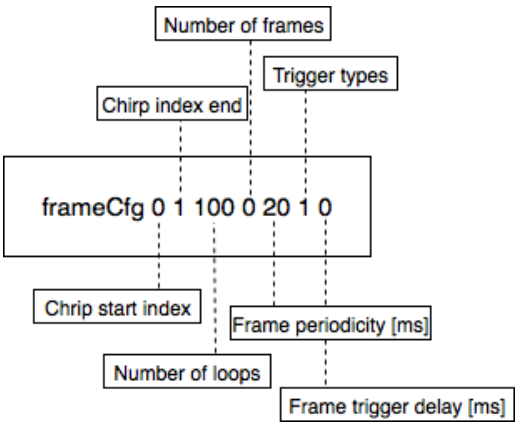


Figure 3.5: An example of a `frameCfg` command.

3.2.2 ROS

Robot Operating System (ROS) is a tool used to simplify the communication between robots and their host computers [15]. It is multi-lingual which allows for Python as the language of choice in this project. Furthermore, it is Peer-to-Peer which means computers in the network talk directly to each other without going through a intermediary server or something similar [16]. There are four main concepts which make up the structure of the communication: nodes, messages, topics, and services [15].

A node takes information and turns it into useful data by performing calculations [17]. For example, there may be a node for controlling the robot's wheels and tracking how far it has traveled. However, before they can begin to communicate they need to register themselves to the master node. By doing this they can locate each other and then utilize peer-to-peer, foregoing the need to talk via the master.

Messages are the way nodes speaks to each other. They contain standard types such as int, boolean, or an array of that type. To do this they need to publish or subscribe to the appropriate topic. They can then send or receive messages without having to talk directly. A node can subscribe or publish to multiple topics and each topic can have unlimited subscribers and publishers.

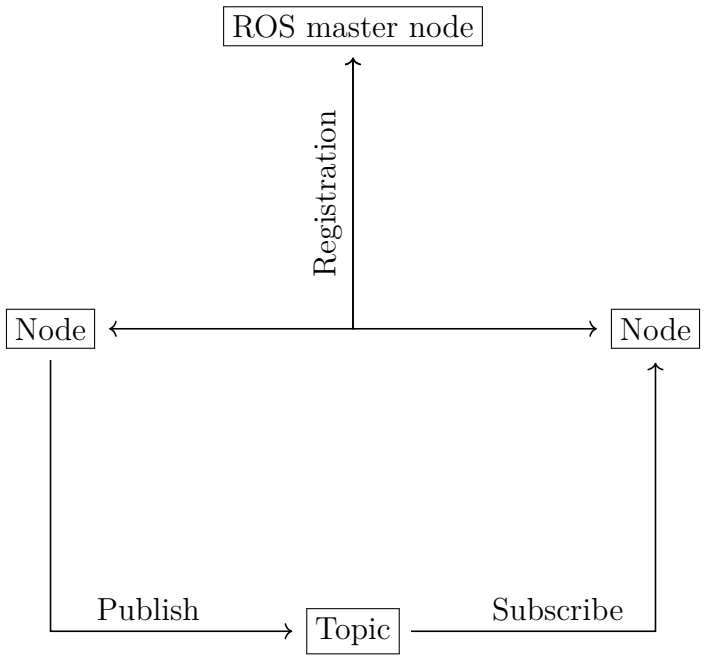


Figure 3.6: How Nodes communicate via Topics.

For synchronous communication, a node creates what is called a service. When another node needs the information, it sends a request messages to the node with the corresponding service name. Then, a reply messages is sent back to the requesting

node. An important note is that in contrast to a topic, only one service with a particular name can exist among all nodes in a ROS network.

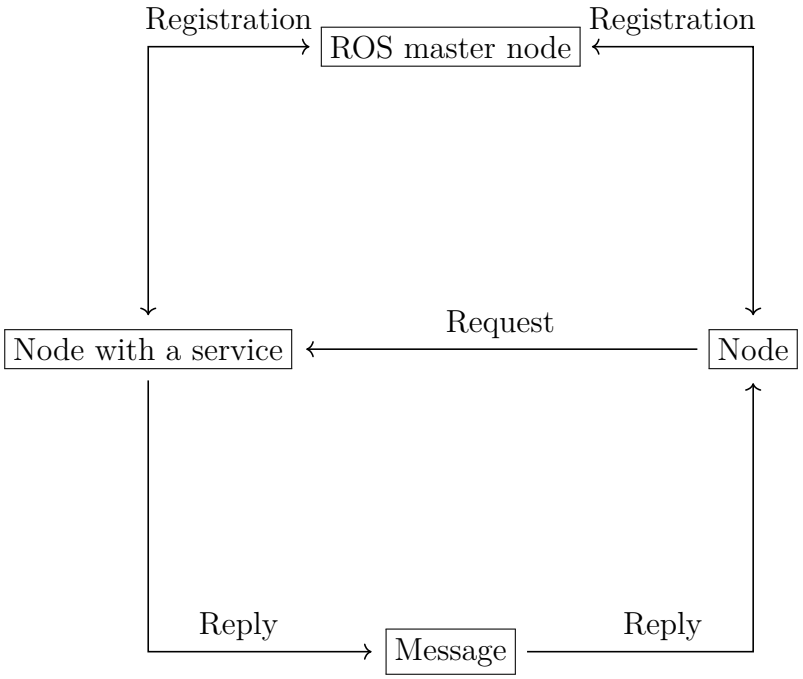


Figure 3.7: How Nodes communicate via Services.

3.2.3 ROSARIA

ROSARIA works as a middle hand between the ROS network and the robot's controller. It is a node which serves between ROS commands and the ARIA (Advanced Robot Interface for Applications) SDK [18]. ARIA is written in C++ and the API can be accessed by a multitude of languages, including Python which is used in this project. The ROSARIA node subscribes to a single topic that takes commands from a host computer to control the robot's velocity. Several topics are published about the status of the robot and two services exist to start or stop the motor.

3.2.4 Python

Python is a programming language, adhering to the imperative, functional, procedural, and reflective programming paradigms. Version 2.7 of Python was used in this project. The simplicity of the language and the implementation of the non-proprietary and generic Robot Operating System module makes it essential for communication with, and control of the robots.

3.2.5 MATLAB

MATLAB is a mathematical computing environment built by MathWorks, implementing and allowing for functional, imperative, procedural, object-oriented, and array programming flows. It is widely used in computation-heavy fields and provides powerful plug-and-play modules for complex problems, such as machine learning [19].

3.2.5.1 Classification Learner Toolbox

The Classification Learner toolbox in MATLAB contains a wide variety of Machine Learning algorithms bundled with an interface that allows for classification on gathered data sets [20]. In order for some algorithms to be able to classify new data sets, it must first be trained with a relatively large number of data sets with known classifications. This is known as supervised learning [20]. The Classification Learner toolbox then creates its own model which then can be exported and used on new incoming data. This could be a variety of different models for example decision trees, discriminant analysis, logistic regression, nearest neighbors, and support vector machines. For the purposes of this study, the classification toolbox will be used on data generated by the radar to classify gestures performed in front of it.

4

Methods

This chapter gives an overview of how the target system for controlling a robot with hand gestures, is constructed. It starts with the radar configuration and programming, and continue with the method of recognizing different gestures via a machine learning algorithm. Having explained them, the control of robot and it's localization is pinpointed through the rest of the chapter.

4.1 Gesture Recognition Using Radar

To set up the radar, the micro-USB port on the radar board was used to connect the radar to a computer. MATLAB was chosen for the implementation as it has powerful out of the box utilities that can easily be used.

Connection to the radar is established in MATLAB with the command `fopen(serial)`, where `serial` is the serialized ports of the host computer which can be found when the radar has been powered on and connected to the computer. While the serial port remains open, `fread(serial)` can read data from it. This is how data is transferred from the radar to the computer running MATLAB. Lastly, the connection can be dropped by `fclose(serial)`, as keeping the port open locks out any other process from accessing that port.

