Master Thesis

Research on the Influence of Doppler Effect on the Classification of Pedestrians in Autonomous Vehicles

Submitted by:
Alam Sher Khan

Matriculation Number
1387324

Under the guidance of
Prof. Dr. Andreas Pech
Prof. Dr. Peter Nauth

Frankfurt University of Applied Sciences, 2024
Information Technology

Declaration

I, Alam Sher Khan, declare that this master's thesis, titled, 'Research on the Influence of Doppler Effect on the Classification of Pedestrians in Autonomous Vehicles' has been researched, worked on, and been solely written by me under the guidance of Prof. Dr. Andreas Pech and Prof. Dr. Peter Nauth. I hereby attest that I have properly credited and acknowledged all sources utilized in the development of this thesis. Any assistance I obtained from organizations or individuals during my research has been duly recognized in this thesis' acknowledgments section.

I am aware of this thesis's accuracy and accept full responsibility for it. I am completely aware that plagiarism, which includes utilizing someone else's work without giving due credit, is a serious academic violation that requires disciplinary action.

I give permission to the Frankfurt University of Applied Sciences to copy, store, and distribute my thesis in whole or in part in any format or medium for scientific or educational purposes.

Alam Sher Khan

November 15, 2024

Abstract

The development of autonomous vehicles (AVs) is expected to significantly improve transportation efficiency and safety. The ability to properly identify and categorize pedestrians is a key step in the autonomous-vehicle (AV) technology that experts believe will help prevent crashes, particularly on low-speed maneuvers such as parking is necessary. This research focuses on the Red Pitaya sensor implemented as a front car unit to analyse how Doppler Effect affects pedestrian classification using ultrasonic sensors. Ultrasonic sensors are widely used in AV systems as they provide a cost-effective solution that works across various environmental conditions. It emits high frequency sound waves that bounce off an object and return to the sensor providing distance measurement and further analysis can be done to understand the characteristics of the target object. However, when there is a relative motion between the sensor and object, then the Doppler Effect comes into play which is a change in the frequency of the waves that are reflected back which could create issues with false distance provision as well as incorrect categorization of objects including pedestrians. This issue is most severe in dynamic parking scenarios, where a vehicle can approach objects including pedestrians either towards, away or sideways.

In this study, we focus on the impact of Doppler Effect to pedestrian detection and classification accuracies as human or not human (object) during parking maneuvers. The data was collected at different pedestrian interactions with the Red Pitaya ultrasonic sensor. Data collected was categorized into Doppler-Effected datasets (when pedestrians or object is moving towards or away from the sensor) and Non-Doppler Effected datasets (while pedestrian or object is standing, stationary or moving sideways/laterally). The sensor data was processed and analyzed using suitable algorithm, which was then used to the train machine learning (ML) model on both kinds of datasets to improve classification performance.

The efficiency of the ML model was estimated on metrics like accuracy, precision, recall and F1 score. The performance of the ML model is compared for both doppler effected and non-doppler effected dataset scenarios. This research is helpful in understanding the Doppler Effect on pedestrian classification in AV systems in parking maneuvers.

Keywords *- Doppler Effect, Pedestrian Classification, Autonomous Vehicles, Ultrasonic Sensors, Red Pitaya, Machine Learning, MLP, CNN*

Acknowledgement

I would sincerely like to thank my thesis supervisors Prof. Dr. Andreas Pech and Prof. Dr. Peter Nauth for providing me with full support through the entire journey of this thesis. The apt technical support and guidance, uncomplicated knowledge transfer and regular feedback were a big help to gain the needed knowledge and include the same to improve my thesis work. I would once again like to thank Prof. Dr. Andreas Pech for being patient enough to guide me throughout the research work. Moreover, I would also like to extend my earnest gratitude to my family and friends for being helpful and supportive during the hard times.

Finally, I would like to thank each and every one who supported and guided me in any way, during my master's in information technology, at Frankfurt University of Applied Sciences.

Sincerely,
Alam Sher Khan

Contents

List of Figures

List of Tables

Listings

Abbreviations

AV: **A**utonomous **V**ehicles

ML: **M**achine **L**earning

CNN: **C**onvolutional **N**eural **N**etworks

MLP: **M**ulti-**L**ayer **P**erception

DAC: **D**igital to **A**nalog **C**onverter

FFT: **F**ast **F**ourier **T**ransform

ADC: **A**nalog to **D**igital **C**onverter

ReLU: **Re**ctified **L**inear **U**nit

OCR: **O**ptical **C**haracter **R**ecognition

DFT: **D**iscrete **F**ourier **T**ransform

GUI: **G**raphical **U**ser **I**nterface

ML: **M**achine **L**earning

Chapter 1

Introduction

Autonomous Vehicles (AVs) are promising a quantum leap in the efficiency and safety of transportation, rising to new levels on every metric by which we measure that performance. This system of sensors and algorithms enables AV to operate on its own as it can also navigate from a different destination without the need for human intervention. One of the main reasons is pedestrian detection and classification, a make-or-break capability that prevents accidents, ensuring safety for vehicle occupants as well as those on foot. This ability is of particular importance during low-speed operations where the vehicle proximity to obstacles and pedestrians results in increased risk of collisions.

The fact that ultrasonic sensors are cost-effective and robust in numerous environmental situations makes them an essential part of AV systems. These sensors operate by projecting high-frequency sound waves that bounce back from obstacles to reach the sensor again, providing accurate detection and classification of objects. Yet, the presence of the Doppler Effect (i.e., a change in frequency experienced by reflected waves due to motion between sensor and object) is responsible for errors that can affect how accurate these numbers are. A possible effect of such frequency shift is that the objects would be misclassified including pedestrians, potentially making AV systems unsafe and ineffective. In this thesis, we provide studies about the effect of Doppler Effect to pedestrian classification with Red Pitaya ultrasonic sensor and examine that advanced machine learning (ML) algorithms can increase detection efficiency.

1.1. Problem Statement

This study targets the Doppler Effect for a main reasoning on ultrasonic sensors performance in pedestrian classification task of Autonomous Vehicles (AVs). The frequency of reflected sound waves modifies due to the Doppler Effect between sensor and object's relative motion. But this adjustment could induce significant errors in object recognition, misidentifying pedestrians as other objects and risking the performance and safety of AV systems.

Challenges

- **Doppler Effect:** Frequency distortion leading to misclassification of pedestrians and objects. For example, this distortion can produce both false positives (in which non-pedestrian objects are identified as

pedestrians) and false negatives (in the sense that no pedestrian is actually found).

- **Misclassification:** Improper classification of pedestrians, or even objects due to distorted sensor data could lead up to collisions. Mislabelling not only undermines the operational safety of the vehicle, but also contributes to undermining confidence in autonomous vehicles amongst the general public.
- **Safety Hazards:** Misclassifying pedestrians (even in low-speed actions like parking) can have dangerous consequences for public safety. As such these maneuvers require accurate classification to prevent collisions given the close proximity with pedestrians.

1.2. Motivation and Objective

What prompted this research was the need to enhance AV Systems reliability and safety by addressing challenges with Doppler Effects on pedestrian classification accuracy. The ongoing development of technology requires AV companies to take another step toward quantifiable sensor data-processing accuracy for the widespread adoption and deployment of self-driving cars.

Objectives

- **Understanding the effect:** To analyse how much the Doppler effect affects pedestrians' ability to detect. Among these is a thorough examination of how frequency changes affect sensor readings and, consequently, pedestrian classification and identification. The precise nature of these distortions must be understood in order to create machine learning models that can classify even the doppler-affected information.
- **Model Development:** To create and apply machine learning models capable of efficiently classifying the sensor data impacted with Doppler Shift. These models will improve the overall resilience of pedestrian categorization systems by utilizing sophisticated algorithms to rectify the distortions brought about by the Doppler Effect.
- **Evaluation of performance:** To assess these models' performance in scenarios that are both Doppler-affected and non-doppler-affected. We'll utilize important metrices like accuracy, precision, recall, and F1 score to gauge how well the models work. This assessment will offer a thorough appraisal of the models' performance under actual settings.
- **Cross-Movement Evaluation of Pretrained Classifiers:** This marks the assessment of how well the suggested machine learning models perform on Doppler effected dataset, compared to that of classifiers trained on non-Doppler dataset. This comparison will also highlight the practical benefits of this research by providing evidence for a significant increase in detection and classification accuracy available using these new models.

1.3. Research Questions

The following are the main questions that this study seeks to answer:

- In what ways does the Doppler Effect affect ultrasonic sensor-based pedestrian classification accuracy?
- Is it possible for modern machine learning algorithms to efficiently reduce the Doppler Effect's influence on sensor data?
- What gains in classification precision and accuracy over the pre-trained ML classifier trained on non-Doppler dataset can be made by utilizing machine learning models specifically trained on Doppler effected dataset?
- Regarding Doppler-Effected vs. Non-Doppler-Effected datasets, what is the performance of the ML models?

1.4. Structure of the Document

The structure of the thesis is as follows:

- **Chapter 2: Concepts and Fundamentals**

The ideas and fundamentals of signal processing, machine learning (ML), the Doppler effect, and ultrasonic sensors are briefly covered in this chapter. Beginning with the fundamentals of machine learning and moving on to the mathematical modelling of Doppler-effect physics, it also establishes the foundation for comprehending the technical components of the work.

- **Chapter 3: Literature Review**

The first part of this chapter will summarize the machine learning applications for sensor data analysis, then effects of Doppler Effect and application of ultrasonic sensors to be used in AV systems. It highlights a missing area in the literature this thesis aims to fulfil, providing an analysis of previous research and reflecting on how this study has made original contributions.

- **Chapter 4: Methodology**

In this chapter we discuss the research design, data collection process using Red Pitaya sensor and dataset classification steps followed by machine learning models implementation. For doing this job, we describe the sensor data analysis approach followed, experimental setup and pre-processing for cleansing as well as formatting of raw data. This chapter also discusses the methods for evaluating how well the ML models are performing.

- **Chapter 5: Observations and Results**

The results of the research, divided into performance of ML models on Doppler-Effected dataset and Non-Doppler Effected datasets are shown in this chapter. It explores the relevance of these results for AV systems by an exhibited comparison with pre-trained ML classifiers in terms of the classification metrices accuracy, precision, recall and F-1 score.

- **Chapter 6: Conclusion and Future Work**

The entire thesis is summarized in this chapter. Furthermore, recommendations, limitations and suggestions for future work are discussed as well. Finally, we discuss avenues for improvements and their significance as future research directions.

Chapter 2

Concepts and Fundamentals

In this chapter, human movements and object classification based on machine learning will be explained following with the theoretical background necessary to understand Doppler shift in ultrasonic data research. To record the movements data correctly, we started with basic understanding of how ultrasonic sensors work by describing some basics in sound wave propagation & reflection. In this very chapter, we also look into Doppler effect and learn how frequency changes while an object in movement approaches towards or separates from sensor, hence can be used to differentiate between different type of motion.

The chapter also deals with the basics of machine learning, focusing on algorithms, like CNN and MLP, used in this research work to classify movement patterns from sensor data. Finally, signal processing methodologies are discussed to improve and prepare the data better for analysis and classification. Such a theoretical framework allows for the basis of experimental strategies and findings presented afterward in the next following chapters.

2.1. Ultrasonic Sensors

An ultrasonic sensor is a technical device which, by using sound waves inaudible to human ears, measures, detects, and estimates many characteristics of objects within their immediate vicinity. They normally operate at an ultrasonic frequency, greater than 20 kHz, with a normal frequency of about 40 kHz. The transducer's emission of ultrasonic pulses is the fundamental mechanism behind the operation of ultrasonic sensors. As these pulses travel through the atmosphere, some of the sound waves are reflected back to the sensor when they come into contact with an item (Pech et al., 2019). Applications for ultrasonic sensors are many and include robots, automobile systems (like parking assistance), industrial automation, and distance measuring equipment. The sensor determines the distance to a target by timing the intervals between sending and receiving the ultrasonic pulse. Ultrasonic sensors are a great way to detect objects that are transparent. No matter an object's colour, surface, or composition, ultrasonic sensors can detect its presence (unless it's made of a very soft material, like wool, in which case it would absorb sound). When optical technologies might not be able to detect transparency or other objects, ultrasonic sensors provide a reliable alternative (Hosur et al., 2016).

2.2. Principle of Ultrasonic Sensor Operation

Depending on the medium that sound waves are passing through, they move at various speeds. Broadly speaking, the speed of sound depends on the density and stiffness of the medium. The speed and physical properties of sound are influenced by its surroundings (Hosur et al., 2016). The sensor's ultrasonic pulse echo mechanism is depicted in Figure 2.1. An ultrasonic pulse is emitted by the sensor's transducer; this pulse travels away from the transducer and is reflected when it strikes an object. The transducer will receive the reflected signal, which it can then process to either measure the object's distance from it (which is determined by the amount of time it takes for an ultrasonic pulse to travel from the transducer back to it after reflection) or combine it with additional analysis of the received waveform to identify the features of the target object (Pech et al., 2019).

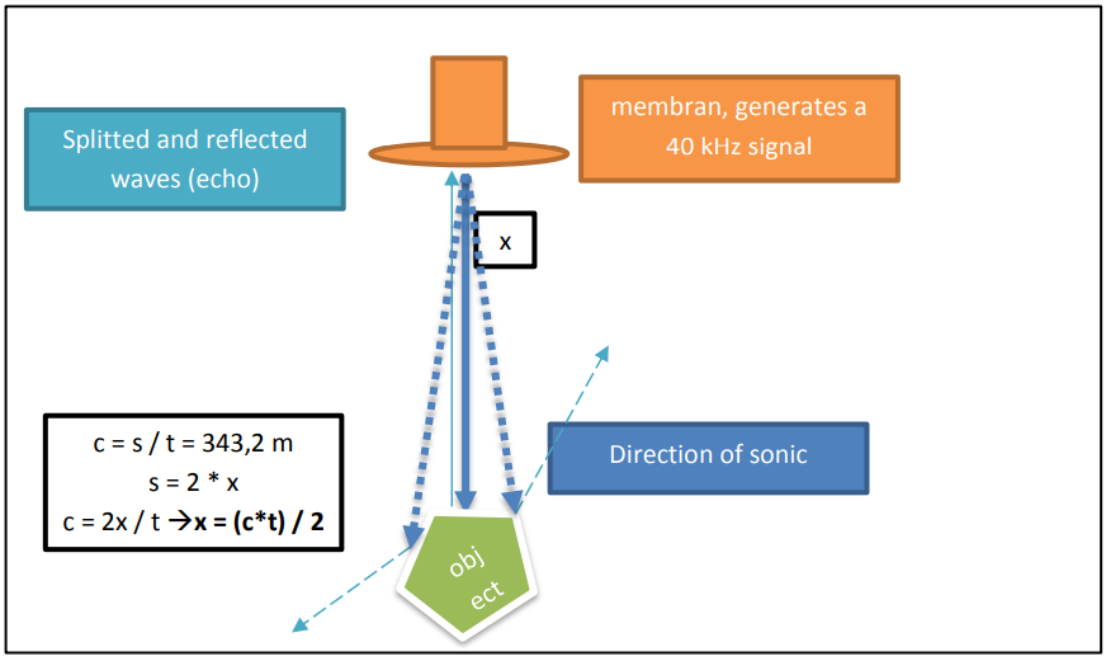


Figure 2.1: Basic principle of an Ultrasonic sensor (Tiniya, 2023)

2.3. Red Pitaya Ultrasonic Sensor

Red-Pitaya is a low-cost STEM Lab board that generates and measures Analog signals using microchip technology. Dual fast DAC (Digital-to-Analog converter) and twin fast ADC (Analog-to-digital converter) on the Red Pitaya make it easy to capture and generate two sine waves with any phase difference at the same time. It can be accessed or controlled using a number of methods, including as the SSH protocol, USB-Serial Console, and browsers on PCs or tablets (Red Pitaya, n.d.). It is powered by Linux. Engineers refer to Red Pitaya as their Swiss Army Knife. This equipment is distinct due to its dual Analog and digital input/output capabilities. It features an effective dual-core processor and several connectivity options, such as USB, Ethernet, and Wi-Fi. In this project, the STEM lab 125-14 is used which is a popular tool for practical learning, experimentation, and research (Arnaldi, 2017).

The sensor hardware used in this project consist of the following two parts:

1. Ultrasonic Sensor SRF02 with a mean frequency of 40 kHz and a power output of 150 mW (manufacturer's data) for sensing (Pech, Nauth, & Michalik, 2019).
2. Embedded System Red Pitaya with a sampling frequency of 1.95 MS/s and a resolution of 14 bit for controlling the SRF02 and for Analog digital conversion of the received signal (Pech, Nauth, & Michalik, 2019).

In order to initiate the emission of an ultrasonic pulse for a measurement, the Red Pitaya system as shown in Figure 2.2 transmits control data to the sonar sensor SRF02 as shown in Figure 2.3 over the I2C interface's wires SDA and SCL. The sensor's reflected signal is supplied into the Red Pitaya's Analog digital conversion input, where it is converted with a resolution of 14 bits and a sampling frequency of 1.95 MHz . Next, the sampled signal is sent via WLAN to a laptop for signal analysis using a data analysis software mentioned in Section 2.4 (Pech, Nauth, & Michalik, 2019).

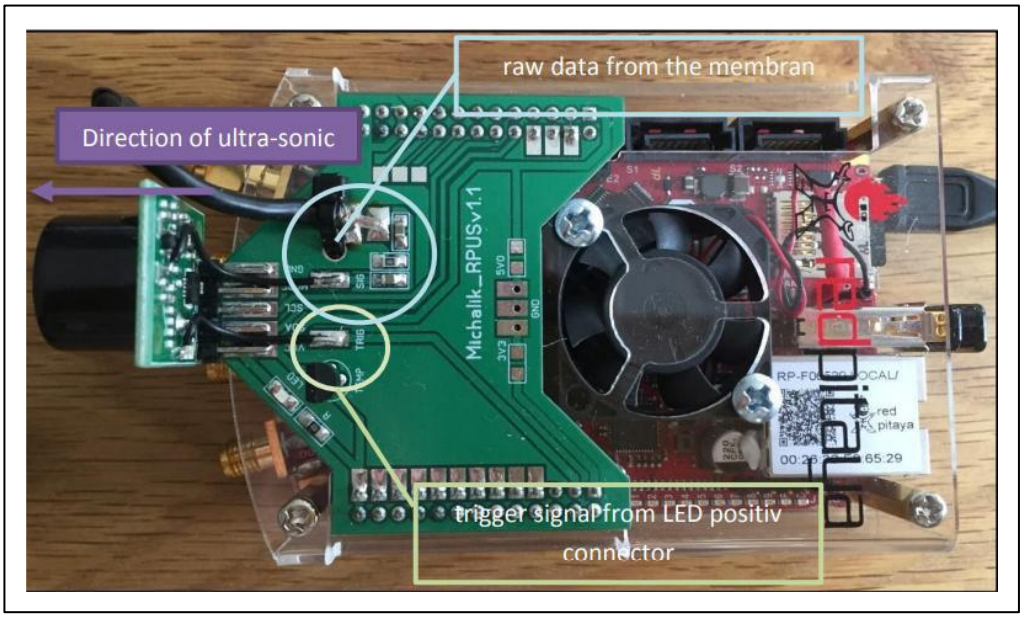


Figure 2.2: Red Pitaya based ultrasonic sensor

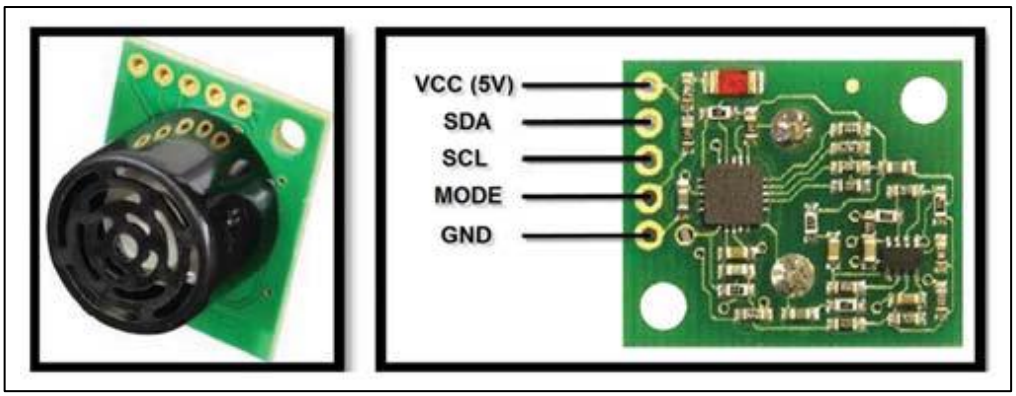


Figure 2.3: Sonar Sensor SRF02 (Pech, Nauth, & Michalik, 2019)

Red Pitaya offers various benefits:

1. It acts as a replacement of bulky lab instruments.

2. It has open-source software, hence allowing the user to alter it accordingly.
3. It can be programmed in multiple languages such as Python, MATLAB etc.
4. It uses the latest real-time signal processing.

2.4. Ultrasonic Data Collection Software

The measurement in terms of raw and analogue data for the ultrasonic sensor was obtained using the Ultrasonic Data Collection Software developed by the students of Master of Information Technology Engineering, Frankfurt University of Applied Science. The frequency values range between 34.9 kHz and 44.9 kHz, while the amplitude values go between 0 and 1000. Data collected is aligned in columns as indicated within the GUI of Figure 2.4 below. This results in 85 columns, with the frequency values thoughtfully spaced at 1.08 to 1.09 kHz. This application can acquire FFT data, but also ADC data. The first four bytes of the received stream of ADC data represent header bytes, while the next four bytes represent the data length. The data length is represented as 32832 - 64, which gives 32768 bytes, by the size of the header in 64 bytes. The number of header information is 16 in total, since its length is represented by 4 bytes, 64/4.

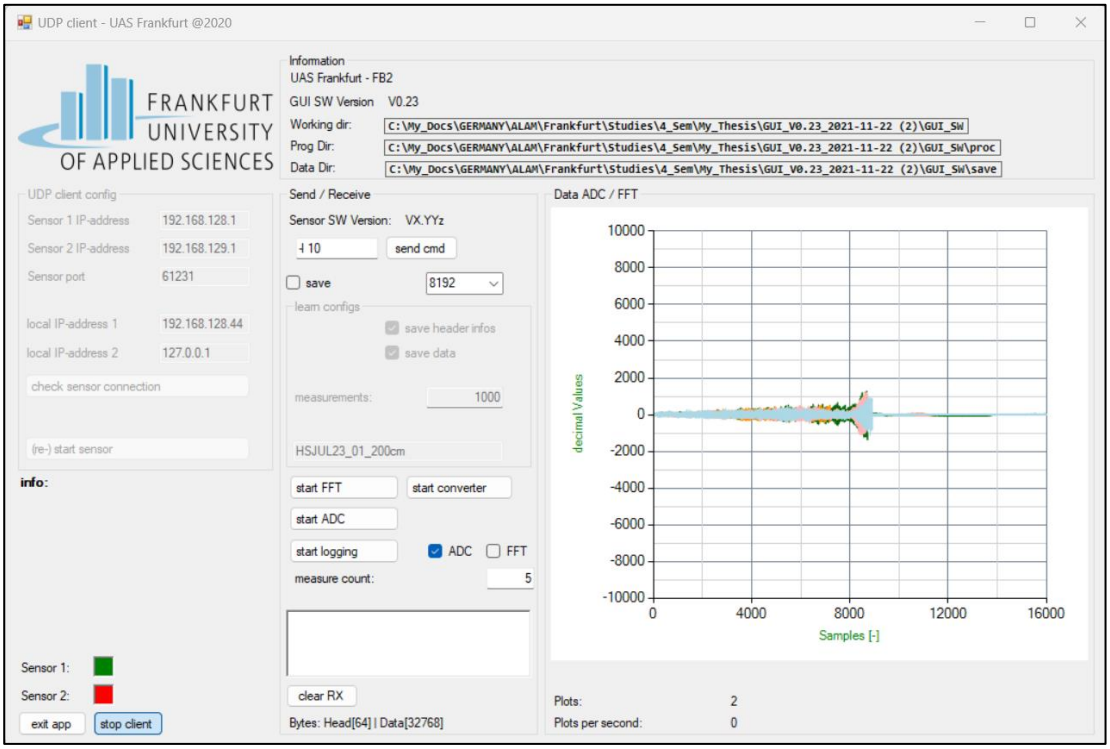


Figure 2.4: GUI of Ultrasonic Data Collection Software

Similarly, the dataset consists of 16384 signed integers (32768/2) because the data length is determined by 2 bytes. Data headers include crucial details including data length, classification outcomes from an embedded model, and sampling frequency (Pongsomboon, 2022). The existence of a classification model signifies that the software can categorize, or label collected data based on predefined standards.

2.5. The Doppler Effect

2.5.1. Explanation of the Doppler Effect

When a wave source and an observer are moving relative to one another, a phenomenon known as the Doppler Effect occurs. The effect, which bears the name of the Austrian physicist Christian Doppler, who initially put forth the idea in 1842, explains how a wave's apparent frequency varies according on source or observer movement. This phenomenon is known as the positive Doppler shift. It occurs when the wave's source moves in the observer's direction, increasing the apparent frequency. Quite the opposite; a negative Doppler shift occurs when the wave's source moves away from the observer, causing the apparent frequency to decrease (Doppler, 1842).

Figure 2.5 shows the motion of sound waves for stationary and moving sources through a stationary air mass. Case (a): stationary source. Case (b): moving source, where the compression of the wave on one side and the stretching on the other result in shorter wavelengths in the direction of motion and longer ones in the opposite direction. If the observers are in motion, Case (c) defines the change in frequency that does increase for the observer moving toward the source and decreases for the one moving away (LibreTexts, 2023).

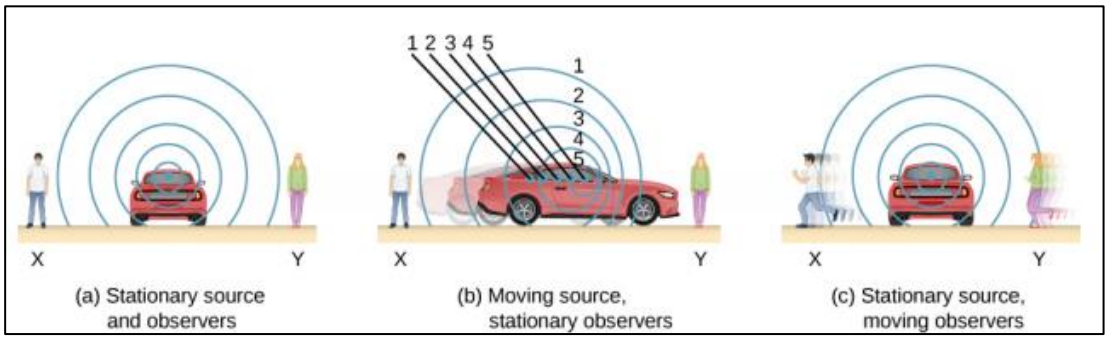


Figure 2.5: Sound waves behaviour for stationary and moving sources (LibreTexts, 2023)

Because the Doppler Effect makes it possible to detect and analyse object motion, it is particularly important in ultrasonic sensing. The frequency of the reflected waves varies with the distance of the moving object from the ultrasonic sensor when high-frequency sound waves bounce off it. By identifying this shift in frequency, the direction and velocity of the object's movement can be determined (Groeningen, Driessen, Söhl, & Vôute, 2018).

2.5.2. Mathematical formulation of Doppler frequency shift

The frequency shift (Δf) caused by the Doppler Effect can be mathematically described by the following equation (LibreTexts, 2023):

$$\Delta f = f' - f = \frac{v_{rel}}{v_s} \cdot f \qquad (1)$$

Where:

- f is the original frequency of the emitted wave.
- f' is the observed frequency after the Doppler shift.
- v_{rel} is the relative velocity between the sensor (or observer) and the object.
- vs is the speed of sound in the medium through which the wave is traveling.

2.5.3. Impact of Doppler Effect in Ultrasonic Sensing

The Doppler Effect in ultrasonic sensing serves the role of taking information about the movement of objects and the human body. For instance, a moving person walking toward the sensor would reflect ultrasonic waves with a positive Doppler shift-a positive frequency increase. On the other hand, if this person walks away from it, then the reflected waves would have a negative Doppler shift, which is a decrease in frequency. In the case of stationary objects, no Doppler shift occurs, and hence the reflected frequency is the same as the frequency that was transmitted. This effect becomes very useful in applications such as security systems when there will be a need to distinguish stationary from moving objects. The pattern of Doppler shift was analyzed as a significant means to bring out the difference between human movement and object movements in this research (Chen, Li, Ho, & Wechsler, 2006).

2.6. Machine Learning

Depending on the test case, there are various kinds of machine learning techniques, however they can be broadly divided into three categories: reinforcement learning, unsupervised learning, and supervised learning. A more comprehensive level classification of the machine learning models is displayed in Figure 2.6.

Therefore, we can define machine learning as supervised learning and unsupervised learning from a wider perspective as described below:

- **Supervised Learning**

  In supervised machine learning technique, a human expert uses training data to train the machine learning program that a fixed output is obtained for a particular kind of input. The model must be able to forecast future trends after a set of trained data. As a result, supervised learning is an approach of machine learning with the goal of learning a general rule that maps inputs to outputs. Classification and regression are two types of supervised learning algorithms (Aery & Ram, 2017).

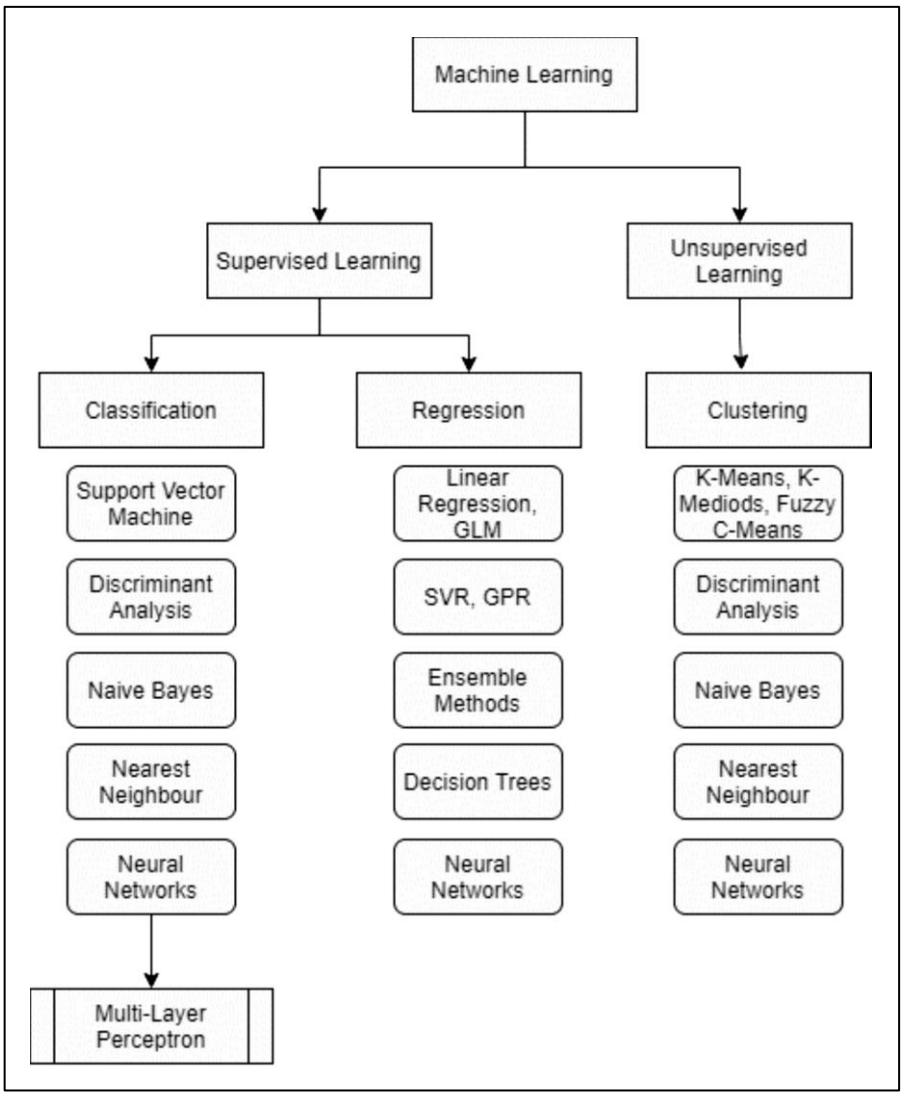


Figure 2.6: Types of Machine Learning Algorithms (Aery & Ram, 2017)

- **Unsupervised Learning**

  In unsupervised learning technique, the computer is educated on unlabeled data. The learning algorithm is given no labels thus leaving it on its own to find structure through the input. So, in this case, the need of an expert is eliminated to assist the computer for learning purposes. Instead, in this type of data-driven learning, unsupervised learning algorithms inform experts about the many types of patterns available in the data. These algorithms apply various methods and procedures to the incoming data to find patterns, mine for rules, organize and summarize data points, to derive valuable insights and better represent the data to consumers (Aery & Ram, 2017).

2.6.1. Confusion Matrix

A confusion matrix is a widely used Machine learning classification performance evolution technique. It is a representation of some parameters that lets one identify and analyze how well a classification model performs on a set of test data for which True or expected values are known. The confusion matrix also compares the actual target values to the machine learning model's prediction. It provides a comprehensive picture of how good the classification

model is working and different types of errors it makes an algorithm for summarizing the performance of a classification algorithm (Brownlee, 2020). The accuracy of classification alone can be misleading if there are an unequal number of observations in each class or if more than 2 classes are present in the dataset. The confusion matrix gives results of the classifier. It provides information regarding the prediction that the classifier makes or the error in classifying a particular object. A simple representation of a confusion matrix is shown in Figure 2.7.

