



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

---

# Deep machine learning for gesture recognition in IoT applications

An extension of the tools of recognition

Bachelor's thesis in Electrical Engineering

JOHAN JAXING  
THOMAS JOHANSEN  
ANDREAS LINDSTRÖM  
AREN MOOSAKHANIAN





BACHELOR'S THESIS 2019: EENX15-19-08

# **Deep machine learning for gesture recognition in IoT application**

An extension of the tools of recognition

JOHAN JAXING  
THOMAS JOHANSEN  
ANDREAS LINDSTRÖM  
AREN MOOSAKHANIAN

Department of Electrical Engineering  
CHALMERS UNIVERSITY OF TECHNOLOGY  
Gothenburg, Sweden 2019

Deep machine learning for gesture recognition in IoT application  
An extension of the tools of recognition  
JOHAN JAXING  
THOMAS JOHANSEN  
ANDREAS LINDSTRÖM  
AREN MOOSAKHANIAN

© JOHAN JAXING, 2019.  
© THOMAS JOHANSEN, 2019.  
© ANDREAS LINDSTRÖM, 2019.  
© AREN MOOSAKHANIAN, 2019.

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 2019: EENX15-19-08  
Department of Electrical Engineering  
Chalmers University of Technology  
SE-412 96 Gothenburg  
Telephone +46 31 772 1000

Typeset in L<sup>A</sup>T<sub>E</sub>X  
Printed by Chalmers Reproservice  
Gothenburg, Sweden 2019



---

## Abstract

This project aims to exploit a millimetre wave radar and machine learning algorithms to perform tasks by allowing interaction with digital systems such as smart home units with hand gestures. The radar records data with over a bandwidth of 3.2 GHz and with the range and velocity resolution of  $d_{\text{res}} = 4.7$  cm and  $v_{\text{res}} = 58$  mm/s. These used specifications focus on measuring movements rather than hand shapes. The machine learning algorithm is implemented as a long short-term memory neural network, a class of recurrent neural network which allows the network to take previous frames into account for classifications, by allowing every layer to learn what previous data to keep. This is applicable to the sequential nature of the gesture data since a sequence of frames classified as some gesture increase the likelihood of further frames being the same gesture. The predictions made by the network on incoming data can be used to control, for example, a music application or some unit like the Google Chromecast<sup>TM</sup> or connected speakers. Based on our experiments some gestures achieved probabilities of correct classification of up to 78 %. These results show promise for the methods and further work can be made to improve this number or investigate different applications.

---

## Sammandrag

Projektets syfte är att använda millimetervågsradar och maskininlärning för att styra enheter för smarta hem med hand gester. Radarn används för att spela in data med focus på att uppfatta rörelser över handformer och maskininlärningsalgoritmen som används är ett Long Short-Term Memory neuralt nätverk. Detta specifika nätverk möjlig gör för klassificeringen att ta hänsyn till tidigare ramar i bedömningen av den senaste vilket passar datan sekventiella natur. Nätverkets klassificeringar används sedan för att kontrollera en musikapplikation vilken kan kontrollera enheter som Chromecast™ eller kopplade högtalare. Resultaten för metoden är lovande med sannolikheten för korrekt bedömning för vissa gester nådde 78 procent. En del genvägar behövdes dock tas på grund av begränsningar i tid för att nå dessa resultat. För att vidareutveckla projektet kan kontinuerlig data från radarn användas istället för diskret. Om överföringshastigheten kan förbättras eller slutapplikationen kan anpassas för lägre upplösning eller långsammare ramshastighet bör användning av den kompletta datamatriken från radarn övervägas. Databehandlingen bör genomföras tidigare under processen och mer tid kan användas till att undersöka andra maskininlärningsmodeller eller till att förbättra datan. Många olika applikationer kan undersökas.

---

# Contents

<b>Figures</b>	<b>vii</b>
<b>Tables</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Aim . . . . .	2
1.3 Problem . . . . .	2
1.3.1 Machine Learning . . . . .	2
1.3.2 Radar . . . . .	2
1.3.3 Data processing . . . . .	2
1.3.4 Use case . . . . .	2
1.4 Scope . . . . .	3
<b>2 Theory</b>	<b>5</b>
2.1 FMCW radar . . . . .	5
2.1.1 Range measurements . . . . .	5
2.1.2 Velocity measurement . . . . .	6
2.1.3 Angular measurement . . . . .	8
2.2 Machine Learning Algorithm . . . . .	10
2.2.1 Training the MLA . . . . .	10
2.2.2 Artificial neural network . . . . .	11
2.2.3 Convolutional neural network . . . . .	13
2.2.4 Recurrent neural network . . . . .	13
2.2.5 Optimizing . . . . .	14
<b>3 Equipment</b>	<b>15</b>
3.1 Hardware . . . . .	15
3.1.1 AWR1642 (Radar) . . . . .	16
3.1.2 Computers . . . . .	19
3.2 Software . . . . .	19
3.2.1 The mmWave SDK . . . . .	19
3.2.2 Python . . . . .	19
3.2.3 Tensorflow and Keras . . . . .	19
3.2.4 Integrated development environments . . . . .	19
3.2.5 GitHub . . . . .	20
<b>4 Method</b>	<b>21</b>
4.1 Selection of gestures . . . . .	21

4.2	Radar structure . . . . .	21
4.2.1	Radar measurement capabilities . . . . .	21
4.2.2	Data structure . . . . .	22
4.3	Gathering training data . . . . .	23
4.4	Preprocessing of data . . . . .	23
4.4.1	Capturing data . . . . .	23
4.4.2	Reformatting data . . . . .	23
4.4.3	Normalizing data . . . . .	24
4.4.4	Shuffling data . . . . .	25
4.5	Synthesized data . . . . .	25
4.6	Machine learning(ML) . . . . .	25
4.7	Applying the results . . . . .	25
<b>5</b>	<b>Results</b>	<b>27</b>
5.1	Application . . . . .	27
5.2	Hand gestures . . . . .	27
5.3	Radar . . . . .	29
5.4	Data processing . . . . .	29
5.5	Machine learning . . . . .	30
5.6	Post-processing . . . . .	31
<b>6</b>	<b>Discussion</b>	<b>35</b>
6.1	Tuning . . . . .	35
6.2	Machine Learning structure . . . . .	35
6.3	Data . . . . .	36
6.3.1	Radar . . . . .	36
6.3.2	Data gathering . . . . .	36
6.3.3	Data Processing . . . . .	37
6.4	Application . . . . .	37
6.5	Hand gestures . . . . .	37
6.6	Conclusions . . . . .	37
6.6.1	Future works . . . . .	38
	<b>References</b>	<b>41</b>
<b>A</b>	<b>Radar configurations</b>	<b>I</b>
A.1	Radar configuration file . . . . .	I
A.2	TI's mmWave radar commands . . . . .	II
A.2.1	chirp profile configuration . . . . .	V
A.2.2	Chirp configuration . . . . .	V
A.2.3	Frame configuration . . . . .	VI
<b>B</b>	<b>Statistical calculations</b>	<b>VII</b>
B.1	Finding gestures close to each other . . . . .	X
B.2	Probability in background . . . . .	X
<b>C</b>	<b>Machine learning optimizers</b>	<b>XI</b>
C.1	Adam optimizer . . . . .	XI
<b>D</b>	<b>Application results</b>	<b>XIII</b>



# Figures

2.1	Both figures are amplitude-time plots. The first demonstrates the input into the mixer from the transmitted signal as well as the reflected wave from the first and second chirp respectively. Note, that all the curves are identical except for a delay in time (extremely exaggerated). The second figure shows the phase shift in the two outputs from the mixer. Observe that both curves have the same constant frequency. . . . .	7
2.2	Shows two chirps begin sent out with $T_c$ between and the result after the range-FFT is applied. Note the phase difference $\Delta\phi = \phi_1 - \phi_0$ . Picture made with draw.io. . . . .	7
2.3	Illustrates the results for the chirp signals after the range-FFT is applied. Note that the results give that the object are at the same range, but with different phase. Doppler-FFT is then applied resulting in the separation of the objects. Picture made with draw.io. . . . .	8
2.4	Illustrates the radar and its scope, with detected objects as dots. Note $d_{\text{res}}$ , $\theta_{\text{res}}$ , $d_{\text{max}}$ , $V_{\text{max}}$ and $V_{\text{res}}$ in the figure. Made in Vectr. . . . .	8
2.5	The black box represents the radar. The blue rectangle illustrates a transmitting antenna Tx and the two black rectangles represent receiving antennas Rx. The black circle is the object. Note the distance between the receiving antennas, $\lambda/2$ and the difference in the distance to the object for the two receiving antennas, $\Delta d$ [7]. The figure is not in scale. Made in Vectr. . . . .	9
2.6	Visualization of parallel incoming signals, and the geometric relations between them. Made in Vectr. . . . .	9
2.7	Visual representation of a simple system . . . . .	10
2.8	An example on how MLAs adjusts itself over time. . . . .	11
2.9	Visual representation of an artificial neuron . . . . .	12
2.10	Graphical representation of a simple neural network [10]. . . . .	12
3.1	A flowchart of the hardware and software used in the project . . . . .	15
3.2	Picture of the AWR1642 evaluation module with relevant parts outlined [19]. Made using draw.io. . . . .	16
3.3	The setup of the radar's transceivers and receivers which allows for the double number of virtual receivers. . . . .	17
3.4	A illustration of range bins. Each bin represents a part of the continues range axis . . . . .	18
4.1	A flowchart demonstrating how data flowed between different programs, storages, and algorithms. . . . .	24

4.2	The format of a standard vector. First comes the amount of objects, then the objects' ranges from smallest to largest, the Doppler, peak x, and y. Values with the same index correspond to the same object. . . . .	24
5.1	Visualization of the button gesture. The sequence goes from left to right. .	27
5.2	Visualization of the swipe next gesture. The sequence goes from left to right.	28
5.3	Visualization of the swipe prev gesture. The sequence goes from left to right.	28
5.4	Visualization of the slide up gesture. The sequence goes from left to right.	28
5.5	Visualization of the slide down gesture. The sequence goes from left to right.	28
5.6	Visualization of the flop gesture. The sequence goes from left to right. . .	29
5.7	The probabilities for each gesture plotted against time, represented as frames. The probability is set to zero when it is lower than the background probability. Otherwise, the full probability is shown. . . . .	31
5.8	The figure presents the probabilities for each gesture over time where each step is a frame, where there is no delay between detections. The probabilities are only shown when they are greater than the probability of background. The dots represent a detection. . . . .	32
5.9	The figure presents the probabilities for each gesture over time where each step is a frame, where there is a delay of 10 frames between detections. The probabilities are only shown when they are greater than the probability of background. The dots represent a detection. . . . .	32
5.10	The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order. . . . .	33
A.1	Configure file (.cfg) used to configure the AW1642 mmWave radar. . . . .	I
A.2	Parameter explanation for mmWave API command "profileCfg" used in the configure file for the AWR1642 radar in the project. Made in draw.io. .	V
A.3	Parameter explanation for mmWave API command "chirpCfg" used in the configure file for the AWR1642 radar in the project. Made in draw.io. . .	VI
A.4	Parameter explanation for mmWave API command "frameCfg" used in the configure file for the AWR1642 radar in the project. Made in draw.io. .	VI
B.1	How an input sequence is divided into subgroups, where the input is assumed to follow (i). . . . .	VIII
D.1	The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order. . . . .	XIV
D.2	The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order. . . . .	XIV
D.3	The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order. . . . .	XV
D.4	The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order. . . . .	XV

# Tables

4.1	Shows the change in radar specifications when the radar parameters increase in value. . . . .	22
5.1	Number of frames and gestures used in the training and verification of the model. . . . .	29
5.2	Hyperparameters used in the training of the models. . . . .	30
5.3	The confusion matrix when training for three gestures and background. . .	30
5.4	The confusion matrix when training for six gestures and background. . . .	31
5.5	The confusion matrix for the application where the likelihood of getting a certain output given an input is shown. . . . .	33
5.6	The confusion matrix for the application where the likelihood of getting a double prediction (i.e a second prediction on an already predicted input) is shown given a certain output and input. . . . .	33
A.1	List of mmWave API commands used to configure the AWR1642 mmWave radar [29]. . . . .	II
B.1	The probability of not missing a gesture in per cent for different values of $L_G$ and $L_s$ and $\rho_G = \rho_B = 0,6$ . . . . .	VII
B.2	Values for (B.1) given different values for $x$ and $l_i$ , notice that the values where $x \approx l_i$ are the ones with highest probability . . . . .	VIII
D.1	The confusion matrix for the application where the amount of predictions are shown given a certain input and output. . . . .	XIII
D.2	The confusion matrix for the application where the amount of double predictions (i.e a second prediction on an already predicted input) are shown given a certain input and output. . . . .	XIII



# Glossary

**A.I.** artificial intelligence. 1

**Adam** adaptive moment estimation. 14

**ADC** analog to digital converter. 6, 16

**ANN** artificial neural network. 1, 11–13

**CNN** convolutional neural network. 13

**FFT** fast fourier transform. vii, 7, 8

**FMCW** frequency-modulated continuous-wave. 5, 16

**IDE** integrated development environment. 15, 19

**LSTM** Long Short-Term Memory. 13, 27

**MLA** Machine Learning Algorithm. vii, 2, 10, 11, 19, 22, 23, 25, 36

**RNN** recurrent neural network. 13

**SGD** stochastic gradient descent. 10, 11

# 1

## Introduction

### 1.1 Background

The pursuit of artificial intelligence (A.I.) comparable to human intelligence is a goal for a big part of the scientific community. The incredibly wide usage of a centralized intelligence that can learn from multiple sources at once, mostly exempt from human error, and be used over a wide range of subjects and distances simultaneously is almost endless. This is not where research is at though. The field of A.I. has a long history dating back to the middle of the 20th century with new methods discovered and explored throughout the latest decades [1]. Recent years have seen a rapid growth of the field and research has been divided into many sub-fields of application such as natural language processing and computer vision.

One such field is the field of gesture recognition. Gestures as a communication tool are widely used by humans, often complementing speech. Some of the main advantages of using gestures over speech are that they do not interrupt an ongoing conversation and can be used in a quiet or public space without drawing unwanted attention. One project exploring this is the Google project Soli [2], using millimetre wave radar and neural networks to interpret fine gestures. The methods are similar to those of computer vision, but the uses are more like those of natural language processing. It is a way for humans to interact and communicate with machines and can be evaluated as such.

A.I. can also be divided into different methods differentiating between supervised and unsupervised learning [1]. These methods can be broken down further by structure. Decision trees, rule-based classifiers, and neural networks are all examples of algorithms for supervised learning [1]. The method of machine learning being explored in this project is to exploit *artificial neural networks (ANNs)* implemented as a deep machine learning algorithm. An ANN is a computational model that is loosely based on the neural circuit found in animals [3] in that it is structured by a collective layer of nodes functioning similarly to neurons. These are connected to transmit signals to one another, process it and then signal an output node with the results. Deep machine learning is simply the method of using an ANN with multiple layers.

This bachelor thesis is based on another bachelor thesis which attempted to make a robot move by reading directional hand gestures [4]. In that project, an averaging algorithm was used to interpret data collected by radar. This limited the possible gestures to what that thesis refers to as linear gestures, gestures only differentiated by the average change in position of the hand, which in turn eliminated the need for machine learning to solve the problem [4]. A decision tree model did reach an accuracy of 96% on average per gesture which was considered good enough for the purposes of that project.

This bachelor thesis focuses on the subject of machine learning. By using machine learning to teach a computer to recognize and understand specific hand gestures, one

could then apply this however one sees fit. This thesis changes the scope by focusing more on the ability to perceive the hand gestures and the comprehension of what they mean which can be used for many use cases i.e, the use case is less important than the computation needed to perform arbitrary tasks.

## 1.2 Aim

The main task of the project is to explore the process of machine learning to train a neural network to recognize a specific set of subtle human hand gestures through a radar and its viability. The network can then be used to classify given gestures in order to extract an intended command for a computer system.

## 1.3 Problem

Beyond the specific aim, the project has been expanded to explore the viability of the machine learning through use cases and the possible applications thereof. This process has been divided into four parts, building and training the machine learning model, optimizing the radar settings, processing the data for the machine learning algorithm and applying the output to an application.

### 1.3.1 Machine Learning

The main problem in this project is to construct and train a *Machine Learning Algorithm (MLA)* with the purpose of being able to recognize the same gestures when presented live with an average accuracy of at least 95%. For this, a specific framework is picked and tuned for best result.

### 1.3.2 Radar

The radar will need to be able to collect data from the gestured as accurately and as frequently as possible. This will allow for more information to be feed into the MLA which will increase its learning capacity and allow for less divergence as a result of inaccurate reading.

### 1.3.3 Data processing

Once the radar can capture data sufficiently it will have to be processed in order to be supplied to the MLA. Ideally, the MLA would be able to process and understand the raw data directly. Otherwise, data manipulation can be used to extract useful features and eliminate others from the data. Once extracted the features will need to be transformed to be better suited for usage with a machine learning model [5].

### 1.3.4 Use case

After the neural network has been trained and is able to produce results these will need to be presented in some way. Therefore some use case needs to be implemented to allow a reactive action that would show a possible application for the project.

