



Master Thesis

A Machine Learning Approach to Infant Carrier Car Seat Occupancy Detection using Ultrasonic Sensors

Submitted by / Eingereicht durch:

First examiner / Erstprüfer:

Second examiner / Zweitprüfer:

Date of start / Startdatum:

Date of submission / Abgabedatum:

Haritha Lakshmi Gopinathan

Prof. Dr. Andreas Pech

Prof. Dr. Peter Nauth

17.02.2023

21.07.2023

Statement

I confirm that I have written this document on my own. No other sources were used except those referenced. Content which is taken literally or analogously from published or unpublished sources is identified as such. The drawings or figures of this work have been created by myself or are provided with an appropriate reference. This work has not been submitted in the same or similar form or to any other examination board.



21.07.2023,

Date, signature of the student

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.



21.07.2023,

Datum, Unterschrift des Studierenden

Content

1 Introduction	5
2 Theoretical Background	7
2.1 Current Sensor solutions	7
2.1.1 Challenges of detecting people	9
2.2 Ultrasonic Waves	10
2.3 Working principle of Ultrasonic Sensor	12
2.3.1 Factors that influence Ultrasonic Sensing	13
2.3.1.1 Transmission medium	13
2.3.1.2 Acoustic Impedance	14
2.3.1.3 Cross-Section Area of Target	15
2.3.1.4 Ambient Conditions	15
2.3.1.5 Choice of Device and Beam Pattern	15
2.4 Possible transformations on time domain data	16
2.4.1 Stochastic Signals	16
2.4.1.1 Stochastic Signal Analysis	17
2.4.2 Windowing Functions	19
2.4.3 Domain Transformations	21
2.4.4 Fast Fourier Transform	21
2.4.5 Signal Peaks Analysis	24
2.5 Deep Learning Classification	24
2.5.1 Classification Problems	24
2.5.2 The Debate : ML vs DL	24
2.5.3 Principle of DL	25
2.5.4 Classification of DL approaches	27

Content	4
2.5.4.1 Supervised Learning	27
2.5.4.2 Unsupervised Learning	27
2.5.4.3 Semi-Supervised Learning	27
3 Requirements Analysis	28
3.1 General Objectives	28
3.1.1 General structure of the system :	28
3.1.2 Previous Work :	29
3.2 Time frames	30
3.3 Restriction on Machine Learning Models	31
3.4 Real Time Classification	31
3.5 Limited Domain Knowledge	32
3.5 Order of Steps	32
4 Realisation	33
4.1 Collecting Measurement Data	34
4.1.1 Structure of the Data	34
4.1.2 Time of Flight Calculation	36
4.1.3 Data Pre-processing & Labelling	37
4.2 Feature Importance and Feature Extraction	40
4.3 Signal Transformations (Windowing, ACF, PSD)	42
4.4 Building MLP Model & Optimization	48
5 Summary and Perspectives	54
6 Acknowledgement	56
7 Abbreviations	57
8 References	60
9 Appendix	63

1 Introduction

In 2018, the European Union set itself a 50% reduction target for road deaths and serious injuries by 2030 in the EU Commission's Strategic Action Plan on Road Safety. But the progress in this regard has been slow. According to a preliminary report [1] published by the European Commission, around 20,000 people were killed in road crashes in 2022, as traffic levels recovered after the pandemic. While the automotive industry around the world and the scientific community has been looking for solutions to reduce fatalities in traffic accidents, another grim yet surprising trend of non-traffic fatalities has arisen in the last decades. These fatalities occur due to hyperthermia (heat strokes) especially among children. According to the NHTSA (National Highway Traffic Safety Administration) research in the US, between 1998-2009, 444 children left behind in cars died due to hyperthermia [2]. Another fatal scenario for infants and children in general are when the airbags deploy at full speed for rear-facing infant seats or for front-facing child seats.

Most of these problems could be solved with a reliable Occupant Classification System (OCS) or in other words, with a reliable Passenger detection system for both adults and children alike. In this light, improvised OCS methods in cars are always being researched to prevent fatalities due to human error. Once this is reliably solved, the output from this OCS can then be passed to an action layer. Actions could include seat belt warning, notifying the driver of the car of a left behind child etc. This signal could also be used to make the decision to turn the airbag deployment on/off, like when a rear-facing infant carrier seat is placed in the front seat of the car, or to adjust the airbag ejection settings, based on the weight and size of the passenger. Another use case would be the monitoring of HOV (High Occupancy Vehicle) lanes. Passenger detection has seen its importance grow also with the e-Call legislation being voted on by the European Parliament in 2015. This regulation requires all new cars to be equipped with e-Call technology, where the sensors in the vehicle can detect a serious accident and automatically an e-Call to the emergency number 112 is triggered. Optionally, the number of passengers and the status of the seat-belt can also be measured by the technology.

The current passenger detection system in automobiles is broadly handled by two categories of devices : Contact and Non-Contact devices. Contact devices include pressure sensors fitted under the seat, seat belt sensors, wearable devices like bracelets and so on. The Non-Contact devices could include cameras, Infrared sensors, Ultrasonic sensors, Radars, Thermal image sensors. Considering the legal landscape in many countries, the presence of in-vehicle cameras may be seen as an invasion of privacy and may not be acceptable by all.

Different factors influence the choice of the sensor for the in-vehicle passenger detection system. These factors include the consideration for personal privacy, legal landscape in a country, the cost of implementation, the accuracy desired, the presence of previous sensor setup in an automobile, environmental conditions and so on.

This work focuses on the scope of using ultrasonic sensors to detect the presence of an infant in a rear facing infant carrier seat. Ultrasonic sensors, as one knows, are immune to varying ambient light and are inexpensive. They perform well in harsh working environments and are not affected by poor visibility due to dust, smoke, fog etc. Further this task is not a simple object classification problem where a simple feature from the ultrasonic reflection profile could be used and a threshold set, to decide the presence of an infant. The object to be classified is more complex and therefore, the help of Machine Learning Models have been sought to help with estimating feature importance and to perform supervised model training.

With ultrasonic sensors, there raises the question of varying performance with temperature. The work also therefore tries to present an account of the relationship between the two. This way, the performance of the classification model can be reliably tested with appropriate knowledge on the measured data.

Also it is to be noted that Object Detection in-vehicle comes with its own challenges like limited space, different vibrations from the car and therefore the solution needs to be robust against these variations. This is where the real strength of Neural Networks can be employed to introduce this robustness. With ultrasonic sensors, there arises some uncertainty in the measurements when environmental conditions change. As an example, the speed of sound waves change with temperature and this will induce a change in the time of flight and therefore the object could be observed in a slightly different set of samples than expected. Training a Neural Network model with such diverse data will help improve robustness in object detection in cars in all kinds of climates.

Further, with restricted domain knowledge, it helps to use more generic Feature Extraction models to weigh the importance of some columns of data over others. Decision Trees in Machine Learning have been able to do this almost without any bias and the work also shows the application of such models to extract the important features.

This work lingers on the data preparation methods and comparison of features extracted. Also, the model training phase and fine-tuning the parameters of the model are also dealt with. Potential features for classification inspired from literature are discussed for both time domain data and frequency domain data. The work tries to identify if there is enough information in the ultrasonic reflection profile to reliably classify a baby and an empty infant carrier seat. Also the classification needed to happen in real-time and with a processing capacity that can be handled by a single board processor chip. With all these limitations in mind, effort has been taken to satisfy these criteria. The work presents the various solutions that have been tested and their respective results. Therefore it can be regarded as a summary of all the techniques that worked well and did not work well, followed by a summary of possible improvements and suggestions for further work.

2 Theoretical Background

This chapter gives a theoretical background of the current state of the passenger detection systems that are in place, listing out the advantages and disadvantages. It then goes on to present a brief overview of the working principle of ultrasonic waves and sensors. It then presents a summary of the mathematics behind Fast Fourier Transforms, Autocorrelation Function, Power Spectral Density and talks about computational complexity. Most importantly, the various features in time domain and frequency domain that can be used to distinguish two types of signals are dealt with in detail. Certain concepts of Feature extraction, estimating feature importance and modelling have also been included. Finally a small summary of the Neural Network model along with the algorithms and error functions that have been used are also presented.

2.1 Current Sensor solutions

The current passenger detection system in automobiles is broadly handled by two categories of devices : Contact and Non-Contact devices[3]. Contact devices include pressure sensors fitted under the seat, seat belt sensors, wearable devices like bracelets and so on. The Non-Contact devices include cameras, Infrared sensors, Ultrasonic sensors, Radars, Thermal image sensors and so on. A brief overview on some of these existing solutions is presented below, as taken from [13].

Car Seat pressure sensors employ a membrane type contact sensor and the contacts are evenly distributed under the stress-bearing surface of the seat. When external force is applied, a trigger is generated. The pressure sensors work only by estimating the weight and can therefore fail to distinguish a heavy object from a human. There are other simple sensors that only monitor the status of a seatbelt lock. These are termed reed/hall-sensor based latch detection. A reed switch consists of two ferromagnetic blades that are separated by a very small distance (in the range of microns). When a magnet comes toward these blades, they pull toward each other, the circuit is closed, letting electricity flow. As shown in Fig 2.1, a magnet is present at the male end of the buckle and a reed sensor is placed in the position where the buckle fits in.



Fig 2.1 Reed sensor based belt latching system -
Source [13]



Fig 2.2 Detector matrix for Thermopile sensor based
Person Detection
- Source [13]

The output from this sensor is given to a seatbelt reminder system that displays a warning sign or it could also be given to the airbag system.

There is also the possibility of using wearable medical sensors, either in the form of bracelets or as part of technical gloves or socks for the purpose of passenger detection. They have the inherent ability to not only detect the passenger but also measure heart rate, blood pressure and breathing rate of the passenger. This can be useful in scenarios where sleep deprivation of a driver needs to be estimated or in scenarios where the health status of passengers can be known to make the right medical help available quickly, like in the case of an accident. Wearable devices provide accurate information, but they may be inconvenient to wear all the time and may not be accepted by many due to privacy reasons.

Coming to the category of non-contact devices, Image recognition has also been widely used for occupancy detection. But it has to be noted that specific illumination settings are needed for cameras to function properly. Therefore, these may fail when there is insufficient light as when a car enters a tunnel or a closed parking garage. On the other extreme, very bright sunlight or unexpected reflections or stray light can cause challenging conditions for cameras to function properly.

Infrared (IR) sensors are also widely used in object detection as well as motion detection. The sensor setup usually consists of an IR Light emitting Diode (LED), acting as a transmitter, emitting radiations that are not visible to the human eye and a photodiode on the other hand receiving these radiations after being reflected off of an object. The infrared light reflected back to the receiver activates the photodiode. The change in the photodiode's resistance and the output voltage is proportional to the amount of light reflected back. Lighter coloured surfaces reflect more light than darker surfaces. This is the reason, IR sensors can also be commonly used in line-following robots. But these types of IR sensors will be of disadvantage if it does not detect darker clothes worn by humans in the car. A slightly different form of IR sensors called Thermopile IR Temperature sensors measure the heat from an object at a distance. The thermopile sensing element is the key component and it contains thermocouples connected in series that measure the temperature difference between the incoming radiation and the sensor itself. A matrix of these sensors can provide a rough heatmap of a car seat and can clearly distinguish the presence of a human, as shown in Fig 2.2. Experimental results with thermopile sensors show that there are differences in the heatmap generated when the person is clothed differently [14]. Also the results look different depending on the number of passengers seated in the car. Again, climate and external temperature variations also significantly influence the level of output voltage of the thermopile sensors. Conventional IR sensors, called Passive IR (PIR) Sensors do not include a transmitter and only detect the natural IR emitted by objects. With PIR sensors, the signals by objects or humans in motion cross the slot of receivers and this triggers the output. When there is no motion, there is no such crossing of signals happening and thus conventional PIR sensors cannot recognize stationary targets [3]. Other methods have been proposed to overcome this difficulty by keeping the sensor in a constant state of motion, to detect stationary objects. But this may not be well suited to the environment inside a car, where there is already a lot of vibration and movement.

Alternatively, time of flight (TOF) cameras could also be used for passenger detection. A modulated light signal is transmitted from these ranging cameras and the time for this light signal to get reflected back is noted on a pixel level. This time is used to determine the distance. These cameras are already used in industrial machine vision and in gaming consoles like Kinect, Xbox, etc. A sample image classified from such a TOF camera is shown in Fig 2.3. As seen from the figure, the position of each passenger and the classification is pretty accurate. But the disadvantage is the high cost of the sensor's evaluation kit.

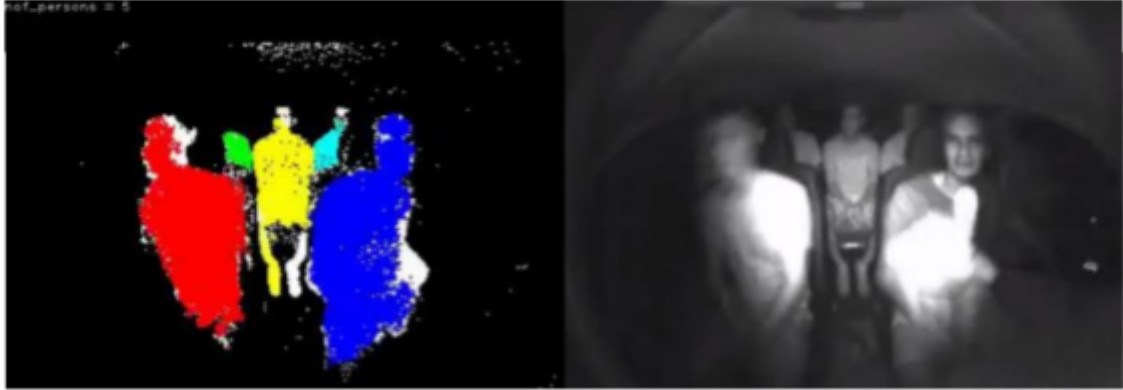


Fig 2.3 Passenger recognition and counting with TOF Camera - Source [13]

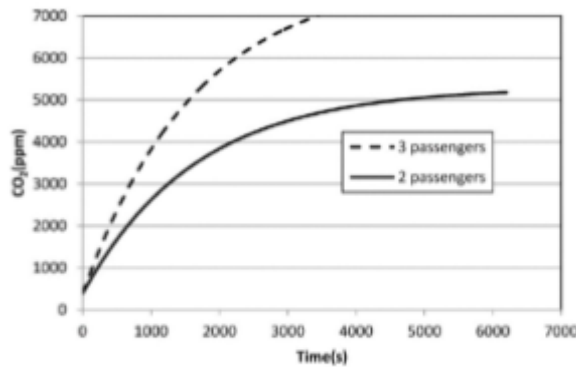


Fig 2.4 CO₂ concentration buildup inside a car with different number of occupants - Source [13]

Gas sensors are also proposed to be used in automobiles to detect passengers. In a closed chamber like a car, the concentration of gases, especially those related to human breathing, keeps changing over time. Measuring the levels of CO₂ can help identify the number of occupants. Additionally, this can also help monitor the CO₂ levels inside the car and turn on automatic ventilation from outside, if dangerous levels of CO₂ are reached. The other challenge with using a gas sensor, would be due to the fact that CO₂ levels can be very sensitive to the ventilation level, window status in the car, car speed, etc. But even with all this uncertainty CO₂ buildup rates could distinguish the number of passengers as shown in Fig 2.4., where the CO₂ levels are measured in ppm (parts per million). However it is seen in the x axis that it takes a very long time to make this classification.

2.1.1 Challenges of detecting people

The previous section showed the various possibilities to detect people inside a vehicle. The advantages and disadvantages of each have also been mentioned. But to understand these advantages and disadvantages better, the challenge of person detection inside a vehicle must first be understood. Usually with sensors in general, the surface area of the object that is to be detected matters. In case of people, this surface area is not a lot, compared to measuring the distance to a wall or the liquid in a tank. The person's clothing, size and body build are all creating extra variables to be taken care of in the classification formula. Further there are other variations caused by the motion of humans as well, particularly pose variation (sitting upright, slouching, leaning toward one side, leaning front, leaning back). All of these poses may have slightly different sensor reflection profiles and therefore training a detection model should involve all such possibilities and also leave some gap for new combinations of possibilities. Further, the internal environmental conditions also play a role in affecting the operation of the sensor, namely, temperature, illumination, humidity, dust, etc.

2.2 Ultrasonic Waves

The ultrasound waves are sound waves that are well above the human hearing range of 20 Hz to 20,000 Hz. The idea to use these waves in several fields ranging from medicine, military to control systems in industry has been tried and successfully improved over several decades. As a result of continued optimizations and demand, ultrasound devices have become cost-effective, reliable and affordable over the years. It is evident that Ultrasound systems provide one of the greatest health expenditure values when compared with other imaging systems like computed tomography and magnetic resonance imaging, as stated in a research study [9]. In the automotive space, these sensors are used in a variety of Driver-Assistance systems, like with parking assist, etc

The reason ultrasound waves are very popular in many fields, is due to the nature of testing with ultrasonic waves being non-invasive and non-destructive. Over the years, there are several low cost ultrasonic sensors that have been developed and these function reliably, are compact in size, simple to use and have a longer functional life. All these reasons have made ultrasonic sensors very popular in diverse fields.

However, there are certain additional environmental factors to be kept in mind while working with ultrasonic waves. Like any wave, the speed of ultrasonic waves depends on the medium that it travels in. To gain a better understanding of these factors, a brief overview from [10] has been represented below.

The wave equation (2.2.1),

$$v = f \cdot \lambda \quad (2.2.1)$$

relates the velocity of any wave (v) to its frequency (f) and wavelength (λ). Another way, the velocity of a wave can be arrived at, is by considering the density of the medium that it travels through and the stiffness/rigidity of the medium that it travels through. This is because, when molecules in the medium are very rigid, they experience stronger intermolecular forces and are bound together. As a result, any disturbance gets transmitted faster along the medium. And the disturbance here refers to a sound wave that usually travels in the form of compressions and rarefactions. Considering the other factor of density, as the medium is more dense, it has more molecules per unit volume and thus, experiences more inertia and has little room for changes in movement or oscillations. As a result, the disturbance (sound wave) cannot travel faster across dense mediums.

Stiffness increases the velocity of a sound wave in a medium and the density reduces the velocity of a sound wave in a medium. This is characterised by equations (2.2.2) and (2.2.3) for solids and liquids respectively.

$$v = \sqrt{\frac{Y}{\rho}} \quad (2.2.2)$$

$$v = \sqrt{\frac{B}{\rho}} \quad (2.2.3)$$

Here, v is the velocity of sound, Y is the Young's modulus, to measure the rigidity of a solid and B is the Bulk modulus of a material, to measure the rigidity of a liquid and ρ is the density of the material, which is the mass per unit volume of the material. As can be noticed, these two factors are contradictory for objects that are both stiff and dense, like steel. The influence of the stiffness of steel overtakes the influence of density of steel on the speed of sound. This is usually the case for other rigid materials. The rigidity more than compensates the effect of the density. Due to this fact, the speed of sound is highest in solids, followed by liquids and then by gases, as seen in (2.2.4).

$$v_{solid} > v_{liquid} > v_{air} \quad (2.2.4)$$

But the effect of density cannot be completely ignored. Within the same medium with the same stiffness, like air, the density sometimes changes as a result of temperature. This also causes the speed of sound to vary. This is better represented by the equation (2.2.5), as taken from [10].

$$c(t) = 331.45 \sqrt{1 + \frac{t}{273.16}} \quad (2.2.5)$$

Here, t is the temperature of the air in Celsius and $c(t)$ refers to the speed of sound as a function of the air temperature and 331.45 m/s is the speed of sound in dry air at 0°C. When air gets heated up, the density of the air decreases, as molecules move far apart and the reverse is true with cooler air. As a result, sound waves travel faster through hot air than through cold air. Applying Taylor series expansion to the equation (2.2.5), the simplified version is

$$c = 331.45 + 0.607 * t \quad (2.2.6)$$

A graph displaying the values of the velocity of sound waves with varying temperatures is shown in Fig 2.5. As can be seen from this figure, the speed of sound in air at 20°C is 343 m/s, whereas the speed of sound in air at 0°C is 331 m/s. These values are important and will be referred to by a later Section.

Now It should also be noted that relative humidity also plays a small role in affecting the speed of sound as stated in [11]. It has already been established that speed is inversely proportional to the square root of density. Humid air has more molecules of water. As the mass of water molecules is less than that of oxygen, carbon dioxide or nitrogen molecules, the density is less and therefore, the speed of sound is higher in humid air compared to dry air. Fig 2.6 represents the graph of percentage increase in the speed of sound over increasing relative humidity for various temperatures.