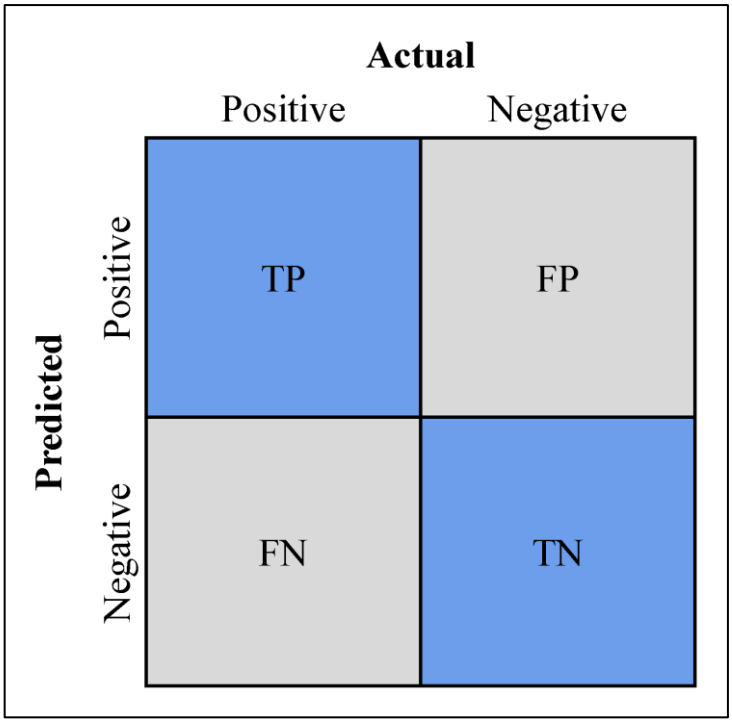


Figure 2.7: Confusion Matrix (Song, et al., 2024)

The features of a confusion matrix are as follows:

- A 2 x 2 table represents the classifier matrix for two prediction classes, a 3 x 3 table for three classes, and so forth. The matrix divides the entire number of forecasts into two dimensions, along with the predicted and actual values. Predicted values are those that the model expects, while actual values are the actual values for the given data.

- **True Positive (TP):** When both the projected value and the actual value are positive (Narkhede, 2018).

- **True Negative (TN)**: When both the predicted value and the actual value are negative (Narkhede, 2018)

- **False Positive (FP)**: Instances where the model incorrectly predicted the positive class. The is also known as Type 1 mistake (Narkhede, 2018)

- **False Negative (FN)**: Instances where the model incorrectly predicted the negative class. This is also referred to as the Type 2 mistake (Narkhede,

2018).

- **True Positive Rate (TPR), Sensitivity, Recall:** It is calculated by dividing the total number of positive samples by the number of correctly classified positive samples. Sensitivity is another name for recall, or the true positive rate. The model's ability to identify positive examples is indicated by a higher engagement score. Conversely, a lower recall score suggests that the model isn't particularly good at identifying affirmative cases (Evidently AI Team, n.d.).

$$\text{Recall} = \text{TPR} = \frac{\text{TP}}{(\text{TP} + \text{FN})} = \frac{\text{TN}}{\text{P}} \tag{2}$$

- **True Negative Rate (TNR), Specificity:** The specificity of the classifier is the system's ability to assess false (negative) tests, if they are truly wrong (Argoud, Sovierzoski, & Mendes de Azevedo, 2008).

$$\text{Specificity} = \text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} = \frac{\text{TN}}{\text{N}} \tag{3}$$

- **Positive Predictive Value (PPV), Precision:** By dividing the number of correctly identified positive samples by the total number of positive samples, precision is measured. Precision can be thought of as a metric for how accurate or good something is. Mathematically, precision can be represented as:

$$\text{Precision} = \text{PPV} = \frac{\text{TP}}{(\text{FP} + \text{TP})} \tag{4}$$

- **Accuracy:** Accuracy is a statistic that sums up how well a model performs. It is computed by the ratio of the number of right guesses and the total number of forecasts. In mathematical form, accuracy is represented as:

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FN} + \text{TN} + \text{FP})} \tag{5}$$

- **F-1 Score:** The F1-score is a measurement of a test's accuracy in binary categorization analysis. It is indeed measured using the test's precision and recall, with precision equaling the true positives results divided by number of favorable findings, including those that were incorrectly identified, and recall equaling the number of true positive results divided by the total number of specimens that is identified as positive (Wood, n.d.).

$$\text{F1 score} = \frac{(2\text{TP})}{(2\text{TP} + \text{FN} + \text{FP})} \tag{6}$$

One way to assess a model's accuracy on a particular dataset is to look at its F-score, also called the F1-score. Examples are classified as either "positive" or "negative" using binary classification techniques. Integrating the model's precision and recall can be done with the F-score, which is the geometric mean of the accuracy. F-score is a widely used metric for assessing various machine learning models, especially in natural language processing, and information retrieval systems, such as browsers (Wood, n.d.).

Accuracy is a measure of overall model performance, while precision and recall provide information about how well the model performs in predicting positive instances and identifying all positive instances, respectively. In the event that the dataset is unbalanced, the F1-score offers a reasonable balance between recall and accuracy.

2.6.2. Multi-Layer Perceptron Classifier

Pattern categorization is a vital component in many data interpretation operations since it entails assigning an object to one of several pre-specified categories or classes. To alter parameters, it's typical to split the training set in half and create a validation set. Cross-fold validation can be utilized when the number of examples is limited (Windeatt, 2008).

The multi-layer perceptron (MLP) is a feed forward neural network augmentation. The input layer, output layer, and hidden layer are the three types of layers represented in Figure 2.8. The input signal to be processed is received by the input layer. The output layer is responsible for tasks such as prediction and categorization (Abirami & Chitra, 2020).

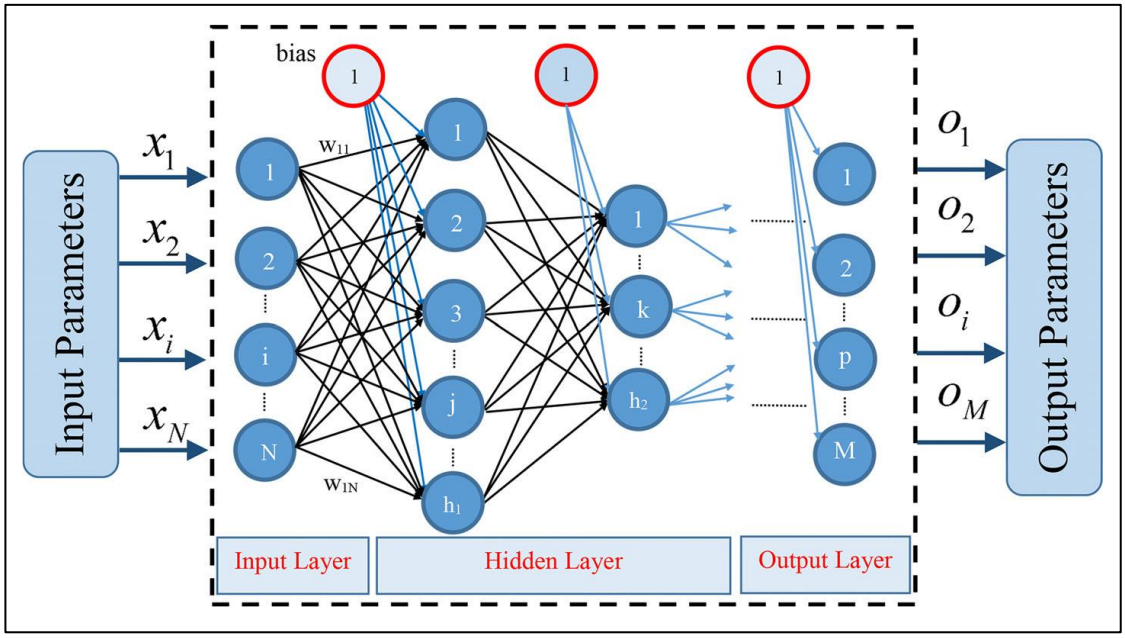


Figure 2.8: Schematic of a Multilayer Perceptron Neural Network (Nematollahi & Khaneghah, 2019)

The true computational engine of the MLP is an arbitrary number of hidden layers inserted between the input and output layers. In an MLP, data flows from input to output layer in the forward direction, like a feed forward

network. The back propagation learning algorithm is used to train the neurons in the MLP. MLPs can tackle issues that are not linearly separable and are designed to approximate any continuous function. Pattern categorization, recognition, prediction, and approximation are some of MLP's most common applications (Abirami & Chitra, 2020).

The perceptron is fed 'n' inputs or features ($x = x_1, x_2, x_3, x_4, \ldots, x_n$) and each of these features can be linked with its weight. The input has to be strictly numeric in nature. The input function 'u' gets the features as an input, to compute the weighted sum (Menzies, Kocagüneli, Minku, Peters, & Turhan, 2015).

$$u(x) = \sum_{i=1}^{n} w_i \, x_i \tag{7}$$

The output of the perceptron is produced by passing the result of this calculation onto an activation function (Menzies, Kocagüneli, Minku, Peters, & Turhan, 2015). The activation function is a step function in the original perceptron. There are different types of activation functions to perform complex computations in the hidden layer. Such as sigmoid, tanh, SoftMax and Rectified Linear Unit (ReLU) (Nicholson C. V., n.d.). MLPs can approximate any continuous function, and they achieve this by merging several neurons organized in at least three layers. The input layer distributing the characteristics of the input to the first hidden layer. Subsequently, the other hidden layers receiving the output of each perceptron from the previous hidden layer and finally one output layer for each perceptron delivering its output to the last hidden layer (Goyal, 2024). Assuming w denotes the vector of weights, x is the vector of inputs, b is the bias and Φ is the activation function for the i^{th} neuron, and the output is y . The output of each neuron in MLP can be defined by the following equation (Menzies, Kocagüneli, Minku, Peters, & Turhan, 2015) :

$$y = \left(\sum_{i=1}^{n} w_i x_i + b \right) = \varphi(w^T x + b) \tag{8}$$

Multi-Layer Perceptron classifiers are often applied to supervised learning problems. They train on a set of input-output pairs and learn to model the correlation between inputs and outputs. To train the algorithm MLP adjust the weights through a technique called 'Backpropagation' (Menzies, Kocagüneli, Minku, Peters, & Turhan, 2015). An in-depth overview of the classifier function is shown in Figure 2.9.

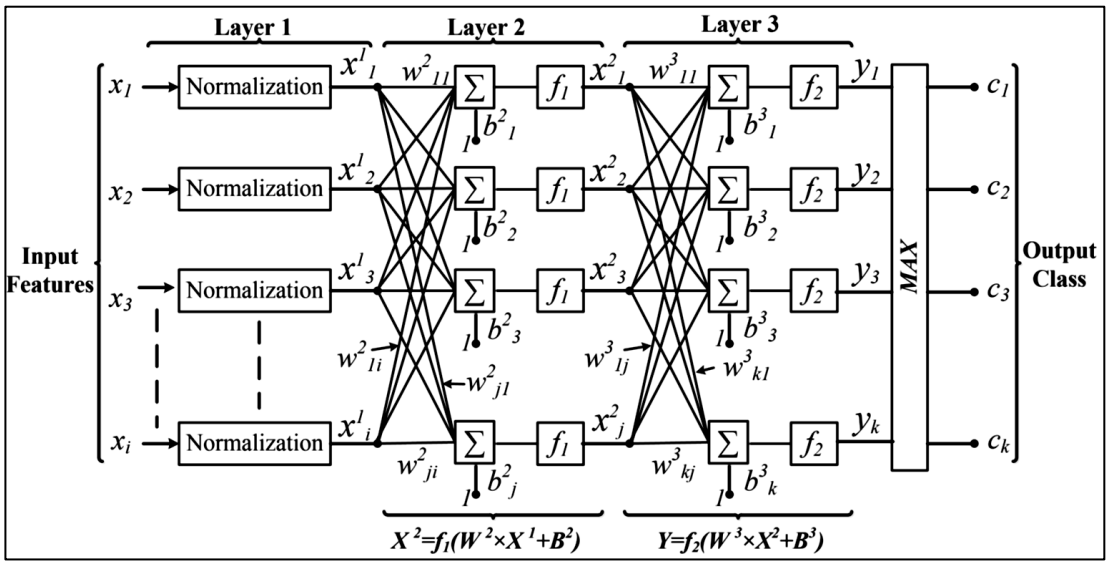


Figure 2.9: In-depth view of the MLP Classifier (Gaikwad, Tiwari, Keskar, & Shivaprakash, 2019)

The idea of the backpropagation algorithm is, based on error (or loss) calculation, to recalculate the weights array w in the last neuron layer, and proceed this way towards the previous layers, from back to front, that is, to update all the weights w in each layer, from the last one until reaching the input layer of the network, for this doing the backpropagation of the error obtained by the network. In other words, we calculate the error between what the network predicted to be and what it was indeed, then we recalculate all the weights values, from the last layer to the very first one, always intending to decrease the neural network error. It consists of the following parts (Leite, 2018):

- Initialize all the weights with small random values (Leite, 2018).
- Enter input into the network, then compare the resultant value to the intended output value to determine the error function's value. We are aware of the value of the right response in advance since the learning process is supervised. Differentiability of the error function is crucial ((Leite, 2018).
- The error function's gradients with regard to each weight are computed in order to reduce the error. Calculus taught us that the gradient vector shows the direction of a function's greatest gain; all we have to do is look in the opposite direction of the gradient vector to shift the weights in the direction of the error function's greatest reduction (Leite, 2018).

Since the gradient vector has been calculated, each weight is updated in an iterative way, and we keep recalculating the gradients at the beginning of each training iteration step, until the error becomes lower than a certain established threshold, or the maximum number of iterations is reached, when finally, the algorithm ends and, hopefully, the network is well trained (Leite, 2018).

2.6.3. Convolutional Neural Networks

Neural networks that can classify photos (i.e., label what they see), group images based on similarities and identify objects within scenes are known as convolutional neural networks. Convolutional networks are like the way neurons in the human brain connect to one another and are modelled after the

structure of the visual cortex. Additionally, convolutional networks are employed in more routine but significant commercial tasks, such as optical character recognition (OCR), which digitizes text and permits natural-language processing on Analog and handwritten documents, where the images stand in for translated symbols (Nicholson C. , 2020). Figure 2.10 shows a typical representation of CNN.

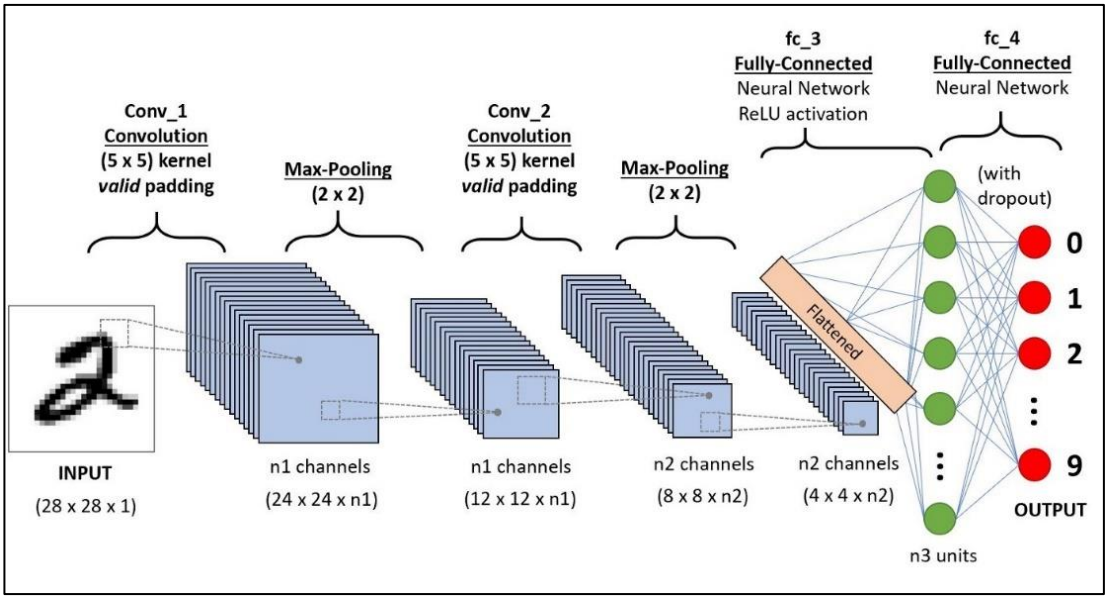


Figure 2.10: Figure showing the schematic representation of a CNN (Pranshu, 2023)

CNN is different from a normal neural network in terms of weight sharing and signal transmission between neurons and layers. CNN employs backpropagation to transmit signals backward for training, while traditional neural networks broadcast information in a single direction along the input-output channel, preventing signals from looping back into earlier layers (a technique known as feed-forward). Conventional networks need all neurons to be fully connected at all times, which leads to too complex network topologies. Consequently, the cost of complexity increases as the network is trained with large datasets because of storage constraints for performing calculations among all neurons and storing all parameters.

CNN has a special structure of partial connections in addition to complete connections, which reduces calculation time. Stated differently, each neuron is connected according to how dependent it is on the subsequent layer (Zheng , 2018). Only in the Receptive Field, a tiny portion of the visual field, can a single neuron react to inputs. To fill the entire visual field, a number of related fields can be piled on top of one another. A CNN model typically has three layers: a fully connected layer, a pooling layer, and a convolutional layer.

- **Convolutional Layer:** Convolution is basically the process of extracting distinctive features from the input, which consists of lower-layers and higher-layers generally. The lower layers extract low-level characteristics such as edges, lines, and corners. On the other side, the upper layers extract high-level features that are more abstract. For example, if the convolutional layer has H kernels and the input picture is N*N, the size of each kernel is

K*K. With H kernels, each kernel operates on the input individually, and each kernel creates one output feature. The result will generate H features on its own. Once the top-right corner has been reached, the kernel is shifted one element downward, and then from left to right, one element at a time. The same operation is repeated till the bottom right corner of the image is reached by the kernel (Zheng , 2018). Figure 2.11 shows a kernel movement downwards from left to right. Convolutional filters are convolved with the preceding layer to create feature maps in the convolutional layer.

- **Subsampling/Pooling Layers:** The pooling or subsampling layer is normally introduced after the convolutional layer, which aims to down sample the feature maps. Therefore, in each channel, the resultant features will be more resistant against noise and distortion.

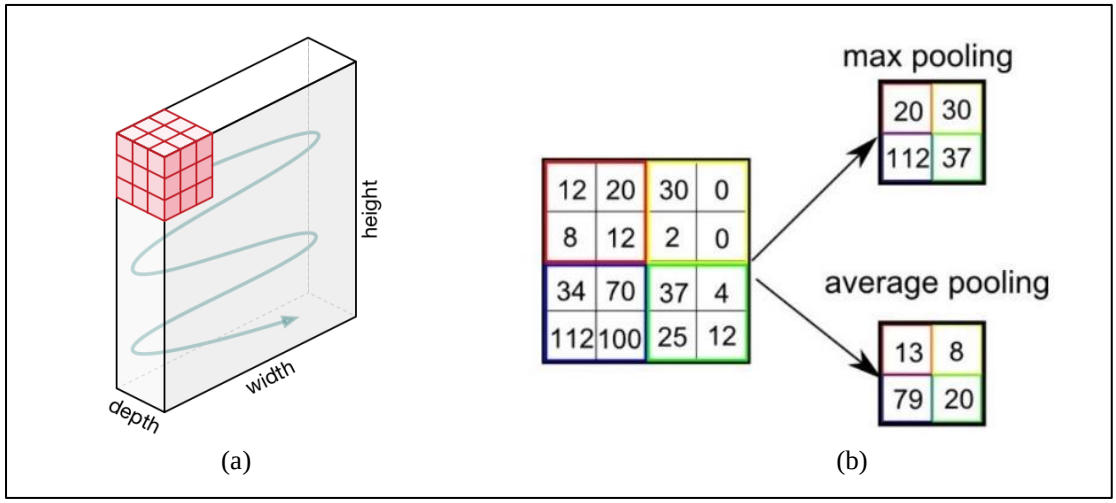


Figure 2.11: Figure (a) shows the movement of Kernel and (b) showing Max and Average Pooling (Saha, 2018)

There are many ways to do pooling; two of the common techniques are max and average pooling, as shown in Figure 2.11 below. The input is separated into two-dimensional zones that do not overlap in both scenarios (Zheng , 2018). Max Pooling yields the highest value from the region of the image that the kernel covers. Average Pooling yields the average of all values from the Kernel-covered portion of the image (Saha, 2018).

- **Mathematical Representation of Convolution and Max Pooling:** The whole multiplication and matrix input to the kernel form the convolution operation, and every movement of kernel depends on the stride value which is predetermined. The Convolutional formula determined in (Bachtiar & Adiono, 2019) is:

$$G[m,n] = (f * h) = \sum_j \sum_k h[j,k]\, f[m-j, n-k] \qquad (9)$$

where, $G[m, n]$ is the Convolutional Result, f is the Input, h is kernel, m is row number, n is column number, j is the shifting row and k is the shifting column as represented in Figure 2.12 and Figure 2.13. Max pooling can be calculated as below (Bachtiar & Adiono, 2019):

$$\alpha\,[m, n] = \max\left(\{\sum_j \sum_k f[m - j, n - k]\}\right) \quad (10)$$

where, $\alpha\,[m, n]$ is the maxpool result and $f\,[m, n]$ is the matrix input.

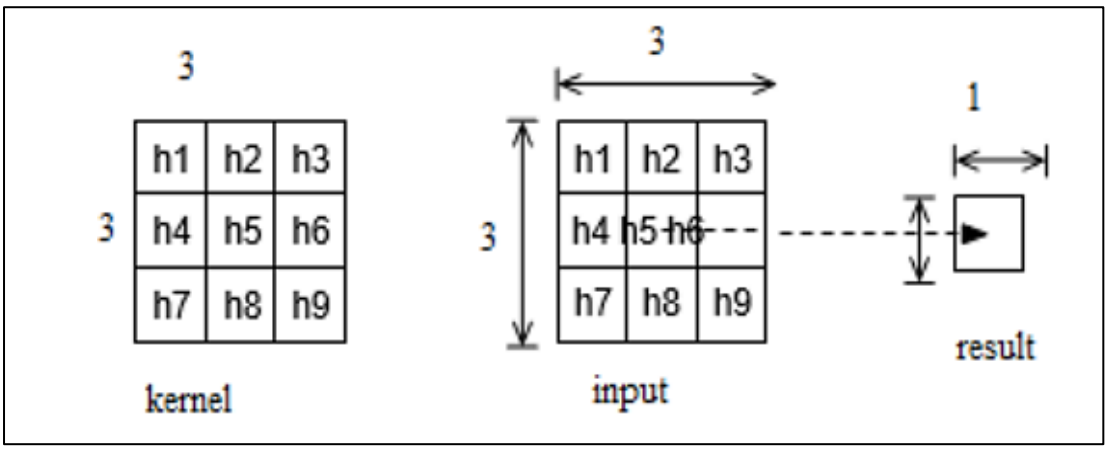


Figure 2.12: Figure shows Kernel, Input, and Convolutional Result (Bachtiar & Adiono, 2019)

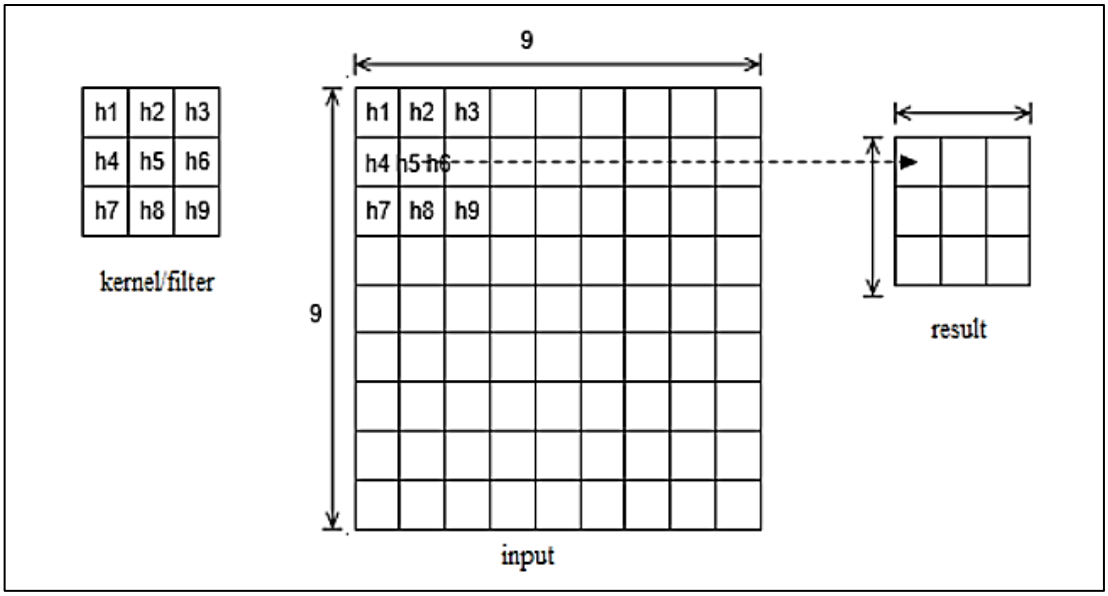


Figure 2.13: Figure shows Kernel, Matrix Input, and Maxpool Result (Bachtiar & Adiono, 2019)

- **Fully Connected Layers:** The fully connected layer of CNN is analogous to an ANN. The previous layer's feature maps are flattened into a vector to generate the first layer of this fully connected component. Further fully connected hidden layers are present after the first fully connected layer. The last layer, also called the output layer of a CNN is made up of neurons in the same number as the classes which are to be predicted. The nodes in CNN's output layer represent the classes that it has calculated. The disparity between the class estimated by CNN and the true class is specified by a loss function. The settings of CNN are altered to minimize this loss during the training phase. The loss functions are defined as the mean squared error,

SoftMax cross entropy and sigmoid cross entropy loss (Saha, 2018). Figure 2.14 shows the typical architecture of a fully connected layer.

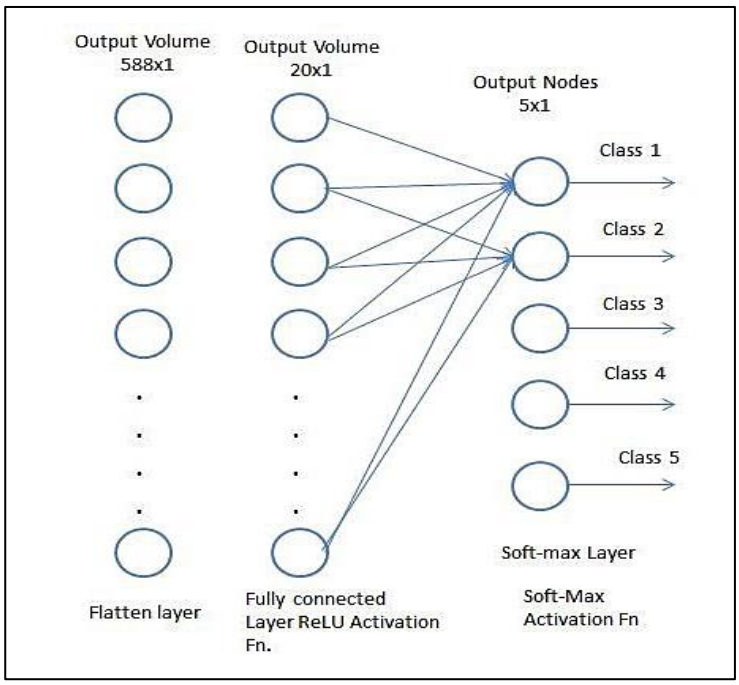


Figure 2.14: Figure shows a typical Fully Connected Layer (Saha, 2018)

2.7. Signal Processing Techniques

Processing the raw data coming from the ultrasonic sensors is essential in transforming raw data into useful information for further analysis. The current section outlines the most relevant techniques applied in this study: signal processing, such as windowing, autocorrelation, FFT, together with the basic mathematical concepts of feature extraction, including bandwidth, entropy and spectral centroid. The data from the ultrasonic sensor consisted of raw ADC data. For meaningful information, signal processing needs to be performed on raw ADC data. The idea here is to carry time-domain data into the frequency domain to see characteristics such as Doppler shifts and spectral patterns.

Signal processing methods such as autocorrelation, windowing, and Fourier transforms are essential for analysing signals, especially in the context of moving objects, where Doppler shifts in frequency can provide valuable information about relative motion (Oppenheim & Schafer, 2010).

2.7.1. Autocorrelation in Discrete-Time Signal Processing

A fundamental idea in signal processing, autocorrelation is used to examine the structure and recurring patterns in signals. It gauges how closely a signal resembles a delayed (or time-shifted) version of itself. This method improves signal analysis prior to frequency-domain transformations, including Fast Fourier Transform (FFT) analysis, by recognizing periodic components and lowering noise (Oppenheim & Schafer, 2010) (Proakis & Manolakis, 1996).

- **Definition of Autocorrelation:** For a discrete-time signal x[n] , the autocorrelation function Rx[k] is defined as the sum of the products of the signal values at each point with the values of the signal at a shifted (lagged) position. Mathematically, the autocorrelation function for lag *k* is defined as:

$$R_{\dot{x}}[k] \sum_{n=-\infty}^{\infty} x\,[n] \cdot x[n+k] \tag{11}$$

where *x[n]* is the discrete-time signal, *k* is the time shift or lag and $R_x[k]$ is the autocorrelation function at lag *k* (Proakis & Manolakis, 1996).

- **Benefits of using Autocorrelation**: This autocorrelation becomes a crucial preprocessing step before the signal's transformation into the frequency domain. It enhances periodic elements and dampens the noise so that further analysis, using FFT or other methods in the frequency domain, will be far more effective and clear in output (Oppenheim & Schafer, 2010).

2.7.2. Windowing and Hamming Window in Signal Processing

Windowing is a technique used in discrete-time signal processing to reduce spectral leakage that may result from truncating a signal to a finite length before operations such as the Fast Fourier Transform are applied. This is because the time domain truncation of a signal has a non-ideal frequency domain representation. Allowing for the application of a window function has this effect of smoothing the edges of the signal, thereby reducing huge discontinuities at the boundaries (Oppenheim & Schafer, 2010).

For a discrete-time signal *x[n]*, windowing is performed by multiplying the signal with a window function *w[n]*, producing the windowed signal $x_w[n]$ as:

$$x_w[n] = x[n] \cdot w[n], \qquad n = 1,2,3, N-1 \tag{12}$$

where, *x[n]* is the original signal, *w[n]* is the window function, $x_w[n]$ is the windowed signal and N is the total number of samples.

There are numerous window functions available, including the Hanning, Blackman, and Hamming windows. The Hamming window is a widely used technique in signal processing because it can effectively balance the trade-off between side lobe attenuation and main lobe width (Proakis & Manolakis, 1996).

- **The Hamming Window**: The Hamming window is a specific type of window function that smooths the edges of the signal to minimize spectral leakage. The mathematical definition of the Hamming window is given as:

$$w[n] = 0.54 - 0.46\,cos\left(\frac{2\pi n}{N-1}\right), \qquad 0 \le n \le N-1 \tag{13}$$

where, N is the total number of samples in the signal and w[n] represents the value of the window at the nth sample.

The Hamming window provides a gradual tapering effect on the signal, reducing the discontinuities that would occur at the edges if no window was applied. This window balances the trade-off between frequency resolution and leakage, making it effective in scenarios where moderate performance in both areas is required (Oppenheim & Schafer, 2010).

- **Benefits of applying Hamming Window**: The Hamming window reduces discontinuities at the boundaries of the truncated signal; hence, there is minimal leakage of energy into adjacent frequency bins during the FFT process. This, therefore, means accurate spectral representation, particularly for signals that are not strictly periodic (Proakis & Manolakis, 1996). The aim of this analysis was to determine the Doppler shifts and other frequency-related features; hence, the use of the Hamming window guarantees that these frequency components will be accurately represented. The FFT is then performed after windowing to transform the signal from the time domain into the frequency domain, where such features as spectral centroid and Doppler shifts become clearly observable (Proakis & Manolakis, 1996).

2.7.3. Fast Fourier Transform (FFT) in Signal Processing

The Fast Fourier Transform, or FFT, is an extremely efficient algorithm that calculates the DFT of a signal. It is a method through which a time-domain continuous signal is converted to its equivalent frequency-domain representation, consisting of the magnitudes of the frequency components in the signal. FFT forms the basis for ultrasonic signal processing, enabling it to determine the frequency shift through the Doppler effect, which in turn can identify whether such frequency shift is due to moving humans or objects. It provides the frequency-domain data that are to be used for extracting critical features such as centre frequency, bandwidth, and Doppler shift.

- **Discrete Fourier Transform (DFT)**: The DFT of a discrete-time signal x[n] with N samples is defined mathematically as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}nk}, \qquad k = 0, 1, 2, .., N-1 \qquad (14)$$

where, X[k] is the DFT of the signal, representing the magnitude and phase of the frequency component at index k, x[n] is the input signal in the time domain, N is the total number of samples in the signal, k represents the frequency index (or bin), and $e^{-j\frac{2\pi}{N}nk}$ is a complex exponential representing the basis functions for frequency decomposition (Oppenheim & Schafer, 2010).