## 1.4 Scope

Since it's not feasible to train a network to interpret every possible hand gesture a limited set of gestures have to be selected. The selection will be made with consideration to the capability of the radar and what would fit a selected use case. The gestures would need to be clear enough for the radar to pick up and need to be intuitive for the user with regards to the desired action in the use case. The decision of what gestures are intuitive is based simply on the experience of the project group. The degree of intuition of the gestures does not affect the classification process but would apply to a marketed final product.

Another constraint is to focus solely on machine learning for solving the classification problem, no comparisons are made to other kinds of algorithms. The problem is well suited for machine learning and similar projects have had success with this approach [2].

The project will not research different kinds of radar or how they would affect the possible gestures which can be classified.

It is also important to note that the focus of the project is on exploring the classification and therefore the application to the use case is allowed to be somewhat simple.





# 2

## Theory

This chapter presents the theory this thesis is built on. The first section describes how the radar works, what kinds of data it can capture and the physics involved in doing so. The second section describes the theory behind different machine learning algorithms and their usage.

### 2.1 FMCW radar

In this project, a *frequency-modulated continuous-wave (FMCW)* radar is used, which measures the range, the radial distance, to the objects as well as the Doppler shift, the shift in frequencies between the measurements. The Doppler shift can be converted into a measurement of the relative speed of the objects moving towards the radar.

#### 2.1.1 Range measurements

To measure range, the radar sends out *chirps* which are transmitted electromagnetic signals. The signal's frequency increases linearly over time with a slope of  $S$  from a starting frequency  $f_s$  to an end frequency  $f_e$ , giving a total chirp time of  $T_c$ . Assuming there is an object in front of the radar at a distance of  $d$ , a chirp will reach the object in the time  $t = d/c$ , since it propagates at the speed of light,  $c$ . The signal is then reflected and arrives back at the radar after the same amount of time  $t$ , making the total flight time

$$\tau = \frac{2d}{c}. \quad (2.1)$$

At the radar, the reflected signal is received and sent to a mixer along with the originally transmitted signal. A mixer takes two sinusoids and gives the output

$$\begin{aligned} f_{\text{Mixer}}(x_1, x_2) &= f_{\text{Mixer}}\left(\sin(\omega_t t + \phi_t), \sin(\omega_r t + \phi_r)\right) \\ &= \sin\left((\omega_t - \omega_r)t + (\phi_t - \phi_r)\right), \end{aligned} \quad (2.2)$$

where  $x_1$  and  $x_2$  are the two input signals,  $\omega$  is the angular frequency and  $\phi$  is the phase of the signal. For the case of two signals which are sinusoidal in every instance, e.g. the reflected and original signal, the instantaneous output frequency equals the difference between the input signals' instantaneous frequencies and the output phase is simply the difference of the two input phases[6]. Assuming that there is only one object in front of the radar, the mixer output will be a signal of constant frequency, even though the input signals were ramping. The reason is that the received signal is the same signal as the original, only delayed by  $\tau$  [6]. So, both frequencies are ramping by the same factor,  $S$ . The difference between them is that the original signal is at the frequency  $f_s + S\tau$ ,

when the reflected signal reaches the mixer. Hence, the frequency of the mixer output is  $f_c = S\tau$ , which when combined with equation (2.1) gives,

$$d = \frac{cf_c}{2S}, \quad (2.3)$$

i.e. the distance of a single object in front the radar as a function of the mixer frequency.

When looking at more than one object, the mixer output will consist of several different frequencies. Therefore a range-FFT [6] is applied to the mixer output to extract the different distances of objects. The resolution of the range-FFT is given by

$$d_{\text{res}} = \frac{c}{2B}, \quad (2.4)$$

where  $B$  is the bandwidth swept by the chirp ( $B = ST_c$ ), and the maximum unambiguous range is given by

$$d_{\text{max}} = \frac{F_s c}{2S}, \quad (2.5)$$

where  $F_s$  is the analog to digital converters (ADCs) sampling rate of the radar.

### 2.1.2 Velocity measurement

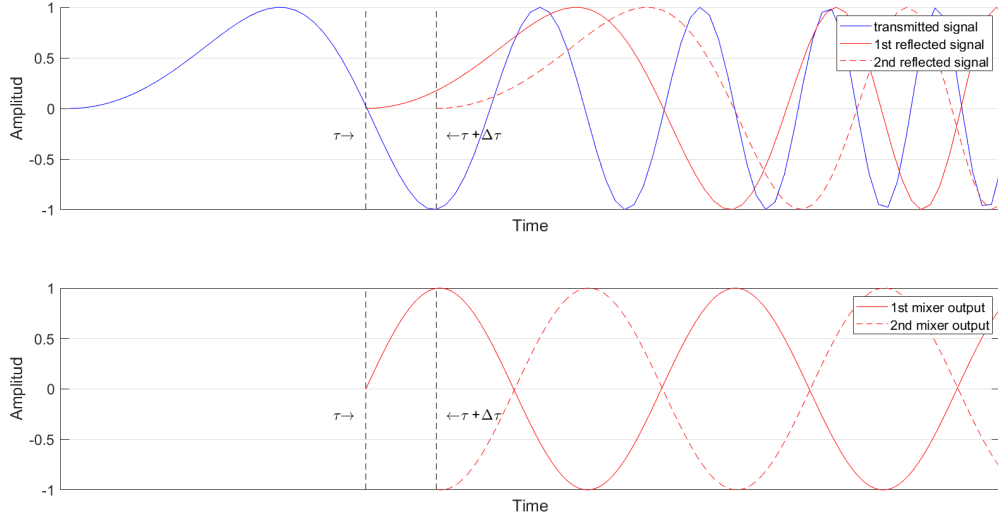
When an object is moving with a certain velocity towards the radar its distance from the radar  $d$  will change in between two chirps are sent. Since the time between two chirps is on the scale of microseconds, the change in distance will be much lower than the distance resolution, which is on the scale of centimetres. So, to measure the velocity, the other part of the mixer output is needed, the phase. As mentioned in the previous section, the phase of the mixer output is determined by the difference in the initial phase of the two signals. An example is shown in Figure 2.1, the phase difference is zero between the original and the first signal but  $\pi/2$  between the original and the second signal. The phase of a signal varies from 0 to  $2\pi$  for each cycle of the signal. Therefore, the phase difference is simply

$$\Delta\phi = 2\pi f_s \Delta\tau. \quad (2.6)$$

The phase difference can also be represented by the difference in distance travelled by the two chirp signals, by combining equation (2.6) and equation (2.1),

$$\Delta\phi = \frac{4\pi\Delta d}{\lambda_s}, \quad (2.7)$$

where  $\lambda_s$  is the start wavelength.



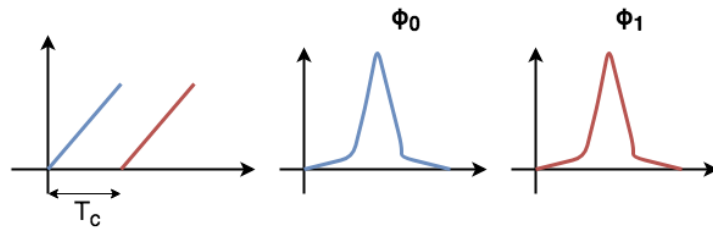
**Figure 2.1:** Both figures are amplitude-time plots. The first demonstrates the input into the mixer from the transmitted signal as well as the reflected wave from the first and second chirp respectively. Note, that all the curves are identical except for a delay in time (extremely exaggerated). The second figure shows the phase shift in the two outputs from the mixer. Observe that both curves have the same constant frequency.

The distance traveled by the object between two chirps is  $\Delta d = vT_c$ , with  $v$  being the velocity of the object in the radial axis of the radar. The velocity can now be expressed as [6]

$$v = \frac{\lambda_s \Delta\phi}{4\pi T_c}. \quad (2.8)$$

Since the maximum unambiguous phase difference between chirps is  $\Delta\phi_{\max} = \pi$ , then the maximum unambiguous velocity can be described as

$$v_{\max} = \frac{\lambda}{4T_c}. \quad (2.9)$$



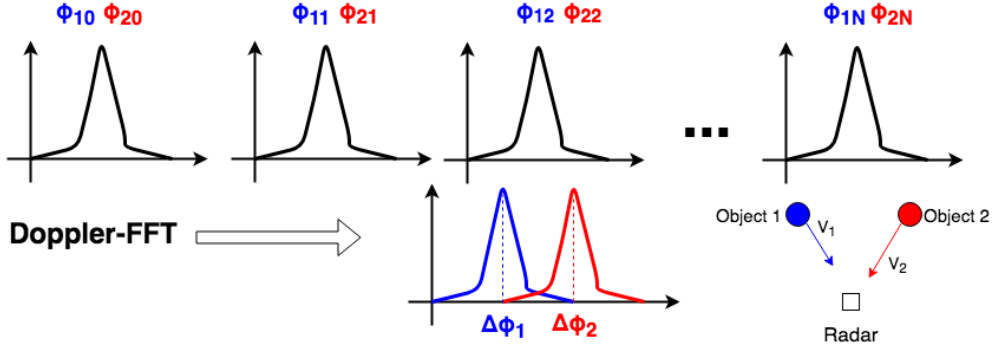
**Figure 2.2:** Shows two chirps begin sent out with  $T_c$  between and the result after the range-FFT is applied. Note the phase difference  $\Delta\phi = \phi_1 - \phi_0$ . Picture made with draw.io.

However, this method is only valid if there is at most one objects at a given range. To differ between objects at the same range a sequence of  $N_c$  chirp, called a frame, is used. The range-FFT, mentioned in the previous section, is applied to every chirp which separates each object by range. To this result another FFT is applied, called the Doppler-FFT [6]. The Doppler-FFT makes it possible to separate the object at the same range,

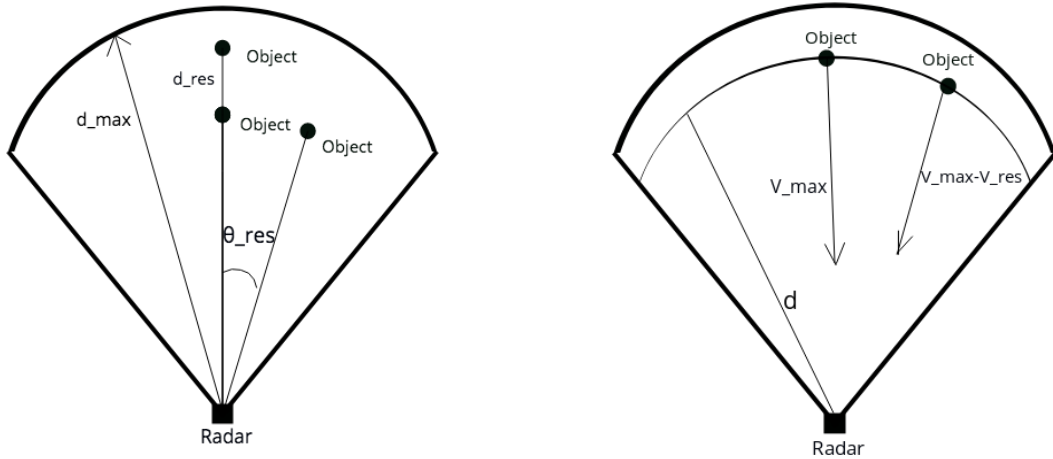
by velocity, as seen in Figure 2.3 , with the resolution of

$$v_{\text{res}} = \frac{\lambda}{2T_f}, \quad (2.10)$$

. where  $T_f = T_c N_c$  is the frame time[7]. Figure 2.4 illustrates  $v_{\text{res}}$  and  $v_{\text{max}}$ .



**Figure 2.3:** Illustrates the results for the chirp signals after the range-FFT is applied. Note that the results give that the object are at the same range, but with different phase. Doppler-FFT is then applied resulting in the separation of the objects. Picture made with draw.io.



**Figure 2.4:** Illustrates the radar and its scope, with detected objects as dots. Note  $d_{\text{res}}$ ,  $\theta_{\text{res}}$ ,  $d_{\text{max}}$ ,  $V_{\text{max}}$  and  $V_{\text{res}}$  in the figure. Made in Vectr.

### 2.1.3 Angular measurement

When estimating the angle of arrival for an object compared to the normal angle of the radar, two receiving antennas must be used. When a signal is sent, the two receivers will have a slight phase difference depending on the angle of arrival. The difference is given by

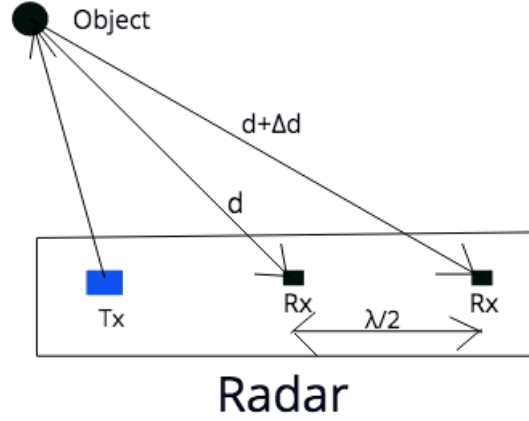
$$\Delta\phi = \frac{2\pi\Delta d}{\lambda}, \quad (2.11)$$

with  $\Delta d$  being the difference in distance traveled by the signal for the two receiving antennas [7], as illustrated in Figure 2.5, note that this distance is only from the object to the receivers and not both ways which gives the factor 2 instead of 4 in Equation (2.7).

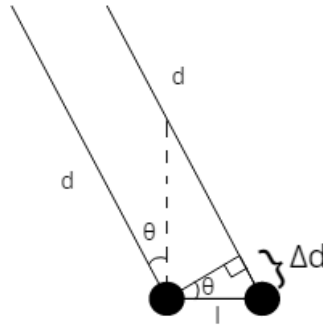
The distance can be approximated using basic geometry, given that the signals arrive approximately parallel to each other,

$$\Delta d = l \sin(\theta), \quad (2.12)$$

with  $l$  and  $\theta$  corresponding to the length between the two receiving antennas and the angle of arrival for the object respectively, see Figure 2.6.



**Figure 2.5:** The black box represents the radar. The blue rectangle illustrates a transmitting antenna Tx and the two black rectangles represent receiving antennas Rx. The black circle is the object. Note the distance between the receiving antennas,  $\lambda/2$  and the difference in the distance to the object for the two receiving antennas,  $\Delta d$  [7]. The figure is not in scale. Made in Vectr.



**Figure 2.6:** Visualization of parallel incoming signals, and the geometric relations between them. Made in Vectr.

Combining equation (2.11) and equation (2.12) we get the angle of arrival

$$\theta = \sin^{-1} \left( \frac{\lambda \Delta \phi}{2\pi l} \right). \quad (2.13)$$

It is worth noting that  $\Delta \phi$  has a nonlinear dependency with respect to  $\theta$ . Meaning that  $\theta$ 's estimation validity depends on what value the real  $\theta$  has. The estimation is more correct when  $\theta$  is small[6].

The maximum angle of arrival is given by

$$\theta_{\max} = \pm \sin^{-1} \left( \frac{\lambda}{2l} \right). \quad (2.14)$$

With a spacing of  $\lambda/2$  between the receiving antennas, the maximum angle of  $\theta_{\max} = \pm 90^\circ$  is achieved [7].

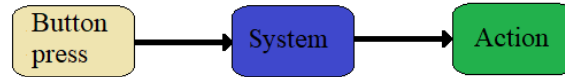
To distinguish two objects at different angles, a similar method as the one mentioned in the previous section is used. First are signals divided into different ranges by a range-FFT followed by the use of a 2D-FFT[6], which corresponds to the Doppler-FFT. Instead of looking at the number of chirps, as the Doppler-FFT does, the 2D-FFT uses the number of receivers,  $N_r$ , which gives

$$\theta_{\text{res}} = \frac{\lambda}{N_r l \cos(\theta)}. \quad (2.15)$$

The reason to why this is so unlike Equation (2.13) is the nonlinearity of the problem[6].

## 2.2 Machine Learning Algorithm

Most computer systems can be simplified as a sequence of input, processing, and output as in Figure 2.7. When considering a system built to recognize a button press and make an action correlating to the said button press, it is not difficult to see how this could be coded by hand into an application. The system used in the project, where the data collected with the radar needs to be recognized as a specific hand gesture, becomes more complicated. When the input is something as varying as human gestures, every possible starting position, speed, hand shape, slight deformity, and angle need to be individually included into what the application takes into account in order to make it run with any kind of accuracy. The amount of time it would take to manually code this would make the task next to impossible. For that reason, this system is built around a machine learning algorithm instead.