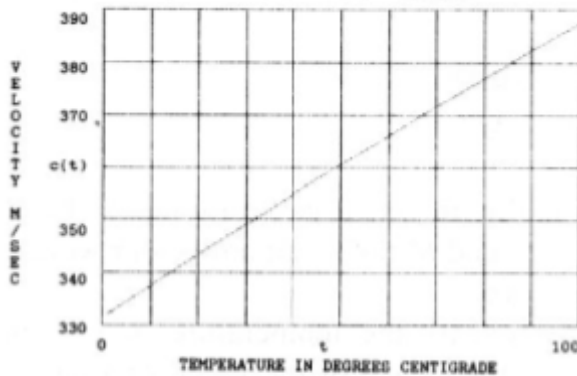


Fig 2.5 Effect of temperature on the speed of sound
- Source [10]

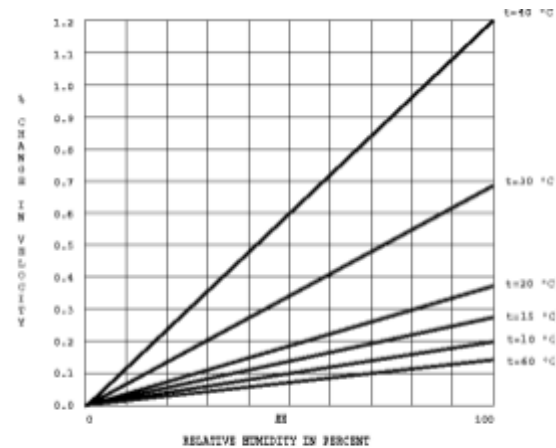


Fig 2.6 Effect of temperature on the speed of sound
- Source [10]

Frequency (kHz)	Relative humidity (%)										
	0	10	20	30	40	50	60	70	80	90	100
2	4.14	38.2	17.4	10.9	8.34	7.14	6.55	6.28	6.19	6.21	6.29
4	8.84	102	62.3	38.9	28.0	22.2	18.7	16.6	15.2	14.2	13.6
6.3	14.9	154	135	90.6	65.6	51.3	42.5	36.7	32.7	29.8	27.7
10	26.3	202	261	205	155	123	102	87.3	77.0	69.3	63.5
12.5	35.8	224	338	294	232	187	156	134	118	106	96.6
16	52.2	250	428	423	355	294	248	214	189	170	155
20	75.4	281	511	564	508	435	374	326	289	261	238

Fig 2.7 Total Sound Absorption (in db/km) for varying Frequency and Relative Humidity - Source [10]

2.3 Working principle of Ultrasonic Sensor

Now that the characteristics of ultrasonic waves have been studied, the next focus is on the working principle of an ultrasonic sensor. There are several applications for an ultrasound wave. But this work deals with distance measurement and object detection. Thus, the concept of importance in this aspect is the time-of-flight sensing. This concept is very similar to the echolocation principle used by bats to detect objects in proximity. There are ultrasonic waves emitted by a transmitter. These waves after travelling, hit an object and get reflected back to the receiver. The time taken to be transmitted and to be reflected from the object is proportional to the distance between the sensor and the object, as shown in Equation (2.3.1).

$$d_{\text{oneway}} = \frac{t_{\text{roundtrip}} \cdot v_{\text{sound}}}{2} \quad (2.3.1)$$

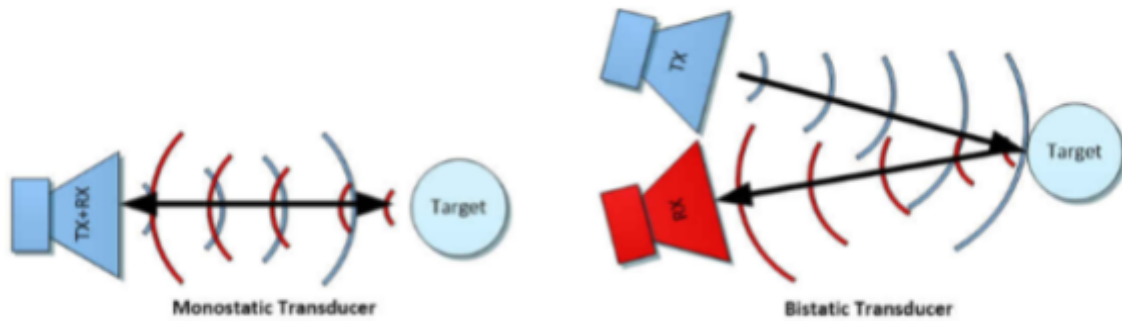


Fig 2.8 Transducer Topologies - Source [15]

There are different types of arrangements for the ultrasonic sensor receivers and transmitters. The terms monostatic topology and bistatic topology are used to indicate the measurement scenario. When a single device acts as both transmitter and receiver, it is called monostatic. When the receiver and transmitter are separate, they are termed bistatic. The topologies are depicted in Fig 2.8. Monostatic is the lower cost method preferred in most applications. The advantage is that it requires no calibration prior to using. But on the negative side, the transmitted pulse's excitation takes a while to decay and during this time, the signals cannot be received well. This portion is called the blinding zone. As a result, it reduces the minimum detection range. With the bistatic configuration, it requires prior calibration and extra complications in the measurement, like measuring the angle of echo when it is entering the receiver. On the bright side, since the receiver and transmitter are two separate devices, the receiver does not need to wait for the transmitted excitation to decay before it starts measuring echos. There is also the possibility of having one transmitter and multiple receivers or multiple transmitters and one receiver. There is also the possibility of having an array of transmitters and receivers. This all depends on the measurement scenario at hand and the expertise to accurately take measurements or perform calibration.

In this section, however, a simple yet common setup is discussed. This includes the ultrasonic piezoelectric transducer where the same device operates as a transmitter and also acts as the receiver. The piezoelectric property allows conversion of electrical signals into mechanical vibrations, necessary for transmission and the conversion of mechanical vibrations back into electrical signals, necessary for reception. Once the echos are received by the sensor, these signals are first amplified and then they are sent to an Analog to Digital Conversion (ADC) unit. Once the signals are digitised, any out-of-band noise is removed and the resulting signal looks like in Fig 4.11(a) .

Often, frequencies of 20 KHz to above 2 Mhz are used by ultrasonic range finders. The higher the frequency of the ultrasound wave, the higher is the accuracy and resolution of the ultrasonic device, but on the other hand this also limits the range that can be monitored with a high frequency device. Attenuation also increases with

increase in frequency [12]. This is because, as seen earlier from Equation (2.2.1), when the frequency is doubled, the wavelength has to be halved in order to maintain the constant v . When the wavelength is half, it means the wave makes more oscillations /movement changes over half the period of time as before. When these movements have to occur more quickly in half the space as before, the effect of density in the medium is also perceivably double. Due to these two factors, the energy lost is four times as usual, when the frequency is doubled. This is generalised in the equation (2.3.2),

$$\alpha \propto (f)^2 \quad (2.3.2)$$

Here, α is the attenuation coefficient and f refers to the frequency of the sound wave. Even though high frequency signals experience a lot of attenuation, they have their specific uses. In industrial applications, when there is a lot of background noise, which could easily involve signals in the range of 100 KHz, high frequency ultrasonic devices are the only choice. Otherwise, when a lower frequency of operation is chosen, the background noise can falsely influence the operation of the device. Also, high frequency ultrasonic devices are more useful in applications that need to have a narrow field of view. A relation between the frequency of sound waves and its absorption is already presented in the form of a table in Fig 2.7. Here absorption means the energy that is lost from the sound wave during the travel.

The operating characteristics of ultrasonic sensors have been studied. Below are a list of reasons why they are so popular :

- An object's presence, position and distance can be measured
- Objects can be detected regardless of colour, translucence, surface roughness
- Non-invasive, no contact measurement
- Easy to operate, reliable and affordable
- Robust in conditions of smoke, fog, darkness, etc

Overall, ultrasonic sensors tackle most of the challenges laid out in Section 2.1.1. However, they also suffer from a few disadvantages. The speed of sound varies with temperature and humidity as seen in the previous Section. This can cause the ultrasonic echos to be either preponed or postponed in time, than expected.. This variation is an additional complication that needs to be incorporated in the detection model.

2.3.1 Factors that influence Ultrasonic Sensing

Some factors that affect the speed of sound were already seen in Section , like temperature, humidity. But there are several other factors that influence the amount of reflection of ultrasonic waves from an object. Here is an overview of other factors that affect the operations of ultrasonic sensors in general :

2.3.1.1 Transmission medium

The speed of sound varies in different mediums. This affects the time for the sound wave to do a round trip to the sensor. For the purposes of this work, the air medium is the only one of interest. The Table I represents the speed in various media.

TABLE I. SPEED OF SOUND IN VARIOUS MEDIA -[18],[19]

State	Material	Speed (m/s)
Solids	Aluminium	6420
	Nickel	6040
	Steel	5960
	Iron	5950
	Brass	4700
	Glass	3980
Liquid	Sea Water	1531
	Distilled Water	1498
	Ethanol	1207
	Methanol	1103
Gases	Hydrogen	1284
	Helium	965
	Air	346
	Oxygen	316
	Sulphur Dioxide	213

TABLE II. ACOUSTIC IMPEDANCE OF VARIOUS MEDIA -[18],[19]

Material	Velocity of sound (m/s)	Density (kg/m ³)	Acoustic Impedance, Z (10 ⁶ kg/m ² s)
Steel	5900	7800	46.02
Water	1480	1000	1.48
Polypropylene, White	2660	0.89	2.36
Polypropylene, Profax	2740	0.88	2.40
Muscle	1590	1065	1.69
Skin	1730	1150	1.99
Air	330	1.3	0.000429

2.3.1.2 Acoustic Impedance

Acoustic Impedance (Z) is the measure of opposition of a system to the acoustical flow. It indicates how much sound pressure is generated by the vibration of molecules of a particular medium [17]. It is characterised by the product of density and acoustic velocity. The difference in acoustic impedance between two adjacent mediums is termed ‘impedance mismatch’. This is an important measure in determining the amount of energy reflected back. The more the impedance mismatch, the higher is the amount of reflection. This can be demonstrated with the help of an equation (2.3.3), as taken from [20].

$$R = \left(\frac{Z_2 - Z_1}{Z_2 + Z_1} \right)^2 \quad (2.3.3)$$

Here, R is the reflection coefficient and Z_1 , Z_2 are the impedances of the two mediums numbered in the order of travel of the sound wave through them. The Table II shows the Acoustic Impedance of Various materials. Using equation (2.3.3) to calculate the reflection coefficient in the case of ultrasonic waves reflecting off human skin (either in the case of an infant or adult human),

$$R = \left(\frac{Z_{skin} - Z_{air}}{Z_{skin} + Z_{air}} \right)^2 = \left(\frac{1.99 - 0.000429}{1.99 + 0.000429} \right)^2 = 0.99914$$

Also, the calculation of the reflection coefficient in the case of the interface between Air and Polypropylene (the material most commonly used in the manufacture of an infant carrier seat), is shown below

$$R = \left(\frac{Z_{skin} - Z_{air}}{Z_{skin} + Z_{air}} \right)^2 = \left(\frac{Z_{skin} - Z_{air}}{Z_{skin} + Z_{air}} \right)^2 = \left(\frac{2.4 - 0.000429}{2.4 + 0.000429} \right)^2 = 0.99928$$

Thus it can be seen that the reflection from an air-polypropylene interface is slightly higher than an air-skin(human) interface. This fact may be useful later for comparing the ultrasonic reflection profile of a surface like plastic versus human skin.

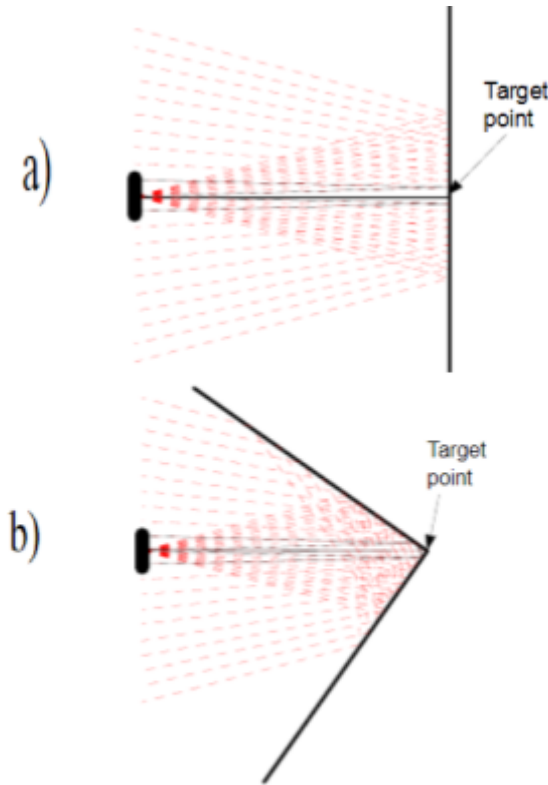


Fig 2.9 Reflection of Ultrasonic Echoes from a) Wall b) Right angle (Corner) (Solid Lines: Rays that reach the transducer, dotted lines: rays that do not reach the transducer) - Source [21]

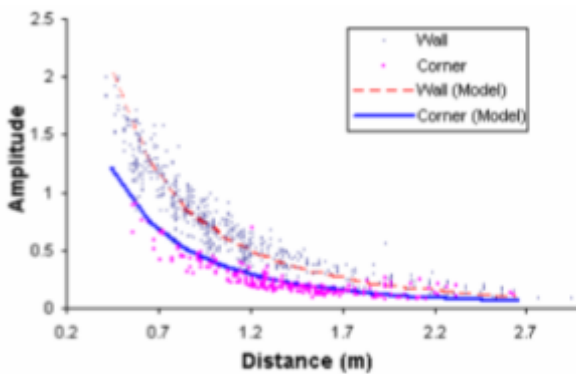


Fig 2.10 Amplitude versus Distance Plot of Ultrasonic Echoes From Walls and Corners of a Room - Source [21]

2.3.1.3 Cross-Section Area of Target

It is seen that flatter, larger, denser and smoother surfaces placed at an angle of 90° to the receiver provide better reflection of ultrasonic waves and are easier objects of detection [15]. On the contrary, objects that are more round, tilted at an angle to the receiver, have rough surfaces create strong deviations and reduce the amount of reflected energy returning to the receiver. Fig 2.9 shows the path of reflected ultrasonic waves when hitting a wall perpendicularly and when hitting a right angled corner. For the same material type, distance and incidence angle, the echoes from these two surfaces vary slightly. This is due to the fact that the echoes reflected from the wall only touch the surface once, whereas echoes from the corner are reflected twice during the flight towards the receiver. That is why the research performed in [21], states that 'a corner will produce smaller amplitude echoes than a wall of the same characteristics placed at the same distance'. This object setup was used in an experiment and the results of the experiment are shown in Fig 2.10. It shows the received ultrasonic echos from types of targets, namely walls and corners. As seen, the amplitude values for the echoes from the walls are higher than those echoes reflected off of corners.

2.3.1.4 Ambient Conditions

The factors like Temperature, Humidity have already been dealt with in detail in Section 2.2. Ultrasonic sensors are relatively robust to the presence of dust, rain, snow etc. But a very high level of dust or wind could affect the operation of the ultrasonic sensor.

2.3.1.5 Choice of Device and Beam Pattern

Most commercial ultrasonic sensors radiate sound in a conical pattern. The sensitivity and range of an ultrasonic sensor is defined with the help of a beam pattern. The

beam pattern defines the relative sensitivity of an ultrasonic transducer as a function of spatial angle. The beam pattern of an ultrasonic sensor can be classified as narrow angle or wide angle. The beam patterns can vary from being very narrow to being omnidirectional. Narrow angle patterns (10°) are useful in detecting surfaces that are relatively large and smooth. Eg : Finding the level of a still liquid in a large tank. Narrow beams can travel longer distances with less dispersion [22]. On the other hand, Broader beam patterns are very helpful in detecting objects that are uneven, have a rough surface etc. This can be better illustrated with the help of Fig 2.11. The Scenario (a) in this figure illustrates the case where the target to be detected has a smooth surface. As a result, the reflections resulting from this surface travel mostly back to the sensor without any deviations. Hence, a narrow beam sensor in this case works well. Coming to the Scenario (b), the target to be detected is a turbulent liquid. The irregularities and the tilted surface angle to the sensor, causes the reflected waves to mostly fall outside the sensor's range. The scenario (c) fixes the error in scenario (b), by using a broad beam sensor to sense the same type of uneven target. Even though the reflected beam is highly dispersed this time, since the sensor's range is broader, there is a higher chance of the reflected waves falling in the sensor's range. Therefore, choosing the right beam pattern and range sensitivity is important, according to the target to be detected.

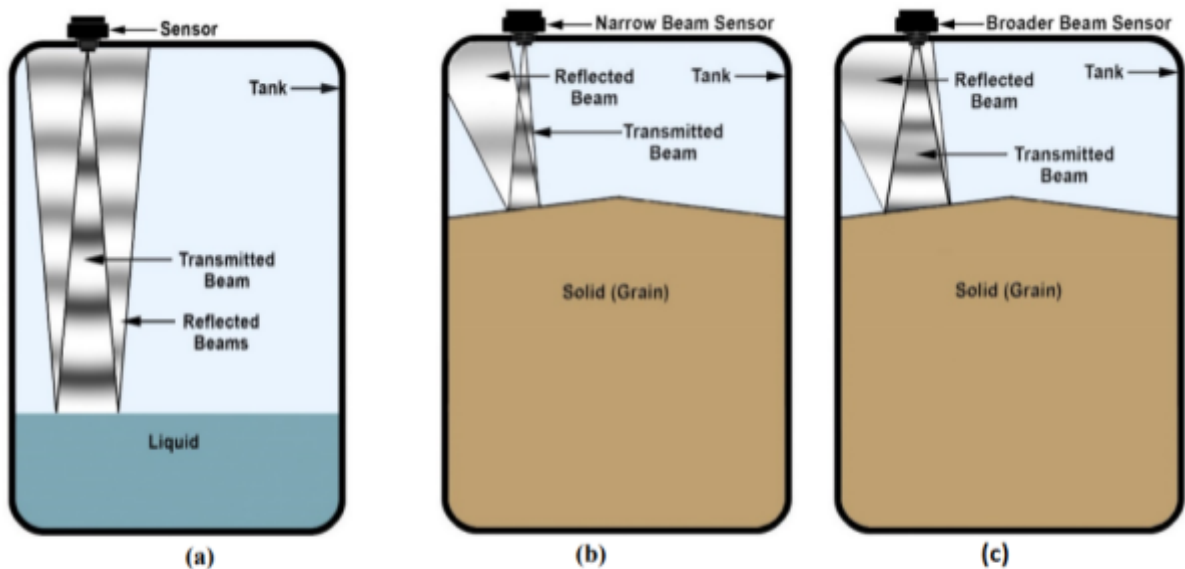


Fig 2.11 (a) Narrow Beam Pattern Ultrasonic Sensor, with a flat even target
 (b) Narrow Beam Pattern Ultrasonic Sensor, with a turbulent uneven liquid target
 (c) Wide beam Ultrasonic Sensor, with a turbulent uneven liquid target- Source [22]

2.4 Possible transformations on time domain data

2.4.1 Stochastic Signals

A signal can be random (stochastic) or deterministic in nature. Signals that can be modelled by a mathematical formula are deterministic signals and there is no uncertainty about the value these signals take at any specified point in time. Because of this lack of uncertainty, future values can also be predicted with the help of past and current values. Those signals whose behaviour cannot be captured by a formula and vary over time are random (stochastic) signals. Most of the real world signals are stochastic in nature [7]. There is no dependence of the present value on the past or future values. Examples include speech signals, the output from an ultrasonic sensor etc. These signals are therefore studied in specific intervals of time after digitising them. Analog to Digital Converters (ADC) help convert these continuous random signals to discrete signals for further analysis. An ADC is also an important component present in the ultrasonic unit, that converts the ultrasonic echo received into a discrete form.

2.4.1.1 Stochastic Signal Analysis

Since stochastic Signals cannot be expressed as a simple function of time, they can only be characterised by statistical measures or modelled in probabilistic terms. Some statistical properties to describe stochastic signals include Mean, Variance etc. Before getting into the analysis of stochastic signals, stochastic signals can further be classified as stationary and non-stationary signals. Stationary signals are signals whose properties do not change over time, whereas non-stationary signals are those whose statistical properties change over time. Example of a stationary signal would be a Gaussian white noise process, as its mean is zero and its variance is also constant throughout time. If one looked at the distribution of white noise in various sections in time with the help of a sliding window, the distribution would stay the same. On the other hand, a speech signal would be an example of a non-stationary process. The frequencies contained in the speech signal vary with time, so does the mean, variance and so on vary with time as well. Some of the common statistical measures to describe a Stochastic Signal are discussed below. Additionally, the use of each measure as a Machine Learning Feature is also discussed. Apart from statistical measures, higher level abstract properties of the signal can also be uncovered by Machine Learning. A brief overview of the statistical properties of stochastic signals are presented below, as taken from [7].

2.4.1.1.1 Mean

The mean is a measure of the central tendency of the probability distribution of a random variable. It is commonly denoted as μ . It can be calculated as per the formula, as shown in Equation (2.4.1). Here x_i represents the random variable's value at the i^{th} instant in time among a total of N samples. The values of the variable at all instants in time is summed up and then divided by the total number of samples to give an average of the variable. Sometimes, the mean also is called the expectation (E) of a signal.

$$\mu = \frac{\sum_{i=0}^N x_i}{N} \quad (2.4.1)$$

2.4.1.1.2 Variance

This is a measure of spread or dispersion around a mean value. It is often denoted by the symbol σ . The variation can be thought of as a deviation of a random variable from its expected (mean) value. It can be calculated as per the formula, as shown in Equation (2.4.2). The difference between the random variable's value at each instant and its mean is squared and summed. The total is then divided by one less than the Number of samples. The denominator in this case is N-1 and not simply, N. This is because only the

$$\sigma^2 = \frac{\sum_{i=0}^N (x_i - \mu)^2}{N-1} \quad (2.4.2)$$

In probability, population refers to the entire group one wants to draw conclusions about, whereas the sample is just a smaller subset of this population. The sample variance is more practical to calculate in most cases than the whole population. This sample variance differs from the population variance. To get an unbiased estimate of the variance from the sample data, one needs to consider the Degrees of Freedom. The degree of freedom takes into account the number of independent values that affects the estimate. As seen from Equation (2.4.2), the calculation of variance depends on the value of the mean. As a result, the Degrees of Freedom have to be reduced by 1 and become N-1.