- **Fast Fourier Transform**: The Fast Fourier Transform (FFT) is an optimized algorithm that significantly accelerates the computation of the DFT. By leveraging the symmetry and periodicity properties of the DFT, the FFT reduces the number of computations needed to transform the time-domain signal into the frequency domain.

The FFT is particularly suitable for large datasets, such as the Analog-to-Digital Converter (ADC) data collected by ultrasonic sensors. In our case, after pre-processing the ADC data using autocorrelation and windowing, the FFT is applied to obtain the frequency spectrum. This frequency spectrum reveals critical information about the periodic and non-periodic components of the signal, especially those introduced by object or human movements.

The FFT computation for a signal with N points can be described in its simplest form as a recursive decomposition of the DFT into smaller DFTs, such that:

$$X[k] = X_{even}[k] + X_{odd}[k] \cdot e^{-j\frac{2\pi}{N}k} \quad (15)$$

where, $X_{even}[k]$ is the FFT of the even-indexed samples of the signal, $X_{odd}[k]$ is the FFT of the odd-indexed samples and k represents the frequency bin (Proakis & Manolakis, 1996).

- **Benefits of FFT**: FFT makes frequency analysis fast and efficient for huge datasets; hence, it is suitable for real-time applications like the detection and classification of motion using ultrasonic sensors. Also, periodic components in the signal are brought forth by FFT, including those introduced by motion-like walking or object movement, which is critical in identifying Doppler shifts. FFT provides the data required to extract features that can be used to classify the signal. These features are being used in this research to distinguish between human and object movements-whether stationary or moving.

2.7.4. Spectral Features in Signal Processing

In this study, some spectral features are extracted from the frequency-domain representation to analyze the differences between human and object movements. These features give meaningful insights into the characteristics of the signal to identify the difference between different motion patterns. Among the key spectral features, the applied center frequency, spectral bandwidth, spectral entropy, Doppler shift, and energy distribution are very crucial for this study. These features are derived from the Fast Fourier Transform FFT of the windowed auto-correlated ADC data.

- **Center Frequency**: The location of the spectrum's "center of mass" is indicated by a measurement known as the center frequency. To put it simply, it is the mean frequency divided by the spectrum's magnitude. A crucial component utilized in many different applications, including audio processing and motion analysis, the spectral centroid is frequently connected to the perceived "brightness" of a signal. From a mathematical definition, it is:

$$Center\ Frequency = \frac{\sum_{k=0}^{N-1} f[k]\ .\ |X[k]|}{\sum_{k=0}^{N-1}|X[k]|} \quad (16)$$

where, f[k] is the frequency at bin k, |X[k]| is the magnitude of the spectrum at bin k and N is the total number of frequency bins (Oppenheim & Schafer, 2010).

- **Spectral Bandwidth**: Spectral bandwidth describes the spread of the spectrum around the spectral centroid. It provides an indication of how wide the range of frequencies is within the signal. A narrow bandwidth indicates a concentrated signal around a particular frequency, while a wide bandwidth suggests that the signal is spread over a larger range of frequencies. Spectral bandwidth is mathematically defined as:

$$Spectral\ Bandwidth\ = \sqrt{\frac{\sum_{k=0}^{N-1}(f[k]-C)^2 \cdot |X[k]|}{\sum_{k=0}^{N-1}|X[k]|}} \tag{17}$$

Where C is the center frequency, f[k] is the frequency at bin k and |X[k]| is the magnitude of the spectrum at bin k. Spectral bandwidth provides insights into the spread of the frequency content, which can vary depending on the type of motion (e.g., walking, standing still). This concept is crucial for understanding how different motion types cause different frequency distributions in signals (Proakis & Manolakis, 1996).

- **Spectral Entropy**: Spectral entropy is a measure of the randomness or unpredictability of the frequency content in a signal. It quantifies the distribution of energy across different frequencies and is commonly used in both signal processing and information theory. Claude Shannon's entropy model is adapted to quantify the disorder in the spectrum:

$$Spectral\ Entropy\ = \ -\sum_{k=0}^{N-1} P[k] \cdot \log_2(P[k]) \tag{18}$$

where, P[k] is the normalized power spectrum at bin k, defined as

$$P[k] = \frac{|X[k]|^2}{\sum_{k=0}^{N-1}|X[k]|^2} \tag{19}$$

Spectral entropy is particularly useful for distinguishing between more predictable signals (e.g., a stationary object) and more complex signals (e.g., a moving human). This formulation is inspired by Shannon (1948), where entropy is used to quantify information.

Chapter 3

Literature Review

3.1. Doppler Effect in Autonomous Sensing

The Doppler effect is a phenomenon wherein frequency changes with the relative speed between a source and an observer; it finds essential applications in detecting motions for autonomous vehicles. It has seen wide applications in radar and ultrasonic sensors for moving object detection and the analysis of variations in speed. Doppler shift research has shown that such frequency changes can provide significant information about velocity magnitude and direction, both critical in autonomous driving systems for the execution of obstacle avoidance or pedestrian detection tasks.

In the context of pedestrian classification, Doppler shifts enrich the activity by changing the spectral characteristics of the received signal depending on the direction (towards or away from the sensor) and speed. To be sure, knowledge of such variations enhances the ability of a vehicle to differentiate between stationary and moving objects, including pedestrians, whereby increasing the robustness of autonomous systems.

3.2. Ultrasonic Sensing and its Applications in Vehicle Safety

With their low cost, small size, and good functionality in low-speed contexts such as parking, ultrasonic sensors have been widely used in autonomous vehicles for close-range detection. In this respect, research has established that the sensor is especially reliable for the detection of objects within a range of a few meters, a sufficiently reasonable range for effective pedestrian detection in urban environments or when parking (Yeong, Velasco-Hernandez, Barry, & Walsh, 2021). These sensors, however, suffer from a great deal of disadvantages, including susceptibility to environmental noise, limited range, and reduced effectiveness at high speeds. These limitations also seem to extend in the work of authors such as (Hyun , Jin, Park, & Yang, 2020) on the use of ultrasonic sensors within dynamic environments, for example, their limited detection ranges and susceptibility to multipath reflection. The study also showed how such limitations in the use of ultrasonic sensors can be minimized through the integration of ultrasonic sensing with sophisticated signal processing techniques and Doppler-based analysis.

In the context of pedestrian classification, Doppler shifts enrich the activity by changing the spectral characteristics of the received signal depending

on the direction (towards or away from the sensor) and speed. To be sure, knowledge of such variations enhances the ability of a vehicle to differentiate between stationary and moving objects, including pedestrians, whereby increasing the robustness of autonomous systems.

3.3. Doppler Effect in Human and Object Classification

One of the most fundamental approaches to the distinction between human and static objects is Doppler-based classification in AV technology. (Hyun , Jin, Park, & Yang, 2020) identified that human movement due to limb motion generates particular Doppler signatures, quite different from those by rigid objects such as poles or vehicles. This could further classify pedestrians in a radar-based system because human gait may introduce complex frequency patterns which are quite hard to generate with static or purely mechanical objects.

This research also points out that, depending on the dynamics of motion, Doppler shifts can vary, and humans walking make their Doppler variations complex, which ultrasonic sensors take advantage of to develop ways whereby AV systems can detect and classify pedestrians with greater accuracy. The principle hereby forms a foundation in this study for applying similar techniques to ultrasonic data and then evaluating their effectiveness for pedestrian classification, despite the reduced Doppler sensitivity of ultrasonic sensors.

3.4. Machine Learning in Object Classification Using Ultrasonic Data

In particular, neural networks have performed very well in the object classification from ultrasonic sensor data. Traditional object classification in autonomous systems uses hand-engineered features and rule-based algorithms. However, most of the recent deep learning work has evidenced the power of the Multi-Layer Perceptron and Convolutional Neural Networks in modeling complex patterns across high-dimensional ultrasonic data.

3.4.1. Multi-Layer Perceptron (MLP) in Classification Tasks

The MLPs are feed-forward neural networks and perform very well for tasks that involve nonlinear relationships between data. A study illustrated that MLPs are capable of processing real-time human activity classification based on Doppler data from ultrasonic sensors. Simplicity in the MLPs allows computational efficiency that could be suitable for real-time processing where speed and resource optimization is vital (Gaikwad, Tiwari, Keskar, & Shivaprakash, 2019). Previous works involving applications of MLPs have also shown their prowess in processing frequency domain data for the pattern recognition task at hand. In their study Hyun et al. (2020), had applied MLPs to perform object classification based on Doppler signatures in radar data with very high performance, since the MLP learned intricacies of the spectral data.

In the context of this work, MLP was chosen for its balance between computational simplicity and the ability to capture frequency-domain patterns associated with Doppler shifts. Works such as Gupta et al. (2021) have further

validated MLPs on similar tasks of detecting objects in short range, thereby justifying the use of the same in pedestrian vs. object classification based on ultrasonic FFT data.

3.4.2. Convolutional Neural Network (CNN) for Signal-Based Classification

CNNs are specifically designed for hierarchical feature extraction from structured data, such as time-frequency transformations of signals. They work well in the analysis of spectrograms emanating from ultrasonic and Doppler-shifted signals, which enable the CNN to pinpoint key features enabling object classification by shape and motion. Authors Eisele et al. (2023) used a CNN to process ultrasonic data in the form of time-frequency images from raw signals in 2023. Data augmentation techniques, including rotation and scaling, further increased the robustness of the models, showing accuracies above 90% in controlled environments, while outdoors, the accuracy was about 66%, hence CNN has a very great potential for reliable object classification in AV systems (Eisele et al., 2023).

Recent developments have adapted CNNs-from their original inception in image recognition-to one-dimensional data signals, such as audio and radar signals. Works like De Jong Yeong et al. (2021) show CNNs to be powerful in the representation of local features in spectral data, hence fitting for applications with time-frequency transformations, such as Fourier Transforms used in Doppler-based detection. These have been proved better in pedestrian detection studies due to their convolutional layers, which capture the translational and spatial invariances in spectral data. CNNs also do very well in learning complicated Doppler shifts, as was done for radar data in distinguishing object types based on their movement signature in (Reda, El-Sheimy, & Moussa, 2023). CNN performance was compared in this study against MLP for the purpose of proving its robustness in cross-movement classification.

3.5. Challenges in Pedestrian Detection Using Ultrasonic Sensors

While ultrasonic sensors are effective at short-range detection, their key detection of pedestrians poses greater difficulties. Poor resolution, along with high sensitivity to ambient noise, makes outdoor pedestrian detection among other objects extremely difficult. Indeed, work by Mubarak (2013) echoed these difficulties and demonstrated that ultrasonic sensors are not truly reliable in standing alone on an AV, with poor detection of oblique or small objects under variable environmental conditions.

Multi-sensor fusion and machine learning have been investigated as solutions to enhance pedestrian detection. By integrating data from radar, LiDAR, and ultrasonic sensors, AVs can compensate for individual sensor weaknesses, improving the classification of objects and pedestrians. In addition, CNNs applied Doppler-shifted ultrasonic data help capture specific movement patterns, such as human gait, which improves pedestrian detection accuracy

(Galvao, Abbod, Kalganova, & Palade, 2021). Such integrations form a robust framework for AVs, especially in dynamic environments where pedestrian movement is unpredictable.

3.6. Integrating CNNs and Doppler Analysis for Enhanced Classification

Very recently, a combination of CNNs with Doppler analysis has turned out to be a very effective strategy for improving the classification capability of AV objects. The effective fusion of feature extraction by CNNs and Doppler-based velocity measurement helps AVs achieve more improved accuracy in distinguishing stationary from moving objects across diverse motion scenarios. For instance, Eisele et al. (2023) proposed employing CNNs in analyzing Doppler time-frequency data, which resulted in substantial improvement in the accuracy of classification even in noisy environments that are challenging.

These further improvements in robustness could be achieved by adaptive learning techniques such as data augmentation and transfer learning, enabling CNNs to adapt dynamically to various pedestrian speeds and directions. Such models would serve AVs moving in urban environments where the nature of pedestrian movement is quite unpredictable. This further point out the adaptability of CNNs with Doppler data for their great potential in real-world AV systems requiring responsive and accurate object classification in real time.

Chapter 4

Methodology

This chapter outlines the methodology followed in this research, starting from data collection using ultrasonic sensors to the final classification between human and object movements using machine learning techniques. Each phase of the study was designed to process and analyze the ultrasonic signals, detect Doppler shifts caused by movement, extract spectral features, and apply machine learning models to classify the observed patterns.

4.1. General structure of the system

The general structure of the system used for data acquisition, signal processing, and machine learning classification is illustrated in Figure 4.1. The system is composed of two main components: Hardware and Software, both of which play crucial roles in capturing, processing, and analyzing the data for Doppler Effect analysis and training machine learning models to distinguish between human and object movements.

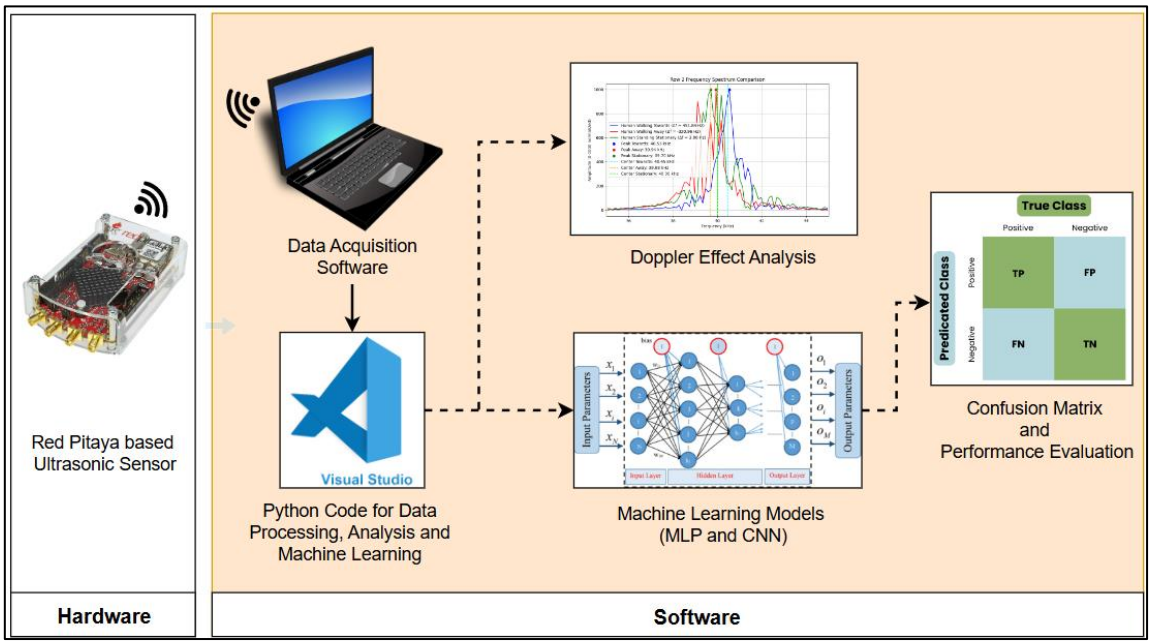


Figure 4.1: General architecture of the system

4.1.1. Hardware

The basic hardware includes a Red Pitaya-based ultrasonic sensor that would transmit the waves and capture back the reflection signals. It has a base operating frequency of 40 kHz. The movement dynamics of the target in the environment are reflected in the Analog-to-Digital Converted data captured by the Red Pitaya board. Captured ADC data is sent to the laptop through Wi-Fi connectivity between the sensor and the laptop through a data acquisition software, which will collect the raw data from the sensor for further processing.

4.1.2. Software

The final processing and analysis of the captured ADC are done with the following key software components that work in conjunction. The raw data acquired by an ultrasonic sensor based on Red Pitaya are collected using Data Acquisition Software, while the data is processed to make them ready for detailed analysis in the form of code in Python under a Visual Studio environment to study the Doppler Effect in detail. The dataset recorded in this setting serves as inputs to the machine learning models Multilayer Perceptron and Convolutional Neural Networks, which classify the movements into human or object categories. Performance evaluation will also be presented by the confusion matrix analysis, which will calculate some metrics like accuracy, precision, recall, and F1 score.

4.2. Experimental Environment and Setup

4.2.1. Physical Environment Setup

The project is conducted in the underground car parking of a modern university building to provide close to a real-world scenario of a car parking maneuver where either a human or object is approaching towards, moving away, passing laterally in front of the car or standing stationary. The Red Pitaya-based ultrasonic sensor was placed on the front bumper of a parked car approximately at a height of 40 cm from the ground to record measurements as shown in Figure 4.2. The sensor was pointed towards an empty driveway of about 4 m in width and 20 m in length to provide sufficient space for target human/object human. The laptop was placed on a side table for taking measurements. The Red Pitaya device is powered by connecting it to a power source via a charging cable. The ultrasonic sensor transmitted 40 kHz sound waves, and the reflections from both human and object movements were captured by the Red Pitaya's ADC (Analog-to-Digital Converter). The sensor was tested in various conditions, including human movement towards and away from the sensor, as well as stationary and sideways positions for both humans and objects.

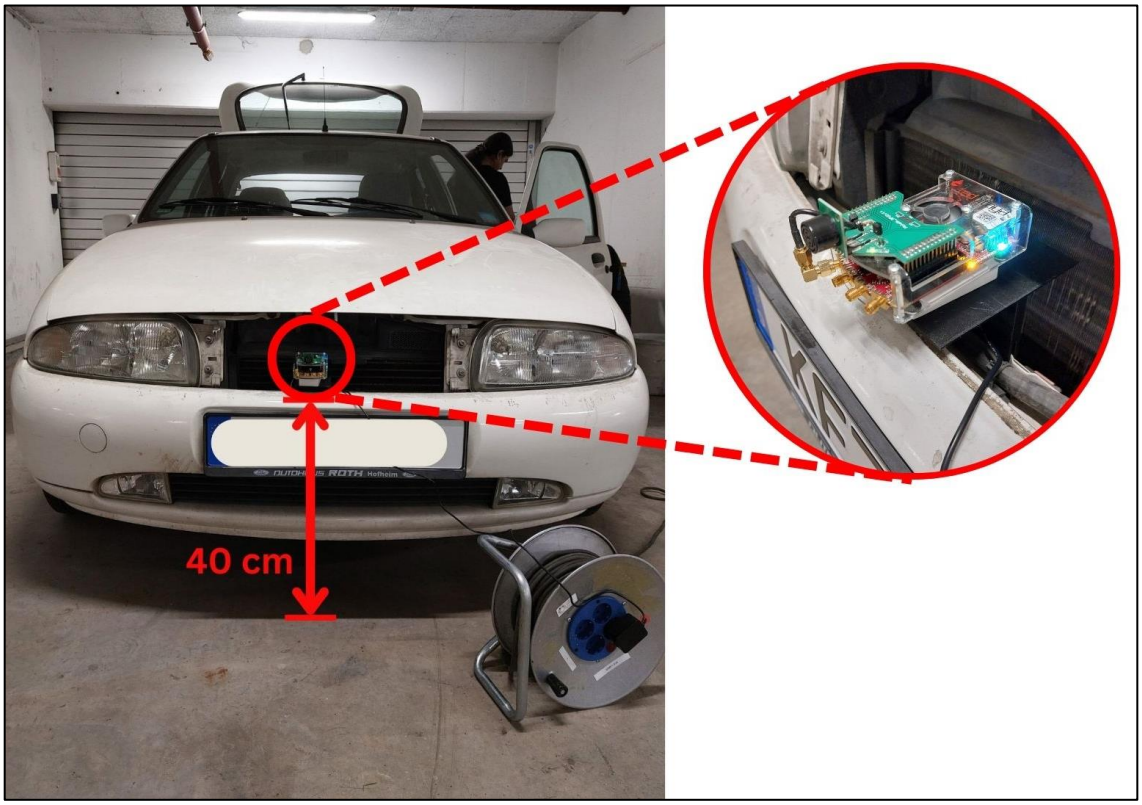


Figure 4.2: Figure of Front Bumper of car with Sensor Installed with marking showing height from ground.

Figure 4.3 depicts the actual context in which the research was conducted and displays the graphical representation of the environmental setup and specifications in details.

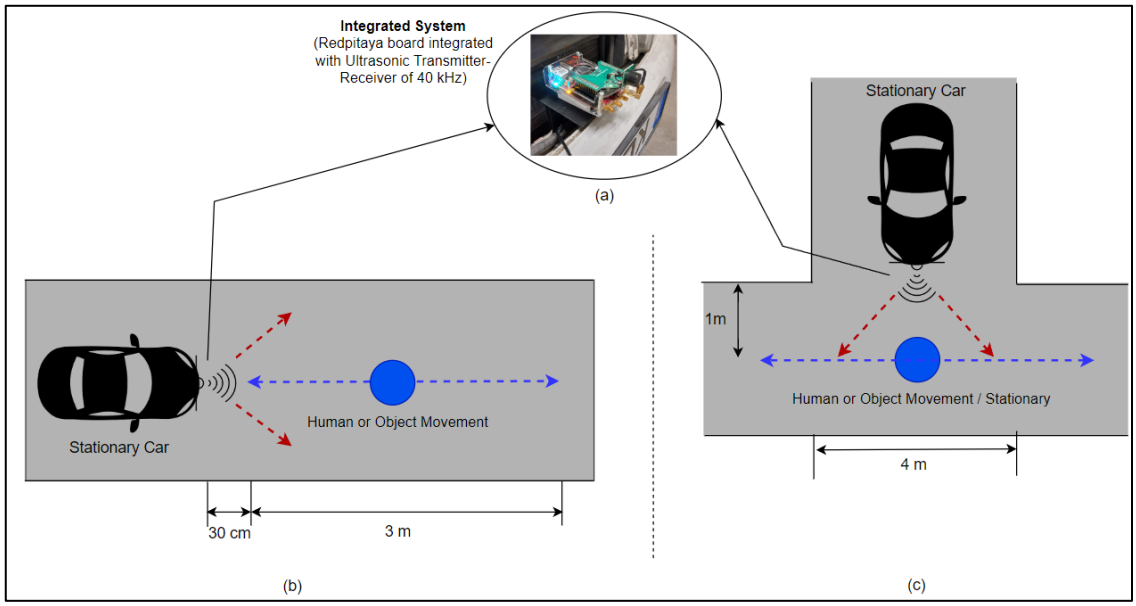


Figure 4.3: (a) Actual Red pitaya integrated Ultrasonic Sensor, (b) Representation of Towards and Away Movement Setup and (c) Sideways/Perpendicular Movement Setup

4.2.2. Software Environment Setup

Python 3.8 was used to implement the signal processing, feature extraction, and machine learning workflow in this project. Python's large libraries for data analysis, machine learning, and scientific computing made it the preferred language. Visual Studio Code (VS Code) 1.84.0 was used for both the development and execution of the code. It is an IDE that is lightweight, feature-rich, and well-integrated with Python environments. The complete execution of the project was done in windows laptop with windows version 11. Below are the steps followed to set up the Python software environment and the necessary tools used for this project:

1. **Setting up the virtual environment**

   a) First, a folder was created within the main project's folder with name 'Python' to store all code files.

   b) Within the 'Python' folder, the Python virtual environment was created using the following command:

   ```
   python -m venv <ve>
   ```

   c) This command creates a local environment where all required dependencies for the project are installed.

2. **Activating the Virtual Environment and installing dependencies**

   a) The virtual environment was activated using the below command:

   ```
   <ve>\Scripts\activate
   ```

   b) Once the virtual environment was activated, the following libraries were installed:

   ```
   pip install numpy
   pip install scipy
   pip install matplotlib
   pip install seaborn
   pip install pandas
   pip install scikit-learn
   pip install xlsxwriter
   ```

   c) **NumPy** was used for numerical operations, especially to handle large datasets of ADC data and matrix operations in FFT and autocorrelation calculations.

   d) **SciPy** provided signal processing capabilities such as autocorrelation and FFT transformations.

e) **Matplotlib** and **Seaborn** were essential for plotting various graphs, including amplitude vs. frequency plots, confusion matrices, and Doppler effect analysis.

f) **Pandas** was used for managing datasets, reading and writing Excel files, and organizing results into tables.

g) **scikit-learn** offered machine learning functionalities, including the implementation of models like MLP and CNN, and for calculating metrics such as accuracy, precision, recall, and F1 score.

h) **XlsxWriter** was used to export processed data and results, including the classification reports and confusion matrices, into Excel files for further analysis.

3. Saving and Managing Dependencies

1. After installing all dependencies, a requirements file was generated to document the project's dependencies. This file allows easy replication of the environment:

```
pip freeze > requirements.txt
```

2. This file includes all installed packages and their respective versions, ensuring the environment can be easily recreated using:

```
pip install -r requirements.txt
```

Below Figure 4.4 the actual python virtual environment which was setup using the above steps:

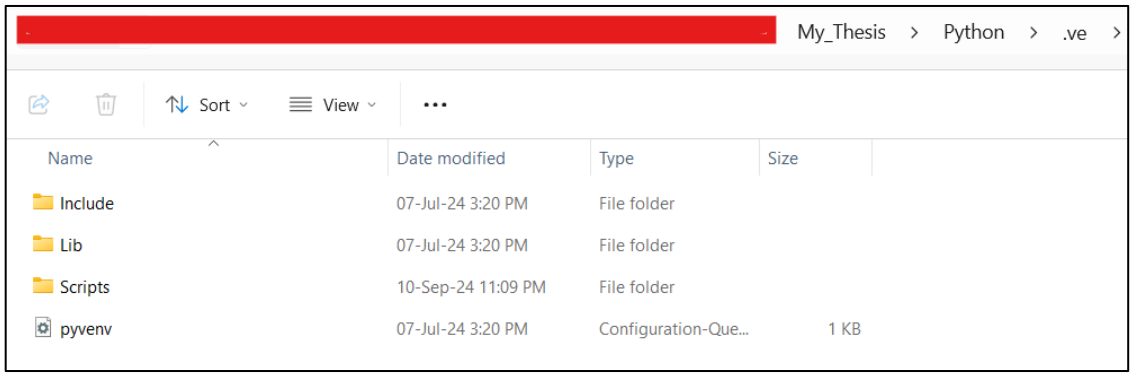


Figure 4.4: Folder of the Python Virtual Environment

This software environment setup was crucial in maintaining the reproducibility of the code and ensuring that all scripts ran smoothly across different systems or after any future modifications. The environment management through a virtual environment also simplified version control and allowed isolated development without interference from system-wide Python packages.

4.3. Data Capturing Process

For setting up the GUI software which is used for taking the measurements the following steps are involved:

- Once the UDP Client software is loaded and the wireless network credentials for the particular sensor (in our case Sensor 1) is provided then the green signal at bottom left indicates that connection to 'Sensor 1' is successfully established.

- The GUI saves the readings in a designated folder. The folder's or directory's full path is **'\\GUI_V0.23_2021-11-22\GUI_SW\save'**.

- Based on the instruction received on the socket, the C application executing in Red Pitaya transmits bytes of ultrasonic data. The "<ip_address_of_redpitaya:port_number>" address is where the Red Pitaya server is listening. **192.168.128.1** is the IP address of the Red Pitaya used and **61231** is the port number on which the server is listening.

- Red Pitaya's program transmits data in response to the following commands:
    1. "-a 1": The designated socket will be used to transmit ADC data to the client.
    2. "-a 0": Data transmission will stop until another instruction is received.
    3. "-f 1": The connected client will get the FFT data.
    4. "-f 0": "Stop sending data until you receive another command.

In this project the data was collected only ADC data was captured and saved in the form of .txt files with suitable naming conventions as described in later sections. The ADC data files captured from the ultrasonic sensor contained rows of discrete-time measurements representing the reflected ultrasonic waves. Each ADC file consisted of **16,384 samples** per row, which corresponded to the time window in which the sensor captured the reflected sound. Figure 4.5 shows a sample ADC file from the UDP client, with each row representing one instance of time-domain data capture. The first 16 columns of data in every data file are headers, which are used to record the measurement's setting. Table 4.1: Detailed description of ADC data file headersshows the detailed description of these headers.

1	64	32768	1	1	512	0	1953125	12	0.0	6404	2.12	0	0	V0.2	0	0	17 7 6
2	64	32768	1	1	512	0	1953125	12	0.0	6440	2.12	0	0	V0.2	0	0	-23 -14 6
3	64	32768	1	1	512	0	1953125	12	0.0	6428	2.12	0	0	V0.2	0	0	-82 -87 -72
4	64	32768	1	1	512	0	1953125	12	0.0	5852	2.02	0	0	V0.2	0	0	-15 -5 -2
5	64	32768	1	1	512	0	1953125	12	0.0	5014	1.88	0	0	V0.2	0	0	-4 3 10
6	64	32768	1	1	512	0	1953125	12	0.0	4879	1.86	0	0	V0.2	0	0	-11 3 6
7	64	32768	1	1	512	0	1953125	12	0.0	4829	1.85	0	0	V0.2	0	0	-24 -10 -1
8	64	32768	1	1	512	0	1953125	12	0.0	4690	1.82	0	0	V0.2	0	0	38 27 27
9	64	32768	1	1	512	0	1953125	12	0.0	4491	1.79	0	0	V0.2	0	0	-6 -3 -5
10	64	32768	1	1	512	0	1953125	12	0.0	4703	1.83	0	0	V0.2	0	0	-7 -8 -6
11	64	32768	1	1	512	0	1953125	12	0.0	4926	1.86	0	0	V0.2	0	0	-70 -62 -53
12	64	32768	1	1	512	0	1953125	12	0.0	4738	1.83	0	0	V0.2	0	0	10 74 16

Figure 4.5: Sample ADC data file obtained from UDP Client

Table 4.1: Detailed description of ADC data file headers

No.	Headers	Length (byte)	Value	Unit
1	Header length (count for fields)	4	64	-
2	Data Length	4	32768	ADC_DATA: 32768 FFT_DATA: 170
3	Class detected	4	1 or 2	1: Object 2: Human
4	Measurement type	4	0 or 1	0: FFT 1: ADC
5	Frequency resolution of a sampling time t depends on measurement type	4	512	Hz (1/s) or t (ns)
6	Normation	4	0,1,2	
7	Sampling frequency	4	1953125	Hz (1/s)
8	ADC Resolution	4	12	
9	Ambient temperature	4	-500…500	°C
10	Distance between sensor and first object(round-trip-time)	4	$0\text{-}2^{32}$	t (µs)
11	Time of flight of US in air for and backward--> distance to first object	4	$0\text{-}2^{32}$	t (µs)
12	FFT Window length	4		
13	FFT Window Offset (index --> freq_min_index)	4		
14	Software Version (RP) as a string	4	V0.2	
15	reserved1	4		
16	reserved2	4		
17	Data [...]	64		2 byte each

4.4. Experimental Cases

The experiments were divided into several cases, each designed to explore a specific type of movement. These cases included both human and object movements, such as **moving towards**, **moving away**, **standing still and moving sideways (perpendicular)**. Each case was stored in separate folders with suitable naming convention which is explained below: The folder structure is shown in Figure 4.6

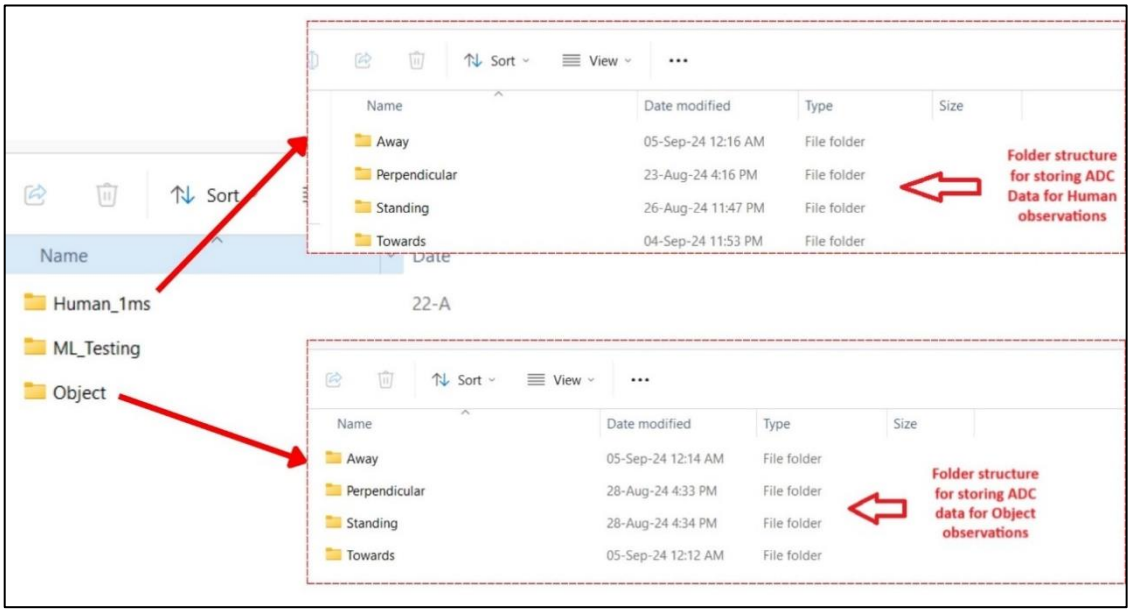


Figure 4.6: Folder structure for the collected Human and Object Data

Figure 4.7 and Figure 4.8 show the actual environment and experiment case for both Human and Object respectively.