**Figure 2.7:** Visual representation of a simple system

### 2.2.1 Training the MLA

The MLA used is trained using supervised learning. This means that the algorithm is fed the input data( $x$ ) and the output label( $y$ ) separately, similar to

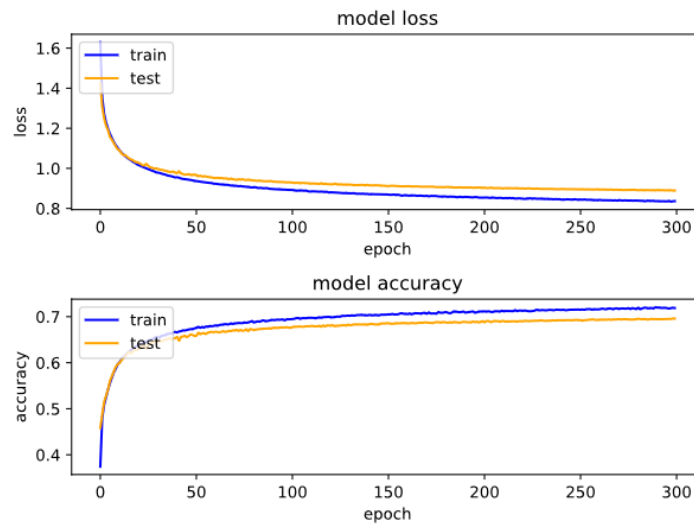
$$y = f(x). \quad (2.16)$$

The  $f(x)$  is the mathematical function the MLA creates and fits to the data. For each input data, the algorithm makes a prediction the likelihood for it belonging to each specific label according to current values in the function. This is then compared with the correct label( $y$ ) to calculate the loss value, which the algorithm uses with an algorithm for stochastic gradient descent (SGD) to make adjustments to its function[8]. Gradient descent is a way to find the slope of the function between parameters, similar to partial derivatives, as seen in Equation 2.17.

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right) \quad (2.17)$$

The MLA is taught in iterations which allows the algorithm to use the gradient descent to minimize the loss value, calculated through categorical cross-entropy[9]. More details in 2.2.5.1.

The batch gradient descent, which is more common, uses the entire data set to calculate the descent. The stochastic gradient descent instead selects a few samples out of the data set to use for the gradient descent. Using SGD, the MLA can calculate in which direction the slope is angled down. It then uses the variable learning rate to know how much the parameters need to be adjusted. An example of how a MLA learns and improves its own loss value and accuracy for each iteration can be observed in Figure 2.8.



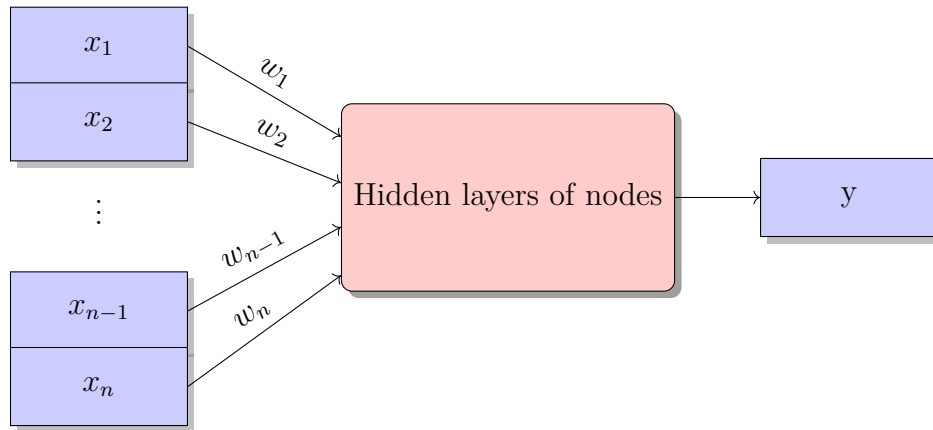
**Figure 2.8:** An example on how MLAs adjusts itself over time.

The calculations of the gradient descent, the handling of the learning rate and adjustment of the parameters takes care of by the optimizer. More details on 2.2.5.

## 2.2.2 Artificial neural network

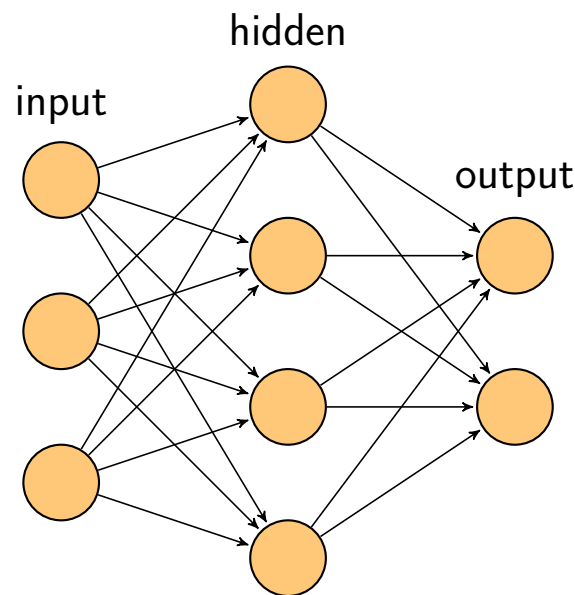
Artificial neural network (ANN) is a machine learning algorithm based on the neural circuit found in animals [3]. The simplest kind of ANN is a perceptron consisting of only one neuron. It can receive multiple inputs which are all multiplied by a weight before being fed to an activation function [1, pp. 90-91]. This activation function produces an output based on the weighted sum of inputs. Many kinds of functions can be used but popular ones are the step, sign, linear and sigmoid functions[1, p. 92]. A bias can be introduced to change the activation threshold of the function by adding an additional input with a value of one with this bias as weight.





**Figure 2.9:** Visual representation of an artificial neuron

A typical model for machine learning is composed of multiple layers of perceptrons. Data is entered at an input layer and passes through one or multiple hidden layers before reaching an output layer collecting and presenting the processed data to an output format. Figure 2.10 shows a simple neural network with one hidden layer.



**Figure 2.10:** Graphical representation of a simple neural network [10].

The process of training this basic neural network is the same as for more complex ones. Through backpropagation, the weights and the bias is updated based on a calculated prediction error until this error is minimized [1]. More on this process in Section 2.2.5.

### 2.2.2.1 Deep learning

While ANN and other types of algorithms use a single layer of nodes to analyze the data, a deep machine learning algorithm uses multiple hierarchical levels of hidden layers of nodes. The hierarchical function of deep learning systems enables machines to process data with a nonlinear approach [11]. Put in the simplest of terms: “The more layers the algorithm has, the deeper it is”.

### 2.2.3 Convolutional neural network

Convolutional neural network (CNN) is a class of deep neural network whose main use is to recognize and categorize different images. This includes logos, face recognition and street signs. A CNN “reads” an image as a collection of pixels which is sorted in an array with the axes  $\text{height}(i) \times \text{width}(j) \times \text{dimensions}(k)$  as axis. This array will pass through multiple convolutional layers, each with nodes with weights. For each hidden layer, the information on the pixels are handled with the weights and more or less dense with the equation:

$$V_{ijk} = g\left(\sum_{pqr} w_{pqr} x_{p+i-1, q+j-1} - \theta_k\right) \quad (2.18)$$

Where  $V_{ijk}$  is the neuron at the specific coordinates  $(ijk)$  and the result of  $g(b)$  that is an activation function. Also,  $w$  is the weight in the specific node,  $x$  is the specific pixel in the input array and  $\theta$  is the bias [3, p. 127]. At the end of the network, there is a layer that classifies the image with a probabilistic value between 0 and 1. The prediction is decided by the class with the highest probability.

### 2.2.4 Recurrent neural network

Recurrent neural network (RNN) is a class of ANN that can be used when a simpler layout is not enough. While a simple form of ANN usually is only feed-forward, as seen in Figure 2.10, this class of layer has feed-forward with feedbacks. This means that a node in the RNN to receive the output of a node down the line or even within the same layer [3, p. 147]. The algorithm applied is:

$$V_i^\ell = g\left(\sum_j w_{ij}^\ell V_j^{\ell-1} - \theta_i^{(\ell)}\right) \quad (2.19)$$

on the  $i$ -th node in layer  $\ell$ . This allows a RNN to recognize a data’s sequential characteristics and use patterns to predict the next likely scenario. It does this by using loops to access previous predictions together with new data to process the input as a sequence. The most common uses for RNN’s is language modeling and speech recognition.

One of the reasons RNN has such a great appeal is that it allows for predictions over extended periods of time. An example of this is that it can allow a prediction of the next word in a text by looking at previous words in a sentence. The problem with RNNs is that the connection to earlier predictions decays with distance [12].

#### 2.2.4.1 Long Short-Term Memory network

Long Short-Term Memory (LSTM) network is a specific architecture-type of RNN built as chains of modules each constructed by four neural network layers interacting. These interactions allow the network to make decisions about which data to keep or forget [12] which better preserves some connections between recent and older data. It does this by having a somewhat altered algorithm from what is used for the RNN:

$$V_j^{(\ell)} = g\left(\sum_k w_{jk}^{\ell, \ell-1}\right) V_k^{\ell-1} - \theta_j^{(\ell)} + \sum_n w_{jn}^{(\ell, \ell-2)} V_n^{(\ell-2)} \quad (2.20)$$

### 2.2.5 Optimizing

When the model is defined, layers and nodes are built and connected, allowing the model to produce an output based on an input. However, in order to train on data and improve these outputs two more things are needed. The first one is a loss metric. A numerical constant if the output was correct or not and how far-off it was. The second thing is an optimizer. A function used to backpropagate the results and updating the values of the nodes. Essentially applying the learning to the model.

While there are many loss and optimizer functions, this project is mainly built around one of each; the categorical cross-entropy loss function [13] and the Adam optimizer [14].

#### 2.2.5.1 Categorical cross-entropy

Categorical cross-entropy is a method for calculating the difference between a predicted result and an actual value. It is used when the desired value is a one hot vector representing a single correct class out of a number of classes and the output is a vector with each index representing the predicted probability of that class being correct. Using categorical cross-entropy these vectors can be compared to calculate a loss value used for optimization [13].

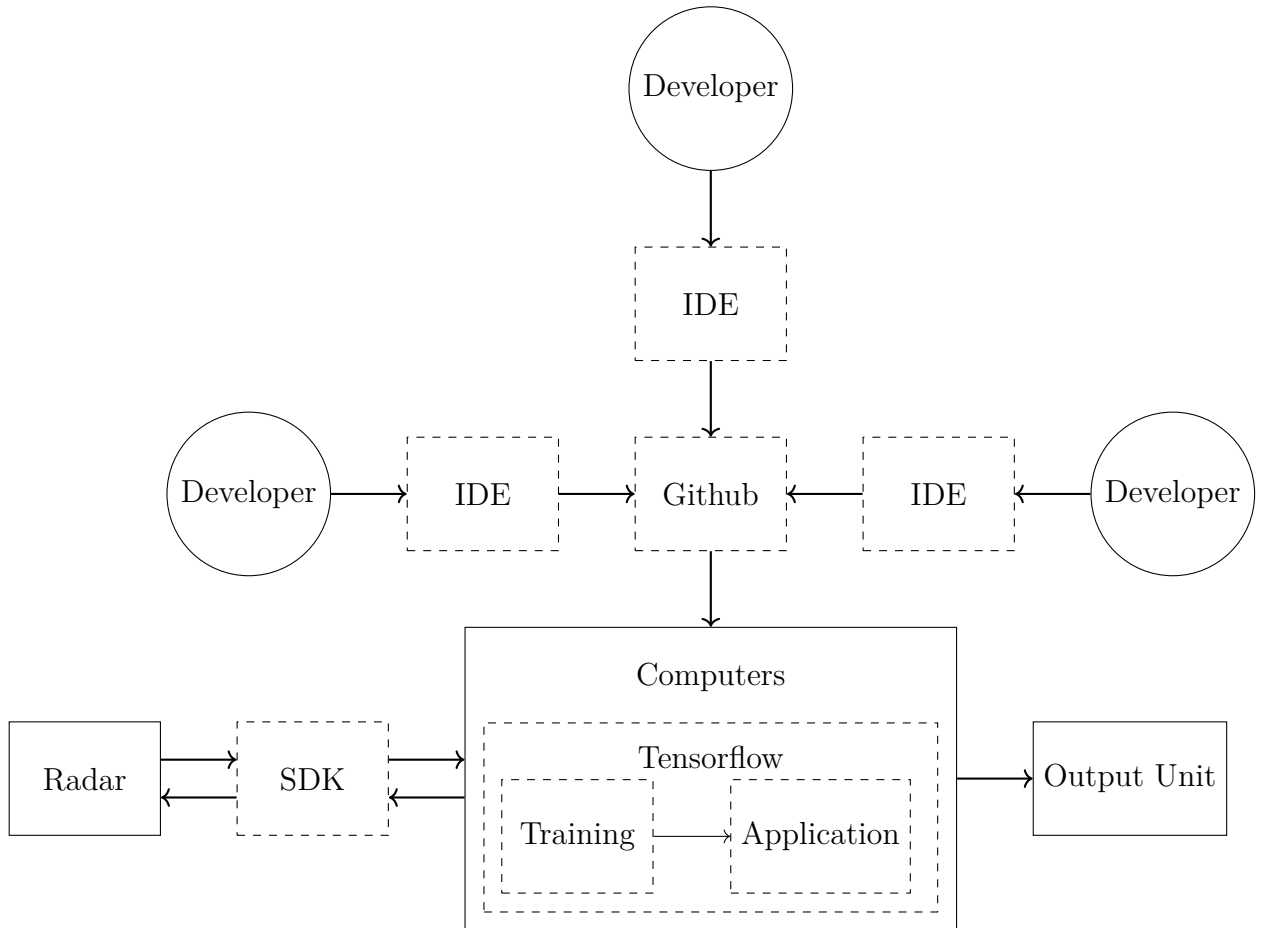
#### 2.2.5.2 Adaptive Moment Estimation

Adaptive moment estimation (Adam), is an algorithm for efficient stochastic optimization, using stochastic gradient descent, which is used to find a locally optimum solution to an optimization problem. Stochastic optimization is best used when slightly less than the optimal solution is adequate, and the algorithm can't be tailored exactly to the problem[15]. Adam was proposed as an optimization method for machine learning in 2015 combining two popular methods, AdaGrad [16] and RMSProp [17], to create a generally applicable method. The convergence of the algorithm was analyzed, and experiments were made to prove that it performs on par with or better than other algorithms for different kinds of machine learning problems [14]. Pseudo-code for the algorithm can be found in Appendix C.1.

# 3

## Equipment

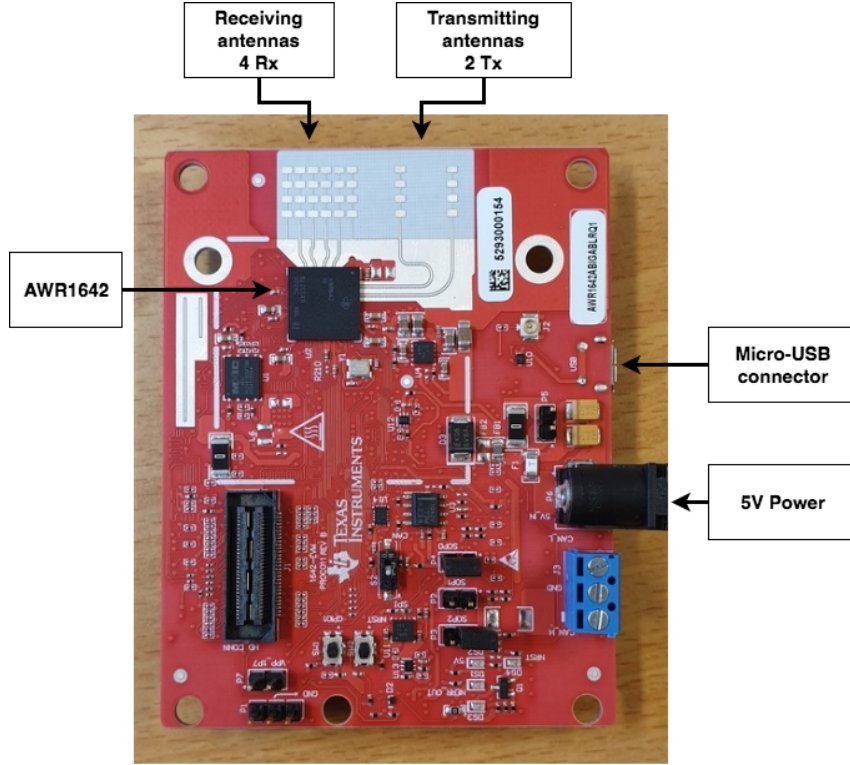
This chapter will present and describe the software and devices used in the project. A single-chip radar was used alongside multiple kinds of software including necessary software to access and control the radar, frameworks and API's enabling the machine learning and IDEs to improve coding efficiency.



**Figure 3.1:** A flowchart of the hardware and software used in the project

### 3.1 Hardware

The AWR1642 chip was the only specific hardware used for this project. This section describes the specifications of the chip, how the theory applies to the hardware and the data structure used to handle and access the recorded data.



**Figure 3.2:** Picture of the AWR1642 evaluation module with relevant parts outlined [19]. Made using draw.io.

### 3.1.1 AWR1642 (Radar)

The radar used in the project is “AWR1642 single-chip 76-GHz to 81-GHz automotive radar sensor evaluation module”. It is one of the radars in TI’s FMCW mmWave radar series. The AWR1642 has 2 transmitting antennas with a spacing of  $2\lambda$  between them and 4 receiving antennas with a spacing of  $\lambda/2$  between them. The radar is powered by a 5V connection using Biltrema’s universal adapter 5V 2.25A. The board can be connected to a computer using a micro-USB cable.

Some specifications for the board are its baud rate at 921600 bits per second, maximum bandwidth at 4 GHz, maximum chirp slope at  $100 \text{ MHz}/\mu\text{s}$  and maximum sampling rate at 5 MHz. More specific specifications about the AWR1642 evaluation board can be found in the manual at [18].

