



HENRIQUE PINTO DA CRUZ FERNANDES

Licenciado em Ciências da Engenharia

CONTEXT-AWARENESS THROUGH ACTIVE RF SENSING

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2022

CONTEXT-AWARENESS THROUGH ACTIVE RF SENSING

HENRIQUE PINTO DA CRUZ FERNANDES

Licenciado em Ciências da Engenharia

Adviser: Rodolfo Alexandre Duarte Oliveira
Associate Professor, NOVA University Lisbon

Examination Committee

Chairs: Rui Miguel Amaral Lopes
Assistant Professor, NOVA University Lisbon
Nuno Miguel Goncalves Borges de Carvalho
Full Professor, Universidade de Aveiro

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2022

Context-awareness through Active RF Sensing

Copyright © Henrique Pinto da Cruz Fernandes, NOVA School of Science and Technology, NOVA University Lisbon.

The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

An alle, die das Innern am Leben erhalten.

ACKNOWLEDGEMENTS

O trabalho que aqui demonstro não se resume unicamente à tese final do curso; na verdade, estão aqui presentes cinco anos de trabalho, cinco anos cujo resultado não é fruto apenas do meu esforço, mas também de todos aqueles que me rodeiam. De todos, levo a inspiração, a motivação e as memórias que constituem os nossos momentos.

Em primeiro lugar, quero agradecer ao Professor Rodolfo Oliveira, por toda a disponibilidade e conhecimento na área que me foi transmitido, não só ao longo deste projeto, mas desde o início do curso. Gostaria de agradecer ao Instituto de Telecomunicações por me ter permitido desenvolver este trabalho através dos projetos Infocent-IoT (PT-DC/EEI/TEL/30433/2017) e RFSense (UIDB/50008/2020), financiados pela Fundação para a Ciência e a Tecnologia. De igual forma, um enorme obrigado aos meus companheiros de laboratório, Américo e João, com quem partilhei almoços e ideias construtivas. Não podia ter pedido melhor companhia para esta experiência.

À minha família, pela educação e carinho que me deram ao longo da minha vida. Aos meus avós maternos, pela boa comida que cozinham, pelo serviço de taxi prestrado e pelo bonito vocabulário que me ensinam. Aos meus avós paternos, pelo incentivo a estudar e a ser ambicioso com uma carreira. Aos meus pais, por sempre me terem dado tudo o que precisei para seguir os meus estudos, e por me deixarem aprender com todos os erros que cometi na minha vida, dando-me um raspanete quando necessário. À minha irmã mais velha, Catarina, que decidiu que éramos poucos em número a fazer a árvore de Natal, e decidiu resolver esse problema. Às minhas irmãs mais novas, espero que cresçam bem e com saúde. À minha prima, Carolina, por ser a doida varrida da família desde cedo, e por me emprestar alguma dessa loucura. Obrigado.

Aos meus amigos da faculdade, por todas as memórias que levo de vocês. Em especial, ao Tomás, à Carolina, ao Marco, à Miriam, ao Rui e à Bea, por me terem proporcionado várias oportunidades para aprender com vocês. Aprender não implica apenas conhecimento técnico; implica também o que me ensinaram sobre amizade. São verdadeiramente mágicos. À tia Carolina, por surpreendentemente ter uma paciência gigantesca para me ouvir e ajudar nos meus momentos mais difíceis, dentro e fora do contexto da tese. À Ângela, por me aturar em cerca de 40 cadeiras diferentes e 35 trabalhos de grupo, cuja presença está inevitavelmente neste trabalho também. És uma grande mulher. Obrigado.

A tutta la gang, soprattutto a Shana, per avermi fatto ridere ogni giorno che stiamo insieme. Tu sei il vero army di Shogun. Insieme, arriveremo a raggiungere grandi numeri, con o senza

Shenhe. Grazie.

Por fim, a alguém muito especial. É difícil escrever sobre ti. Pelos momentos que passámos juntos, por tudo o que me ensinaste, por todo o carinho, pela tatuagem que deixas na minha vida. Por seres um verdadeiro amigo e ouvinte. Não tenho palavras. Tudo o que partilhámos juntos até agora, levo para a vida. E tudo o que ainda vamos partilhar deixa-me com um sorriso na cara. Um obrigado de mil e um aviões a passar, *Miguel*.

A todos os outros cujo nome não foi mencionado, mas cuja vivência com vocês contribui para quem sou hoje e para o trabalho que aqui apresento, um enorme obrigado.

*“If you do not change direction, you may end up where you
were heading.” (Lao Tzu)*

ABSTRACT

Classification of contexts using Radio Frequency (RF) sensing devices is a complex task due to the way how the sensing signal is represented, particularly when compared to video cameras. Nonetheless, it brings several advantages, especially in conditions where vision blockage affects camera performance, such as fog. In addition, RF sensing technology does not raise privacy concerns and anonymity is usually protected.

The main purpose of this thesis is the development of a prototype, based on a Frequency-modulated continuous-wave (FMCW) device operating in the 76-81 GHz range band, capable of performing real time classification of static contexts. To do so, datasets were acquired and characterized, in order to develop a classification method capable of achieving high accuracy and fast run time needed for real time applications. The proposed solution is based on range masks, obtained from the characterized datasets, which when correlated with a frame produce high accuracy results. Furthermore, the solution adopted is independent of the room and target distance. Lastly, since human context-awareness can be seen as a time series problem, sequential classification algorithms (namely Viterbi algorithm and Recurrent Neural Networks (RNNs)) were also applied in order to improve the accuracy from the base classifier, at the cost of additional time to collect a few sequential samples.

The results achieved with the proposed methodology show that the RF sensing technologies are an alternative to video cameras to perform context-awareness. The developed classification method proves to be interesting due to the fact it is independent of the room where masks were acquired and classification is being performed. Furthermore, the use of cross-correlation in the classification method allows the target to be at various distances while only having one mask per class. These features can be adapted in future works, such as classification of non-static contexts, and its effectiveness is validated in this thesis through a rigorous performance evaluation methodology.

Keywords: Context-awareness, RF Active Sensing, Machine Learning, Deep Learning, Pattern Classification.

RESUMO

A classificação de contextos utilizando sensores ativos de radiofrequência é mais complexa quando comparada com outras tecnologias, como câmeras de vídeo, pela maneira como o sinal recebido é representado. No entanto, a mesma traz uma série de vantagens, sobretudo em condições onde a obstrução de visão afeta os resultados provenientes de sistemas com câmeras de vídeo, como nevoeiro. Em adição, este tipo de tecnologia não captura dados que possam comprometer a identidade de uma pessoa e costumam manter o seu anonimato.

O propósito principal desta tese é o desenvolvimento de um protótipo, com base num sistema de radar de onda contínua modulada por frequência (FMCW) operando na banda dos 76-81 GHz, capaz de classificar contextos estáticos em tempo real. Para tal, adquiriram-se conjuntos de dados, com o propósito de desenvolver um método de classificação capaz de satisfazer os requisitos necessários de aplicações em tempo real, como elevada precisão e tempo de resposta rápido. A solução desenvolvida baseia-se em máscaras de alcance, obtidas através do estudo dos conjuntos de dados anteriores, que quando correlacionadas com um *frame* obtêm elevada precisão de classificação. Por último, dado que a classificação de contextos humanos pode ser vista como um problema sequencial em tempo, aplicaram-se algoritmos de classificação sequencial (nomeadamente o algoritmo de Viterbi e Redes Neurais Recorrentes), de maneira a elevar a exatidão de classificação através do custo de algum tempo de resposta.

Os resultados obtidos com este trabalho demonstram que sensores ativos de radiofrequência são uma alternativa às câmeras de vídeo para realizar classificação de contextos. Ademais, o método desenvolvido demonstra ser interessante devido ao facto de conseguir realizar classificação de contextos em salas diferentes das usadas para adquirir as máscaras das classes. Por fim, o uso de correlação cruzada no método utilizado permite a classificação de alvos a várias distâncias recorrendo apenas a uma máscara por classe. Estas características podem ser adaptadas em trabalhos futuros, como classificação de cenários não estáticos, e a sua efetividade é validada nesta tese através de uma rigorosa metodologia de avaliação de desempenho.

Palavras-chave: Classificação de contextos, Sensores Ativos RF, *Machine Learning*, *Deep Learning*, Reconhecimento de Padrões.

CONTENTS

List of Figures	xix
List of Tables	xxi
Acronyms	xxv
1 Introduction	1
1.1 Motivation	2
1.2 Objectives	2
1.3 Thesis Structure	3
2 Related Work	5
2.1 Introduction to Sensing Technologies	5
2.1.1 Sensing Technologies	5
2.1.2 Machine Learning Classification Techniques	6
2.2 FMCW Radar	8
2.2.1 History of Radar	8
2.2.2 FMCW Radar Overview	8
2.2.3 Distance measuring	10
2.2.4 Radial Velocity measuring	13
2.3 Classification Features	14
2.3.1 Time-Domain Features	15
2.3.2 Frequency-Domain Features	16
2.3.3 Feature Engineering	17
2.4 Deep Learning	20
2.4.1 Introduction to Artificial Neural Networks	20
2.4.2 Convolutional Neural Networks	25
2.4.3 Recurrent Neural Networks	26
2.4.4 Generative Adversarial Networks	30
2.4.5 Autoencoders	31
2.5 State of the Art in RF Sensing	32
3 Experimental Setup, Scenarios and Datasets	35

CONTENTS

3.1	Experimental Setup	35
3.1.1	Radar Description	35
3.1.2	Radar Configuration	37
3.2	Classification Scenarios	38
3.3	General Methodology and Dataset Description	39
3.3.1	General Methodology	39
3.3.2	Dataset Acquisition and Description	40
3.4	Characterization of the Datasets	42
4	Classification Process	47
4.1	Classification Masks	47
4.1.1	Definition of Clutter Frame	47
4.1.2	Types and Definition of Masks	48
4.2	Classification Method	50
4.2.1	Classifying the None Context	51
4.2.2	Classifying Standing and Prostrated Contexts	52
4.3	Sequential Classification	53
4.3.1	Viterbi Sequential Classification	54
4.3.2	RNN Sequential Classification	55
5	Prototype and Performance Evaluation	57
5.1	Architecture	57
5.2	Realtime Acquisition, Calibration and Classification	58
5.3	Acquisition of the Datasets with the Prototype	61
5.4	Performance Evaluation Results	62
5.4.1	Static Scenario Classification	62
5.4.2	Non-Static Scenario Classification	63
6	Conclusions and Future Work	65
6.1	Conclusions	65
6.2	Future Work	66
6.2.1	Increasing the Dataset Size	66
6.2.2	Increasing the Variety of Rooms	66
6.2.3	Exploration of Radar Parameters	66
6.2.4	Addition of Static and/or Non-Static Contexts	66
6.2.5	Adaptation of Classification Model	67
	Bibliography	69

LIST OF FIGURES

2.1	Block Diagram of a FMCW Radar, adapted from [34].	9
2.2	Sawtooth Shaped Frequency Modulated Signal in (a) the time domain and (b) in the frequency domain, adapted from [34].	9
2.3	Detection of multiple targets in (a), and the respective transmitted and received echoes of the signal in (b), as well as the IF signals; in (c), the beat frequencies from each IF signal are visible in the frequency spectre. Adapted from [34].	10
2.4	Operating principle of the FMCW Radar.	11
2.5	Mixing operations of: Real (on the left) and Complex (on the right) baseband architectures in the frequency domain, adapted from [35].	12
2.6	A fully connected neural network (on the left) and a scheme of an artificial neuron (on the right), adapted from [41].	21
2.7	Examples of accuracy and loss plots of the training and validation per epoch. The plots on the left show a convergent loss error network, in which the accuracy is maximized and the loss is minimized for both the training and validation datasets. On the contrary, the plots on the right achieve convergence only for the training dataset, and it is an example of overfitting.	22
2.8	Plot of the sigmoid activation function.	23
2.9	Plot of the hyperbolic tangent activation function.	24
2.10	Plot of the ReLU activation function.	25
2.11	General architecture of a Convolutional Neural Network, adapted from [41].	26
2.12	General structure of a folded RNN cell (on the left) and an unfolded RNN cell (on the right), adapted from [49].	27
2.13	General structure of a LSTM cell, adapted from [49].	28
2.14	General structure of a GRU cell, adapted from [49].	29
2.15	Structure of a GAN, adapted from [52].	30
2.16	General architecture of an Autoencoder, adapted from [58].	31
2.17	General architecture of a Variational Autoencoder, adapted from [58]. . . .	32
3.1	Front view picture of the IWR1843BOOST Evaluation Module used in this work, taken from [67].	35
3.2	Photo of the experimental setup in room R.1.	36

LIST OF FIGURES

3.3	Range-Doppler Map (RDM) processing chain.	36
3.4	<i>mmWave Demo Visualizer</i> v3.5.0, available at [68].	37
3.5	Visual representation of classification scenarios, alongside a range array describing them.	39
3.6	General methodology scheme representation of the developed classifier. . .	40
3.7	Schematic of the Acquisition Rooms R.1 and R.2 (figure is not to scale). . .	42
3.8	Mean of Range Arrays per Label from Dataset Group 1.	43
3.9	Mean of Range Arrays per Label with clutter removed from Dataset Group 1.	44
3.10	Histograms for Range Bin 25 separated by class label for Dataset Group 1.	45
3.11	Histograms for Range Bin 45 separated by class label for Dataset Group 1.	45
4.1	Frame before and after undergoing the clutter subtraction process.	48
4.2	Frame before and after undergoing the range bins restriction process. . . .	49
4.3	Mask before and after undergoing the normalization process. The plot on the right represents an example of possible final mask $M_{subtraction_i}$ for the <i>Standing</i> scenario.	49
4.4	Frame before and after undergoing the clutter division process.	50
4.5	Gaussian Distribution. The standard deviation of the distribution is represented as σ . The graphic describes which percentage of values is within one, two and three standard deviations of the mean.	52
4.6	Frame undergoing the classification process (right) through the cross-correlation with each scenario's mask (left). It is visually possible to check that the frame to be classified has the same pattern as the prostrated mask M_2	53
4.7	Architecture of the Recurrent Neural Network (RNN) to perform sequential classification.	56
5.1	Block diagram representing the architecture of the developed prototype. . .	58
5.2	FIFO Queue data transmission system used between the acquisition thread and the classifier thread.	59

LIST OF TABLES

3.1	Radar parameters used in this work.	38
3.2	Classification scenarios and respective description.	38
3.3	Dataset conditions description.	41
3.4	Number of Files per Dataset Group.	42
3.5	Dataset label count.	42
5.1	Configurable parameters used in the developed prototype, alongside their description and value.	58
5.2	Accuracy values obtained for static scenario classification.	63
5.3	Processing runtime values obtained for static scenario classification.	63
5.4	Accuracy values obtained for non/static scenarios classification.	64
5.5	Processing runtime values obtained for non-static scenario classification.	64

LIST OF LISTINGS

1	Calibration phase of the <i>run</i> function, present in the classifier thread. .	59
2	Classification stage of the <i>run</i> function, present in the classifier thread.	60
3	Sequential Classification segment for Viterbi and RNN in the developed prototype.	61
4	Dataset record option of the <i>run</i> function, present in the classifier thread.	62

ACRONYMS

ADC	Analog to Digital Converter 10, 11, 36, 38
AEs	Autoencoders 31, 32
AI	Artificial Intelligence 3, 6, 8
ANNs	Artificial Neural Networks 7, 8, 21, 25
CNN	Convolution Neural Network 5, 6, 25, 33, 34, 67
DAEs	Denosing Autoencoders 31
DL	Deep Learning 1, 3, 20, 33
DNN	Deep Neural Network 1, 2, 3
DRDT	Dynamic range-Doppler trajectory 33
DSP	Digital Signal Processing 10
FE	Feature Extraction 18, 19
FFT	Fast Fourier Transform 10, 13, 14, 15, 16, 33, 36, 37
FIFO	First-In First-Out xx, 59
FMCW	Frequency-modulated continuous-wave xiii, xix, 1, 3, 8, 9, 11, 16, 32, 33, 34, 35, 65
FS	Feature Selection 18, 19
GAN	Generative Adversarial Network xix, 30, 31
GPUs	Graphics Processing Units 6
GRU	Gated Recurrent Units xix, 27, 29, 30
GUI	Graphic User Interface 58, 59
I	In-phase component 12
I/Q	In-phase and Quadrature components 11, 15, 16, 36, 57
ICA	Independent Component Analysis 20
IF	Intermediate Frequency xix, 10, 11, 12, 13, 14
IoT	Internet of Things 2

ACRONYMS

KNN	K-Nearest Neighbour 7, 33
KPIs	Key Performance Indicators 3, 61, 62, 63, 64
LASSO	Least Absolute Shrinkage and Selection Operator 19
LSTM	Long Short-Term Memory xix, 23, 27, 28, 29, 30, 33, 34, 55, 67
MI	Mutual Information 18
ML	Machine Learning 1, 3, 6, 14, 33, 41, 42, 65, 66
MSE	Mean Squared Error 22
PCA	Principal Component Analysis 19, 20, 31
Q	Quadrature component 12
RDM	Range-Doppler Map xx, 36, 37, 40, 41, 47, 57, 58, 67
RF	Radio Frequency xiii, 1, 2, 3, 32, 33, 65, 66
RMS	Root Mean Square 15
RNN	Recurrent Neural Network xix, xx, xxiii, 26, 27, 55, 56, 59, 61
RNNs	Recurrent Neural Networks xiii, 56, 63, 64, 65, 67
RS	Remote sensing 1
RTD	Round-Trip Delay 13
RTS	Real-Time System 1, 33
RX	receiving 35, 36, 37
SVM	Support Vector Machine 7
ToF	Time of Flight 6, 8
TX	transmitting 35
UWB	Ultra-Wideband 33
VAEs	Variational Autoencoders 31, 32
VCO	Voltage Controlled Oscillator 9, 11

INTRODUCTION

Remote sensing (RS) is, by definition, the process of acquiring information without relying on physical contact. In addition, Fussel et al. [1] suggest that this process is not only remote, but also indirect. Consequently, Radio Frequency (RF) sensing can be described as a type of RS whose source of information relies on Radio Frequency (RF) waves; furthermore, active sensing implies that the system is both the source (transmitter) and the receiver of these waves [2].

The concept of context-awareness can be split in two: the word **context** searches to identify the surrounding, learning *where* and *who* is around; the term **context-awareness** describes a system's capability to do so [3].

As expected, RF sensing systems fulfil the role in applications like context-awareness who require sensing of some sort, such as the measurement of distances and velocities to one or multiple targets. However, the patterns that describe a context through RF sensing might exhibit high similarity for different contexts, making the process of distinguishing the contexts even more challenging and complex.

In the past two decades, Machine Learning (ML) and Deep Learning (DL) techniques have been demonstrating great potential in understanding and identifying patterns from an input data [4]. Thus, they have been in the foundation of many emerging applications, such as natural language processing [5] and classification algorithms [6]. Naturally, they have been used to overcome the difficulties that context-awareness poses.

Nonetheless, most of today's applications demand a short answer time to match their Real-Time System (RTS) nature. Feature engineering has been used to select and/or extract the most relevant ones before feeding them to classification algorithms or Deep Neural Network (DNN)s, in attempt to decrease the complexity and computation time, as well as improving the accuracy.

In this thesis, an active Frequency-modulated continuous-wave (FMCW) radar is used

to provide raw data to a context-awareness system. At a final stage, we compare different classification methods based on multiple DNNs, which are developed to measure and compare the classification accuracy and the computation time of the raw data with the selected features of each algorithm.

1.1 Motivation

The recent and fast development of technology over the past decades has brought a wide variety of tools that can be used to improve our daily routines, especially with the 'smart' trend of devices and Internet of Things (IoT). Specifically, context-awareness can be applied in many of these different emerging areas. For instance, in health, it allows the measurement of vital signs in a contactless way, removing the need for wearable sensors [7]. In smart-cities, it can optimize the uptime and downtime of a semaphore at a crosswalk, based on the distance and/or presence/absence of pedestrians, or act as an alarm in buildings, detecting the movements of an unknown individual inside. Elderly care is also an example that could benefit from context-awareness: a system can detect potentially dangerous scenarios, like the fall of an elderly person, and act accordingly. Lastly, context-awareness is thought to be in the core of sixth generation mobile networks, in order to have a more agile and adaptive radio system [8].

Context-awareness can be achieved with different sensing technologies: video cameras, LiDAR sensors, and radars, are just some examples. These will be further discussed in the next chapter. The use of radars has numerous advantages: to start with, radars can operate in different weather conditions, as the operating frequency and wavelength can be adjusted to overcome them [9]. Likely, they can operate behind electromagnetically transparent materials, like walls or cars, not affecting aesthetics when they are of importance. Furthermore, unlike video camera systems, radar performance is not significantly affected by low lightning or occlusion, and does not compromise the identity of individuals, satisfying data protection regulations.

1.2 Objectives

This thesis' goal consists in developing a context-awareness system through RF sensing with high accuracy. Furthermore, the prototype developed should aim to fit real-time applications requirements, such as high speed response. To achieve this goal, a human is placed in front of the radar system, performing static actions. Three scenarios were defined: Standing and Prostrated, where a person is performing the correspondent action in front of the radar, and None, which represents the absence of a person in front of the radar. The goals to be achieved in this work are listed as follows:

1. Exploring commercial hardware to set up a RF sensing system solution. The system needs to be configured so that the signals it captures can reflect the context around

- it. This includes both the hardware and the software domains;
2. Obtain the dataset(s) with the developed RF system. An environment is set to capture labeled samples according to the various contexts;
3. Selection and/or extraction of the most relevant features. Information provided from these systems is often hard to be used in classification algorithms without being pre-processed, resulting in either poor accuracy or slow computation times. Reducing the system complexity is therefore fundamental to not only filter data, but simultaneously handle the most important features that effectively contribute the most for the success of the context recognition;
4. Utilize existent ML/DL algorithms to construct a classifier that fulfills real-time applications requirements;
5. Evaluation of the developed RF solution performance.