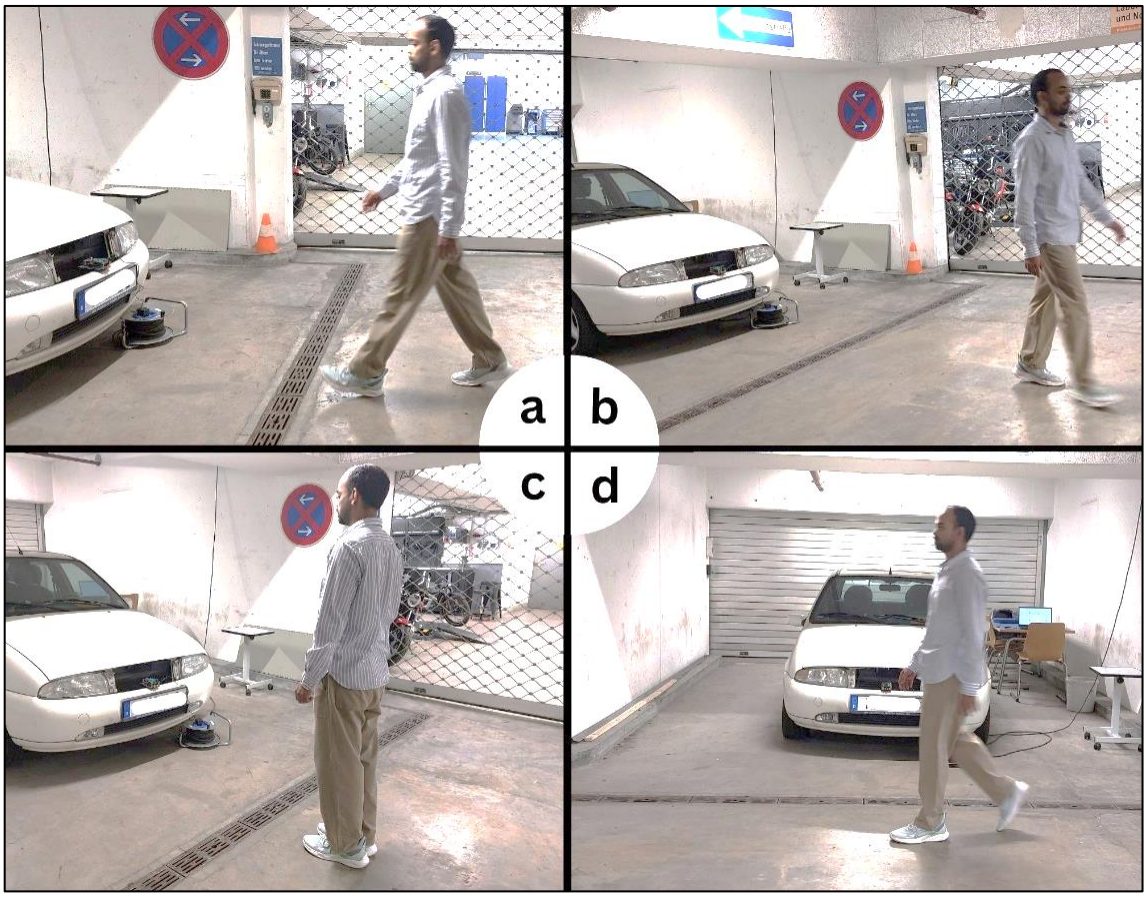


Figure 4.7: Four experimental cases for Human subject (a) Moving Towards, (b) Moving Away (c) Standing Stationary and (d) Moving Sideways/Perpendicular to the sensor

Figure 4.8: Showing the Flat metal sheet used as Object and the Roller stool to move the flat sheet for movement scenarios

1. **Human Walking Towards Sensor-** As the name suggests, in this case a human subject was walking towards the sensor at a natural walking speed. The distance of the walk was approximately 3 m from the sensor as shown in Figure 4.8. The subject walked from a distance of 3m to the closest distance of 30cm from the sensor and the measurement were recorded during this duration of walk only. The sensor was placed just above the number place of the car as described in previous sections. The naming convention used for saving ADC data files for this case is '**adc_HWT<Two-Digit Number>_**' where adc_HWT denotes the ADC data for **H**uman **W**alking **T**owards and the variable two-digit number denotes the file count.

2. **Human Walking Away from Sensor-** In this experiment, the same human subject walked away from the sensor starting from the closest distance of 30cm from the ultrasonic sensor till a distance of 3 m away from the it. During this case the human subject was facing away from the sensor while the sensor was recording the measurements till the duration of walk. The naming convention used for saving ADC data files for this case is '**adc_HWA<TwoDigitNumber>_**' where adc_HWA denotes the ADC data for **H**uman **W**alking **A**way and the variable two-digit number denotes the file count.

3. **Human standing Stationary –** In this case the human subject was standing stationary in front of the sensor at a distance of 1m from the sensor. Measurements were taken by the ultrasonic sensor to saved enough ADC data points for further analysis. The naming convention used for saving ADC data files for this case is '**adc_HS<Two-Digit Number>_**' where adc_HS denotes the ADC data for **H**uman **S**tationary and the variable two-digit number denotes the file count.

4. **Human Walking Sideways-** In this case the human subject was walking in a direction perpendicular to the sensor at a distance of 1 m in the transmitting direction as shown in experimental setup figure 2.2c. The sensor recorded measurements only during the 4m sideways walk of the

human subject and files were saved with the naming convention '**adc_HWP<Two-Digit Number>_**' where adc_HWP denotes the ADC data for **H**uman **W**alking **P**erpendicular and the variable two-digit number denotes the file count.

5. **Object Moving Towards Sensor**- In this case a flat smooth metal sheet was moved towards the sensor at a constant speed. The sheet was tied to a rolling stool and the stool was moved at a contact speed from a distance of 3m to a closest distance of 30 cm towards the sensor. During this period the sensor recorded the measurements. The naming convention to save the files for this case is '**adc_OMT<Two-Digit Number'**> which is similar to the case of human moving towards sensor.

6. **Object Moving Away**- In this case a flat smooth metal sheet was moved away from the sensor at a constant speed. The sheet was tied to a rolling stool and the stool was moved at a constant speed from the closest distance of 30cm to a distance of 3m away from the sensor. During this period the sensor recorded the measurements. The naming convention to save the files for this case is '**adc_OMA<Two-Digit Number'**> which is similar to the case of human moving away from sensor.

7. **Object Stationary –** In this case the stool with tied metal plate was held stationary in from of the sensor at a distance of 1m facing the sensor and measurements were recorded. The naming convention to save the files for this case is '**adc_OS<Two-Digit Number'**> which is similar to the case of human stationary.

8. **Object Moving Sideways –** In the case the metal sheet tied to a rolling stool was moved in a direction perpendicular to the sensor with the sheet still facing it at a distance of 1 m from the sensor. The sensor recorded measurements only during the 4m sideways movement of the metal sheet and files were saved with the naming convention '**adc_OMP<Two-Digit Number>_**' where adc_OMP denotes the ADC data for **O**bject **M**oving **P**erpendicular and the variable two-digit number denotes the file count.

For each of the above experiment cases, two datasets were recorded to find the cases in which Doppler effect can be better analyzed. Table 4.2 summarizes the specifications for both of the datasets for each of the experimental cases of human and object.

Table 4.2: Specifications of the collected datasets

Cases	**Naming Convention**	**Specifications**	**Dataset 1 Average Speed**	**Dataset 2 Average Speed**
Human Walking Towards	adc_HWT<Two-Digit Number>	Normal Human Walk wearing shoes with hands in motion from 3m to 30cm	1 m/s	1.28 m/s

Human Walking Away	adc_HWA<Two-Digit Number>	Normal Human Walk wearing shoes with hands in motion from 30 cm to 3m with back facing the sensor.	1 m/s	1.28 m/s
Human Stationary	adc_HS<Two-Digit Number>	Human Standing stationary at 100 cm from the sensor	NA	NA
Human Walking Sideways/ Perpendicular	adc_HWP<Two-Digit Number>	Normal Human walking, wearing shoes with hands in motion	1 m/s	1.28 m/s
Object Moving Towards	adc_OMT<Two-Digit Number>	The object used is a Metal Sheet placed with flat face in front of the sensor and moved on a roller stool from 3m to 30cm	1.12m/s	2.6 m/s
Object Moving Away	adc_OMA<Two-Digit Number>	The object used is a Metal Sheet placed with flat face in front of the sensor and moved on a roller stool from 30cm to 3m	1.12m/s	2.6m/s
Object Stationary	adc_OS<Two-Digit Number>	The object used is a Metal Sheet placed with flat face in place at 100 cm in front of the sensor	NA	NA
Object Moving Sideways/ Perpendicular	adc_OMP<Two-Digit Number>	The object used is a Metal Sheet moved on a roller stool across the sensor	1.12m/s	2.6m/s

4.5. Decoding of captured ADC data

First, the raw ADC data coming from the ultrasonic sensor were decoded and pre-processed in the first stage of the signal processing pipeline. Captured ADC data of the sensor are stored in text files, which include 16 columns or 64 bytes of header data before the actual ADC data. The creation of a Python script that cycles through the raw ADC files, removing the header columns and saving the cleaned data into another format, did isolate useful data. This was a necessary step in the preprocessing to extract real ADC values and render the data appropriate for subsequent stages of processing such as applying an autocorrelation, hamming windowing, and performing FFT. The following steps outline this process:

1. **Directory Structure** - The input ADC data files were organized in a specified directory, with a separate output directory created to store the extracted, cleaned ADC data files.

2. **Reading the Files** - For each file in the input directory, the script opened the file and read all its lines. Each line in the file represented one row of ADC data, which initially contained 16 columns of header information followed by the actual ADC values.

3. **Removing Header Information** - Using Python's string manipulation capabilities, the first 16 columns of each row were removed, leaving only the ADC values for each sample.

4. **Saving the Cleaned Data** - After the header information was removed, the remaining ADC values were saved to a new file. The output files were stored in a different directory, maintaining the original file names but appending the suffix "_extracted" to indicate that they had been processed. Figure 4.9 shows the ADC data files after removing the headers.

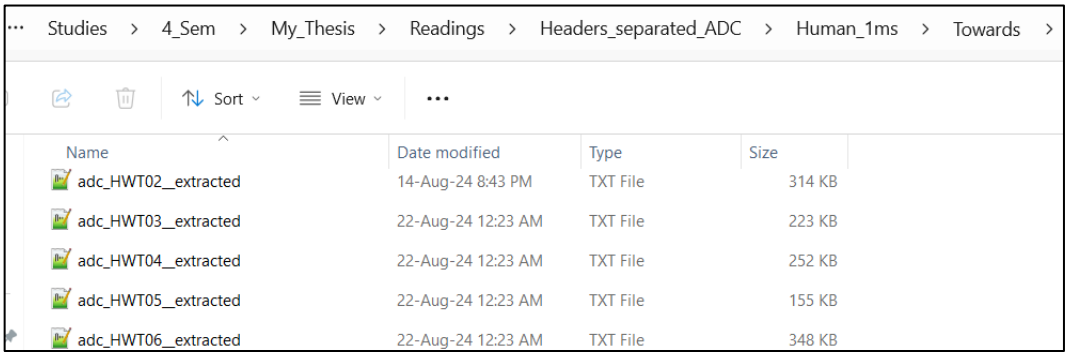


Figure 4.9: Screenshot of the header removed ADC data files

The Python Listing 4.1 script used for this task is shown in figure below:

```
 1. import os
 2. import pandas as pd
 3.
 4. # Define paths
 5. input_path = ("C:/My_Docs/GERMANY/ALAM/Frankfurt/Studies/4_Sem/"

6.                "My_Thesis/Readings/Headers_separated_ADC/Object/Perpendicular/New
")
 7. output_path = ("C:/My_Docs/GERMANY/ALAM/Frankfurt/Studies/4_Sem/"

8.                "My_Thesis/Readings/Headers_separated_ADC/Object/Perpendicular/New
/extracted")
 9.
10. # Iterate over files in the specified directory
11. for entry in os.scandir(input_path):
12.     if entry.is_file():  # Ensure that we're processing files only
13.         print(f"Processing {entry.name}")
14.
15.         # Read the file
16.         with open(entry.path, 'r') as f:
17.             lines = f.readlines()
18.
19.         data = []
20.         for line in lines:
21.             values = line.split()[16:]  # Adjust the slice as needed
22.             data.append(values)
```

```
23.
24.         # Convert the data to a DataFrame
25.         df = pd.DataFrame(data)
26.
27.         # Prepare the output file name
28.         # Get the base file name without extension
29.         base_name = os.path.splitext(entry.name)[0]
30.         output_file_name = f"{base_name}_extracted.txt"
31.         output_file_path = os.path.join(output_path, output_file_name)
32.
33.         # Save DataFrame to .txt file without commas
34.         df.to_csv(output_file_path, index=False, header=False, sep=' ')
35.
36. print("Done!")
37.
```

Listing 4.1: Code to clean and save ADC data files

4.6. Signal Processing Stages

This part of the thesis explains the detailed signal processing, step by step, from loading the ADC data to extracting the Doppler shift and other spectral features as shown in Figure 4.10. The methodology is important, including autocorrelation, Hamming windowing, FFT, and spectral feature extraction, to analyze and visualize the Doppler effect that represents a key issue in distinguishing between human and object movements in experimental data. References to the theoretical foundations of each processing step have already been discussed in detail in the **Theoretical Background** section of the thesis.

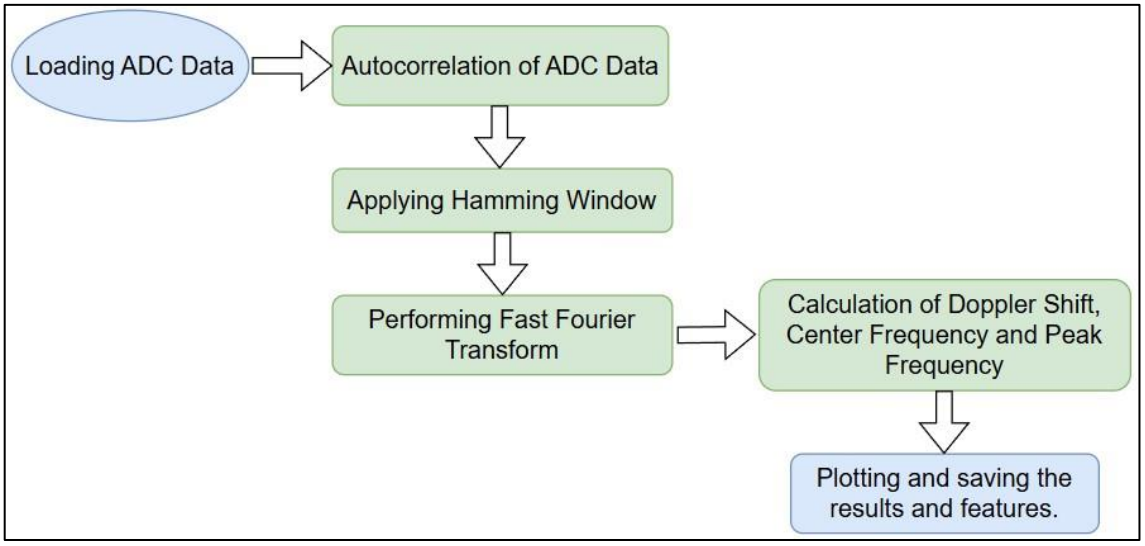


Figure 4.10: Flowchart of the Signal Processing Stage

4.6.1. Loading ADC Data

The first step in the processing pipeline involves reading the header-removed ADC data obtained from the decoding of the captured raw ADC data from the Red Pitaya ultrasonic sensor system stored in a specific directory. The data was sampled at a high frequency of 1.953125 MHz, and each row in the ADC file represents a snapshot of the signal in the time domain.

During this process, the code loops into all the '.txt' files in the given

directory path, whose filename contains "adc". One after another, each file's content is loaded through a function, *load_adc_data()*, given by the following fragment of Listing 4.2. The function reads the ADC data line by line, converting it into a matrix where each row represents discrete time samples of the signal.

```
 1. # Directory containing ADC data files
 2. directory_path = "path/to/adcfile"  # Replace with actual directory path
 3.
 4. # Parameters
 5. sampling_frequency = 1953125  # Hz
 6. transmitted_frequency = 40000  # Hz (base frequency of your ultrasonic
sensor)
 7.
 8. # Iterate over all files in the directory
 9. for filename in os.listdir(directory_path):
10.     if filename.endswith(".txt") and 'adc' in filename:
11.         file_path = os.path.join(directory_path, filename)
12.
13.         # Load ADC data
14.         adc_data = load_adc_data(file_path)
15.
```

Listing 4.2: Code to iterate over ADC data files in a directory

This data represents the starting point of the Doppler analysis, as changes in frequency over time will be examined to detect relative motion between the sensor and the object. Listing 4.3 shows the part of the code for loading ADC data to start the signal processing phase.

```
1. # Function to load ADC data from text file with multiple rows
2.
3. def load_adc_data(file_path):
4.     with open(file_path, 'r') as f:
5.         lines = f.readlines()
6.         adc_data = [list(map(int, line.strip().split())) for line in
lines]
7.     return np.array(adc_data)
```

Listing 4.3: Code to load ADC data from the saved ADC data files.

4.6.2. Autocorrelation of ADC Data

Autocorrelation of the loaded ADC data is performed to enhance periodic patterns in the signal and suppress random noise. This bears most relevance for Doppler analysis, whereby it brings to prominence the periodic features in the signal, usually those affected by the Doppler shift. The autocorrelation function measures the resemblance of the signal to a time-shifted version of itself. That helps, in our context, to find periodic structures induced by the movement, like Doppler shifts. The second half of the result from the autocorrelation is used to focus on the forward-looking components of the signal, since those bear most on the detection of motion-related frequency shifts.

Listing 4.4 shows the code which performs the autocorrelation of the loaded ADC data.

```
1. def perform_fft_with_acf(adc_data_row, sampling_frequency):
2.     # Compute the autocorrelation function (ACF)
3.     acf = correlate(adc_data_row, adc_data_row, mode='full')
4.     acf = acf[len(acf) // 2:]  # Take the second half
```

Listing 4.4: Code for applying Autocorrelation on the ADC Data

4.6.3. Applying the Hamming Window

Before applying the FFT, we applied a Hamming window to the autocorrelated signal. The Hamming windowing diminishes spectral leakage, spurious frequency components of the frequency-domain representation from the edges of the signals, and hence is an important first step in this process. Listing 4.5 displays the code which performs the Hamming window on the ACF data.

The Hamming window smooths the signal and thus makes sure that the discontinuities at the borders are minimal. That is an important step when analysing Doppler shifts, since even small discontinuities could introduce noise or artificial frequency components interfering with the accurate detection of Doppler. By applying the window, we ensured the FFT will produce a cleaner frequency spectrum; hence, we could detect subtle Doppler shifts created by slow or small movements more accurately.

```
1.     # Apply Hamming window to the ACF
2.     hamming_window = np.hamming(len(acf))
3.     windowed_acf = acf * hamming_window
```

Listing 4.5: Code for applying Hamming window to the ACF Data

4.6.4. Fast Fourier Transform (FFT)

Once the signal is autocorrelated and windowed, we applied Fast Fourier Transform to transfer the time-domain signal into the frequency domain. In any form of Doppler analysis, FFT is a crucial stage since the underlying frequency shift due to the Doppler effect is interpretable only in the frequency domain. FFT decomposes the signal into its constituent frequencies, thus enabling us to view how energy is distributed across the frequency bands. In the context of a Doppler, this step is very critical because now we will be able to find the Doppler frequency shifts associated with motions. Amplitude and direction of the Doppler shift, giving the speed and direction of the object's movement relative to the sensor could then be determined through examination of the resulting frequency spectrum. Listing 4.6 performs FFT on the windowed ACF data.

```
1.     # Perform FFT on the windowed ACF
2.     num_samples = len(windowed_acf)
3.     fft_values = np.fft.fft(windowed_acf)
4.     frequency_axis = np.fft.fftfreq(num_samples, 1/sampling_frequency)
5.     positive_freq_indices = np.where(frequency_axis > 0)
6.     frequency_axis_positive = frequency_axis[positive_freq_indices]
7.     amplitude_spectrum = np.abs(fft_values[positive_freq_indices]) /
num_samples
```

Listing 4.6: Code for performing FFT on the windowed ACF

4.6.5. Frequency Range Consideration Based on Theoretical Doppler Shift Calculations

In this paper, a frequency range of 35-45 kHz has been selected for analysis. This choice was directly related to the transmitted ultrasonic frequency and the expected Doppler shifts due to object/human movements.

The ultrasonic sensor sends out signals at a base frequency of 40 kHz, which was at the centre of our frequency range. In our experiments, objects and humans moved with an average maximum speed of 1.28 m/s. We calculated the range of frequencies we were likely to get owing to this motion using the theoretical formulation of the Doppler shift.

Theoretical Doppler Shift Calculation

The Doppler effect explains how a wave's frequency changes when an observer moves in respect to the wave source. Here, the ultrasonic sensor serves as the observer, and the moving object or person is the source of the reflected wave. The frequency shift (Δf) caused by the Doppler Effect can be mathematically described by the following equation (LibreTexts, 2023):

$$\Delta f = f' - f = \frac{v_{rel}}{v_s} \cdot f \qquad (20)$$

Where:

- Δf is the doppler shift (in Hz)
- f is the original frequency of the emitted wave (40,000 Hz or 40 kHz).
- f' is the observed frequency after the Doppler shift.
- v_{rel} is the relative velocity of the object or human (in meters per second).
- v_s is the speed of sound in air (~343 m/s at room temperature)..

For an object moving toward the sensor at **1.28 m/s** (the maximum speed observed in the experiments), the Doppler shift is calculated as:

$$\Delta f = \frac{1.28}{343}.40000 \approx 149.85\ Hz$$

Thus, when an object or human moves toward the sensor, the observed frequency increases to:

$$f' = f + \Delta f = 40000 + 149.85 \approx 40.150\ Hz$$

Similarly, when an object or human moves away from the sensor at the same speed, the observed frequency decreases to:

$$f' = f - \Delta f = 40000 - 149.85 \approx 39.850\ Hz$$

Based on these calculations, we confined our analysis to a frequency range between **35 kHz and 45 kHz**. This range comfortably encompasses the

observed frequency shifts due to the relative motion between the sensor and objects/humans in the experiment. The centre frequency of 40 kHz corresponds to no relative motion between the sensor and the object (i.e., when the object or human is stationary).

Any deviations from 40 kHz in either direction (up to 45 kHz or down to 35 kHz) are attributed to the Doppler shift caused by movement. This is the theoretical foundation for choosing this specific band of frequency for signal processing steps (FFT and spectral feature extraction). It ensures that our analysis captures the most relevant frequency shifts due to motion, allowing us to precisely observe and quantify the Doppler effect and its impact on ultrasonic signal processing.

With the above in mind theoretically, our Doppler analysis focuses on signals in the range of 35-45 kHz. Hence, we applied a filter to the FFT results by discarding all the frequency components outside the above-mentioned interval. Listing 4.7 shows the code snipped for filtering the frequency band:

```
1.    # Filter to only include frequencies in the specified range (35 kHz to
45 kHz)
2.     frequency_range_min = 35 * 1000  # 35 kHz in Hz
3.     frequency_range_max = 45 * 1000  # 45 kHz in Hz
4.     within_range_indices = np.where((frequency_axis_positive >=
frequency_range_min) & (frequency_axis_positive <= frequency_range_max))
5.     frequency_axis_range = frequency_axis_positive[within_range_indices]
6.     amplitude_spectrum_range = amplitude_spectrum[within_range_indices]
```

Listing 4.7: Code for performing FFT filtering

This then gave the opportunity to analyse the relevant frequency bands where, in fact, the Doppler shifts could take place. The signals that fall out of the range are just noise or irrelevant frequency components. The proper focus of the sensor is on a narrow band around 35-45 kHz to assure higher accuracy and precision in the detected Doppler shift.

4.6.6. Spectral Feature Extraction with Focus on Doppler Effect

As described in the second chapter of this thesis, we compute important spectral features after getting the frequency-domain representation in order to gain understanding of the Doppler effect and other signal properties. The Doppler change, which shows the frequency change brought on by the relative velocity between the sensor and the moving object, is the main aspect of importance in this thesis. Listing 4.8 shows the code for extracting spectral features.

- **Center Frequency** - The center frequency corresponds to the average amplitude-weighted frequency of a spectrum, which yields the general motion pattern. During Doppler analysis, center frequency shift indicates the Doppler shift of the signal.

- **Doppler Shift** -It gives the difference of the center frequency of the received signal and the transmitted frequency, 40kHz. The positive shift thus indicates

that the object is moving towards the sensor while the negative shift shows that it is moving away. This will be the main feature for the analyses in regard to the influence of the Doppler effect in this thesis.

```
# Feature extraction functions
# Spectral Centroid (Center Frequency)
def calculate_spectral_centroid(frequency_axis, amplitude_spectrum):
    return np.sum(frequency_axis * amplitude_spectrum) / np.sum(amplitude_spectrum)

# Spectral Bandwidth
def calculate_spectral_bandwidth(frequency_axis, amplitude_spectrum, centroid):
    return np.sqrt(np.sum(((frequency_axis - centroid) ** 2) * amplitude_spectrum) / np.sum(amplitude_spectrum))

# Spectral Entropy
def calculate_spectral_entropy(amplitude_spectrum):
    normalized_spectrum = amplitude_spectrum / np.sum(amplitude_spectrum)  # Normalize the spectrum
    return -np.sum(normalized_spectrum * np.log2(normalized_spectrum + 1e-12))  # Calculate entropy

# Doppler Shift (using center frequency)
def calculate_doppler_shift(center_frequency, transmitted_frequency):
    return center_frequency - transmitted_frequency
```

Listing 4.8: Code for calculating spectral features

Other features extracted are **spectral bandwidth** and **spectral entropy**. These may not have any direct relation with Doppler analysis, but they convey more information with respect to the nature of complexity and the signal, thus helping in differentiating between human and object movements.

4.6.7. Plotting and Saving the Results

We plotted and saved the frequency spectrum for each row of ADC data in order to visualize the Doppler effect among other signal features. Key spectral features were also marked into the plot that include the center frequency and peak frequency, which enable us to track how the Doppler shift affects the frequency content through time. These graphs therefore really gave insight into the behavior of the signal as object/human moved closer or away from the sensor. From these plots, the centered frequency difference due to Doppler was easily picked and further helped identify the direction of movement. Figure 4.11 shows the resulting amplitude-frequency for a row of ADC data file for the case of a human moving towards the sensor.

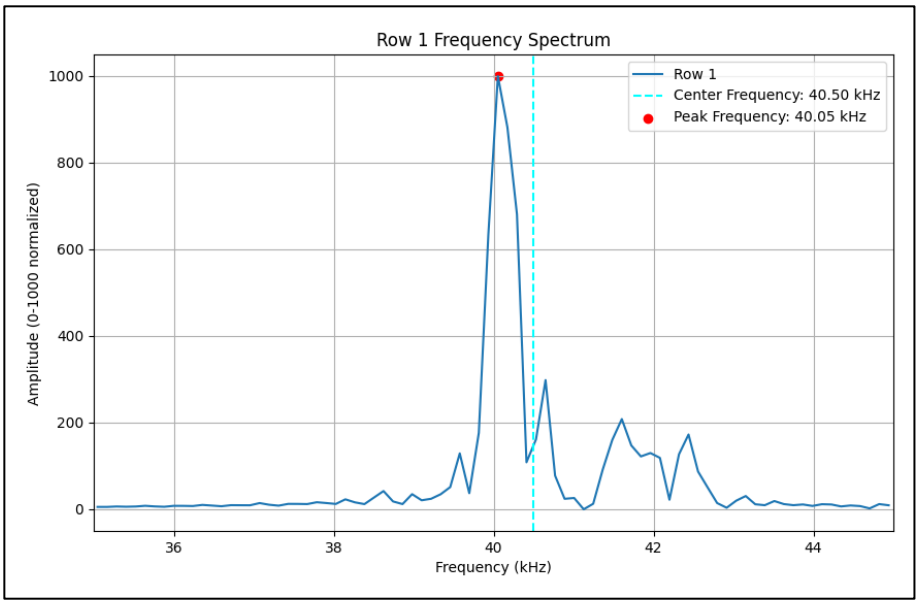


Figure 4.11: Amplitude-frequency for a row of ADC data file for the case of a human moving towards the sensor

4.6.8. Saving the Extracted Features

After processing every row, these features were saved in an Excel file for further analysis. Some of the important features included in this file are center frequency, bandwidth, peak frequency, entropy and Doppler shift. These extracted features were then used for classification purposes or further analysis of the effect of motion on the spectral characteristics of the signal. In the case of Doppler analysis, the Doppler shift feature is of great importance, as it gives critical insight into the relative motion of the sensor and object. Figure 4.12 show the folders created by the code for each of the ADC Data file in the Human Walking Towards the sensor case, within each corresponding folder the extracted features are saved along with the images of the amplitude-frequency plots according to the number of rows in each ADC data file.

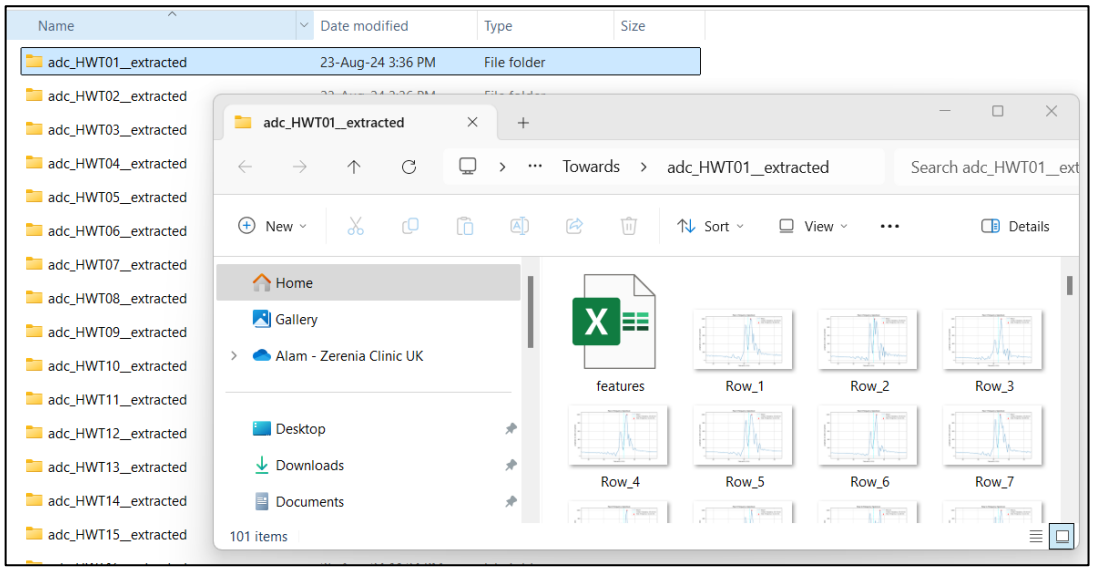


Figure 4.12: Folders created by the code for each of the ADC Data file

4.7. Machine Learning Models Implementation

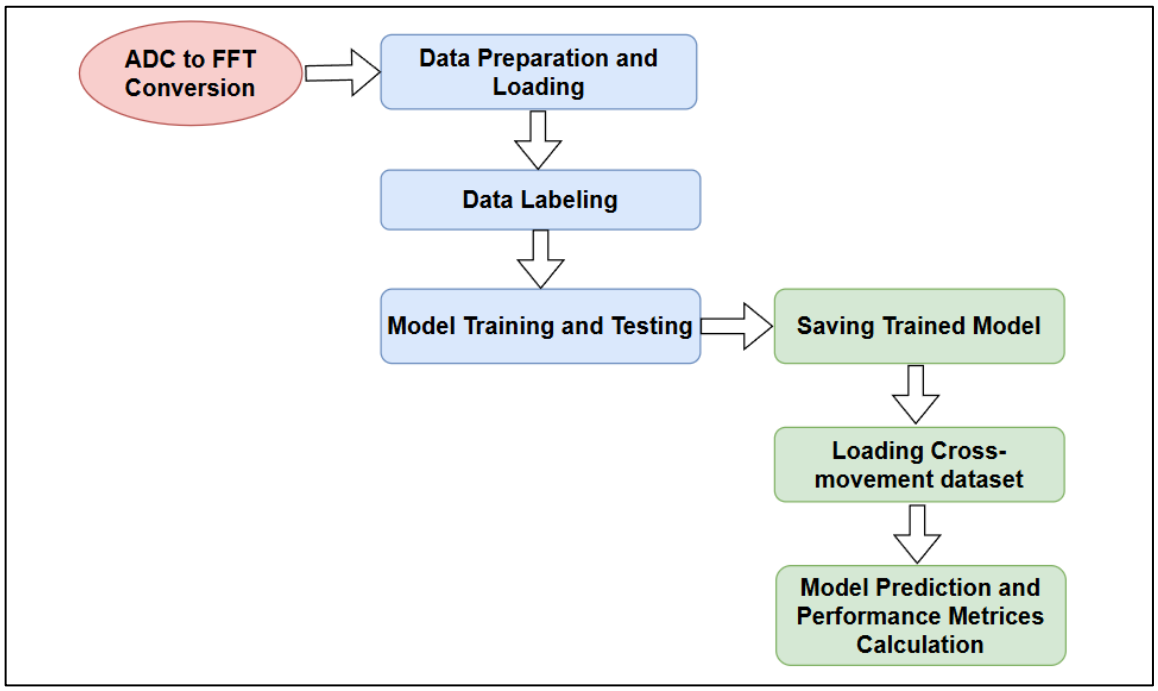


Figure 4.13: Flowchart of the Machine Learning Models Implementation

4.7.1. MLP Classifier for Human and Object Classification

The following section describes the methodology that involves classifying between human and object movements using an MLP classifier, based on FFT data after the signal processing phase of the ADC data coming from the ultrasonic sensor to leverage frequency-domain signatures to distinguish between human and object movement patterns. This is one kind of neural network model well known for its capability to capture complex relationships between input data and target classes. It is chosen because it has proven effective in handling high-dimensional frequency data emanating from human and object motions.