The variance is useful as a feature of classification of signals, when two different signals have the same mean, but they have different variance, i.e. different spread of values around the central tendency (i.e. mean)

2.4.1.1.3 Correlation

Correlation tells the similarity between signals. It helps find a pattern of similarity and is one of the predominant approaches for random signal analysis. When the correlation is calculated between two different signals, it is called cross-correlation and is defined by the equation (2.4.3)

$$C_{xy}(\tau) = x \star y(\tau) = \sum_{i=-N}^N x(i)y(i + \tau) \quad (2.4.3)$$

Here, x and y are the two different digital signals and the number of elements in the signal is described by N and the time lag is represented by τ . The symbol \star here represents correlation. When the correlation of a signal with itself is calculated, it is called Auto Correlation and is defined by the equation (2.4.4)

$$C_{xx}(\tau) = x \star x(\tau) = \sum_{i=-N}^N x(i)x(i + \tau) \quad (2.4.4)$$

At lag = 0, usually the Auto correlation is maximum as the signal is most similar to an exact copy of itself. Afterwards for periodic signals, there are maximums and minimums based on the periodicity. The usefulness of Autocorrelation Function (ACF) has been demonstrated in Fig 2.12. Here two different speech signals are used and these are hard to work with in the raw format. But after taking the ACF of these, the pitch of the sound can be extracted by finding out the dominant components in the ACF and performing a few calculations, as demonstrated in detail in [23].

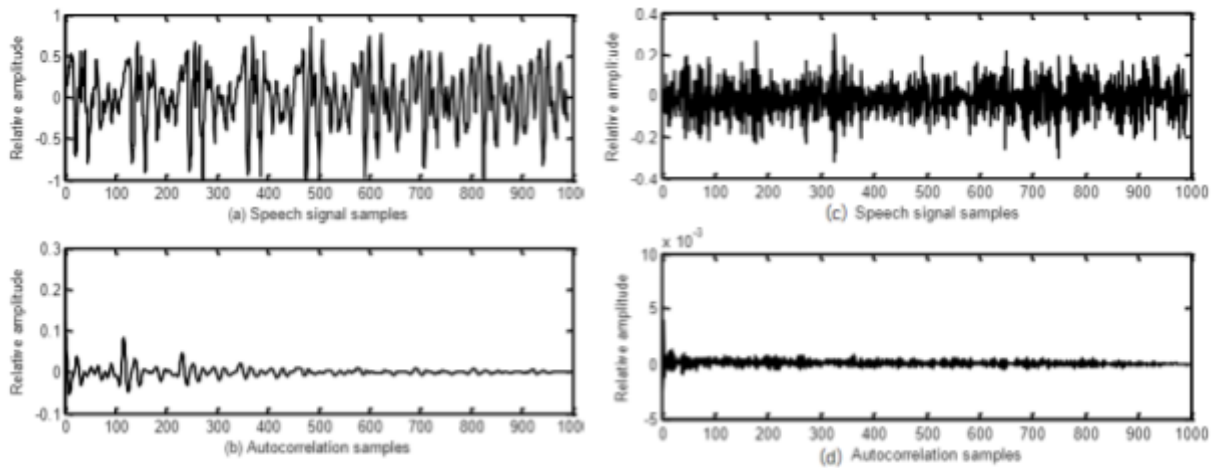


Fig 2.12 (a) Speech sample of the phonetic unit /a/ (b) ACF of the sample : a
(c) Speech sample of the phonetic unit /s/ (d) ACF of the sample : c
- Source [23]

2.4.1.1.4 Covariance

Covariance is a measure of how random variables vary together. To compare the covariances over data sets, the scale must be normalised in both as a first step. Then the covariance between 2 random variables can be measured according to Equation (2.4.5),

$$cov_{x,y} = \frac{\sum_{i=0}^N (x_i - \bar{x})(y_i - \bar{y})}{N-1} \quad (2.4.5)$$

Here, x and y are the random variables and their respective means are \bar{x} and \bar{y} . The sample size is N. The result of Equation (2.4.5), if positive, indicates that both the random variables move together, i.e. increase or decrease in both variables happen together. A Negative Covariance on the other hand implies that an increase in one

variable will cause a decrease in the other. The movement of both variables are inverse in direction in this case. If the two variables have zero covariance, then their movements are not really related to each other. So far, the variance was discussed in the context of 2 random variables.

There is also the possibility to calculate the covariance between a stochastic signal and lagged samples of the same stochastic signal. In this case, this is termed as Auto Covariance and is defined by Equation (2.4.6)

$$cov_{x,x(\tau)} = \frac{\sum_{i=0}^N (x_i - \bar{x})(x_{i+\tau} - \bar{x})}{N-1} \quad (2.4.6)$$

Here x and $x(\tau)$ represent a random variable and its lagged version, with τ denoting the amount of lag and \bar{x} denoting the mean of the random variable. This is useful in cases where the amplitude at a particular time instant can be inferred from the amplitude at another instant. The autocovariance feature is especially used in predictive modelling in Stock Market Price Estimation. The future change in price of a stock could depend on the past and previous price changes, to a varying degree based on the value of the autocovariance.

2.4.1.1.5 Coherence

Coherence is the measure of similarity between the signals in the frequency domain [7]. It is useful to determine common frequencies between signals. It is calculated according to Equation (2.4.7)

$$C_{xy}(\omega) = \frac{P_{xy}(\omega)}{P_{xx}(\omega)P_{yy}(\omega)} \quad (2.4.7)$$

Here, $C_{xy}(\omega)$ refers to the coherence, where ω refers to the frequency domain, $P_{xx}(\omega)$ and $P_{yy}(\omega)$ refer to the power spectrum of signals x and y . Additionally, $P_{xy}(\omega)$ refers to the cross power spectrum of both signals x and y .

2.4.1.1.6 Power Spectral Density

The power spectral density (PSD) describes how the power of a time domain signal is spread across the different frequency bands. For a stochastic signal, the power spectral density can be calculated as per Equation (2.4.8),

$$P_n(\omega) = \sum_{k=0}^{N-1} (C_{xx})_k \cdot e^{-j2\pi kn/N} \quad (2.4.8)$$

Here, the symbol ' ω ' denotes the frequency domain. $P(\omega)$ represents the power spectral density. The C_{xx} represents the Autocorrelation of the random variable x with ' N ' number of samples, as seen earlier in Equation (2.4.4). This approach to calculate the Fourier Transform of the ACF of a random process is termed Wiener-Khintchine theorem [24]. This approach is followed instead of transforming every single pattern function of a random process. The corollary of this theorem is the fact that the Autocorrelation function can be obtained by the inverse Fourier Transform of its Power Spectral Density.

2.4.2 Windowing Functions

Windowing is a very important signal processing technique. It is usually applied to the time domain signals before taking Fourier transforms to reduce spectral leakage. It does this by minimising the effect of the samples located at the very beginning and at the very end of a time domain sequence. Without it, the abrupt change from zero to one at the boundary of a signal in the time domain causes sidelobes to occur in the frequency domain. To reduce the amplitudes of the sidelobes in the frequency domain, these discontinuities in the time domain need to be smoothed. The smoothing effect is provided by various windows. Some popular window functions include Hamming, Hanning, etc. The effect of windowing on sidelobes in the frequency domain is depicted in Fig .

Here, the comparison is between the DFT of a signal that has been windowed and the DFT of a signal that has not been windowed. The not windowed case is termed as a rectangular window response in the figure. This is because, theoretically to compare any window, one uses the rectangular window as a benchmark. The rectangular window has the sense of having a series of ones for the signal duration and the other samples are just zeros. The other window in comparison here is a Hanning window, this has the shape of a bell curve, touching zero at both ends. A look at the shape of these windows can be found in [25].

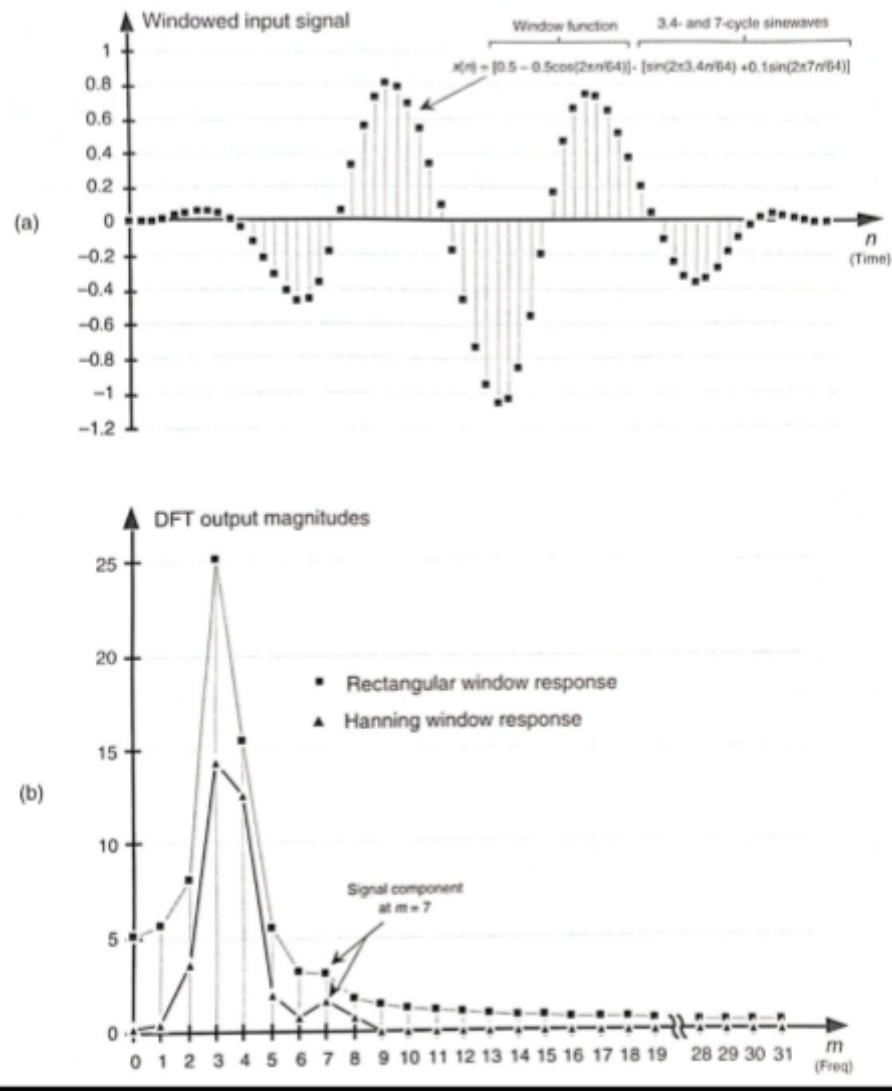


Fig 2.13 Increased signal detection sensitivity (a) 64 sample product of Hanning window and Input wave (b) DFT response of Input signal windowed with Hanning vs Rectangular windows - Source [25]

The input wave in Fig (2.13 a) is a combination of 2 sine waves, one with 3.4 Hz frequency and the other with 7 Hz frequency. As a result, the frequency spectra should show two peaks at these two frequencies. Looking at the Fig (2.13 b), the rectangular window response has a taller main lobe at 3.4 Hz, but the side lobe at 7 Hz is too

low, almost indistinguishable as a peak. This is because there is a larger leakage of spectral energy in this case, all the way up to 30 Hz. On the other hand, the Hanning window response has a relatively smaller main lobe at 3.4 Hz, but distinguishable enough to make it out as a peak at 3.4 Hz and it has a much more distinct side lobe peak at 7 Hz. This level of spectral resolution is achievable as the leakage of spectral energy is reduced and can be seen that it has reached an amplitude of zero already around 9 Hz.

2.4.3 Domain Transformations

Signals can be analysed either in the time domain or frequency domain. The domain is chosen based on the characteristics of the signal and according to ease of computation. Different transformations are available for the conversion of a signal from the time domain to the frequency domain. A commonly used transform is the Fourier Transform which helps convert a signal in the time domain to its respective frequency domain representation.

2.4.4 Fast Fourier Transform

Not all signals make a lot of sense when being visualised in the time domain. A clear example of this would be an audio signal. In the time domain representation, the different tones in speech, that represent different frequencies of sine waves, are all added up together and one confusing looking wave results, as shown in Fig . But converting this same confusing wave into the frequency domain shows clear peaks in certain frequencies and very low to zero amplitude in the other frequency bands. From knowing the frequencies present in this signal, the original sound can be recreated. Frequency domain representation is also indispensable for removing noise from an audio signal, for performing filtering operations, for image processing, communication systems and many more applications. Thus conversion from time to frequency domain and vice versa is very crucial to perform certain basic tasks.

Converting a continuous waveform from time domain to frequency domain can be done by the Fourier Transform according to the equation (2.4.9), as taken from [5]

$$X(f) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-j\omega t} dt \quad , \omega = 2\pi f \quad (2.4.9)$$

Here, $x(t)$ represents the time domain signal, x as a function of time, t and ω represents the angular frequency. $X(f)$ represents the same function in the frequency domain. This formula is effective on infinitely extending signals. The concept behind this transformation is the idea that no matter how confusing and complex a signal may be in the time domain, it can be decomposed into a sum of simpler sine and cosine waves, as found by Joseph Fourier. In practical scenarios, there are only limited lengths of signals available and therefore Discrete Fourier Transform (DFT) computes the frequency domain representation as a sequence of samples, as given by Equation (2.4.10)

$$\bar{X}_k = \sum_{m=0}^{N-1} x_m e^{-j2\pi km/N} \quad , \text{ for } 0 \leq k \leq N-1 \quad (2.4.10)$$

Here, x_m represents a particular sample from the consecutive samples of the time domain, $x(mT)$ and the sequence \bar{X}_k represents the sequence of samples $\bar{X}(kf)$ in the frequency domain. N represents the length of the sequence. This sequence is an approximation of the integral form in the previous equation. However a DFT requires $(N - 1)^2$ complex multiplications and $N(N-1)$ complex additions [5], where N refers to the number of samples. Overall, the DFT requires N^2 complex operations to be computed. For a large N , the computational load increases enormously. This is the reason the Fast Fourier Transform (FFT) was developed. It is a computationally fast and more efficient way of getting the result. Several algorithms have been written over the decade to improve upon existing ones, starting right from the Gaussian algorithm to the Cooley-Tukey algorithm in the last century to algorithms that are conceived to be used on Quantum computers in the future, called Quantum FFT. But the core idea in all of these remains the same.

The FFT takes a signal and decomposes it into N time domain signals. FFT exploits the symmetries present, takes advantage of the periodicity with sine and provides a computationally fast and efficient way to compute the DFT. The Radix-2 FFT is one of the popular algorithms, whose flow graph is depicted in Fig 2.15. Here, the signal $x[n]$ in time domain is processed in various branches to produce the frequency domain output $X[k]$. The default scaling of a branch is unity, wherever it is not mentioned. Otherwise the scaling factor is represented by W_N^x where these W coefficients are calculated accordingly. The detailed table of coefficients and the working of the radix-2 FFT algorithm can be read from [26]. FFT only requires $N \cdot \log N$ operations to be computed in total. This proves to be really efficient as N grows larger. The gravity of computational load could be better demonstrated with an example. A simple audio signal for a duration of 10 seconds sampled at 44 kHz will have the sample size as $N = 4.4 \cdot 10^5$. When the DFT is used, this would lead to roughly 10^{11} (100 billion) computations, whereas the FFT would only require roughly 10^5 (roughly 1 million) computations. The graph comparing the computational complexities of FFT (red curve) versus the DFT (blue curve) can be seen in Fig 2.14. This figure also futuristically shows how much the complexity will be reduced upon using the Quantum FFT in quantum computers (yellow curve).

The FFT algorithm used by the Python numpy module is a modified radix-2 Cooley-Tukey FFT [6] and can also take in array sizes that are not exactly a power of 2. A few things to be noted while using the FFT :

- The sample size needs to be a power of 2 to make use of the techniques mentioned earlier. Modern algorithms overcome this limitation by zero padding and other techniques. Overall, the performance is best, when the sample size is a power of 2.
- The windowing functions can be made use of to reduce the spectral leakage, as these input time domain signals are limited in length. Windowing reduces the amplitude of the discontinuities at the boundary and this helps reduce spectral leakage.
- The Nyquist sampling rate needs to be adhered to for the input ADC signal. Otherwise, the effect of Aliasing can be felt and this can introduce unexpected, false frequencies during the analysis.

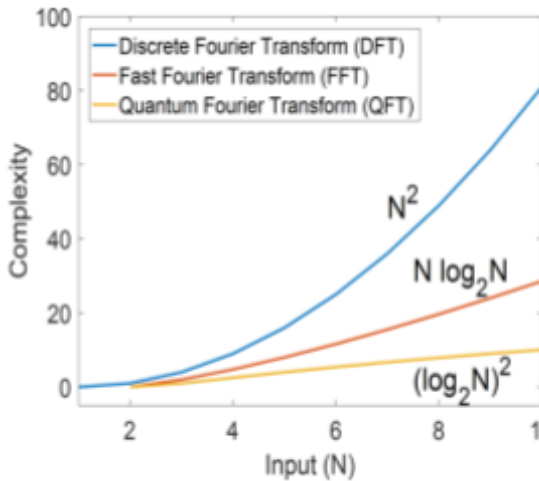


Fig 2.14 Computational Complexity between DFT and FFT - Source [8]

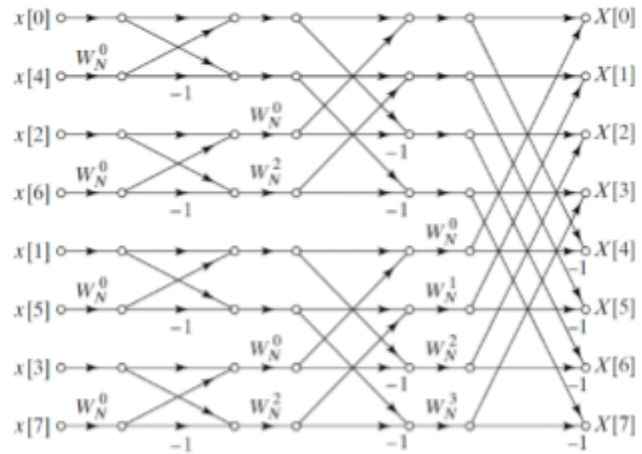
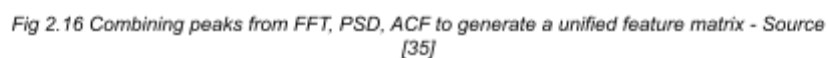


Fig 2.15 Flow graph for calculating the 8-point FFT of a signal using Radix-2 algorithm - Source [26]

When a signal is converted from the time domain to the frequency domain, the axis also changes from time (in seconds) to Frequency (in Hertz). But the scaling differs. This section focuses on the equivalent bin widths for various time signals. The bin width can be thought of as the frequency resolution of the FFT. The bin width (df) can be calculated by dividing the sampling rate (F_s) by the total length of the sample (N), as shown in Equation (2.4.11).

The bin width can also be calculated in other ways, involving the duration of the signal in the time domain. But this formula shown above would be the method followed in later Sections of this work. When music is recorded, a very common format is to use a sampling rate of 44.1 KHz (F_s). With this sampling rate, if the number of samples recorded were 4096 (N), then the bin width (df) according to Equation (2.4.11) would be 10.76 Hz. Due to this relation, the desired bin width can be obtained either by choosing a desired sampling rate or taking the desired number of samples. It is to be also noted that a larger bin width results in a lower frequency resolution and vice versa.

The other thing to be noted is with respect to the symmetric nature of the Fourier Transform. The Equation (2.4.9) for Fourier Transform extends from $(-\infty, +\infty)$. This means that the actual Fourier Transform also contains negative frequencies. But negative frequencies do not have any physical sense and these can be ignored for the purposes of this work. For this purpose, the latter half of the samples after calculating FFT are ignored and the spectrum extends from 0 to $F_s/2$.



2.4.5 Signal Peaks Analysis

Instead of just sticking to one of the transformations or statistical features from above, a combination of the above can also be used. This was proposed in the work [35]. Here the peaks of the signal from FFT, ACF and PSD can also be combined into one big feature matrix. An example of how this feature matrix would look is shown in Fig 2.16. Here, the peaks from the FFT are named the ' f_x^1 ', and the peaks from PSD are denoted ' p_x^1 ', and the peaks from ACF are named ' t_x^1 '. The x and y coordinates of all the peaks are recorded as features. By combining, no time domain specific nor frequency domain specific information would be lost.

2.5 Deep Learning Classification

The signals' characteristics and ways to transform it have been seen so far. The next stage is to feed the signals directly or the modified information obtained from these signals to a classifier. Thus the next few sections deal with types of Classification Problems and the domain of Deep Learning classifiers.

2.5.1 Classification Problems

Among the domains of deep learning or machine learning, the classification problems are usually categorised into binary, multi-class and multi-label. The binary classification is the simplest out of these. The target can be only one of two possible classes. Eg : In the case of fraud detection with credit cards, the target can be either classified as a fraud or not a fraud. Thus the binary classification problems usually deal with one class representing the 'normal' state and the other represents the 'abnormal' state. Some of the common metrics to measure the performance of a binary classifier include Accuracy, F1 score, Log-loss ratio, AUC-ROC Score.