#### 3.1.1.1 Hardware measurements

When using actual hardware, like the project’s AWR1642, some problems will arise when transitioning from the theoretical calculations.

The maximum range of the radar is still Equation (2.5), since it only depends on the ADCs sampling rate. The resolution, however, is dependent on the frequency bandwidth swept by the measurement. The chirp still sweeps the full 4 GHz but the ADCs sampling rate and the amount of samples,  $N_s$ , gives a sampling time of,  $N_s/F_s$ . The sampling time multiplied with the slope gives the measured bandwidth,

$$B = \frac{N_s S}{F_s}, \quad (3.1)$$

which makes the resolution

$$d_{\text{res}} = \frac{F_s c}{2SN_s}, \quad (3.2)$$

according to Equation (2.4). The full bandwidth could no be used since amount of samples had to be in the form of  $N_s = 2^n$ , where  $n$  is a positive integer[20].

To calculate the maximum velocity, the actual chirp time is needed. The actual chirp time differs from the theoretical since the radar computes the range-FFT in between every chirp. These calculations need to be finished before the next chirp is done [20]. To solve this, the chirp has an added ideal time between the chirp. This makes the actual chirp time

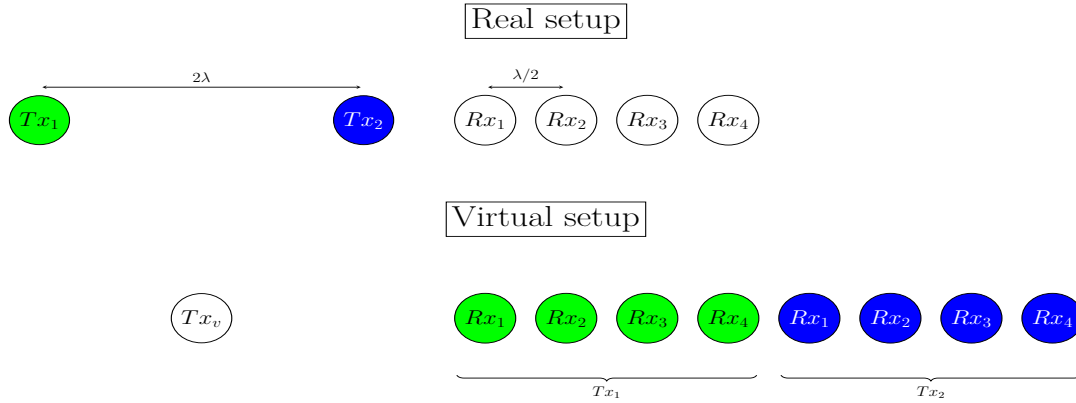
$$T_{\text{actual}} = T_{\text{idle}} + T_{\text{chirp}}, \quad (3.3)$$

so to optimize the actual chirp time the ideal time needs to be as small as possible. All in all, this makes the maximum velocity

$$v_{\text{max}} = \frac{\lambda}{4T_{\text{actual}}}. \quad (3.4)$$

The frame time, controlling the velocity resolution, is as a consequence  $T_f = T_c N_c$ . The true time between frames is longer since, similarly to  $T_{\text{actual}}$ , there is a need for time to make calculations and to send the data to the computer.

For the angular calculations, the maximum is still the same but for the resolution, a concept called virtual antennas is used which improves the resolution[6]. There are four receivers but there are also two transmitters at a distance of  $2\lambda$ . All of the transmitters and receivers lie on the same line, which combined with that chirp are send in the alternating pattern of  $(Tx_1, Tx_2, Tx_1, \dots)$  which makes the amount of virtual receivers, 8, double the amount of regular receivers, as seen in Figure 3.3.



**Figure 3.3:** The setup of the radar's transceivers and receivers which allows for the double number of virtual receivers.

### 3.1.1.2 Data Structure

The data collected by the AWR1642 mmWave radar is stored in packages. The package structure for the radar is designed such that the user can, through the mmWave SDK, choose which parts of the package the radar should send. One whole package is sent every frame, which parts it contains is set in the configure file for the radar [21].

The structure initiates with the Header, which contains information about the whole package. The Header always follows the same formula and consists of 36 bytes. It contains

the version, total packet length, platform, frame number, time, number of detected objects, and number of data structures in the package. The Header also contains a “magic word” that signals the start of a packet [21]. The Header is the only mandatory part of the package and cannot be unselected by the mmWave API commands.

The next part of the structure is Detected Objects. It initiates with 12 bytes of information containing a structure tag, the length of the structure, and a descriptor. The Descriptor represents the number of objects detected and the Q-format the data is in [21]. The Detected Object payload also contains the range, velocity, intensity, and angle information for every object detected by the radar. More specifically, the features acquired are range index, Doppler index, the peak value of intensity, x-value, y-value and z-value. The size for the package is therefore dependent on the number of objects detected. The size can be described as [21]

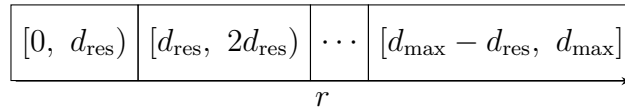
$$\text{size} = 12 + N_{\text{objects}} \cdot 12 \text{ bytes}, \quad (3.5)$$

where  $N_{\text{objects}}$  is the number of objects.

The object detection is followed by Range Profile and Noise Profile, both with the size of

$$\text{size} = \# \text{RangeBins} \cdot 4 \text{ bytes}, \quad (3.6)$$

where range bins are the discrete boxes in which the range-FFT divides the signals, see Figure 3.4, and there are  $N_s$  of them. Both profiles contain a structure tag and the length of the structure. They also contain a 1D array of data, where the range profile has a 1D array of log magnitude range FFTs and the noise profile have a 1D array of data which is considered noise by the radar [21].



**Figure 3.4:** A illustration of range bins. Each bin represents a part of the continues range axis

The next part of the structures are the heat maps, more specific the Range-Azimuth Heat Map and the Range-Doppler Heat Map. They both contain a structure tag, the length of the structure and a heat map [21]. A heat map is a matrix whose elements consist of intensities and its indexes are arbitrary quantities.

The Range-Azimuth Heat Map contains a range-Azimuth heat map, which column indices represent the range and row indices represents angle to the normal of the radar. The size of the range Azimuth heat map is given by

$$\text{size} = \# \text{rangeBins} \cdot \# \text{virtualAntennas} \cdot 4 \text{ bytes}. \quad (3.7)$$

The Range-Doppler Heat Map contains a range-Doppler heat map. This heat map has range values as its column indices and Doppler values as its row indices. The size for the range-Doppler heat map is given by

$$\text{size} = \# \text{rangeBins} \cdot \# \text{DopplerBins} \cdot 4 \text{ bytes}, \quad (3.8)$$

where Doppler bins, works in the same way as range bins, only for Doppler and the amount is given by

$$\# \text{DopplerBins} = \frac{\# \text{Chirps per frame}}{\# \# \text{Transmitting antennas}} \quad (3.9)$$

The end of the data package sent by the radar is Statistics Profile and Padding. The Statistics Profile has a size of 32 bytes and contains the structure tag, length of the structure, inter-frame processing time, transmit output time, inter-frame processing margin, inter-chirp processing margin, active frame CPU load, and inter-frame CPU load. The Padding is a filler, with size between 0-31 bytes, to ensure that the package is the right length, a multiple of 32 bytes [21].

### 3.1.2 Computers

When recording gestures, a Lenovo laptop of model 720S-13IKB (Type 81BV) was used [22]. For training the machine learning model various standard, commercial laptops and desktops were used.

## 3.2 Software

Multiple kinds of software were used in this project. Millimeter Wave Software Development Kit (mmWave-SDK) was used to configure the radar. Python was used with tensorflow and Keras for the machine learning and for the development application. Finally, the IDEs PyCharm and Visual studio Code and GitHub were used to facilitate coding and collaboration.

### 3.2.1 The mmWave SDK

mmWave Software Development Kit (SDK) is a program developed by Texas Instruments as an Application Programming Interface (API) for their mmWave radars. The mmWave API commands used in this project are explained in table A.1. The chirp and frame configuration commands are explained in more detail in Appendix A.2.1, A.2.2 and A.2.3.

### 3.2.2 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics with built-in data structures that allow for dynamic coding and structuring [23]. Python 3.6 was used in this project. Its simplicity as a language allowed for easy structuring of the MLA and coding of the application.

### 3.2.3 Tensorflow and Keras

Tensorflow and Keras were used to easily construct the machine learning model. Tensorflow is an open source platform for machine learning which allows building and training of machine learning models with varying levels of complexity [24]. Keras is a high-level API running on top of Tensorflow to allow easy structuring of models [25].

### 3.2.4 Integrated development environments

An integrated development environment (IDE) is a software suite that consolidates the basic tools required to write and test software. It allows for easy development as the



inclusion and use of the text editor, available libraries and compiler all comes included [26].

#### **3.2.4.1 Pycharm and Visual Studio Code**

PyCharm and Visual Studio Code were the most used development environments and functioned interchangeably. They both allow for debugging and convenient management of Python's virtual environments and packages.

#### **3.2.5 GitHub**

GitHub is a Git repository hosting service used for collaboration on code and version control. It allows multiple people working on the same code to synchronize, branch off and merge for efficiency.

# 4

## Method

In order to complete the task stated, gestures were selected with a degree of intuition based on the project group's experience and detectability tested with the radar. The required radar measurement capabilities were determined by the gestures and based on this the proper configuration was calculated.

The amount of data that could be used was calculated by looking at the baud rate of the transmission between the radar and the computer together with the size of different data structures and the desired frame rate. Based on this usable data was selected.

Gathering of the data was done by both the group members and others by performing the gestures in front of the radar while corresponding labels were set from the computer's keyboard. The data was then captured, reformatted, normalized and shuffled in order to be usable by the machine learning model. Additional data were also synthesized by translating the recorded data.

The machine learning model was constructed by using Tensorflow [24] and Keras [25] in Python by adapting solutions to similar problems. Finally, the predictions of the machine learning model were processed by interpreting a sequence of predictions to determine if a gesture was detected.

### 4.1 Selection of gestures

The type and amount of hand gestures were chosen to be intuitive given the selected use case and discernible by the radar. For every function performed by the end application, a different gesture was required. The decision of which gestures were intuitive was based on the experience and intuition of the project group. Once intuitive gestures were chosen, they were evaluated using the radar. If the radar did not react properly to a gesture it was modified or replaced.

### 4.2 Radar structure

The two following sections declare the method of choosing the radar settings and what parts of radar's data structure is going to be sent to the computer.

#### 4.2.1 Radar measurement capabilities

The first step to configure the radar was to find the wanted values of  $d_{\max}$ ,  $d_{\text{res}}$ ,  $v_{\max}$ , and  $v_{\text{res}}$ . Therefore, TI's demo visualizer software [27] was used, to get a feel for what the selected gestures would need for specifications, and saved data from the radar was analysed.  $d_{\max}$  and  $v_{\max}$  only need a minimum since gestures was preformed at roughly the same distance from the radar with a max speed. The minimum were,  $d_{\max} \geq 2$  m and

$v_{\max} \geq 1$  m/s. Since the project was focusing on intricate gestures the resolutions had to be minimized but all the parameters are somewhat connected, as shown in table 4.1. The needed idle time between chirps and frames were also affected by the parameters since more data need more idle time for calculation and transmission, but trials with the gestures proved that a high velocity resolution was need for gestures to be noticeable. The result from experimentation proved that  $v_{\text{res}}$  needed to be on the 10 mm/s scale. To acquire this the chirp and frame configuration in Figure A.1, were chosen. This resulted in picking  $S = 100$  MHz/ $\mu$ s,  $F_{ADC} = 2 \cdot 10^6$  samples/s,  $N_{\text{samples}} = 64$  samples,  $T_{\text{actual}} = 260$   $\mu$ s,  $N_{\text{chirp}} = 64$  chirps/frame, which gave  $d_{\max} = 3$  m,  $d_{\text{res}} = 4.7$  cm,  $v_{\max} = 1.9$  m/s, and  $v_{\text{res}} = 58$  mm/s. The  $\theta_{\max}$  and  $\theta_{\text{res}}$  were fixed at  $\pm 90^\circ$  and  $1/2 \cos(\theta)$ , respectively.

**Table 4.1:** Shows the change in radar specifications when the radar parameters increase in value.

Spec \ Parameter	$S$	$F_{ADC}$	$N_{\text{samples}}$	$T_{\text{actual}}$	$N_{\text{chirp}}$
$d_{\text{res}}$	Lowers	Increases	Lowers	None	None
$v_{\text{res}}$	None	Lower $T_{\text{actual}}$	Increases $T_{\text{actual}}$	Lower	Lowers
$d_{\max}$	Lowers	Increases	None	None	None
$v_{\max}$	None	Lower $T_{\text{actual}}$	Increases $T_{\text{actual}}$	Lowers	None

### 4.2.2 Data structure

The parts of the data structure that were considered viable for this project were the Object-Detection and the Range-Doppler Heat Map since the object's velocity is an essential part of detecting its movement. Therefore the Range Profile, Noise Profile, Statistics Profile, and the Range Azimuth Profile were rendered as obsolete in this project.

The object list was essentially a list of points in the doppler heat map with high intensities. This meant that the heat map had more information for the MLA to work with, and could thereby have been the better candidate for this project. The problem, however, was that the heat map carried too much information. The radar at a fixed baud rate of 921600 bits per second, which equals 115200 bytes. So a package carrying more information would take a longer time to transmit. So, the transmission time was an essential part of choosing data structure, therefore the sizes were calculated.

To calculate the size of the heat map the `#rangeBins` and `#dopplerBins` had to be calculated,

$$\text{\#rangeBins} = \text{number of ADC samples} = 64, \quad (4.1)$$

$$\text{\#dopplerBins} = \text{\#ChirpsPerFrame} / \text{\#TXantennas} = 128 / 2 = 64. \quad (4.2)$$

The size of the range doppler heat map could then be calculated as

$$\text{size} = \text{\#rangeBins} \cdot \text{\#dopplerBins} \cdot 4 \text{ bytes} = 64 \cdot 64 \cdot 4 = 16384 \text{ bytes}. \quad (4.3)$$

The size of Object Detection is dependent on the number of objects detected by the radar. Therefore, the radar was tested with different hand gestures and the highest observed amount of detected objects by the radar was 15. To calculate the maximum size, the detected objects was set to 20, corresponding to a size of

$$\text{size} = 12 + \text{\#objects} \cdot 12 \text{ bytes} = 12 + 20 \cdot 12 = 252 \text{ bytes}. \quad (4.4)$$

Both structures had a header with 36 bytes in front of them bringing the total byte count up to 16420 and 288. The transmission times of these were  $16416 / 115200 = 143$  ms

and  $284/115200=2.47$  ms, while the frame time (ignoring idle time) was 33,3 ms. The frame rates were thereby 5.67 and 28.0 frames per second. Since gestures could take less than half a second to perform, the detected object list was chosen for its superior frame rate.

### 4.3 Gathering training data

Data was gathered from five individuals not connected to the project, in addition to the project group members. Before recording a gesture class, they got instructed on how to perform the gesture and encourage to repeat it until it felt natural. A recording was then started where approximately 100 gestures were captured for the class and the process was repeated for the other classes. This resulted in a total of approximately 500 recorded gestures for each class from independent sources. The four members of the project group contributed as well with a total of approximately 1000 gestures per class.

When recording, one of the members of the project group labeled the data as background or gesture, trying to time the start and end of the gesture. To speed up the process, the background data from these recordings were not used. Instead, background data was recorded separately and merged with the gesture data in a later step.

### 4.4 Preprocessing of data

To feed the MLA radar data some preprocessing was needed. First, the radar data packets had to be read and stored in a, for both humans and computers, readable fashion so that it could be analyzed and reused. The saved data then had to be formatted in a readable way for the MLA and lastly, there was a need to normalize the data to make the MLA more accurate. All steps were done in python and how they were connected is visualized in Figure 4.1.

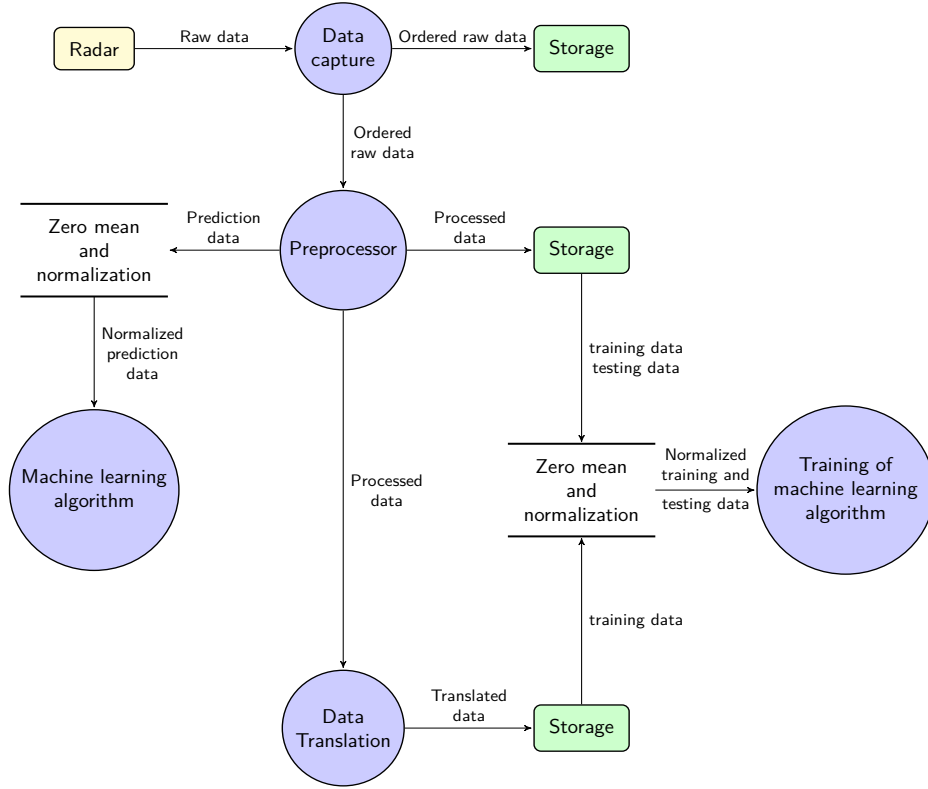
#### 4.4.1 Capturing data

To read the data a modified version of user *ibaiGorordo*'s Github code [28] was used. This gave a python dictionary, divided into features, as an output. The output was stored in a CSV format for analyses and future use.