1.3 Thesis Structure

This thesis is organized as follows. Chapter 2 describes the related work in this area by presenting a list of sensing technologies, followed by an explanation on the basics of Frequency-modulated continuous-wave (FMCW) radar operation. Additionally, a list of possible features to be selected from the radar output data is described, as well as a detailed explanation on typical models for DNNs and other AI techniques. Furthermore, a review on the state-of-the-art in RF sensing is presented. Chapter 3 describes the experimental setup utilized to acquire the datasets for both offline and online classification. In addition, the structure of the acquired datasets through this setup is also detailed. Lastly, the three scenarios chosen to classify are presented, and an adequate characterization of their statistical properties is performed through different plots and histograms. Chapter 4 is centered on the description of the method to classify samples. Specifically, core concepts of the classification, such as the calibration phase, the definition of the classification masks, and the classification of the scenarios, both individually (based on a single classification) and sequentially (based on consecutive classifications), are explained. Chapter 5 details the developed prototype, used to acquire offline datasets and/or perform real time classification using the methods detailed in the previous chapters. Furthermore, the results obtained with all classification methods are presented, where the Key Performance Indicators (KPIs) are mainly focused on the classification accuracy and time needed to classify the data. Lastly, in Chapter 6, the results achieved with this work are discussed, with relation to the goals proposed in the previous section. Furthermore, possible future directions are also discussed, aiming at improving the developed RF sensing classification prototype.

RELATED WORK

2.1 Introduction to Sensing Technologies

2.1.1 Sensing Technologies

Sensors are devices capable of capturing and measuring a physical property from the real world. As such, they are crucial components in every data acquisition hardware and, by extension, in context-awareness systems. The signals obtained from sensors are usually stored as data in a computer, which can then be processed and analyzed by software [10].

There are numerous types of sensors with different properties, which makes them fit a certain role better than others. Below, a list of common sensors is presented, along with a brief description of some advantages and disadvantages.

Video Camera: Video cameras are devices capable of recording and storing visual information in the format of images. They come in various formats and shapes, which makes them suitable for many applications. Furthermore, through the usage of Convolution Neural Network (CNN)s, camera systems offer great accuracy when it comes to classification [11]; however, this demands powerful hardware. Since they rely on image, their accuracy is highly dependable on the weather, illumination conditions and visual obstruction.

Infrared: Infrared are electromagnetic waves with higher wave length than visible light. Their radiation cause a change in the dipole moment of molecules, which makes it a powerful technology for studying energy states of a target, such as temperature. They operate remotely and under low lightning conditions, which justifies their usage in night vision and thermography applications. However, they require line of sight, as common objects block them, and are affected by weather conditions.

Infrared waves are also known to damage the human eye [12], which can be a reason to prefer other sensing technologies.

Ultrasonic: The principle of ultrasonic sensors is simple: they emit a sound wave (at a higher frequency than humans can hear), which is reflected by the target. Based on the Time of Flight (ToF) of the sound wave, the distance is measured. Small targets or soft materials can disturb the wave by scattering or absorption phenomena; they are also sensible to temperature changes. Other than that, these sensors can sense all types of materials. Their biggest advantage is the low-cost and compact size.

LiDAR: Light Detection and Ranging sensors operate similarly to ultrasonic sensors. They calculate the distance to a target by emitting a laser and measuring the ToF. LiDAR sensors are very accurate and precise, offering excellent range [13] and velocity [14] resolutions. As a consequence, these sensors require great computing power, and are often very expensive.

Radar: Radars rely on radio waves in order to indirectly measure the distance and velocity to a target by emitting a wave and listening for its echo. This measurement can be achieved by several different methods, depending on the type of radar used. Generally, they can operate under all weather and lightning conditions, and can also penetrate most objects, depending on the chosen wave length. However, classification through radar can be difficult.

2.1.2 Machine Learning Classification Techniques

Artificial Intelligence (AI) can be simply described as intelligence associated to machines [4]. In fact, the idea is to incorporate certain characteristics from the human brain into machines, such as the concept of "learning".

In 1959, Arthur Samuel described Machine Learning (ML) as a specific field of AI which "gives computers the ability to learn without being explicitly programmed"[4]. This means that a computer can resort to a mathematical algorithm to identify patterns from data provided by sensors in order to classify them, for example. Although ML research started in the 1950s, recent development on hardware, such as implementations of CNNs in Graphics Processing Units (GPUs) [15], have made this technology spread.

ML algorithms can be divided in three categories [16]. Each instance from the dataset can be labeled (categorized), in order to help classification; this is *supervised learning*. Contrarily, *unsupervised learning* does not have its dataset instances labeled, which can lead to unexpected but interesting classifications. *Reinforcement learning* helps the learner by giving a measure on how well it is performing. However, it does give the learner some freedom, by not giving a direct answer in its feedback.

Since the datasets used in this thesis are labeled, a supervised learning approach is taken. Classification consists of learning the boundaries between different classes,

correctly identifying which items belong to which class. Through the learning dataset, a mapping function successively learns how to distinguish different categories. Several algorithms used in radar systems are described in the list below.

K-Nearest Neighbour: the K-Nearest Neighbour (KNN) algorithm is very simple to implement, and its core is based on a very basic concept. A new point is classified based on the class of the K nearest points close to it, where K is an integer. Although it is fundamentally simple to implement and has no training associated to it, this algorithm can be computationally expensive, depending on the size of the dataset. The bigger the dataset, the more distances need to be calculated, hence increasing the computational cost. Since it was originally studied in 1951, KNN classifiers have been improved to better suit the needs of different modern applications, such as introducing weights in the classifier [17].

Naive Bayes: the Bayes theorem, in which this classifier is based on, states that it is possible to calculate the conditional probability of two events, as long as certain information about them is known. As a classifier, the algorithm calculates the probability of the class Y being assigned to the point X, or in other words, $P(Y|X)$. To do so, it iterates through the whole dataset, calculating the necessary probabilities needed for the theorem. Afterwards, it assigns the class Y with the highest probability to X. Mathematically, it assumes that all events are independent. Although this is rarely true, it does not affect the outcome significantly [18, 19]. Its computational cost is quite efficient, scaling linearly with the size of the dataset; nonetheless, its real-time operation is heavy.

Support Vector Machine: in its classic form, Support Vector Machine (SVM) is an algorithm intended for linear binary classification, that is, to distinguish between two classes separated by a linear function. SVM builds a hyper-plane capable of dividing the two classes by maximizing the margin between them [20]. As of today, kernel functions can be applied to perform non-linear classification, by projecting the data to a higher-dimensional space. Despite its high performance in high dimensional spaces [21], Yang et al. [22] point out the fact that SVMs "need to solve quadratic programming problems repeatedly" during its training process, which is a clear disadvantage when compared to other algorithms.

Artificial Neural Networks: Artificial Neural Networks (ANNs) operate by mimicking certain functionalities of the human brain. They are composed of an input layer and an output layer, often connected by multiple inner layers (typically called hidden layers). Every layer has one or more neurons, each of them performing a linear operation and passing the result to the neurons of the next layer. ANNs are commonly used as classifiers; however, their flexibility makes them suitable for other applications, such as mapping, regression, modeling and clustering [21]. Due to

their complexity, ANNs have a very high training time. Nonetheless, their flexibility with the input data and low operational time makes them suitable for real-time applications.

2.2 FMCW Radar

2.2.1 History of Radar

Radar systems are devices that are capable of determining an object's distance, angle, or velocity through the use of radio waves. In fact, the term RADAR stands for Radio Detection and Ranging system. The idea of utilizing these radio waves' properties is usually credited to WWII; however, initial research dates back to 1897, when Alexander Popov realized that a passing ship caused interference in wireless signals [23]. In 1900, Nikola Tesla proposed that electromagnetic waves could be used to determine not only a target's position, but also its speed and course [24].

The first functional systems were built soon after, when in 1904 Christian Hülsmeyer developed the "telemobiloscope", a continuous wave device which could detect the presence of ships at the sea in a 3.2 km range under dense fog conditions [23]. Experiments continued throughout the years until the WWII, where radar systems played a significant role in the Alliance's victory.

Radar development continued, as they proved to be useful in applications of public interest, such as aviation [25] and meteorology [26]. With the emergence of Artificial Intelligence (AI) in the past decades, radars have also become important tools in new areas, namely in autonomous driving [27, 28], motion classification [29], object recognition [30], and healthcare monitoring [31, 32].

Radars can generally be divided in two categories. Active radars act as both transmitter and receiver: after transmitting a radio signal, they wait for an echo to be reflected on an object's surface. On the other hand, passive radars listen for echoes transmitted from an antenna which is usually placed in a different location [2].

Furthermore, as described by Hasch et al. in [28], active radars have two predominant base bands: one in the 24GHz frequency and another in the 77GHz frequency. At the cost of being more challenging to develop, higher frequency radars offer several advantages, such as being about three times smaller, having a better relative bandwidth (approximately 5% vs the 24GHz's 17%) and a longer operation range, between others.

2.2.2 FMCW Radar Overview

Frequency-modulated continuous-wave (FMCW) radars are a type of active radars. They operate by continuously transmitting a wave. As a consequence, properties such as the distance to an object cannot be measured directly through the Time of Flight (ToF) of the reflected wave, unlike classical pulse radars; instead, the frequency of the transmitted

wave changes over time, allowing these properties to be retrieved [33]. A typical block diagram for an FMCW Radar is shown in Figure 2.1.

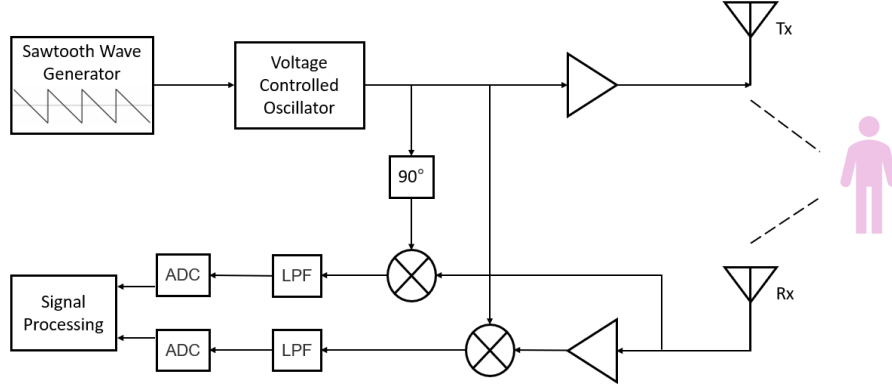


Figure 2.1: Block Diagram of a FMCW Radar, adapted from [34].

A waveform generator is responsible for generating a signal whose shape will be used to modulate the frequency of the transmitted signal over time in the Voltage Controlled Oscillator (VCO). In Figure 2.2, the transmitted signal is modulated in a sawtooth shape, but other shapes can be used, such as ramp, sine or triangular. This process is periodic, in which each of these signals is defined as a chirp, and each of the periods is defined as a sweep. The modulated signal is amplified before being transmitted by the antenna.

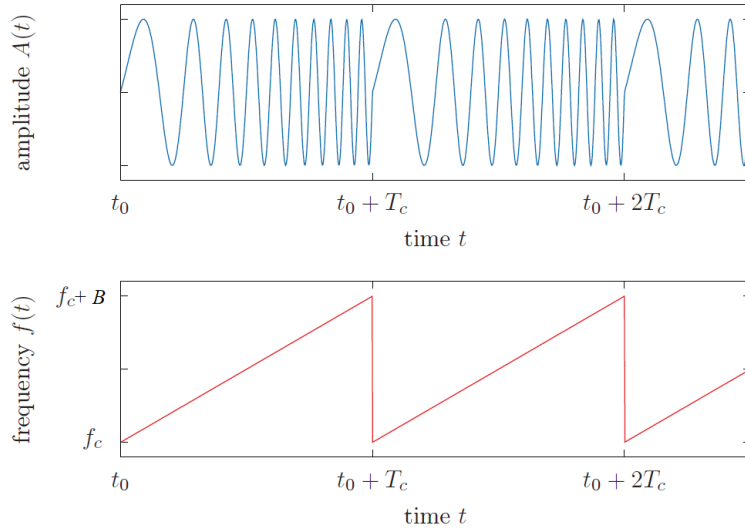


Figure 2.2: Sawtooth Shaped Frequency Modulated Signal in (a) the time domain and (b) in the frequency domain, adapted from [34].

Afterwards, the receiver waits for an echo to be reflected off an object's surface. The received signal is then amplified to nullify the effects of path attenuation. The following block mixes the original transmitted signal with the received one. This originates what

is called the Intermediate Frequency (IF) signal, characterized by the beat frequencies, which is then submitted to a low-pass filter. This process can be seen in Figure 2.3.

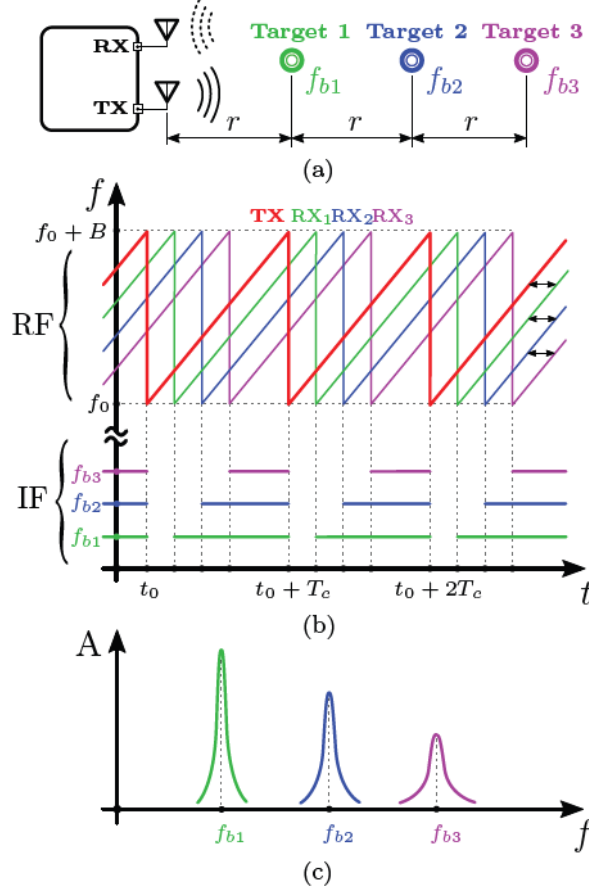


Figure 2.3: Detection of multiple targets in (a), and the respective transmitted and received echoes of the signal in (b), as well as the IF signals; in (c), the beat frequencies from each IF signal are visible in the frequency spectre. Adapted from [34].

The last step includes submitting the IF signal to an Analog to Digital Converter (ADC), which in turn serves as the input to the Digital Signal Processing (DSP) block. This block applies Fast Fourier Transform (FFT), which can be used to extract information such as the range, velocity, and angle of the targets.

2.2.3 Distance measuring

When a sawtooth shape is used to modulate the frequency of the signal transmitted by the antenna, the transmitted and received signals of a FMCW radar are presented in Figure 2.4.

The frequency of the signal transmitted by the antenna can be described as

$$f(t) = f_c + \frac{B}{T_c}t, \quad (2.1)$$

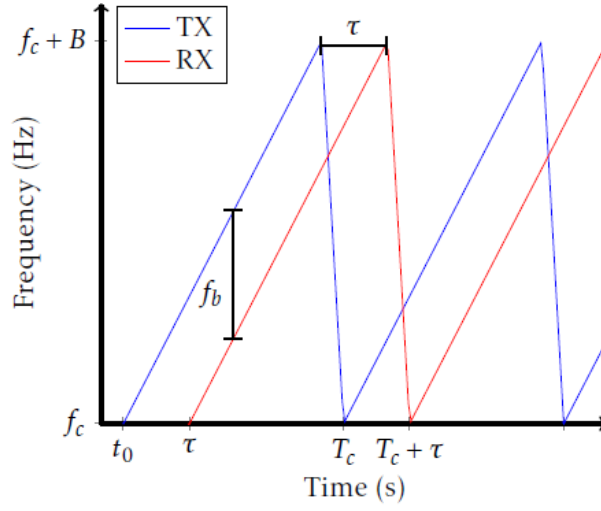


Figure 2.4: Operating principle of the FMCW Radar.

where f_c is the carrier frequency, and B and T_c correspondent to the chirp's bandwidth and duration, respectively.

Generally, a signal can be described as

$$S(t) = \Re \left[e^{j2\pi f t} \right] = \cos(2\pi f t). \quad (2.2)$$

Due to the VCO's integration effect on (2.1), and substituting it into (2.2), the equation of the transmitted signal can be described as

$$S_t x(t) = \Re \left[e^{j2\pi t \left(f_c + \frac{B}{2T_c} t \right)} \right] = \cos \left(2\pi \left(f_c t + \frac{B}{2T_c} t^2 \right) \right). \quad (2.3)$$

Since the received signal is just a delayed and attenuated version of the transmitted signal, from (2.3) the equation of the received signal can be written as

$$S_r x(t) = S_t x(t - \tau) = \cos \left(2\pi \left(f_c (t - \tau) + \frac{B}{2T_c} (t - \tau)^2 \right) \right). \quad (2.4)$$

As described in the previous section, the signals S_{tx} and S_{rx} are mixed in order to create the IF signal. In FMCW radars, this is achieved using either a real or a complex baseband architecture. A real baseband architecture only mixes the in-phase component of the signals, whereas in a complex baseband architecture a quadrature mixer is utilized, mixing both the In-phase and Quadrature components (I/Q). The mixing process of both is depicted in Figure 2.5. Consequently, real baseband architectures only need one IF and ADC chain, effectively reducing the hardware cost when compared to complex baseband architectures (which require two of these chains, as illustrated in Figure 2.1) and is thus preferred in most FMCW radars; however, for the additional cost, complex baseband architectures offer a better noise profile and tolerance to interference [35].

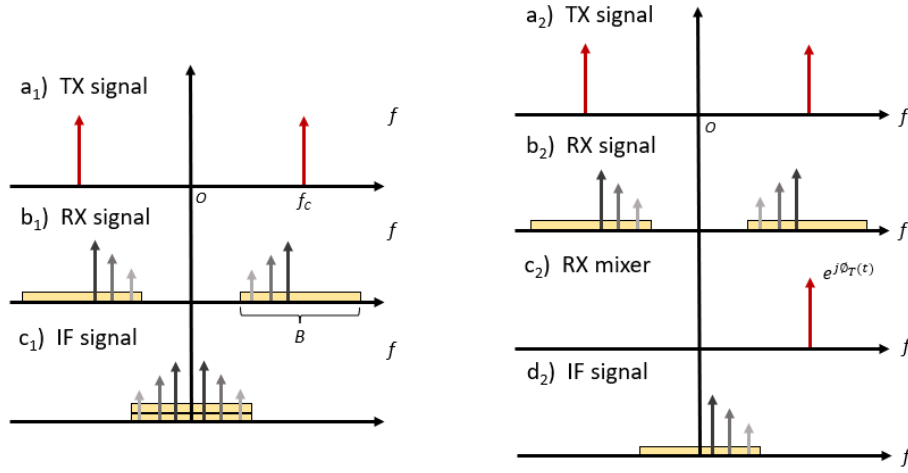


Figure 2.5: Mixing operations of: Real (on the left) and Complex (on the right) baseband architectures in the frequency domain, adapted from [35].

In order to understand the complex mixing operation, the following trigonometric identities are used:

$$\cos(a)\cos(b) = \frac{1}{2}(\cos(a+b) + \cos(a-b)), \quad (2.5)$$

$$\cos(a)\sin(b) = \frac{1}{2}(\sin(a+b) - \sin(a-b)). \quad (2.6)$$

The mixed signal's In-phase component (I) can be described as

$$\begin{aligned} S_{tx}(t) \cdot S_{rx}(t) = & A \cos \left(2\pi \left((2f_c - \tau)t + \frac{Bt^2}{T_c} + \frac{\tau^2}{2T_c - f_c\tau} \frac{Bt^2}{2T_c} \right) \right) \\ & + A \cos \left(2\pi \left(\frac{B\tau t}{T_c} + f_c\tau - \frac{B\tau^2}{2T_c} \right) \right). \end{aligned} \quad (2.7)$$

The Quadrature component (Q) can be obtained by shifting a copy of the transmitted signal by 90°, mixing it with the received signal as follows:

$$\begin{aligned} S_{tx}^{90}(t) \cdot S_{rx}(t) = & A \sin \left(2\pi \left((2f_c - \tau)t + \frac{Bt^2}{T_c} + \frac{\tau^2}{2T_c - f_c\tau} \frac{Bt^2}{2T_c} \right) \right) \\ & + A \sin \left(2\pi \left(\frac{B\tau t}{T_c} + f_c\tau - \frac{B\tau^2}{2T_c} \right) \right). \end{aligned} \quad (2.8)$$

The first sinusoidal term in both equations (2.7) and (2.8) is usually eliminated by the low-pass filters at each of their respective chains, or by the mixer's cutoff frequency and following radio components [36].

The IF signal is finally characterized as

$$S_{IF}(t) = \exp \left(j2\pi \left(\frac{B\tau t}{T_c} + f_c\tau - \frac{B\tau^2}{2T_c} \right) \right). \quad (2.9)$$

If only one target is in front of the radar, then by differentiating the phase term of the IF signal in (2.9) with respect to time, the value of the beat frequency can be written as

$$f_b = \frac{d}{dt} \left(\frac{Bt\tau}{T_c} + f_c\tau - \frac{B\tau^2}{2T_c} \right) = \frac{B\tau}{T_c} \quad (2.10)$$

If more than one target is within the radar's range, then each of the beat's frequency can be obtained by applying a Range FFT on the digital signal.

Given that τ represents the full Round-Trip Delay (RTD), then the range of a target can be intuitively calculated as

$$R = \frac{c\tau}{2}, \quad (2.11)$$

where c is the speed of light, approximately equal to 299792458 m/s. As shown in (2.10), the beat frequency f_b is linearly proportional to the RTD. Rewriting (2.10) and using (2.11) leads to

$$R = \frac{cf_b T_c}{2B}, \quad (2.12)$$

which allows the range of a target to be calculated based on the beat frequency obtained from the IF signal.

Lastly, if the radar is required to detect multiple targets, then it is necessary to find the range resolution. The range resolution is a radar's capability to distinguish between various distances and should be based on the required precision. It is expressed as

$$\Delta R = \frac{c\Delta f_b T_c}{2B} = \frac{c\frac{1}{T_c} T_c}{2B} = \frac{c}{2B}. \quad (2.13)$$

2.2.4 Radial Velocity measuring

Radars can also infer the velocity of a moving target. In fact, a moving target causes a frequency shift in the received signal: it increases if it's moving towards the radar, and decreases in the reverse case. This phenomenon is known as the Doppler effect. Since the Doppler frequency shift depends on the radial velocity of the target, radar systems can calculate the latter by knowing this frequency shift [37].