4.1.1 Selection of Radar Parameters

The AWR1642 can utilize a bandwidth of up to 4 GHz, ranging from 77 GHz to 81 GHz. Testing showed that 2 GHz gave good results for the purposes of this project and was therefore decided upon. With a constant bandwidth of 2 GHz put into Equation 2.3, this gives a d_{res} of:

$$d_{res} = \frac{299\ 792\ 458 \text{ m/s}}{2 \cdot 2 \text{ Ghz}} = 0.07494 \text{ m} \tag{4.1}$$

The rest of the radar parameters are only affected by either the chirp duration

or amount of chirps as seen in Section 2.1.7. For the purpose of this project, the maximum range of the radar was set to a relatively low number, because there was no need for long range detection. This led to an unnecessarily high value for the maximum velocity, but it was needed to get the velocity resolution low enough.

When the desired maximum range and velocity was set, the only way to enhance the velocity resolution was to increase the number of chirps in a frame. However, this has the downside of also increasing the amount of data that has to be processed for every read. The system performance would be negatively affected as the radar does not transmit a new frame until the previous frame's data has been processed. So v_{res} was set to 0.06 m/s. The used values of the variables and parameters can be seen in Table 4.1 and Table 4.2.

Table 4.1: The variables of the chirp and frame.

Variable	**Value**
Bandwidth	2 GHz
Wavelength	3.89 mm
Slope	100 MHz/µs
ADC sample rate	5333 kSps
Chirp duration	20 µs
Frame duration	33.33 ms

Table 4.2: The parameters of the radar.

Parameter	**Value**
Range Maximum	7.99 m
Range Resolution	0.07494 m
Velocity Maximum	48.67 m/s
Velocity Resolution	0.058 m/s

4.1.2 Configuration of AWR1642

The radar allows for specialized programming of its processors. This is done by setting a jumper over the SOP pins on the circuit board of the AWR1642 and then flashing the memory. There is a TI utility app that was used for this purpose. A basic program provided by TI was used that allows for signal configuration during run time.

Once connection is established, commands can be sent to the radar by writing them to the serial port. Several commands can be issued in sequence by writing a configuration file (.cfg) to the serial port line by line. The configuration used in this project can be seen in Appendix A.1. This configuration allowed for satisfying

range and precision for the gesture recognition of the project. When applying a configuration, the radar should first be turned off using the `sensorStop` command. The last line should be `sensorStart`, which tells the radar to start acting on the given behavior. In between these commands, the signal programming of the radar takes place as seen in Appendix A.1, which is explained in Section 3.2.1. Once the radar has been started, it continuously sends payloads of data through the serial port, which can be read by using `fread(serial)` in MATLAB.

4.1.3 Radar Data Payloads

The radar sends a bytestream of data through the serial port. MATLAB reads this bytestream in 512-element column vectors where each element is the size of one byte. This data is made up of two blocks: a header which consist of 40 bytes, and Tag-Length-Value (TLV) items. The TLV items are themselves made up of three blocks: tag, length, and payload where each block has a different size. The tag field contains four bytes which are used as an identifier for what type of data the current TLV item represents. The length field also contains four bytes and represent the length of each payload, in bytes, depending on how many object the radar has detected. The payload itself consists of a list descriptor which describes how many objects were detected and in which Q-format the position bytes are presented. The radar outputs 16-bit numbers in signed Q9 format, meaning each number using the format has 9 fractional bits, 6 integer bits, and 1 sign bit. The information about each object is stored in a 12 byte structure of six values divided up into most and least significant byte, which is shown in Table 4.3.

Table 4.3: Data structure of a detected object.

Byte	Value	Significance
1	Range index	LSB
2	Range index	MSB
3	Doppler index	LSB
4	Doppler index	MSB
5	Peak value	LSB
6	Peak value	MSB
7	x-coordinate	LSB
8	x-coordinate	MSB
9	y-coordinate	LSB
10	y-coordinate	MSB
11	z-coordinate	LSB
12	z-coordinate	MSB

From these data structures, objects are extracted into column vectors with the proper number format for each element. The coordinates of the objects are converted

from Q-format to signed doubles, and the Doppler index is converted to a signed integer. The other values only have their bytes merged into one 16-bit unsigned integer. It should be noted that the values of the range and Doppler indices are still in bin form, meaning that they are not in [m] or $[\frac{m}{s}]$ yet. After this, the objects are ready for further processing depending on if they are supposed to be used for gesture recognition or localization.

4.2 Gesture Data Processing

In order to record gestures performed in front of the radar, several objects have to be sampled by the radar over time. However, the radar may pick up a different amount of objects each time it reads a gesture and as such this ordered set of objects within gesture range must be condensed into a uniform format that is the same size no matter how many objects were detected during read time. This is done because the machine learning requires all data sent into a model to be of the same size and have the same amount of determinants to be able to do classification.

To solve this, the objects detected within gesture range are first filtered according to their peak values. Objects with low peak values are considered clutter and are disregarded while objects with high peak value are kept. After this, the x- and y-coordinates of all objects are extracted and averaged to create a single average point for that payload. This is done for each data payload from the radar, giving several average points in the gesture range per gesture. These averages are then compared to each other so that the changes over time can be seen. Finally, all those deltas are summed up resulting in an average shape of the gesture. An overview of this workflow can be seen in Figure 4.1. This average shape is a two-element array and consists of change in x-direction and change in y-direction over time. This array can now be fed to the machine learning model as input.

4.2.1 Learning

In the machine learning toolbox in MATLAB, different types of statistical methods for machine learning can be chosen. Depending on the kind of data collected, different methods yield different results and levels of accuracy. By training with all methods and comparing the accuracy of each algorithm, the most suitable solution could be chosen.

4.2.2 Training Data

Machine learning algorithms require sets of training data to be able to make predictions. These can be constructed manually or be manufactured if the specifics of

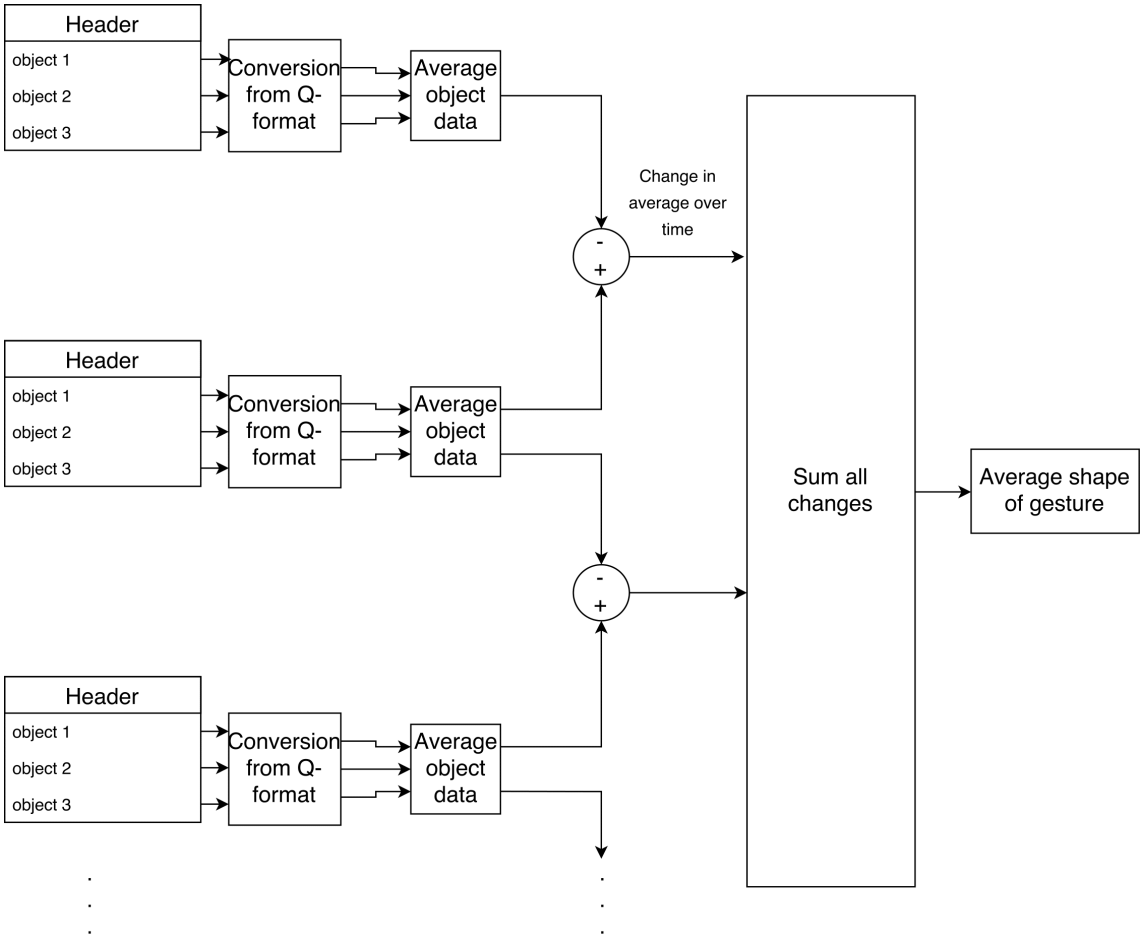


Figure 4.1: Data processing workflow for each gesture

the data are known. In this project, the machine learning algorithm was trained on a dataset with 10,000 generated examples of each gesture. These training gestures were randomly generated according to a certain set parameters describing what kinds of tolerances we wanted for each gesture. These parameters can be seen in Table 4.4. The training set for "No gesture" was generated from the complement of these sets, within the interval [-1.2, 1.2].