Multi-class classification refers to situations where the target object could belong to one of N possible classes. A good example of this would be face recognition. The target could be classified as person A, person B or ... person N. The other type of classification is the Multi-Label Classification, where the target object could belong to more than one type of classification labels. This is very common in predicting the genre of a movie. A movie could belong to both 'History' genre and 'fiction' genre at the same time.

2.5.2 The Debate : ML vs DL

This work focuses on the Binary Classification Task and there are several classifiers to handle this task. Broadly, one can attempt to use simpler Machine Learning (ML) algorithms to solve it, or resort to more complex Deep Learning classifiers to handle the task. Machine Learning algorithms would attempt to solve the task by making use of a sigmoid function or logistic regression or even a simpler linear regression to try and solve it. It is just the perfect combination of computation and statistics. This approach may work with small and medium-sized datasets where the features influencing the result are well understood and documented. For example, the case of predicting a credit score for a person is a good example of Machine Learning where an algorithm like logistic regression can be employed. This is because regulators in the Finance industry often ask for the reason behind a credit score decision and with the help of an algorithm like Logistic Regression, it is easily explainable to the regulators and there are no unexpected or unclear decisions made. But this approach leaves one with the disadvantage of identifying features and assigning the importance to these features. This is called the process of Feature engineering. This becomes a huge burden as the dataset gets more and more complex and large. Another challenging scenario would be when the classification objects have features that do not exactly fall into a sigmoid shape or some other statistical function. A clear example of such a challenging scenario would be image recognition (cat or dog?)

To overcome the challenges mentioned above, Deep Learning (DL) comes into play. It is a specialised subset of Machine Learning and it attempts to emulate a neural network like in a human brain. This artificial neural network can consist of multiple layers and is made up of a series of connected nodes. Each node forms the foundational unit of a neural network. As the information is processed through the series of layers, higher levels of patterns can be identified in data. The DL algorithm improves its accuracy through the processes of backpropagation and gradient descent. These are discussed in detail in the later sections.

A comparison can be made between the two techniques to solve the same Image recognition problem (cat or dog?). The order of various steps to be performed is shown in Fig 2.17. Here, the top flowgraph shows the steps required to be performed when a Machine Learning algorithm is to be involved in the task. The input data needs to be preprocessed first. In case of image recognition with multiple objects in the same image, this may involve even adding a bounding box to a single object first. Then the features from this bounded box need to be extracted manually. Care needs to be taken in this step, as biased feature extraction may lead to incorrect discrimination between classes [29]. Then only the important features are selected and the others discarded. The flow graph below shows the steps involved in a Deep Learning model. Here, unlike the conventional ML methods, the steps of feature sets learning are automated and combined into the same step as the model training step.

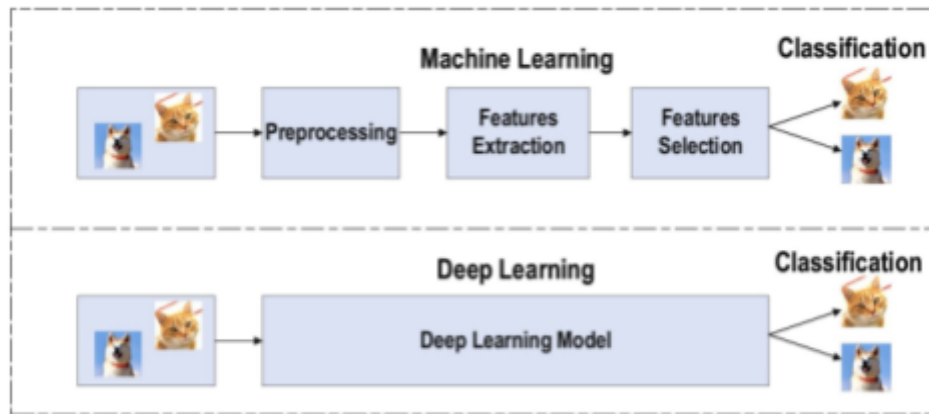


Fig 2.17 Process Flow of Steps in Machine Learning and Deep Learning - Source [29]

Deep Learning has emerged in significance in the last decade due to the tremendous increase in computing power and emergence of big data sources and data sharing technology. Since they eliminate the need for feature extraction, they require less human intervention and are more robust in scaling to large complex data sets [28]. Also they serve as a perfect solution where there is lack of domain expertise, as it eliminates the need for hard core feature extraction. Also, DL models have better performance as the amount of data used to train the model increases. This is in contrast to older statistical ML models, where after a certain increase in the amount of data, the performance of the model plateaus and there is no further improvement in performance. So, DL models are suited to situations where there is a vast amount of data. Thus DL networks are universal, more generalised, robust and are highly scalable [29].

2.5.3 Principle of DL

As mentioned before, DL neural networks consist of multiple layers of interconnected nodes, as shown in Fig 2.18 (b). The structure of a Multi-Layer Perceptron (MLP) model, a type of DL neural network, is discussed here. Each node is called a neuron, or sometimes even termed a perceptron, based on the exact DL model. This neuron, as shown in Fig 2.18 (a) forms the smallest computational unit of a DL network. The working of a neuron can be explained with the help of an equation (2.5.1), as taken from [28].

$$y = f\left(\sum_{i=1}^D w_i x_i + b\right) \quad (2.5.1)$$

Here, D is the dimension of input space that represents the various features in a dataset. x represents the input vector and w is another vector that represents the weights corresponding to the input vector. This can be understood as the importance or weightage each input node gets or the importance each branch in the neural network gets in the higher layers. The term b refers to the bias and f refers to the activation function used. Looking at the structure of an individual neuron and at the equation (2.5.1), the output resulting from this individual neuron has $D+1$ tunable parameters (i.e. D weights and one bias) that produce a case of linear regression. One single neuron by itself does not do much to solve the challenges faced by simpler ML algorithms. But when a series of neurons are interconnected in layers as shown in Fig 2.18 (b), there are much more tunable parameters and therefore can model more complex problems. The first layer of neurons after the inputs, is also called the ‘first hidden layer’ and has $D \cdot n_1$ weights and n_1 biases, where n_1 is the number of neurons in the first hidden layer. Progressing further, the second hidden layer would have $n_1 \cdot n_2$ weights and n_2 biases. This pattern follows on till the output layer. The total number of layers and the number of neurons in each layer are ‘hyperparameters’ that can be specified by the user. Further the activation function (f) is also chosen by the user.

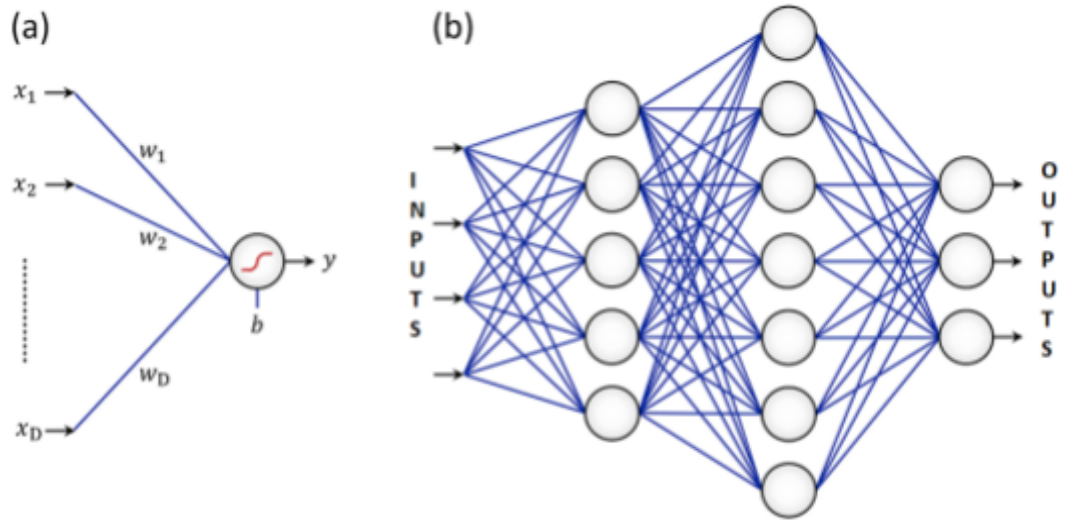


Fig 2.18 (a) Individual perceptron structure (b) MLP network with 2 hidden layers - Source [28]

This flow of information, sequentially from one layer to the next, starting from the input layer is called a Feed-Forward structure. The reason this is called a feed forward structure is that information and factors influencing decisions only flow in one way. As seen earlier in Fig 2.18, the value of all neurons except the ones at the input layer are affected by the weights, biases according to Equation (2.5.1). But nothing affects the input layer neurons. This curiosity led to the idea of other types of DL neural networks.

What caused new additions to the DL network architecture was the concept of ‘back-propagation’ (BP) algorithms proposed in 1986 [28]. This is an optimization algorithm that is aimed at reducing a loss function. This loss function is calculated at the output layer of a neural network and is defined as the sum of squares of the difference between the expected output and the actual output, as described in Equation (2.5.2), taken from [28].

$$F = \sum_{k=1}^M \sum_{j=1}^{n_d} (T_j^k - y_j^k)^2 \quad (2.5.2)$$

Here, F represents the loss function that is to be minimised. y_j^k is the actual output of the j^{th} neuron in the k^{th} layer. T_j^k is the expected (desired) output of the same previously mentioned neuron. M is the size of the training data and n_d is the number of neurons in the output layer.

The loss is calculated and now this loss needs to be fed back into the network, such that the network parameters (weights and biases) can be adjusted accordingly. This process is referred to as one iteration of the optimization process. Once the new weights and biases are found, another iteration runs through the network. This time the loss is even further minimised and the cycle goes on. Each iteration is termed an 'epoch', in ANNs. The algorithm used to find the new weights is an optimizer that tries to reduce the loss function, seen in Equation (2.5.2). This optimizer can be a vanilla Gradient Descent algorithm or a more popular Adam optimizer (improved Gradient Descent) and so on.

2.5.4 Classification of DL approaches

The previous section talked about the various network architectures. But there are other approaches to DL with regards to the type of data used and the training performed. The Deep Learning approaches are mostly classified as Supervised Learning, Semi-supervised and Unsupervised Learning. This classification is common and is with regards to the availability of labeled data. Labeled data refers to the raw data (images, text) that comes with a tag that provides an informative context. An example of Labeled data would be the MNIST dataset that contains over 70,000 images of handwritten numerals, each labeled with the number being represented.

2.5.4.1 Supervised Learning

This relies on the usage of Labeled data. Input data (x) is fed through a neural network, the output ($y = f(x)$) is then compared to the labels present in the data and it tries to keep error function (F) to a minimum. The model then tries to classify unseen data and tries to better learn the relationship between the input and the output. Overall this technique is simpler compared to the others, providing a high performance. But the real issue arises when data is falsely classified or when labeled data is not available. This technique is more commonly seen in tasks of predictive nature.

2.5.4.2 Unsupervised Learning

With no labeled data to begin with, unsupervised learning tries to learn the inherent patterns and features within a dataset. It tries to learn the relationships within the input data. Similarities are grouped together. Dimensionality reduction and clustering are common examples of unsupervised learning [29]. This technique is more commonly seen in tasks that are exploratory in nature.

2.5.4.3 Semi-Supervised Learning

When only a limited amount of labeled data is available, a combination of the two techniques mentioned above can be used. It uses a small amount of labeled data and a lot of data that does not have labels. The model is initially trained on the small subset of data that has labels. Then this model is used to predict labels for the rest of the data. Not all of these labels may be correct, as the training dataset was too small. As a result, only the labels that have a very high confidence score are retained and the other datasets are pushed back into the dataset with no labels. Now the model is again trained, but with an expanded dataset (original labeled datasets + newly found highly confident labels). As a result, a more improved model results and the iteration goes on. This process may not be successful with all datasets, but have been useful in certain situations.

3 Requirements Analysis

3.1 General Objectives

This project mainly seeks to detect the presence of an infant in a rear-facing infant carrier seat fitted to the front passenger seat in a car. The detection is done by distinguishing ultrasonic signals reflected off of the infant carrier seat and received by an ultrasonic sensor fitted to the front dashboard. This classification of ultrasonic echoes is done with the help of Machine Learning, especially Deep Learning algorithms. The scope of this work was to mainly extract features from the data that could be reliably used for classification and also developing a ML model for an optimum classification result. The factor of importance for this classification to happen is the time necessary for processing. This classification is envisaged to happen in real-time.

3.1.1 General structure of the system :

The overall experimental setup is inside a Ford Fiesta v16 car parked in the basement garage at Frankfurt University of Applied Sciences, as shown in Fig 3.2. The ultrasonic sensor (SRF 02) is mounted on the dashboard. The infant carrier seat is fitted onto the passenger seat in the front, with its backside facing the sensor on the dashboard. There is a slight angle of inclination created for the infant carrier seat when fitted to the passenger front seat, to model real-world scenarios where there is usually an infant carrier base that is fitted in most modern car seats. This base (light grey in colour) slopes on one side to create the inclination as shown in Fig 3.1. The base is manufactured this way to keep the baby seat (blue in colour) in a semi-reclined position and this is important to keep the airway open, according to the NHTSA. As seen in Fig 3.1 , the height of the base of the infant carrier seat at position A is greater than the base's height at position B. This height difference creates the inclination necessary to fit the infant-carrier seat according to generally accepted norms. Therefore this position has been recreated in the experimental setup by tilting the infant carrier seat a little to the back and is then fixed to the passenger seat. This is a procedure performed in older cars where a base cannot be attached.



Fig 3.1 Infant carrier seat attached with inclined base [31]



Fig 3.2 The car hosting the experimental setup, at Frankfurt University of Applied Sciences

Now that the setup of the infant carrier seat has been discussed, the next step is to introduce the sensors and processors involved. The ultrasonic sensor SRF02 was used in the setup. It is a single transducer ultrasonic rangefinder in a small footprint PCB [30]. It requires no user calibration and operates at a mean frequency of 40 KHz and produces a power output of 150 mW. It can measure distances from a minimum of 15 cm (6 inches) and can function with a 5V grounded power supply.

The next component involves an embedded Red Pitaya single board processor. The Red Pitaya uses an FPGA as a controller and can do a variety of measurements, thereby effectively replacing an oscilloscope, spectrum analyzer, multimeter, signal generator and so on. The RedPitaya acts as the control interface for the Ultrasonic Sensor and also performs Analog-to-Digital Conversion (ADC) of the received Ultrasonic echo pulse. It also supports LAN and WiFi connections, thereby offering the possibility to transfer this collected data to another Laptop/System for further processing.

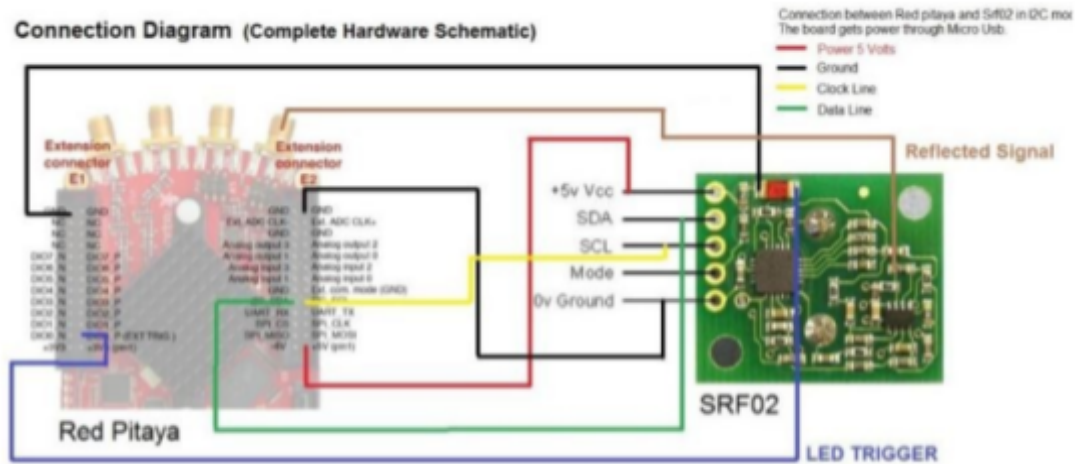


Fig 3.3 Connection between Red Pitaya and Ultrasonic Sensor SRF02
- Source [33]

3.1.2 Previous Work :

In the previous stages of this experiment, the connection between the Red Pitaya and the Ultrasonic sensor was established as shown in Fig 3.3 and can be seen in detail in [33]. When measurement needs to be initiated, then the Red Pitaya sends the appropriate control data via the SDA and SCL of the I2C interface to the ultrasonic sensor SRF02. This triggers the emission of an ultrasonic pulse. The Red Pitaya then receives the reflected Analog ultrasonic echo and samples the echo at a rate of 1.95 MHz and with a resolution of 14 bits, as referenced from one of the previous works in [32].

Apart from the hardware design seen already, a UDP client software has been developed that helps communicate from a Laptop/PC with the Red Pitaya over WiFi, over port 6000. This is made possible even without internet connection capabilities or an exclusive offline router's presence. A wifi access point is enabled on the Red Pitaya for the purposes of enabling this communication. Connecting to this wifi access point is possible after entering the password configured for the access point, as in Appendix 8.5. The GUI of the client software developed is shown in Fig 3.4. It shows the status of the sensor connected to it (box 1), with a green colour indicating successful connection and a red colour indicating an unsuccessful or non-existent connection. It also provides options of viewing the ultrasonic reflected signals plot (box 2) in time ('start ADC' button - box 3) or frequency domain ('start FFT button - box 4) instantaneously. Most importantly, it lets one save the reflected signals either in time or frequency domain in the local Laptop/PC for further processing, with the help of the 'save' checkbox

(box 5). The data can be stored either with the header information or without it, by enabling the 'save header infos' checkbox (box 6) and below which the number of desired measurements can also be specified. The data is saved in a text file, with a TSV file format. The structure of the data is discussed in detail in Section 4.1.1. The name of the file to be saved can also be specified in the latest version of this UDP Client software. In the following sections, the Time domain data is more commonly referred to as 'ADC Data' and the representation of the same data in the frequency domain is referred to as 'FFT Data'. This nomenclature is followed to have a synchronisation with the terminology used in the software, as seen already in Fig 3.4.

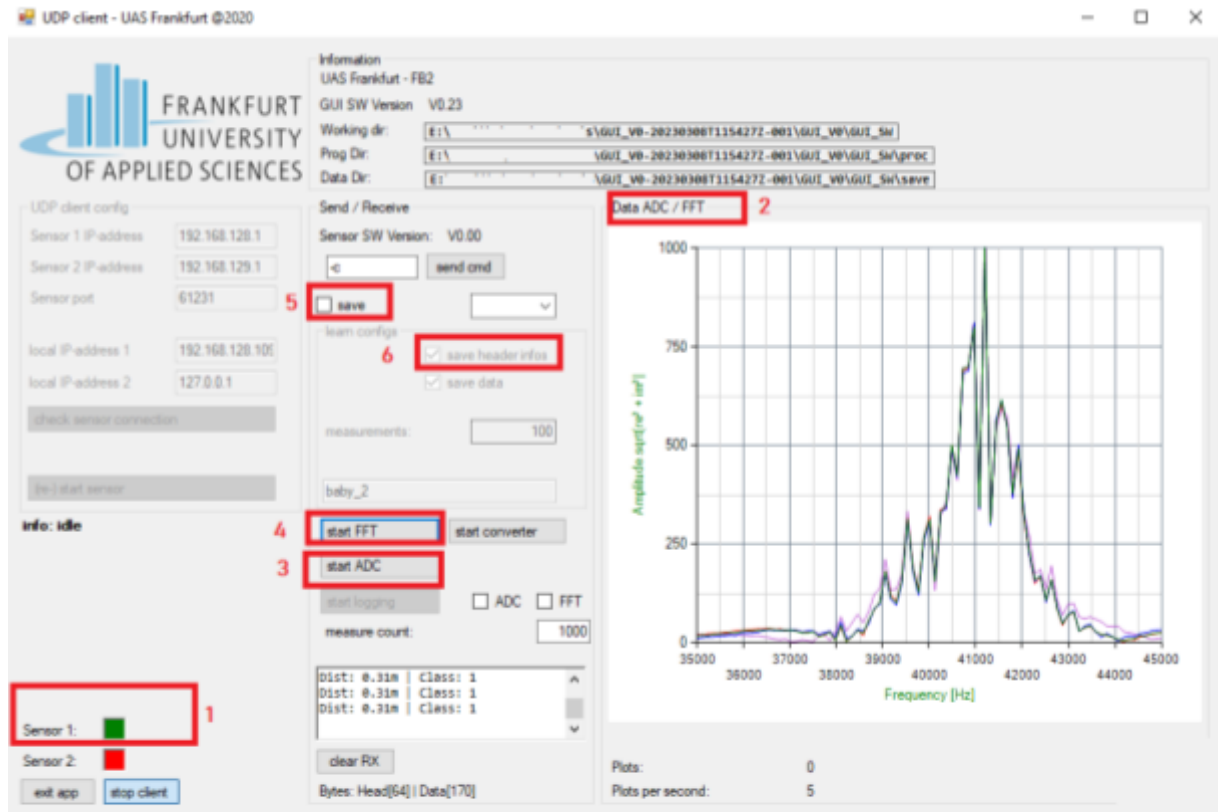


Fig 3.4 GUI of the UDP Client Software displaying a FFT plot

3.2 Time frames

In this section, the time frames for the various segments of the projects are presented.