The Doppler frequency is characterized as

$$f_d = \frac{2v}{c} f_c. \quad (2.14)$$

Replacing (2.14) into the IF signal described in (2.9), we obtain the following equation:

$$S_{IF}(t) = \exp \left(j2\pi \left(\left(\frac{B\tau}{T_c} + \frac{2v}{c} f_c \right) t + f_c\tau - \frac{B\tau^2}{2T_c} \right) \right). \quad (2.15)$$

As explained in the previous section, the range of a target can be obtained with just a single chirp. However, the frequency of the IF signal in (2.15) is dependent on not

only the distance to the target, but also the velocity at which it travels, as shown with the presence of the Doppler component. Therefore, multiple chirps are transmitted in sequence (recall Figure 2.4). Given the short period of each chirp, the target's range does not change significantly to the point where a new range value will be assigned to it. Furthermore, given that the phase of the IF signal in (2.15) is the integral of its frequency, it will only slightly change between successive chirps, due to the velocity of the target.

Two successive FFTs are applied to find this phase difference. Considering that M chirps are transmitted, each one sampled N times, a matrix $N \times M$ is constructed. The first FFTs are applied column-wise, which determine the range of a target (hence called Range FFTs). Afterwards, row-wise FFTs are performed to determine a target's radial velocity (hence called Doppler FFTs). Together, these FFTs create the Doppler spectrum. Both the in-phase and quadrature components are submitted to this process separately.

Therefore, the phase difference value is characterized as

$$\Delta\omega_v = 2\left(\frac{2\pi v T_c}{\lambda}\right), \quad (2.16)$$

where v is the radial velocity of a target and λ is the wavelength of the transmitted chirp.

Rearranging in order to the radial velocity leads to

$$v = \frac{\lambda \Delta\omega_v}{4\pi T_c}. \quad (2.17)$$

Due to the FFT properties, applying an FFT to a sequence of N samples can separate the angular frequencies by no more than $\frac{2\pi}{N}$ radians, or in other words,

$$\Delta\omega_v = \frac{2\pi}{N}. \quad (2.18)$$

The velocity resolution (that is, the minimum values in the scale of velocities) is given by equaling (2.14) to (2.16):

$$\Delta V = \frac{\lambda}{2NT_c}. \quad (2.19)$$

2.3 Classification Features

A feature is a "measurable property of the process being observed"[38]. In the Machine Learning (ML) context, features are characteristics derived from the dataset being analyzed. Often, the features are the inputs of the classification algorithms due to the underperformance associated to the raw data.

Different features explore different properties, sometimes in different domains. Below, a selection of features is described, separated in two categories: time-domain and frequency-domain.

2.3.1 Time-Domain Features

During the operation of the radar explained in section 2.2, the I/Q samples are retrieved from the different receiver channels (see Figure 2.1). These signals have not been submitted to any FFT, therefore still being in the time-domain. Several common features that can be calculated in this domain include:

1. **Peak to Peak Value:** simply described as the difference between the maximum and the minimum value of a signal $x(k)$:

$$PP = |\max(x(k)) - \min(x(k))|. \quad (2.20)$$

2. **N-th Raw Moment:** if a size m is fixed for the sample window, then the raw moments can be described as

$$\mu'_n = \frac{1}{m} \sum_{i=1}^m x_i^n. \quad (2.21)$$

3. **N-th Central Moment:** similarly, for a fixed window size of m , central moments are described as

$$\mu_n = \frac{1}{m} \sum_{i=1}^m (x_i - \mu'_1)^n, \quad (2.22)$$

where x_i is the value of each sample and μ'_1 is the 1st raw moment, also called the simple mean.

4. **Magnitude and Phase:** the magnitude and phase of the I/Q signals are also time-domain features, and are described as

$$M = \sqrt{I^2 + Q^2}, \quad (2.23)$$

$$\phi = \arctan\left(\frac{Q}{I}\right). \quad (2.24)$$

5. **Root Mean Square:** also called as the quadratic mean. The Root Mean Square (RMS) squares the signal before calculating the mean, in order to make all values positive. It is defined as

$$RMS = \sqrt{\frac{1}{n} \sum_i^n x(k)^2}. \quad (2.25)$$

6. **Discrete Cross-Correlation:** this feature measures the similarity between a signal x with shifted versions of a signal y . In addition, one of the signals can be a reference signal in order to extract specific characteristics. Cross-Correlation is defined as

$$\hat{r}_{xy}(k) = \frac{1}{M} \sum_{m=0}^{M-1} x(m)y^*(m+k), \quad (2.26)$$

where M is the length of the cross-correlation, and $x(k)$ and $y^*(k)$ respectively represent the amplitude and the conjugate amplitude of the two different signals.

2.3.2 Frequency-Domain Features

As mentioned in subsection 2.2.4, FFTs are performed in the received I/Q signals, from which frequency-domain features can be retrieved. Nonetheless, the window size needs to be carefully chosen to match the finite length of an FFT. If the window size fails to be a multiple of the signal period, then spectral leakage can occur, adding noise in the frequency domain. Therefore, there are multiple algorithms to choose a window size, each suitable for specific applications. For an FMCW radar, the Hamming window function is shown to achieve a better performance, due to having a bigger peak amplitude in the frequency domain [39].

In the following list, examples of frequency-domain features are given.

1. **Variance Trend:** splitting the FFT in L vectors of equal size, the variance (μ_2) of each vector is calculated. Afterwards, every vector subtracts its variance with the vector before it, summing all subtractions made, as shown in the equation below. Note that other central moments, such as the skewness (μ_3) or kurtosis (μ_4) give other type of information on the FFT.

$$\mu_{2_T} = \sum_{n=2}^L (|\mu_{2_n} - \mu_{2_{n-1}}|). \quad (2.27)$$

2. **Windowed Variance Trend:** similar to the previous feature, with the exception that the distance of the current vector is measured in relation to the FFT variance (and not the previous vector).

$$\mu_{2_D} = \sum_{n=1}^L (|\mu_2 - \mu_{2_n}|). \quad (2.28)$$

3. **Power Spectral Entropy (PSE):** this feature measures the uncertainty of the frequency-domain of the signal, that is, how spread the signal is in the frequency-domain. This is important as each range is associated to a beat frequency (see Figure 2.3). To calculate this feature (shown in (2.31)), the Power Spectral Density and its corresponding

distribution function need to be calculated, being computed as in (2.29) and (2.30), respectively.

$$\hat{P}(\omega_i) = \frac{1}{L} |X(\omega_i)|^2, \quad (2.29)$$

$$p_i = \frac{\hat{P}(\omega_i)}{\sum_{i=1}^L \hat{P}(\omega_i)}, \quad (2.30)$$

$$H = - \sum_{i=1}^L p_i \log p_i. \quad (2.31)$$

4. **Kullback-Leibler Distance (KLD):** it is a statistical distance which measures the dissimilarity between two probability functions defined on the same probability space. In the context of this work, it measures the dissimilarity between two power spectral density functions. This value is higher the bigger the difference between the functions, or zero if both are equal. It is defined as

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right). \quad (2.32)$$

5. **Spearman's Rank Correlation Coefficient:** given two different functions, this coefficient measures how well their correlation can be described with a monotone function. This value ranges from -1 to 1, being equal to zero when no monotonic relationship exists between the data. In this work, Spearman's Rank Correlation Coefficient can be calculated for two frequency spectra, measuring their similarity as follows

$$r_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}, \quad (2.33)$$

where d_i represents the distance between two ranks, and n is the number of observations.

2.3.3 Feature Engineering

Classification is often performed based on the extracted features of the dataset. Therefore, choosing the right features has various benefits, such as reducing the computational cost, reducing the noise from irrelevant variables, avoiding the curse of dimensionality and, overall, improving the system's performance [38].

This problem can be tackled by either selecting which features are important (eliminating the rest), or transforming the feature space into another that better describes the

problem. A combination of the two is also possible. In the next sections, several Feature Selection (FS) and Feature Extraction (FE) methods are presented.

2.3.3.1 Feature Selection

Feature Selection (FS) consists of discovering which features better describe the problem or distinguish different classes, eliminating irrelevant or redundant features. There are three possible ways to do this:

Filter Methods: filter methods are called so because they filter which features are passed to classification. They are based on ranking or scoring features according to their relevance. After scoring, a threshold can be applied to select which features are considered relevant.

One simple criteria function is the Pearson correlation coefficient, which is used to detect linear dependencies:

$$R(n) = \frac{cov(x_n, Y)}{\sqrt{var(x_n) \cdot var(Y)}}, \quad (2.34)$$

where x_n is the n -th variable and Y is a class label.

Another filter method is Mutual Information (MI), which scores how much a variable X depends on a variable Y or, in other words, if they are redundant. For two discrete variables, MI is described as

$$I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p_{(X,Y)}(x, y) \log \left(\frac{p_{(X,Y)}(x, y)}{p_X(x)p_Y(y)} \right), \quad (2.35)$$

where $p_{(X,Y)}$ is the joint probability mass function of X and Y , and p_X and p_Y are the marginal probability mass functions of X and Y respectively.

Filter methods often have the advantage of being computationally light. Nonetheless, depending on the algorithm used, the selected feature subset might still have some redundancy [38].

Wrapper Methods: wrapper methods are much greedier in their approach: out of all feature subsets from the original one, they intend to find the best, based on the performance of the classifier used. Naturally, the higher the dimensionality of the problem, the slower this approach would be, due to the number of possible subsets. To solve this problem, a searching algorithm selects which subsets are tested [38].

When compared to filter methods, wrapper methods achieve higher classifying accuracies, due to fact their subsets are evaluated based on the performance of specific classifiers [38]. However, the price is its increased computational cost.

Embedded Methods: embedded in the training process of the classifiers, these methods take advantage of the qualities of both filter and wrapper methods, having an in-between computational performance and accuracy [38]. As an example, features can be first selected using a filter method, with which a wrapper method then selects the best subset within these pre-selected features. Another algorithm is the Least Absolute Shrinkage and Selection Operator (LASSO), which is based on linear regression with L1 regularization. This means that polynomial coefficients of higher dimensions are penalized and shrunk to zero, reducing the model's complexity. The features whose coefficients are equal to zero at the end of the training process are discarded.

2.3.3.2 Feature Extraction - PCA Algorithm

Feature Selection (FS) methods rely on selecting the best subset of features and cutting out the irrelevant or redundant ones. On the other hand, Feature Extraction (FE) methods transform these features into more significant ones, usually reducing the complexity and dimensionality of the problem [40].

A very popular algorithm is the Principal Component Analysis (PCA) in which an orthogonal transformation is applied to the input variables. New features are generated as a linear combination of the original ones, while also being linearly uncorrelated. The PCA algorithm works as follows:

1. The first step is the normalization of the data. This ensures that all values are on the same scale, making possible a fair comparison between them. Assuming that each feature has m values, a $p \times m$ matrix is constructed, with p being the total number of features. For the k -th feature vector $1 \times m$, its normalization X_p^k is given by

$$X_p^k = \frac{X^k - \mu_k}{\sigma_k}, \quad (2.36)$$

where X^k is the k -th feature values vector.

2. Obtain the $p \times p$ covariance matrix C , which expresses the covariance value for every pair of features. The covariance between two variables expresses how related two variables are to each other.
3. Obtain the covariance matrix's eigendecomposition, which equates to calculating its eigenvalues and eigenvectors. Eigendecomposition is the transformation of a matrix into a canonical form, where it is represented by its eigenvectors (representative of its direction) and the eigenvalues (the magnitude of these directions), as given by the equation

$$CV_k = \lambda_k V_k, \quad (2.37)$$

where V_k represents the eigenvector of the k -th feature and λ_k the k -th feature eigenvalue.

The eigenvector having the higher eigenvalue is considered the first principal component. The rest of the order follows the same principal.

4. The next step is the transformation of the features into the new space representation as in

$$X_u = VX, \quad (2.38)$$

where X_u represents the matrix of the new features and V represents the $p \times p$ matrix of eigenvectors.

Dimension reduction can be performed by choosing the q most relevant features out of the p existent. As mentioned in the previous point, the eigenvalues can be sorted in order of relevance, with the highest value having the highest relevance. Higher relevance means that the feature better represents the original feature set. Therefore, similarly to (2.38), the selected transformed features q are given by

$$X_{uq} = V_q X. \quad (2.39)$$

In order to measure how much information is preserved when selecting the q most relevant features, a variance ratio can be calculated. Note that in a covariance matrix, the diagonal elements are representative of the variance. Hence, eigenvalues can be used to calculate this ratio, represented by

$$r_{var} = \frac{\sum_{i=1}^q \lambda_i}{\sum_{j=1}^p \lambda_j}. \quad (2.40)$$

Other variants of PCA exist, such as Independent Component Analysis (ICA), which prioritizes minimizing the statistical independence of variables instead of minimizing the covariance like PCA.

2.4 Deep Learning

2.4.1 Introduction to Artificial Neural Networks

To better understand the concepts intrinsic to DL, this section aims to explain the basics of feedforward neural networks, such as the functioning of neurons, dense layers, backpropagation algorithm and activation functions.

2.4.1.1 Artificial Neurons

Artificial neurons are the core units of Artificial Neural Networks (ANNs), and are inspired in the human brain. Biological neurons are composed of three elements: the dendrite, which receives signals from other neurons; the axon, which feeds a signal to other neurons; and the cell body itself. Similarly, artificial neurons have representatives of these elements: they receive inputs from other neurons, process them, and send a signal to other neurons. A scheme of an artificial neuron is illustrated in Figure 2.6.

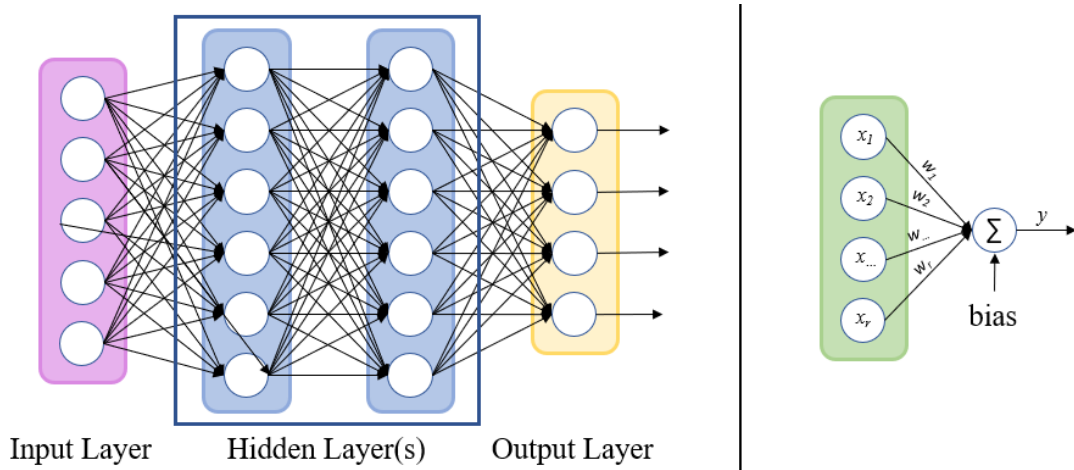


Figure 2.6: A fully connected neural network (on the left) and a scheme of an artificial neuron (on the right), adapted from [41].

An artificial neuron receives signals (i.e. values) from the previous r neurons. In order to measure the importance of each signal received, the neuron associates weights to each connection, represented by w_r . Accounting on these weights (and summing a scalar known as *bias*), the neuron calculates the output y , which is given by an activation function (see subsection 2.4.1.3). This output is then passed to the succeeding neurons.

2.4.1.2 Training an Artificial Network

Training an artificial neural network consists in adjusting the weights and the biases of the neurons present in every layer. This is achieved through an iterative process:

1. Separate the dataset in three: training, validation and testing. The training dataset is fed to the network and will be a part of the network training process that adjusts the weights and biases of the network. Validation and testing datasets are important to understand whether or not the network is overfitting these weights to the training data. Overfitting (and its counterpart underfitting) will further be discussed ahead;
2. Calculate the output of the neural network based on the current weights;

3. Calculate the error of the predicted output through a loss function. Note that in supervised learning the dataset is labeled. Typical loss functions include Mean Squared Error (MSE) and the Cross-Entropy;
4. Propagate this error backwards, so that the weights are adjusted. Note that the main objective is to find how the error changes depending on the weights, that is, the derivative of error with respect to the weights;
5. Repeat the process from 2. until either the number of maximum epochs (iterations) is reached, or until the loss function returns a value small enough.

During the training process, the values given by the loss function and the accuracy of the network at each epoch are saved. This is in order to plot a graph similar to the ones shown in Figure 2.7. At the end of the training, the weights and biases of the network have been adjusted so that the latter best describes the training dataset, achieved by minimizing the value provided by the loss function and maximizing the accuracy of the output prediction.

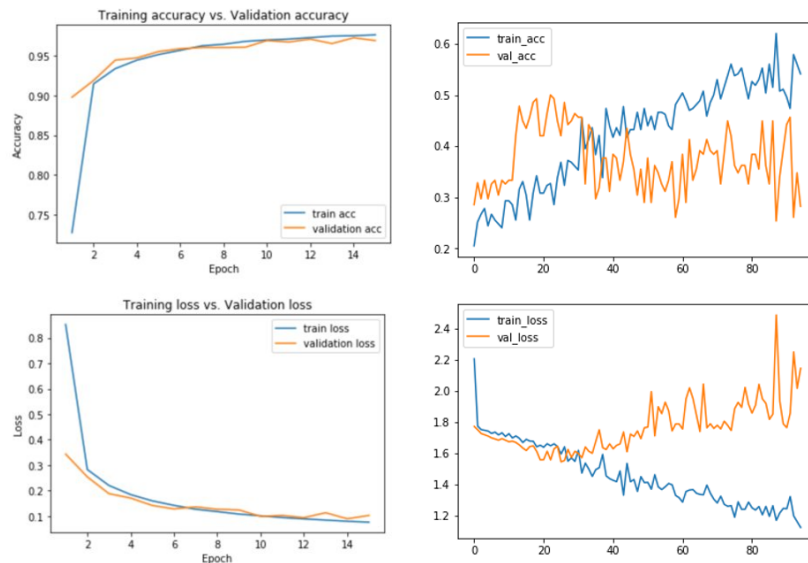


Figure 2.7: Examples of accuracy and loss plots of the training and validation per epoch. The plots on the left show a convergent loss error network, in which the accuracy is maximized and the loss is minimized for both the training and validation datasets. On the contrary, the plots on the right achieve convergence only for the training dataset, and it is an example of overfitting.

Nonetheless, perfectly describing the training dataset often leads to what is called **overfitting** (see Figure 2.7). Overfitting means that the network can achieve high accuracy and low loss for the training dataset, but these properties are not shared with the validation and testing datasets. The opposite case is known as **underfitting**, in which the network weights have not been adjusted enough to describe either of the datasets.

2.4.1.3 Activation Functions

Activation functions are responsible for calculating the output value y of a neuron. Generally, $y(x)$ is a function of the input signals x_i and their respective weights w_i , as well as a scalar bias b as follows

$$y = f\left(\sum w_i x_i + b\right). \quad (2.41)$$

Activation functions must meet the requirements needed for describing complex datasets, in which a simple linear function isn't enough. With that being said, non-linearity is a very important property for these functions that allows a neural network to map the inputs to the corresponding output, even if they are not linearly dependent on each other [42]. This makes choosing the correct activation function a key element in neural networks. Below, a list of activation functions and some of their properties is presented.

Sigmoid: Sigmoid is a highly used function for being non-linear. The outputs of this function range from 0 to 1 in an S shape. Its plot is shown in Figure 2.8 and can be calculated as shown in (2.42). For high absolute values, sigmoid is nearly invariant: whether the input is 10 or 1000, the result is very close to 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (2.42)$$

Sigmoid has some known problems [43]. First, its output is not centered on 0, and all values are positive. Due to its nearly invariance when saturated, this function causes the gradients in the backpropagation algorithm to be nearly equal to 0. In turn, this causes the weights of the network to not update on each iteration, which leads to poor accuracy. This problem is known as the **vanishing gradient problem**. On the other hand, with certain neural network architectures such as LSTMs, sigmoid leads to the **exploding gradient problem**, in which gradients get higher and higher values.

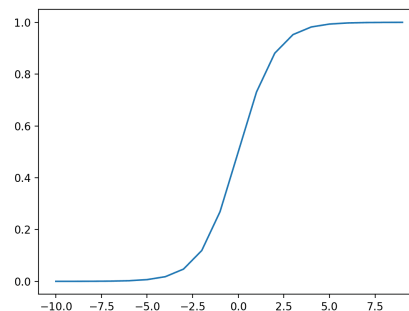


Figure 2.8: Plot of the sigmoid activation function.

Hyperbolic Tangent: This function is extremely similar to the sigmoid, but has shown to be better in hidden layers [43]. Unlike the sigmoid, $\tanh$ is centered around 0, and outputs values with both signs, ranging from -1 to 1. Besides this, it carries on the same problems as the sigmoid relative to the vanishing gradient problem. Its plot can be seen in Figure 2.9 and its equation is given by

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (2.43)$$

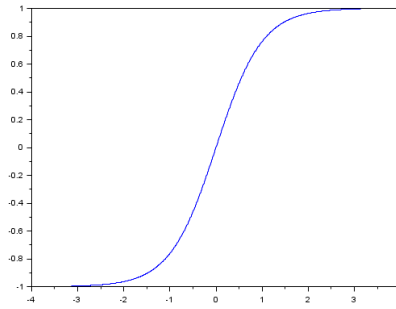


Figure 2.9: Plot of the hyperbolic tangent activation function.

Rectified Linear Unit: Rectified Linear Units, or ReLU, are a type of piecewise linear functions. This is especially interesting because they appear to be linear, while in fact they are not. Therefore, these functions are able to describe the non-linear behaviour that neural networks need while still exhibiting advantages associated with the simplicity of linear functions. ReLU's plot is shown in Figure 2.10 and is calculated as

$$\sigma(x) = \max\{0, x\}. \quad (2.44)$$

ReLU has advantages over classical activation functions like the sigmoid or the hyperbolic tangent [43]. Specifically, it solves the vanishing gradient problem, as the slope is 1 for all positive values. Furthermore, it represents what is called the True 0: for negative values, the derivative is 0 and the weights of the previous connections are not updated, which leads to a faster convergence. Nonetheless, it is still susceptible to the exploding gradient problem, as there is no top limit on the values.

There are a number of variants to ReLU, each having serving a specific purpose. For instances, Parametric ReLU (or PReLU) modifies the slope for negative values, which is demonstrated to reduce overfitting [43]. Similarly, Leaky ReLU changes the slope for negative values to a very small one.