Table 4.4: Tolerances for gestures, in interval notation

Gesture	**Range in x-direction** [m]	**Range in y-direction** [m]
Swipe right	[-0.7, -0.2]	[-0.2, 0.2]
Swipe left	[0.2, 0.7]	[-0.2, 0.2]
Push	(-0.2, 0.2)	[-0.3, -0.05]
Pull	(-0.2, 0.2)	[0.05, 0.3]

4.2.3 Graphical User Interface

To facilitate the usage of the gesture recognition system and to communicate which gesture the radar interpreted, a simple graphical interface application was implemented. It helps the user set up the radar connection and then has a continuous application window. The application window consists of four different pictures of the possible gestures and a button. When the button is pressed the system records objects in front of the radar for a short time and then tries to classify what gesture was performed. The predicted gesture is then marked in the GUI.

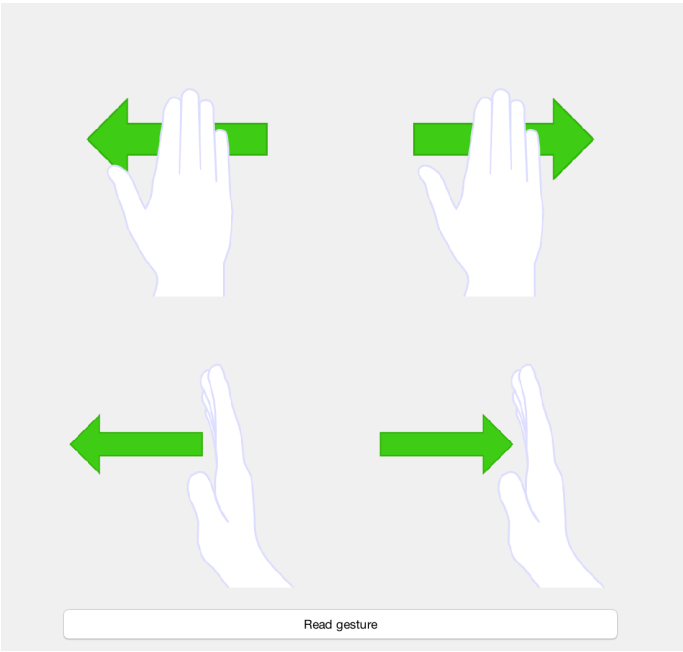


Figure 4.2: The gesture recognition GUI

4.3 Robot Control

The robot system consists of one computer physically connected with the robot and one computer running the ROS master node. Communication with the embedded system of the robot is done through the ROSARIA protocol by the computer on top of it. The host computer that is connected to the radar has also set up a ROSARIA node within the WiFi network. The ROSARIA nodes can send information between each other, which enables the information to go from the gesture in front of the radar, through the MATLAB engine run by Python, then to the computer on the robot and into the robot itself. By subscribing to the proper topic, the robot can then be controlled by altering the velocity of the wheels.

The velocity of the robot was controlled by a simple P-controller, where the velocity was set to be proportional to the distance between the robot and its destination. The gain parameter K_p was set to 0.4 for forward movement and 0.5 for rotational movement. The control system has a tolerance of 0.1 meters, meaning the robot will stop if within a circle of radius 0.1 meters around the destination point.

4.4 Localization

The field of the radar was divided into 5 different sectors to be able to detect up to 3 robots at the same time. The first sector represents the angle between 30 and 70 degrees, the second the angle between 70 and 110 degrees and the third the angle between 110 and 150 degrees, as seen in Figure 4.3.

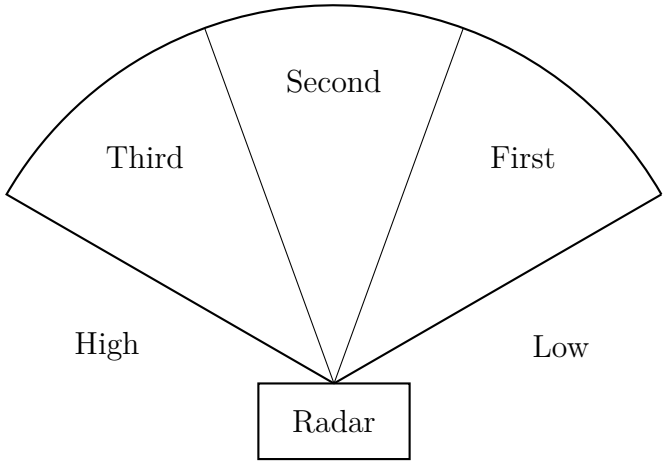


Figure 4.3: The field of the radar is divided up into 5 sectors

Detected points were then categorized by their position and put into one of the five sectors.

Each sector has its objects evaluated by their range and the x and y position to check for robots. When a robot is moving, the direction can be calculated with the help of the angle between a recent and the current position together with the knowledge of which gesture was made. The algorithm evaluates points by distances from the radar depending on the gesture made. For push and swipe gestures, the objects closest to the radar is evaluated and for pull gestures the objects furthest away from the radar are evaluated. This enables the possibility of detecting one robot in each sector of the radar. For example, robots can move at the same time and still be detected and localized simultaneously in the different sectors.

When there is no moving object in the field of the radar, the coordinates and the calculated angle of the last position of the robot are sent from the system to the robot. To check that the two different coordinate systems correlates, the robot coordinate system is updated and if needed the robot will correct its position and once again check with the system if it is at the right position.

4.4.1 Selective Data Culling

For the localization a good way to refine the data set to examine and avoid unnecessary burden is to look at the peak value bytes, since a good reflection will have a high value. To increase the strength of the signals reflected by the robot, it was partially covered in tinfoil.

To select the peak value bytes, the code from the machine learning preparation was re-factored without the average over time and summation parts. To determine which objects are robots, the peak values of static objects and the reflections from a moving piece of foil were analyzed. The results of these tests can be seen in Table 4.5. From these values, a minimum peak value of 800 for the filtering of objects was decided upon.

Table 4.5: Measurements with aluminum foil at various distances

Peak Value	**x** [m]	**y** [m]
805	0.1406	0.6094
966	0.1953	0.5938
998	0	0.625
1856	0.1953	0.5938
1420	0.0469	0.75
868	0.0391	0.625
1160	0.0469	0.5
830	0.0234	0.625
1218	0.0625	0.625
1091	0.0234	0.625
993	0.0781	1.25
923	0.0859	1.375
1165	0	1.375

5

Results

The final version of the system can be seen in Figure 5.1. It consists of two computers, one radar, and one robot. The radar and robot are connected via USB to their respective controller computers, while the ROS and ROSARIA sub-systems communicate using a WiFi connection.

The system has a small amount of software setup in addition to the physical one. The host has to publish a ROS node and the robot controller has to subscribe to it. Then the system is initiated by the host running a Python script, starting a MATLAB engine that communicates with and evaluates data from the radar.

While the Python script is run from the terminal, the MATLAB program presents a simple GUI through which the user sets up the radar configuration and enables gesture recognition. When a gesture is performed the outcome is passed to the Python script which publishes a subtask to the ROS node. It is then read by the robot controller and simply passed on to the robot's internal control system. The position is updated and a new subtask is calculated.

Once the robot has reached its destination the system performs a localization check and then wait for a new gesture to be registered. This is illustrated by the figures in Appendix B.