17.02.2023 to 03.03.2023:	Literature review of current sensor solutions
03.03.2023 to 17.03.2023:	Testing of various external setups (outside the car) with the FIUS kit and understanding the object distance's relationship with the ADC peaks.
07.03.2023:	Initial meeting to clarify the requirements (on using the dashboard lower Sensor, scope of generating Fourier transforms and other feature extraction Methods, use of Neural Networks)
17.03.2023-07.04.2023:	Data preprocessing and formatting for use with ML models. Model training

	with FFT data from sensor and estimating feature importance for extracted features in the frequency domain.
07.04.2023-18.07.2023:	Writing up the documentation
07.04.2023-18.05.2023:	Collecting ADC data from sensors. Model training with ADC data and achieving 85%+ testing accuracy on unseen data. Testing the possibility of other extracted features from the time domain.
18.05.2023:	Reporting of observations via E-mail communication and suggested decision to use one of PSD, ACF, FFT features for model training.
18.05.2023 - 30.06.2023:	Investigating plausibility of other features to be used in the classification Model and indicating preference toward PSD features based on generated feature importance.
30.06.2023-18.07.2023:	Model training and optimization with PSD features serving as input. Tuning of hyperparameters for better and robust classification & Investigation of additional extracted features to be included in the Input layer of the model.
12.07.2023:	Demonstration of model accuracy and classification scenarios with and without sunshade opened in the infant carrier seat.
21.07.2023:	Submission of the Thesis to the examination office.

3.3 Restriction on Machine Learning Models

The scope of this classification task has been limited to use solutions only from a subset of the Machine Learning model space. The idea is to use one of the Deep Learning algorithms to solve the classification task, specifically, to either use CNN or MLP to solve the task. This would lead to better generalisation of the model and make it more robust. This would have the advantage of identifying more complex patterns and all other advantages listed in Section 2.6.2.

3.4 Real Time Classification

The requirement for the outcome of the classification is that it has to happen in real time. This is a very important criteria, as the time needed for classification should be low enough such that all echoes received by the ultrasonic transducer in a second can be classified. Thus the time required for each sample to be pre-processed, formatted, fed as an input to the classifier and classified must be acceptable. The whole process also needs to take place in an offline setup, where the pre-trained model from this work with all the necessary software packages will be deployed in the Red Pitaya.

3.5 Limited Domain Knowledge

Though the previous theoretical sections tried to introduce varying factors affecting the ultrasound reflection profile, there is not a lot of concrete domain knowledge available on how the ultrasound profiles look for complex objects. There are mechanisms of backscattering, dispersion and many more that affect the time taken for reflection and the amount of energy reflected. In situations where the behaviour of the features cannot be 100% theoretically predicted, then appropriate machine learning models must be chosen, so as to tackle variations while classification.

3.6 Order of Steps

To give a brief idea of the flow of work in the upcoming sections, an order is given :

- ❖ First, the structure of the data along with the collection mechanism is discussed. The relationship of the object's characteristics with the ADC profile is chalked out.
- ❖ Then, the data is checked for anomalies to be weeded out and some preprocessing is done to bring it to the right format and to be fed as an input to machine learning models.
- ❖ Then, the stages of feature extraction and feature importance are introduced
- ❖ The transformations used to generate a PSD of the received ultrasonic signal are introduced.
- ❖ Finally the stage of MLP model optimization and hyper parameter tuning are discussed.
- ❖ There is a further section on possible improvements that could be incorporated and tested in the future.
- ❖ Another discussion Session follows on the comparison of the performance of the model under various circumstances and how it would affect the classification result.

4 Realisation

All the measurements were taken with the front passenger seat placed at the farthest position from the dashboard. This farthest position was chosen as otherwise, the distance between the infant carrier seat and the sensor reduces to less than 30 cm. Theoretically, the minimum distance that can be ranged by the SRF02 is 15 cm [30]. But in practice, it is found that any distance lesser than 30 cm results in a sort of clipped output. This error belongs to one of the repetitive types that can be recreated and is shown in Fig 4.2. As a result, the seat position was maintained at the farthest possible position. The setup of the infant carrier seat inside the car with the various components is shown in Fig 4.1. Here, the point (c) in the figure represents the ultrasonic sensor. The point (b) represents the top of the handle at the very centre and is located at a distance of 65.5 cm. The point (a) represents the oval shaped button on both sides of the handle and is located at a distance of 71 cm from the sensor. The point (d) is the first point of contact of the ultrasonic waves and is the first object that is detected and this is the distance that is obtained in the readings, as shown in Fig 4.3. The point (e) in Fig 4.1 represents the position of the baby's head and is usually within a range of 46.5 cm to 50 cm from the sensor. The point (f) represents the position where the baby's feet are expected and is usually within a range of 80.5 cm to 87 cm from the sensor. These ranges are taken into consideration due to the varying position of the infant carrier seat and varying positions of the baby inside the seat. The seat belt to fit the infant carrier seat in place is followed exactly according to the instructions diagram from the manufacturer, as seen on the side of the infant carrier seat. The belt is secured tightly and the seat is fixed such that it is slightly inclined to the back, depicted by position (g) in Fig 4.1. This is done due to the reasons stated in Section 3.1.1.

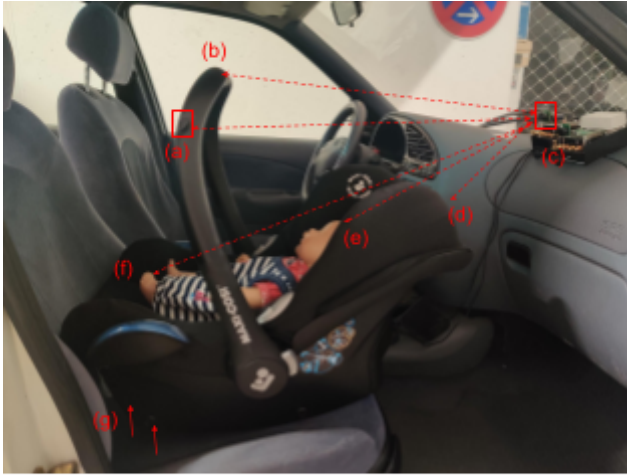


Figure 4.1: Experimental Setup inside the car

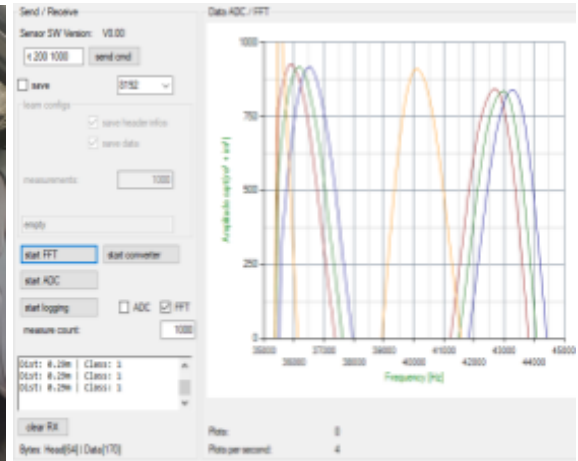


Fig 4.2 FFT plot of ultrasonic echo with distance of infant carrier seat < 30 cm

0	1	2	3	4	5	6	7	8	9	...	16391	16392	16393	16394	16395	16396	16397	16398	16399
64	32768	1	1	512	0	1953125	12	0.0	0	...	-176	-176	-174	-171	-171	-168	-167	-160	-158
64	32768	1	1	512	0	1953125	12	0.0	0	...	-129	-126	-125	-121	-122	-126	-131	-134	-135
64	32768	1	1	512	0	1953125	12	0.0	0	...	-144	-148	-148	-152	-153	-158	-159	-165	-172

Fig 4.3 Whole ADC Data (with header information) as received from the Red Pitaya

4.1 Collecting Measurement Data

With the physical setup of the seat complete, the wifi access point of the Red Pitaya must be connected via Laptop/PC used for collecting the measurement data. Once this connection is successful, the UDP Client software shown in Section 3.1.2 can be launched and measurements can be started. Once the necessary number of measurements are done, these are copied onto another folder in the local filesystem of the Laptop/PC. The names of the output text files were stored accordingly in order to help in labelling later on. The measurements with a dummy baby present are stored with the filename starting as “adc_babyxxxx” and the empty infant carrier seat starting with the name “adc_emptyxxx”. The data from the sensor was saved with the header information. Although the header information is not useful directly as inputs to be fed into the classification model, it is useful for weeding out anomalies or inaccurate, undesired signals from being fed as input. This can be understood better once the structure of the data is dealt with in detail.

4.1.1 Structure of the Data

As seen before, the measurement files are TSV text files. Table III. shows the various columns in the header part of the data and their respective values. Looking at the structure, the column at position 1 indicates the length of the data columns to follow. In the case of ADC, it is expected that 16384 columns of data follow the header and for FFT data, 85 columns of data follow the header. This is half the value for both as seen in the header.

TABLE III. HEADER DESCRIPTION - ADC DATA

Header Column Position	Header Element Type	Value
0	Header length (number of table elements)	64
1	Data Length (ADC : 32768 / FFT : 170)	32768 or 170
2	Class Detected (Object : 1/Human : 2)	1 or 2
3	Measurement type (ADC : 1/FFT : 2)	1 or 2
4	dependent on table element D either frequency resolution Δf or sampling time Δt (ADC Data : sampling time, in ns / FFT Data : frequency resolution in Hz)	512 or 119
5	currently irrelevant	0,1,2
6	sampling frequency f_s (Hz)	1953125
7	ADC resolution (bits)	12
8	currently irrelevant	0.0
9	currently irrelevant	0
10	One-way distance between sensor and first object (in m)	0.31
11	FFT Window length (1024,2048,4096,8192)	-
12	currently irrelevant	0
13	software version (Red Pitaya)	V0.2
14	Aux 1	0
15	Aux 2	0

The RedPitaya already hosts a ML model that can classify the presence of Humans in Car seats and the classification result of this model is shown in the Column at position 2. The type of recorded data, whether ADC or FFT is depicted in column at position 3. The other columns are pretty self-explanatory. The column at 10th position is of importance to this work, as this shows the one-way distance between the sensor and the first object in the beam path of the ultrasonic waves. The values specified here can be used to weed out anomalies, as stated in the previous Section. Consider example scenarios shown in Fig 4.4 (a) & (b), where the infant carrier seat position is depicted in two different positions and the red line indicates the dominant path of the ultrasonic wave travelling.

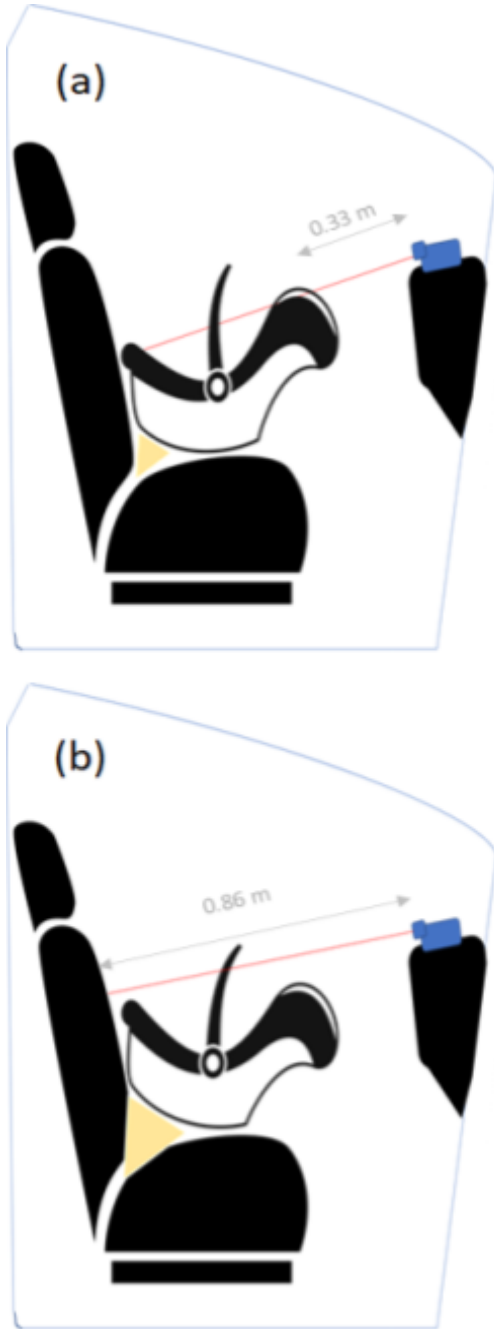


Fig 4.4 Different Tilt positions of infant carrier seat :
 (a) Correct position, First object detected at 0.33m
 (b) Incorrect position, tilted backwards too much, first object detected at 0.86 m. (The yellow shaded area in both figures show the empty space between the infant carrier seat and the passenger seat. Empty space more in (b) compared to (a))

position is depicted in two different positions and the red line indicates the dominant path of the ultrasonic wave travelling. Definitely, the ultrasonic waves travel as a more broader beam. The red line shows just the dominant portion of it. In scenario (Fig 4.4 (a)), most of the waves touch the top of the infant carrier seat first (where the baby's head is supposed to be present) and a portion of energy is reflected back from here and another considerable portion of energy is reflected from the other end of the infant carrier seat (where the baby's feet are supposed to be present). Thus, if the baby is present, considerable energy is reflected off of the baby's head and feet. On the other hand, the empty infant carrier seat at these positions reflect off the energy. But it may happen that the infant carrier seat is not fitted properly at some times. For example, when the infant carrier seat is tilted backwards too much like in Fig 4.4 (b), the majority of the ultrasonic waves do not hit any parts of the infant carrier seat. As a result, the first object detected by the sensor is only the empty passenger seat. This happens, as in this position, the only object that has a considerable area of cross-section for reflecting the ultrasonic waves is the passenger seat. In this scenario, the distance returned in the header of the data is 0.86 m or even 1 m. This distance is practically impossible to be returned from an infant carrier seat in this experimental setup. As a general rule, considering the practical setup and leaving a safe buffer, any distance greater than 0.40 m is identified as an anomaly. These readings are inaccurate in describing the reflection profile from an infant carrier seat and are therefore weeded out from being used in further analysis.

Different types of anomalies may emerge when the infant carrier seat is fitted in different positions in the same car. To generalise the ruling out of anomalies, a K-means clustering based approach can be taken to rule out these anomalies. These anomalies are 'global anomalies' and therefore, one does not need complicated mechanisms to remove them. Also, the occurrence of these anomalies is not very common if the infant carrier seat is fitted properly. As an example, the statistics from the measurements taken for this entire experiment can be considered. Out of a set of 15000 measurements taken, only 21 scans had a distance reading of 0.86 m or 1 m with anomalous profiles.

The data columns following the header contain values with both positive and negative signs. The scaling of these values and the normalisation process is dealt with in a further Section. The dummy baby used for the experimental setup has the dimensions Height (cm)* Width (cm) * Depth (cm).

As seen already in Section 2.3.1.1 and 2.3.1.2, the difference in acoustic impedance of an object to the acoustic impedance of air determines the amount of reflection of ultrasound waves and this in turn, decides the reliability of classification. Most types of plastic have a better or similar acoustic impedance to human skin. As a result the dummy baby can be a good approximation to train the model and make the model extendable to real-world situations.

4.1.2 Time of Flight Calculation

The speed of ultrasound waves, as one knows, varies due to different reasons, as stated in Section 2.2 and Section 2.3.1.1. But for the purposes of time of flight calculation to estimate the distance of the object from the sensor, standard values are assumed. Here, the sound of speed at 20°C is taken, which turns out to be 343 m/s. Assuming this value in the Equation () to calculate the $t_{roundtrip}$, one can then calculate the sample numbers that correspond to this time instant. To understand sample numbers fully, the concept of sampling needs to be appreciated. The Red Pitaya does ADC by sampling the analog ultrasonic echo received from the sensor at 1953125 Samples per second (or at a sampling frequency of 1.95 Mhz). This means, the value of the analog signal is noted down every 1953125 times per second. These observed values are then quantized to one of the discrete levels and thus form a digital signal.

Thus, taking an example calculation, when an object is placed at a distance of 0.34 m to the ultrasonic sensor, it means the ultrasound waves need 1.98 ms (milliseconds) to travel one full round trip between the sensor and the object. This $t_{roundtrip}$ value can be arrived at by directly substituting the value of $d_{oneway} = 0.34\text{ m}$ and $v_{sound} = 343\text{ m/s}$ in the Equation (2.3.1). Then these millisecond values can be converted to sample numbers. Since, for every second, 1953125 samples are generated, for 1.98 ms, 3867 samples are present. The same logic has been extended and Table IV. has been presented with varying distances and corresponding sample numbers. The minimum distance tabulated starts at 0.31 m as it is the minimum practical distance that can be ranged. Therefore common measurements of interest are tabulated.

TABLE IV. ROUND TRIP TIME AND SAMPLE NUMBER

Distance (d_{oneway}) (in meters)	Round Trip time ($t_{roundtrip}$) (in milliseconds)	Sample number	Velocity of sound (v_{sound}) (in m/s)
0.31	1.81	3530	343
0.34	1.98	3867	343
0.37	2.15	4213	343
0.40	2.33	4555	343
0.43	2.51	4897	343
0.46	2.68	5238	343
0.49	2.86	5580	343
0.52	3.03	5922	343
0.80	4.66	9111	343
0.83	4.84	9452	343
0.86	5.01	9794	343
0.90	5.25	10250	343

4.1.3 Data Pre-processing & Labelling

The data from the Red Pitaya is stored in the Local Filesystem of the PC. Both the measurements with a dummy baby and without one are stored in the same folder. Python pandas package is used to read the data from a text file and represent it in the form of a dataframe. Pandas data structure is used, as it is very efficient in quick processing of data that has many columns and rows. In a simultaneous step during the input of data into dataframes, the data is also labelled. Data from files with names starting as “adc_baby...” are labelled with a column Label = 1. On the other hand, data coming from files with names starting as “adc_empty...” are labelled with a column Label = 0. These two separate data frames are merged into one single dataframe with the ‘concat’ method. The entire code can be seen in Appendix 8.8 It is always a good idea to reset the row indices after concatenation and merge operations to avoid having duplicate indices in the data. All of the code and data can be found in the GitHub repository [36].

The next step is to detect the presence of anomalies, if any in the data. This is done by checking the distance measurement in column position 10 (with index starting at 0). As stated in one of the previous sections, distances greater than 0.40 m are to be removed. This is done by executing the first command in Listing 4.1. Here the ‘df’ refers to the name of the data frame containing measurement scans. The ‘.iloc’ operator helps one select the cross-section of desired rows and columns. In this case, the column with index 10 needs to be examined for all the rows. Thus, ‘:’ selects all rows and the ‘10’ refers to the column index. This column is then evaluated for a logical expression with ‘>=’ operator and the columns matching the condition are dropped with the ‘df.drop’ method. After the anomalies are weeded out, the header information is no longer useful and therefore the first 16 columns from the dataframe can be removed as well. This can be achieved by executing the second command in this Listing. It is always a good idea to reset the indices after removal of rows and columns to avoid having duplicate indices in the data. The structure of the data after removal of header information and supplemented with labels would look like as shown in Fig 4.5. Every row in this dataframe represents one measurement and the columns refer to the amplitude of the signal at 16384 sample instants. On further analysis of the data, it was found that the data had an embedded DC offset in the data. This was clear once the raw ADC data was plotted on a graph. The mean of the data was not 0, but rather was some negative value. To rectify this, the mean of each measurement scan had to be subtracted from the entire measurement. This was done by computing a mean for each row and storing it in a new column named ‘Mean’ in the Data frame and then doing a column-wise subtraction on all of the columns. This can be seen in Listing 4.2.

```
df.drop(df[df.iloc[:,10]>=0.41].index,inplace = True)
df.drop(baby.iloc[:,0:17] ,axis=1,inplace=True)
df.reset_index(inplace = True)
```

Listing 4.1 Commands for anomaly detection and removal

	16	17	18	19	20	21	22	23	24	25	...	16391	16392	16393	16394	16395	16396	16397	16398	16399	Label
0	-172	-175	-172	-173	-178	-176	-179	-184	-4	-116	...	-176	-176	-174	-171	-171	-168	-167	-160	-158	1
1	-157	-151	-826	-788	-732	-662	-578	-484	-388	-291	...	-129	-126	-125	-121	-122	-126	-131	-134	-135	1
2	-134	-135	-297	-145	10	164	307	441	553	646	...	-144	-148	-148	-152	-153	-158	-159	-165	-172	1
3	-175	-171	775	761	728	670	586	484	371	249	...	-186	-186	-186	-183	-182	-180	-178	-180	-175	1
4	-173	-169	-165	-163	-806	-835	-846	-846	-836	-803	...	-178	-180	-175	-175	-170	-162	-157	-156	-151	1

Fig 4.5 Structure of ADC Data without Header and with Labels

```
df["Mean"]=df.iloc[:,0:16384].mean(axis=1)
temp = df["Mean"]
df.iloc[:,0:16384] = df.iloc[:,0:16384].sub(temp, axis=0)
```

Listing 4.2 Commands for anomaly detection and removal

The first command in this listing calculates a mean based on the first 16384 columns in the data for every row separately and stores it in a column labelled 'Mean'. The second command copies this newly created column to a temporary variable 'temp'. The third command does a column-wise subtraction on the first 16384 columns with the value in the 'temp' variable. As the size of the data frame increases, it may be more time efficient to store the newly subtracted values in a new dataframe. The data can be plotted again to verify the axis of oscillation of the ADC Data.