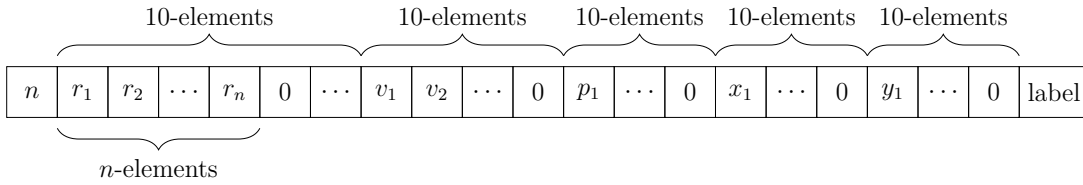
#### 4.4.2 Reformatting data

The MLA needed a standard format to work. Therefore a standardized vector was produced for each frame with a set length. To achieve this, the maximum amount of objects where set. Objects far away were less likely to contain useful information so objects were sorted by the range and if the maximum number was exceeded, the ones furthest from the radar were cut.

The limit was set to 10 objects since this was an observed limit rarely exceeded by the gesture data. If less than 10 objects were found, each feature vector (range index, Doppler index, peak value, x value, y value) was padded with zeros until the length was 10. Finally, the data was structure as seen in Figure 4.2. The label was optional depending on if the data was supposed to be used as training/testing data or prediction data for the application, resulting in a vector of size 51 or 52 elements.



**Figure 4.1:** A flowchart demonstrating how data flowed between different programs, storages, and algorithms.



**Figure 4.2:** The format of a standard vector. First comes the amount of objects, then the objects' ranges from smallest to largest, the Doppler, peak x, and y. Values with the same index correspond to the same object.

Since gestures only were performed within a certain range of the radar, a maximum distance was set to filter out noise. The cut of length was chosen as the 40th range index which corresponds to 1,87 m after which any objects are cut, regardless of the number of objects in the frame.

#### 4.4.3 Normalizing data

The final manipulation of the radar data was to normalize it to values between one and negative one and centering it by making sure the mean value of the data was zero. This was done right before the data was fed to the network. All data was loaded into a long list and separated into different features. The mean was then calculated for each feature and all elements which were not padding were subtracted with the mean value. Then the absolute maximum of each feature was calculated and divided with which resulted in data centered around zero with maximums of  $\pm 1$ . The means and maximums were saved so they could be used when predicting new frames with that model.

#### 4.4.4 Shuffling data

Since the data was recorded by performing the same gesture multiple times while the application should be able to detect any sequence of gestures shuffling of the input data was required. This was done by grouping each continuous sequence with the same label as an individual gesture. These groups were then selected in a random order to be put in a sequence. Between each gesture a varying amount of background frames were inserted to separate them, forming a single sequence of varying gestures with background in between.

### 4.5 Synthesized data

Synthesized data was created in order to increase the amount of data in a more efficient way than further data recording. The synthesized data was only used as training data since it would not give meaningful results as testing data.

The data were synthesized through translation. This was done simply by changing the x and y coordinates of an object and then calculating the new range value. The same translations factors dx and dy were used for all frames from one recording.

### 4.6 Machine learning(ML)

The design of the machine learning structure was primarily done by finding and adapting solutions to similar problems. Of particular note is the Soli project at Google [2]. Using the sequential model of Keras in Python the model was built by adding layers together in sequence. These layers were implemented by running Tensorflow behind the Keras API.

Optimization of hyperparameters was done by comprehensive testing of different combinations and iterative improvements of the more successful combinations.

### 4.7 Applying the results

To interpret the output of the MLA some post-processing was needed. Based in statistics, a sequence of  $L_s$  frames was defined as a gesture of a specific class if more than a given limit,  $N_{\text{lim}}$ , of the frames were classified as a said class of gesture. The detection of a gesture was further defined as  $N_d$  such sequences, classified as the same gesture class, within  $L_D$  frames of each other. After detection, the number of times a sequence had been classified as a certain gesture was reset for all gestures classes. See Appendix B for the statistical calculations.

A good starting point was assessed to be  $L_s = 8$ ,  $N_{\text{lim}} = 4$ ,  $L_D = 9$ , and  $N_d = 2$ . This assessment was, however, done on approximations, so some testing was done to find potentially better values. The end result was  $L_s = 8$ ,  $N_{\text{lim}} = 3$ ,  $L_D = 9$ , and  $N_d = 5$ . This gave a good balance between the ability to suppress false detection and still minimizing the risk of missing a gesture detection.

Resetting values after a detection creates a delay of  $L_D$  frames before a new frame can be detected. This is not a bad thing though since gestures and background had been recorded separately. This makes the network unfamiliar with the transitions between background and gestures. There was, in fact, a need for an extra delay of 10 frames, making the total delay 19 frames, to suppress false classifications.

When a gesture was detected, the program used virtual keys on the computer to control the music. This allowed the program to change the volume, switch songs, and pause the music. If the computer was connected to another device such as a Chromecast it would control the device.

# 5

## Results

The final result is an application able to control a media device by using hand gestures. The accuracy of the gesture detection ranges between 48.1 and 78.5 percent per gesture using continuous data.

Six hand gestures were chosen, and the model was trained on almost 100000 frames of gesture data representing 8686 complete gestures. The machine learning model was build using LSTM layers and reached an average accuracy of 81.6 percent and 85.7 percent for three and six gestures respectively for recorded shuffled data.

### 5.1 Application

The final application uses virtual key presses to control media playback and volume on a PC. Repeat activations of certain buttons during longer gestures are disallowed by requiring detection of background between each subsequent activation of the same function.

One gesture is set up to be used to switch between different units, for example, controlling a lamp instead of a media device, but no further uses have been implemented.

### 5.2 Hand gestures

All hand gestures are made with the right hand.

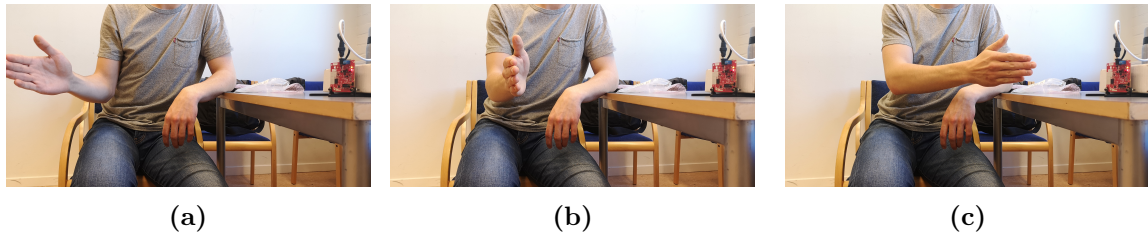
- **Button:** Having the fingers slightly cupped. The action is pressing the thumb down and lifting it back up again. Similar to holding a remote control and pressing a button. The gesture used to stop/start the music.



**Figure 5.1:** Visualization of the button gesture. The sequence goes from left to right.

- **Swipe next:** Keeping the fingers stretched but together. The action is to move the hand and arm from right to left, similar to a slap. The gesture used to go to the next song.





**Figure 5.2:** Visualization of the swipe next gesture. The sequence goes from left to right.

- **Swipe prev:** The reverse of “Swipe next”. The same position of the fingers but the action is to move them from left to right. The gesture used to go to the previous song.



**Figure 5.3:** Visualization of the swipe prev gesture. The sequence goes from left to right.

- **Slide up:** Holding the fingers similar to holding a pole where the tip of the thumb touches the tip of the index finger. The actions are to move the thumb down the index finger while simultaneously extending the rest of the fingers. The end position is to have the fingers stretched similarly to the “Swipe” gestures. The gesture used to turn volume up.



**Figure 5.4:** Visualization of the slide up gesture. The sequence goes from left to right.

- **Slide down:** The reverse of “Slide up”. Starting instead from the end position then sliding the thumb along the index finger while slowly cupping the fingers. The end position is the same as the starting position as “Slide up”. The gesture used to turn the volume down.



**Figure 5.5:** Visualization of the slide down gesture. The sequence goes from left to right.

- **Flop:** The starting position is having an open fist while the palm should be aimed towards the sky. The action is lifting the hand slightly, turning it counter-clockwise

and putting it down again. The movement is similar to turning a page in a book or flipping a card over. The gesture used to change the unit.



**Figure 5.6:** Visualization of the flop gesture. The sequence goes from left to right.

## 5.3 Radar

From the data packet Statistics, Profile statistics from the radar were obtained. The relevant values are as follow the inter-frame processing time at  $1915 \mu s$ , transmit output time at  $3143 \mu s$ , inter-frame processing margin at  $9098 \mu s$ , inter-chirp processing margin at  $2017 \mu s$ .

The calculated frame time was 36 ms, not taking in to account the radar's internal calculation time and other arbitrary processes. Through testing, the radar could function properly at a frame time of 43 ms. The real frame time, calculated through the time command i python, was 60 ms. Through extensive testing to lower the frame time the conclusion could be made that the frame time had a lower limitation of 60 ms.

The calculated range and velocity resolutions are

$$d_{\text{res}} = 4.7 \text{ cm}, \quad (5.1)$$

$$v_{\text{res}} = 58 \text{ mm/s}, \quad (5.2)$$

and the maximum values are

$$d_{\text{max}} = 3.0 \text{ m}, \quad (5.3)$$

$$v_{\text{max}} = 1.9 \text{ m/s}. \quad (5.4)$$

## 5.4 Data processing

In total, the model was trained on 251 020 frames of data including background frames. After being shuffled according to Section 4.4.4, 70 percent of the frames were used for training and the rest for verification. The distribution of gestures and frames per class of gesture can be seen in table 5.1.

**Table 5.1:** Number of frames and gestures used in the training and verification of the model.

	$N_f$ training	$N_f$ testing	$N_f$ total	$N_{\text{gestures}}$
Slide up	16301	6696	22997	1725
Slide down	11350	5042	16392	1499
Button	14848	6080	20928	1528
Swipe next	12617	5555	18172	1709
Swipe prev	9468	4320	13588	1365
Flop	5456	2311	7767	860
Total	70040	30004	99844	8686

Normalizing the data provided a large improvement of the result. With identical hyperparameters, the model reached 83.1 percent accuracy on the data before normalization and 88.5 percent afterward. These results are on different data than the final results but the increase of 5.4 percentage clearly indicates that normalized data should be used.

The artificial data created was mostly unused in the final training. Results did not improve significantly when artificial gestures were used in training. Some artificial background data were used to separate gestures.

## 5.5 Machine learning

The final model is built by three LSTM layers with a dropout layer between the first and the second and a final, dense, softmax layer to translate the results into a probability distribution.

Two optimizations were done, one for three gestures and one for six. Both setups include the possibility to classify gestures as background. After testing, the configurations of hyperparameters shown in table 5.2 produced the best result.

**Table 5.2:** Hyperparameters used in the training of the models.

Number of gestures	3	6
Epochs	500	500
Time-step	10	10
Batch size	10	10
Lstm_output	10	20
Stateful	True	True
Optimizer	Adam	Adam
Learning rate	0.00001	0.00025
Decay	0,000001	0,0000025
Accuracy	0.816	0.857
Loss	0.456	0.539

Table 5.2 shows the overall accuracy for six gestures of 0.857 and for three gestures of 0.816. It's important to note that this does not guarantee that level of accuracy for every class of gesture and that every class of gesture needs to be classified accurately to ensure good performance. The distribution of accuracy between gestures can be seen in table 5.3 and table 5.4.

**Table 5.3:** The confusion matrix when training for three gestures and background.

Predicted \ Actual	Button	Swipe Next	Swipe Prev	Background
Button	70.1	3.0	2.6	24.4
Swipe Next	8.1	69.6	4.7	17.7
Swipe Prev	9.8	9.1	62.3	18.8
Background	3.6	2.8	1.5	92.2

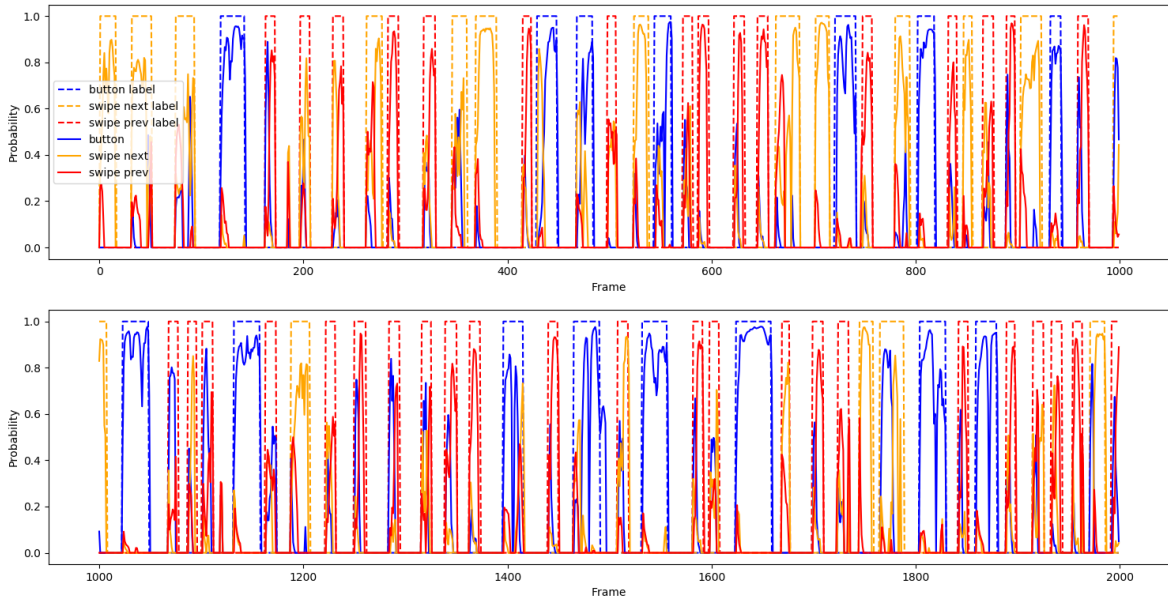
**Table 5.4:** The confusion matrix when training for six gestures and background.

Predicted \ Actual	Slide Up	Slide Down	Button	Swipe Next	Swipe Prev	Flop	Background
Slide Up	73.6	6.1	7.8	0.4	0.6	0.5	11.1
Slide Down	9.8	65.0	8.2	0.8	1.0	1.3	14.0
Button	14.4	5.7	66.8	0.3	1.3	0.7	10.8
Swipe Next	1.2	1.0	0.8	79.8	1.5	2.9	12.7
Swipe Prev	1.1	0.8	2.8	2.5	77.3	1.1	14.4
Flop	1.8	5.2	3.0	10.4	2.0	62.5	15.0
Background	1.2	0.9	0.9	0.8	0.7	0.2	95.3

## 5.6 Post-processing

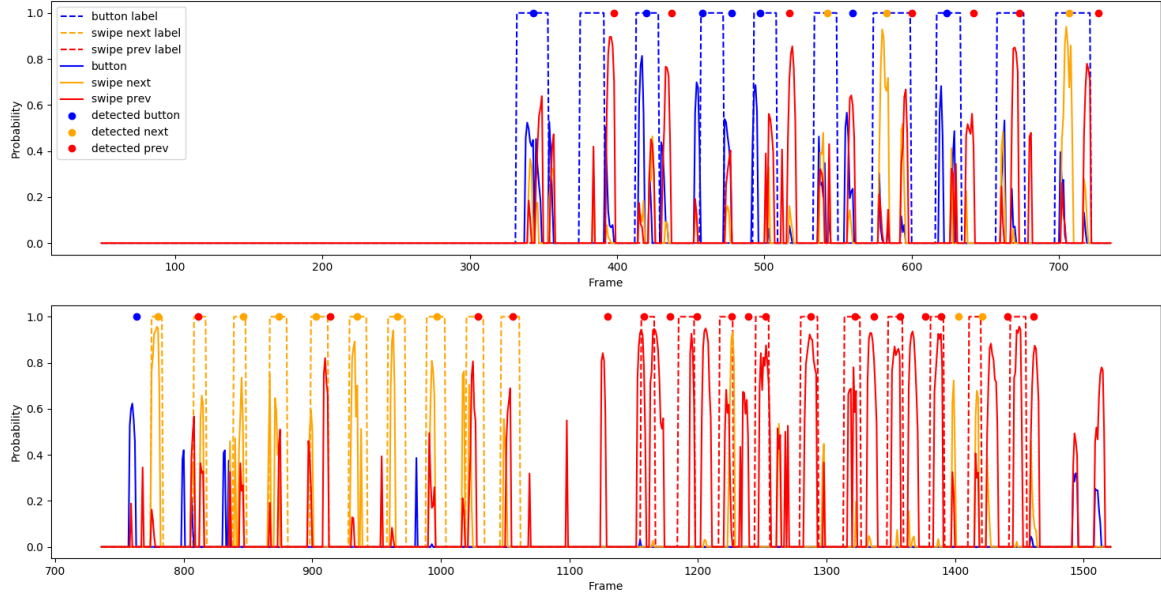
To make classifications of real-time data the trained model makes predictions based on the ten latest frames every time a new frame is captured. The frequency of new frames and therefore the frequency of predictions is approximately 16.6 frames per second.