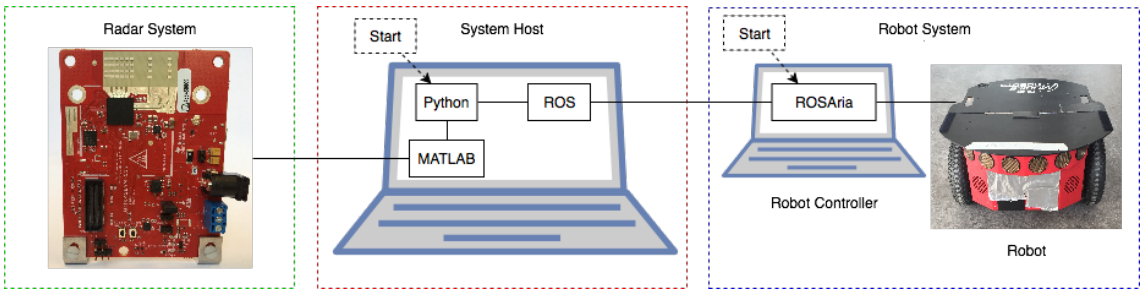


Figure 5.1: This is the final system configuration of the project.

5.1 Gestures

For gesture recognition, the radar filters out objects that are farther away than 40 cm as they are to be considered for localization instead. This splits the radar field into a gesture section and a localization section.

In a test of the final gesture recognition system, where 50 repetitions of each gesture were performed in front of the radar, the accuracy ranged between 88% and 100% with a mean accuracy of 96%. The confusion matrix of this test, showing the number of faults in gesture recognition, can be seen in Figure 5.2.

Actual gesture \ Predicted gesture	**Swipe Right**	**Swipe Left**	**Push**	**Pull**	**None**
Swipe Right	**48**			2	
Swipe Left		**44**		1	5
Push			**49**		1
Pull				**50**	
None				1	**49**

Figure 5.2: Confusion matrix for a gesture recognition test.

After training many different statistical models in the classification learner application, the model with the best result was the Bagged Tree Model. The result after training the model with a total of 50,000 gestures can be sen in Figure 5.3.

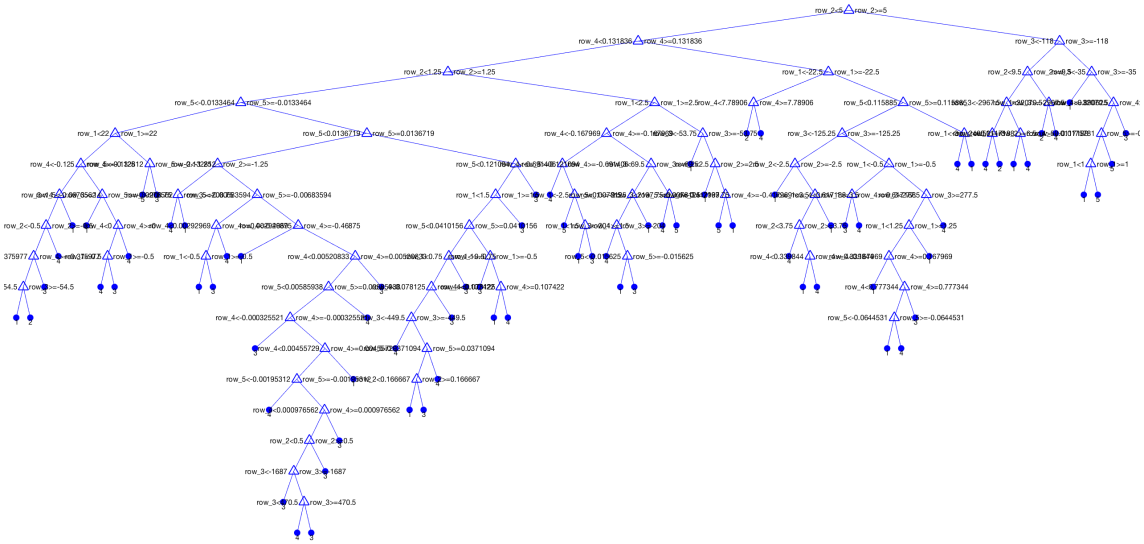


Figure 5.3: Bagged tree model after supervised training

5.2 Localization

To evaluate the localization, five points on a flat surface was laid out as seen in 5.4. The distances to these points relative to the radar can be seen in 5.1. Through gestures preformed by the user the robot was relocated to these points. Directly after a gesture is recognized, the localization process was started and tried to locate where the robot stopped. This cycle continued for the next four points and the whole test was performed 10 times.

The results of the detection in the localization test can be seen in Table 5.2. The success rate in the different coordinates differ a lot. In ten tests, coordinates 2 and 5 had a 0% success rate, and for coordinate 3 the rate was 10%. For coordinates 3 and 4, the localization had a significantly better success rate at 70% for each coordinate. The result of the localization can also be seen in Figure 5.5.

Table 5.1: The predetermined points the robot will go to when testing the localization.

Coordinates	**Value** [m]
$(Start)$	(0.6 , 2.4)
(x_1, y_1)	(0.6 , 1.7)
(x_2, y_2)	(0 , 1.7)
(x_3, y_3)	(0 , 1)
(x_4, y_4)	(0 , 0.4)
(x_5, y_5)	(0.6 , 0.4)

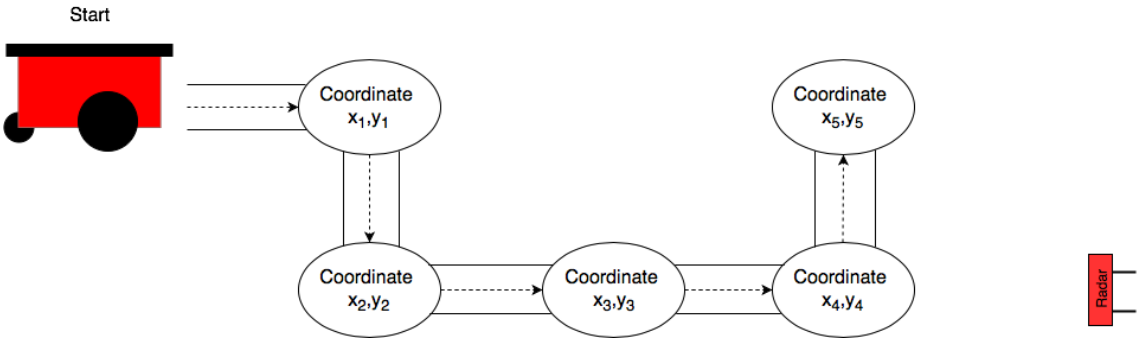


Figure 5.4: Movement pattern of the robot

Table 5.2: Localization results of moving robot

Test	x_1	y_1	x_2	y_2	x_3	y_3	x_4	y_4	x_5	y_5
1	0.5	1.9375	0	0	0	0	-0.1797	0.601	0	0
2	0	0	0	0	0	0	-0.047	0.5	0	0
3	0	0	0	0	0	0	0.0625	0.5	0	0
4	0	0	0	0	0.04	1.25	0	0	0	0
5	0	0	0	0	-0.078	1.25	0	0	0	0
6	0	0	0	0	-0.04	1.25	-0.016	0.5	0	0
7	0	0	0	0	0	1.25	0	0	0	0
8	0	0	0	0	-0.04	1.25	-0.031	0.5	0	0
9	0	0	0	0	0.04	1.25	-0.078	0.49	0	0
10	0	0	0	0	0.039	1.25	-0.015	0.6	0	0

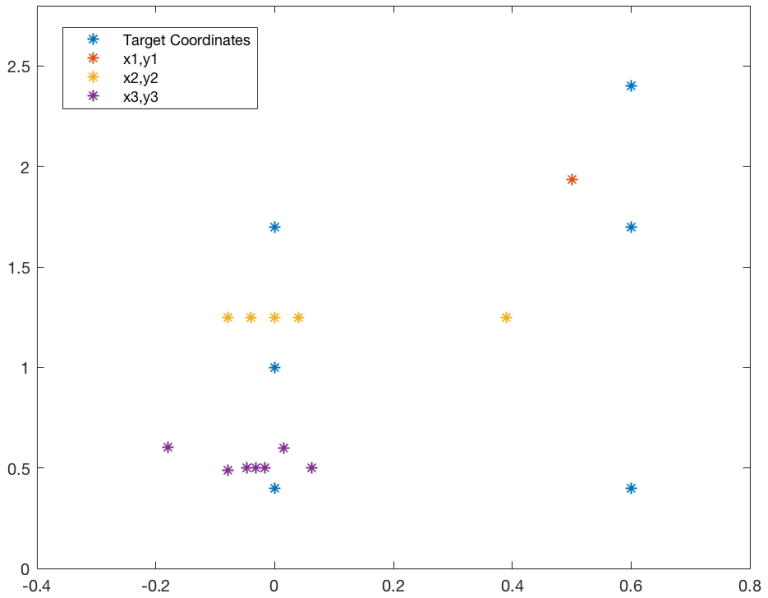


Figure 5.5: Visual representation of the localization hits.