If the classification model needs to be trained with only ADC data, then the preprocessing done upto this stage would be sufficient. The data can be normalised to range from a minimum of -1 to a maximum of +1 and be fed as a 1-D numpy array to the MLP model. But if the model needs to be trained on some other form of the same data, then necessary steps from Section 4. need to be implemented as per requirement. But the decision of using ADC Data directly for training can be taken with getting more information on the usability/reliability of the features in ADC Data for classification purposes.

To do that, a basic relationship on how the difference in the distance of an object affects the sample numbers in an ADC signal. For this purpose a large flat object providing an ample area of cross-section for ultrasonic wave reflection was tested as an object to be detected. This setup was created outside the environment of a car to measure in a clutter-free environment. The large object was moved farther away from the sensor by 3 cm each time and 6 such measurements were made consecutively. A sum of 200 measurement scans were recorded for each distance and there are a total of 1200 measurement scans. The distance ranged from 0.34 m till 0.49 m. The physical distance and the distance from the sensor was noted and verified to be equal. Then all of these measurements were plotted together and the resulting output is shown in Fig 4.6. This shows clearly 6 groups of measurements as expected. The pattern seen here is that, for every 3 cm increase in the distance of the object, the reflection gets delayed by 342 samples. This is roughly the spacing between each of the neighbouring consecutive measurement groups shown in this figure. Also, it can be seen from the image that, as the distance increases, the amplitude of the reflection decreases slightly. This pair of distance and sample matches with the theoretical value calculated as per Table IV.

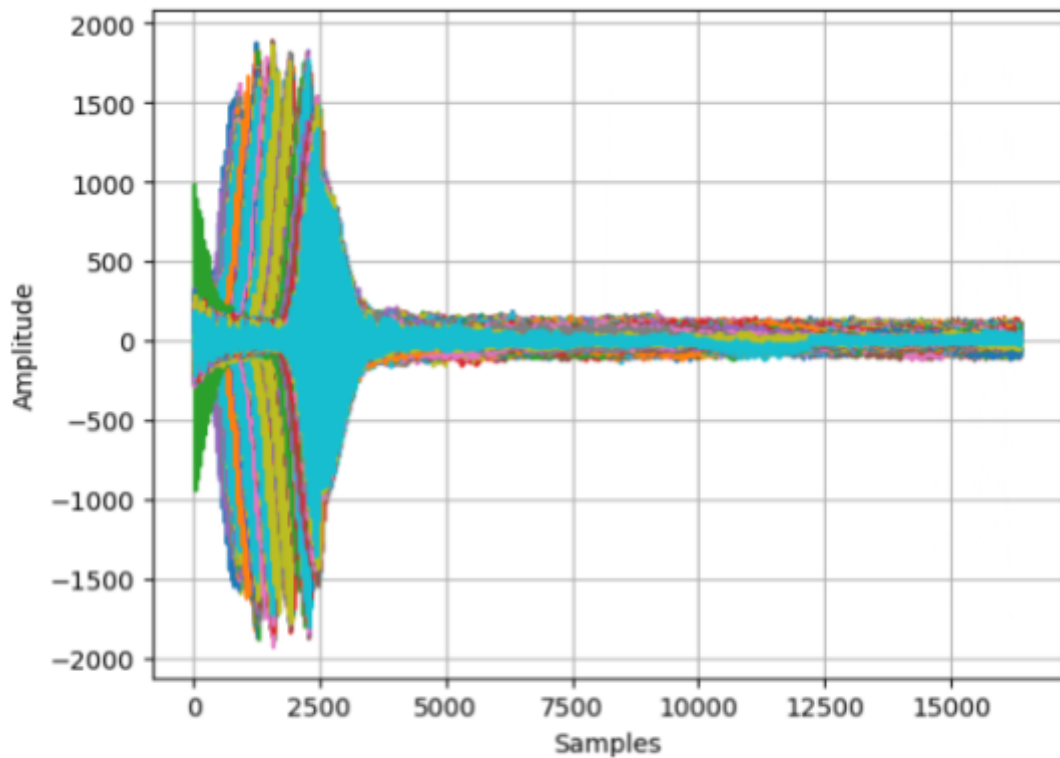


Fig 4.6 Reflection Profile of Large Flat Object in External Setup with varying distances from Ultrasonic Sensor (0.34m,0.37m,0.40m,0.43m,0.46m,0.49m) - Plot of 1200 measurements

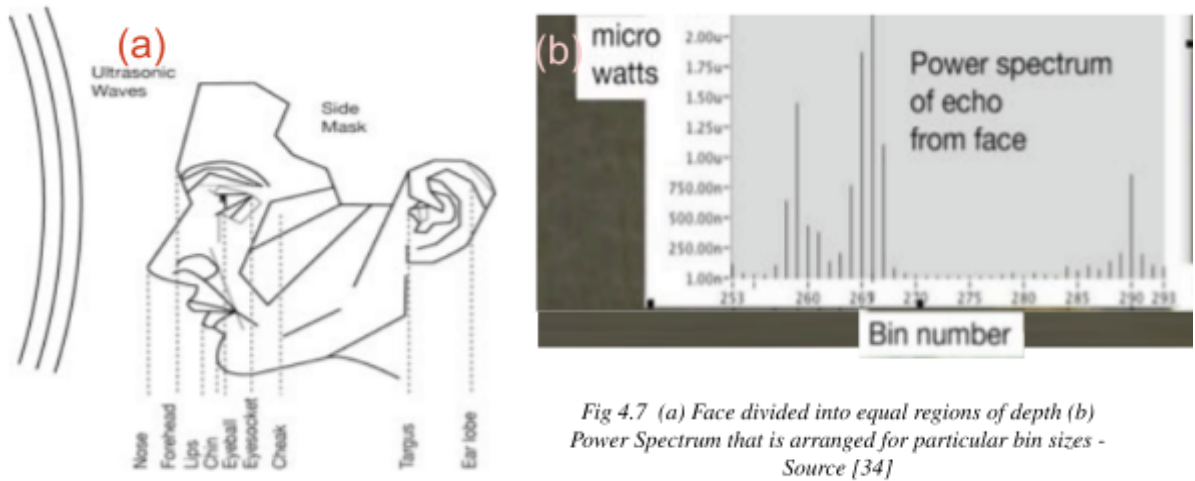


Fig 4.7 (a) Face divided into equal regions of depth (b) Power Spectrum that is arranged for particular bin sizes - Source [34]

A line of reasoning used in this project is adopted from the work done on identifying Facial Geometry with the help of ultrasonic reflection profiles [34]. The Fig 4.7 (a) shows the face being modelled with the help of a 'Repose Mask model' into various geometric regions of similar depth. This work then goes on to show how the time-frequency spectrogram of the received echo is influenced by the facial features. When a face is placed orthogonal to the direction of incident ultrasonic waves, then the first echo can be expected to be from the tip of the nose and the last echo to be from the earlobe. Between these extremes, the amplitude of the echo would be proportional to the area of the face that is orthogonal to the beam.

For example the forehead could produce significant echo amplitudes, due to its large cross-sectional area. The power spectrum obtained from the echo reflected off a human face is shown in Fig 4.7 (b). This spectrum is compared to a side photograph of the human face and both of them are scaled to match each other. This photograph along with the scaled comparison of the power spectrum can be read from [34].

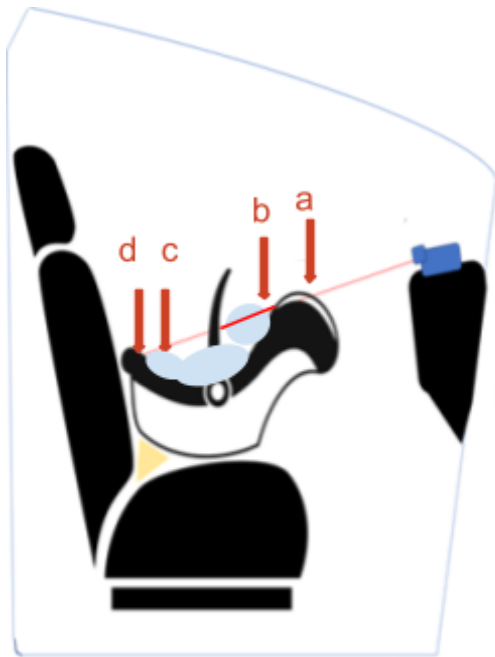


Fig 4.8 Order of objects detected as per the Reflected wave travelling back to the sensor

When comparing each facial feature at a particular position to the power spectral density at this equivalent position in the power spectrum, certain interesting patterns arise. The small value of amplitude at Bin number 253 in Fig 4.7 (b) corresponds to the energy reflected off the nose. This is expected since the nose has a very small cross-sectional area incident to the ultrasonic waves. This also makes sense to appear at the very beginning in the power spectrum, as the nose is the closest part of the face to the ultrasonic sensor. The first peak around bin number 260 lines up with the bridge of the nose and roughly the forehead section. The second peak lines up with the back corners of the eye sockets. In this work, it was proposed to use the distance between the first and second peaks as a feature to classify different faces. As different people possess different ratios and positions of facial features, this idea was proposed. A similar idea like in the project [34] is also extended to the experimental setup in this work to derive preliminary insights about the ultrasonic reflection profile of the complex geometry of an infant carrier seat.

Consider Fig 4.8 again, one can imagine the experimental setup from the point of view of the sensor. An observer looking at the infant carrier seat from this angle would first

observe the edge of the infant carrier seat pointed by the arrow (a) in this figure. This is the closest surface of the infant carrier seat to the sensor and where the ultrasonic waves would experience reflection first. If the baby is present, the next point of contact would be the position indicated by the arrow (b), where the baby's head can be expected. If there is no baby present, then the position of contact is still as directed by this arrow, but a little deeper (and farther by 1 or 2 cm) somewhere on the surface of the infant carrier seat. The third point of contact would be the position as indicated by arrow (c), where the feet of the baby can be expected, if the dummy baby is present. Otherwise, this arrow would point to a deeper point somewhere on the surface of the infant carrier seat, just like with the arrow (b). Finally, the next point of contact would be somewhere at the very tip of the infant carrier seat that is fastened against and touching the passenger seat, which is indicated by the arrow (d). Preliminary logic says that in the reflection profile of an empty infant carrier seat, minor peaks in amplitude can be seen at points indicated by arrows (a) and (d). Whereas if the dummy baby is present, then these minor peaks are more spread out into two or more even smaller peaks in nearby areas.

However, in practice this cannot be exactly expected as the time of travel for backscattered waves can be higher, as they take a longer dispersed route. Also, the acoustic impedance of the fabric material used in the infant carrier seat cannot be judged accurately. Still the overall trend can be seen and will be explained in a later Section.

4.2 Feature Importance and Feature Extraction

The process of identifying which columns of data are useful and more influential on the classification result is termed feature selection. To do that, the feature importance of all the columns must be understood. In the data used in this experiment, there are 16384 columns each representing a value. Naturally, all of these columns are not equally useful in deciding the classification result. To estimate how important each individual feature is, the random forest model can be used. The Listing 4.3 shows the code executed for creating modified random forest decision trees, also known as extra trees regressor. Here 'X_train' refers to all the columns of data (except Label column) in Fig 4.3 after normalisation. 'y_train' refers to the 'Label' column from the same data frame.

The Extra trees create a bunch of decision trees, with each having a random sample of the total population. Each tree only sees a subset of all the features and does the estimation. Finally the results are aggregated. The trees model was used to estimate the feature importance, rather than using simpler linear regression or logistic regression models, because of the higher tolerance to newer data and less sensitivity to the original data. This is very important in the context of this work, as not all possible positions of the infant carrier seat and the infant inside the infant carrier seat can be captured and fed as input to the Machine Learning model. As a result, robustness must be built into the model to handle unexpected situations as well.

```
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
regmod = ExtraTreesRegressor()
regmod.fit(X_train,y_train)

plt.figure(figsize = (12,8))
feat_importances = pd.Series(regmod.feature_importances_, index=X_train.columns)
feat_importances.nlargest(50).plot(kind='barh')
plt.show()
```

Listing 4.3 Commands to generate Feature Importance Plot with Extra Trees Regressor Model

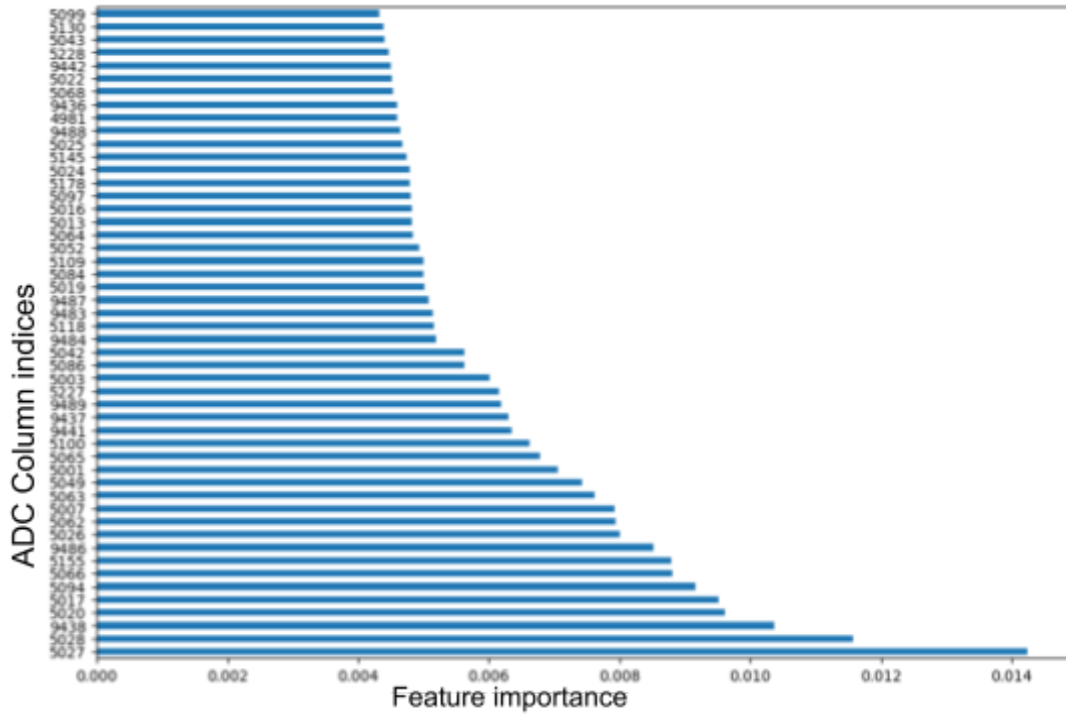


Fig 4.9 Feature Importance of ADC Data Columns (the top 50 features in ascending order are depicted, with the x-axis showing the Importance of the Feature as a decimal)

The output plot of the code in Listing 4.3 is shown in Fig 4.9. It shows the top 50 columns of importance in deciding the classification result in ascending order. On taking a closer look at the position of these columns in the ADC data, it can be seen that it is clustered around 2 groups of sample ranges : one is the group of samples in the range of 5000-5300 samples and the other is the group of samples in the range of 9200-9500 samples. Recollecting the distances shown in Fig 4.1 clarifies that the exact objects at this distance from the sensor include the baby's head and the baby's feet. The theoretical value calculation can be referenced from Table IV. Thus, the feature importance plot verifies the assumption made earlier (based on the order of objects traversed and the amplitude echoes returned).

The level of feature importance achieved with plain ADC data is pretty low. To improve the classification, certain signal transformations have to be performed on the ADC data. This is discussed in the next Section.

TABLE V. IMPORTANT PYTHON PACKAGES USED AND THEIR DESCRIPTION

Python Package	Description
Pandas	Used to read input data and store it in a dataframe format. Efficient for processing huge data sets
numpy	Provides built in mathematical operations, used to generate FFT of time domain samples
statsmodels	Provides readymade tools to generate ACF, other statistical measures
sklearn	Holds the Random Forest model, Extra Trees model for estimating Feature importance
keras	Contains code to customise and build own MLP, CNN models
scipy	Provides built in tools to perform signal processing tasks, eg : peak finding in signals, etc

4.3 Signal Transformations (Windowing, ACF, PSD)

Now, it is clear that the portion of the ADC signal from 4500-9500 samples is more important than the others. Also, any time domain signal in a real world scenario is not measured infinitely and therefore the abrupt start and end boundaries cause leakage in the frequency spectrum representation of the same signal. To solve both these issues, a windowing function can be applied to the ADC Data. The code in Listing 4.4 calculates the window length to be applied. This value is chosen equal to the length of the ADC signal, i.e. 16384 samples.

Then the ‘np.hamming’ method from the numpy package in Python is called to generate a hamming window. A plot of this generated hamming window can be seen in Fig 4.10. As a result, the window applies less weightage to samples at the very beginning and at the very end and applies maximum weightage to those samples in the middle portion. This is good, as the features of importance 4500-9500 also lie in the middle portion of a signal with length of 16384. The third line of code in the Listing 4.4 does an element wise multiplication of the ADC signal represented by the ‘df’ data frame with the hamming window represented by the ‘adcwindow’ variable. The result is stored in another variable ‘windowedadcsignal’.

A comparison of the Raw ADC signal with its windowed version is depicted in Fig 4.11. This is shown for both scenarios where the infant carrier seat is empty and fitted with a dummy baby. Cases (a) and (b) in this figure depict the raw ADC signal. In these two cases, it is hard to distinguish amplitude levels and spacing of peaks and other features in the 4500-9500 sample range. But after windowing, as shown in Cases (c) and (d), it is clearer and amplitudes are more dominant and provides a better input, if fed to the classification model. An overall trend that can be observed in the reflection profiles of baby and empty seat, is that for the baby reflection profile, the peaks at the sample range of 5000’s is more lower and dispersed than that of the empty profile. On the other hand, the amplitude in the sample range of 9000’s contains earlier peaks in a baby profile than that of an empty profile. These are just trends observed in the samples. But this cannot be ensured to happen everytime and cannot be used as a hard measure for classifying. Therefore, other signal processing techniques need to be applied as well.

Since classification in this work needs to happen in Real-Time, the efficiency of code used to generate these signal processing outputs also needs to be examined. A list of the most important Python packages used in this work is listed in Table V. The official documentation of these packages list the algorithms used behind these ready-to-use tools and runtime optimization can be made, if the inbuilt signal processing tools in Red Pitaya offer a better performance than the Python packages mentioned above. This could be especially helpful for tasks like generating ACFs. It can be recalled from Section 2.4.1 that for Stochastic signal analysis, the Autocorrelation Function can be a useful measure to understand the correlation between delayed versions of the same signal. The code in Listing 4.5 can be executed to generate the ACF of a signal with only positive lags, with the maximum number of lags being equal to the length of the signal (16384).

```

windowlength = 16384

adcwindow = np.hamming(windowlength)

plt.plot(adcwindow)

windowedadcsignal =
df.iloc[:,0:16384] * adcwindow

```

Listing 4.4 Generating Hamming Window and Windowed ADC Signal

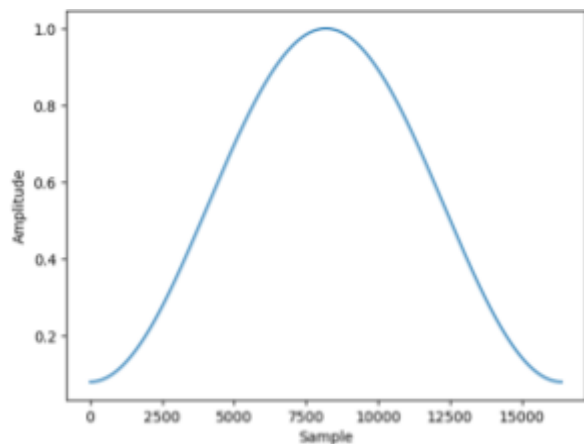


Fig 4.10 Hamming window of length 16384

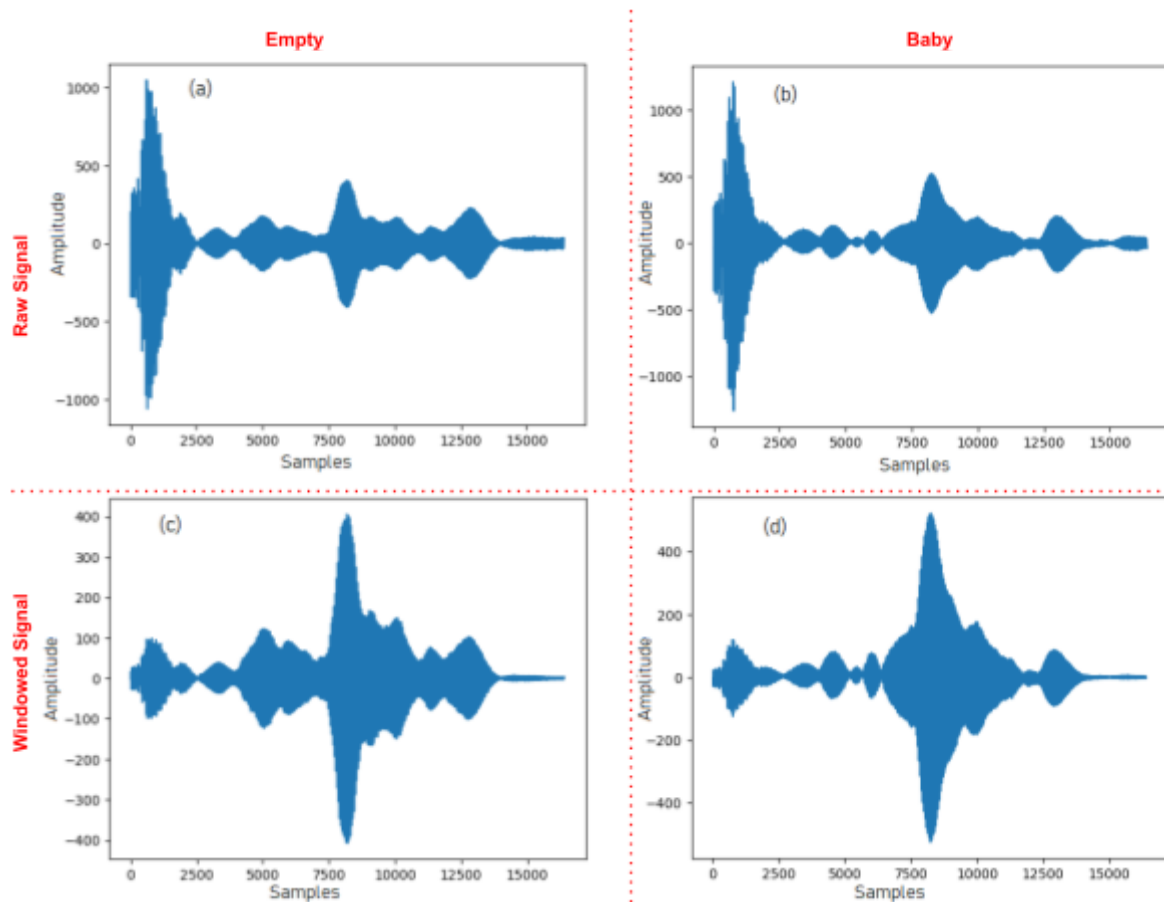


Fig 4.11 Comparison of Raw ADC signals and Windowed ADC signals

- (a) Raw ADC Signal - empty infant carrier seat
- (b) Raw ADC Signal - presence of dummy baby
- (c) Windowed ADC Signal - empty infant carrier seat
- (d) Windowed ADC Signal - presence of dummy baby

```
from statsmodels.tsa.stattools import acf

#Find the number of rows in the dataframe
lengthvar = len(df)

#Iterate one row at a time through the dataframe
positiveacf = pd.DataFrame(acf(df.iloc[i,0:16384], nlags = 16383)
                           for i in range(lengthvar)
                           )

#Affix Label and Save File
positiveacf["Label"] = df["Label"]
positiveacf.to_csv('PlainsignalACF-Part5.csv', encoding='utf-8', index=False)
```