The total result for predicting sequences of recorded gestures varies depending on the input data. When the input data is discontinuous, like the training data, the network will predict with high certainty and accuracy as shown in Figure 5.7. However, when the network is fed continuous data, the certainty and the accuracy of the predictions goes down.

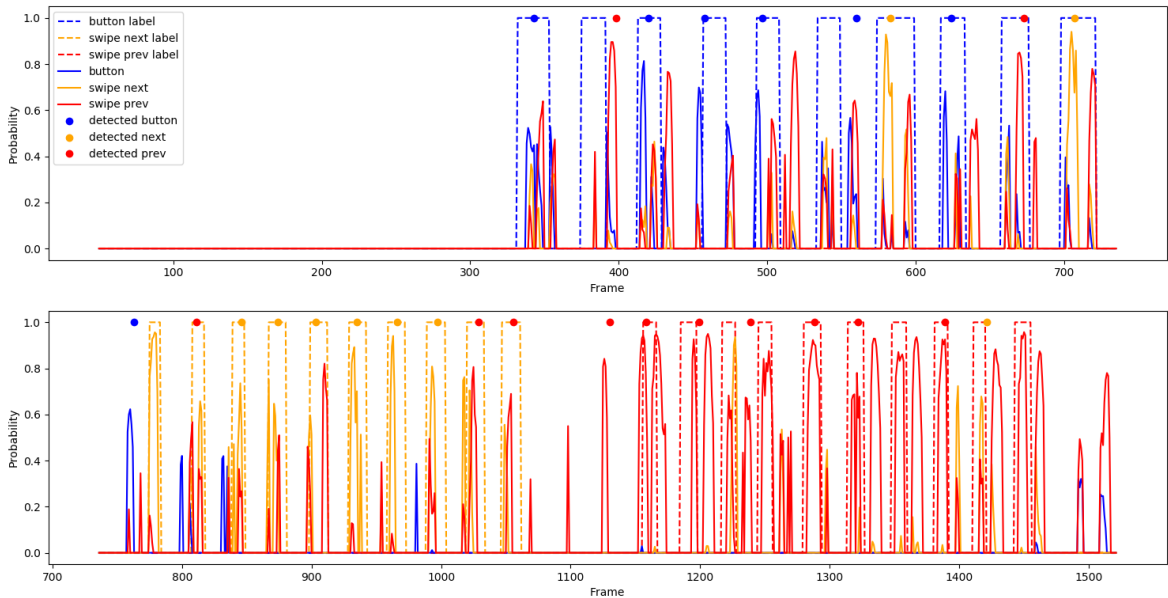
**Figure 5.7:** The probabilities for each gesture plotted against time, represented as frames. The probability is set to zero when it is lower than the background probability. Otherwise, the full probability is shown.

Besides the uncertainties, the network was quite sensitive to the positioning of the user and the radar, and because of the poor results of the six-gestures networks, there is only going to be results for the application using a three-gesture network and with the user and radar in good positions. These results are presented in Figures 5.8 to 5.10 but more are available in Appendix in Figures D.1 to D.3. Figure 5.8 and 5.9 shows the difference

between not having any delay before being able to detect a new gesture compared to the delay of ten frames which is used. Important to notice about these two sequences is that swipe previous spikes detected in between the swipe next and swipe previous sections where only products of miscommunication.

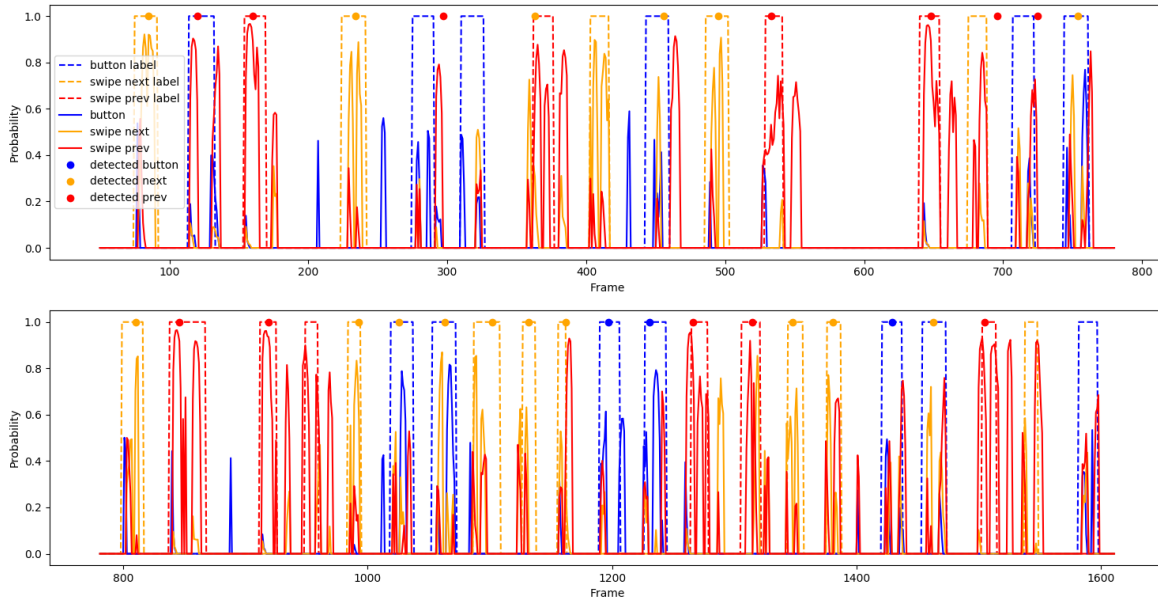


**Figure 5.8:** The figure presents the probabilities for each gesture over time where each step is a frame, where there is no delay between detections. The probabilities are only shown when they are greater than the probability of background. The dots represent a detection.



**Figure 5.9:** The figure presents the probabilities for each gesture over time where each step is a frame, where there is a delay of 10 frames between detections. The probabilities are only shown when they are greater than the probability of background. The dots represent a detection.





**Figure 5.10:** The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order.

The amount of prediction and the likelihood of a certain output given the input are shown in tables 5.5 (to see the amount of each gesture, go to Figur D.1 in Appendix). The tables show that the swipe next gesture had the highest probability of beginning predicted correctly  $\sim 78,5\%$ . Second, came the swipe previous gesture with  $\sim 63,8\%$  and worst was the button gesture which only had a probability of  $\sim 48,1\%$ . Background predicted as gestures was never a problem on its own, but close to gestures were some false detections produced by mislabeling, moving the hand to the starting position, or back from the end position. Furthermore, there were a few gestures which got more than one detection. This is presented in tables 5.6, where detections are coupled with which gesture it was found close to (The amounts are shown table D.2 in Appendix).

**Table 5.5:** The confusion matrix for the application where the likelihood of getting a certain output given an input is shown.

Input \ Output	Next	Button	Previous	Background
Next	78,5	1,3	12,7	7,6
Button	15,6	48,1	9,1	27,2
Previous	20	0	63,8	16,3

**Table 5.6:** The confusion matrix for the application where the likelihood of getting a double prediction (i.e a second prediction on an already predicted input) is shown given a certain output and input.

Input \ Output	Next	Button	Previous	Background
Next	2,53	0	0	0
Button	7,79	1,30	1,30	0
Previous	6,25	0	0	0



# 6

## Discussion

The final result is significantly lower than the 95 percent accuracy per gesture set up in the aim for the project, with the best per gesture result with continuous data being 78.5 percent and the worst being 41.1 percent. Looking at the accuracy per frame with segmented data the results are slightly better with the best per gesture result per frame being 79.8 percent and the worst being 62.3 percent in the machine learning validation.

The result is mostly dependent on three factors; the tuning of the hyperparameters, the machine learning model and the input data. Further sections will discuss how each of those could be improved.

### 6.1 Tuning

The tuning of the hyperparameters was done in a comprehensive although slightly unstructured way. Since most iterations improved on previously successful settings it is possible that some very different combinations of values would provide a better result. However, some iterations included high variations of variables and the number of iterations still range a large section of possible combinations. If a better setting exists it is still unlikely that the improvements are very significant.

### 6.2 Machine Learning structure

The choice of using Tensorflow as the framework for the machine learning was made partly because of its widespread usage resulting in a wide amount of information and tutorials available. The ecosystem of API's and connected frameworks, most importantly Keras, also influenced the decision. Tensorflow with Keras has been easy and fast to use but no comparisons to other frameworks have been made so it is impossible to determine if better results would have been reached with another choice.

The choice of using LSTM layers in the model is supported by the theory. Since the classification of one frame to a certain class of gestures is highly correlated with the classification of nearby frames. This is also supported by the early testing of different combinations of layers.

However, not much testing of different models was done later in the project. Neither types of layers or amount of layers were varied after the tuning of hyperparameters had begun. It is also possible that the use of Keras to build the model resulted in an overly simplified model which was not specialized enough to this problem.

Some possible modifications of the model are likely to improve the result.



## 6.3 Data

The quality of the data is dependent on a few factors. The performance of the gestures when recording, the accuracy of the radar to capture this data, what data the radar is able to output and the processing of the data before it is used in the machine learning model.

### 6.3.1 Radar

A big problem with the radar is the deviating measured frame time. It is, roughly 15 ms slower compared to the frame time of the radar. After doing several tests to find a bottleneck, no results were found. The time used to process the received data was below 2 ms so the code was not the factor. After consolidation with Texas Instrument employees, the hardware of the computer receiving the data was believed to be the reason behind the long frame time. However, after testing the possible underlying issues we could not pinpoint which part of the computer was responsible. All that is clear is that some part of the computer and radar communication was not working properly which must have caused a delay in the process.

Another problem noticed was that the Object Detection list used as output format from the radar might have carried too little information for the neural network. There were a lot of similar or identical frames which left less room for nuances. To combat this, the heat map data structure could be used. However, the AWR1642 evaluation board is not made for extracting a huge amount of data from it, considering that the only connection port is a micro-USB. To achieve a higher frame rate with the heat map, TI has developed a real-time data-capture adapter. The board is called DCA1000 and would increase the transmission speed to 1 GB/s. This is an option that should be considered for a future project with the AWR1642 radar.

### 6.3.2 Data gathering

When training neural network there was need of a lot of recorded data, we were recommended to have at least 1000 recordings of each gesture. Because of the short time span of the project, and the considerable time needed to train each machine learning model, the gestures were recorded separately from the background leading to a discontinuous data sequence. This led to the MLA having trouble predicting frames right before and after gestures. If the gestures would have been recorded continuously with proper background in between each gesture the MLA might even have been better at predicting when a gesture was on its way.

The problem with this is that it would be hard to shuffle data. Either you would have to record thousands of snippets with pre-gesture background, gesture, post-gesture background, stop and shuffle these snippets or the data would have to be recorded with gestures coming in random order. The first method would take way too much time for this bachelor's project. The second, however, could be viable, but it would be confusing for the people being recorded and the person labeling the data.

### 6.3.3 Data Processing

The data processing worked quite well for the project. The number of objects allowed in the standard vector was never changed or tested, neither was the cut-off length. This was since the values were determined by observations. The cut of length was also a decided maximum for how far from the radar a gesture would be performed and a change in the number of objects would increase the processing time.

The zero meaning and normalization of the data proved very effective to improve the result. Before using these methods, the highest probability achieved for three gestures was 75 percent and after, the result was sometimes closer to 95 percent. The choice of processing the data feature by feature instead of the whole standard vector was done to equalize the importance of each feature.

## 6.4 Application

Using virtual key presses to let the application control media setting means the app is highly dependent on the operating system. This creates some limitations. To begin with, the operating system determines what the result of each press is, for example how much one press of volume up increases the volume, and most importantly the app is only tested for one operating system and will most likely not work correctly on another one.

## 6.5 Hand gestures

As seen in the result there is a big discrepancy between the highest and the lowest accuracy of recognized hand gestures with continuous data. One likely explanation for this is that some of our hand gestures produced better data through the radar. With another selection of gestures better results might have been reached, this is, however, likely to result in the gestures being less intuitive.

## 6.6 Conclusions

In conclusion, this thesis explored using fine gestures in machine learning and reached a starting point from which further work can be made. Different machine learning algorithms have been explored and from this, a functional neural network model has been designed with three LSTM-layers and one dropout layer. After exhaustive tuning of the hyperparameters, this model provided adequate results. The object-detection data from the radar has been evaluated and methods for processing this data have been collected. Using this type of data allowed higher frame-rate but some information was lost. Finally, a method for post-processing the output in order to improve stability has been calculated.

The overall result was less than hoped for but it is not likely that better could have been achieved with the limited experience and time available. The accuracy was also less than the thesis this one is based on. The most significant difference is that this thesis uses a different machine learning algorithm which might perform worse but allows solving far more difficult problems. One significant difference in the problem is that the LSTM network allows any gesture to be classified after training, with the limitation being if the radar can get discernible data between gestures, as opposed to the limit of only linear gestures previous. The other major difference is that the application allows for continuous

classification while the application in the previous thesis required the user to signify when a gesture was started.

This thesis provides an improvement in usability from the previous project but some accuracy and reliability have been lost. With the knowledge acquired through the project, several areas of improvement to remedy this have been identified.

### 6.6.1 Future works

First, data gathering could be improved. One possible improvement is to try and get continuous data. Issues and benefits with this have been discussed but one possible improvement could be to implement some system of automatic labeling which could avoid the human error when capturing varying gestures in sequence. More data processing should be done early to avoid training and tuning the model for less than optimal data.

Less time could be spent tuning hyperparameters and instead spent investigating different machine learning models or working with the data.

Finally, it should be considered what data from radar to use. The heatmap is likely preferable if transfer speed can be improved or the application made to work with less resolution or lower frame rate.

# Bibliography

- [1] M. Mohammed, M. B. Khan, and E. B. M. Bashier, *Machine learning: algorithms and applications*. Boca Raton : CRC Press, 2017., 2017. [Online]. Available: <https://www.taylorfrancis.com/books/9781315371658> (visited on 05/09/2019).
- [2] J. Lien, N. Gillian, M. Karagozler, P. Amihoud, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, “Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar”, *ACM Transactions on Graphics (TOG)*, vol. 35, no. 142, p. 19, 2016. [Online]. Available: [http://www.ivanpoupyrev.com/wp-content/uploads/2017/01/siggraph\\_final.pdf](http://www.ivanpoupyrev.com/wp-content/uploads/2017/01/siggraph_final.pdf) (visited on 05/10/2019).
- [3] M. van Gerven and S. Bohte, *Artificial Neural Networks as Models of Neural Information Processing*, ser. Frontiers Journal. Frontiers in Computational Neuroscience, 2018, ISBN: 978-2-88945-401-3.
- [4] C. Berkus, M. Buck, J. Gustafsson, and M. Kauppinen, “Human control of mobile robots using hand gestures”, 2018. [Online]. Available: <http://publications.lib.chalmers.se/records/fulltext/255856/255856.pdf> (visited on 05/10/2019).
- [5] G. Dong and H. Liu, *Feature Engineering for Machine Learning and Data Analytics*. CRC Press, Boca Raton, 2018.
- [6] S. Rao, “Intro to mmwave sensing : Fmcw radars”, 2017. [Online]. Available: [https://training.ti.com/sites/default/files/docs/mmwaveSensing-FMCW-offlineviewing\\_0.pdf](https://training.ti.com/sites/default/files/docs/mmwaveSensing-FMCW-offlineviewing_0.pdf) (visited on 05/10/2019).
- [7] C. Iovescu and S. Rao, “The fundamentals of millimeter wave sensors”, 2017. [Online]. Available: <http://www.ti.com/lit/wp/spyy005/spyy005.pdf> (visited on 05/10/2019).
- [8] J. Brownlee. (2016). Supervised and unsupervised machine learning algorithms, [Online]. Available: <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/> (visited on 05/15/2019).
- [9] R. Roy. (2019). ML | stochastic gradient descent (sgd), [Online]. Available: <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/> (visited on 05/16/2019).
- [10] T. Srivastava, “How does artificial neural network (ann) algorithm work? simplified!”, *Analytics Vidhya*, Oct. 2014. [Online]. Available: <https://www.analyticsvidhya.com/blog/2014/10/ann-work-simplified/> (visited on 05/10/2019).
- [11] H. Marshall, “Deep learning”, Mar. 2019. [Online]. Available: <https://www.investopedia.com/terms/d/deep-learning.asp> (visited on 05/10/2019).
- [12] C. Olah, “Understanding lstm networks”, 201. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (visited on 05/09/2019).