From Table 5.2 and Table 5.1, the average deviations from the measured coordinates were calculated. In coordinates (x_2, y_2) and (x_5, y_5), there were no hits at all, and the results for the other coordinates can be seen in Table 5.3.

Table 5.3: Average deviation from measured coordinates

Coordinate	**Avg. Deviation in x** [m]	**Avg. Deviation in y** [m]
(x_1, y_1)	0.1	0.2375
(x_2, y_2)	No hits	No hits
(x_3, y_3)	0.0462	0.25
(x_4, y_4)	0.0613	0.127
(x_5, y_5)	No hits	No hits

6

Discussion

6.1 Radar Issues

When starting the project, the first issue we faced was that there was little information about how to read data from the radar. Through the help from TI's discussion forum, the issue was solved.

When reading data from the radar, it seemed to output incorrect numbers when placing an object a known distance away from it. Sometimes it seemed to claim that the object was an incredibly large distance away from it when in reality it was very close. The issue turned out to be that the radar gave distances in Q-format, but MATLAB read everything as simple unsigned integers. It does so to be able to send exact coordinate data in a byte stream.

Every read from the radar contains a different amount of detected objects, resulting in object arrays of varying length being created. The machine learning toolbox interface requires a set amount of predictors which means that data must be uniform. The first solution to this problem, when collecting all objects in front of the radar, worked but not well enough. After changing the selection of which data to send to the machine learning model, the performance was increased significantly.

6.2 Gesture Recognition

The way the system handles gesture recognition is by using the averaging algorithm for the shape of gestures and thus, only linear gestures can be classified. Since we can only classify linear gestures, and the parameters of these gestures are known (see Table 4.4), machine learning is not strictly necessary to solve the problem. We could have simply created our own decision tree which would have made our tree much smaller and less complex then the current one which is seen in Figure 5.3. Even though the Bagged tree model created by the classification learner is not huge, the benefit of a less complex and therefore faster gesture recognition would be of some interest. However, if the gesture recognition is to be developed further, creating

decision trees manually would not be a realistic way to realize the classification of gestures as the amount of decisions and determinants would increase the workload and for more complex gestures decision trees may be inefficient. This would require changes to the averaging algorithms as well. For example, if a user were to perform a U shape with their hand in front of the radar, it would currently recognize it as a swiping gesture, since the hand on average moves only sideways. A lot of possibilities for further development lie here, as both accepting advanced shapes or adding a third dimension are obvious desirable extensions.

The gesture recognition system shows good results with the current gestures. Only occasionally will it recognize the wrong gesture, but with a measured accuracy of 96% this system should be considered good enough for the purposes of this project. Even though the test only consisted of 50 repetitions of each gesture the result can still be viewed as representative.

6.3 Robot Control

When implementing the control system of the robot, the first challenge was to run several commands after each other. This was crucial to get the robot to move a predetermined length and turn a specific angle for each gesture. Specifically, the angle was the most problematic part due to confusion over how the system handled angle conversion between radians and degrees. Some documentation claimed that the robot system accepted degrees and then converted internally to radians while other documentation said this was not the case. Also rotations over 180 degrees caused the control system to oscillate, which caused the robot to move unexpectedly. For the specifications for this project this was not necessary so the implemented system turns at most 90 degrees per gesture that requires a turn. It is worth noting that if this system should be implemented in greater scale this rotations problem could be important to solve.

The tolerance could be further improved for more precise movement and the type of regulator could also be changed for a faster response. The reason why the P-controller was chosen was because it is very simple to implement and it can handle the regulating well enough for the purpose of this project. We are simply controlling the position of the robot, so it seems that PI, PD, or PID-controllers are unnecessary.

6.4 Robot Localization

The first algorithm for localization was quite naive and focused on comparing all detected objects without any type of filtering. To determine if an object was a robot or not, the velocity of the objects were compared and for static objects no conclusion could therefore be made. The first solution had problems both with

performance being slow and it only considered the usage of collected data and not how it is collected and catalogued. That is when the peak value became a condition for object selection, as we could inflate it to be greater for the objects we wanted to track in our testing environment using tin foil. The focus also changed from being able to collect and analyze data over a time period to instead do continuous instantaneous tracking of the robot. The new algorithm had better potential to able to detect the position and direction of the robot and then having the possibility to correct its movement.

To break down the problem in smaller parts, a solution for sequentially localizing the robot after each gesture was implemented. In the second version the robot could be located but only when close enough to the radar. This problem was at least partly due to the fact that reflected signals had lower peak values when the robot was moving along the x-axis of the system. To lower the peak value when filtering out objects would increase the number of points detected which could slow down the system significantly and also include incorrect data samples. Due to this challenge, the time frame and the insight that for continuous localization multi-threaded programming was probably necessary, the localization system was not further developed.

When testing the precision of the implemented localization, the programmed route was set up using a measuring tape, and when returning the robot to the origin it was aligned by eyesight. This means that there is a possibility that some measuring variance is due to incorrect starting position and angle in subsequent trials. Contrary to this it turned out that the point furthest away had the biggest deviation in the x direction. In y direction the point second furthest away had the biggest deviation, which suggests less accurate measurements further away from the radar even while still being well within signal range. After ten tests, the decision was made that no more tests would be made as the localization performance were underwhelming. Part of the problem lies in that the localization was not running continuously and getting too few coordinates, and when the robot stood still the angle estimation could not be used.

6.5 Ethical Issues

When it comes to ethical bearings of this project, there are no direct effects. There were neither resource heavy testing, nor any danger or potential harm to the environment or people. The radar technology has been proven to be safe and the robot is environmental friendly and to small in size to cause significant harm. However, there are still ties to the techniques used; for example, radar has seen widespread usage in military applications. Also, autonomy in vehicles and robots can also cause unemployment for many menial tasks. However, there is far more than only negatives to these technologies, the quality of life can drastically be increased from implementations such as automatic delivery systems of all kinds, or collision avoiding systems using radar which, for example, can prevent accidents in low vision conditions.

7

Conclusion and Future Work

This project yielded a proof of concept implementation for a system where a human can control a mobile robot using hand gestures. In particular, a radar is used for data gathering on gestures performed in front of it. A gesture recognition algorithm based on machine learning and classification is used to analyze the data and classify it. In addition to this, a control system for a robot were developed in which the radar was also used for localization.

Although this project implemented a simple gesture recognition in only two dimensions, the FMCW radar technology could very well be scaled for usage in a three-dimensional space.

Even tough the system produced was of simple nature, requiring the user to both have a clear view of the robot and be in the immediate vicinity of it, it showed that a system can benefit from utilizing remote control by hand gestures. Autonomous robots could utilize FMCW radars to both be localized in their environment and to take commands. It is also probable that an actual implementation would have to utilize a parallel or at the very least concurrent program flow, allowing localization to be run continuously.

One could mention that it is better to use more than a single FMCW radar for similar projects. Having a stationary radar for localization limits the movement of the robot to the space covered by the radar. It also forces the gestures to be performed on site near the robots, which negates much of the remote aspect. A recommendation would be that a minimum of two radars should be used: one for the gesture recognition and one for the robot localization. An extension to the system that also could be considered is to have a radar mounted on top of the robot to facilitate collision avoidance.

Bibliography

[1] S. Clark and H. Durrant-Whyte, "Autonomous land vehicle navigation using millimeter wave radar," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 4, May 1998, pp. 3697–3702 vol.4.