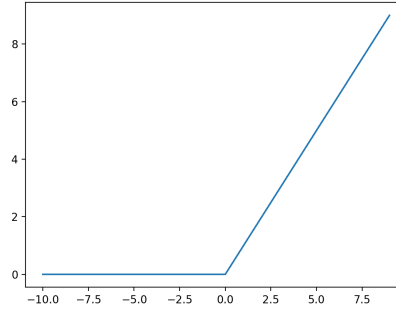


Figure 2.10: Plot of the ReLU activation function.

Softmax: Softmax is a special function which is usually chosen for the output layers. Similar to sigmoid, it outputs values ranging from 0 to 1. However, unlike sigmoid, the sum of the outputs of all neurons in a layer using a softmax activation function will be 1. This means that a probabilistic approach can be taken with softmax, effectively distinguishing between classes and justifying their use in these cases. Softmax at a neuron i is given by

$$\sigma(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}. \quad (2.45)$$

2.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of ANNs designed to understand complex patterns in the input data. Traditional ANNs with dense layers do not scale well with high dimensional input [44, 45]. This problem is further extended the higher the number of dense layers. Albawi et al. [44] describe a small network with two fully connected layers, each having two neurons. For a $32 \times 32 \times 3$ input image, the number of weight connections exceeds 3 million, which is clearly computationally heavy for a small network.

CNNs approach this problem by adding one or more convolutional layers. Their general architecture is represented in Figure 2.11. The main idea is that instead of having full connections between all layers, segments of the image are connected to only a few neurons. In turn, these neurons share connections with only a few from the next layer. This drastically lowers the number of weight connections needed: following the example given before, a CNN can describe the problem with 75 connections, instead of 3 million [44].

The convolutional operation performed by the neurons is given by

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n), \quad (2.46)$$

where I is the input image, K is a kernel of size $m \times n$ and (i, j) are the coordinates of the convoluted image.

In [45], the authors explain three properties of convolution: **sparse interactions**, **parameter sharing** and **equivariant representations**. Sparse interactions is a way of reducing the total number of parameters by using a kernel function of reduced dimension. These kernels are responsible for detecting features such as edges. Parameter sharing means that although each set of neurons is tied to a specific region of the image and are responsible for learning features about that region, all information learned (i.e. the kernels) is shared. Lastly, equivariance is a very important property for images: in short, objects, edges or other features in an image can be translated (shifted to other coordinates) and the network will still be able to identify them. The weights of the kernels, which represent image features, are learned through backpropagation.

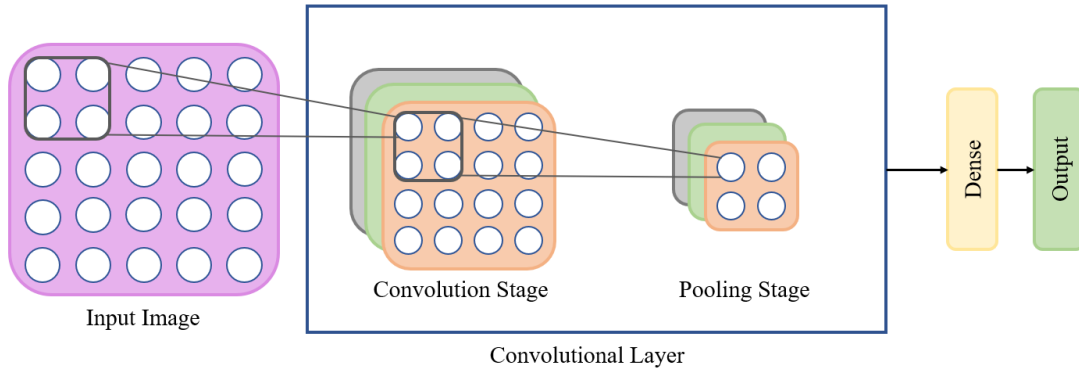


Figure 2.11: General architecture of a Convolutional Neural Network, adapted from [41].

Subsequently, the next layer, called the Pooling layer, reduces the dimensionality of the input image. This helps in not only decreasing the complexity of the problem, but also representing the problem with fewer and fewer features. Furthermore, the authors in [45] also state that pooling improves the invariance of the network: this means that a small change in the input (e.g. a translation) is likely to produce the same output after pooling. Commonly, either average pooling or maximum pooling are applied: for a pixel (i, j) , the maximum value of the 3×3 neighbours is chosen, for example. In general, maximum pooling outperforms average pooling [46]. Lastly, pooling can help in transforming input data to a specific dimension, in order to be fed to a dense layer and its respective output layer.

2.4.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is special network in which a cell receives its previous state(s) as input through a feedback mechanism, as seen in Figure 2.12. This way, information from previous states of the network is learned, making RNNs a very good choice for analyzing sequential data like speech [47] or hand writing [48]. The amount

of states from which the current state receives information from can be customized. Furthermore, unlike conventional networks, parameters are shared between all elements from a sequence. Consequently, this generalizes the model trained, preparing it for sequences not present in the training, as well as a better response if sequences are present in different time steps [49].

For a sequence of values with size T where $x^{(t)}$ represents the input at time t , a RNN cell behaviour can be described by the equations

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}, \quad (2.47)$$

$$h^{(t)} = g(a^{(t)}), \quad (2.48)$$

$$o^{(t)} = c + Vh^{(t)}, \quad (2.49)$$

$$y^{(t)} = g(o^{(t)}), \quad (2.50)$$

where $h^{(t)}$ is the current hidden vector state, $y^{(t)}$ is the output vector, g is an activation function, b and c and bias vectors and U , W and V are the learnt parameters.

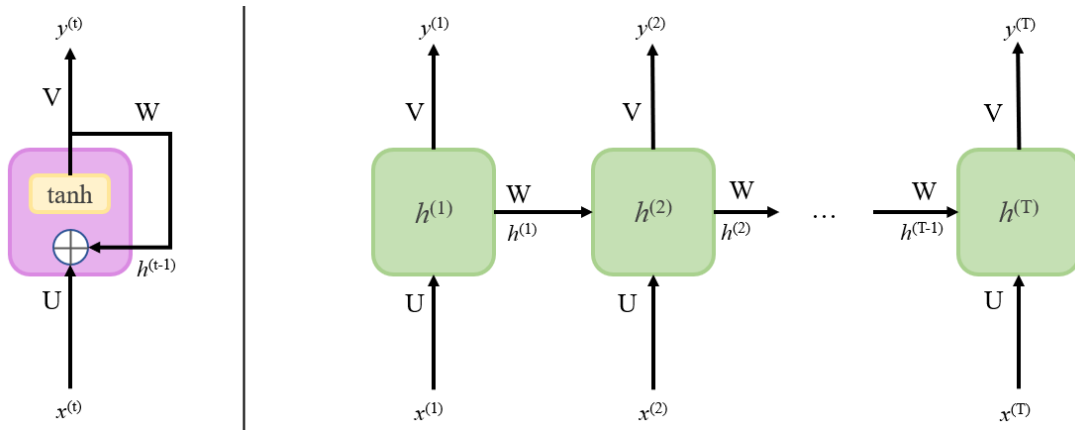


Figure 2.12: General structure of a folded RNN cell (on the left) and an unfolded RNN cell (on the right), adapted from [49].

Typically, the hidden node activation function is the hyperbolic tangent, whereas softmax is used for the output layer. Alternatively, the hidden nodes can also use the ReLU activation function. In any case, RNNs with long sequential lengths are prone to either the vanishing or the exploding gradient problem [50]. Other variants of RNNs, like LSTMs or GRUs, have been vastly used to overcome this problem. These will be discussed in the sections below.

2.4.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) rises as a solution to the vanishing and exploding gradient problem that the RNNs presented above pose. In short, LSTMs introduce a control mechanism, called the cell state $c^{(t)}$. The cell state represents the network's memory, and

is controlled by three gates: the forget, the input and the output, all of which are activated by a sigmoid function. Their general structure is presented in Figure 2.13.

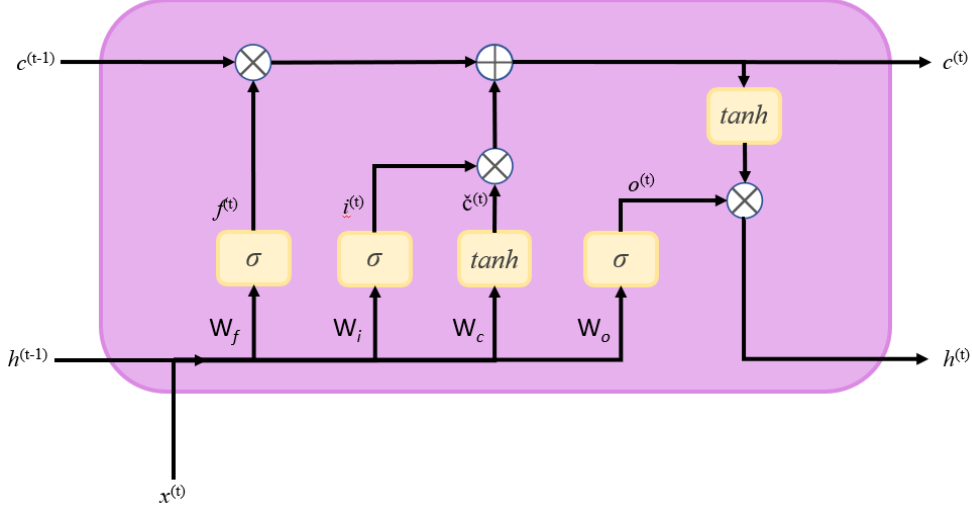


Figure 2.13: General structure of a LSTM cell, adapted from [49].

The input gate i^t manages which information should be added to the current state cell. Its output is given by

$$i^t = \sigma(W_i \cdot [x^{(t)}, h^{(t-1)}] + b_i), \quad (2.51)$$

where $[x^{(t)}, h^{(t-1)}]$ denotes the concatenation of the input vector $x^{(t)}$ and the previous hidden state $h^{(t-1)}$, and W_i and b_i are the learnt weights and biases parameters.

The forget gate f^t is responsible for selecting which information is relevant by discarding (or forgetting) that which isn't. The gate's output is described as

$$f^t = \sigma(W_f \cdot [x^{(t)}, h^{(t-1)}] + b_f), \quad (2.52)$$

where W_f and b_f are the learnt weights and biases parameters.

Likely, candidate values denoted by $\tilde{c}^{(t)}$ carry information that may be added to the cell state. Therefore, they are multiplied with the input gate vector as in

$$\tilde{c}^{(t)} = \tanh(W_c \cdot [x^{(t)}, h^{(t-1)}] + b_c), \quad (2.53)$$

with W_c and b_c being the learnt weights and biases parameters.

Summarizing, the cell state update is characterized as

$$c^{(t)} = f^t * c^{(t-1)} + i^t * \tilde{c}^{(t)}, \quad (2.54)$$

where $*$ denotes the Hadamard product.

The output gate o^t controls the information that is passed from the cell state to the hidden state as expressed by

$$o^t = \sigma(W_o \cdot [x^{(t)}, h^{(t-1)}] + b_o), \quad (2.55)$$

with W_o and b_o being the learnt weights and biases.

Lastly, the hidden state is given by

$$h^{(t)} = o^t * \tanh(c^{(t)}). \quad (2.56)$$

2.4.3.2 Gated Recurrent Units

Gated Recurrent Units (GRU) are a variant of LSTMs. The major difference in GRUs are the absence of the cell state and the existence of only two gates, name the update and reset. Their general structure can be seen in Figure 2.14.

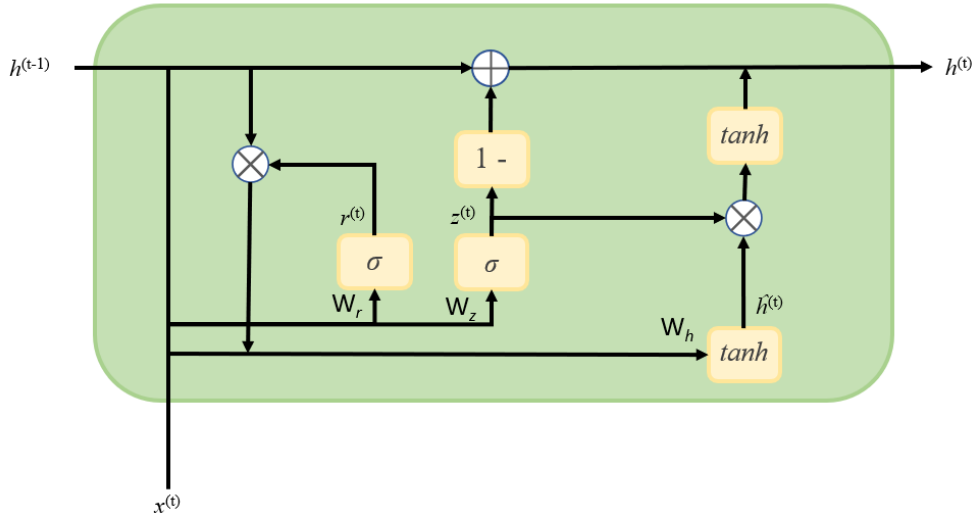


Figure 2.14: General structure of a GRU cell, adapted from [49].

The reset gate r^t is responsible for selecting the information to be discarded from the previous hidden state, given by

$$r^t = \sigma(W_r \cdot [h^{(t-1)}, x^{(t)}] + b_r), \quad (2.57)$$

where the current input vector $x^{(t)}$ and the hidden state at the prior instance $h^{(t-1)}$ are concatenated, and W_r and b_r being the learnt weights and biases.

The update gate z^t controls the information that will pass to the hidden state. It is similar to a mix of both the forget and input layers from LSTM cells. Their expression is calculated as

$$z^t = \sigma(W_z \cdot [h^{(t-1)}, x^{(t)}] + b_z), \quad (2.58)$$

where W_z and b_z are the learnt weights and biases.

The candidate values $\tilde{h}^{(t)}$ are expressed as

$$\tilde{h}^{(t)} = \tanh(W_h \cdot [h^{(t-1)}, x^{(t)}] + b_h), \quad (2.59)$$

where W_h and b_h are the learnt weights and biases.

Lastly, the hidden state is calculated as

$$h^{(t)} = (1 - z^{(t)}) * h^{(t-1)} + z^{(t)} * \tilde{h}^{(t)}, \quad (2.60)$$

where $*$ denotes the Hadamard product.

J. Chung et. al [51] note that GRU units have comparable performance with LSTM units. In fact, they describe there is no way to predict which architecture is better at which task. Nonetheless, GRU have shown to be faster due to their simplicity. Therefore, in situations where LSTM performs better, there is a certain trade-off between computational cost and achieved accuracy.

2.4.4 Generative Adversarial Networks

Generative Adversarial Networks, or GANs, are models capable of generating new data similar to the one they receive as input. They can perceive the characteristics that define classes in a dataset, constructing distributions which can then be sampled to generate new points of data. In contrast, classic neural networks are considered discriminative models, as they learn to discriminate the input data by associating it with one of the available classes. More formally, if X and Y are respectively the input and the class labels, the objective of generative models is to find the joint probability $p(X, Y)$ (that is, associating the input and the labels), whereas the discriminative models' is to find the conditional probability $p(Y|X)$ (that is, the input is classified as the label Y with the highest probability). A scheme of a GAN is shown in Figure 2.15.

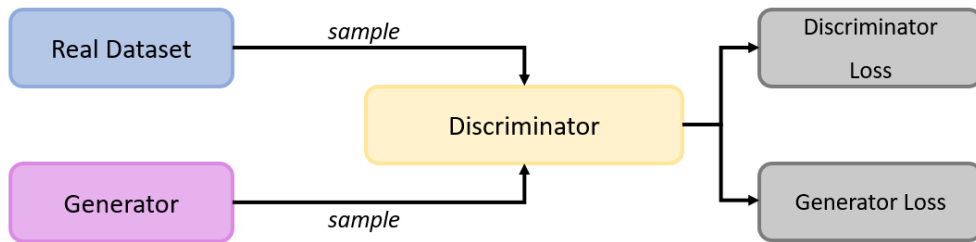


Figure 2.15: Structure of a GAN, adapted from [52].

There are two main components in a GAN, both of them being neural networks: a generator, responsible for generating new (or "fake") points of data, and a discriminator, whose goal is identifying whether or not its input is a real or a fake sample of data. As shown in Figure 2.15, the discriminator can receive a sample from either the real dataset or from the output of the generator. Afterwards, the discriminator classifies the input as either real or fake, penalizing the generator if it successfully recognizes the data point as fake. Each network is trained separately: for a set number of epochs, the discriminator

loss function backpropagates and adjusts the weights and biases of the discriminator; afterwards, the equivalent for the generator loss function is repeated. This process is then repeated alternatively. As the training progresses, the generator outputs better and better results, and the discriminator's accuracy approaches 50%, as it can no longer distinguish between real and fake data.

GANs are known to have certain problems during training [53]. Specifically, the first major problem is that GANs can end up generating a very small number of samples. The second problem is the oscillation of the generator and discriminator during the training, which can never converge. Notwithstanding, GANs provide very useful tools for a vast number of areas, such as image generation [54] and image super resolution [55].

2.4.5 Autoencoders

Autoencoders (AEs) are neural networks whose main purpose is unsupervised feature extraction. They are composed of an input layer, also called the *encoder*, one or more hidden layers, and an output layer known as the *decoder*. This architecture can be seen in Figure 2.16. The main goal of autoencoders is learning how to mimic the input at the output. This may not seem particularly interesting at first, as most neural networks want to predict or classify a given input instead of mimicking it. The trick for feature extraction lies in the latent state of autoencoders: as hidden layers have fewer and fewer dimensions, the network has to learn how to represent the input data with fewer features, while still reconstructing an acceptable copy of it. Compared with other feature extraction solutions, such as Principal Component Analysis (PCA), features provided by autoencoders can lead to an improved classifying accuracy [56]. Nonetheless, autoencoders training times can be much higher [57].

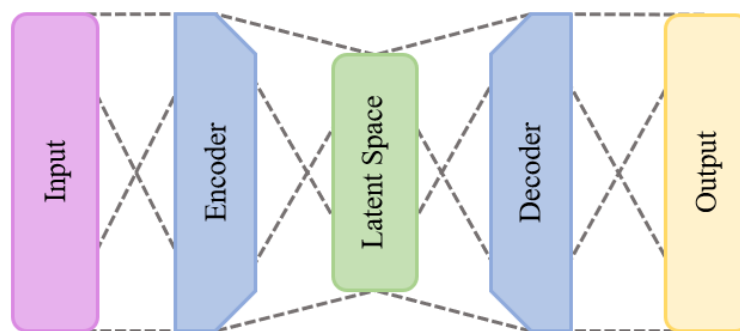


Figure 2.16: General architecture of an Autoencoder, adapted from [58].

Variants to autoencoders exist. Denoising Autoencoders (DAEs) are similar to regular AEs. During their training, noise is added to the input, but not to the output. This effectively forces the autoencoder to learn how to denoise the input. Another variant are the Variational Autoencoders (VAEs), represented in Figure 2.17. In these, the encoder infers statistic properties, such as the mean or standard deviation, of the input data. With

this information, a normal distribution can be defined, and points from that distribution are fed to the decoder. Note that unlike most neural networks, VAEs do not use samples from the original dataset; rather, they use samples from a normal distribution. Hence, VAEs are considered generative models. Compared to regular AEs, VAEs offer better data regularization and less overfitting [59].

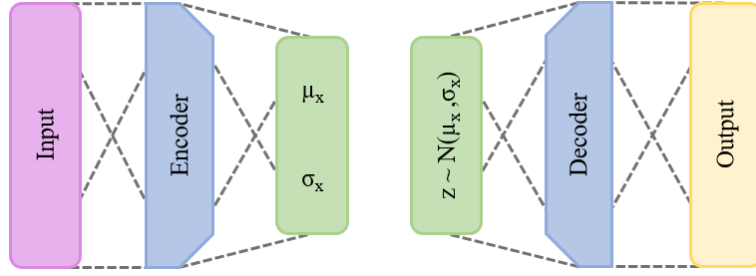


Figure 2.17: General architecture of a Variational Autoencoder, adapted from [58].

2.5 State of the Art in RF Sensing

FMCW radars have been in the background when it comes to context-awareness applications. Mainly, this is due to the fact that they are often complex solutions. Nonetheless, radars offer advantages when compared to other sensing solutions (as discussed in section 2.1.1). In turn, radars have been a topic of research in the recent years in a wide variety of areas for context-awareness, such as vital signs recovery and human-motion gesture classification. In addition, the choice of the feature set used to describe the problem not only helps in reducing the complexity of the problem, but also reduce its computational cost, improving its run-time speed, and are a topic of research as well. Below, several works in this area and their presented solutions are briefly discussed.

In 2019, Park et al. [60] describe a way of detecting the vital signs of the lungs and the heart through RF sensors, in order to monitor the status of a driver in a vehicle. Should the system detect a sleeping driver, for example, certain measures can be taken. Although RF sensors were used, the authors point out that FMCW radars would be able to achieve vital signs detection as well. However, they chose not to use them, as they are complex, costly, and suffer from noise issues. In another work, Wagner et al. [61] came up with a threshold-less solution for interference cancellation. They describe their solution which does not require parameter configuration, unlike other algorithms. This could potentially solve some of the problems pointed by Park's team, allowing FMCW radars to perform successfully in the place of RF sensors.

Another solution for vital signs waveform recovery was presented by Chen's team [7] in 2021, to which they called MoVi-Fi. It is known that radars rely on measuring the distance to a target for RF sensing; naturally, the interference of body movements can be challenging to overcome in order to recover the vital signs. MoVi-Fi is able to recover

vital signs even under "severe body movement" by relying on *deep contrastive learning*, a self-supervised DL technique. This way, the system is able to distinguish the different movements in a signal, e.g. the movements caused by respiration from the ones caused by walking on a treadmill. Furthermore, MoVi-Fi is software-implemented, meaning that it can operate independently from the hardware technology used, as long as it has sufficient bandwidth and the upper front of the target is turned to the radar. Consequently, both UWB and FMCW radars can be utilized for vital signs monitoring.

In [30], a FMCW radar based system was used to perform object classification at a medium range of 25 m. Afterwards, several features were extracted from the FFT plot and were fed to two different light-weighted ML algorithms: a Logistic Regression and a Naive Bayes, with the former having better accuracy results ranging from 86% - 100%, depending on the object. The authors mention that, for long range target detection, the features extracted from the FFT plot are not enough to achieve good results, and suggest the use of micro Doppler map features.

In human-motion context-awareness, several papers have been published. In [62], a FMCW radar is used to capture seven gestures of a driver in a vehicle (e.g. approaching the steering wheel or rotating clockwise). They use features manually selected from the micro Doppler signal, which are fed to a KNN classifier. This classifier has accuracies ranging from 63% - 99%, which can be justified for the relatively simple feature engineering solution applied (i.e., manual selection of features).