- [13] P. de Boer, D. Kroese, and S. Mannor, *A Tutorial on the Cross-Entropy Method*. Kluwer Academic Publishers, 2003. DOI: <https://link-springer-com.proxy.lib.chalmers.se/article/10.1007%2Fs10479-005-5724-z>.
- [14] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimisation”, *Published as a conference paper at ICLR 2015*, 2015. [Online]. Available: <https://arxiv.org/pdf/1412.6980.pdf> (visited on 05/10/2019).
- [15] J. J. Schneider and S. Kirkpatrick. Springer, Berlin, Heidelberg, 2006. [Online]. Available: <https://link-springer-com.proxy.lib.chalmers.se/book/10.1007%2F978-3-540-34560-2%C2%B4> (visited on 05/10/2019).
- [16] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*. 2011, vol. 12, pp. 2121–2159.
- [17] T. Tieleman and G. Hinton, *COURSERA: Neural Networks for Machine Learning*. 2012.
- [18] Texas Instruments. (2018). Awr1642 single-chip 77- and 79-ghz fmcw radar sensor, [Online]. Available: <http://www.ti.com/lit/ds/swrs203a/swrs203a.pdf> (visited on 05/10/2019).
- [19] T. Instruments. (2018). Awr1642 evaluation module (awr1642boost) single-chip mmwave sensing solution user guide, [Online]. Available: <http://www.ti.com/lit/ug/swru508b/swru508b.pdf> (visited on 05/10/2019).
- [20] “[SDK\_directory]/mmwave\_sdk\_02\_01\_00\_04/packages/ti/demo/xwr16xx/mmwave/docs/doxygen/html/index.html,” in MMWAVE-SDK, Version: 2.01.00.04, 2018, [Online] Available: <http://www.ti.com/tool/MMWAVE-SDK>, (visited on: 2019-02-01.)
- [21] Texas Instruments, “Mmw demo data structure v0.1”, [Online]. Available: [https://e2e.ti.com/cfs-file/\\_\\_key/communityserver-discussions-components-files/1023/mmwave-Demo-Data-Structure\\_5F00\\_8\\_5F00\\_16.pdf](https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/1023/mmwave-Demo-Data-Structure_5F00_8_5F00_16.pdf) (visited on 05/10/2019).
- [22] Lenovo, *Ideapad720*, [https://static.lenovo.com/shop/emea/content/pdf/Consumer%20notebook/2017/EMEA\\_IP720S\\_13IKB\\_14IKB\\_DS\\_EN.pdf](https://static.lenovo.com/shop/emea/content/pdf/Consumer%20notebook/2017/EMEA_IP720S_13IKB_14IKB_DS_EN.pdf), 2017.
- [23] Python, “What is python? executive summary”, [Online]. Available: <https://www.python.org/doc/essays/blurb/> (visited on 05/10/2019).
- [24] Tensorflow, “Tensorflow guide”, 2019. [Online]. Available: <https://www.tensorflow.org/guide> (visited on 05/10/2019).
- [25] Keras contributors, “Keras documentation”, [Online]. Available: <https://keras.io/> (visited on 05/10/2019).
- [26] “Integrated development environment (IDE)”, [Online]. Available: <https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment> (visited on 05/06/2019).
- [27] Texas Instruments, *Mmwave demo visualizer*, 2018. [Online]. Available: [https://dev.ti.com/gallery/view/mmwave/mmWave\\_Demo\\_Visualizer/ver/3.1.0/](https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/3.1.0/) (visited on 05/10/2019).
- [28] ibaiGorordo. (). Readdata\_awr1642, [Online]. Available: <https://github.com/ibaiGorordo/AWR1642-Read-Data-Python-> (visited on 05/10/2019).

- [29] Texas Instruments, “Mmwave sdk user guide”, 2019. [Online]. Available: [https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwie2IfQoYTiAhVF\\_KQKHXLnCSUQFjAAegQIABAC&url=https%3A%2F%2Fti.com%2Ffiles-file%2F\\_key%2Fcommunityserver-discussions-components-files%2F1023%2F4034.mmwave\\_5F00\\_sdk\\_5F00\\_user\\_5F00\\_guide.pdf&usg=A0vVawOwKdEnX0cloBwPgamVQAoG](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwie2IfQoYTiAhVF_KQKHXLnCSUQFjAAegQIABAC&url=https%3A%2F%2Fti.com%2Ffiles-file%2F_key%2Fcommunityserver-discussions-components-files%2F1023%2F4034.mmwave_5F00_sdk_5F00_user_5F00_guide.pdf&usg=A0vVawOwKdEnX0cloBwPgamVQAoG).
- [30] R. A. Wienclaw, “Normal and binomial distributions”, *Salem Press Encyclopedia*, 2017. [Online]. Available: <http://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=89163889&site=eds-live&scope=site> (visited on 05/10/2019).
- [31] F. J. Anscombe, “Sampling theory of the negative binomial and logarithmic series distributions”, *Biometrika*, vol. 37, no. 3–4, pp. 358–382, 1950. [Online]. Available: <https://www.jstor.org/stable/2332388> (visited on 05/10/2019).
- [32] S. Glen, “Geometric distribution: Definition & example”, 2015. [Online]. Available: <https://www.statisticshowto.datasciencecentral.com/geometric-distribution/> (visited on 05/10/2019).



# A

## Radar configurations

### A.1 Radar configuration file

The AWR1642 mmWave radar is configured by sending the radar a configure file (.cfg). Figure A.1 shows the configure file for this specific project.

```
sensorStop
flushCfg
dfeDataOutputMode 1
channelCfg 15 3 0
adcCfg 2 1
adcbufCfg -1 0 0 1 0
profileCfg 0 77 220 7 40 0 0 100 1 64 2000 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
chirpCfg 1 1 0 0 0 0 0 2
frameCfg 0 1 64 0 42 1 0
lowPower 0 1
guiMonitor -1 1 0 0 0 0 1
cfarCfg -1 0 0 8 4 4 0 5120
cfarCfg -1 1 0 8 4 4 0 5120
peakGrouping -1 1 1 1 1 63
multiObjBeamForming -1 1 0.5
clutterRemoval -1 0
calibDcRangeSig -1 0 -5 8 256
extendedMaxVelocity -1 0
bpmCfg -1 0 0 1
lvdsStreamCfg -1 0 0 0
nearFieldCfg -1 0 0 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
CQRxSatMonitor 0 3 4 99 0
CQSigImgMonitor 0 31 4
analogMonitor 1 1
sensorStart
```

**Figure A.1:** Configure file (.cfg) used to configure the AW1642 mmWave radar.



## A.2 TI's mmWave radar commands

Table A.1 explains the mmWave API commands and parameters used in the configure file (see Figure A.1) for the radar. See [29] for a more comprehensive explanation.

**Table A.1:** List of mmWave API commands used to configure the AWR1642 mmWave radar [29].

Configuration command	Command details	Command Parameters
sensorStop	Stops the mmWave Front End and the processing chain.	-
flushCfg	Flushes the old configuration. Should be located after sensorStop.	-
dfeDataOutputMode	Sets the transmitting mode for the radar.	<mode Type>
channelCfg	Receives antenna mask from the radar.	<rxChannelEn>, <txChannelEn>, <cascading>
adcCfg	Sets the number of ADC bits and the output format.	<numADCBits>, <adcOutputFmt>
adcbufCfg	adcBuf hardware config.	<dont_care>, <adcOutputFmt>, <SampleSwap>, <ChanInterleave>, <Chirp Threshold>
profileCfg	See section A.2.1.	<profileId>, <startFreq>, <idleTime>, <adcStartTime>, <rampEndTime>, <txOutPower>, <txPhaseShifter>, <freqSlopeConst>, <txStartTime>, <numAdcSamples>, <digOutSampleRate>, <hpfCornerFreq1>, <hpfCornerFreq2>, <rxGain>

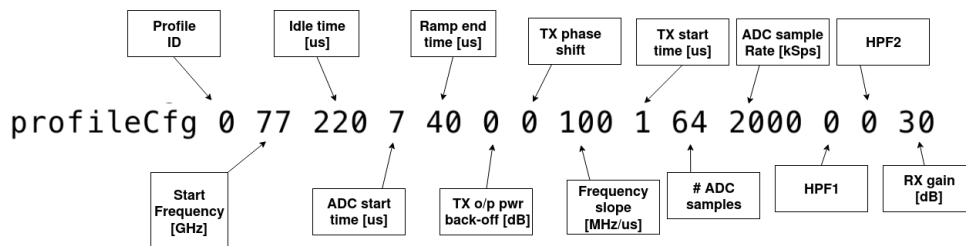
chirpCfg	See section A.2.2.	<chirpStartIdx>, <chirpEndIdx>, <profileId>, <startFreqVar>, <freqSlopeVar>, <idleTimeVar>, <ADCstartTimeVar>, <txEnMask>
frameCfg	See section A.2.3.	<chirpStartIdx>, <chirpEndIdx>, <numOfLoops>, <numOfFrames>, <framePeriod>, <trigSelect>, <FrameTrigDelay>
lowPower	Low Power mode config message.	<don't_care> <ADCMode>
guiMonitor	Enables export of different data gathered by the radar.	<subFrameIdx>, <detected objects>, <log_magnitude_range>, <noise profile>, <rangeAzimuthHeatMap>, <rangeDopplerHeatMap>, <statsInfo>
cfarCfg	CFAR config message to datapath.	<subFrameIdx>, <procDirection>, <mode>, <noiseWin>, <guardLen>, <divShift>, <dont_care>, <ThresholdScale>
peakGrouping	Enables the reporting of a cluster of detected neighboring points as only one point, the highest one, this reducing the total number of detected points per frame.	<scheme>,  <PGinRangeDir>, <PGinDopplerDir>, <EndRangeIdx>

multiObjBeamForming	Allows radar to separate reflections from multiple objects originating from the same range/Doppler detection.	<subFrameIdx>,  <Feature Enabled>, <threshold>
clutterRemoval	Static clutter removal algorithm implemented by subtracting from the samples the mean value of the input samples to the 2D-FFT.	<enabled>
calibDcRangeSig	When enabled the antenna coupling signature is estimated during the first N chirps, and then it is subtracted during the subsequent chirps.	<subFrameIdx>,  <enabled>, <negativeBinIdx>, <positiveBinIdx>, <numAvg>
extendedMaxVelocity	Extends max velocity for object detection to $2 \cdot v_{\max}$ .	<subFrameIdx>,  <enabled>
bpmCfg	Enables configuration to TDM-MIMO scheme and provides SNR improvement by running 2Tx simultaneously.	<subFrameIdx>,  <enabled>, <chirp0Idx>, <chirp1Idx>
lvdsStreamCfg	Enables the streaming of various data streams over LVDS lanes.	<subFrameIdx>,  <enableHeader>, <dataFmt>, <enableSW>
nearFieldCfg	Enables near field correction algorithm.	<subFrameIdx>,  <enabled>, <startRangeIndex>, <endRangeIndex>
compRangeBias- AndRxChanPhase	Command for datapath to compensate for bias in the range estimation and receive channel gain and phase imperfections.	<rangeBias>,  <Re(A,B)>, <Im(A,B)>

measureRangeBias-AndRxChanPhase	Command for datapath to enable the measurement of the range bias and receive channel gain and phase imperfections.	<enabled>,  <targetDistance>, <searchWin>
CQRxSatMonitor	Rx Saturation Monitoring config message for Chirp quality.	<profile>, <satMonSel>, <priSliceDuration>, <numSlices>, <rxChanMask>
CQSigImgMonitor	Signal and image band energy Monitoring config message for Chirp quality.	<profile>,  <numSlices>, <numSamplePerSlice>
analogMonitor	Enables/Disables monitor features.	<rxSaturation>, <sigImgBand>
sensorStart	Starts the sensor. This function triggers the transmission of the frames as per the frame and chirp configuration. Starts the mmWave Front End and the processing chain.	-

### A.2.1 chirp profile configuration

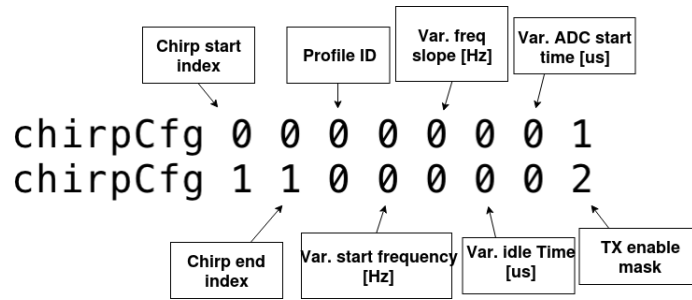
The mmWave SDK allows for up to four different chirp profiles, which every chirp is based on. The commands variables are explained in Figure A.2.



**Figure A.2:** Parameter explanation for mmWave API command "profileCfg" used in the configure file for the AWR1642 radar in the project. Made in draw.io.

### A.2.2 Chirp configuration

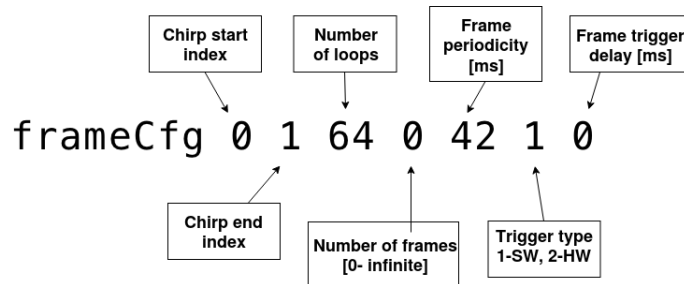
Every chirp is based on a chirp profile and the chirp configuration command allows for variations to those variables. The mmWave SDK allows for 512 individual chirps in each chirp profile. An explanation for the variables in the command is shown in Figure A.3.



**Figure A.3:** Parameter explanation for mmWave API command “chirpCfg” used in the configure file for the AWR1642 radar in the project. Made in draw.io.

### A.2.3 Frame configuration

The frame configuration command sets the frame parameters. The number of loops sets the number of times the chirp sequence is run until it is considered a whole frame. See Figure A.4 for the command parameters explanations.



**Figure A.4:** Parameter explanation for mmWave API command “frameCfg” used in the configure file for the AWR1642 radar in the project. Made in draw.io.

# B

## Statistical calculations

To calculate the statistics some assumptions are required:

- i The true input follows the alternating pattern, gesture, background, gesture...
- ii A given gesture has the probability  $\rho_G$  of being predicted correctly.
- iii Background has the probability  $\rho_B$  of being predicted correctly.
- iv If a background frame is predicted incorrectly it is equally likely that any one of the gestures is predicted instead.

Assumption (i) comes from how the application is intended to be used, while Assumption (ii) and (iii) are approximations that each gesture and background frame has the same probability no matter where in the chain they are placed. The final assumption, (iv) is an approximation that seems justified when looking at the confusion matrices from the neural network in table 5.3, for three gestures, and table 5.4, for all gestures.

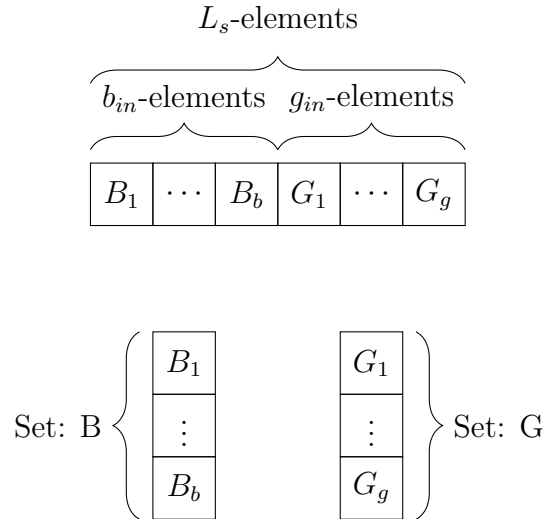
**Table B.1:** The probability of not missing a gesture in per cent for different values of  $L_G$  and  $L_s$  and  $\rho_G = \rho_B = 0,6$

$L_s \backslash L_G$	4	5	6	7	8	9	10
4	0,3983	0,0714	0,0128	0,0023	0,0004	0,0001	0,0000
5		2,8873	0,9165	0,2909	0,0924	0,0293	0,0093
6			0,0817	0,0146	0,0026	0,0005	0,0001
7				0,7259	0,2104	0,0610	0,0177
8					0,0163	0,0028	0,0005
9						0,1769	0,0472
10							0,0032

**Table B.2:** Values for (B.1) given different values for  $x$  and  $l_i$ , notice that the values where  $x \approx l_i$  are the ones with highest probability

$x \backslash l_i$	4	5	6	7	8	9	10
0	2.5600	1.0240	0.4096	0.1638	0.0655	0.0262	0.0105
1	15.3600	7.6800	3.6864	1.7203	0.7864	0.3539	0.1573
2	34.5600	23.0400	13.8240	7.7414	4.1288	2.1234	1.0617
3	34.5600	34.5600	27.6480	19.3536	12.3863	7.4318	4.2467
4	12.9600	25.9200	31.1040	29.0304	23.2243	16.7215	11.1477
5	0	7.7760	18.6624	26.1274	27.8692	25.0823	20.0658
6	0	0	4.6656	13.0637	20.9019	25.0823	25.0823
7	0	0	0	2.7994	8.9580	16.1243	21.4991
8	0	0	0	0	1.6796	6.0466	12.0932
9	0	0	0	0	0	1.0078	4.0311
10	0	0	0	0	0	0	0.6047

A sequence of frames with a length of  $L_s$  frames is according to Assumption (i) going to consist of a sequence of  $g_{in} \in [0, L_s]$  gesture frames of a certain gesture, and  $b_{in} = L_s - g_{in}$  background frames. If  $L_s$  is less than the sum of the length of the shortest gesture and the length of the shortest sequence of background this sequence can at most contain one gesture. The gesture frames and background frames can be separated into two sets, G and B, shown in Figure B.1. This separation combined with Assumption (ii) and (iii) allows for an easy way to calculate the likelihood of  $x$  frames in a given set being predicted as the correct class of gesture.