[2] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, "Soli: Ubiquitous gesture sensing with millimeter wave radar," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 142:1–142:19, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925953

[3] Z. Zhou, Z. Cao, and Y. Pi, "Dynamic gesture recognition with a terahertz radar based on range profile sequences and doppler signatures," *Sensors*, vol. 18, no. 2, p. 10, Dec 2017. [Online]. Available: http://dx.doi.org/10.3390/s18010010

[4] P. Molchanov, S. Gupta, K. Kim, and K. Pulli, "Short-range fmcw monopulse radar for hand-gesture sensing," in *2015 IEEE Radar Conference (RadarCon)*, May 2015, pp. 1491–1496.

[5] Z. Zhang, Z. Tian, and M. Zhou, "Latern: Dynamic continuous hand gesture recognition using fmcw radar sensor," *IEEE Sensors Journal*, vol. 18, no. 8, pp. 3278–3289, April 2018.

[6] S. Rao, "Introduction to mmWave sensing: FMCW radars," 2017. [Online]. Available: https://training.ti.com/sites/default/files/docs/mmwaveSensing-FMCW-offlineviewing_3.pdf

[7] P. S. Bogumil Kamiński, Michal Jakubczyk, "A framework for sensitivity analysis of decision trees," vol. 26, pp. 135–159, 2018. [Online]. Available: https://link.springer.com/article/10.1007%2Fs10100-017-0479-6

[8] L. Breiman, "Machine learning," vol. 24, pp. 123–140, 1996. [Online]. Available: https://link.springer.com/article/10.1007%2FBF00058655

[9] Mathworks, "Classification tree," 2011. [Online]. Available: https://se.mathworks.com/help/stats/classificationtree-class.html

[10] B. Lennartson, *Reglerteknikens grunder*. Lund: Studentlitteratur, 2002.

[11] Texas Instruments, 2009. [Online]. Available: http://www.ti.com/europe/downloads/mspromchapter1.pdf

[12] ——, "AWR1642 single-chip 77- and 79-ghz FMCW radar sensor," 2017. [Online]. Available: http://www.ti.com/lit/ds/swrs203/swrs203.pdf

[13] Adept Technology Inc., "Pioneer 3-dx," 2011. [Online]. Available: http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx

[14] Texas Instruments, "mmWave SDK User Guide," 2018. [Online]. Available: http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/latest/index_FDS.html

[15] M. Quigley *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[16] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proceedings First International Conference on Peer-to-Peer Computing*, Aug 2001, pp. 101–102.

[17] Open Source Robotics Foundation, "Concepts," 2018. [Online]. Available: http://wiki.ros.org/ROS/Concepts

[18] ——, "Rosaria," 2018. [Online]. Available: http://wiki.ros.org/ROSARIA

[19] Mathworks, "Whymatlab," 2018. [Online]. Available: https://se.mathworks.com/products/matlab/why-matlab.html

[20] ——, "Classification learner," 2015. [Online]. Available: https://se.mathworks.com/help/stats/classificationlearner-app.html

Appendix A

Radar Configuration

This appendix covers details of the configuration of the radar used in the project.

A.1 Configuration File

The AWR1642 can be set up using a configuration file. Figure A.1 shows the configuration file that was used in this project.

```
sensorStop
flushCfg
dfeDataOutputMode 1
channelCfg 15 3 0
adcCfg 2 1
adcbufCfg -1 0 0 1 0
profileCfg 0 77 45 7 20 0 0 100 1 64 5333 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
chirpCfg 1 1 0 0 0 0 0 2
frameCfg 0 1 128 0 33.333 1 0
guiMonitor -1 1 0 0 0 0 0
cfarCfg -1 0 0 8 4 4 0 5803
cfarCfg -1 1 0 8 4 4 0 5803
peakGrouping -1 1 1 1 1 63
multiObjBeamForming -1 1 0.5
clutterRemoval -1 1
calibDcRangeSig -1 0 -5 8 256
extendedMaxVelocity -1 0
compRangeBiasAndRxChanPhase 0.0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
measureRangeBiasAndRxChanPhase 0 1.5 0.2
sensorStart
```

Figure A.1: Configuration file used to program the radar's behavior.

A.2 mmWave Commands Explanation

Table A.2 describes the different mmWave API commands that can be given to the radar in the configuration file. For a more technical description of all arguments, see [14].

Table A.1: Commands used in the radar configuration (.cfg) file.

mmWave API Command	Arguments	Description
sensorStop	-	Turns off transmitters (TX), recievers (RX) and signal processing.
flushCfg	-	Removes the current configuration.
dfeDataOutputMode	<mode Type>	Sets the signal transmitting mode.
channelCfg	<rxChannelEn>, <txChannelEn>, <cascading>	Sets which antennas are to be utilized.
adcCfg	<numADCBits>, <adcOutputFmt>	Sets number of ADC bits and on what format the output will be.
adcbufCfg	<dont_care>, <adcOutputFmt>, <SampleSwap>, <ChanInterleave>, <Chirp Threshold>	Sets output format, and calculation behaviour for the ADC Buffer
profileCfg	<profileId>, <startFreq>, <idleTime>, <adcStartTime>, <rampEndTime>, <txOutPower>, <txPhaseShifter>, <freqSlopeConst>, <txStartTime>, <numAdcSamples>, <digOutSampleRate>, <hpfCornerFreq1>, <hpfCornerFreq2>, <rxGain>	See explanation in 3.2.1.

chirpCfg	<chirpStartIdx>, <chirpEndIdx>, <profileId>, <startFreqVar>, <freqSlopeVar>, <idleTimeVar>, <ADCstartTimeVar>, <txEnMask>	See explanation in 3.2.1.
frameCfg	<chirpStartIdx>, <chirpEndIdx>, <numOfLoops>, <numOfFrames>, <framePeriod>, <trigSelect>, <FrameTrigDelay>	See explanation in 3.2.1.
guiMonitor	<subFrameIdx>, <detected objects>, <log_magnitude_range>, <noise profile>, <rangeAzimuthHeatMap>, <rangeDopplerHeatMap>, <statsInfo>	Sets which data will be sent out from the radar.
cfarCfg	<subFrameIdx>, <procDirection>, <mode>, <noiseWin>, <guardLen>, <divShift>, <dont_care>, <ThresholdScale>	Sets computational behaviour for the CFAR calculations
peakGrouping	<scheme>, <PGinRangeDir>, <PGinDopplerDir>, <StartRangeIdx>, <EndRangeIdx>	Enables peak grouping and how it behaves.
multiObjBeamForming	<subFrameIdx>, <Feature Enabled>, <threshold>	Enables the radar to separate reflections from multiple objects originating from the same range or Doppler detection.
clutterRemoval	<enabled>	Enables self-reflection clutter removal.
calibDcRangeSig	<subFrameIdx>, <enabled>, <negativeBinIdx>, <positiveBinIdx>, <numAvg>	Sets behaviour for the DC range calibration which happens during startup of the radar.

extendedMaxVelocity	<subFrameIdx>, <enabled>	Enables a simple behaviour that increase possible v_{max}
compRangeBiasAndRx -ChanPhase	<rangeBias>, <Re(A,B)>, <Im(A,B)>	Sets behaviour for compensation in range, recieve channel gain and phase imperfections.
measureRangeBiasAnd -RxChanPhase	<enabled>, <targetDistance>, <searchWin>	Enables measurement of the values of the previous compRangeBiasAndRxChanPhase command
sensorStart	-	Turns on transmitters (TX), recievers (RX) and signal processing.

Appendix B

Program Flowcharts

This appendix contains a complete flowchart of the system developed in this project. It is divided in the five different figures Figure B.1, B.2, B.3, B.4, and B.5 being sequenced in the given order and looping from third to fifth while the program is running.