Similarly, Ding et al. [29] explore a FMCW radar solution for human motion context-awareness, in which six motions need to be identified, including falling, walking and jumping. The authors propose a Dynamic range-Doppler trajectory (DRDT) method for selecting meaningful information from conventional Doppler frames. This method is capable of describing human motions in real-time. Furthermore, through a peak search method, sequential human motions present in a DRDT map can be separated and identified simultaneously. Afterwards, 28 features from these maps are calculated and fed to a KNN classifier. Single motion classification achieved a 95% accuracy, while continuous motion classification accuracy averaged around 91%.

Additionally, several features are proposed in [63] to perform classification on a RF sensing solution for human gesture recognition. A neural network with one hidden layer was used to perform classification of these gestures over a manually selected subset of features. Overall, the system is able to fulfil RTS applications requirements, like low memory usage and fast run time, while still maintaining a high accuracy of 96%.

In [64], Maitre et al. used a UWB radar in order to detect human fall motion. Movement of any sort can be seen as a time-series problem; human fall motion is no exception. The authors preprocessed the data, transforming the scattering matrix provided by the radar into a time series of frames, each with a normalized amplitude. Consequently, this time series is fed to a neural network composed of a 1D CNN followed by two LSTM layers and a fully connected layer. This CNN-LSTM architecture is better at learning features than a pure LSTM one due to the CNN layers, while still learning time-series

patterns by maintaining the LSTM layers. In addition, it is also faster in training (100 minutes vs 2 minutes and 30 seconds) and shows better accuracy results (76% - 89% vs around 90%), which are the reasons the authors preferred the CNN-LSTM architecture.

Lastly, in [65], Shrestha et al. proposed a human activity context-awareness solution based on FMCW radars. They compared two different classifiers (a classic unidirectional LSTM network and a Bi-LSTM network) and two different inputs (range-domain data provided from the radar vs Doppler-domain data). They concluded that Bi-LSTM networks outperform unidirectional LSTM, as the former's output is a function of both on the past and future inputs, unlike the latter, in which the output is only a function of past inputs. Furthermore, the networks performed better when receiving input data from the Doppler-domain. In conclusion, the proposed Bi-LSTM classifier achieved over 90% accuracy.

EXPERIMENTAL SETUP, SCENARIOS AND DATASETS

3.1 Experimental Setup

3.1.1 Radar Description

To acquire and classify data, a complex-baseband FMCW radar device was used, with similar core functioning and architecture to the FMCW radars described in section 2.2.2. Specifically, a Texas Instrument’s (TI) IWR1843 radar chip [66] was used, integrated in the IWR1843BOOST Evaluation Module [67], presented in Figure 3.1. The experimental setup is pictured in Figure 3.2.

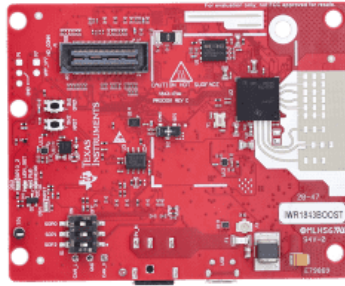


Figure 3.1: Front view picture of the IWR1843BOOST Evaluation Module used in this work, taken from [67].

The module is composed of transceivers which can operate between the 76-81 GHz band. In more detail, it contains 3 transmitting (TX) antennas and 4 receiving (RX) antennas. This means that each antenna transmits a chirp (as described in subsection 2.2.3), followed by a longer pause period (as shown in Figure 3.3). This behaviour is necessary to obtain Doppler information, hence each group of three chirps is denoted as Doppler chirp.

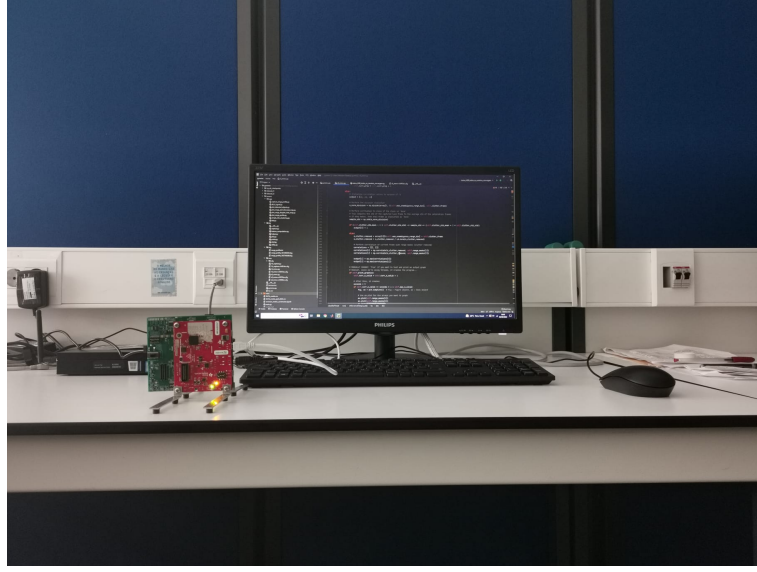


Figure 3.2: Photo of the experimental setup in room R.1.

After transmitting each Doppler chirp, the radar listens for echoes through its 4 RX antennas (for simplicity reasons, only a single received signal is represented in Figure 3.3). In order to construct a frame, the radar transmits 16 Doppler chirps sequentially. For every Doppler chirp, the ADCs in each RX channel collect raw data from I/Q samples, grouping them in a raw data matrix.

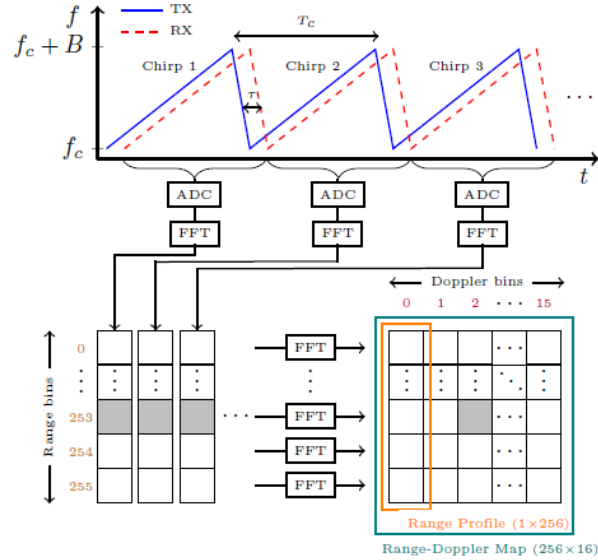


Figure 3.3: Range-Doppler Map (RDM) processing chain.

With the raw data matrix constructed, it is now possible to compute the FFT algorithm in order to find the range and radial velocity of targets, as explained in subsection 2.2.2. First, column-wise FFTs are applied to the matrix, which are used to calculate the

distance of targets. Hence, each row represents a **range bin**, associated to a specific range. Consecutively, row-wise FFTs are applied to this new matrix and their log-magnitude is computed. This results in another matrix, in which each row still represents a range bin, and each column represents a **Doppler bin**, containing information about a specific Doppler shift. Finally, the mean log-magnitude of each RX antenna matrix is calculated, resulting in the final frame, named Range-Doppler Map (RDM). The RDM is also represented in Figure 3.3 as the blue square. The orange square in the same Figure denotes the Range Profile, since a Doppler shift of 0 represents static targets for the duration of the Doppler chirp.

3.1.2 Radar Configuration

The radar system can be configured to better suit specific applications. These configurations are relative to either each of the individual chirps or to the Doppler chirps. To exemplify, changing the chirp duration can increase or decrease the range resolution.

Texas Instruments (TI) provides a tool for configuring these parameters, known as *mmWave Demo Visualizer* [68], depicted in Figure 3.4. Instead of manually configuring each of the parameters, *mmWave Demo Visualizer* allows the user to select the final conditions in which the radar is going to be used, such as range resolution and maximum unambiguous range, and reverse engineers them to determine the chirps' parameters. These can then be saved and loaded to the radar.

Figure 3.4: *mmWave Demo Visualizer* v3.5.0, available at [68].

The parameters used in this work are shown in Table 3.1. These are used to acquire the datasets mentioned in the next sections, and are used in the prototype developed and detailed in Chapter 5.

Parameter	Value
Range Bins	256
Doppler Bins	16
Chirps per Frame	48
ADC Sample Rate	2.727 MHz
Chirp Duration (T_C)	66.67 μ s
Chirp Slope	6 MHz / μ s
Frame Period	250 ms
Bandwidth (B)	4 GHz
Start Frequency (f_c)	77 GHz
Range Resolution	0.043 m
Maximum Unambiguous Range	6 m
Maximum Radial Velocity	1 m/s
Velocity Resolution	0.13 m/s

Table 3.1: Radar parameters used in this work.

3.2 Classification Scenarios

In agreement with the objectives set for this thesis, the main goal is to identify the action performed by an individual in front of the radar. For this reason, two different scenarios were set to be classified by the radar system. In both scenarios, an individual is placed in front of the radar and performs a specific action. Additionally, a third scenario was added that represents the absence of a target in front of the system. These scenarios are presented in Table 3.2.

Label	Scenario
0	None
1	Standing
2	Prostrated

Table 3.2: Classification scenarios and respective description.

In the *None* scenario, no targets are placed in front of the radar, which translates into reading the static objects present in a room. In the *Standing* scenario, an individual is placed in front of the radar, with their arms on the side, standing still in front of it. Lastly, in the *Prostrated* scenario, the individual bends their back, being almost parallel with the floor. A picture of each scenario alongside its representation on the range array captured by the system is depicted in Figure 3.5. The X axis represents the range bins: the higher the bin, the higher the distance to the radar. In turn, the Y axis represents the intensity

of the received signal, in which high intensity represents the presence of a target at that distance.

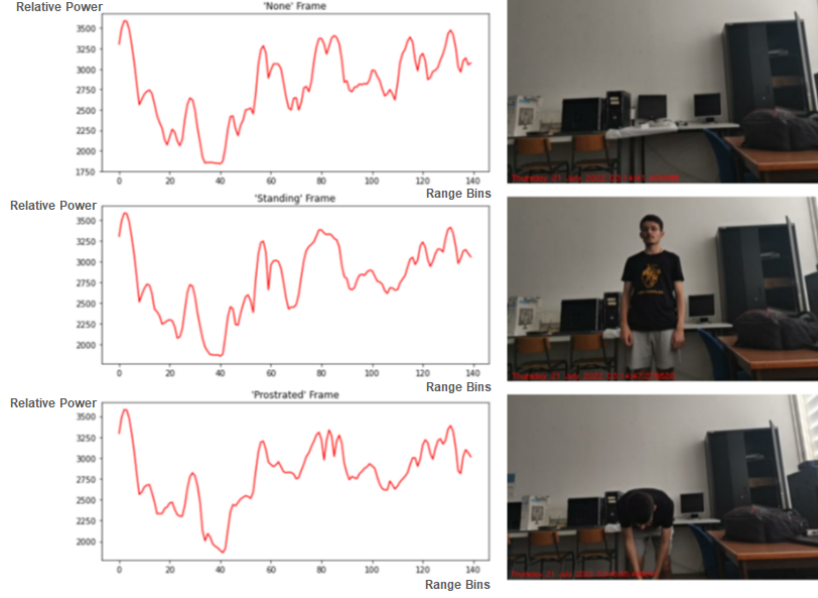


Figure 3.5: Visual representation of classification scenarios, alongside a range array describing them.

3.3 General Methodology and Dataset Description

3.3.1 General Methodology

In order to understand the reasons behind the opted structure of the datasets, it is convenient to introduce the basic idea behind the system's classifier. In addition, this chapter introduces the terms adopted throughout the rest of this thesis, giving a first understanding of these concepts.

The basic idea of the classifier is to correlate patterns representative of each class with a captured live frame. The higher a correlation with a pattern, the more likely it is that the captured frame is of said class.

To do so, we characterize the datasets in offline and retrieve statistical properties representative of each class. Specifically, the idea is to define the pattern that describes a class and use it as a tool to compare to another frame. This pattern is denominated as a **mask**. For every class label l , there can be a mask representative of it, denominated M_l , represented as a range array.

However, the radar system also records information of the static background in these datasets. Therefore, if no processing is done on the datasets, these masks will have representative information of the room's background, denominated *clutter*, and would only serve to classify other frames captured in the same environment. The objective is to

eliminate the room's clutter from these masks, so they can represent a class independently of the room background they were captured in.

In order to achieve this goal, we propose what is called the **calibration phase**, where the system records information of the background with no human targets present on its sights. This information serves to define the **clutter frame**, denominated as $F_{clutter}$, representative of the background clutter of a room. This is used to eliminate the presence of the background clutter from the masks. Consequently, each mask M_l defined from any set of labeled frames is considered to represent the class, independently on the room it was captured in.

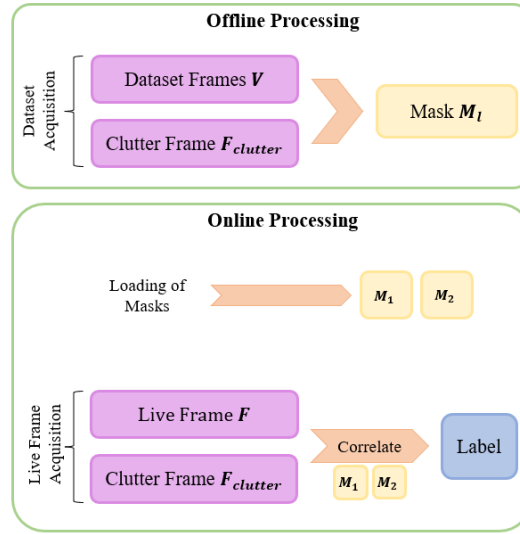


Figure 3.6: General methodology scheme representation of the developed classifier.

The basic methodology is represented in the scheme shown in Figure 3.6. The first phase includes the acquisition of datasets (named **dataset frames** from here onward, and represented as V_l for every class label l) and their respective clutter frame $F_{clutter}$. From these, a mask M_l can be defined for every class. The second phase is the live classifier, in which the masks M_l can be loaded and used to correlate with the live captured frames. The highest scoring class mask determines the system's classifier output.

3.3.2 Dataset Acquisition and Description

Dataset acquisition is the process of storing information provided by the radar system. This is a core step in studying the different possibilities for classification of the scenarios described in the previous section. Consequently, each dataset is composed of:

1. **Dataset Frames V** : during acquisition, each frame provided by the radar system is represented in a Range-Doppler Map (RDM), from which the range array is extracted and saved as dataset frames. It is worth mentioning that only range information is saved, and the Doppler information is not used in this work, since the

main goal is to classify static contexts. Every frame V_n is labeled manually. This is important since supervised ML algorithms require an *actual label*;

2. **Dataset Clutter** $F_{clutter}$: clutter (background noise of a room) associated with the dataset frames V . This information is obtained from the calibration phase, and is used to define the classification masks and perform classification, as explained later in Chapter 4;
3. **Prototype Classification**: the prototype classifier can provide a file with the classification results given by the prototype for each frame. This is used as the *predicted label* in Sequential Classification training, as explained later in section 4.3;
4. **Image Frames**: this directory contains images taken from a video camera, each associated to a frame, captured at the same time. While it is not used in any way for the classifier, it is an auxiliary function to help label frames manually.

To acquire the dataset frames, an open project written in Python, known as "Open Radar", serves as the starting point [69]. This script provides the basic user interface to communicate with the radar system. Furthermore, it receives data from the system through one or more frames at a time, which can then be processed and/or saved on a file. For this thesis, a new frame is sampled and received every 250 ms, composed of a Range-Doppler Map (RDM) (see section 3.1 for more details). Considering that all actions to be classified are static, Doppler bins from the RDM are ignored, and classification is performed solely based on the Range bin.

The conditions in which datasets are acquired can affect the classification results. If all datasets were to be obtained under the same conditions, then classification in other circumstances might become problematic, due to the classifier being possibly biased on the acquisition conditions. These include, but are not limited to, individuals, classification target distances and room backgrounds (i.e., the static objects in a room). Therefore, a total of three individuals, three classification distances and two backgrounds were used to construct dataset groups. Table 3.3 explains under which conditions each group was obtained. In turn, Table 3.4 shows the number of files for each dataset group, where each file contains between 50 and 100 sample frames, resulting in the label count described on Table 3.5. Individuals I.1, I.2 and I.3 are 165 cm, 170 cm and 185 cm tall respectively. A brief scheme for rooms R.1 and R.2 is presented in Figure 3.7.

-	Dataset Group 1	Dataset Group 2
Room Background	R.1	R.2
Individuals	I.1 and I.2	I.1 and I.3
Target Distances	1/2/3 meters	1/2 meters

Table 3.3: Dataset conditions description.

Number of Files / -	Dataset Group 1	Dataset Group 2
None	3	3
Standing	13	14
Prostrated	7	15
Total	23	32

Table 3.4: Number of Files per Dataset Group.

Label	Number of Frames
None	540
Standing	2680
Prostrated	1820

Table 3.5: Dataset label count.

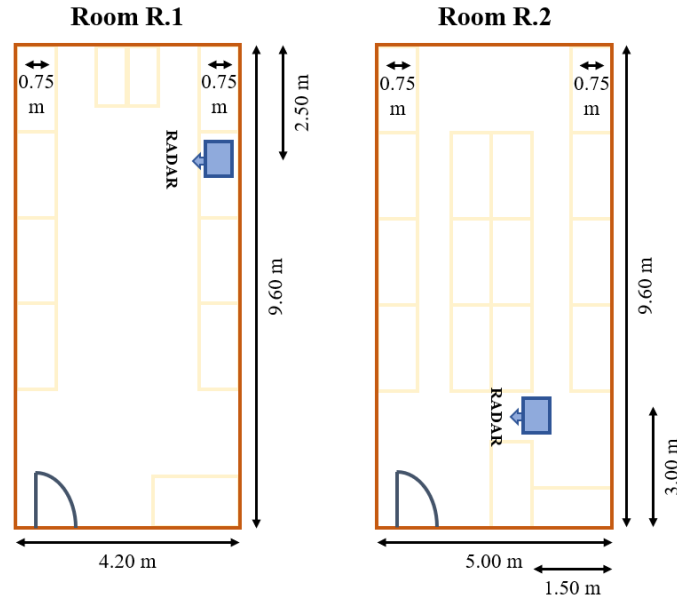


Figure 3.7: Schematic of the Acquisition Rooms R.1 and R.2 (figure is not to scale).

3.4 Characterization of the Datasets

Many ML algorithms and techniques rely on statistical information about the data to be run. Due to this fact, statistical analysis of data is an important step to both understand the data and what methods can be used to achieve the goals. Specifically, data plots and histogram analysis help by visually presenting data. If patterns are visible with the naked eye, then likely the statistical properties of datasets differ. In turn, this means it is possible to perform classification based on these properties.

To start the data analysis process, a simple process was performed. Taking all dataset frames with the same label belonging to Dataset Group 1 (see Table 3.4), without removing their respective clutter frames, and performing the mean bin by bin, the plots in Figure 3.8 were constructed. Although the plots are similar, there are differences around

the middle range bins, specifically around bins 30-80. This can be justified as the targets in Dataset Group 1 were at 1, 2 and 3 meters away from the radar, which theoretically corresponds around bins 25-75.

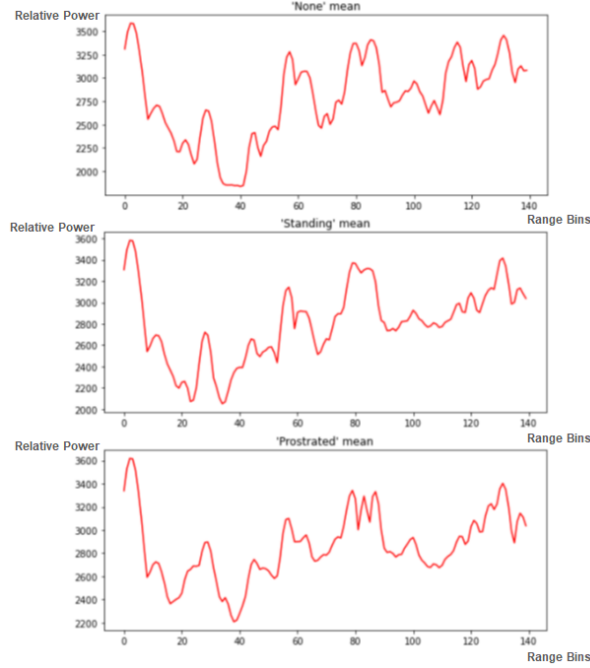


Figure 3.8: Mean of Range Arrays per Label from Dataset Group 1.

Although the previous plots provide useful information, data can be treated by removing the associated clutter. As explained already, clutter corresponds to the room's background static targets, which influence the statistical properties of these datasets. In order to remove their presence, every dataset has a clutter file associated with it, recorded during a calibration phase (see subsection 4.1.1 for more details). The plot in Figure 3.9-a) represents an example of a clutter frame. Following this, every plot presented in the previous picture can subtract the value in the clutter frame bin by bin, resulting in the plots shown in Figure 3.9-b) through d). Note that only 140 range bins are presented, corresponding to the maximum unambiguous distance resulted from the radar configuration parameters.

From here, two interesting conclusions can be made. First, when subtracting the clutter frame from the *None* scenario, only a small amount of noise is left. Note that the values in Figure 3.9-c) are several orders below their original value (specifically, the values go from around 3000 to around 4 at maximum). As expected, this step confirms that the clutter and the *None* scenario are very similar, as they both record the room as it is with no targets in front of the radar. Second, patterns emerge from the other labels after removing the clutter. In Figure 3.9-b), representing the *Standing* scenario, there are three spikes, corresponding to the various distances at which the target was when data was acquired. Similarly, in Figure 3.9-d), corresponding to the *Prostrated* scenario, values

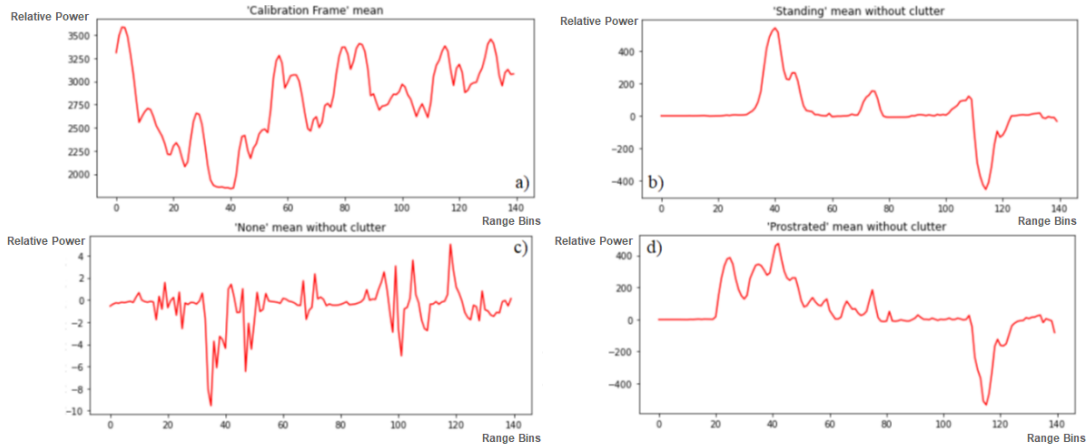


Figure 3.9: Mean of Range Arrays per Label with clutter removed from Dataset Group 1.

are present not only in the same bins as Figure 3.9-b), but also around these same values. This is due to the fact that a target prostrating sweeps all range bins between their legs and their head, resulting in frames with spread values.