Listing 4.5 Generating ACF with only positive lags

The ‘acf’ method from the statsmodels package accepts only one-dimensional arrays as input. Therefore, the data frame is iterated one row at a time with the help of a ‘for’ loop. This is not the most efficient way to generate an ACF. This can be optimised definitely, since the RedPitaya has a number of efficient built-in signal processing tools. The label is then affixed to the Positive portion of the ACF generated. The last line of code in this Listing helps to save the ACF generated in a local file in the ‘.csv’ format. This is done to be able to later perform intermediary analysis on using ACF features to train the model or to estimate feature importance. The plot of a sample ACF is shown in Fig 4.13

The next step is to test the plausibility of using ACF features to train the model and test the reliability of classification. For this purpose, again the feature importance is estimated using the Extra Trees Regressor model seen earlier, as in Listing 4.3. The feature importance of ACF lags is shown in Fig 4.12. It can be seen that the feature importance is even lower for the ACF data than the windowed ADC data. Therefore additional transformations need to be performed.

Generally for the analysis of time domain signals in the medical field, like ECG, EEG or EMG signals, it usually helps to analyse these signals in the frequency domain. Some of the frequency domain representations include the FFT, PSD, Wavelet Transform and so on. First, the FFT can be generated and tested to see if it satisfies the criteria of being good features for classification. The transformation from time to frequency domain is carried out efficiently by the FFT algorithm, as seen in Section 2.4.5. The code in Listing 4.6 can be executed to calculate the fourier transform of the windowed ADC signal. There are real and imaginary components in the result of the FFT. The absolute value is only extracted (`np.abs(x)`) and this value is divided by the number of samples (`N`) used to generate the FFT. This way the amplitude obtained is normalised. It is seen that the first 8192 values in the FFT result matter and the second half is just a mirrored image of it. Amidst these first 8192 points, many of them only have zero amplitude. The important and non-zero columns among these are filtered and shown in Fig 4.14.

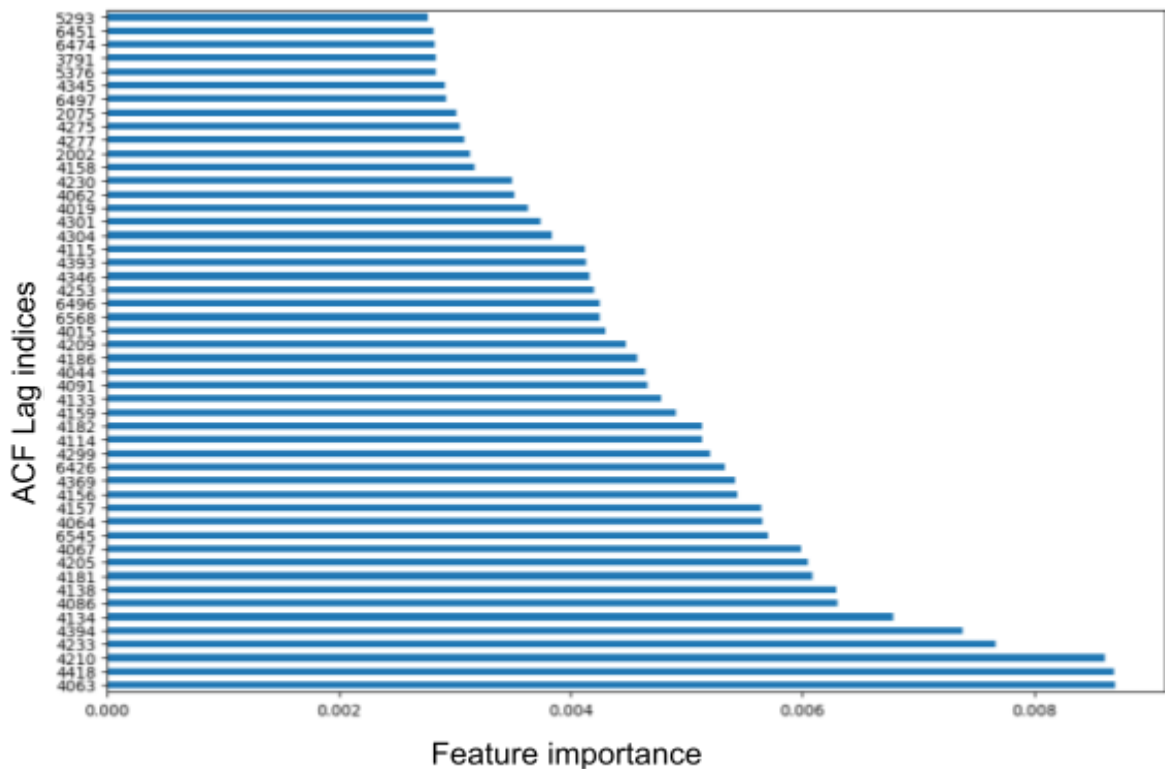


Fig 4.12 Feature Importance of ACF Lags (the top 50 features in ascending order are depicted, with the x-axis showing the Importance of the Feature as a decimal)

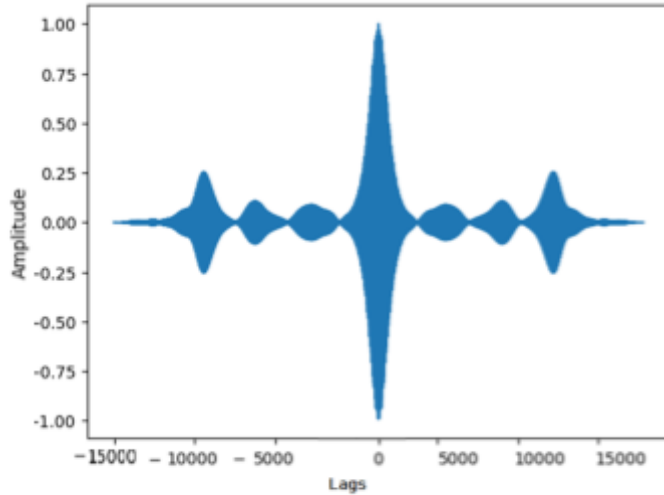


Fig 4.13 AutoCorrelation Function plot of a sample ADC signal with both negative and positive lags (Total length = 32768)

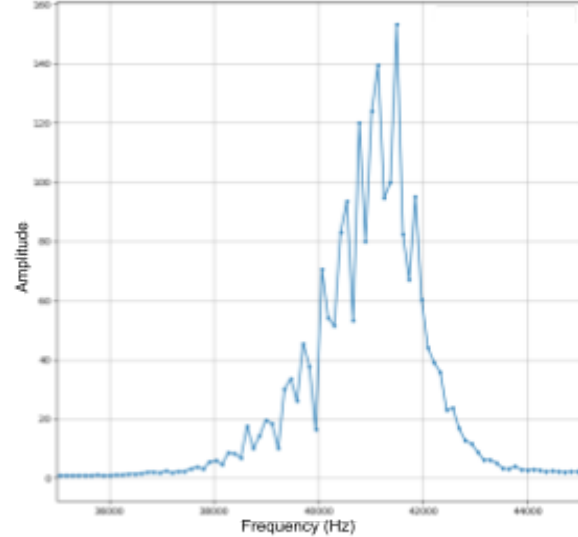


Fig 4.14 FFT generated

The last line of the code in Listing 4.6 helps store the FFT generated in a local file in the '.csv' format. The x-axis in the plot is scaled with the help of the code shown in Listing 4.7. Here, the frequency axis for the FFT plot is generated using the numpy package's 'linspace' operator. The idea here is to generate a linear space ranging from 0 to 1953125 each with a spacing of F_s/N , where F_s is the sampling frequency and N is the number of samples in the time domain data. The reason for the spacing being calculated this way is explained in Section 2.4.5.

```
from numpy.fft import fft, ifft
X_fft = np.fft.fft(windowedadcsignal)
N = 16384
X_fftmag = np.abs(X_fft)/N

X_fft_df = pd.DataFrame(X_fftmag)

X_fft_df["Label"] = df["Label"]
X_fft_df.to_csv('FFT_1.csv', encoding='utf-8', index=False)
```

Listing 4.6 FFT Generation

```
#Sampling frequency
Fs = 1953125
#Number of samples
N = 16384

#Frequency axis creation
f1 = np.linspace(0, Fs/2, int(N/2))
#Plotting the FFT
plt.plot(f1,X_fft_df.iloc[x,0:16383])
```

Listing 4.7 Frequency axis generation and FFT plotting

The plot is almost similar to the FFT generated by the Red Pitaya in the sensor kit. The last line of code in the Listing 4.7 executes the plotting command with the frequency axis and FFT arrays previously generated. For clarity, just one FFT sample is plotted. This sample is picked by mentioning the row index 'x' in the 'iloc' operator on the 'X_fft_df' dataframe.

The next step is to test out the possibility of using the PSD of this stochastic signal as a reliable feature for classification. To calculate the PSD for stochastic signals, the formula from Section 2.4.1.1.6 can be used. Referring to the technique in the previously mentioned Section, the ACF needs to be calculated first and then the FFT of this ACF will yield the PSD of the signal. This approach is executed with the help of the code in the Listing 4.8. The code is similar in structure to the code in Listing 4.6. But, only the input to the fft function changes. In this case, it is the 'acf_signal'. In the previous Listings, the code for generating ACF was seen. This gives only a half of the ACF signal with only positive lags. The 'acf_signal' has both the negative and positive lags and is symmetric around Lag=0. This signal has a total length of 32768 data points. This is fed as input to the fft function in the first line of code in this listing. Then the number of samples this time is adjusted to be 32768, as the 'acf_signal' contains both positive and negative lags. Again, the absolute values are considered and are scaled by a factor $1/N$ with the only difference that $N=32768$ this time. Accordingly, an appropriate frequency axis 'f2' is generated.

```
X_fft = np.fft.fft(acf_signal.iloc[:,1:32768])
N = 32768
X_fftmagpsd = np.abs(X_fft)/N
f2 = np.linspace(0, Fs/2, int(N/2))
```

Listing 4.8 PSD Generation

The code in Listing 4.9 plots the PSD. Here 'x' refers to an index of a sample with the Label = 0 and 'y' refers to an index of a sample with the Label = 1. The graph is showing only the important non-zero columns of PSD data with the newly generated frequency axis 'f2'. The output of this code is seen in Fig 4.15 The y-axis contains normalised values of amplitude of the PSD. The usefulness of the PSD data columns as features for classification can be tested by running feature importance on these as well. The results of the feature importance generated is displayed in Fig 4.17. It can be seen that the overall importance factor is 3-4 times better than that of the ADC data features.

```
#Plotting the PSD
plt.plot(f2,X_fftmagpsd.iloc[x,0:16383],'.-', label='PSD,empty')
plt.plot(f2,X_fftmagpsd.iloc[y,0:16383],'.-', label='PSD,baby')
plt.grid(True)
plt.xlim([35000,45000])
plt.legend()
plt.show()
```

Listing 4.9 Plotting the PSD for infant carrier seats that are empty and containing dummy baby

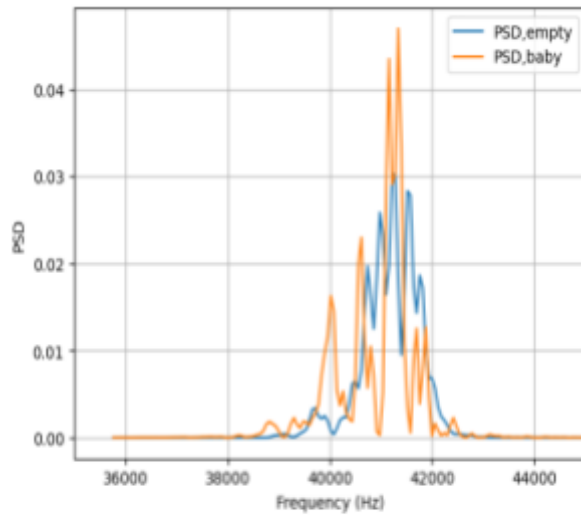


Fig 4.15 PSD Profile of infant carrier seat that is empty and infant carrier with dummy baby

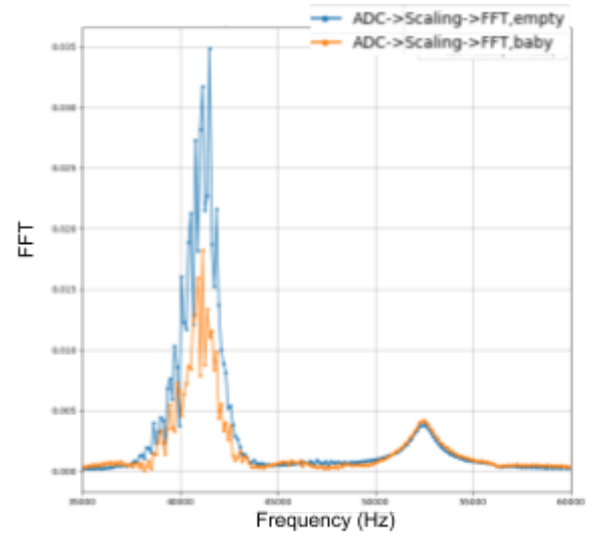


Fig 4.16 FFT Profile of infant carrier seat that is empty and infant carrier with dummy baby

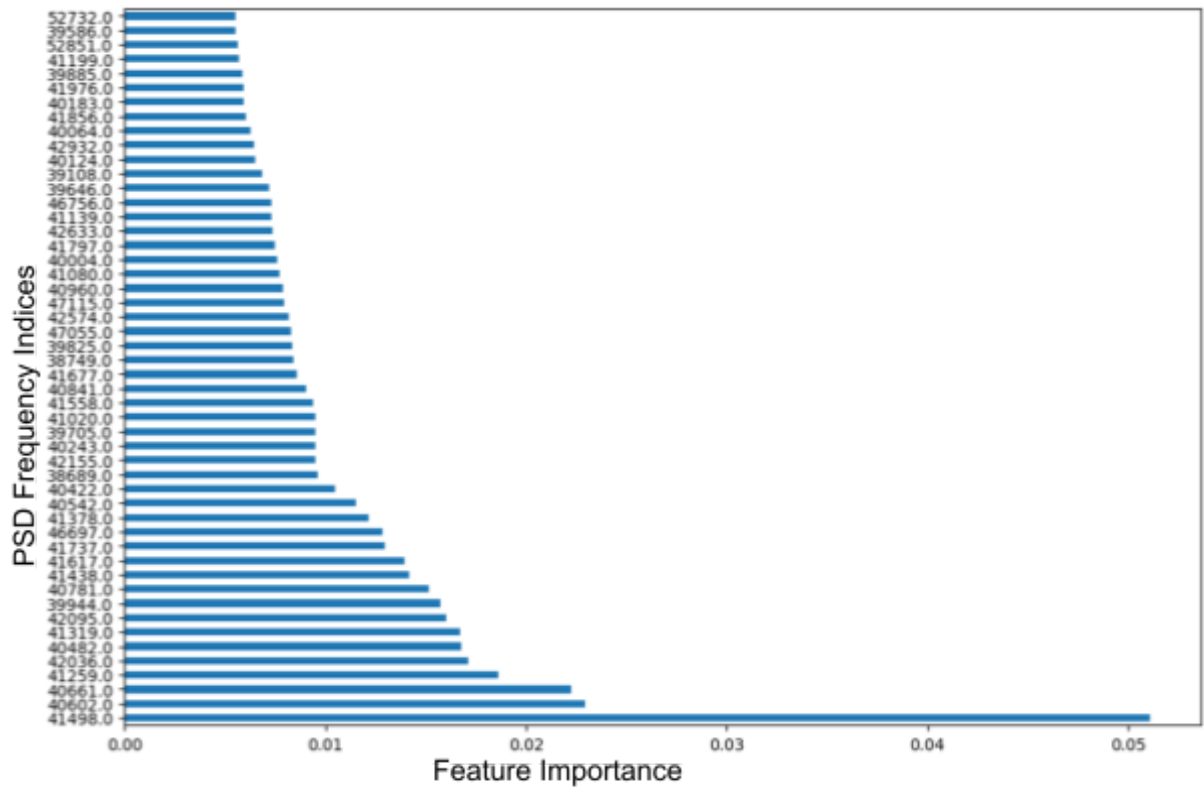


Fig 4.17 Feature Importance of PSD Data Columns (the top 50 features in ascending order)

As an interesting comparison, the FFT profile of empty infant carrier seats versus the infant carrier with a dummy baby is portrayed in Fig 4.16. Comparing the PSD profiles in Fig 4.15 with the FFT profiles in Fig 4.16, it is seen that the PSD Profile with the presence of a baby has higher peaks relative to the PSD Profile of empty infant carrier seat. On the other hand, the FFT profile of the empty seat has higher amplitude than the infant

carrier with the presence of a baby. The same time domain signals have varying characteristics when represented in other ways. The next section deals with training the model with PSD as input features.

4.4 Building MLP Model & Optimization

The idea is to try training the MLP model with the PSD data obtained before as input features. The first step is to format the data into the right shape before feeding them as input features to a MLP model. The first line in the code in Listing 4.10 splits the data into a training and testing set with 80% ratio of the data going for training and the remaining 20% for testing. This is executed with a random shuffle, so that empty and baby labelled data are mixed thoroughly. Then the training data needs to have the target column (column : 'Label') separated from it. The same needs to be done for the test data as well. Thus the label information is held in 'y_train' and 'y_test' arrays. The PSD local files saved contain 400 columns of data. But since only the first 200 are non-zero, these are only considered for the purposes of both training and testing.

```
train, test = train_test_split(total2, test_size=0.2, random_state=42,
                               shuffle=True)
X_train = train.iloc[:,0:200]
y_train = train['Label']
X_test = test.iloc[:,0:200]
y_test = test['Label']
```

Listing 4.10 Splitting the data into Training and Test Data Sets

The next step after the train-test split is to bring the data arrays / data frames to the right shape. For example in the case of an MLP model, the input needs to be flattened. This means, it has to be brought to a one-dimensional form. To do that, the code in Listing 4.11 transforms the data frame 'X_train' to a numpy array 'X_train_np' and then reshapes it in the second line. The same is done for 'X_test'. For example, a data frame that was initially 13344 * 200 is reshaped into 13344 * 200 * 1.

```
X_train_np = X_train.to_numpy()
X_train_np = np.reshape(X_train_np, (X_train.shape[0], X_train.shape[1], 1))
X_train_np.shape

X_test_np = X_test.to_numpy()
X_test_np = np.reshape(X_test_np, (X_test.shape[0], X_test.shape[1], 1))
X_test_np.shape
```

Listing 4.11 Changing the input feature format to 1D numpy array

The next step is to build the layers in the MLP model. The code in Listing 4.12 is used to realise it. The input dimensions are a necessary input when trying to build the input layer. The number of neurons, epochs and batch size are stored as variables. This way these can be changed to various values either manually or through a schedule and can then be run several times to find the best fit. The 'Sequential' model is chosen, as the idea here is to build a MLP model. The sigmoid activation function is chosen and the Binary cross entropy loss function is

chosen, as this is only a binary classification problem. The optimizer chosen here is Adam, as this has proven to be the most efficient for most of the Machine Learning models and use cases. Once all the layers with the necessary number of neurons are added, the model can be compiled and the model summary is seen in Figure 4.18. This estimates a total of 12929 parameters (network weights) to be trained, which is relatively very low in comparison with most MLP or CNN models.

```
input_dim = len(X_train.columns)
neurons = 64
epochs = 50
batch_size = 50
model = Sequential()
model.add(Dense(neurons, input_dim = input_dim, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.summary()
```

Listing 4.12 MLP Model Building

```
Model: "sequential_3"
Layer (type)                 Output Shape                 Param #
-----
dense_6 (Dense)              (None, 64)                  12864
dense_7 (Dense)              (None, 1)                   65
-----
Total params: 12,929
Trainable params: 12,929
Non-trainable params: 0
```

Fig 4.18 MLP Model summary

The model can then be trained with the input prepared earlier. This is realised with the code in Listing 4.13. After the training step, the evaluation happens on the test set split earlier. Before diving into the discussion on the trends of accuracy and loss, some important things are to be noted regarding the splitting of data into training and test sets. ADC measurement scans were made mostly in batches of 100 or 200, due to the long time required for all of the 16384 columns to be transferred serially from the Red Pitaya to the Laptop. Therefore the data was structured such that 100 rows of the same event (eg: empty seat) existed together. The next 100

rows could be again of the same event (empty seat) or a different event (seat with baby). This varied as many days of measurements were combined in the training dataset. This order was also preserved while generating ACFs and PSDs, as it was the only way to affix labels to these generated ACFs and PSDs later.