1. **Importing the packages**:

The MLP classifier was implemented using the Python libraries and tools listed below:

- **sklearn.neural_network** - The MLP model is created and trained using sklearn.neural_network.
- **sklearn.metrics -** The functions to compute accuracy, precision, recall, F1 score, confusion matrix, and classification reports are provided by sklearn.metrics.
- **sklearn.model_selection** -Provides functions to calculate accuracy, precision, recall, F1 score, confusion matrix, and classification reports.
- **pandas -** Pandas is used to handle and store data in Excel files, such as metrics and confusion matrices.

- **matplotlib.pyplot and seaborn -** Seaborn and matplotlib.pyplot are used to visualize the confusion matrix.
- **numpy-** For managing arrays and doing numerical calculations.
- **tkinter -** Using tkinter, a graphical user interface (GUI) is created to save the model and results and choose input FFT data files.
- **joblib -** This is used to store the MLP model that has been trained for later analyses.

```
 1. import tkinter as tk
 2. from tkinter import filedialog
 3. from sklearn.neural_network import MLPClassifier
 4. from sklearn.metrics import confusion_matrix, f1_score,
accuracy_score, precision_score, recall_score, classification_report
 5. from sklearn.model_selection import train_test_split
 6. import pandas as pd
 7. import matplotlib.pyplot as plt
 8. import seaborn as sns
 9. import numpy as np
10. import joblib
```

Listing 4 9: Importing necessary packages from Python Libraries

2. Data Preparation and Input Structure

The FFT data extracted from the ADC measurements of the ultrasonic sensor were used as the input to the MLP classifier. The steps followed for data preparation are as follows:

- **FFT Data Extraction** - As described in section 4.6, each ADC data file was processed to generate the corresponding FFT data. These FFT data files carried the frequency-domain representation of each movement case in human or object and formed the basis of classification. The following Listing 4.10 has been used to generate the relevant frequency bins and save the FFT data file:

```
 1. # Function to save the FFT data to a new file in the specified
format without decimals
 2.
 3. def save_fft_data(file_path, frequency_axis, fft_data):
 4.     new_file_path = file_path.replace('adc', 'fft')
 5.     with open(new_file_path, 'w') as f:
 6.         # Write the frequency components in the first row,
rounded to the nearest whole number
 7.         f.write('\t'.join(f'{int(round(freq))}' for freq in
frequency_axis) + '\n')
 8.         # Write the amplitude data row by row, rounded to the
nearest whole number
 9.         for row in fft_data:
10.             f.write('\t'.join(f'{int(round(amp))}' for amp in row)
+ '\n')
11.
```

Listing 4.10: Code to generate the relevant frequency bins and save the FFT data file

- **Combining FFT Data** - In each movement case, the data from several measurements of FFT were combined into one single "master FFT file" for both human and object movements. For instance, the FFT data from cases such as human walking towards, object moving away, stationary cases, etc., were stored in different master files. Each of the master files consisted of 594 rows of FFT data from each of the measurements. The first row of each FFT file, representing frequency bins, was excluded from either training or testing, as it had no impact on the classification process.

This process provided us a dataset that consisted of two master FFT files for each of the movement cases, one regarding human and one for object movements. These would later provide the input for the MLP classifier. We created four separate directories for storing the master FFT files of each movement case. The structure and naming of the folders are shown in Figure 4.14.

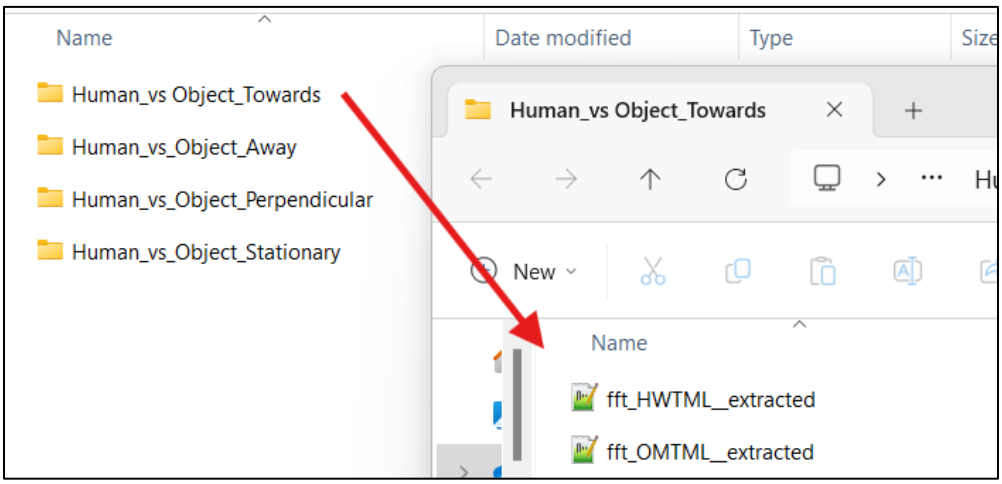


Figure 4.14: Folder structure and file naming convention for Machine Learning

3. Code to Load FFT Data

We used the *numpy* library to handle FFT data and the *pandas* library to manage structured data processing. The following Listing 4.11reads and processes FFT data from text files, skipping the first row - containing frequency bins - for training and testing purposes as shown in Figure 4.15.

```
 1. # Function to load FFT data while skipping the top row (frequency
bins)
 2. def load_fft_data(file_path):
 3.     with open(file_path, 'r') as f:
 4.         lines = f.readlines()[1:]  # Skip the first line
 5.     data = []
 6.     for line in lines:
 7.         values = line.split()
 8.         numValues = [int(x) for x in values]
 9.         data.append(numValues)
10.     return np.array(data)
```

Listing 4.11: Code to read FFT data from text files, skipping the first row

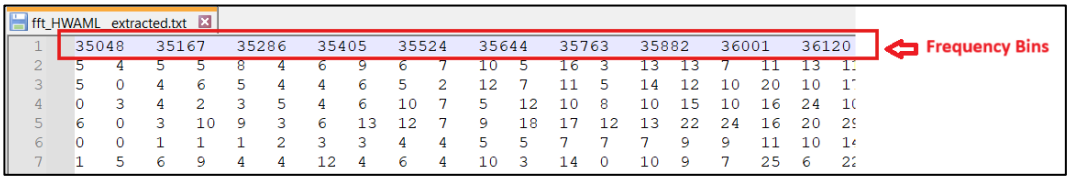


Figure 4.15: Sample FFT file with top row as Frequency bins

4. Labeling of Data

To train the model in a supervised manner, we labelled the FFT data based on the movement type:

- **Human movements were** labelled as **Class 2**.
- **Object movements** were labelled as **Class 1**.

This labelling allowed the MLP classifier to learn the distinguishing features between human and object movements. The data from each FFT file was combined into a matrix, where each row represented a segment of FFT data, and corresponding labels were appended.

The labelling was done using Listing 4.12:

```
1. # Combine data and labels
2. X = np.vstack([fft_data_object, fft_data_human])
3. y = np.array([1] * len(fft_data_object) + [2] *
len(fft_data_human))  # 1 for Object, 2 for Human
```

Listing 4.12: Code to label FFT Data as Human or Object

In the above code, **fft_data_object** comprises all the rows from object movement experiment and **fft_data_human** comprises all the rows from human movement experiment. Here, the corresponding label vector **y** was created and assigns the label 1 to all object rows and label 2 to all human rows.

5. Graphical User Interface for File Selection

A Graphical User Interface (GUI) with the **Tkinter** module was developed to enable the user to choose FFT data files for human and object motions to train and test the MLP model. The development of this GUI eliminated the need for fixed file paths. Listing 4.13 displays GUI twice, first for selecting FFT file for Object and then for selecting Human FFT file. Figure 4.16 shows the GUI for selecting FFT file for Object data.

```
 1. # Function to select files using a user interface
 2. def select_file(title):
 3.     root = tk.Tk()
 4.     root.withdraw()
 5.     file_path = filedialog.askopenfilename(title=title,
filetypes=[("Text Files", "*.txt")])
 6.     return file_path
 7.
```

```
# Select FFT data files for human and object
file_path_object = select_file("Select FFT Data File for Object")
file_path_human = select_file("Select FFT Data File for Human")
```

Listing 4.13: Code for File Selection

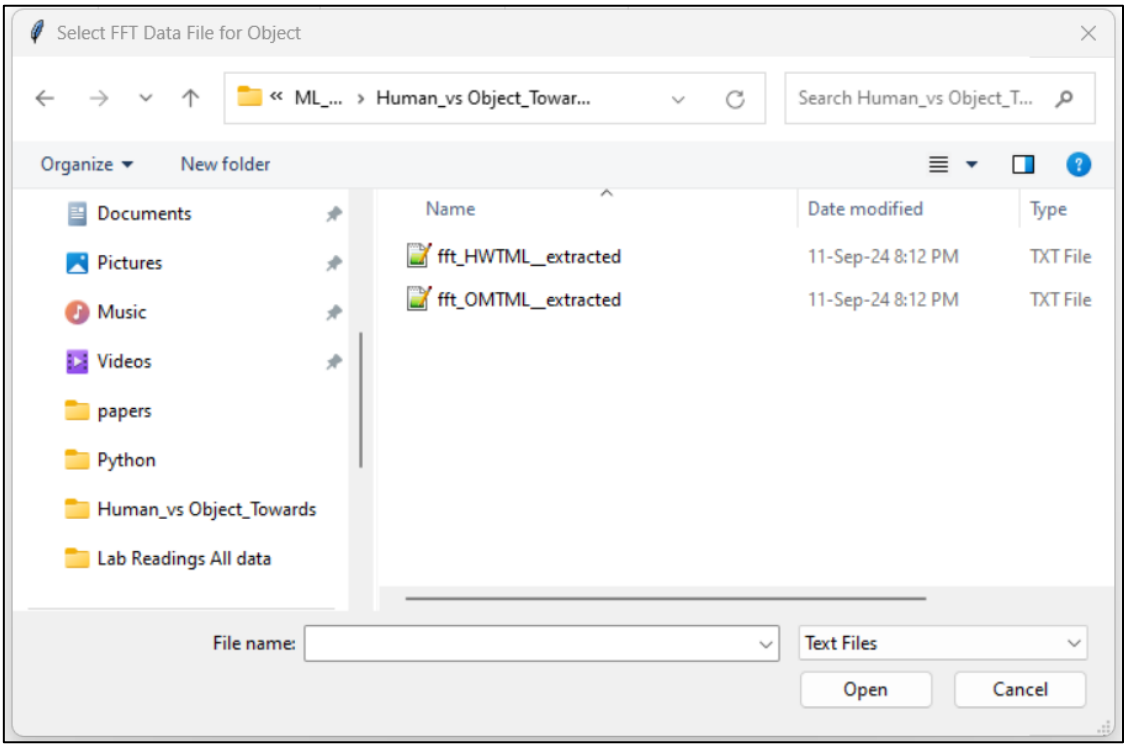


Figure 4.16: GUI for selecting Object FFT File

6. MLP Classifier Architecture and Training

Classification of human and object movements was performed with an MLP classifier based on the FFT values. The feed-forward neural network is acknowledged to model complex nonlinear relationships between the input features and output labels. The model was constructed with the use **MLPClassifier** from *sklearn.neural_network* module, chosen as the library for machine learning where models are provided to be easily consumed in an intuitive API. Besides that, as a numerical library, *numpy* was also used. Following are the detailed specification and parameters for the MLP Classifier:

- **Input Data**: The input to the MLP consisted of the FFT values from the ADC data (after excluding the frequency bin row).

- **Hidden Layers**: The classifier had two hidden layers, with **100 neurons** in the first hidden layer and **50 neurons** in the second hidden layer. This configuration provided sufficient capacity to model the intricate patterns in the FFT data.

- **Activation Function**: The Rectified Linear Unit (ReLU) was selected as the activation function for the hidden layers due to its ability to avoid issues like the vanishing gradient and speed up convergence.

- **Solver**: The '*lbfgs*' solver was used for optimization. It is particularly effective for smaller datasets where second-order optimization techniques are feasible.

- **Maximum Iterations**: A limit of **300 iterations** was imposed for training, which was sufficient for convergence without overfitting.

- **Random State**: A seed of **42** was used to ensure the reproducibility of the results.

- **Data Splitting and Model Training:** We split the combined FFT data into training and testing set, a split ratio of 70% for training and 30% for testing was chosen in order to ensure that the model has enough to learn from, yet it still leaves sufficient data for evaluation.

  The below Listing 4.14 shows the code used for data splitting, MLP initialization and training.

```
1. # Split data into training and testing sets
2. X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
3.
4. # Train MLP classifier
5. mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=300,
random_state=42)
6. mlp.fit(X_train, y_train)
7.
```

Listing 4.14: Code for MLP Classifier Initialization and Training

7. Model Evaluation Metrics and Confusion Matrix

The performance of the MLP trained in this context was measured by multiple metrics in order to find out how accurate and robust the model is. These metrics include **accuracy, precision, recall, F1-score**, and **confusion matrix**. In this regard, functions for evaluation from the **scikit-learn** library are used to ensure that the performance of classification is strictly gauged. The **matplotlib** and **seaborn** libraries supported the visualization of the confusion matrix, showing graphically a comparison between the model's predictions versus the true labels for human and object movements. Listing 4.15 and Listing 4.16calculate the performance metrices and plots the confusion matrix, respectively.

```
 1. # Make predictions on test set
 2. pred = mlp.predict(X_test)
 3.
 4. # Calculate confusion matrix and
 5. cf_matrix = confusion_matrix(y_test, pred)
 6. accuracy = accuracy_score(y_test, pred)
 7. f1 = f1_score(y_test, pred, average='weighted')
 8. precision = precision_score(y_test, pred, average='weighted')
 9. recall = recall_score(y_test, pred, average='weighted')
11. # Extract TP, TN, FP, FN from the confusion matrix
12. TP = cf_matrix[1, 1]
13. TN = cf_matrix[0, 0]
```

```
14. FP = cf_matrix[0, 1]
15. FN = cf_matrix[1, 0]
16.
17. # Print metrics
18. print("Confusion matrix:")
19. print(cf_matrix)
20. print(f"Accuracy: {accuracy:.5f}")  # Display the exact accuracy
up to 5 decimal places
21. print(f"F1 score: {f1:.5f}")  # Display the exact F1 score up to
5 decimal places
22. print(f"Precision: {precision:.5f}")  # Display the precision up
to 5 decimal places
23. print(f"Recall: {recall:.5f}")  # Display the recall up to 5
decimal places
24. print(f"True Positives (TP): {TP}")
25. print(f"True Negatives (TN): {TN}")
26. print(f"False Positives (FP): {FP}")
27. print(f"False Negatives (FN): {FN}")
```

Listing 4.15: Code for Performance Evaluation and Confusion Matrix

```
1. # Plot the confusion matrix with TP, TN, FP, FN annotations
2. plt.figure(figsize=(8, 6))
3. sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Object', 'Human'], yticklabels=['Object', 'Human'],
annot_kws={"size": 18})
4. plt.title('Confusion Matrix',fontsize=18)
5. plt.xlabel('Predicted', fontsize=18)
6. plt.ylabel('Actual', fontsize=18)
7. plt.xticks(fontsize=14)                    # Increase font size
of x-axis ticks
8. plt.yticks(fontsize=14)                    # Increase font size
of y-axis ticks
```

Listing 4.16: Code for Plotting the Confusion Matrix

8. Saving the Trained Model and Metrics

We added the functionality of saving the model as a serialized file using the *joblib* library to facilitate reusing the trained model. Once saved, it was later loaded and then evaluated on different datasets, such as human and object movements under various conditions to analyze the efficiency of the model trained on Doppler effected dataset like Stationary or Perpendicular vs non-Doppler dataset like Human/Object moving Towards or Away.

All metrics computed-confusion matrix, accuracy, precision, recall, and F1-score were written in an Excel document for recordkeeping and further analysis using ***pandas*** and ***xlsxwriter***. Listing 4.17 saves the confusion matrix, key performance metrics, and the detailed classification report to an Excel file. It allows for a clear record of how the MLP model performed, which was later utilized to compare the performance of MLP model trained under different experimental cases and further analysis.

```
# Create an Excel file to save all displayed metrics and the classification report
output_filename = filedialog.asksaveasfilename(defaultextension=".xlsx", title="Save Metrics and Confusion Matrix", filetypes=[("Excel Files", "*.xlsx")])
if output_filename:
    # Save metrics and confusion matrix to Excel
    with pd.ExcelWriter(output_filename, engine='xlsxwriter') as writer:
        # Save confusion matrix
        cf_matrix_df = pd.DataFrame(cf_matrix, index=['Object', 'Human'], columns=['Predicted Object', 'Predicted Human'])
        cf_matrix_df.to_excel(writer, sheet_name='Confusion Matrix')

        # Save metrics
        metrics_df = pd.DataFrame({
            'Metric': ['Accuracy', 'F1 Score', 'Precision', 'Recall', 'TP', 'TN', 'FP', 'FN'],
            'Value': [accuracy, f1, precision, recall, TP, TN, FP, FN]
        })
        metrics_df.to_excel(writer, sheet_name='Metrics', index=False)

        # Save classification report
        report_data = []
        lines = report.split('\n')
        for line in lines[2:-3]:
            row_data = ' '.join(line.split()).split(' ')
            report_data.append(row_data)

        report_df = pd.DataFrame(report_data, columns=['Class', 'Precision', 'Recall', 'F1-Score', 'Support'])
        report_df.to_excel(writer, sheet_name='Classification Report', index=False)

    print(f"Metrics and confusion matrix have been saved to {output_filename}")
```

Listing 4.17: Code for saving the trained model and metrices

4.7.2. Convolutional Neural Network for Human and Object Classification

This section describes the methodology for the classification of human and object movements by using a CNN that acts on FFT data obtained after the signal processing phase of the ADC data coming from the ultrasonic sensor to leverage frequency-domain signatures to distinguish between human and object movement patterns. CNNs are rather suitable to identify features and patterns of such ordered input data, represented by the FFT spectra. The implementation of the CNN classifier was carried out in Python programming using the TensorFlow and Keras libraries as shown in Listing 4.18.

1. **Importing the packages:**

The following Python libraries and tools were utilized for implementing

the CNN classifier:

- **TensorFlow & Keras:** - These were used for building, training, and saving the CNN model. The *Sequential* model API was used to define a feedforward CNN architecture.
- **sklearn.metrics** -Provides functions to calculate accuracy, precision, recall, F1 score, confusion matrix, and classification reports.
- **sklearn.model_selection** -Provides functions to calculate accuracy, precision, recall, F1 score, confusion matrix, and classification reports.
- **pandas -** For handling and saving data (e.g., metrics, confusion matrices) into Excel files.
- **matplotlib.pyplot and seaborn -** Used for visualizing the confusion matrix.
- **numpy-** For numerical computations and handling arrays.
- **tkinter -** To create a graphical user interface (GUI) to select input FFT data files and save the model/results.

```
 1. import tkinter as tk
 2. from tkinter import filedialog
 3. from sklearn.metrics import confusion_matrix, f1_score,
accuracy_score, precision_score, recall_score, classification_report
 4. from sklearn.model_selection import train_test_split
 5. import pandas as pd
 6. import numpy as np
 7. import matplotlib.pyplot as plt
 8. import seaborn as sns
 9. from tensorflow.keras.models import Sequential
10. from tensorflow.keras.layers import Conv1D, MaxPooling1D,
Flatten, Dense, Dropout
11. from tensorflow.keras.utils import to_categorical
```

Listing 4.18: Importing necessary Python libraries to run the CNN model

2. Data Preparation and Labeling

Input for the CNN classifier was the same as used in the MLP approach as explained below in brief:

- **FFT Data Extraction** - FFT data, generated from ADC data files, served as the input for training and testing the CNN. The code used to read and structure the data is similar to the MLP approach, where we load the FFT values from text files and skip the first row containing the frequency bins.
- **Data Labelling** - The FFT data was labeled as "1" for object movements and "2" for human movements. This labeling was used to differentiate between the two classes during training.

To reshape the data appropriately for CNN input, we added an additional

dimension to represent the number of features. The data for each sample was reshaped into a 3D array, where each sample had one feature dimension. Listing 4.19 shows the code for data preparation.

```
# Load FFT data
fft_data_object = load_fft_data(file_path_object)
fft_data_human = load_fft_data(file_path_human)

# Combine data and labels
X = np.vstack([fft_data_object, fft_data_human])
y = np.array([1] * len(fft_data_object) + [2] *
len(fft_data_human))  # 1 for Object, 2 for Human

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Reshape data for CNN (samples, time steps, features)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1],
1))  # Add the third dimension for features
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# One-hot encode the labels for CNN
y_train = to_categorical(y_train - 1, num_classes=2)
y_test = to_categorical(y_test - 1, num_classes=2)
```

Listing 4.19: Code for data preparation and labeling

3. CNN Model Architecture

The architecture applied in the present work was composed of an ensemble of convolutional and pooling layers, followed by fully connected layers for class labels. In this way, this architecture allowed the CNN to extract relevant features from the FFT data, such as frequency patterns associated with human and object movements.

Following are the detailed specification and parameters for the CNN Classifier used:

- **Input Layer:** The input shape was defined with the dimensionality of FFT data, including rows and frequency bins.
- **Convolutional Layers:** This model employs two layers of Conv1D, with 64 and 32 layers, respectively, scanning over the inputs to flag only the most relevant features in the frequency domain. ReLU was an excellent choice as the activation function.
- **Pooling Layers:** Two MaxPooling1D layers have been used, subsequent to every convolutional layer, because they down sample the input and reduce the dimensionality.
- **Dropout:** To avoid overfitting problems during model training, this layer is implemented after the fully connected Dense layer.
- **Output Layer:** The last Dense layer had two neurons and an activation of

SoftMax in order to predict the class labels object or human.

The following Listing 4.20 defines the CNN architecture:

```
# Create the CNN model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))  # Prevent overfitting
model.add(Dense(2, activation='softmax'))  # 2 classes: Object and
Human
```

Listing 4.20: Code for CNN model architecture

4. Training and Model Compilation

The CNN model was compiled with the Adam optimizer, which converges faster. Since we have two classes, namely object and human, we used the categorical cross-entropy loss function. Our model went through 20 rounds of training, epochs, with a batch size of 32. The training used 70% of the data while the remaining 30% are used for testing as displayed in code Listing 4.21

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the CNN model
model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_test, y_test))
```

Listing 4.21: Code for CNN compilation and training

5. Model Evaluation Metrics and Confusion Matrix

The CNN model was evaluated on the same metrices as used for the evaluation of MLP model namely, Accuracy, Precision, Recall and F1 Score. A confusion matrix was also generated for each experiment. The confusion matrix and performance metrics were printed, and a heatmap of the confusion matrix was generated for visualization using the code Listing 4.22

```
# Calculate confusion matrix and other metrics
cf_matrix = confusion_matrix(y_true, y_pred)
accuracy = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred, average='weighted')
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')

# Calculate TP, TN, FP, and FN
TP = cf_matrix[1, 1]
TN = cf_matrix[0, 0]
FP = cf_matrix[0, 1]
FN = cf_matrix[1, 0]
```

```
13.
14. # Print metrics
15. print("Confusion matrix:")
16. print(cf_matrix)
17. print(f"Accuracy: {accuracy:.5f}")
18. print(f"F1 score: {f1:.5f}")
19. print(f"Precision: {precision:.5f}")
20. print(f"Recall: {recall:.5f}")
21. print(f"True Positives (TP): {TP}")
22. print(f"True Negatives (TN): {TN}")
23. print(f"False Positives (FP): {FP}")
24. print(f"False Negatives (FN): {FN}")
25.
26. # Plot the confusion matrix with TP, TN, FP, FN annotations
27. plt.figure(figsize=(8, 6))
28. sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Object', 'Human'], yticklabels=['Object', 'Human'],
annot_kws={"size": 18})
29. plt.title('Confusion Matrix',fontsize=18)
30. plt.xlabel('Predicted', fontsize=18)
31. plt.ylabel('Actual', fontsize=18)
32. plt.xticks(fontsize=14)                    # Increase font size of
x-axis ticks
33. plt.yticks(fontsize=14)                    # Increase font size of
y-axis ticks
34.
35. # Add a legend to display TP, TN, FP, FN values
36. handles = [
37.     plt.Line2D([0], [0], color='white', marker='o', markersize=0,
label=f'TP = {TP}'),
38.     plt.Line2D([0], [0], color='white', marker='o', markersize=0,
label=f'TN = {TN}'),
39.     plt.Line2D([0], [0], color='white', marker='o', markersize=0,
label=f'FP = {FP}'),
40.     plt.Line2D([0], [0], color='white', marker='o', markersize=0,
label=f'FN = {FN}')
41. ]
42. plt.legend(handles=handles, loc='upper center', bbox_to_anchor=(0.5,
-0.15), ncol=4, borderaxespad=0.)
43.
44. plt.tight_layout()
45. plt.show()
46.
```

Listing 4.22: Code for Model Evaluation Metrics and Confusion Matrix

6. Saving the Trained Model and Metrics

After training and testing the model, the trained CNN was then saved in **.h5** format for later testing with cross experimental datasets as described in next section. Also, all the important metrics, being accuracy, precision, recall, F1 score, along with the confusion matrix, were saved in an Excel file for further analysis. This is because, by saving the model along with metrics, we are able to test the CNN model on any other dataset afterwards, for example, FFT data representing movement cases like stationary, moving away, or moving perpendicular, and then do a comparison with the MLP model. Listing 4.23 was used for saving the trained model and metrices.

```
# Save the confusion matrix, classification report, and overall metrics in an Excel file
output_filename = filedialog.asksaveasfilename(defaultextension=".xlsx", title="Save Metrics and Confusion Matrix", filetypes=[("Excel Files", "*.xlsx")])
if output_filename:
# Save the confusion matrix, classification report, and metrics in Excel
    with pd.ExcelWriter(output_filename, engine='xlsxwriter') as writer:
# Save confusion matrix
        cf_matrix_df = pd.DataFrame(cf_matrix, index=['Actual Object', 'Actual Human'], columns=['Predicted Object', 'Predicted Human'])
        cf_matrix_df.to_excel(writer, sheet_name='Confusion Matrix')

# Save classification report
        report_df = pd.DataFrame(classification_report(y_true, y_pred, target_names=['Object', 'Human'], output_dict=True)).T
        report_df.to_excel(writer, sheet_name='Classification Report')

# Save overall performance metrics (accuracy, precision, recall, f1 score, TP, TN, FP, FN)
        metrics_df = pd.DataFrame({
            'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'True Positives (TP)', 'True Negatives (TN)', 'False Positives (FP)', 'False Negatives (FN)'],
            'Value': [accuracy, precision, recall, f1, TP, TN, FP, FN]
        })
        metrics_df.to_excel(writer, sheet_name='Overall Metrics', index=False)

    print(f"Results have been saved to {output_filename}")
else:
    print("No directory selected for saving. Results were not saved.")

```

Listing 4.23: Code for Saving the Trained Model and Metrics

4.8. Cross-Movement Performance Evaluation of Pretrained MLP and CNN Models

Here, we are going to describe the methodology followed to evaluate each of the MLP and CNN classifier, trained on both stationary or perpendicular human-object dataset, respectively for the classification of FFT data from moving human and object cases like moving towards or away. The main goal is to observe whether models, which were trained with a dataset of a stationary and perpendicular movement, classify with movement towards or away from the sensor, and finally analyze if any performance improvement is achieved when models are being trained for a specific movement case. This methodology will outline the steps involved in the process of reloading pre-trained models, apply those models to new test data, and save the results of the evaluation.

4.8.1. Loading Pretrained Models and Test Data Selection

As seen in the previous section, the MLP and CNN models were trained on FFT data for Stationary as well as Perpendicular Human-Object classifications. The MLP models have been saved as *.pkl* files using the j*oblib* library whereas the CNN models have been saved as *.h5* files using the save functionality of *Keras*. In both these models, the previously saved files were loaded in this evaluation to avoid retraining the models. This gave the opportunity for an effective comparison in the performance of these models when tested on datasets which they were not explicitly trained on. For the test FFT data, we used the same FFT data format and structure but the following specific comparison cases:

Model Training Scenarios for Evaluation:

1. **Stationary Pretrained Model:**

- **Training Data**: Human vs. object stationary data.
- **Testing Data**: Human vs. object data for "moving towards" and "moving away" cases.

2. **Perpendicular Pretrained Model:**

- **Training Data**: Human vs. object perpendicular movement data.
- **Testing Data**: Human vs. object data for "moving towards" and "moving away" cases.

The FFT test data for human and object movements were selected using python's *Tkinter* module which provided graphical file selection interface. The data was loaded into *NumPy* arrays, excluding the top row, which contained frequency bins as was done for training. Below Listing 4.24 and Listing 4.25 show the code snippet from CNN and MLP evaluation script to load the pretrained CNN and MLP models.