**Figure B.1:** How an input sequence is divided into subgroups, where the input is assumed to follow (i).

The probabilities will follow a binomial distribution [30] as follows,

$$P_i(x, l_i, \rho_s, \rho_f) = \binom{l_i}{x} \rho_s^x (1 - \rho_s)^{l_i - x}, \quad (\text{B.1})$$

where  $i$  denotes the set,  $x = 0 \dots l_i$  and represents as mentioned the number of frames predicted correctly as the gesture,  $l_i$  is the number of frames in the set, and  $\rho_s$  is the

probability for success, which in this case means predicting a frame as the gesture. The probability of getting at least  $x$  successes is simply

$$\sum_{x=n}^{l_i} P_i(x, l_i, \rho_s, \rho_f). \quad (\text{B.2})$$

Given equation (B.1), the probability of predicting  $x$  frames of the set  $G$  as the wanted gesture follows,

$$P_G(x) = \binom{g_{in}}{x} \rho_G^x (1 - \rho_G)^{g_{in}-x}. \quad (\text{B.3})$$

For set  $B$ ,  $\rho_s$  is unknown but given (iii) and (iv) it is easily calculated to  $\rho_s = (1 - \rho_B)/6$  and  $1 - \rho_s = 1 - (1 - \rho_B)/6 = (5 + \rho_B)/6$  which makes equation (B.1) for set  $B$ ,

$$P_B(x) = \binom{b_{in}}{x} ((1 - \rho_B)/6)^x ((5 + \rho_B)/6)^{b_{in}-x}. \quad (\text{B.4})$$

Now it is possible to calculate the probability that the neural network will predict a majority of the frames in a sequence as gesture frames, which this project defines as a correctly predicted gesture. The minimum frames required is  $N_{lim} = L_s/2$  rounded up. If  $g_{in} \geq N_{lim}$  probability of correctly predicting a gesture in that sequence is

$$P_{SG}(g_{in}) = \sum_{i=N_{lim}}^{g_{in}} P_G(i) + \sum_{i=0}^{N_{lim}-1} P_G(i) \left( \sum_{j=N_{lim}-i}^{b_{in}} P_B(j) \right), \quad (\text{B.5})$$

notice that the second term is for when there is not enough of  $G$ -frames predicted correctly so there need's to be  $B$ -frames predicted incorrectly as the gesture. This term only works for  $i$ 's where  $i + b_{in} \geq N_{lim}$ , the term should otherwise be zero.

The probability of missing a gesture of length  $L_G$ , where  $L_G \geq L_s$ , is

$$P_{\text{miss gesture}} = \prod_{i=N_{lim}}^{L_s-1} (1 - P_{SG}(i))^2 \cdot (1 - P_{SG}(L_s))^{L_g-L_s+1}, \quad (\text{B.6})$$

since there are two sequences which contain  $g_{in} \in [N_{lim}, L_s - 1]$ , one at the start and one at the end, and there is  $L_G - L_s + 1$  sequences of only gesture frames. Important to note is that even values of  $L_s$  give a significantly lower probability of missing a frame compared to its odd counterparts. This stems from the fact that the even number compared to the odd number right below it, will have the same  $N_{lim}$  but an extra term in both equation (B.3) and equation (B.4). This extra term is one of the middle terms of the series which carries with it a high probability thanks to the binomial term. So sequences with odd values of  $L_s$  lose a lot of probability. This is further shown in table B.1 and table B.2.

The equations for calculating the probability of incorrectly finding gestures in background is similar to the equations above. Since background segments, in general, are much longer than gesture segments, it is pointless to look at the edges of the segment. Therefore, there is only a need for calculating the probability of sequences with the pure background as input. The probability of finding a gesture in a sequence of pure background is then

$$P_{SB}(L_s) = 6 \sum_{i=N_{lim}}^{L_s} P_B(i), \quad (\text{B.7})$$

since there are six different gestures that the background can convert to.



## B.1 Finding gestures close to each other

A stricter definition of what classifies as a detected gesture is that more sequences of frames need to be classified as a gesture before being considered a detected gesture. This suppresses false positives even better, yet keeps the probability of not missing a gesture high. The probability of finding two gestures within a span of  $L_D$ —frames of each other is complicated. A simplification is to just look at a gesture of length  $L_D$  and see what the probability of finding only one or none of the gesture sequences. The probability can be found by using equation (B.5) and (B.6). The probability of getting a detection in one of the sequences but not the others is simply

$$P_{1\text{miss}}(g_{in}) = \frac{P_{SG}(g_{in})P_{\text{miss gesture}}}{1 - P_{SG}(g_{in})}. \quad (\text{B.8})$$

This makes the total probability of not detecting two or more gestures in the span of the whole gesture,

$$P_{2\text{miss}} = P_{\text{miss gesture}} + 2 \sum_{i=N_{lim}}^{L_s-1} P_{1\text{miss}}(i) + (L_G - L_s + 1)P_{1s}(L_s). \quad (\text{B.9})$$

The actual probability is lower than this since the window is moving along the gesture and can detect gestures even when not right on top of it.

## B.2 Probability in background

If the number of times a sequence is classified as a gesture is reset every time a gesture is detected, then the assumption that false positives only occurs in sequences of only background is valid. The probability of finding a sequence falsely classified is  $P_B(x)$ , being the probability function from equation (B.4). So the possibility of finding two falsely classified sequences within a certain amount of frames  $L_F$  will follow a negative binomial distribution [31]

$$\begin{aligned} P_{FF}(L_s, L_F) &= \binom{L_F - 1}{1} P_{SG}(L_s)^2 (1 - P_{SG}(L_s))^{L_s - 2} \\ &= L_F \cdot P_{SG}(L_s)^2 (1 - P_{SG}(L_s))^{L_s - 2}. \end{aligned} \quad (\text{B.10})$$

This means that the probability of having found a false positive after exactly  $L_F + x$  sequences will be a geometric distribution [32],

$$P_F(L_s, L_F, x) = (1 - P_{FF}(L_s, L_F))^{x-1} P_{FF}(L_s, L_F). \quad (\text{B.11})$$

The sum of this expression gives the probability of not classifying a gesture over  $x$  gives the probability of making it  $L_F + x$  before getting.

# C

## Machine learning optimizers

### C.1 Adam optimizer

The pseudo-code algorithm for the adam optimizer [14].

---

**Algorithm 1:** Adam, our proposed algorithm for stochastic optimization.  $g_t^2$  indicates the elementwise square  $gt \odot gt$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

---

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1st moment vector)

$v_0 \leftarrow 0$  (Initialize 2nd moment vector)

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_t - 1)$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

**end while**

**return**  $\theta_t$  (Resulting parameters)



# D

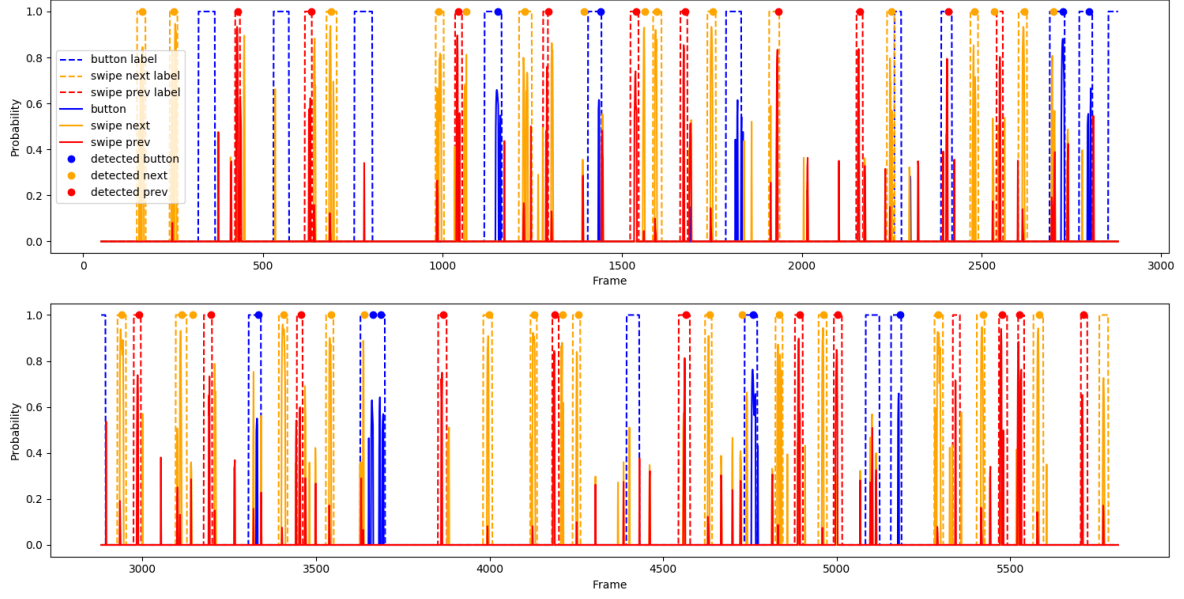
## Application results

**Table D.1:** The confusion matrix for the application where the amount of predictions are shown given a certain input and output.

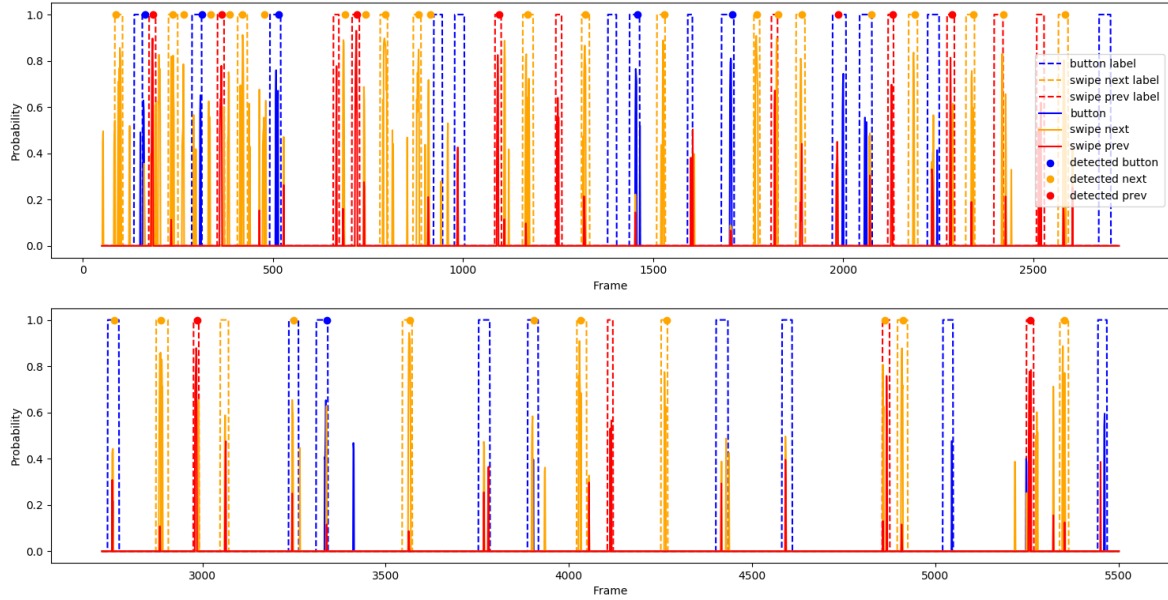
Input \ Output	Next	Button	Previous	Background	Total
Next	62	1	10	6	79
Button	12	37	7	21	77
Previous	16	0	51	13	80

**Table D.2:** The confusion matrix for the application where the amount of double predictions (i.e a second prediction on an already predicted input) are shown given a certain input and output.

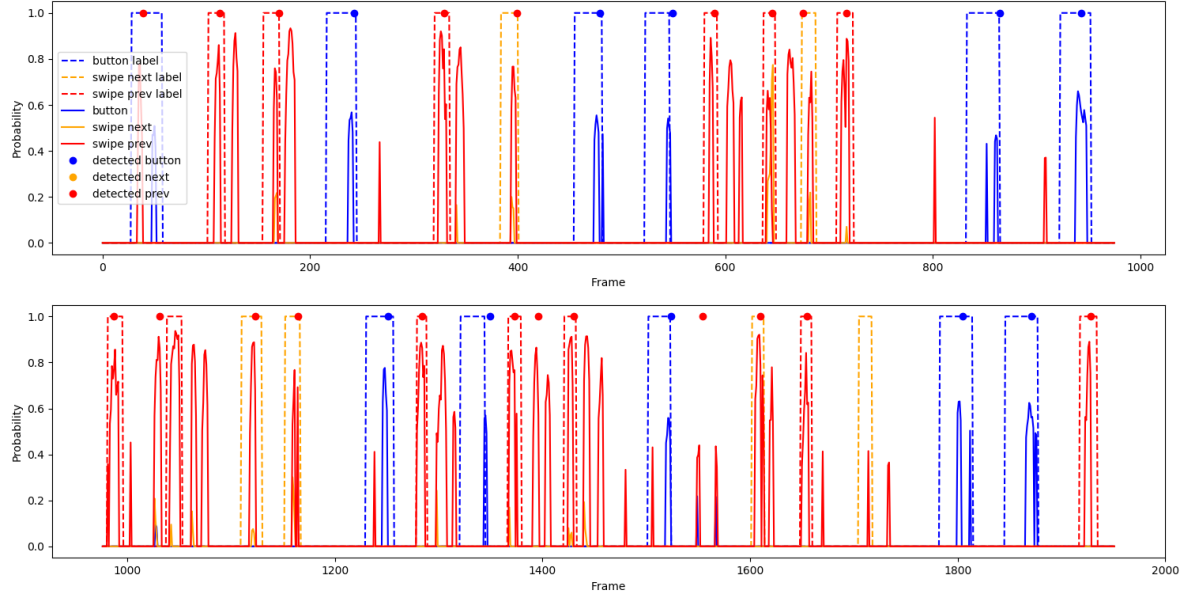
Input \ Output	Next	Button	Previous	Background	Total
Next	8	0	0	0	8
Button	2	1	1	0	4
Previous	6	0	0	0	6



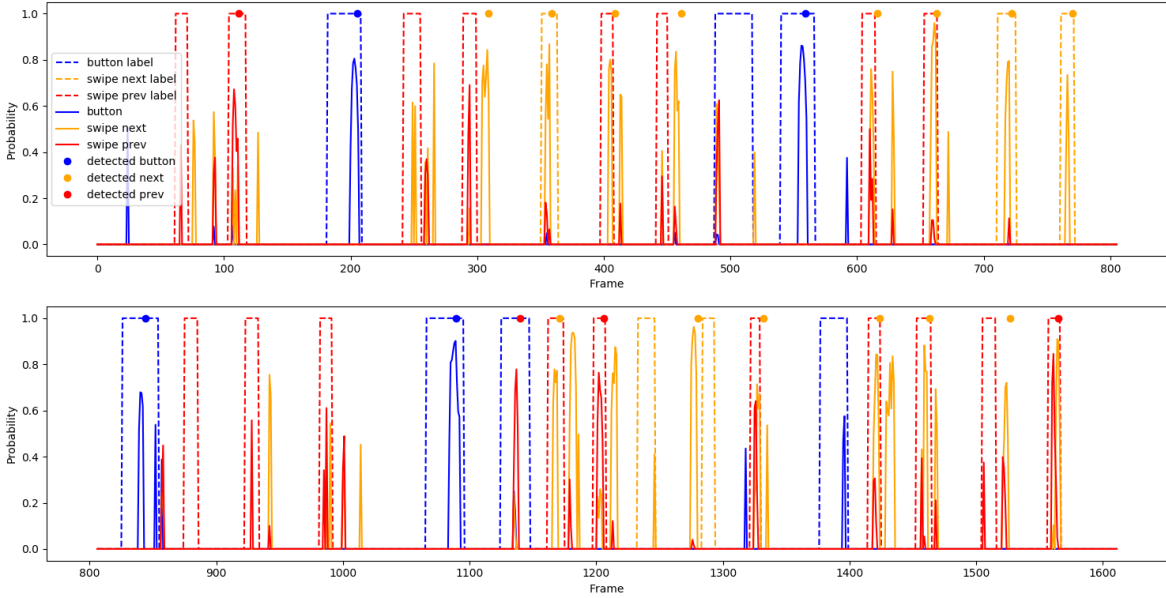
**Figure D.1:** The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order.



**Figure D.2:** The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order.



**Figure D.3:** The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order.



**Figure D.4:** The probabilities for each gesture plotted against time in the form of frames for gestures recorded in mixed order.