In a second phase, the correspondent subtracted from every dataset frame. Afterwards, a count of the values presented in each bin was put in a histogram. It is expected to see a normal distribution of values in bins where no targets were present during acquisition. With no present targets and with clutter removed, this would indicate that noise follows a normal distribution. On the contrary, bins in which humans were present during acquisition should not follow a symmetric distribution, which should translate into having non-Gaussian distribution. In accordance, the histograms in Figures 3.10 and 3.11 represent the count of values for bins 25 and 45 respectively, separated in 30 bins, for each class label.

Figure 3.10, which represents the histograms for range bin 25, shows that noise is present in this bin for classes 0 and 1, since these follow a normal distribution. Class 2, however, already has some statistical information present in it, given that a prostrated person is closer to the radar than a person standing. Similarly, in Figure 3.11, which represents the histograms for range bin 45, represents noise for class 0, given its normal distribution, which can be explained given that no targets are present in any bin on the *None* scenario. However, this bin contains targets present in front of the radar, given the distance it represents. Therefore, two non-Gaussian distributions can be seen for classes 1 and 2, which means this bin contains information that can be used for classification through machine learning algorithms.

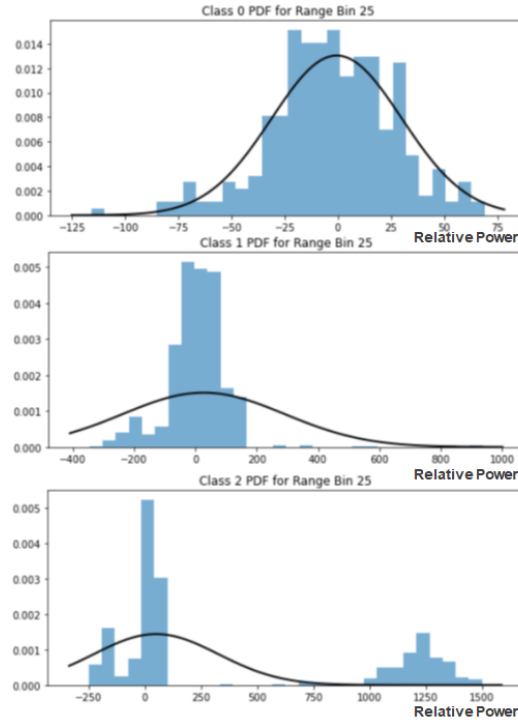


Figure 3.10: Histograms for Range Bin 25 separated by class label for Dataset Group 1.

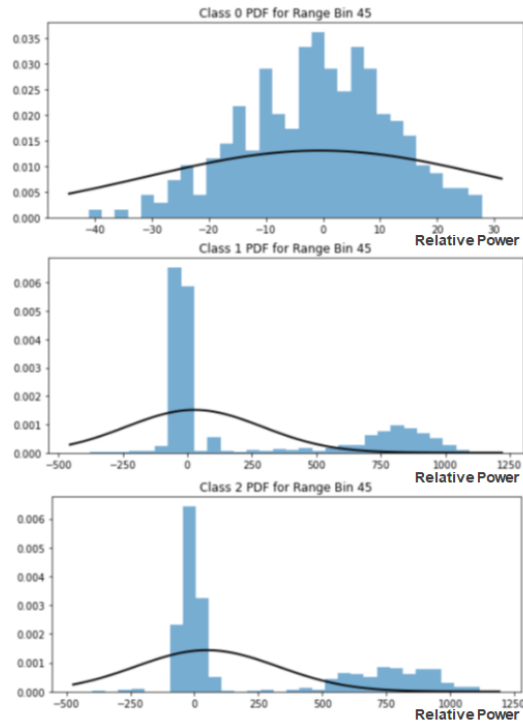


Figure 3.11: Histograms for Range Bin 45 separated by class label for Dataset Group 1.

CLASSIFICATION PROCESS

In this chapter, the classification process is presented and explained. Understanding this process is of great importance, as it serves as the base for the prototype developed, which is detailed in the next chapter. The classification process is described by steps, starting with the definition of the classification masks, followed by the regular classification method and finishing with a sequential classification enhancement of the method.

4.1 Classification Masks

In this section, the core methods of the classification process are presented. This is due to the fact that the classifier, described in section 4.2, relies on masks to return a result. Masks can be defined from any set of acquired data (see section 3.3 for further details on acquiring data). Note that even though the radar returns a RDM, only the range array is being used in all steps of the classification method, due to the fact that all scenarios are static. Therefore, a **dataset frame** refers to the range array taken from the RDM captured by the radar. In the next sections, the processes needed to define the masks are detailed.

4.1.1 Definition of Clutter Frame

In order to define the masks, it is necessary to perform an initial step, which is denominated **calibration phase**. The calibration phase consists on capturing a series of frames with no targets in front of the radar, which effectively translates into understanding the surroundings of the scenario where the target is sensed, constructing what is called the **clutter frame** $F_{clutter}$. This information is of extreme importance, since it allows the system to perform classification independently of both the room where the masks dataset was obtained and the room where the contexts are classified. As explained in section 3.3, each dataset has an attached file to it representing the clutter frame $F_{clutter}$, calculated as

$$F_{clutter} = \frac{1}{n_{calib}} \cdot \sum_{n=1}^{n_{calib}} V_{calib_n}, \quad (4.1)$$

where n_{calib} is the number of frames acquired during the calibration phase, and V_{calib_n} represents the n -th calibration frame.

4.1.2 Types and Definition of Masks

The next step is to define the masks M_l for every class label l , based on labeled dataset frames that will be used to correlate with live frames. These datasets contain the associated clutter frame $F_{clutter}$ in addition to the range array and label. Two different methods can be applied to define masks, each with its own different uses. The first one is based on subtraction operations, whereas the other one is based on division, hence separating them in **Subtraction Masks** and **Division Masks**.

4.1.2.1 Subtraction Mask

In this method, each dataset frame subtracts from itself the associated clutter frame $F_{clutter}$, which effectively removes any objects present in the room during the calibration phase, resulting in isolating the presence of new targets that appear in the radar's vision posterior to calibration. This way, it is possible to identify patterns related to each class, representative of the mask M_l , independently on the room the dataset frames were acquired on.

By definition, the subtraction masks are calculated as:

$$M_{sub_l} = \frac{1}{n_l} \cdot \sum_{n=1}^{n_l} (V_{l,n} - F_{clutter}), \quad (4.2)$$

where l represents the class label, n_l is the total number of frames of class l , $V_{l,n}$ represents the n -th dataset frame of class l , and $F_{clutter}$ represents the associated clutter frame. Figure 4.1 represents a dataset frame before and after undergoing the subtraction process.

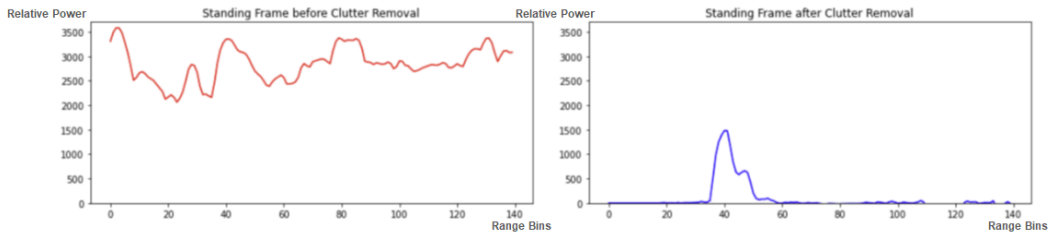


Figure 4.1: Frame before and after undergoing the clutter subtraction process.

Subsequently, these masks are manually observed to identify which bins contain significant information. Significant information can be described as any value far from 0,

since 0 means no change happened between the calibration phase and the acquired frame on that range bin. Figure 4.2 shows an example of this step.

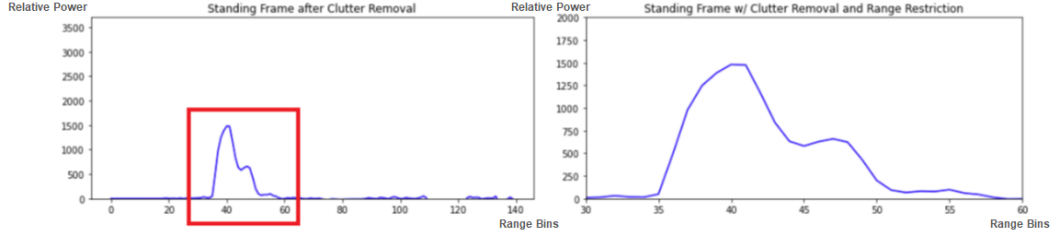


Figure 4.2: Frame before and after undergoing the range bins restriction process.

Lastly, the acquired masks must undergo the normalization process. Since this method is based on subtraction from a reference frame, values can be non-positive. Therefore, we divide the masks by their norm as follows:

$$M_{sub_norm_l} = \frac{M_{sub_l}}{\|M_{sub_l}\|}, \quad (4.3)$$

where the norm of the array can be described as follows (if the array contains real numbers only)

$$\|X\| = \sqrt{\sum_{i=1}^n X_i^2}, \quad (4.4)$$

where i represents the i -th element of the array X , and n is the total number of elements. This results in the final mask, which is represented in Figure 4.3 on the right.

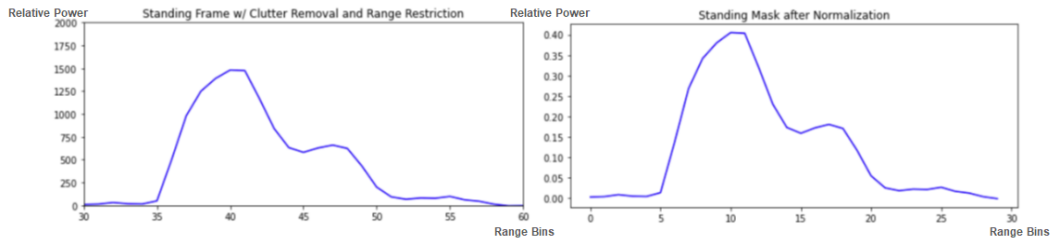


Figure 4.3: Mask before and after undergoing the normalization process. The plot on the right represents an example of possible final mask $M_{subtraction_l}$ for the *Standing* scenario.

Subtraction masks has a major limitation: they are extremely unreliable to define a pattern for the *None* scenario. This is due to the nature of the scenario itself: subtracting the clutter frame from a frame labeled as *None* results in an array with values close to 0, representative of noise. Often, normalizing these values results in peaks, which do not describe the *None* scenario and lead the system to classification errors.

4.1.2.2 Division Mask

The Division method is a bit simpler than the Subtraction one. Instead of subtracting the clutter frame from each dataset frame, the latter are instead divided by the clutter frame. The patterns for each class can be determined by performing the mean on all divided frames, as follows

$$M_{div_norm_l} = \frac{1}{n_l} \cdot \sum_{n=1}^{n_l} (V_{l,n}/F_{clutter}), \quad (4.5)$$

where l represents the class label, n_l the total number of frames of class l , $V_{l,n}$ the n -th dataset frame of class l , and $F_{clutter}$ the clutter frame.

This results in a pseudo-normalized array, which has values around 1 when there is no change between the calibration phase and the acquired frames, and has peaks with height relative to the difference between the acquired frames and the calibration frame. This array is therefore not limited in positive values. Figure 4.4 shows examples of masks obtained through this method.

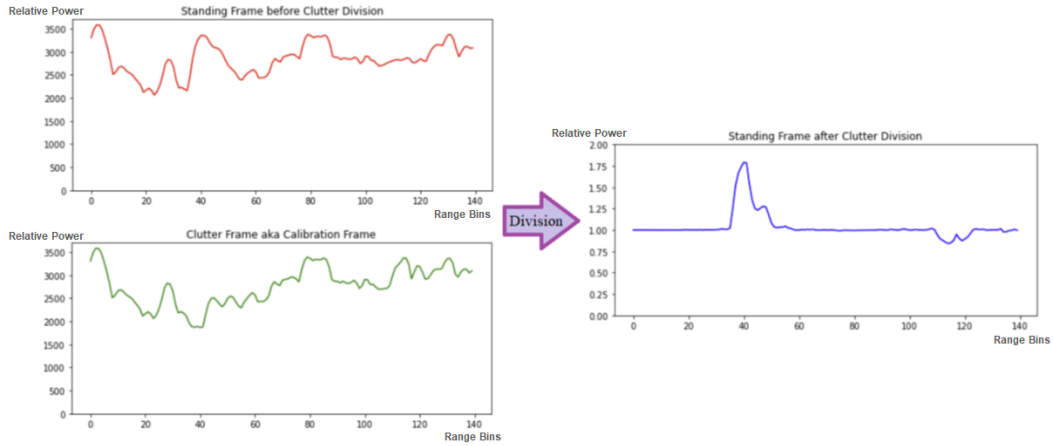


Figure 4.4: Frame before and after undergoing the clutter division process.

The Division method serves as a reliable alternative for the Subtraction method when defining a mask for the *None* scenario. When a frame labeled with *None* is divided by the clutter frame, the expected result is an array of values close to 1, slightly oscillating around this value. This information can be used to detect and classify a frame as *None*. However, no suitable method was found to normalize these masks. Consequently, the use of Subtraction masks is preferred for classifying all other scenarios, i.e. Standing and Prostrated.

4.2 Classification Method

To perform classification, the system undergoes a series of sequential steps that were summarized in subsection 3.3.1. First, two masks need to be defined offline for the

Standing and *Prostrate* scenarios, that is, M_1 and M_2 need to be defined. As explained in subsection 4.1.2, the subtraction masks are preferred for these cases. These masks are then loaded when the classifier is run, and will be used to correlate with live frames, where the highest correlation sets the output label of the system. Nonetheless, to mimic the steps used to define these masks, the live classifier still needs to undergo the calibration phase, in order to define the clutter frame $F_{clutter}$.

As mentioned previously, the *None* scenario can not be classified recurring to a subtraction mask, due to the fact that normalizing a noise array results in a series of random peaks, which does not represent the class. Instead, the division mask for class 0 is defined during the live calibration phase, and two features from this mask are used to classify this scenario. Due to this, the classification process is separated in two. These processes are explained in the sections below.

4.2.1 Classifying the None Context

Correctly classifying a frame as *None* is a rather simple process when recurring to the division mask. Note that unlike the other contexts, the mask used to define this scenario is obtained during the live calibration phase. This is due to the fact that the pattern describing the *None* context depends on the room where live classification is being performed.

The system uses frames from the calibration phase to define the division mask for the *None* context. From this mask, two features are calculated: the mean of the standard deviation and the standard deviation of the standard deviation of each frame obtained in the calibration phase relative to the clutter frame. These features are calculated as follows:

$$clutter_{std_mean} = \mu'_1(\mu_2(V_{calib}/F_{clutter})), \quad (4.6)$$

$$clutter_{std_std} = \mu_2(\mu_2(V_{calib}/F_{clutter})), \quad (4.7)$$

where V_{calib} is an matrix of range arrays, containing frames obtained during calibration phase, and μ'_1 and μ_2 respectively represent the simple mean and the second central moment, as described earlier in (2.21 - 2.22).

Noise can be considered to follow a random distribution where samples are statistically independent from each other. We consider that noise is approximately distributed as a Gaussian Distribution, based on the mean and standard deviation of the samples. Effectively, in a Gaussian Distribution, around 95% of samples are within two standard deviations from the mean, as shown in Figure 4.5.

To classify any frame as *None*, the system simply calculates the standard deviation of the frame to classify after diving it by the calibration frame. If this value fits within two standard deviations from the mean of the standard deviation, then it is considered a

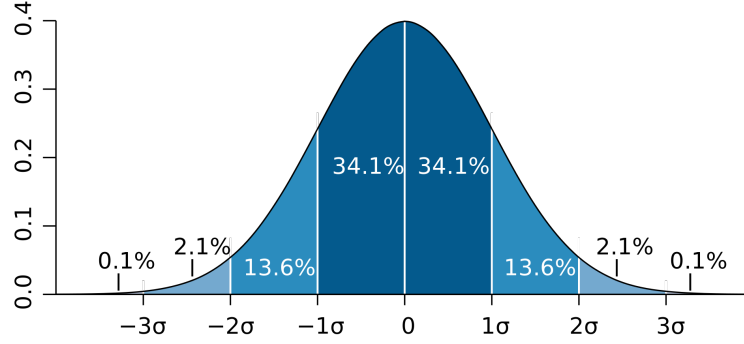


Figure 4.5: Gaussian Distribution. The standard deviation of the distribution is represented as σ . The graphic describes which percentage of values is within one, two and three standard deviations of the mean.

None scenario frame and the classification process is finished. If not, the system proceeds to the following classification method.

4.2.2 Classifying Standing and Prostrated Contexts

If a frame is not classified as *None*, the system proceeds to classify the frame as one of the remaining classes. Note that unlike the previous method, the system relies on subtraction masks obtained offline, defined as explained in section 4.1.2.1. These masks will be used to correlate with the frames captured live, as frames with the same label should have similar patterns.

To do so, the system starts by subtracting the calibration frame $F_{clutter}$ from the frame to be classified as follows:

$$F_{no_clutter} = (F - F_{clutter}), \quad (4.8)$$

where F is the frame to be classified and $F_{clutter}$ is the clutter frame.

To normalize the data, the normalization introduced in subsection 4.1.2.1 is also applied as follows:

$$F_{normalized} = \frac{F_{no_clutter}}{\|F_{no_clutter}\|}. \quad (4.9)$$

As already mentioned, the objective is to identify if specific patterns of a class, described by their respective mask, are present in the frame $F_{normalized}$. To help visualize this process, in Figure 4.6 it is possible to see two mask patterns (left) and the frame to be classified (on the right). The objective is to score which scenario mask is the most similar compared to the captured frame. In this case, the capture frame is from class *Prostrated*. To do so, the system performs a **cross-correlation** for each mask on the frame, which returns a **correlation array**, as described in

$$X_l[n] = \sum_{m=0}^{N-n-1} F_{normalized}[m] \cdot M_{sub_norm_l}[m+n], \quad (4.10)$$

where l represents the class label, M_l represents the class subtraction mask and $X_l[n]$ is the correlation array for every class l .

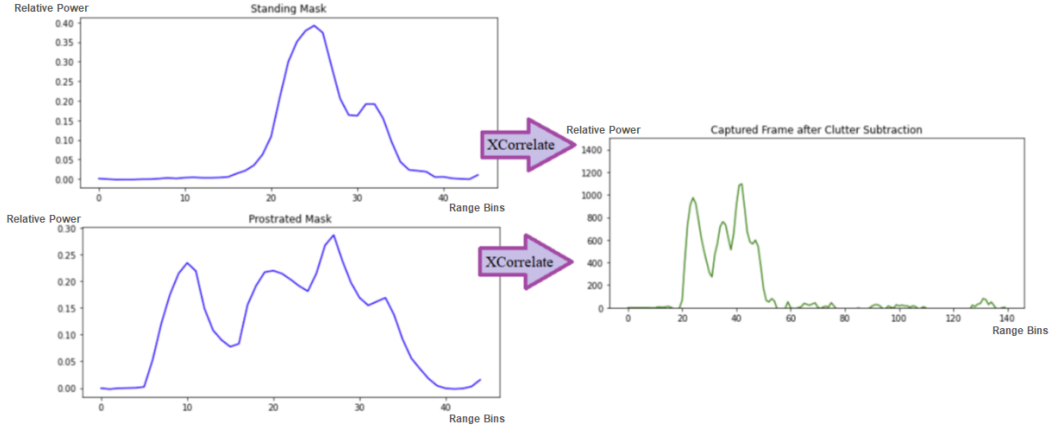


Figure 4.6: Frame undergoing the classification process (right) through the cross-correlation with each scenario's mask (left). It is visually possible to check that the frame to be classified has the same pattern as the prostrated mask M_2 .

Due to the fact that both the masks and the frame are normalized, the maximum correlation result is 1. Furthermore, this step grants flexibility in the target distance; independently of the range bins where the target is located in the frame, cross-correlation will produce a maximum. The system is then left with a correlation array for each remaining class other than *None*. Classification is performed based on the maximum correlation value, that is, the class which had the highest correlation:

$$class = argmax(max(X_l)), \quad (4.11)$$

where l is the class label, X_l is the correlation array for every class l , max denotes the maximum operator and $argmax$ returns the position of the maximum argument.

4.3 Sequential Classification

As discussed already, human motion classification can be described as a time series problem. This means that classification can be seen as dependent on previous contexts. Although this thesis only deals with static human targets, it can still be seen as a time series problem. As an example, given that frames are received for classification every 250 ms, it should be impossible to alternate between two scenarios every two consecutive frames.

The classifier described in the previous sections has no memory, that is, it does not take the previous classification results into account. As such, misclassification that could

be prevented can happen. In the following subsections, two methods for sequential classification are presented, which turn the classification performance more robust against isolated misclassification events.

Both algorithms are supervised, that is, they need two types of labels: first, the **actual label**, which represents the truth about the dataset frame. These are present in the Dataset Frames directory listed in section 3.3. In addition, the algorithms also need a **predicted label**, which represents the label attributed by the classifier described in the previous sections. These are present in the Prototype Classification directory, also listed in section 3.3.

4.3.1 Viterbi Sequential Classification

The Viterbi algorithm is a sequential classification algorithm that aims to calculate the most probable path. A path is the sequence of the last n outputs given from the described classifier. If the last classification result does not correspond to the most probable path given by Viterbi algorithm, the classification is altered to match it.