```
1. # Load pretrained model
2. print("Please select the pretrained MLP model file (trained on
stationary dataset).")
3. model_filename = select_file("Select the pretrained MLP model file
(trained on stationary dataset)")
4. mlp = joblib.load(model_filename)
```

Listing 4.24: Code Snippet for reloading the trained MLP Model

```
1. # Load pretrained CNN model
2.
3. print("Please select the pretrained CNN model file.")
4. model_filename = select_file("Select the pretrained CNN model file
(.h5)")
5. cnn_model = load_model(model_filename)
```

Listing 4.25: Code Snippet for reloading the trained CNN Model

After loading, the test data for human and object movement were combined into one dataset. Labels were assigned as follows using the code Listing 4.26:

- **1** for object data
- **2** for human data

```
1. # Combine moving dataset data and labels for testing
2. X_test_moving = np.vstack([fft_data_object, fft_data_human])
3. y_test_moving = np.array([1] * len(fft_data_object) + [2] *
len(fft_data_human))  # 1 for Object, 2 for Human
```

Listing 4.26: Code Snippet for labeling the test data

The same test data was used for evaluating both MLP and CNN models for consistency. The test dataset was combined and used to evaluate the models' performance on classifying Doppler-shifted movements (towards and away).

4.8.2. Model Prediction and Performance Metrices Calculation

Once the test data was loaded, both the MLP and CNN models made their predictions of whether a human or object caused the movement. This involved passing the FFT data through each pre-trained model.

- **MLP Model Prediction:** It used previously calculated weights for making the prediction of human object classification on test data moving towards and going away. The predicted labels were compared with the actual labels, and performances metrics were calculated.
- **CNN Model Prediction**: For the CNN model, the test data was reshaped into a form that could easily feed into the model as taken-in input format (samples, time steps, and features). The CNN model made predictions for the classification of a human-object; the output labels were matched to the true labels.

The following code Listing 4.27 snippet shows the calculation of these metrics for the CNN model (the same process was used for MLP):

```
1. # Calculate performance metrics
2. cf_matrix_moving = confusion_matrix(y_test_moving, pred_moving_classes)
3. accuracy_moving = accuracy_score(y_test_moving, pred_moving_classes)
4. f1_moving = f1_score(y_test_moving, pred_moving_classes,
average='weighted')
5. precision_moving = precision_score(y_test_moving, pred_moving_classes,
average='weighted')
6. recall_moving = recall_score(y_test_moving, pred_moving_classes,
average='weighted')
```

Listing 4.27: Code Snippet for performance metrices calculation

The results provided valuable insights into how well models trained on stationary or perpendicular data generalized to movement-induced Doppler-shifted data, such as moving towards or away.

4.8.3. Saving the Results

For both MLP and CNN evaluations, the results—including the confusion matrix, classification report, and overall performance metrics—were saved to Excel files for further analysis. The results were organized clearly to enable comparison between models trained on different movement cases and tested on Doppler-shifted data.

- **Confusion Matrix**: This showed a breakdown of true and false predictions with respect to human and object classification.
- **Classification Report**: Precision, recall, and F1 score were given for the human and object classes.
- **Overall Metrics**: Summarized the key performance indicators of accuracy, precision, recall, and F1 score.

The following code Listing 4.28 shows how these results were saved for the CNN evaluation (the same approach was used for MLP):

```
# Choose directory to save the results
save_directory = select_save_directory()

if save_directory:
    output_filename = os.path.join(save_directory, 'CNN_performance_moving_dataset.xlsx')
    print(f"Saving results to: {output_filename}")

    # Save the confusion matrix, classification report, and overall metrics in Excel
    with pd.ExcelWriter(output_filename, engine='xlsxwriter') as writer:
        # Save confusion matrix
        cf_df = pd.DataFrame(cf_matrix_moving, index=['Actual Object', 'Actual Human'], columns=['Predicted Object', 'Predicted Human'])
        cf_df.to_excel(writer, sheet_name='Confusion Matrix')

        # Save classification report
        report_df = pd.DataFrame(classification_report(y_test_moving, pred_moving_classes, target_names=['Object', 'Human'], output_dict=True)).T
        report_df.to_excel(writer, sheet_name='Classification Report')

        # Save overall performance metrics (accuracy, precision, recall, f1 score)
        metrics_df = pd.DataFrame({
            'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
            'Value': [accuracy_moving, precision_moving, recall_moving, f1_moving]
        })
        metrics_df.to_excel(writer, sheet_name='Overall Metrics', index=False)