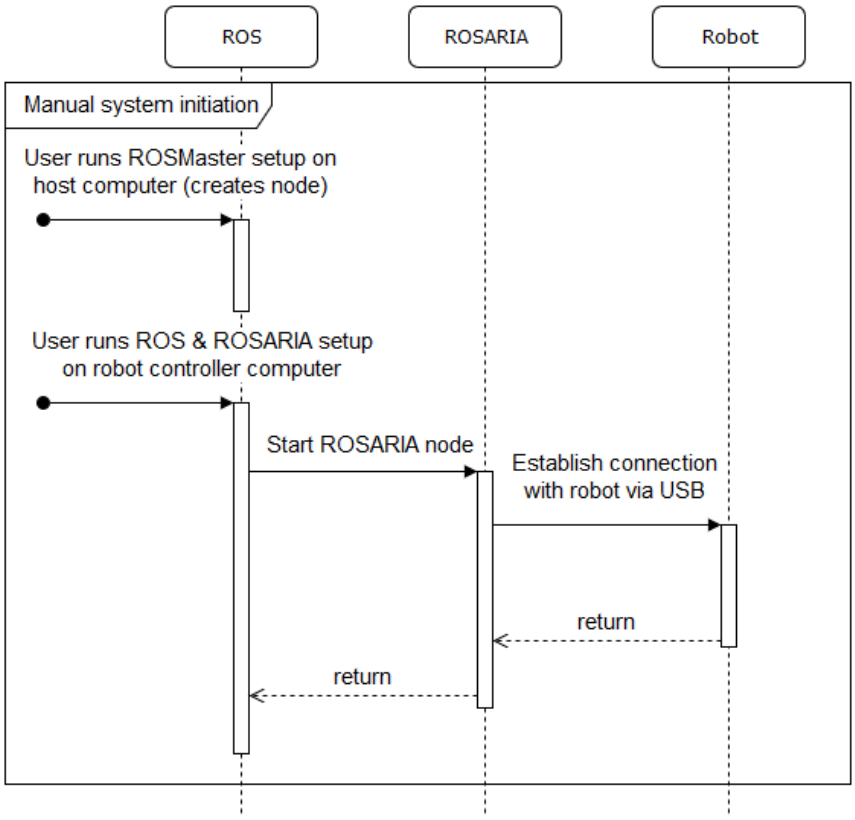


Figure B.1: Program system flowchart depicting the startup required for the ROS system.

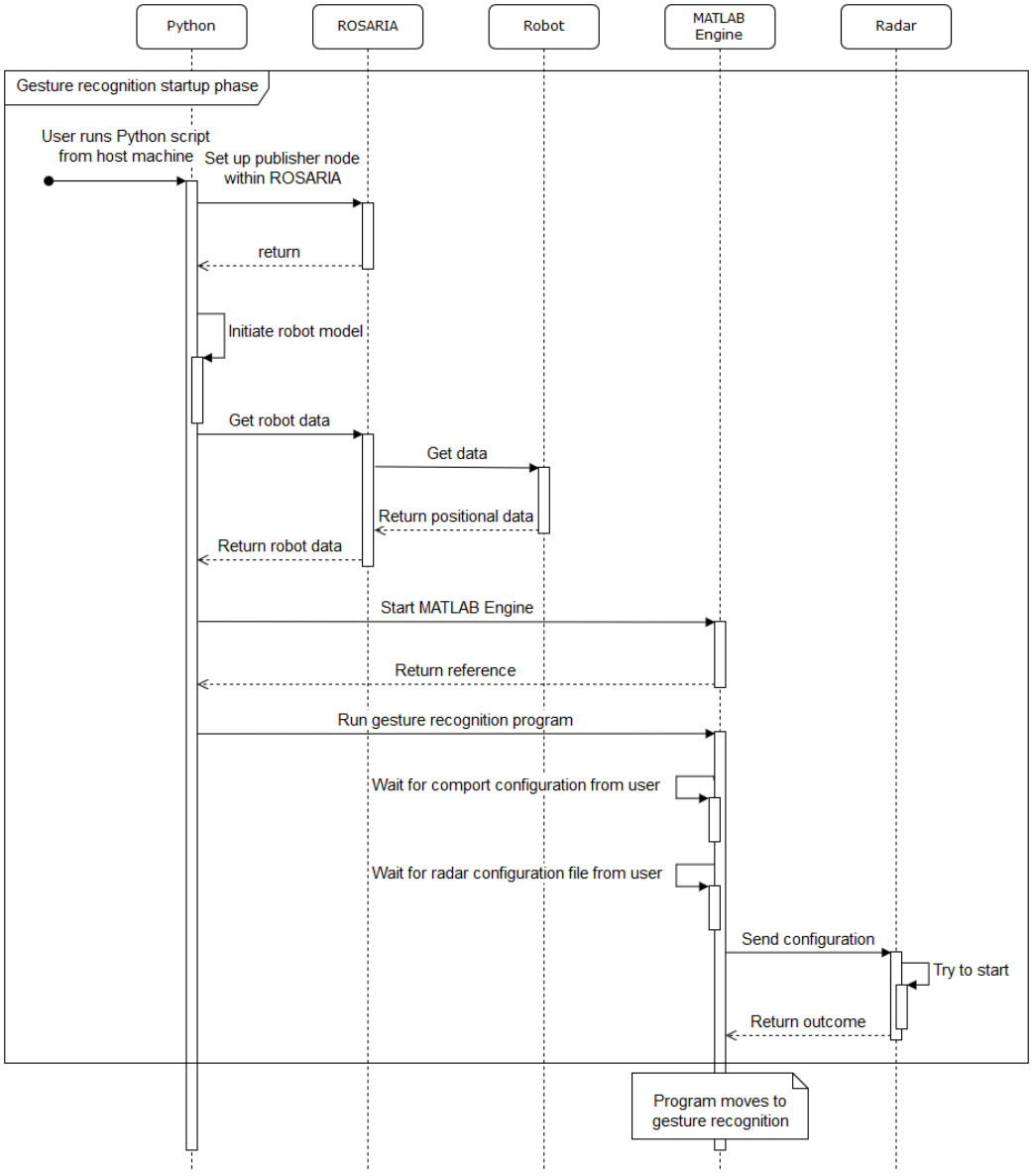


Figure B.2: Program system flowchart depicting the startup of the gesture recognition system.

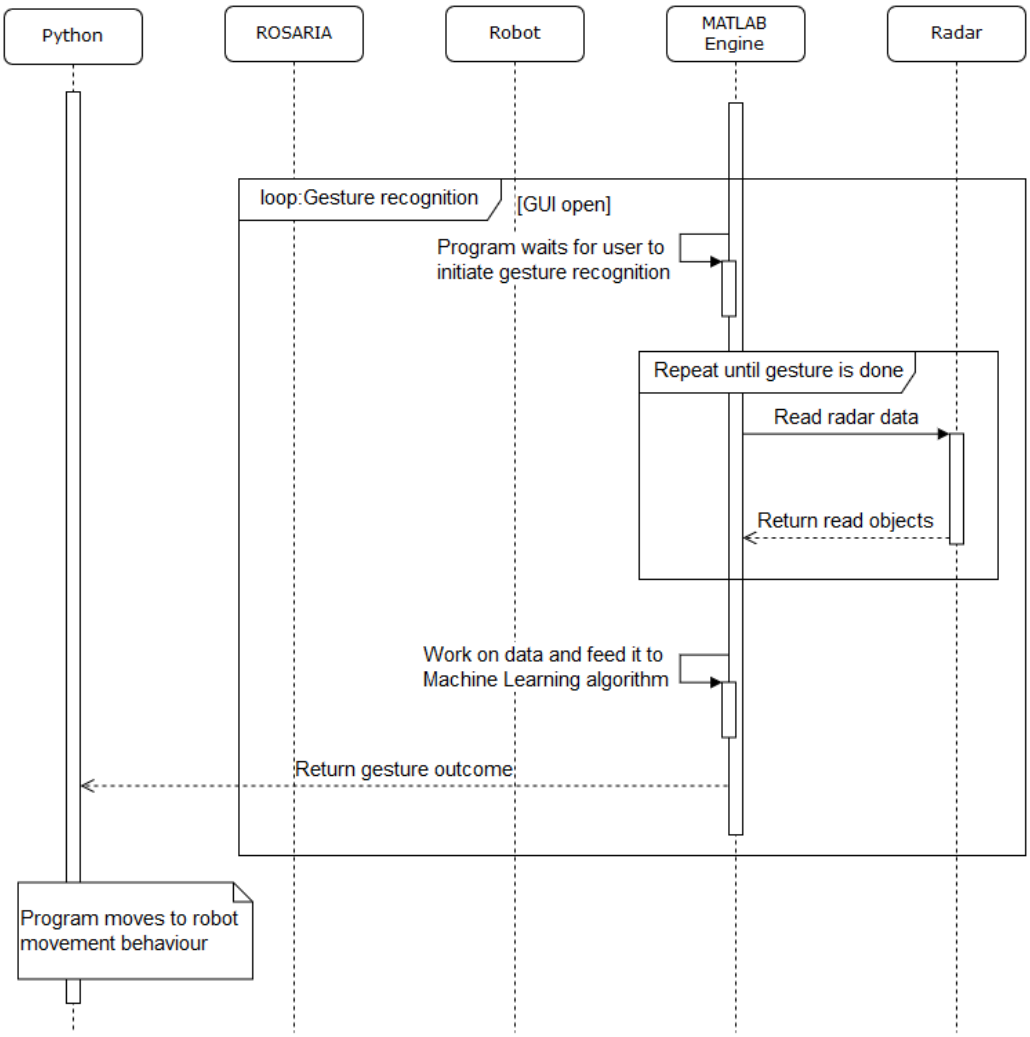


Figure B.3: Program system flowchart depicting the flow of the gesture recognition system.