The Viterbi algorithm requires three to four inputs to be run:

1. **Observation State Sequence:** the sequence of the last n outputs provided by the classifier. In this work, n was chosen to have a size of 4;
2. **Transition Matrix:** sequential human motion classification can be represented as a Markov Chain, that is, a frame has a set probability of transitioning from one class to another. This matrix represents the probability for each state to transition to another. It is manually defined as:

$$T = \begin{bmatrix} 0.5 & 0.5 & 0.0 \\ 0.(3) & 0.(3) & 0.(3) \\ 0.0 & 0.5 & 0.5 \end{bmatrix}, \quad (4.12)$$

where $T_{i,j}$ represents the probability of transition from state i to state j . States 0, 1 and 2 represent, respectively, the *None*, *Standing* and *Prostrated* classes. For both practical and academic purposes, it is assumed that a target cannot go directly from *None* to *Prostrated* and vice-versa. Practically, it is hard for a target to appear in front of the radar already prostrated. Academically, if the probability from transitioning between these states is zero, then the Viterbi algorithm should not allow direct transition between them, and should correct the classifier output.

3. **Emission Matrix:** this matrix represents known information of classification error. As explained already, each dataset has two files representing the actual and predicted labels therefore being possible to count correct and incorrect outputs from the classifier. Each entry in the matrix represents the following:

$$E_{i,j} = P(\text{predicted} = j | \text{actual} = i), \quad (4.13)$$

where i and j represent classes *None*, *Standing* and *Prostrated*, respectively.

To exemplify a specific case, the value in $E_{1,2}$ represents the probability (which can be translated as a percentage if multiplied by 100%) of total cases where the classifier output was class 2, but in reality it should be class 1. The emission matrix was calculated for all frames in Dataset Groups 1 and 2, and is represented below.

$$E = \begin{bmatrix} 0.970370 & 0.024074 & 0.005556 \\ 0.065835 & 0.923691 & 0.010474 \\ 0.083621 & 0.173276 & 0.743103 \end{bmatrix}, \quad (4.14)$$

4. **(Optional) Initial State Probabilities:** if set, it represents the probabilities of the first state for every class. If not set, all classes are equally probable to be the first. In this work, the latter was chosen.

The most probable path is therefore dependent on the transition probabilities, the error probability of the classifier for both current and past classifications and on the current classification path provided by the past 4 outputs. In section 5.4, the results for running this algorithm are presented and discussed.

4.3.2 RNN Sequential Classification

Recurrent Neural Networks (RNNs) are an alternative method to Viterbi algorithm to perform sequential classification. In a similar way, RNNs use information of previous inputs and states to determine the new output. Their functioning is detailed in section 2.4.3. In the context of this thesis, the objective of the RNN is to determine the most probable class for a frame K , based on the last 4 predicted labels (present in the Prototype Classification file listed in subsection 3.3) provided by the classifier while knowing the actual labels (present in the Dataset Frames file, also listed in subsection 3.3).

The chosen RNN architecture is presented in Figure 4.7. Classification from frames corresponding to instances K through $K-3$, provided by the classifier in the Prototype Classification file, are saved in an array, and go through a process of One-Hot Encoding. Effectively, this means the input to the RNN is a 4×3 matrix. It is composed of 3 LSTM cells, followed by a Dense layer with 3 cells activated by a Softmax function. Therefore, the output is a 3 elements array, in which each position is the probability associated with classes 0, 1 and 2, respectively.

The RNN is trained with the actual labels present in the Dataset Frames file, where its weights are adjusted to better fit the 4 predicted labels to the 4 actual labels. All datasets from Dataset Groups 1 and 2 are used to train the RNN. The last step includes to save the RNN weights in order to be loaded by the live prototype. The results obtained with

RNNs for sequential classification are presented in the next chapter, under the subsection 5.4.

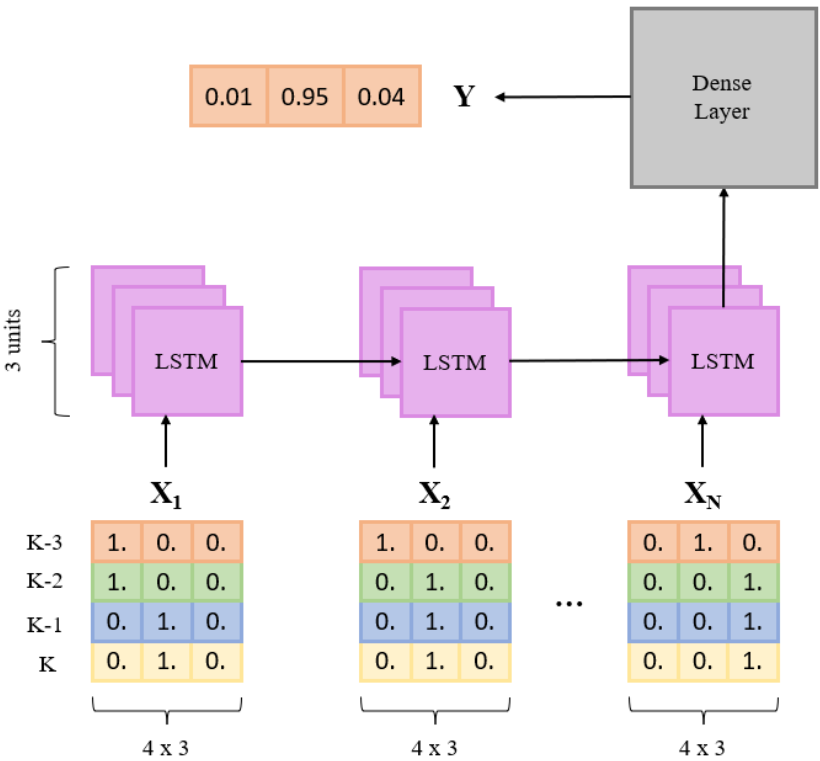


Figure 4.7: Architecture of the Recurrent Neural Network (RNN) to perform sequential classification.

PROTOTYPE AND PERFORMANCE EVALUATION

Following the thesis' objectives, a prototype capable of real-time frame acquisition and classification, following the previously described methods, has been developed. In this Chapter, the prototype is presented through a high-level interpretation of its functioning. Furthermore, relevant pieces of code that compose the prototype are explained.

5.1 Architecture

To understand the developed prototype, a block diagram representing its architecture is shown in Figure 5.1. The left part represents the radar acquisition system, which is the same system described in section 3.1 and used throughout this thesis. The radar parameters setup can be seen in Table 3.1. The right part of the diagram represents the computer software, responsible for preprocessing data and feeding it to the algorithm described in Chapter 4.

The radar systems begins by acquiring the I/Q samples and processing them internally, outputting a Range-Doppler Map (RDM). This process is continuously done with a periodicity of 250 ms. Every RDM is then transmitted to the computer via USB.

The computer is running a software composed of two threads, implemented in Python 3.9 and run in the PyCharm [70] integrated development environment. The first thread, named acquisition thread, is responsible for listening to the USB port, continuously waiting for bytes to arrive. When data is received, the thread checks for its integrity. Afterwards, the data parsing block rearranges (parses) the RDM map data into a matrix, which represents the received frame, and sends it to the classifier thread.

As mentioned before, the classifier thread's main purpose is to run the classification method described in Chapter 4. Therefore, the thread extracts the range profile (discarding the Doppler bins from the RDM, which are not used in the classification process),

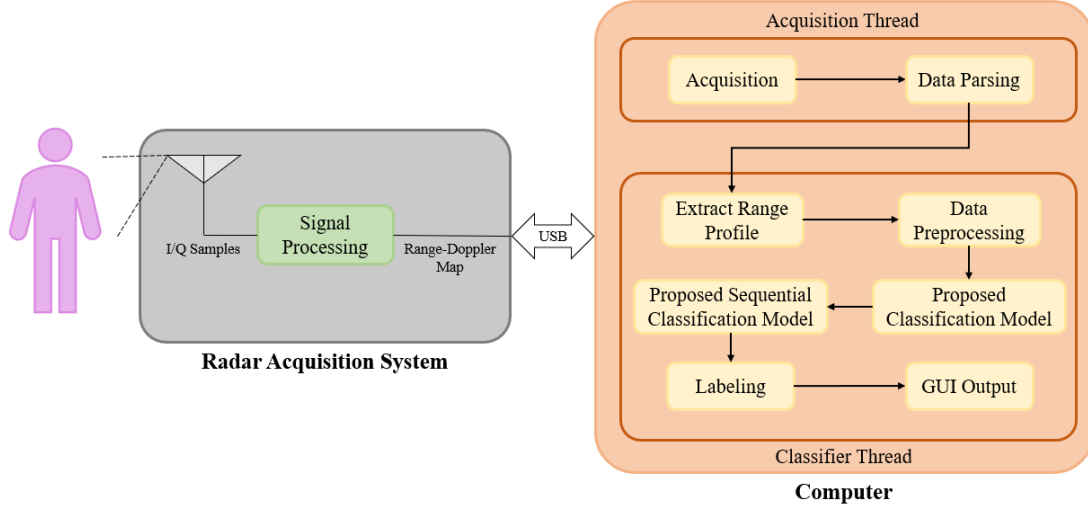


Figure 5.1: Block diagram representing the architecture of the developed prototype.

and normalizes the data. The next process includes feeding the data into the proposed classification model, in which its output is then fed into one of the proposed sequential classification models (see section 4.3). Finally, with the frame labeled, the output is provided to the user through a Graphic User Interface (GUI).

5.2 Realtime Acquisition, Calibration and Classification

The previous section describes a general view over the architecture of the developed prototype. In this section, the implementation of realtime acquisition, calibration and classification are explained in greater detail, presenting the corresponding sections of code. Furthermore, a list of all configurable parameters and what they represent is shown in Table 5.1.

Parameter Name	Description	Value
<i>max_n_calib</i>	# of frames in calibration phase	40
<i>print_graphics</i>	Sets the print of plots	Boolean
<i>camera_on</i>	Sets the recording of images	Boolean
<i>capture_dataset</i>	If set, frames acquired are saved	Boolean
<i>capture_duration</i>	Duration in seconds of datasets	15 - 25

Table 5.1: Configurable parameters used in the developed prototype, alongside their description and value.

As described throughout this thesis, the classification process starts with the acquisition of data. When the RDM data is received from the radar system and rearranged into a matrix in the acquisition thread, it is sent to a queue object, as depicted in Figure 5.2. This queue safely guarantees transmission of data from the acquisition thread to the classification thread. Note that a frame is received every 250 ms, so the classification

thread needs to read the queue object at least every 250 ms. Otherwise, the system will be classifying old frames as the queue fills up. This effectively puts a time limit to frame classification at 250 ms.

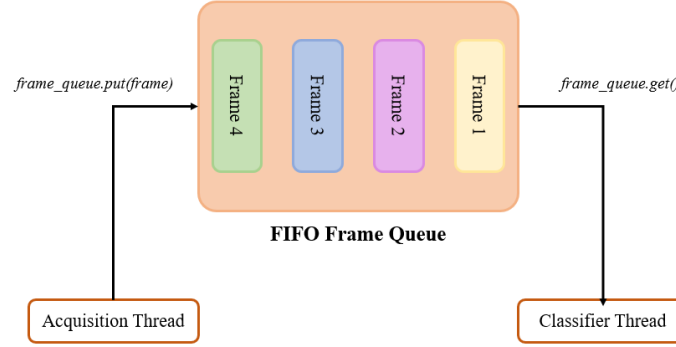


Figure 5.2: FIFO Queue data transmission system used between the acquisition thread and the classifier thread.

The classifier thread has two core methods: the `__init__` and the `run` methods. The `__init__` function is responsible for setting up all necessary objects when the thread is created, including the GUI window, the queue object and the loading of the trained RNN. The `run` function is composed of a loop that is executed as long as the thread is not killed by its `stop` function. Inside the loop, the queue object is awaiting the arrival of a new frame. As described in section 4.1.1, the set amount of first frames will be used to perform room calibration. Afterwards, all frames will be directed to the classification method.

```
1  # This is the calibration phase.
2  self.calibration_frames[self.curr_n_calib] = array[0][0:self.max_unambiguous_range_bin]
3  self.curr_n_calib = self.curr_n_calib + 1
4
5  # Checks if calibration phase is finished
6  if self.curr_n_calib == self.max_n_calib:
7      self.calibration = False
8
9  # Calculates clutter (calibration) frame and the two features
10 self.clutter_frame = np.divide(np.sum(
11     self.calibration_frames, 0), self.max_n_calib)
12 self.clutter_std_mean = np.mean(np.std(
13     np.divide(self.calibration_frames, self.clutter_frame), 1))
14 self.clutter_std_std = np.std(np.std(
15     np.divide(self.calibration_frames, self.clutter_frame), 1))
16
17 self.vs = self.setup_camera(self.camera_on) # camera setup
```

Listing 1: Calibration phase of the `run` function, present in the classifier thread.

In Listing 1, which represents the calibration phase, all frames received are saved in a list. In this work, the calibration phase lasts for 40 frames, which is equal to 10 seconds. When the 40-th frame arrives, the system switches from calibration phase to classification, and calculates the calibration frame (4.1), and the clutter's standard deviation's mean and standard deviation (4.6-4.7), used to classify the *None* scenario.

The classification stage starts by trying to identify if the received frame is a *None* frame. To do so, it performs the division classifier (see section 4.2.1), where the current frame is divided by the calibration frame, bin by bin. If the standard deviation of the resulting array is within the set parameters, it is considered a *None* frame. If not, the subtraction method (see section 4.2.2) is run, and whichever scenario has the highest correlation defines the class of the frame. This process is shown in Listing 2.

```
1  # Initializes correlation values to a minimum of 1
2  output = [-1, -1, -1]
3
4  # Perform the division classifier (for "None")
5  s_norm_division = np.divide(
6      array[0, 0:self.max_unambiguous_range_bin], self.clutter_frame)
7
8  # Compare the std of the captured live frame
9  # to the average+-2std std of the calibration frames
10 # If they match, this frame is classified as "None"
11 sample_std = np.std(s_norm_division)
12
13 if(self.clutter_std_mean - 2*self.clutter_std_std <=
14     sample_std <=
15     self.clutter_std_mean + 2*self.clutter_std_std:
16     output[0] = 1
17
18 else:
19     # Perform correlation of current frame with subtraction mask
20     s_clutter_removed = array[0][0:self.max_unambiguous_range_bin] -
21         self.clutter_frame
22     s_clutter_removed = s_clutter_removed / LA.norm(s_clutter_removed)
23
24     correlations = [[], []]
25     correlations[0] = np.correlate(s_clutter_removed, self.range_masks[2])
26     correlations[1] = np.correlate(s_clutter_removed, self.range_masks[3])
27
28     output[1] = np.max(correlations[0])
29     output[2] = np.max(correlations[1])
30
31 label = argmax(output)
```

Listing 2: Classification stage of the *run* function, present in the classifier thread.

Furthermore, in Listing 3 the code segment for running both the Viterbi algorithm and the previously trained RNN (explained in subsections 4.3.1 and 4.3.2 respectively) is presented. The program saves a list with the previous 4 standard classifications, which serves as the input for both Viterbi and RNN. In the case of Viterbi algorithm, the final label is the last class in the Viterbi path. For the RNN, the final label is given by the highest probability of the last layer of neurons, each corresponding to a class. These labels, as well as the runtime for each algorithm, are saved in variables, and are later used to calculate the KPIs.

```
1  # Stores the last 4 Standard Classification results in a list
2  memory_states = 4
3  self.update_prev_classification_list(label, memory_states)
4
5  # Run Viterbi
6  timer_begin = time.time() # to calculate runtime
7  if(len(self.prev_classifications) == memory_states: # list must have 4 elements
8      viterbi_path, T1, T2 = viterbi(np.array(self.prev_classifications),
9          self.trans_matrix, self.emission_matrix)
10     viterbi_label = viterbi_path[-1] # current frame is last on viterbi path
11     viterbi_runtime = time.time() - timer_begin
12
13  # Run RNN
14  timer_begin = time.time()
15  if(len(self.prev_classifications) == memory_states:
16      one_hot_enc = np.array(tf.keras.utils.to_categorical(
17          y=self.prev_classifications, num_classes=3))
18      rnn_label = np.argmax(self.model_predict(one_hot_enc.reshape(1, 4, 3)))
19      rnn_runtime = time.time() - timer_begin
```

Listing 3: Sequential Classification segment for Viterbi and RNN in the developed prototype.

5.3 Acquisition of the Datasets with the Prototype

The developed prototype was further enhanced with options to record all frames captured and their respective classification result. In fact, this is how the datasets described in Chapter 3 where acquired. Listing 4 depicts this process.

If the option to record a dataset is set, then after classification the system saves the range array (frame) and the classification result in a list. Furthermore, if the camera mode is on, a picture is taken at the time of classification and is saved on a list. When the set amount of frames to compose a dataset is reached, the system shuts down and saves the range array, the classification and the images in a new directory, composing a new dataset acquisition, as mentioned in a list in section 3.3.

```
1  # Checks if option is enabled for classification and dataset saving
2  if self.capture_dataset:
3      # Append the classification and captured frame
4      # these will be saved when the set amount of frames to capture are reached
5      self.classification_results.append(np.argmax(output))
6      self.realtime_frames.append(array[0][0:self.max_unambiguous_range_bin])
7
8      # Get Camera image
9      if self.camera_on:
10         [recording_time_str, recording_time, frame] = self.get_image(self.vs)
11         self.realtime_images.append(frame)
12
13     # Increments current n_calib and checks if set number is reached
14     self.curr_n_calib = self.curr_n_calib + 1
15
16     if self.curr_n_calib == self.max_n_calib + self_capture_duration:
17         # Saves range arrays, clutter frame, classifications and images...
```

Listing 4: Dataset record option of the *run* function, present in the classifier thread.

5.4 Performance Evaluation Results

As mentioned several times throughout this thesis, the main objective is the development of a prototype capable of classifying contexts in real time. As such, two Key Performance Indicators (KPIs) were defined: the **accuracy** of the classifier and the **processing runtime** needed for each method to provide said classification. The developed prototype was submitted to several tests, grouped and presented in the list below, in which the KPIs described above were calculated.

1. **Static Scenario Classification:** in each of these tests, a human performs a static action for the whole duration of the test. The objective is to evaluate the basic capability of the system to classify static scenarios;
2. **Non-Static Scenario Classification:** in these tests, a human performs a series of determined actions for a determined amount of time. Note that frames corresponding to transitions between classes are discarded and are not taken into account to calculate the above KPIs. The objective is to evaluate the capability of the system to change between the various scenarios, specifically where sequential classification algorithms should be more impactful.

5.4.1 Static Scenario Classification

Static scenario classification are tests where a human performs a static action for the duration of the whole test. Specifically, a total of 5 tests for each of the classes in Table

3.2 were realized, each with a total of 15 seconds, adding up to 60 frames per test and a total of 300 frames per class. Below, the results for the accuracy and processing runtime KPIs are presented.

Table 5.2: Accuracy values obtained for static scenario classification.

Standard Classifier		
None	Standing	Prostrated
$91.86 \pm 3.29 \%$	$99.33 \pm 1.33 \%$	$91.08 \pm 10.81 \%$
Viterbi Algorithm		
None	Standing	Prostrated
$91.86 \pm 3.29 \%$	$99.33 \pm 1.33 \%$	$91.08 \pm 10.81 \%$
Recursive Neural Networks		
None	Standing	Prostrated
$98.64 \pm 0.68 \%$	$95.33 \pm 4.52 \%$	$99.08 \pm 1.85 \%$

Table 5.3: Processing runtime values obtained for static scenario classification.

Standard Classifier		
None	Standing	Prostrated
$0.200 \pm 0.167 \text{ ms}$	$0.545 \pm 0.375 \text{ ms}$	$0.591 \pm 0.370 \text{ ms}$
Viterbi Algorithm		
None	Standing	Prostrated
$0.760 \pm 0.440 \text{ ms}$	$0.828 \pm 0.580 \text{ ms}$	$0.869 \pm 0.581 \text{ ms}$
Recursive Neural Networks		
None	Standing	Prostrated
$46.836 \pm 4.337 \text{ ms}$	$46.529 \pm 5.013 \text{ ms}$	$47.331 \pm 6.152 \text{ ms}$

The results presented in Tables 5.2 and 5.3 are of high interest. At first sight, standard classification and Viterbi algorithm provide the same results for static classification. These perform better classification for the *Standing* scenario than RNNs. Nonetheless, the later has higher accuracy for the other two scenarios, as well as less uncertainty. Looking up to the processing runtime, RNNs are slower than the other two algorithms, as expected; however, they are still well below the 250 ms threshold defined for this work, thus becoming an almost irrelevant fact.

5.4.2 Non-Static Scenario Classification

In these tests, a human was given a script of actions to be performed sequentially and timed. As mentioned already, the frames corresponding to transitions between scenarios were discarded and not accounted for the calculation of the KPIs. Specifically, the given script is:

1. **None** for 5 seconds;
2. **Standing** for 10 seconds;

3. **Prostrated** for 5 seconds;
4. **Standing** for 5 seconds;
5. **Prostrated** for 10 seconds;

In total, each test has an expected duration of around 35 seconds, corresponding to 140 frames. Five tests were performed, hence around 700 frames were classified. The calculated KPIs are presented in the tables below.

Table 5.4: Accuracy values obtained for non/static scenarios classification.

Standard Classifier		
None	Standing	Prostrated
$92.23 \pm 7.18 \%$	$97.21 \pm 1.99 \%$	$92.67 \pm 12.23 \%$
Viterbi Algorithm		
None	Standing	Prostrated
$91.43 \pm 8.59 \%$	$97.21 \pm 1.99 \%$	$92.67 \pm 12.23 \%$
Recursive Neural Networks		
None	Standing	Prostrated
$97.76 \pm 1.84 \%$	$90.07 \pm 6.20 \%$	$97.44 \pm 2.22 \%$

Table 5.5: Processing runtime values obtained for non-static scenario classification.

Standard Classifier		
None	Standing	Prostrated
$0.250 \pm 0.214 \text{ ms}$	$0.663 \pm 0.366 \text{ ms}$	$0.634 \pm 0.344 \text{ ms}$
Viterbi Algorithm		
None	Standing	Prostrated
$0.997 \pm 0.601 \text{ ms}$	$1.050 \pm 0.675 \text{ ms}$	$0.947 \pm 0.540 \text{ ms}$
Recursive Neural Networks		
None	Standing	Prostrated
$49.428 \pm 6.722 \text{ ms}$	$51.793 \pm 7.028 \text{ ms}$	$51.233 \pm 6.643 \text{ ms}$

The results shown in Tables 5.4 and 5.5 don't differ much from the ones presented in subsection 5.4.1. RNNs show a better accuracy result for the classes *None* and *Prostrated*, as well as less uncertainty, being more consistent when classifying these results. However, the standard classifier and the Viterbi algorithm show better results when classifying the *Standing* scenario. Similarly, RNNs have a higher processing time than the other two classifying methods, but are still well below the 250 ms requirement to perform real time classification.