    print(f"Results have been saved to {output_filename}")
else:
    print("No directory selected for saving. Results were not saved.")
```

Listing 4.28: Code Snippet for saving the results

For each model, four Excel reports were generated:

1. Model trained on stationary dataset, tested with moving towards data.
2. Model trained on stationary dataset, tested with moving away data.
3. Model trained on perpendicular dataset, tested with moving towards data.
4. Model trained on perpendicular dataset, tested with moving away data.

The extensive evaluation allowed us to analyze how these models which were previously trained on non-Doppler dataset (Stationary and Moving Sideways/Perpendicular) perform on classifying human-object movement induced Doppler dataset.

Chapter 5

Observations and Results

This chapter presents the result and related analysis for both the signal processing and classification phases of experiments conducted to distinguish between human and object movements using MLP and CNN classifiers. The first sections of this chapter deal with the Signal Processing Results: especially, Doppler shift observations and the analysis of the spectral features bring insight into how the frequency-domain characteristics of human and object movements will differ w.r.t. experimental cases of moving towards, moving away, stationary and moving sideways/perpendicular. The subsequent sections discuss the performance of the MLP and CNN classifiers, when both are trained with both stationary and perpendicular datasets and tested on Doppler-shifted cases, such as moving towards and away. These sections also describe the performance metrics-accuracy, precision, recall, and F1 score-of the classifiers along with the impact of the Doppler effect on the classification performance. Lastly, a comparison of the two models is given to find out which of them performed better under different movement conditions. For each section in discussions, corresponding tables, figures, and plots are given so that a comprehensive presentation of results can be delivered with key information from these results.

5.1. Doppler Shift Analysis

This section presents the analysis of Doppler shift observed in both human and object movements in various scenarios. In the following, an attempt is made to highlight how Doppler shift would vary according to relative motion that will be induced by the target-human or object-with respect to the ultrasonic sensor.

5.1.1. Human Walking Towards

In this case, the positive frequency shift of 300 Hz was observed when the target (human) approached the sensor as shown in the below amplitude-frequency plot. The average speed of human subject was 1.28 m/s which theoretically leads to a positive Doppler shift of 149.85 Hz, which is almost half of the observed Doppler Shift in the experiment. However, since the ultrasonic sensor was placed at a level below the knees of the walking human subject, the

speed of leg movement was also considered to finally compare the theoretical doppler shift with observed values. Below is the mathematical calculation for calculating the Leg speed of human moving towards the sensor at 1.28 m/s.

- **Stride Length**: This is the distance covered by one step of leg during walking which was found to be 0.75 meters for the human subject considered in the experiments.
- **Cadence:** This is the number of steps per second, which can be calculated as:

$$Cadence = \frac{Walking\ Speed}{Stride\ Length} = \frac{1.28\ m/s}{0.75\ m} \approx 1.71\ steps\ per\ second$$

- **Leg Speed:** The speed of leg movement would be

$$Leg\ Speed = Cadence \times 2 \times Stride\ Length$$

Substituting the values:

$$Leg\ Speed = 1.71 \times 2 \times 0.75 = 2.565\ m/s$$

This derived the conclusion that if the subject is moving at 1.28 m/s, then the legs would be moving at approximately 2.57 m/s. Doppler shift was then calculated considering the speed of leg movement as per formula already described in chapter 2 for calculating the doppler shift.

$$\Delta f = \frac{2.57}{343} \times\ 40000 \approx 300\ Hz$$

So, the theoretical Doppler Shift was found to be approximately 300 Hz when the human subject was moving at 1.28 m/s with legs moving at a speed of 2.57 m/s. The calculation matches closely with the observed values of Doppler shift. However, slight deviations were noted due to leg motion dynamics, where the periodic leg movement caused variations in the velocity profile of the human subject. The other reason for slight variations in observed Doppler shift could be due to multipath reflections from the experimental environment causing some fluctuations in the signal.

Table 5.1 summarizes the theoretical vs observed doppler shifts for each row of the ADC data file obtained from a single instance of a human subject moving towards the sensor from a distance of 3 m to 30 cm towards the ultrasonic sensor.

Table 5.1: Theoretical vs Observed Doppler Shift (Human Walking Towards)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	300 Hz	292.85	40.29	40.29
Row -2	300 Hz	304.68	40.3	40.17
Row -3	300 Hz	305.81	40.31	40.29
Row -4	300 Hz	291.09	40.29	40.17
Row -5	300 Hz	166.78	40.17	40.17

It can be observed from the above table for amplitude-frequency plots that as the human subject started moving marked with Row-1 and reached in the middle of the 3 meter walking path at Row-3, the positive Doppler Shift went from 292.85 Hz to 305.81 Hz but as the human subject approaches the sensor the walking speed slightly decreased while stopping at a distance of 30 centimetres from the sensor as marked in Row-4 and Row-5. Due to decrease in speed the positive Doppler Shift slightly decreased from 291.09 Hz to 166.78 Hz respectively and at this point the sensor stopped recording the measurements. Figure 5.1 shows the Amplitude-Frequency plot for Row-2 of the ADC Data file of human walking towards, highlighting the positive Doppler shift.

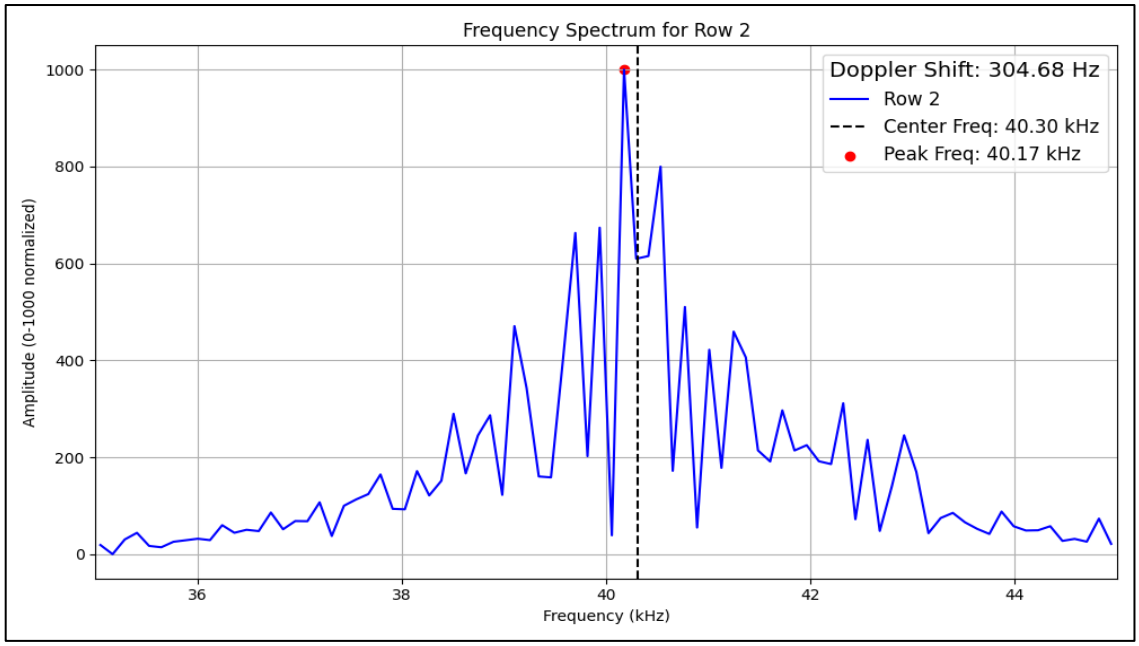


Figure 5.1: Amplitude-Frequency plot for Row-2 of the ADC Data file of human walking Towards

5.1.1 Human Walking Away

In this case, a negative frequency shift of 300 Hz was observed when the target (human) walked away from the sensor as shown in the below amplitude-frequency plot. The average speed of human subject was 1.28 m/s which

theoretically leads to a negative Doppler shift of -149.85 Hz, which is almost half of the observed Doppler Shift in the experiment. However, since the ultrasonic sensor was placed at a level below the knees of the walking human subject, the speed of leg movement was considered in this case also to finally compare the theoretical doppler shift with observed values. The calculation for leg movement speed is the same as done in the case of Human moving towards. This derived the conclusion that if the subject is moving away at 1.28 m/s, then the legs would be moving at approximately 2.57 m/s. So, the theoretical Doppler Shift was found to be approximately -300 Hz. The calculation matches closely with the observed values of Doppler shift. However, slight deviations were noted due to leg motion dynamics, where the periodic leg movement caused variations in the velocity profile of the human subject. The other reason for slight variations in observed Doppler shift could be due to multipath reflections from the experimental environment causing some fluctuations in the signal.

Table 5.2 summarizes the theoretical vs observed doppler shifts for each row of the ADC data file obtained from a single instance of a human subject moving AWAY from the sensor from a distance of 30 cm to 3 m away from the ultrasonic sensor.

Table 5.2: Theoretical vs Observed Doppler Shift (Human Walking Away)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	300 Hz	-136.36	39.86	40.29
Row -2	300 Hz	-320.96	39.68	39.94
Row -3	300 Hz	-347.61	39.65	39.58
Row -4	300 Hz	-243.88	39.76	39.58
Row -5	300 Hz	-222.66	39.78	39.94

It can be observed from the above table for amplitude-frequency plots that as the human subject started moving away marked with Row-1 and reached in the middle of the 3 meter walking path at Row-3, the negative Doppler Shift went from -136.36 Hz to -347.61 Hz, the walking speed slightly decreased while stopping at a distance of 3 meters from the sensor as marked in Row-4 and Row-5. Due to decrease in speed the negative Doppler Shift slightly increased from -243.88 Hz to -222.66 Hz respectively and at this point the sensor stopped recording the measurements. Figure 5.2 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of human walking away, highlighting the negative Doppler shift.

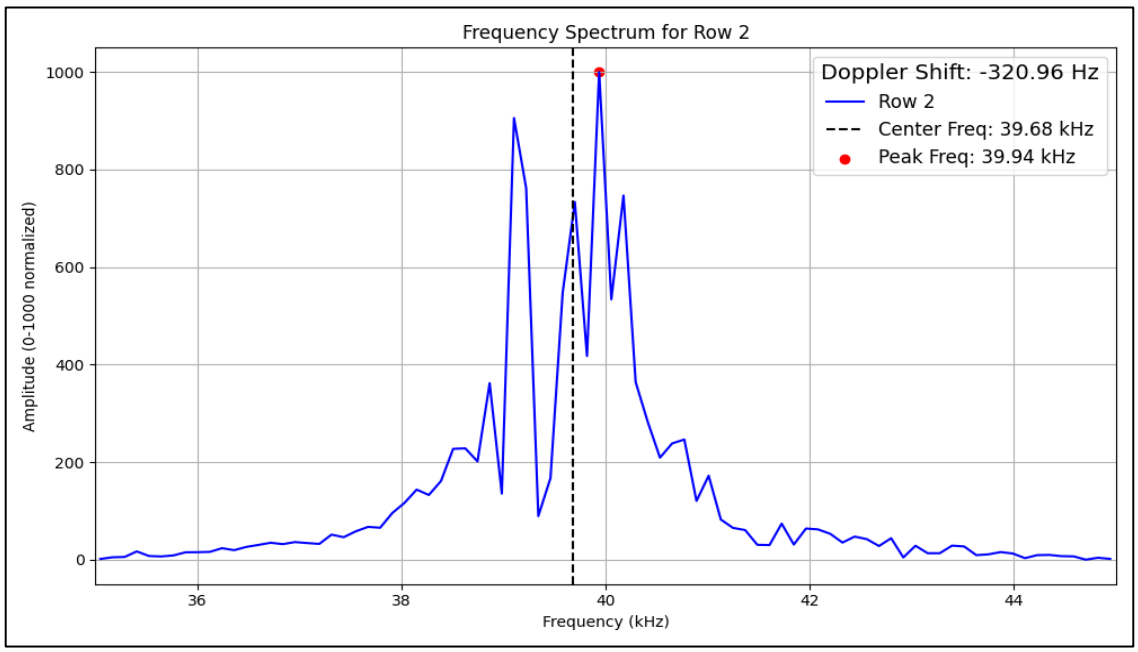


Figure 5.2: Amplitude-Frequency plot for Row-2 of the ADC Data file of human walking Away

5.1.2. Human Walking Sideways/Perpendicular Direction

In this case, no significant frequency shift was observed when the target (human) walked in perpendicular direction to the sensor. However, there is a slight fluctuation in frequency due to the sensor coverage angle and human leg dynamics. These fluctuations are not significant to be classified as Doppler shifts but may be a source of noise in some cases. The average speed of human subject was still 1.28 m/s but since there was no change in speed of subject in the direction parallel to sensor, the Doppler shift was not present as shown in Table 5.3 for an ADC data file obtained from one of the measurements. Figure 5.3 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of human walking sideways, highlighting the negligible Doppler shift.

Table 5.3: Theoretical vs Observed Doppler Shift (Human Walking Sideways / Perpendicular)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	0	-4.77	39.78	40.29
Row -2	0	1.59	40	39.7
Row -3	0	46.96	40	39.7
Row -4	0	-27.47	40.05	40.17
Row -5	0	-41.52	39.97	39.82

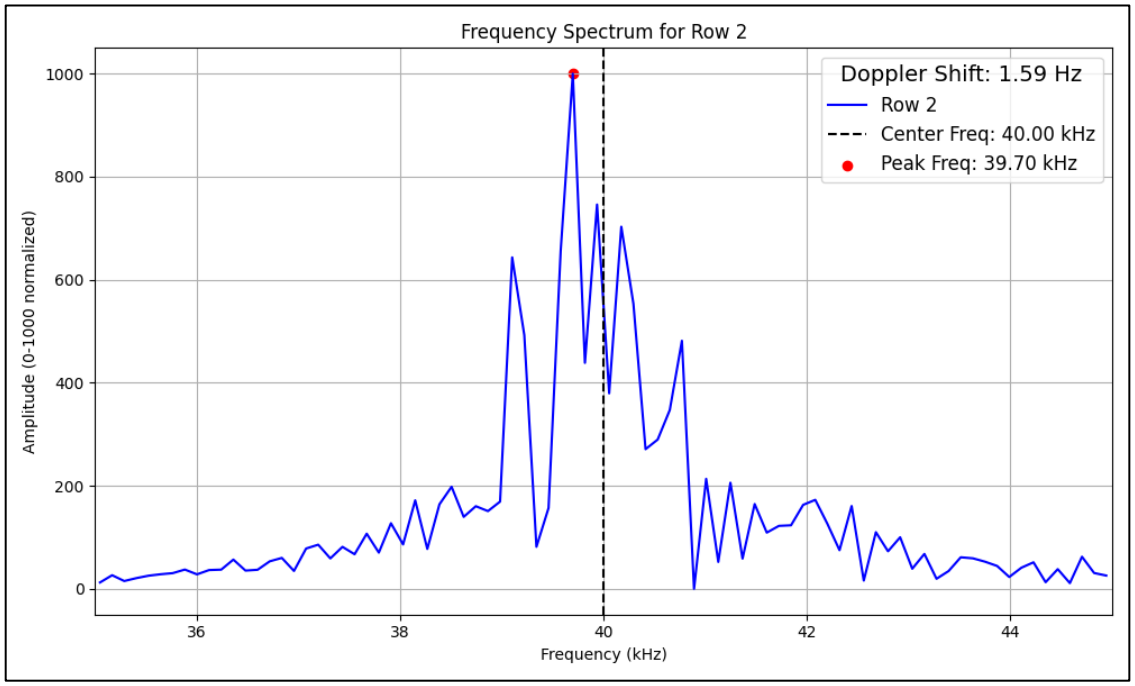


Figure 5.3: Amplitude-Frequency plot for Row-2 of the ADC Data file of human walking Sideways/Perpendicular

5.1.3. Human Standing Stationary

In the stationary case, no Doppler shift was observed, as expected. The frequency remained centered at the transmitted frequency (40 kHz). The were some fluctuations in received frequency due to multipath reflections as well as environmental noise interference in the parking as recorded in Table 5.4.

Table 5.4: Theoretical vs Observed Doppler Shift (Human Standing Stationary)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	0	-9.39	39.99	39.7
Row -2	0	2.88	40	39.7
Row -3	0	34.85	40.03	39.94
Row -4	0	7.05	40.01	39.82
Row -5	0	63.02	40.06	39.82

Figure 5.4 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of human standing stationery, highlighting the negligible Doppler shift.

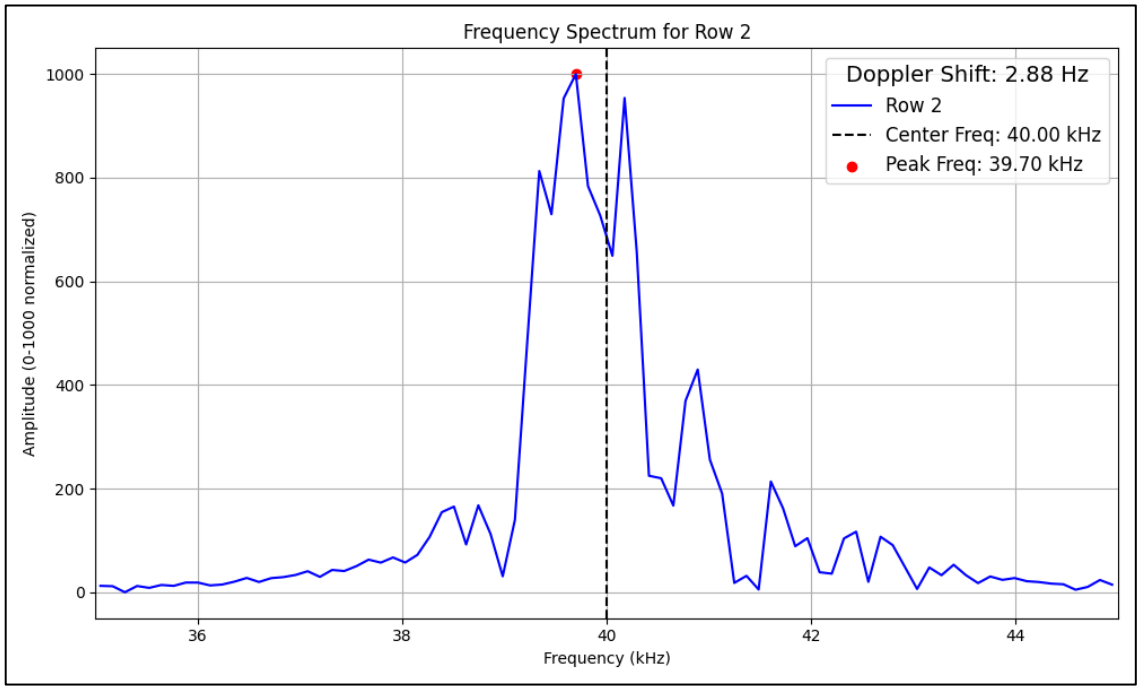


Figure 5.4: Amplitude-Frequency plot for Row-2 of the ADC Data file of Human Standing Stationary

5.1.4. Object Moving Towards

As described in section 4.4, the object was moved at a speed of around 2.6 m/s from 3m to 30 cm close towards the direction of sensor to observe measurable Doppler shift. The Doppler shift was similar to that of the human moving towards case. Due to the higher moving speed of the object, an average of 4 rows of ADC data measurement were collected in a single instance as shown Table 5.5. The absence of leg motion made the Doppler shifts more consistent as shown in the amplitude-frequency plot. Multipath reflections caused by surrounding surfaces were still a factor but less pronounced compared to human movement.

Table 5.5: Theoretical vs Observed Doppler Shift (Object Moving Towards)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	300	256.93	40.26	40.65
Row -2	300	335.43	40.34	40.41
Row -3	300	301.57	40.3	40.25
Row -4	300	104.68	40.10	40.17

It can be observed from the above table for amplitude-frequency plots that as the metal sheet (Object) started moving marked with Row-1 and reached in the middle of the 3 meter path at Row-2, the positive Doppler Shift went from 256.93 Hz to 335.43 Hz but as it approaches the sensor the moving speed

slightly decreased while stopping at a distance of 30 centimetres from the sensor as marked in Row-3 and Row-4. Due to decrease in speed the positive Doppler Shift slightly decreased from 301.57 Hz to 104.68 Hz respectively and at this point the sensor stopped recording the measurements. Figure 5.5 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of Object Moving Towards the sensor, highlighting the positive Doppler shift.

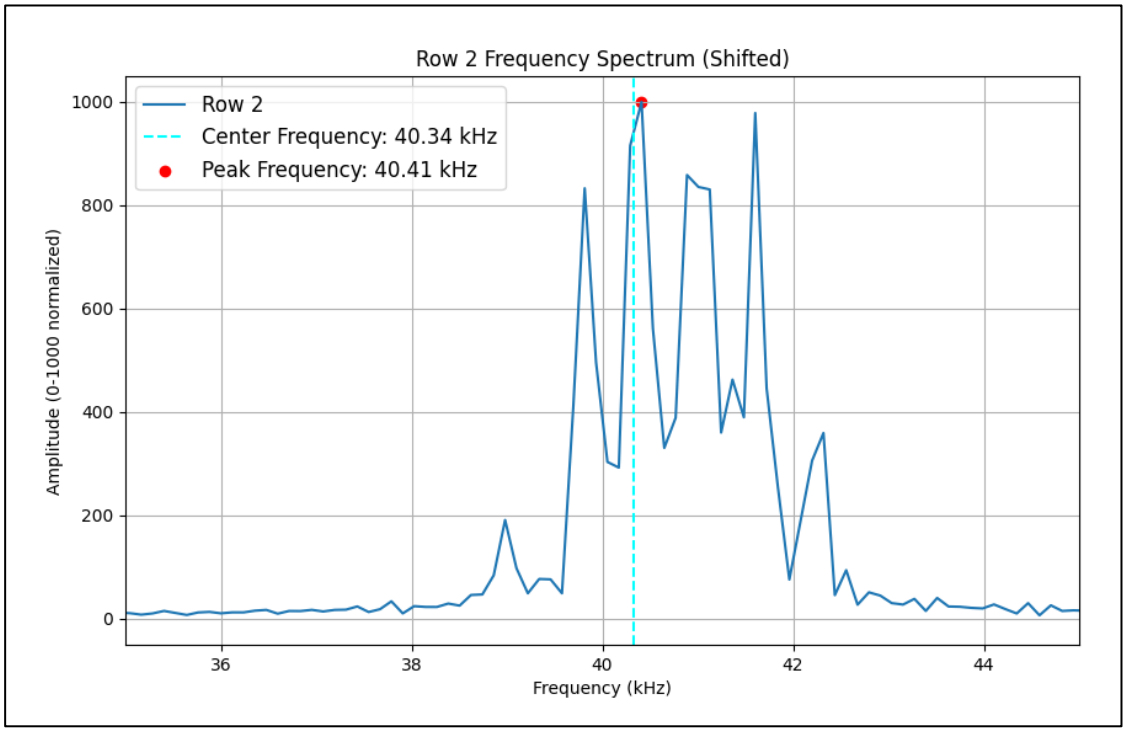


Figure 5.5: Amplitude-Frequency plot for Row-2 of the ADC data file of Object Moving Towards the sensor, highlighting the positive Doppler shift.

5.1.5. Object Moving Away

As expected, a negative Doppler shift for the object moving away was recorded. The shifts were smooth and very close to the theoretical values, since there is a lack of body dynamics as those in humans.

Table 5.6: Theoretical vs Observed Doppler Shift (Object Moving Away)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	-300	-196.48	39.8	40.29
Row -2	-300	-285.98	39.71	40.41
Row -3	-300	-414.78	39.59	40.41
Row -4	-300	-126.38	39.87	41.01

It can be observed from Table 5.6 for amplitude-frequency plots that as the object started moving away marked with Row-1 and reached in the middle of the 3 meter walking path at Row-3, the negative Doppler Shift went from –196.48 Hz to -414.78 Hz, the movement speed slightly decreased while stopping at a distance of 3 meters from the sensor as marked in Row-4. Due to decrease in speed the negative Doppler Shift slightly increased from -414.78 Hz to -126.38 Hz respectively and at this point the sensor stopped recording the measurements. Multipath reflections caused by surrounding surfaces were still a factor but less pronounced compared to human movement. There might also be some interference from the sound created by movement of the Object since it was moved on a stool fitted with rollers. Figure 5.6 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of Object Moving Away from the sensor, highlighting the negative Doppler shift.

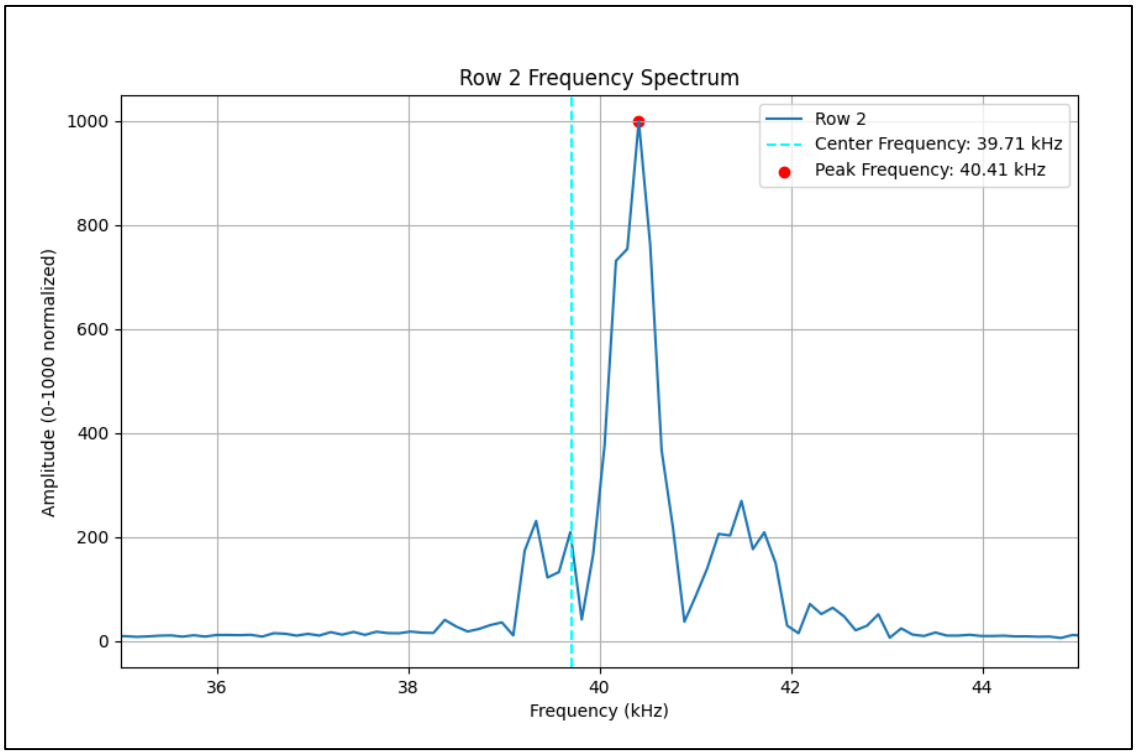


Figure 5.6: Amplitude-Frequency plot for Row-2 of the ADC data file of Object Moving Away from the sensor

5.1.6. Object Moving Sideways/Perpendicular Direction

In this case, no significant frequency shift was observed when the target (Object) was moved in perpendicular direction to the sensor with the received signal centered around the transmitted frequency. However, there is a slight fluctuation in frequency due to the sensor coverage angle. These fluctuations are not significant to be classified as Doppler shifts but may be a source of noise in some cases due to noise created by moving roller stool on which the object was attached. Since there was no change in speed of subject in the direction parallel to sensor, the Doppler shift was not present as shown in Table 5.7 for an ADC data file obtained from one of the measurements.

Table 5.7: Theoretical vs Observed Doppler Shift (Object Moving Sideways / Perpendicular)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	0	66.65	40.07	40.29
Row -2	0	79.75	40.08	41.25
Row -3	0	-73.75	39.93	40.65
Row -4	0	72.71	40.07	40.29

Figure 5.7 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of Object Moving Perpendicular/Sideways, highlighting the negligible Doppler shift.

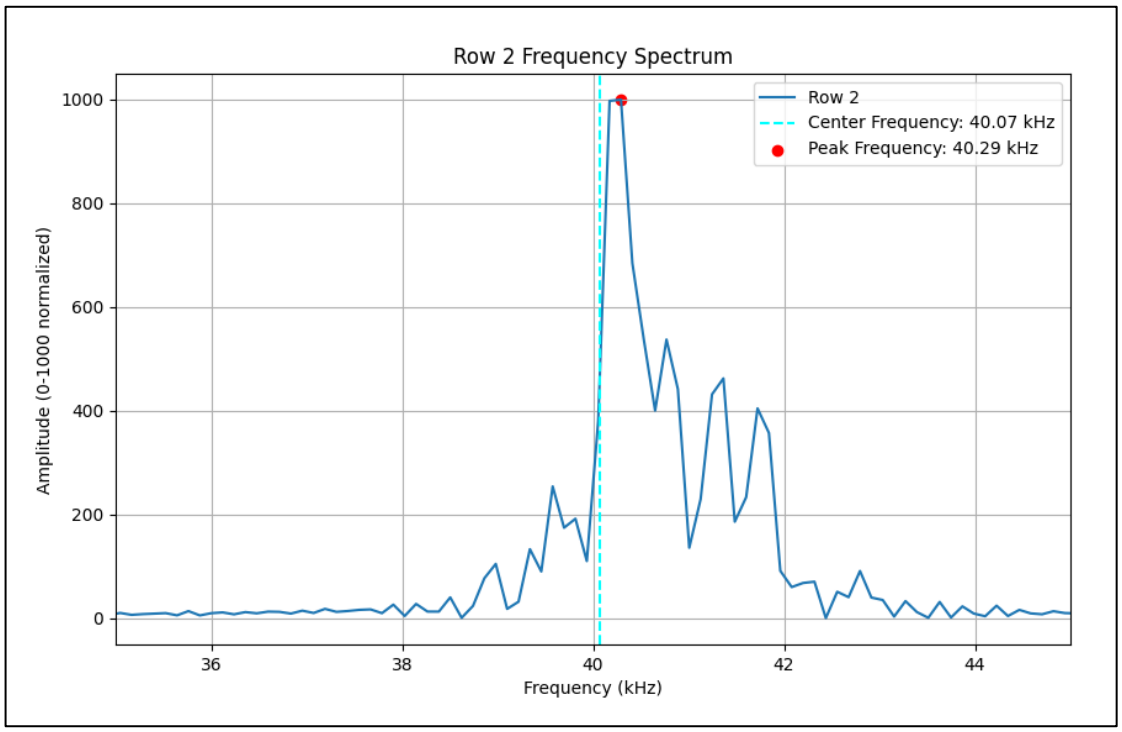


Figure 5.7: Amplitude-Frequency plot for Row-2 of the ADC data file of Object Moving Perpendicular/Sideways

5.1.7. Object Stationary

Similar to standing Human, in the case of stationary object also, no Doppler shift was observed, as expected. The frequency remained centered at the transmitted frequency (40 kHz). The were some fluctuations in received frequency due to multipath reflections as well as environmental noise interference in the parking as recorded in Table 5.8. Figure 5.8 shows the Amplitude-Frequency plot for Row-2 of the ADC data file of Object standing stationery, highlighting the negligible Doppler shift.

Table 5.8: Theoretical vs Observed Doppler Shift (Object Stationary)

Row No.	Theoretical Doppler Shift (Hz)	Observed Doppler Shift (Hz)	Center Frequency (kHz)	Peak Frequency (kHz)
Row -1	0	19.38	40.02	40.05
Row -2	0	51.47	40.05	41.48
Row -3	0	-8.15	39.99	41.37
Row -4	0	72.21	40.07	40.29

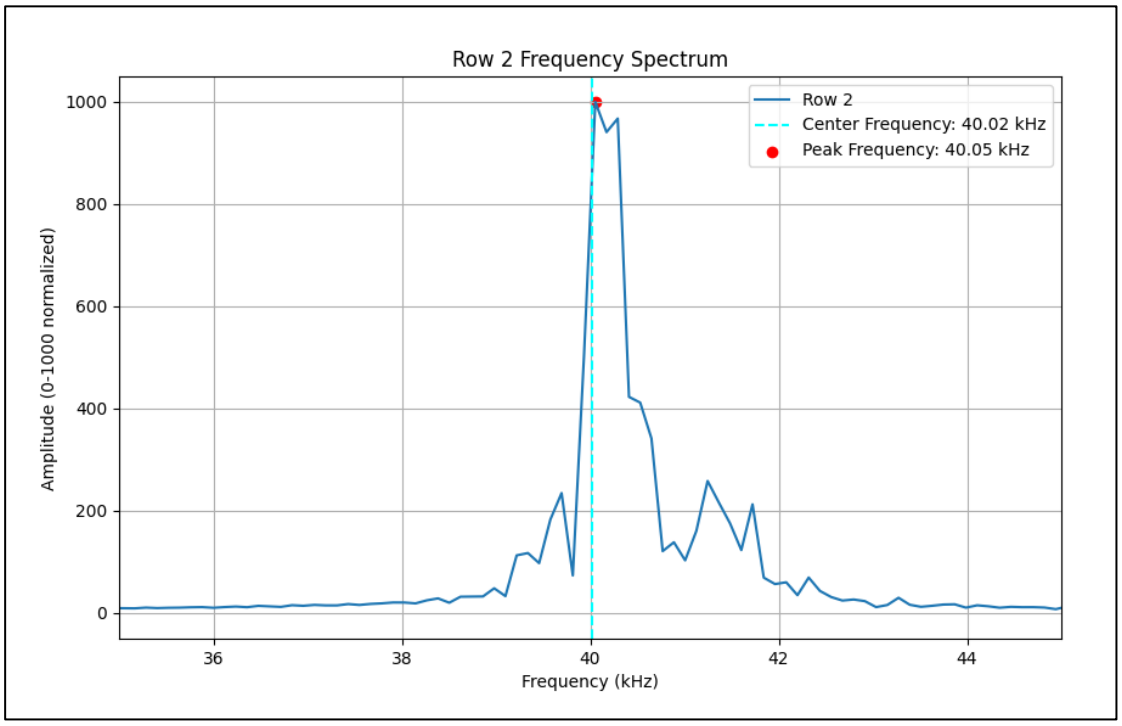


Figure 5.8: Amplitude-Frequency plot for Row-2 of the ADC data file of Object Stationary

5.2. Comparative Analysis of Human and Object Movements

Figure 5.9 and Figure 5.10 shows the comparative analysis between amplitude-frequency plots for human and object movements in four cases: moving towards, moving away, moving perpendicular, and stationary. We try to evidence distinct frequency-domain characteristics that might be different depending on movement direction and type, by overlaying the amplitude-frequency plot of each case in a single frame. These comparative analyses outline the manner with which Doppler shifts and spectral features change with movement type and provide a basis for distinguishing human from object movements by unique frequency patterns.

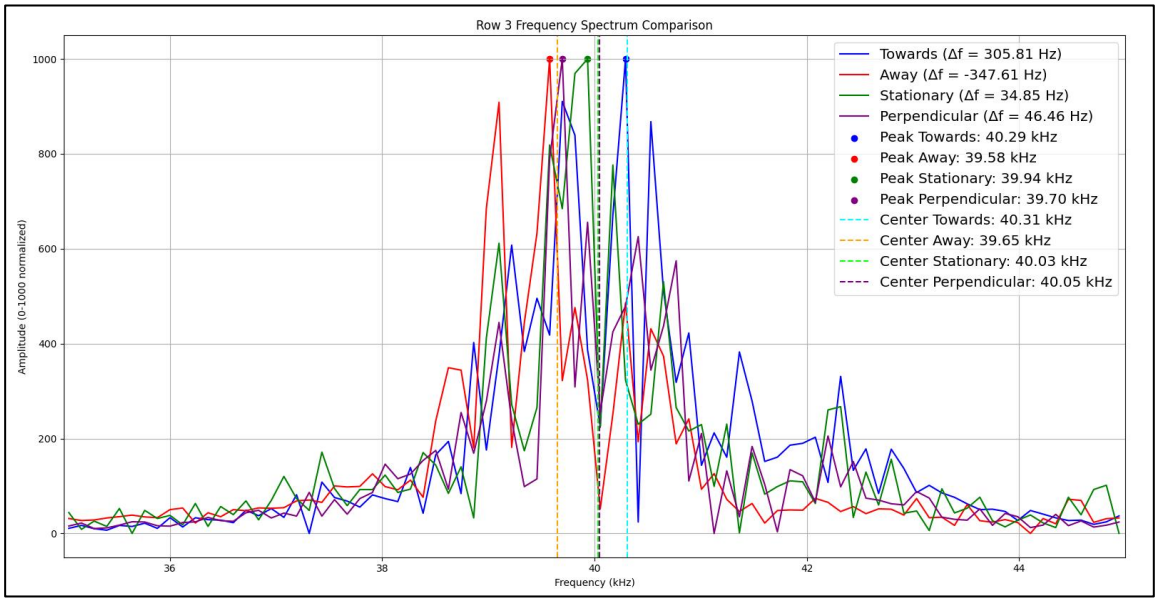


Figure 5.9: Overlaid amplitude-frequency plots of Row-2 for all four human movement cases demonstrating the shift in frequency

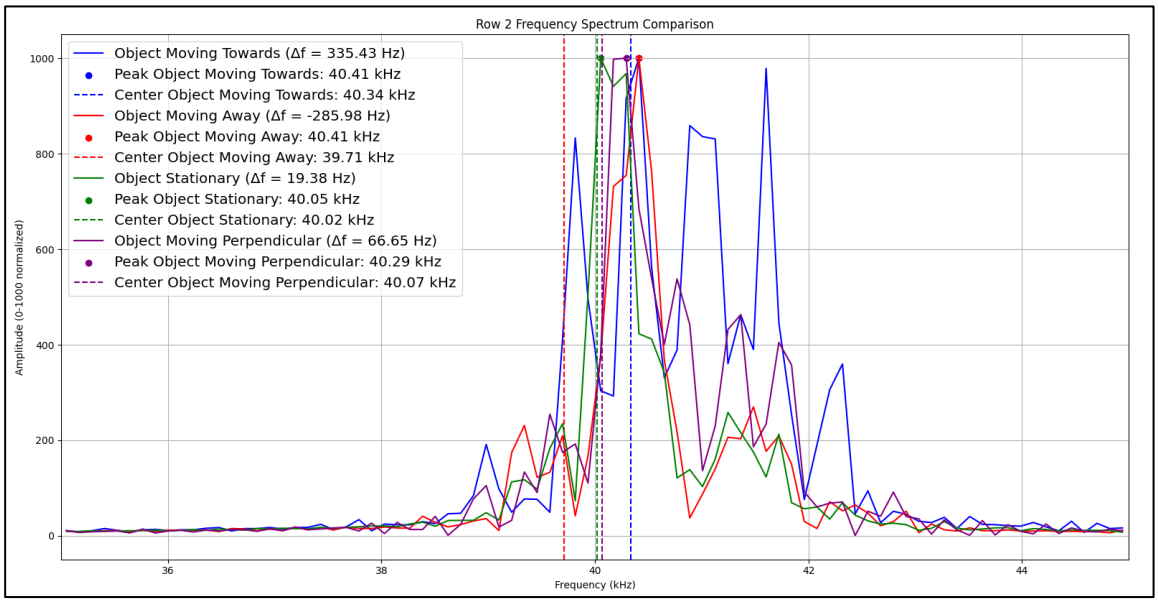


Figure 5.10: Overlaid amplitude-frequency plots of Row-2 for all four Object movement cases demonstrating the shift in frequency

From the above plots, it can be observed that human motions showed more variations in the Doppler shift due to the dynamics of the legs and irregular movement. Object movements, however, were smooth and relatively less fluctuating. The body structure reflected multipath effects more strongly for humans than for objects which caused more fluctuations.

5.3. Machine Learning Model Performance

This section highlights the classification performance for two different machine learning models - MLP and CNN, trained separately with FFT data, representing human and object movements captured under different conditions. Both were separately trained and assessed with regard to particular cases: stationary, perpendicular, moving towards, and moving away. In

addition, various performance metrics were computed: accuracy, precision, recall, and F1-score, developing a view to establish how effectively each model performed to discriminate between human and object motions.

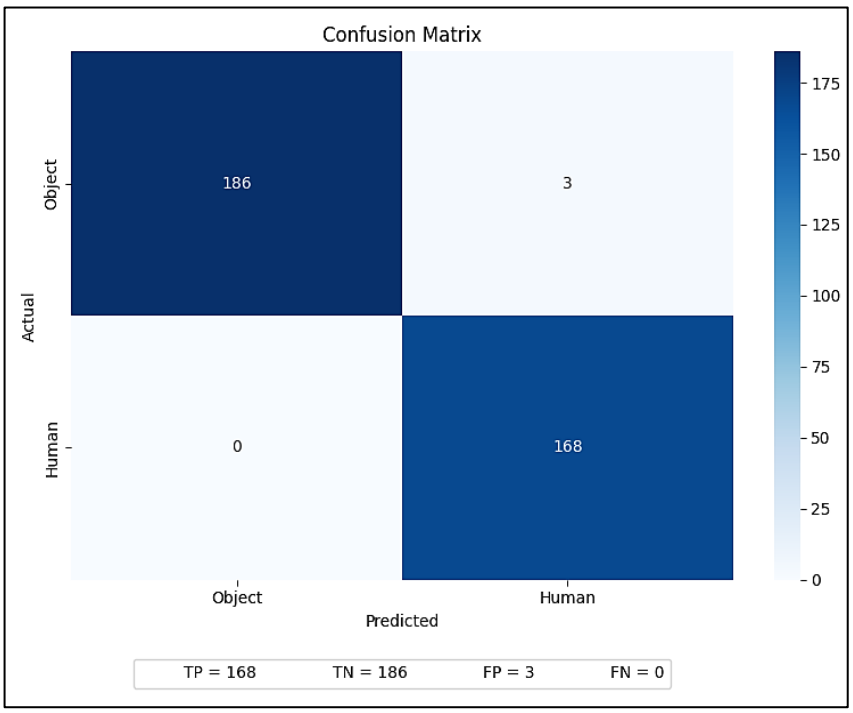
5.3.1. Model Performance on Stationary Human vs. Object Dataset

This section highlights the performance of both MLP and CNN models that were trained on the stationary human-object dataset and evaluated on the same stationary dataset to establish a baseline performance. Table 5.9 summarises the performance metrices of both MLP and CNN for stationary datasets.

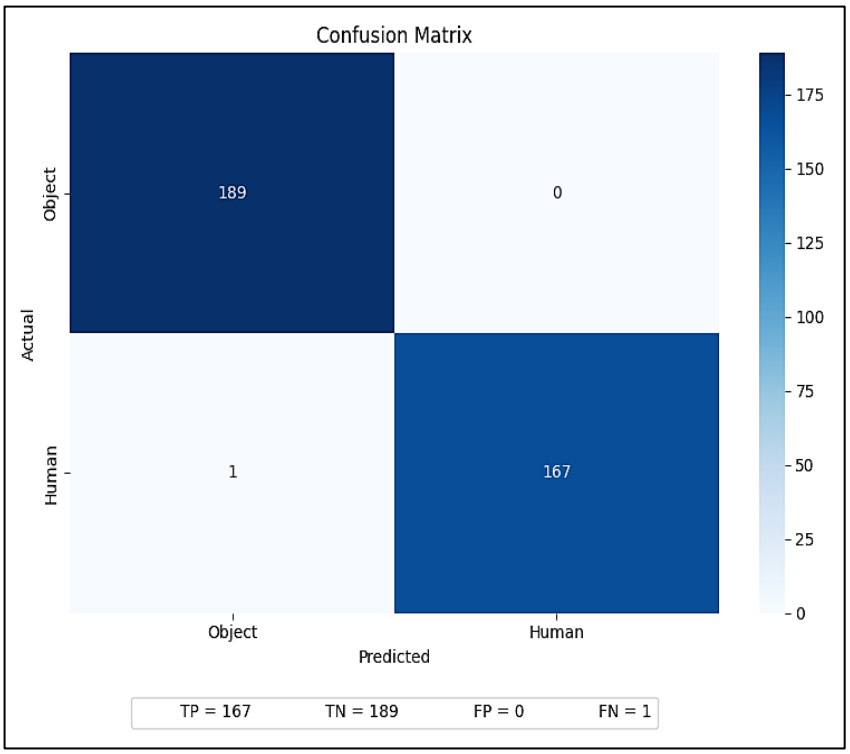
Table 5.9: Performance summary of MLP and CNN models on stationary human vs. object dataset

Metric	MLP (Stationary)	CNN (Stationary)
True Positive (TP)	168	167
True Negative (TN)	186	189
False Positive (FP)	3	0
False Negative (FN)	0	1
Accuracy	0.9915	0.9971
Precision	0.9917	0.9972
Recall	0.9915	0.9971
F1 Score	0.9916	0.9971

- **MLP Model Evaluation:** From the above table and Figure 5.11 it can be clearly observed that high accuracy, precision, F1 score and recall of 99.1% achieved in the stationary data performance of the MLP model signified that the system is able to identify the signature of the stationary human and object well. In fact, the absence of Doppler shifts in the stationary data makes it simple for the MLP model to focus exclusively on the frequency-based characteristics unique for the class.
- **CNN Model Evaluation:** The CNN model had performance metrics that were also strong, much like the MLP model as can be observed from Figure 5.11. The model was very robust for stationary data, benefiting from the capability of its convolutional layers to extract local frequency features, giving a slightly higher accuracy, precision, recall and F1 Score of 99.7%.



(a)



(b)

Figure 5.11: Confusion matrices for (a) MLP and (b) CNN models trained and tested on stationary data.

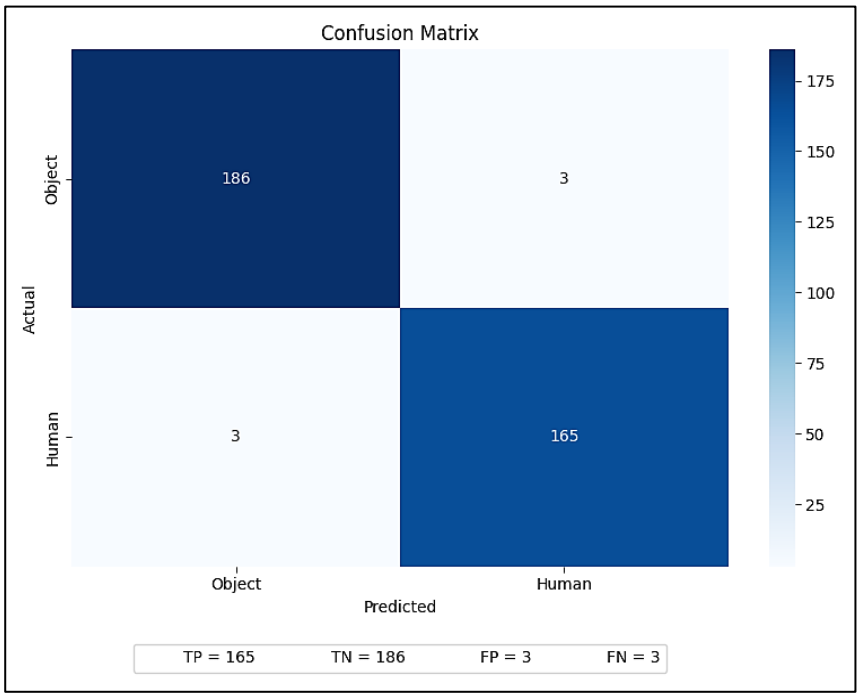
5.3.2. Model Performance on Perpendicular/Sideways Human vs. Object Dataset

In this case, MLP and CNN models were trained and evaluated on data representing human and object movements perpendicular to the sensor; hence,

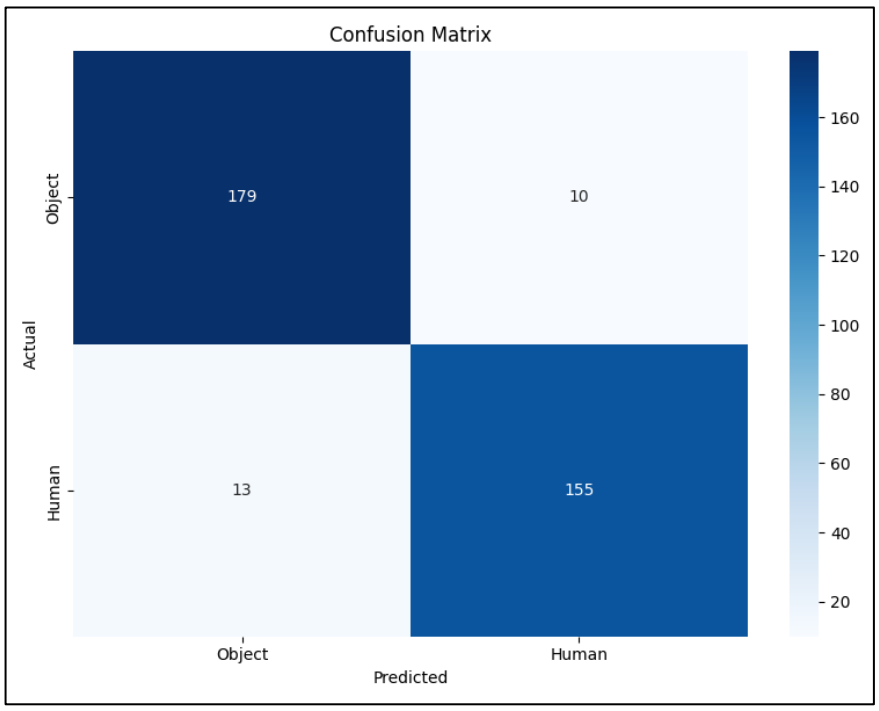
this scenario evaluates the robustness of the models on non-Doppler-influenced data, since the perpendicular movements induce minimal Doppler shifts. Table 5.10 summarises the performance metrices of both MLP and CNN for perpendicular/sideways movement datasets.

Table 5.10: Performance summary of MLP and CNN models on perpendicular human vs. object dataset

Metric	MLP (Perpendicular)	CNN (Perpendicular)
True Positive (TP)	165	155
True Negative (TN)	186	179
False Positive (FP)	3	10
False Negative (FN)	3	13
Accuracy	0.9831	0.9355
Precision	0.9831	0.9356
Recall	0.9831	0.9355
F1 Score	0.9831	0.9355



(a)



(b)

Figure 5.12: Confusion matrices for (a) MLP and (b)CNN models trained and tested on perpendicular/sideways movement data.

- **MLP Model Evaluation:** From the above table and Figure 5.12 it can be clearly observed that high accuracy, precision, F1 score and recall of 98.31% achieved in the perpendicular data performance of the MLP model signified that the system is able to identify the signature of the sideways moving human and object well. The performance is slightly less than what was observed in Stationary dataset since during sideways motion there was instances when the human or object was not directly in front of the sensor, causing occasional frequency overlapping, leading to certain level of misclassification.
- **CNN Model Evaluation:** The CNN model also showed a consistent performance under all the four metrices with an accuracy, precision, F1 score and recall of 93.55 % as can be observed from above table and Figure 5.12: Confusion matrices for (a) MLP and (b)CNN models trained and tested on perpendicular/sideways movement data.Figure 5.12. Though the overall performance is good but the efficiency of the CNN in this case is approximately 5% lesser than the MLP model for the same dataset and 6% lesser when trained and tested on stationary dataset. The slight drop in performance for perpendicular movement, might be because the signal patterns due to this type of movement are intrinsically more complex or variable. Slight inconsistency in the ultrasonic reflections may be created with small lateral or angular deviations in the path followed by the object/human, making any accurate pattern hard to capture for a model.

5.3.3. Model Performance on Moving Towards Human vs. Object Dataset

In this case, MLP and CNN models were trained and evaluated on data representing human and object movements towards the ultrasonic sensor, a condition that leads to Doppler shifts resulting in a unique set of frequency-based characteristics. Table 5.11 summarises the performance metrices of both MLP and CNN for movement towards datasets.

Table 5.11: Performance summary of MLP and CNN models on moving-towards human vs. object dataset

Metric	MLP (Stationary)	CNN (Stationary)
True Positive (TP)	168	164
True Negative (TN)	189	187
False Positive (FP)	0	2
False Negative (FN)	0	4
Accuracy	1.0	0.9831
Precision	1.0	0.9832
Recall	1.0	0.9831
F1 Score	1.0	0.9831

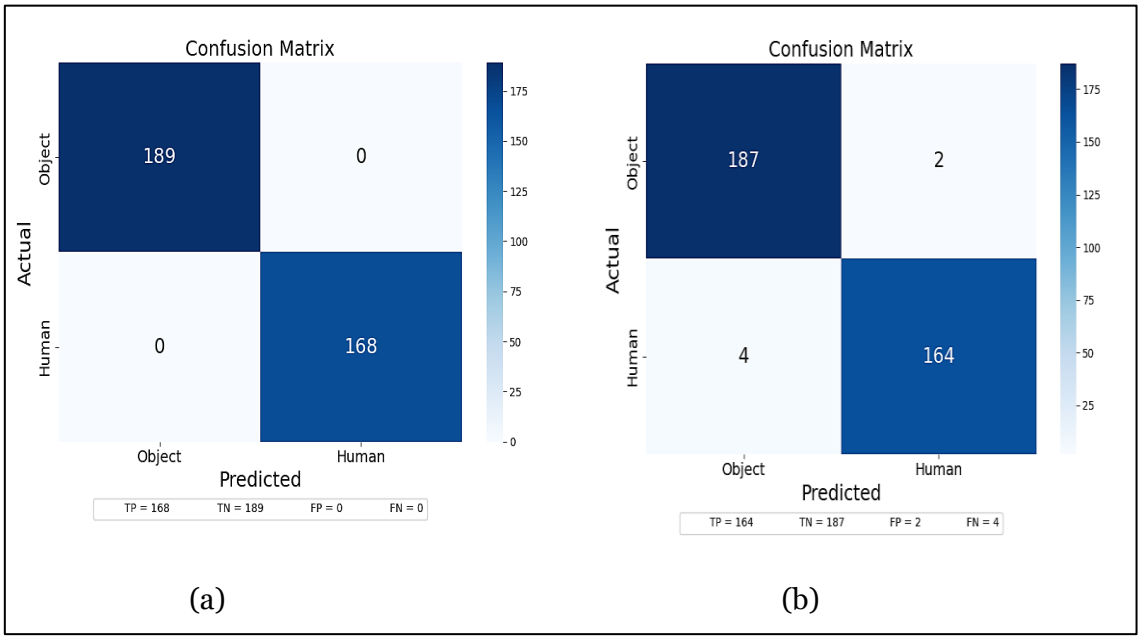


Figure 5.13: Confusion matrices for (a) MLP and (b)CNN models trained and tested on movement towards the sensor data.

- **MLP Model Evaluation:** The performance of the MLP model is almost perfect, with all metrics achieving a value of 1.0: accuracy, precision, recall, and F1 score as observed in Figure 5.13. That is quite a great performance-the model answers with sure-footedness in knowing how to differentiate between human and object motions of the situation under consideration. These high

values are indicative that the frequency patterns and their spectral features for the case "moving towards" were well-captured in the MLP model. Considering that the MLPs have a simpler architecture, it may be the case that this type of model is more sensitive to sharp frequency shifts associated with the Doppler effect, which is rather pronounced in this case. That configuration may result in clearer frequency changes which the MLP can effectively classify.

- **CNN Model Evaluation:** By contrast, the CNN model had slightly less performance values compared to the MLP; its accuracy, precision, recall, and F1 scores reached 0.9831 as observed in Figure 5.13. This minor decrease in performance indicates that while the CNN can almost perfectly classify most of the classes, there are some cases that cannot be classified accurately, probably because the CNN is extremely sensitive to slight variations in spatial patterns in the frequency domain. Generally speaking, 1D convolutional layers from CNN are more effective in feature extraction of spatial dependencies and work better in cases where there can be more hidden, detailed patterns across cases. With the "moving towards" case, given the more pronounced and linear increase of the Doppler shift and frequency, the simpler nature of the MLP model could have justified a lesser level of sensitivity to small variances. The performance of 0.9831 for all metrics for the CNN model, although well performing, is small in comparison with that of the MLP.

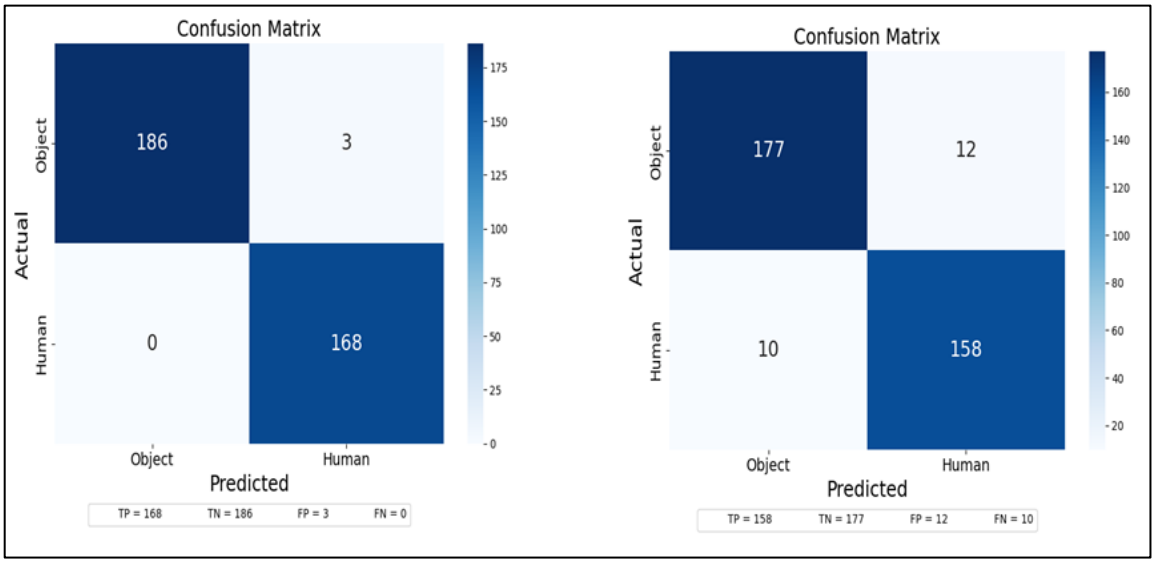
5.3.4. Model Performance on Moving Away Human vs. Object Dataset

Here, we show the performance of classification of MLP and CNN trained and tested with the "moving away" human vs. object dataset. This configuration introduces unique Doppler-induced frequency shifts whereby objects moving away from the sensor produce a frequency drop. We will then be able to study in detail the performance of each model in terms of its capability of differentiating between human and object motion for this given condition. *Table 5.12* summarises the performance metrices of both MLP and CNN for movement Away datasets.