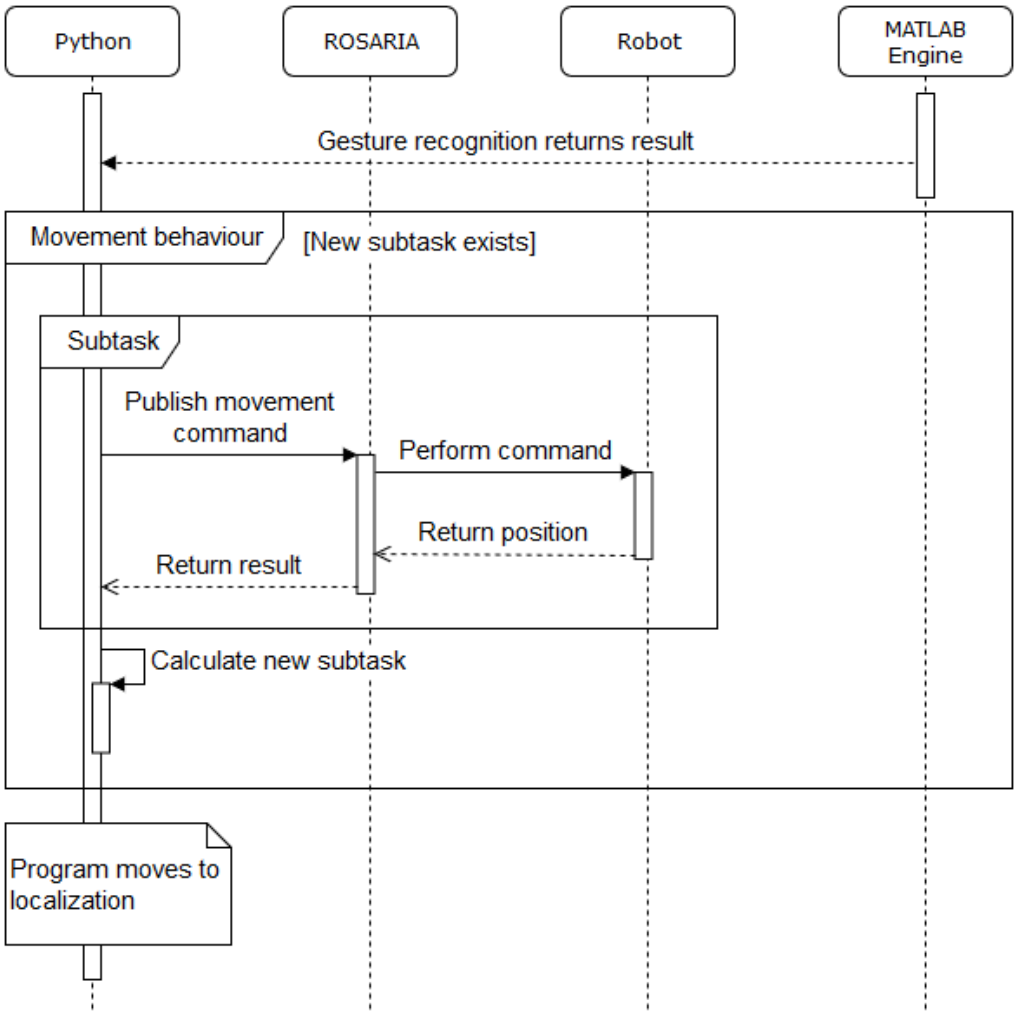


Figure B.4: Program system flowchart depicting the robot control system.

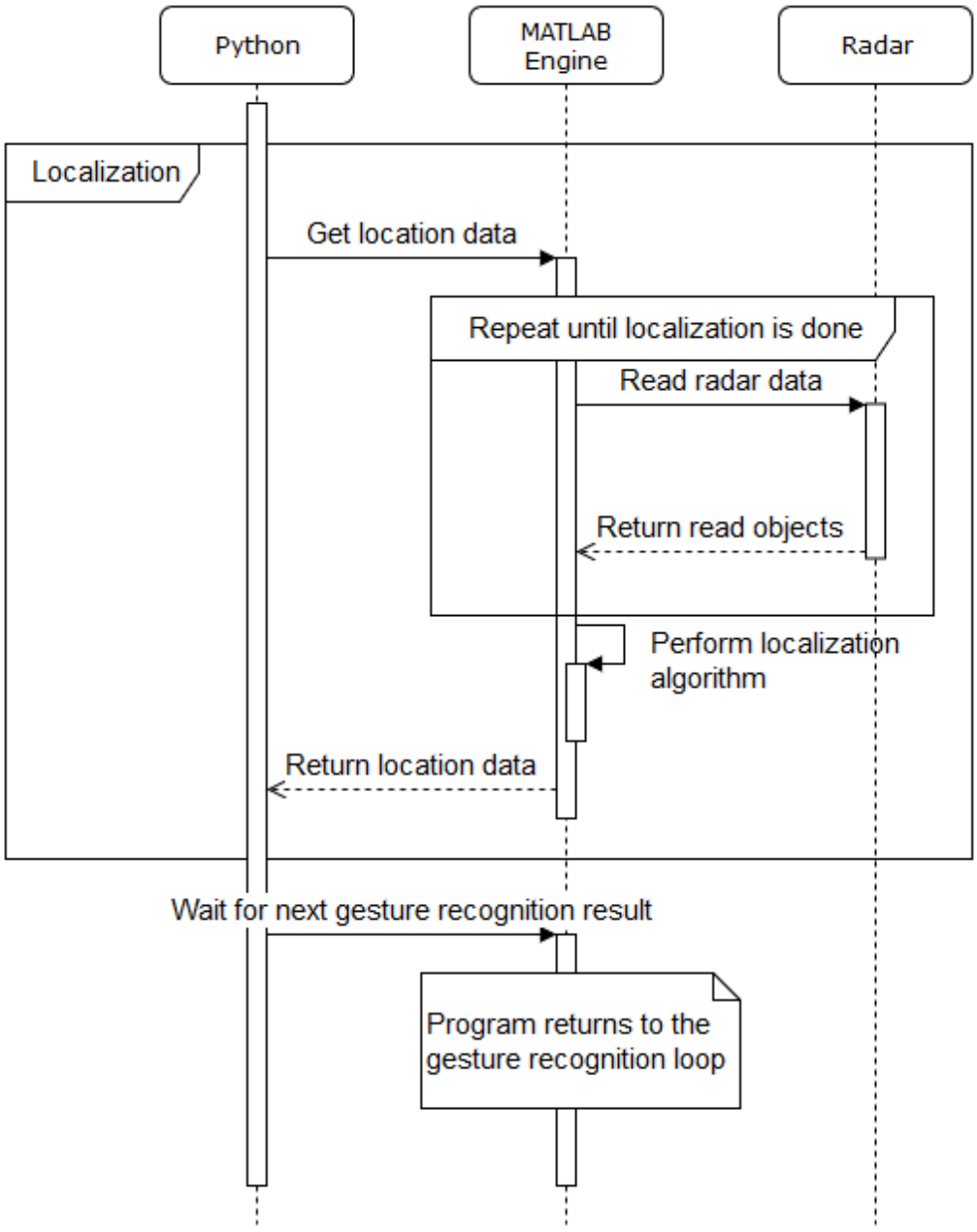


Figure B.5: Program system flowchart depicting the localization system.