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The main objective of this thesis consisted in the development of a prototype capable of performing real-time classification of contexts through a RF sensing solution. In accordance, the prototype was developed resorting to a FMCW radar, where three scenarios were proposed and several datasets were acquired for studying.

The datasets collected proved to have representative statistical information for every class. However, background information of the room where the radar operates is also recorded in these datasets, meaning that these were only representative for frames recorded with the same background. Therefore, a mechanism for eliminating the background information of the capture scenario, also known as clutter, was created. By eliminating the clutter from the datasets, these are independent of the room they were captured in. Consequently, patterns were identified for each class, defining the class masks M_l , used to correlate and classify captured live frames.

Since it is possible to consider context awareness as a time-series problem, two sequential classification ML algorithms were applied to the data: Viterbi algorithm and Recurrent Neural Networks (RNNs). The objective was to identify possible misclassification provided by the classifier, preventing and correcting them when detected.

The results obtained by the developed classifier are significantly good, achieving over 90% accuracy in all three methods. In specific, RNNs achieved the best results, with over 95% accuracy for most tests performed. However, they have lower accuracy when classifying the *Standing* scenario compared to the other two methods. Furthermore, RNNs have significantly heavier processing runtime, averaging around 45 - 50 ms; nonetheless, this value is still well below the 250 ms threshold requirement to perform real time capture and classification.

6.2 Future Work

This section explores potential future steps that can be taken following the achieved results with this thesis. Specifically, these ideas aim to increase the developed system's robustness and accuracy, through the use of bigger datasets, a wider variety of rooms, exploration of radar parameters and addition of new contexts. Furthermore, the system's complexity can be raised; since the classifier's highest runtime is around 60 ms, there is a time margin that allows the system to be more complex, if it means to achieve higher classification accuracy and/or robustness.

6.2.1 Increasing the Dataset Size

Most Machine Learning (ML) algorithms rely on statistical properties of the data in order to perform well. For ML classifiers, bigger datasets are usually preferred, leading to a better identification of patterns related to each class. The size of the dataset could be expanded, which would probably lead to better accuracy results, without increasing the processing runtime. However, a bigger dataset size implies more outliers are present, meaning that a robust outlier detector should be implemented in order to eliminate their presence before defining the masks.

6.2.2 Increasing the Variety of Rooms

Similar to increasing the dataset size, increasing the variety of rooms, even when removing the clutter, could lead to a better identification of the patterns that really describe each of the classes, resulting in better masks. If the defined masks are better, then the classifier could increase its accuracy, while maintaining the same system complexity and processing runtime.

6.2.3 Exploration of Radar Parameters

It can be interesting to have a mask defined with a set of radar parameters, but perform classification on a radar with different parameters, to increase usability across different configurations or even different devices. In addition to the cross-correlation, the mask could be shrank or expanded, in order to detect a pattern that would otherwise not correspond to the mask, adapting to the new radar's parameters.

6.2.4 Addition of Static and/or Non-Static Contexts

This work was a very early exploration towards context-awareness using RF sensing. Therefore, it was only intended to differentiate between a few contexts, namely *None*, *Standing* and *Prostrated*. A very important step would be to increase the amount of static contexts, testing if the system keeps the same accuracy results, verifying its viability in a real world context-awareness application. Furthermore, the addition of non-static

contexts, like *Walking* or *Sitting*, is an equally important step. It is important to note that non-static contexts should have statistical information of classes in Doppler bins of the Range-Doppler Map (RDM), requiring a new approach.

6.2.5 Adaptation of Classification Model

While the results achieved are very satisfying, scoring over 90% accuracy for every test performed, there is still room for improvement. Furthermore, RNNs, which have the highest processing runtime, take around 50 ms to classify a frame. Effectively, this means that the system's model complexity can be increased, should it translate into better accuracy or robustness results. Instead of classic RNNs, the usage of alternative architectures, such as CNN-LSTM [64] or Bi-LSTM [65], can provide interesting results.

In parallel, this work relies on the average of dataset frames labeled as a class l to define the masks M_l . However, as seen in [38] and [40], features can be used to decrease the system's complexity, dimensionality and the effect of outliers. Therefore, other features can be selected or extracted from the datasets other than the average, which could translate into better accuracy and/or processing runtime results.

BIBLIOGRAPHY

- [1] J. Fussell, D. Rundquist, and J. A. Harrington. "On defining remote sensing." In: *Photogrammetric Engineering Remote Sensing* 52.9 (1986), pp. 1507–1511. ISSN: 0099-1112 (cit. on p. 1).
- [2] M. I. Skolnik. *Introduction to Radar Systems*. 3rd. Vol. 19. 2. McGraw-Hill, 2001, p. 772. ISBN: 0-07-118189-X. DOI: 10.1108/sr.1999.08719bae.001 (cit. on pp. 1, 8).
- [3] J. J. Bisgaard, M. Heise, and C. Steffensen. "How Is Context and Context-awareness Defined and Applied? A Survey of Context-awareness". In: *Aalborg university* (2004), pp. 31–40. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.4517&rep=rep1&type=pdf> (cit. on p. 1).
- [4] P. Ongsulee. "Artificial intelligence, machine learning and deep learning". In: *International Conference on ICT and Knowledge Engineering* (2018), pp. 1–6. ISSN: 2157099X. DOI: 10.1109/ICTKE.2017.8259629 (cit. on pp. 1, 6).
- [5] L. Marquez and J. G. Salgado. *Machine Learning and Natural Language Processing*. 2000 (cit. on p. 1).
- [6] M. C. Yu. "Multi-criteria ABC analysis using artificial-intelligence-based classification techniques". In: *Expert Systems with Applications* 38.4 (2011), pp. 3416–3421. ISSN: 09574174. DOI: 10.1016/j.eswa.2010.08.127. URL: <http://dx.doi.org/10.1016/j.eswa.2010.08.127> (cit. on p. 1).
- [7] Z. Chen et al. *MoVi-Fi*. Vol. 1. 1. Association for Computing Machinery, 2021, pp. 392–405. ISBN: 9781450383424. DOI: 10.1145/3447993.3483251 (cit. on pp. 2, 32).
- [8] K. B. Letaief et al. "The Roadmap to 6G: AI Empowered Wireless Networks". In: *IEEE Communications Magazine* 57.8 (2019), pp. 84–90. ISSN: 15581896. DOI: 10.1109/MCOM.2019.1900271. arXiv: 1904.11686 (cit. on p. 2).
- [9] T. W. Company. *What is Weather Radar?* 2020. URL: <https://www.ibm.com/weather/industries/broadcast-media/what-is-weather-radar> (visited on 11/29/2021) (cit. on p. 2).
- [10] P. John and M. Steve. *Practical Data Acquisition for Instrumentation and Control Systems*. Ed. by Elsevier. 2003, p. 425 (cit. on p. 5).

- [11] P. Wang et al. “Deep learning-based human motion recognition for predictive context-aware human-robot collaboration”. In: *CIRP Annals* 67.1 (2018), pp. 17–20. ISSN: 17260604. DOI: 10.1016/j.cirp.2018.04.066. URL: <https://doi.org/10.1016/j.cirp.2018.04.066> (cit. on p. 5).
- [12] P. R. Boyce. “Review: The impact of light in buildings on human health”. In: *Indoor and Built Environment* 19.1 (2010), pp. 8–20. ISSN: 1420326X. DOI: 10.1177/1420326X09358028 (cit. on p. 6).
- [13] K. K. Singh et al. “LiDAR-Landsat data fusion for large-area assessment of urban land cover: Balancing spatial resolution, data volume and mapping accuracy”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 74 (2012), pp. 110–121. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2012.09.009. URL: <http://dx.doi.org/10.1016/j.isprsjprs.2012.09.009> (cit. on p. 6).
- [14] D. Gohring et al. “Radar/Lidar sensor fusion for car-following on highways”. In: *ICARA 2011 - Proceedings of the 5th International Conference on Automation, Robotics and Applications* (2011), pp. 407–412. DOI: 10.1109/ICARA.2011.6144918 (cit. on p. 6).
- [15] E. László, P. Szolgay, and Z. Nagy. “Analysis of a GPU based CNN implementation”. In: *International Workshop on Cellular Nanoscale Networks and their Applications* (2012). ISSN: 21650160. DOI: 10.1109/CNNA.2012.6331451 (cit. on p. 6).
- [16] I. G. Maglogiannis. *Emerging artificial intelligence applications in computer engineering: real word ai systems with applications in ehealth, hci, information retrieval and pervasive technologies*. Vol. 160. Ios Press, 2007 (cit. on p. 6).
- [17] R. J. Samworth. “Optimal weighted nearest neighbour classifiers”. In: *Annals of Statistics* 40.5 (2012), pp. 2733–2763. ISSN: 00905364. DOI: 10.1214/12-AOS1049. arXiv: 1101.5783 (cit. on p. 7).
- [18] E. P. Lee et al. “An ab initio study of RbO, CsO and FrO (X₂⁺; A₂) and their cations (X₃⁻; A₃)”. In: *Physical Chemistry Chemical Physics* 3.22 (2001), pp. 4863–4869. ISSN: 14639076. DOI: 10.1039/b104835j (cit. on p. 7).
- [19] G. I. Webb. “Encyclopedia of Machine Learning and Data Mining”. In: *Encyclopedia of Machine Learning and Data Mining* April (2014). DOI: 10.1007/978-1-4899-7502-7 (cit. on p. 7).
- [20] A. Patle and D. S. Chouhan. “SVM kernel functions for classification”. In: *2013 International Conference on Advances in Technology and Engineering, ICATE 2013* (2013). DOI: 10.1109/ICAdTE.2013.6524743 (cit. on p. 7).
- [21] C. Di Ruberto and L. Putzu. “A fast leaf recognition algorithm based on SVM classifier and high dimensional feature vector”. In: *VISAPP 2014 - Proceedings of the 9th International Conference on Computer Vision Theory and Applications* 1 (2014), pp. 601–609. DOI: 10.5220/0004740606010609 (cit. on p. 7).

- [22] “The research of the fast SVM classifier method”. In: *2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing, ICCWAMTIP 2015* x (2015), pp. 121–124. DOI: 10.1109/ICCWAMTIP.2015.7493959 (cit. on p. 7).
- [23] M. Guarnieri. “The early history of radar”. In: *IEEE Industrial Electronics Magazine* 4.3 (2010), pp. 36–42. ISSN: 19324529. DOI: 10.1109/MIE.2010.937936 (cit. on p. 8).
- [24] R. Page. “The Early History of Radar”. en. In: *Proc. IRE* 50.5 (May 1962), pp. 1232–1236. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1962.288078. URL: <http://ieeexplore.ieee.org/document/4066844/> (visited on 11/17/2021) (cit. on p. 8).
- [25] W. Semke et al. “Analysis of radar and ADS-B influences on aircraft Detect and Avoid (DAA) systems”. In: *Aerospace* 4 (Sept. 2017), p. 49. DOI: 10.3390/aerospace4030049 (cit. on p. 8).
- [26] M. Yearly et al. “A Brief Overview of Weather Radar Technologies and Instrumentation”. In: *IEEE Instrumentation and Measurement Magazine* 17 (Oct. 2014). DOI: 10.1109/MIM.2014.6912194 (cit. on p. 8).
- [27] S. Sun, A. P. Petropulu, and H. V. Poor. “MIMO Radar for Advanced Driver-Assistance Systems and Autonomous Driving: Advantages and Challenges”. In: *IEEE Signal Processing Magazine* 37.4 (2020), pp. 98–117. ISSN: 15580792. DOI: 10.1109/MSP.2020.2978507 (cit. on p. 8).
- [28] J. Hasch et al. “Millimeter-wave technology for automotive radar sensors in the 77 GHz frequency band”. In: *IEEE Transactions on Microwave Theory and Techniques* 60.3 PART 2 (2012), pp. 845–860. ISSN: 00189480. DOI: 10.1109/TMTT.2011.2178427 (cit. on p. 8).
- [29] C. Ding et al. “Continuous human motion recognition with a dynamic range-doppler trajectory method based on FMCW Radar”. In: *IEEE Transactions on Geoscience and Remote Sensing* 57.9 (2019), pp. 6821–6831. ISSN: 15580644. DOI: 10.1109/TGRS.2019.2908758 (cit. on pp. 8, 33).
- [30] J. Bhatia et al. “Object Classification Technique for mmWave FMCW Radars using Range-FFT Features”. In: *2021 International Conference on COMMunication Systems and NETWORKS, COMSNETS 2021* 2061 (2021), pp. 111–115. DOI: 10.1109/COMSNETS51098.2021.9352894 (cit. on pp. 8, 33).
- [31] C. Li et al. “A review on recent advances in doppler radar sensors for noncontact healthcare monitoring”. In: *IEEE Transactions on Microwave Theory and Techniques* 61.5 (2013), pp. 2046–2060. ISSN: 00189480. DOI: 10.1109/TMTT.2013.2256924 (cit. on p. 8).

- [32] M. Mercuri et al. "Digital Linear Discrete FMCW Radar for Healthcare Applications". In: *IEEE MTT-S International Microwave Symposium Digest 2019-June* (2019), pp. 144–147. ISSN: 0149645X. DOI: 10.1109/mwsym.2019.8701012 (cit. on p. 8).
- [33] S. Pal. *FMCW – Radar Design*. Vol. 65. 4. 2019, pp. 19–56. ISBN: 9781630815677. DOI: 10.1080/03772063.2019.1615392 (cit. on p. 9).
- [34] V. Milovanovic. "On fundamental operating principles and range-doppler estimation in monolithic frequency-modulated continuous-wave radar sensors". en. In: *Facta Univ Electron Energ* 31.4 (2018), pp. 547–570. ISSN: 0353-3670, 2217-5997. DOI: 10.2298/FUEE1804547M. URL: <http://www.doiserbia.nb.rs/Article.aspx?ID=0353-36701804547M> (visited on 02/14/2022) (cit. on pp. 9, 10).
- [35] Karthik Ramasubramanian. "Using a complex-baseband architecture in FMCW radar systems". In: *Texas Instruments White Paper* (2017), p. 10. URL: <http://www.ti.com/lit/wp/spyy007/spyy007.pdf> (cit. on pp. 11, 12).
- [36] G. Brooker and G. M. Brooker. "Understanding millimetre wave FMCW radars Radar Education View project Biomedical and Biomechatronic Systems View project Understanding Millimetre Wave FMCW Radars". In: January 2005 (2014). URL: <https://www.researchgate.net/publication/228979037> (cit. on p. 12).
- [37] V. C. Chen. *The micro-Doppler effect in radar*. Artech House, 2019 (cit. on p. 13).
- [38] G. Chandrashekar and F. Sahin. "A survey on feature selection methods". In: *Computers and Electrical Engineering* 40.1 (2014), pp. 16–28. ISSN: 00457906. DOI: 10.1016/j.compeleceng.2013.11.024. URL: <http://dx.doi.org/10.1016/j.compeleceng.2013.11.024> (cit. on pp. 14, 17–19, 67).
- [39] F. D. Enggar et al. "Performance comparison of various windowing on FMCW radar signal processing". In: *2016 International Symposium on Electronics and Smart Devices, ISESD 2016* (2017), pp. 326–330. DOI: 10.1109/ISESD.2016.7886743 (cit. on p. 16).
- [40] S. Khalid, T. Khalil, and S. Nasreen. "A survey of feature selection and feature extraction techniques in machine learning". In: *Proceedings of 2014 Science and Information Conference, SAI 2014* (2014), pp. 372–378. DOI: 10.1109/SAI.2014.6918213 (cit. on pp. 19, 67).
- [41] T. Wang et al. "Deep learning for wireless physical layer: Opportunities and challenges". In: *China Communications* 14.11 (2017), pp. 92–111. ISSN: 16735447. DOI: 10.1109/CC.2017.8233654. arXiv: 1710.05312 (cit. on pp. 21, 26).
- [42] S. Sharma, S. Sharma, and A. Anidhya. "Understanding Activation Functions in Neural Networks". In: *International Journal of Engineering Applied Sciences and Technology* 4.12 (2020), pp. 310–316 (cit. on p. 23).

- [43] “The Most Used Activation Functions: Classic Versus Current”. In: *2020 15th International Conference on Development and Application Systems, DAS 2020 - Proceedings* (2020), pp. 141–145. DOI: 10.1109/DAS49615.2020.9108942 (cit. on pp. 23, 24).
- [44] S. Albawi, T. A. M. Mohammed, and S. Alzawi. “Layers of a Convolutional Neural Network”. In: *Ieee* (2017), p. 16 (cit. on p. 25).
- [45] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 25, 26).
- [46] Y. L. Boureau et al. “Learning mid-level features for recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), pp. 2559–2566. ISSN: 10636919. DOI: 10.1109/CVPR.2010.5539963 (cit. on p. 26).
- [47] T. Suligoj et al. “Horizontal current bipolar transistor (HCBT) - a low-cost, high-performance flexible BiCMOS technology for RF communication applications”. In: *Facta universitatis - series: Electronics and Energetics* 28.4 (2015), pp. 507–525. ISSN: 0353-3670. DOI: 10.2298/fuee1504507s (cit. on p. 26).
- [48] A. Graves et al. “Unconstrained online handwriting recognition with recurrent neural networks”. In: *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference* (2009), pp. 1–8 (cit. on p. 26).
- [49] R. Neves. “An Overview of Deep Learning Strategies for Time Series Prediction”. In: (2018) (cit. on pp. 27–29).
- [50] A. Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. ISSN: 01672789. DOI: 10.1016/j.physd.2019.132306. arXiv: 1808.03314. URL: <https://doi.org/10.1016/j.physd.2019.132306> (cit. on p. 27).
- [51] J. Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE] (cit. on p. 30).
- [52] Google. “Generative Adversarial Networks”. In: (2019). URL: <https://developers.google.com/machine-learning/gan> (cit. on p. 30).
- [53] L. Metz et al. *Unrolled Generative Adversarial Networks*. 2017. arXiv: 1611.02163 [cs.LG] (cit. on p. 31).
- [54] Y. Cheng et al. “Sequential Attention GAN for Interactive Image Editing”. In: *MM 2020 - Proceedings of the 28th ACM International Conference on Multimedia* (2020), pp. 4383–4391. ISSN: 2331-8422. DOI: 10.1145/3394171.3413551. arXiv: 1812.08352 (cit. on p. 31).

- [55] A. Bulat, J. Yang, and G. Tzimiropoulos. “To learn image super-resolution, use a GAN to learn how to do image degradation first”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11210 LNCS (2018), pp. 187–202. ISSN: 16113349. DOI: 10.1007/978-3-030-01231-1_12. arXiv: 1807.11458 (cit. on p. 31).
- [56] G. E. Hinton and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647. URL: <https://www.science.org/doi/abs/10.1126/science.1127647> (cit. on p. 31).
- [57] J. Almotiri, K. Elleithy, and A. Elleithy. “Comparison of autoencoder and Principal Component Analysis followed by neural network for e-learning using handwritten recognition”. In: *2017 IEEE Long Island Systems, Applications and Technology Conference, LISAT 2017* (2017), pp. 12–16. DOI: 10.1109/LISAT.2017.8001963 (cit. on p. 31).
- [58] B. Rocca and J. Rocca. *Understanding Variational Autoencoders (VAEs)*. 2019. URL: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73> (cit. on pp. 31, 32).
- [59] H. Gunduz. “An efficient dimensionality reduction method using filter-based feature selection and variational autoencoders on Parkinson’s disease classification”. In: *Biomedical Signal Processing and Control* 66.January (2021), p. 102452. ISSN: 17468108. DOI: 10.1016/j.bspc.2021.102452. URL: <https://doi.org/10.1016/j.bspc.2021.102452> (cit. on p. 32).
- [60] J. K. Park et al. “Noncontact RF Vital Sign Sensor for Continuous Monitoring of Driver Status”. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.3 (2019), pp. 493–502. ISSN: 19324545. DOI: 10.1109/TBCAS.2019.2908198 (cit. on p. 32).
- [61] M. Wagner et al. “Threshold-Free Interference Cancellation Method for Automotive FMCW Radar Systems”. In: *Proceedings - IEEE International Symposium on Circuits and Systems* 2018-May.3 (2018). ISSN: 02714310. DOI: 10.1109/ISCAS.2018.8351077 (cit. on p. 32).
- [62] Y. Sun et al. “Gesture Classification with Handcrafted Micro-Doppler Features using a FMCW Radar”. In: *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility, ICMIM 2018* Icmim (2018), pp. 18–21. DOI: 10.1109/ICMIM.2018.8443507 (cit. on p. 33).
- [63] P. Goswami et al. “Real-Time Multi-Gesture Recognition using 77 GHz FMCW MIMO Single Chip Radar”. In: *2019 IEEE International Conference on Consumer Electronics, ICCE 2019* (2019), pp. 77–80. DOI: 10.1109/ICCE.2019.8662006 (cit. on p. 33).

- [64] J. Maitre, K. Bouchard, and S. Gaboury. “Fall Detection with UWB Radars and CNN-LSTM Architecture”. In: *IEEE Journal of Biomedical and Health Informatics* 25.4 (2021), pp. 1273–1283. ISSN: 21682208. DOI: 10.1109/JBHI.2020.3027967 (cit. on pp. 33, 67).
- [65] A. Shrestha et al. “Continuous Human Activity Classification from FMCW Radar with Bi-LSTM Networks”. In: *IEEE Sensors Journal* 20.22 (2020), pp. 13607–13619. ISSN: 15581748. DOI: 10.1109/JSEN.2020.3006386 (cit. on pp. 34, 67).
- [66] N. Pll. “IWR1843 Single-Chip 76- to 81-GHz FMCW mmWave Sensor datasheet”. In: SWRS228 (2019) (cit. on p. 35).
- [67] T. Instruments. *IWR1843BOOST Evaluation Board*. URL: <https://www.ti.com/tool/IWR1843B00ST> (cit. on p. 35).
- [68] T. Instruments. *mmWave Demo Visualizer*. URL: https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/3.5.0/ (cit. on p. 37).
- [69] PreSenseRadar. *OpenRadar Project*. 2020. URL: <https://github.com/PreSenseRadar/OpenRadar> (cit. on p. 41).
- [70] JetBrains. *PyCharm*. URL: <https://www.jetbrains.com/pycharm/> (cit. on p. 57).