Table 5.12: Performance summary of MLP and CNN models on moving-away human vs. object dataset

Metric	MLP (Stationary)	CNN (Stationary)
True Positive (TP)	168	158
True Negative (TN)	186	177
False Positive (FP)	3	12
False Negative (FN)	0	10
Accuracy	0.9915	0.9383
Precision	0.9917	0.9384

Recall	0.9915	0.9383
F1 Score	0.9916	0.9383



(a) (b)

Figure 5.14: Confusion matrices for (a) MLP and (b)CNN models trained and tested on movement away the sensor data.

- **MLP Model Evaluation:** This "moving away" classification also saw very good performance from the MLP model, at approximately 0.9916 for accuracy, precision, recall, and F1 score as observed in Figure 5.14. These near-perfect metrics are indicative that, under these conditions, the movement made either by the human or the object could be effectively classified using the MLP model. The success of the model here underlines the robustness of the MLP in capturing and leveraging consistent frequency shifts associated with Doppler effects. Although not as pronounced as in the "moving towards" case, there is distinct spectral change in the "moving away" case that the MLP model is able to correctly classify. High values of the metrics suggest that the MLP model effectively generalized across variations in spectral features present in this dataset.
- **CNN Model Evaluation:** In this scenario, the metrics are a little lower for the CNN model, averaging about 0.9383 for accuracy, precision, recall, and F1 score as observed in Figure 5.14. This decrease shows that there is some drop in the performance of the CNN classifying human and object movements when events related to the "moving away" scenario are considered. This performance from the CNN shows that the network had some problems learning the pattern of frequency shifts that were consistent in this scenario. While CNNs do better with complex data or when frequencies are spatially variant, they tend to be more susceptible to cases that involve simple and linearly changing frequencies, such as "moving away," since changes of amplitude might not make much difference in overall classification. Although effective, the CNN metrics are higher in misclassification rate compared to the MLP model.

5.4. Cross-Movement Evaluation of Pretrained Models

This section explores the classification performance of MLP and CNN models, pre-trained on stationary and perpendicular datasets, and tested on unseen scenarios of movement: moving towards and moving away. This approach allowed us to understand how the models which are trained on non-Doppler dataset (Stationary and Perpendicular) perform on predicting the labels of dataset effected with Doppler Effect (Moving Towards or Away) and also allowed us to observe the improvement in classification performance when the models were specifically trained on movement scenarios (moving towards or away).

5.4.1. Evaluation of Stationary Pretrained Model on Moving Towards Dataset

Table 5.13 summarizes the performance of the pre-trained MLP and CNN models trained on stationary data for the cross-evaluation of moving towards dataset.

Table 5.13: Cross-movement evaluation of stationary-trained MLP and CNN models on moving Towards datasets

Metric	MLP (Stationary, tested on Moving Towards)	CNN (Stationary, tested on Moving Towards)
True Positive (TP)	546	594
True Negative (TN)	90	39
False Positive (FP)	504	555
False Negative (FN)	48	0
Accuracy	0.5353	0.5328
Precision	0.5860	0.7584
Recall	0.5353	0.5328
F1 Score	0.4550	0.4024

a) MLP Model Performance on cross-evaluation

The stationary-trained MLP model when tested with moving towards dataset gives an accuracy of 0.5353 for the moving towards dataset, with precision and recall equal to 0.5860 and 0.5353, respectively, and F1 score equal to 0.4550. By contrast, the MLP model, when trained and tested on the dataset representing "Moving Towards", achieved an ideal score of 1.00 for all metrics: accuracy, precision, recall, and F1 score. This is quite a stark difference and underlines the model's inability to generalize easily from stationary to moving towards cases. Below graph Figure 5.15 shows the drop in performance of the MLP model when given the task of predicting moving towards dataset.

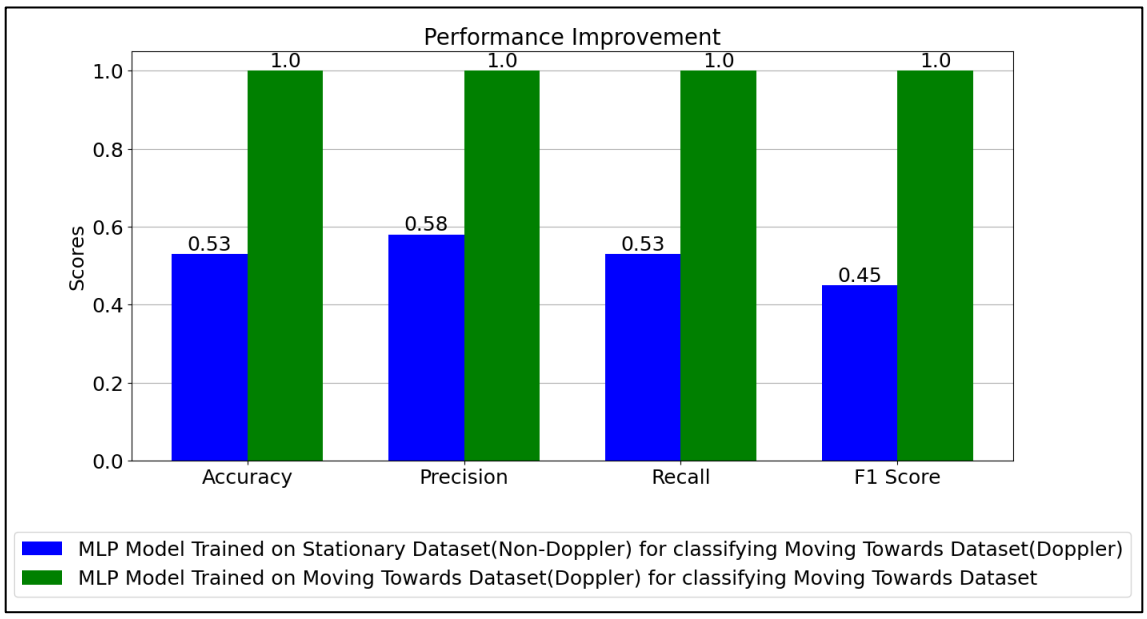


Figure 5.15: Graph comparing MLP cross-evaluation on stationary vs. moving towards dataset with original MLP results for moving towards dataset.

b) CNN Model Performance on cross-evaluation

From the above Figure 5.16 it can be observed that the CNN model that was trained with the stationary data and applied for the moving towards dataset yielded an accuracy of 0.5328, a precision of 0.7585, recall of 0.5328, and an F1 score of 0.4024. While training and testing the CNN model on the dataset of moving towards, all categories reached close to 0.99 metrics, showing its robustness when trained only on that particular scenario. In this regard, a decline shown in cross-evaluation further shows the impact of different Doppler shifts in cross-movement testing.

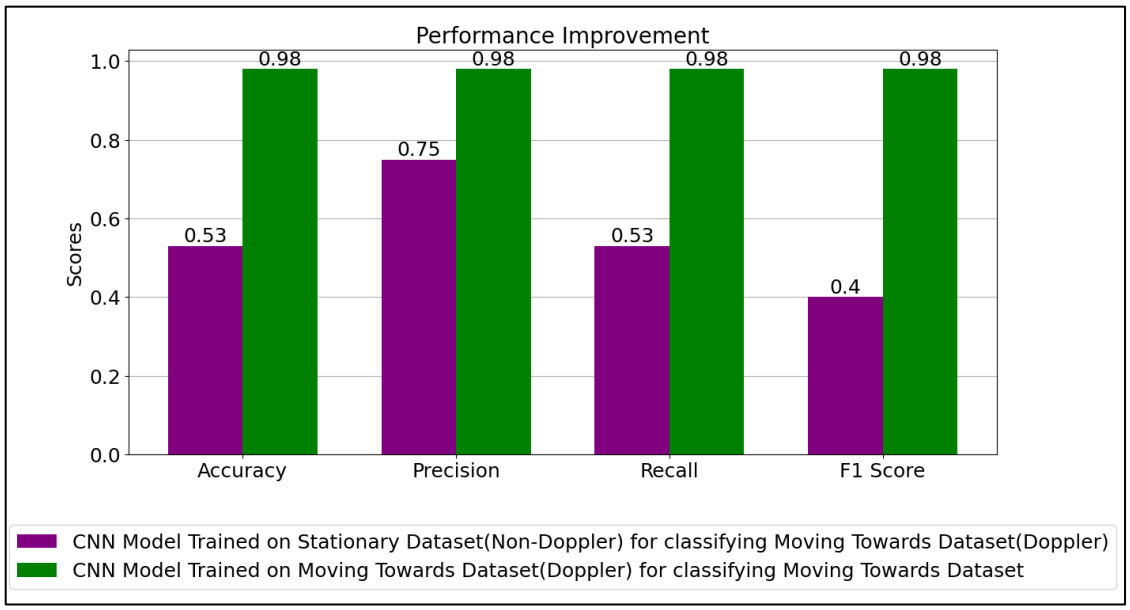


Figure 5.16: Graph comparing CNN cross-evaluation on stationary vs. moving towards dataset with original CNN results for moving towards dataset.

5.4.2. Evaluation of Stationary Pretrained Model on Moving Away Dataset

Table 5.14 summarizes the performance of the pre-trained MLP and CNN models trained on stationary data for the cross-evaluation of moving away dataset.

Table 5.14: Cross-movement evaluation of stationary-trained MLP and CNN models on moving Away datasets

Metric	MLP (Stationary, tested on Moving Away)	CNN (Stationary, tested on Moving Away)
True Positive (TP)	486	591
True Negative (TN)	129	39
False Positive (FP)	465	555
False Negative (FN)	108	3
Accuracy	0.5176	0.5303
Precision	0.5276	0.7221
Recall	0.5176	0.5303
F1 Score	0.4697	0.4009

a) MLP Model Performance on cross-evaluation

When testing the stationary-trained MLP model with the dataset for moving away scenario, the results achieved an accuracy of 0.5176, precision of 0.5276, recall of 0.5176, and F1 score of 0.4697. However, the accuracy, precision, recall, and F1 score were around 0.99 when training and testing was done on the same kind of dataset (moving away) to test the MLP model. This big difference here indicates that the model generalizes poorly to hold cases with different Doppler shifts. Below graph in Figure 5.17 shows the drop in performance of the MLP model when given the task of predicting moving away dataset.

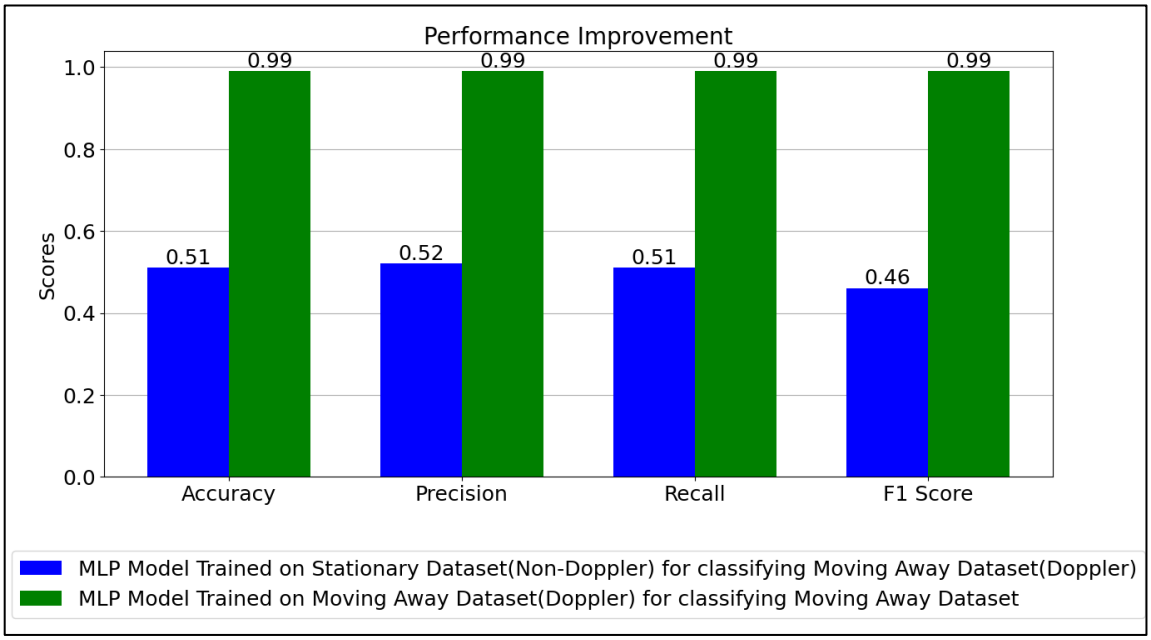


Figure 5.17: Graph comparing MLP cross-evaluation on stationary vs. moving Away dataset with original MLP results for moving Away dataset.

b) CNN Model Performance on cross-evaluation

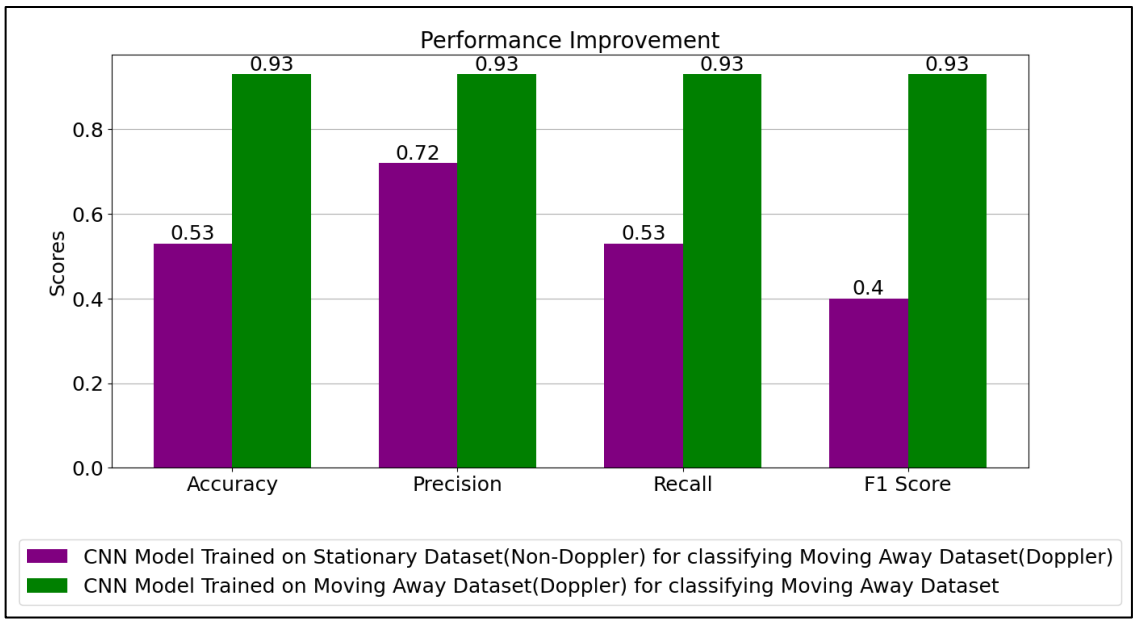


Figure 5.18: Graph comparing CNN cross-evaluation on stationary vs. moving Away dataset with original CNN results for moving Away dataset.

From the above Figure 5.18 , it can be observed that the CNN model based on the stationary recorded data has achieved an accuracy of 0.5303, precision-0.7221, recall-0.5303, and F1 score of 0.4010 in predicting the moving away dataset. However, when it was trained and tested with the moving away dataset, performance metrics for the CNN model were at approximately 0.93 across all performance indicators, emphasizing that CNN has been more robust compared to MLP when tested on new Doppler-shifted data.

5.4.3. Evaluation of Perpendicular Pretrained Model on Moving Towards Dataset

Table 5.15 summarizes the performance of the pre-trained MLP and CNN models trained on perpendicular data for the cross-evaluation of moving towards dataset.

Table 5.15:Cross-movement evaluation of perpendicular-trained MLP and CNN models on moving Towards datasets.

Metric	MLP (Perpendicular, tested on Moving Towards)	CNN (Perpendicular, tested on Moving Towards)
True Positive (TP)	318	315
True Negative (TN)	357	432
False Positive (FP)	237	162
False Negative (FN)	276	279
Accuracy	0.5681	0.6287
Precision	0.5684	0.6339
Recall	0.5681	0.6287
F1 Score	0.5677	0.6251

a) MLP Model Performance on cross-evaluation

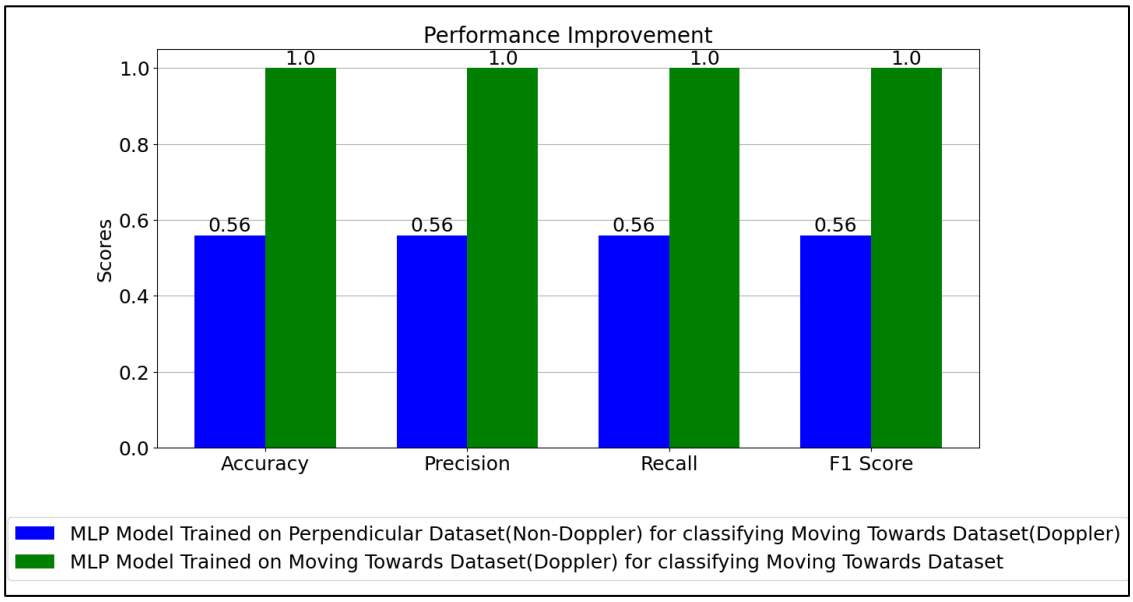


Figure 5.19: Graph comparing MLP cross-evaluation on Perpendicular vs. moving Towards dataset with original MLP results for moving Towards dataset.

Figure 5.19 shows that the perpendicular-trained MLP model on the moving towards dataset yielded 0.5681 in terms of accuracy, 0.5684 in terms of

precision, 0.5681 in terms of recall, and 0.5677 for the F1 score. The cross-movement performance metrics are significantly lower compared to the original performance; when the MLP model was trained and tested on the moving towards dataset, it reached 1.00 across all metrics.

b) CNN Model Performance on cross-evaluation

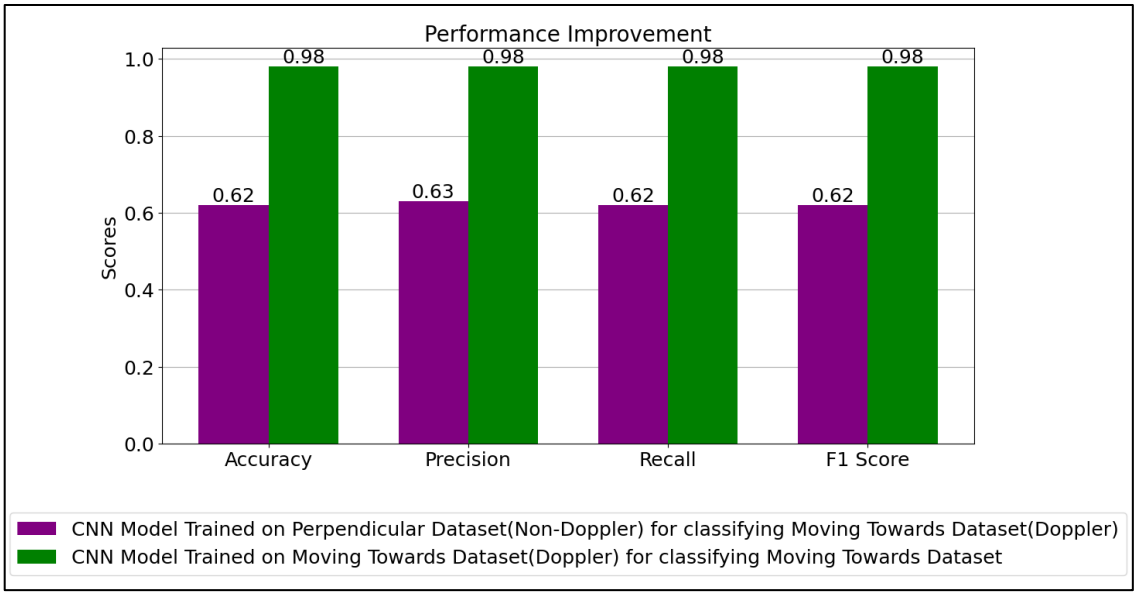


Figure 5.20: Graph comparing CNN cross-evaluation on perpendicular vs. moving Towards dataset with original CNN results for moving Towards dataset.

From the above Figure 5.20 it can be observed that the perpendicular-trained CNN model yielded an accuracy of 0.6288, a precision of 0.6340, a recall of 0.6288, and an F1 score of 0.6252 for the moving towards dataset. On the other hand, when both training and testing was on the moving towards dataset, this CNN model resulted in metrics around 0.99, showing a big performance difference in cross-evaluation, which again points out the dependence of performance on training data coming from a given type of movement.

5.4.4. Evaluation of Perpendicular Pretrained Model on Moving Away Dataset

Table 5.16 summarizes the performance of the pre-trained MLP and CNN models trained on perpendicular data for the cross-evaluation of moving Away dataset.

Table 5.16: Cross-movement evaluation of perpendicular-trained MLP and CNN models on moving Away datasets.

Metric	MLP (Perpendicular, tested on Moving Away)	CNN (Perpendicular, tested on Moving Away)
True Positive (TP)	270	363
True Negative (TN)	384	519
False Positive (FP)	210	75

False Negative (FN)	324	231
Accuracy	0.5505	0.7424
Precision	0.5524	0.7603
Recall	0.5505	0.7424
F1 Score	0.5463	0.7379

a) MLP Model Performance on cross-evaluation

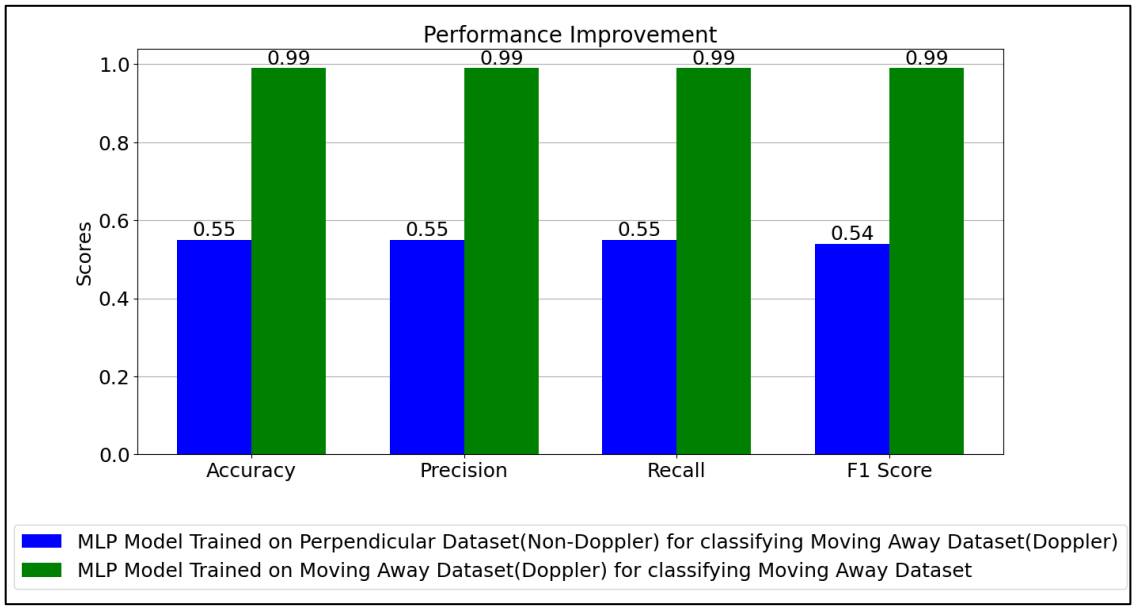


Figure 5.21: Graph comparing MLP cross-evaluation on Perpendicular vs. moving Away dataset with original MLP results for moving Away dataset.

From the above graph Figure 5.21 it can be observed that the perpendicular-trained MLP model yielded an accuracy of 0.5505, a precision of 0.5524, a recall of 0.5505, and an F1 score of 0.5463 when tested with moving away dataset. In contrast, when training and testing the MLP model on moving away dataset yielded a 0.99 score for all metrics, it indicated that it drastically degraded in performance compared to cross-evaluation.

b) CNN Model Performance on cross-evaluation

From Figure 5.22, it can be observed the best result in cross-evaluation has been achieved by the CNN model, which has been trained on perpendicular data: its accuracy on the dataset of moving away reached 0.7424, precision-0.7604, recall-0.7424, F1-0.7379. However, this is still much lesser with the performance metrics of CNN, which were around 0.93 when trained and tested on the moving away dataset.

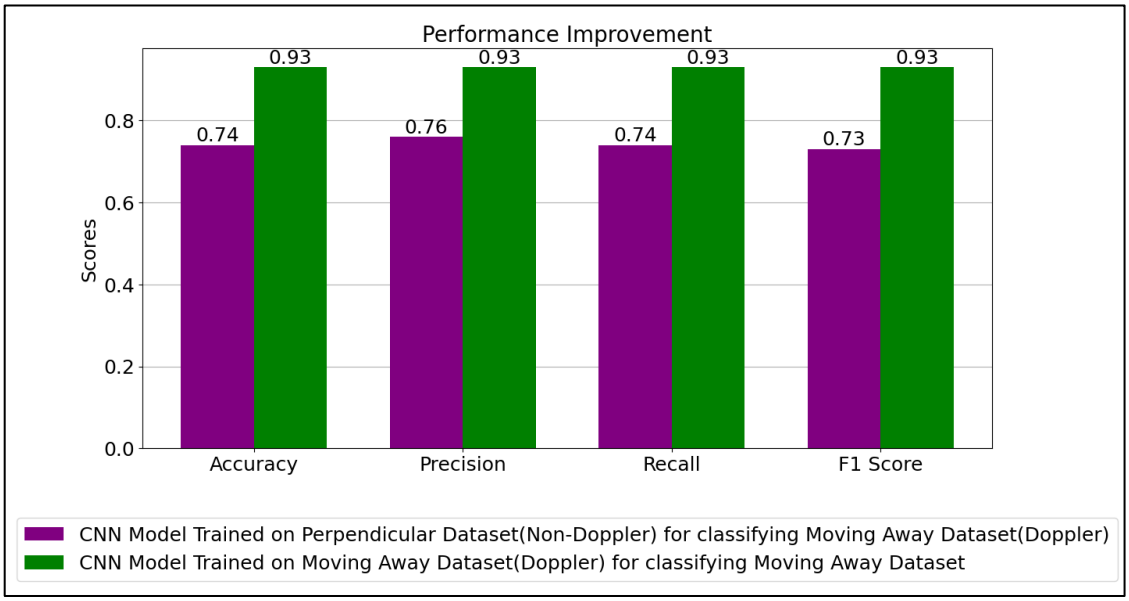


Figure 5.22: Graph comparing CNN cross-evaluation on perpendicular vs. moving Away dataset with original CNN results for moving Away dataset.

5.5. Interpretation of Results

- **Model Generalization**: The results clearly identify that the performance of both MLP and CNN is at its peak when working with the datasets the models actually got trained on. But CNN was found to be more adaptable to cross-movement testing with greater precision and F1 score.
- **Doppler Effect Challenges:** The significant degradation in performance in the cross-movement tests indicates that both models were challenged by different Doppler shifts in ultrasonic signals, signifying the importance and need for training in specific movement scenarios to improve the classification performance.
- **Benchmark Insights:** Testing on the same movement dataset brings forth the highest possibly achievable scores for both models. Indeed, when training both models on a particular dataset and testing on that very same dataset yields perfect or near-perfect scores, while cross-movements reveal enormous degradations in performance.

Chapter 6

Conclusion and Future Scope

The entire thesis is summarized in this chapter. Furthermore, recommendations, limitations and suggestions for future work are discussed as well.

6.1. Conclusion

The thesis investigated the influence of the Doppler effect on the data output of ultrasonic sensors for pedestrian classification in an autonomous driving environment. Emphasizing the variation of frequency-domain features due to the movement of pedestrians and other objects, it was able to demonstrate how Doppler shifts might influence performance under different conditions of motion. Multi-Layer Perceptron and Convolutional Neural Network models were used in this research study to investigate the performance of the classification under different conditions of training and testing.

Key Findings and Contributions

1. **Doppler Effect on Classification:** Doppler shifts certainly have an impact on classification, especially in moving scenarios like "towards" and "away." Stationary and perpendicular cases, on the contrary, faced minimal Doppler shifts; hence, different spectral handling was required.

2. **Performance of MLP vs. CNN:** Both models did extremely well on matching datasets for training and testing, with MLP achieving perfect scores in some cases. However, the CNN model gave slightly better robustness against certain cross-movements like "Moving Away". Overall, the performance significantly dropped during cross-movement evaluations.

3. **Limitations:** Due to the very short 3-meter walking path, generalization of the model was limited. Single sensor data also constrained model generalizability. More varied and broader datasets considering different patterns of movement would improve model performance generally.

6.2. Recommendations for Developing a Doppler-Responsive Classifier

Below approaches might be beneficial to construct a classifier that would operate over a wide range of vehicle speeds:

- **Speed-Adaptive Training Sets:** The collection needs to be done over a wide range of speeds with a Doppler shift that corresponds to the higher speeds of the vehicle. In fact, datasets corresponding to low-speed and high-speed scenarios will make such a classifier adaptable to real-world conditions.

- **Multi-Classifiers Approach for Different Speed Ranges:** Classifiers have to be designed, each of them devoted to a particular range of speed. For instance, a low-speed classifier shall focus on parking scenarios, whereas another class shall be devoted to a high-speed classifier for faster approaches. Dynamic switching of the classifier depending on the vehicle speed.

- **Multi-Sensor Fusion:** Integration of ultrasonic sensor data with radar or LiDAR for capturing of more detailed Doppler information; such integration would increase the accuracy by adding more dynamics of movement compared to single-sensor limitations.

6.3. Future Work

Further studies should address the limitations in this research, in addition to proposing more feasible strategies that may further enhance performance in real driving conditions using autonomous vehicles.

- **Collection of Expanded Dataset:** Gathering data on wider paths and across more diverse pedestrian behaviors will enrich the variety in the training dataset for increased capability of classification by the model.

- **Real-Time and Edge Optimization:** Deploy optimized models for low-latency real-time classification on edge devices in autonomous vehicles. Techniques include cloud and mobile computing can be employed for this purpose.

- **Adaptive Classifier Framework:** Dynamic classifiers can be selected based on speed which will allow continuous adaptation based on the range of Doppler shifts.

6.4. Final Remark

This thesis has demonstrated how Doppler effects will influence pedestrian classification within autonomous vehicle environments and also illustrated how Doppler-sensitive features may be used by machine learning

models to distinguish effectively between pedestrians and objects. However, a variety of limitations were identified, especially regarding dataset size and sensor types, which expose areas where the classifiers can be further enhanced in future work. This can be realized by expanding the datasets, exploring multi-sensor fusion, and building adaptive classifiers. Possible future work will improve the potential of Doppler-informed pedestrian classification for enhancing the safety and reliability of autonomous driving applications.

The recommendations and future work identified in this chapter hint at the need to adapt pedestrian classification systems to account for variations in movement speeds and Doppler shifts. This, in fact, will enable an autonomous vehicle to better handle a complex pedestrian interaction and, therefore, provide a safer driving environment in very diverse settings within urban environments.

References

1. Abirami , S., & Chitra, P. (2020). Chapter Fourteen - Energy-efficient edge based real-time healthcare support system. In P. Raj , & P. Evangeline (Eds.), *The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases* (Vol. 117, pp. 339-368). Elsevier. doi:https://doi.org/10.1016/bs.adcom.2019.09.007

2. Aery , M. K., & Ram, C. (2017). *A REVIEW ON MACHINE LEARNING: TRENDS AND FUTURE PROSPECTS.* Punjab, India.

3. Argoud, F., Sovierzoski, M., & Mendes de Azevedo, F. (2008). Performance Evaluation of an ANN FF Classifier of Raw EEG Data using ROC Analysis. *2008 International Conference on BioMedical Engineering and Informatics* (pp. 332-336). Sanya, China: IEEE. doi:10.1109/BMEI.2008.220

4. Arnaldi, L. H. (2017). Implementation of an AXI-compliant lock-in amplifier on the RedPitaya open source instrument. *2017 Eight Argentina Symposium and Conference on Embedded Systems* (pp. 1-6). Buenos Aires, Argentina: IEEE. doi:10.23919/SASE-CASE.2017.8115374

5. Bachtiar, Y., & Adiono, T. (2019). Convolutional Neural Network and Maxpooling Architecture on Zynq SoC FPGA. *2019 International Symposium on Electronics and Smart Devices (ISESD)* (pp. 1-5). Badung, Indonesia: IEEE. doi:10.1109/ISESD.2019.8909510

6. Brownlee, J. (2020, August 15). *What is a Confusion Matrix in Machine Learning*. Retrieved July 31, 2024, from https://machinelearningmastery.com/confusion-matrix-machine-learning/

7. Chen, V., Li, F., Ho, S., & Wechsler, H. (2006). Micro-Doppler effect in radar: phenomenon, model, and simulation study. *IEEE Transactions on Aerospace and Electronic Systems, 42*, 2-21. doi:10.1109/TAES.2006.1603402

8. Doppler, C. (1842). *Ueber das farbige Licht der Doppelsterne und einiger anderer Gestirne des Himmels.* (F. STUDNlCKA, Ed.) Prague: Verlag der Königl. Böhm. Gesellschaft der Wissenschaften. doi:https://de.wikisource.org/wiki/%C3%9Cber_das_farbige_Licht_der_Doppelsterne_und_einiger_anderer_Gestirne_des_Himmels

9. Eisele, J., Gerlach, A., Maeder, M., & Marburg, S. (2023, April). Convolutional neural network with data augmentation for object classification in automotive ultrasonic sensing. *The Journal of the Acoustical Society of America, 153*(4), 2447-2447. doi:10.1121/10.0017922

10. Evidently AI Team. (n.d.). *Accuracy vs. precision vs. recall in machine learning: what's the difference?* Retrieved August 15, 2024, from Evidently AI: https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall#:~:text=Recall%20is%20a%20metric%20that,the%20number%20of%20positive%20instances.

11. Gaikwad, N., Tiwari, V., Keskar, A., & Shivaprakash, N. (2019, February). Efficient FPGA Implementation of Multilayer Perceptron for Real-Time Human Activity Classification. *IEEE Access, PP*(99), 1-1. doi:10.1109/ACCESS.2019.2900084

12. Galvao, L., Abbod, M., Kalganova, T., & Palade, V. (2021, October). Pedestrian and Vehicle Detection in Autonomous Vehicle Perception Systems—A Review. *Sensors, 21*(21), 7267. doi:10.3390/s21217267

13. Goyal, K. (2024, February 15). *6 Types of Activation Function in Neural Networks You Need to Know*. Retrieved August 31, 2024, from UpGrad: https://www.upgrad.com/blog/types-of-activation-function-in-neural-networks/

14. Gupta , A., Anpalagan , A., Guan, L., & Khwaja, A. S. (2021, February 23). Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array, 10*, 100057. doi:https://doi.org/10.1016/j.array.2021.100057

15. Hosur, P., Basavaraj Shettar, R., & Potdar, M. (2016). *Environmental awareness around vehicle using ultrasonic sensors.* Jaipur, India: 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). doi:10.1109/ICACCI.2016.7732200

16. Hyun , E., Jin, Y., Park, J., & Yang, J. (2020, October 30). Machine Learning-Based Human Recognition Scheme Using a Doppler Radar Sensor for In-Vehicle Applications. *Sensors 2020, 20*(21, 6202). doi:https://doi.org/10.3390/s20216202

17. Leite, T. (2018, May 10). *Neural Networks, Multilayer Perceptron and the Backpropagation Algorithm*. Retrieved October 10, 2024, from Medium: https://medium.com/@tiago.tmleite/neural-networks-multilayer-perceptron-and-the-backpropagation-algorithm-a5cd5b904fde

18. LibreTexts. (2023). *The Doppler Effect*. (LibreTexts) Retrieved 07 31, 2024, from https://phys.libretexts.org/Bookshelves/University_Physics/University_Physics_(OpenStax)/Book%3A_University_Physics_I_-_Mechanics_Sound_Oscillations_and_Waves_(OpenStax)/17%3A_Sound/17.08%3A_The_Doppler_Effect

19. Menzies, T., Kocagüneli, E., Minku, L., Peters, F., & Turhan, B. (2015). Chapter 24. Using Goals in Model-Based Reasoning. In *Sharing Data and Models in Software Engineering.* doi:10.1016/B978-0-12-417295-1.00024-2

20. Mubarak, M. (2013). *Outdoor obstacle detection using ultrasonic sensors for an autonomous vehicle ensuring safe operations.* Master of science thesis, tampere university of technology, faculty council of automation, mechanical and materials engineering .

21. Narkhede, S. (2018, May 9). *Understanding Confusion Matrix*. Retrieved August 8, 2024, from Towards Data Science: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

22. Nematollahi , M. A., & Khaneghah, M. (2019, 08). Neural network prediction of friction coefficients of rosemary leaves. *Journal of Food Process Engineering*. doi:10.1111/jfpe.13211

23. Nicholson, C. (2020). *A.I. Wiki*. Retrieved October 8, 2024, from Pathmind: https://wiki.pathmind.com/convolutional-network

24. Nicholson, C. V. (n.d.). *A Beginner's Guide to Multilayer Perceptrons (MLP)*. Retrieved August 26, 2024, from Pathmind: https://wiki.pathmind.com/multilayer-perceptron

25. Oppenheim, A., & Schafer, R. (2010). *Discrete-Time Signal Processing* (3rd ed.). Prentice Hall.

26. Pech, A., Nauth, P., & Michalik, R. (2019). *A new Approach for Pedestrian Detection in Vehicles by Ultrasonic Signal Analysis.* Novi Sad, Serbia: IEEE EUROCON 2019 -18th International Conference on Smart Technologies. doi:10.1109/EUROCON.2019.8861933

27. Pongsomboon, P. (2022). *Using Image Processing for Autonomous Illumination Sensor Optimization*. Retrieved from GitHub.

28. Pranshu. (2023, August 24). *Basic Introduction to Convolutional Neural Network in Deep Learning*. Retrieved October 10, 2024, from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/

29. Proakis, J., & Manolakis, D. (1996). *Digital Signal Processing: Principles, Algorithms, and Applications* (3rd ed.). PRENTICE-HALL INTERNATIONAL, INC.

30. Puthanpurayil, T. V. (2023). *Automatic Labelling System using ML*. Retrieved 07 20, 2024, from GitHub: https://github.com/TiniyaVinod/Automatic-Labelling-System-using-ML

31. Red Pitaya. (n.d.). *Red Pitaya*. Retrieved 2024, from https://redpitaya.com/

32. Reda, A., El-Sheimy, N., & Moussa, A. (2023, December). DEEP LEARNING FOR OBJECT DETECTION USING RADAR DATA. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, X-1/W1-2023*, 657-664. doi:10.5194/isprs-annals-X-1-W1-2023-657-2023

33. Saha, S. (2018, December 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Retrieved October 12, 2024, from Towards Data Science: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

34. Shannon, C. (1948, July). A mathematical theory of communication. *The Bell System Technical Journal, 27*(3), 379-423. doi:10.1002/j.1538-7305.1948.tb01338.x

35. Song, Z., Shi, Z., Yan, X., Zhang, B., Song, S., & Tang, C. (2024). An Improved Weighted Cross-Entropy-Based Convolutional Neural Network for Auxiliary Diagnosis of Pneumonia. *Electronics 2024, 13*(15). doi:https://doi.org/10.3390/electronics13152929

36. Groeningen, T. V., Driessen, H., Söhl, J., & Vôute, R. (2018). AN ULTRASONIC SENSOR FOR HUMAN PRESENCE DETECTION TO ASSIST RESCUE WORK IN LARGE BUILDINGS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-4/W7*, 135--140. doi:10.5194/isprs-annals-IV-4-W7-135-2018

37. Windeatt, T. (2008). *Ensemble MLP classifier design* (Vol. 137). Computational Intelligence Paradigms. doi:10.1007/978-3-540-79474-5_6

38. Wood, T. (n.d.). *F-score*. Retrieved August 25, 2024, from Deepai.org: https://deepai.org/machine-learning-glossary-and-terms/f-score

39. Yeong, D., Velasco-Hernandez, G., Barry, J., & Walsh, J. (2021, March 18). Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. *Sensors 2021, 21*(6), 2140. doi:https://doi.org/10.3390/s21062140

40. Zheng , Y. (2018, September). Evaluation and Implementation of Convolutional Neural Networks in Image Recognition. *Journal of Physics: Conference Series, 1087*(6), 062018. doi:10.1088/1742-6596/1087/6/062018