With that known, one has to understand how the train-test data splitting happens in the code in Listing 4.10. Here if the parameter 'shuffle' in the 'train_test_split' method were set to 'False', then it means data is not randomly split into training and testing data sets, but rather it split orderly based on the order maintained in the data frame. Usually order needs to be preserved for time series data sets, where forecasting needs to happen. Therefore future values cannot be fed as an input to the model. But in this work, ordering has no meaning with respect to the events happening (baby seat or empty seat). Therefore, it was decided to have the 'shuffle' parameter set to 'True'. What this essentially means is, there are no longer 100 rows of the same event stacked together. When the randomised splitting happens, the test set is assigned random indices of rows (as partially decided by the 'random_state' parameter in the code). Common understanding of probability would enable one to come to this conclusion : It is highly likely that some measurements of the same batch end up in the Training set and few other measurements from the same batch end up in the Testing set. Measurements of the same batch when plotted together mostly are very similar and overlap each other, as shown in Fig 4.19. On the other hand, different measurements from various batches depicting the same event would look like Fig 4.20. Clearly, the amount of variance is more with Fig 4.20 than Fig 4.19. Thus, the testing set derived in the code in Listing 4.10 is just like

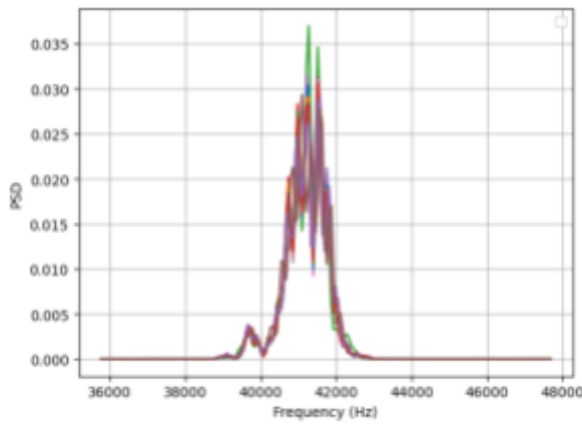


Fig 4.19 Empty infant carrier PSDs from the same measurement batch

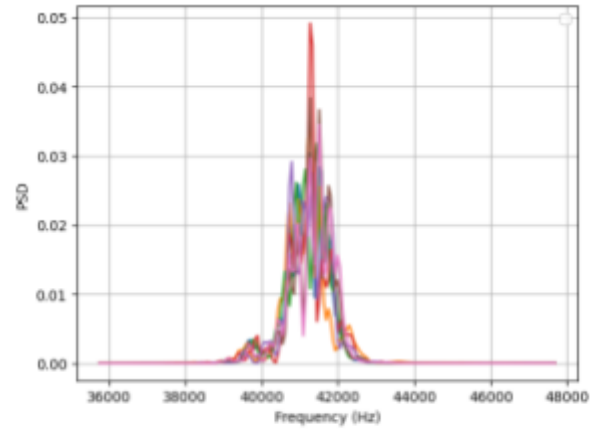


Fig 4.20 Empty infant carrier PSDs from different measurement batches

an improved validation set. Essentially, this renders the test set not very effective in assessing the performance of the classifier.

As a result it was decided to obtain the real test set not via the code mentioned earlier. But rather these measurements are stored in a separate ‘test’ folder and therefore do not have any sort of connection with the training data sets or the validation data sets. They are then read into a separate data frame. The code to perform this is shown in Appendix 8.9. Therefore in the remainder of this work, training accuracy and validation accuracy refers to the performance of the classifier on the training dataset. Whereas the term ‘testing accuracy’ or ‘testing loss’ refers to the performance of the classifier on the unseen test data, recovered from separate folders.

The history of accuracy of training with each epoch is stored in a variable to be plotted on a graph later on. This graph is displayed in Fig 4.21(a) The history of the validation dataset is termed ‘val’ by convention and this is followed in the legends in the graph. It can be seen that accuracy increases with the number of epochs, but after a certain point, the increase is not significant.

Then the comparison graph of the testing vs validation loss is plotted in Fig 4.21 (b) and is shown epoch-wise. In these situations, the divergence between the training and the validation loss needs to be noted. If the divergence grows larger, this is a good indicator to denote the point where the training should be stopped. This is because increasing divergence means that the model is overfitting to the training dataset. In this figure, it could be said that beyond 50 epochs, the divergence grows larger and therefore, 50 is a reasonable number of epochs to limit the training phase to, as it is a simple model.

```
history = model.fit(X_train_np,y_train, epochs=epochs, batch_size=batch_size,
verbose=1, validation_split=0.2)
predictions= model.predict(X_test_np)
scores = model.evaluate(X_test_np,y_test, verbose=1)
print(scores)
```

Listing 4.13 MLP Model Training

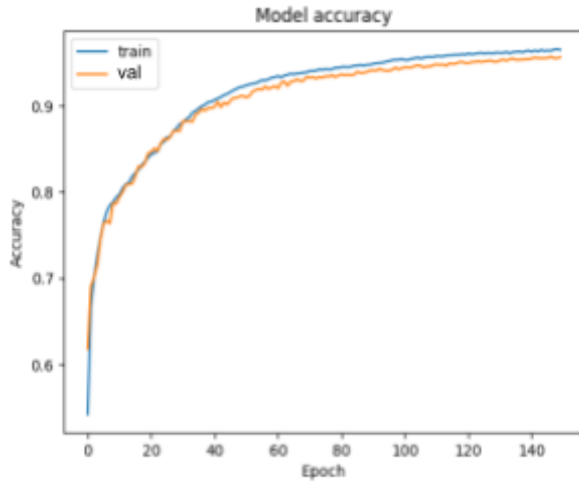


Fig 4.21 (a) History of Training and Validation Accuracy of MLP Model with parameters as mentioned in Table VI

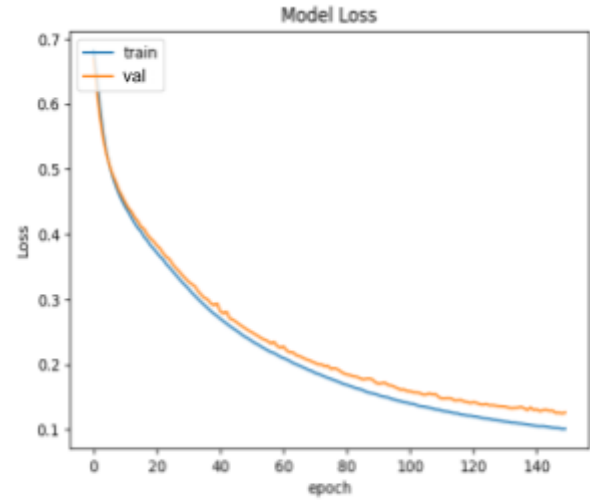


Fig 4.21 (b) History of Training and Validation Loss of MLP Model with parameters as mentioned in Table VI

TABLE VI. MLP MODEL PARAMETERS-I

Batch_size	50
Neurons	64
Order of Layers	Dense (64) with ReLu, Dense(1) with Sigmoid
Optimizer	Adam
Loss	Binary cross entropy
Training datasets	PSDs - Part1, Part 1.2, Part2, Part3.2, Part3, Part4, Part5, Part 7

TABLE VII. MLP MODEL PARAMETERS-II

Batch_size	128,64,32,16,1
Neurons	64

The number of epochs and neurons were fixed. The next question was to optimise the batch size. Different parameters for the batch size as per Table VII were tried. The output is presented in Fig 4.22 (a) and 4.22 (b). Now that the model is optimised, different sets of testing data were tried. The unseen data that has been kept from the model training phase are placed in a folder called “PSDfromwindowedADC_test” according to the folder structure in Appendix 8. The file named “PSDfromWindowedADC-Part6” is normal test data with the sunshade of the infant carrier tucked in properly. The file named “PSDfromWindowedADC-Part9” is test data with the sunshade of the infant carrier pulled out fully. Model prediction was carried out individually on each of these two files. The results are shown below. Fig 4.23 shows the confusion matrix of the results pertaining to the test data file, Part6 (which contained normal test data) Fig 4.24 shows the confusion matrix of the results pertaining to the test data file, Part9 (which contained data with sunshade pulled out). Surprisingly, the results for Part 9 were better than Part 6 in this case. This proves how robust the model is. The model was able to handle a situation not well known to it. The corresponding classification metrics are tabulated in Table VIII and Table IX.

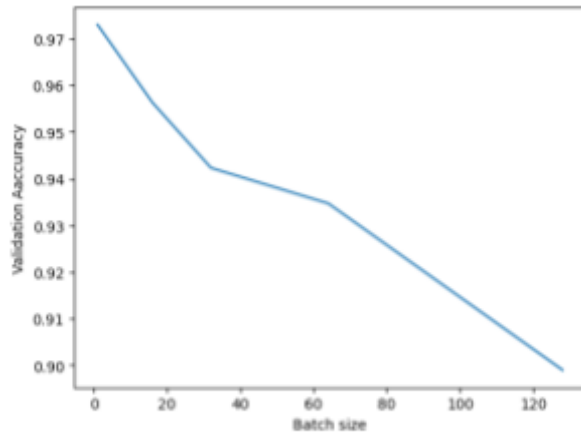


Fig 4.22 (a) Validation accuracy for various batch sizes as per parameters in Table VII

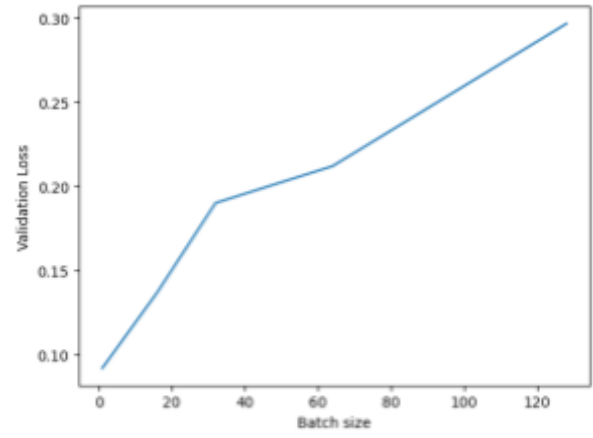


Fig 4.22 (b) Validation loss for various batch sizes as per parameters in Table VII

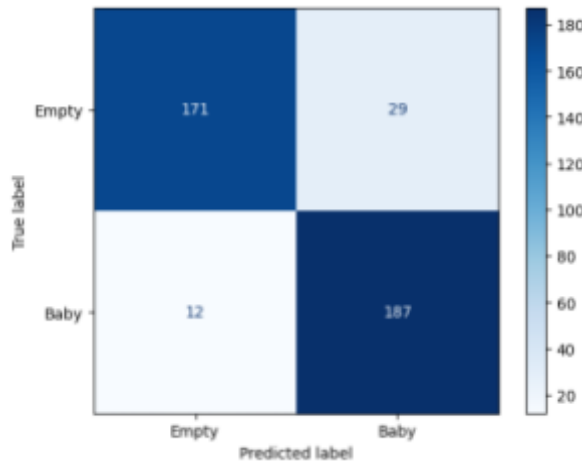


Fig 4.23 Confusion Matrix of Model with parameters as per Table VI and testing data = "PSDfromWindowedADC-Part6.csv"

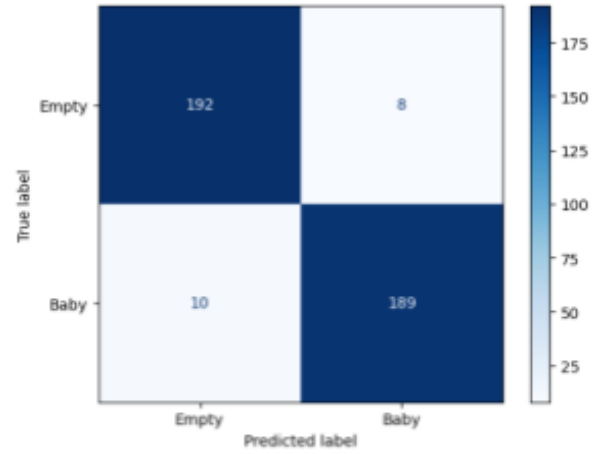


Fig 4.24 Confusion Matrix of Model with parameters as per Table VI and testing data = "PSDfromWindowedADC-Part9.csv"

TABLE VIII. CLASSIFIER METRICS OF FIG 4.23

Classifier Metric	Value
Accuracy	0.897243
Precision	0.865741
Recall	0.939698
F1 score	0.901205

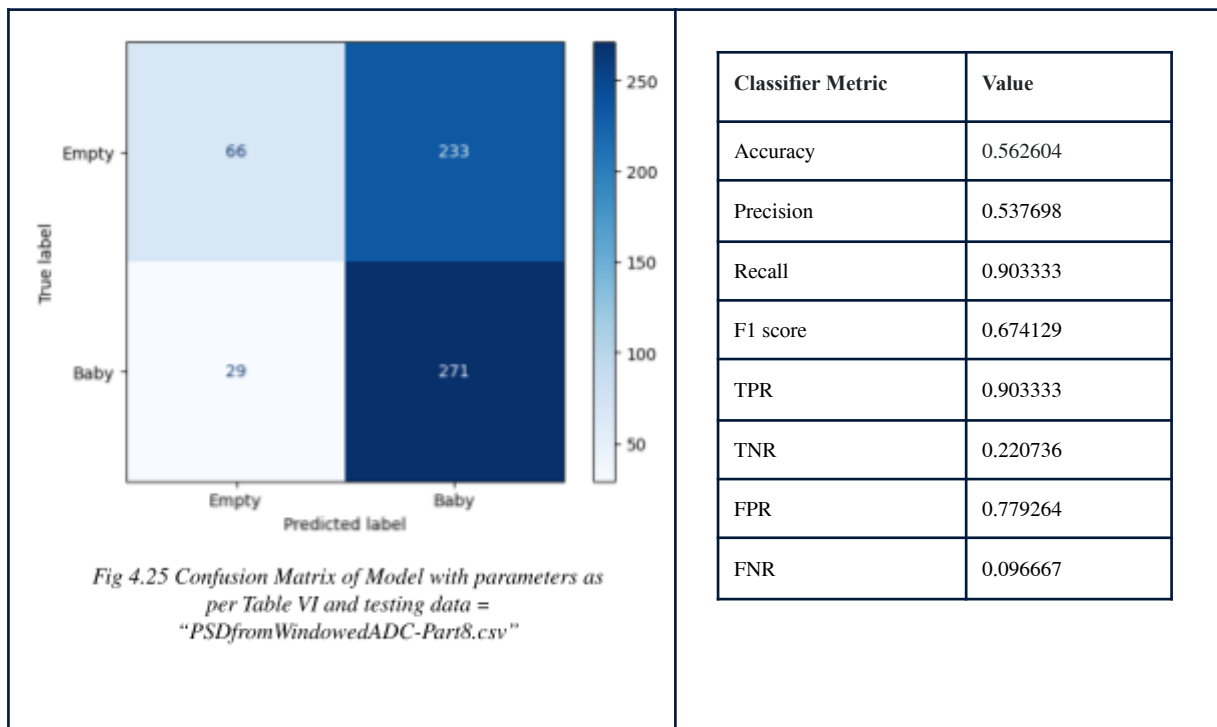
TABLE IX. CLASSIFIER METRICS OF FIG 4.24

Classifier Metric	Value
Accuracy	0.954887
Precision	0.959391
Recall	0.949749
F1 score	0.954545

TPR	0.939698	TPR	0.949749
TNR	0.855000	TNR	0.960000
FPR	0.145000	FPR	0.040000
FNR	0.060302	FNR	0.050251

Another set of test data was also supplied to the same model as described in Table VI. The test data set file is named "PSDfromWindowedADC-Part8" and this pertained to just normal measurements (with the sunshade properly tucked in). But unfortunately the performance on this test data set was poor. The resulting confusion matrix is displayed in Fig 4.25 and the corresponding classification metrics are listed in Table X. It can be seen that the classifier was too biased towards classifying a sample with the "baby" label. This bias needs to be corrected in future versions of the model.

TABLE X. CLASSIFIER METRICS OF FIG 4.25



5 Summary and Perspectives

Various techniques to improve the robustness of infant detection were seen. Some of them like the model using the PSD data as input features have the potential to become robust classifiers as more and more diverse training data is fed to this model. In the current setup with lower than desired amount of training data, the choice on the size of each layer in MLP was limited. Only a simple architecture with a limited amount of neurons could produce better results. Otherwise, there is a case of severe overfitting as seen with results of more complex models. Therefore, the next natural idea is to be able to employ more complex architectures to increase robustness of classification. But to do that, more training data needs to be collected. This would be a very safe and guaranteed way of optimising the classification model.

But there are still several other techniques that can be tried in the future for the same use case. A combination of features can be derived from various representations. For example, a set of features from ACF data, a set of features from PSD data and another subset of features from FFT of the signal can all be fed to one giant classifier model. The idea to build such a model is presented in more detail in [35], as already introduced in Section 2.4.6. Also the robustness of the model was checked by feeding data with the sunshade pulled out in the infant carrier seat. The sunshade in the infant carrier seat used in this experiment was less stiff and lesser rigidity allowed less ultrasonic reflection, as per Theory in Section 2.2. As a result, more energy penetrated the sunshade to reach the parts behind. But the same experiment can be also carried with more rigid or stiff sunshades to see how the classification is in this case. This is again a case for testing the robustness of the model. Also it gives a chance to see if any other features provide a distinction in this case.

Another type of Signal peak analysis would be extracting the locations and values of the top 20 or 30 most dominant peaks in one of the three (ACF, PSD, FFT) signals and feeding these limited features as input to a classifier. Additionally, the distance between these peaks could also be used as a feature after testing for the feature importance between them. This could also provide interesting results. On a side note, a very small improvement would be augmenting the data by zero padding to increase the spectral resolution (i.e resolution in the frequency domain).

This work did not consider omitting the subset of least important features from training altogether. This decision was taken as the model in its current form is pretty simple and only 200 features were fed as input. But when the number of features to be fed increases, due to the previous idea of combining features from various representations (eg : combine peaks from ADC, FFT, PSD), then dimensionality reduction becomes a necessity.

Another way to improve the quality of the training data would be to check for collinearity among the various feature columns. This could be done either with the help of a feature importance heatmap correlation matrix. If there are collinear features, then these need to be deduplicated or removed. Once that is done, all possible models can be tested with the help of 'Pycaret' python package. Distributed cloud computing can also help in doing a whole grid search for hyperparameter optimization of the model.

Another deep study in its own domain can be made on analysing the performance of the model by storing the weights generated for the Neural Network weights in an array for each layer and applying it to a sample input. This lets one view the intermediate output the model sees after each layer and can perhaps help in deciding if the important features are picked up. This is most useful with CNN networks, where the inputs and outputs are in the form of 2D Images.

So far, the suggestions and discussions pertained to the machine learning model and signal processing techniques. Another aspect to be considered is the expected behaviour of the ultrasound wave reflections

themselves. As seen already, the temperature plays a huge role in deciding the speed of sound. So in future measurements, perhaps the datasets need to be labelled also with the measured temperature, when measurements from summer and winter need to be combined and fed as training data to a classifier. Also their characteristics can be studied by passing the ADC signal to a Low-Pass Filter (LPF) and then comparing the resulting envelopes. If the summer and winter measurements are just delayed versions of the critical envelope shapes between the 4500-9500 sample range, then the model in its current state should be able to classify well. But on the other hand, if there is a drift found in the feature importances between the two, then the model needs to be trained appropriately. Although the car being used for measurements currently stays in the garage, where the influence of external temperature is a little less, nevertheless these variations need to be recorded and the behaviour studied.

Passenger detection systems cannot be overlooked in the future. They certainly not only help the private automobile space, but also help modernise the public transport systems and for military purposes as well. With this growing importance, there will be growing interest in the domain to come up with cost effective, reliable solutions. The combination of Machine Learning models with Ultrasonic Sensors is a great tool to forge further research.

6 Acknowledgement

I sincerely thank Prof. Pech for guiding me on this project, with relevant clarifications whenever needed. I thank you for your patience and feedback. I also appreciate you making time for project discussions. I also thank Mr. Umansky for his support and help with the sensor kits, operational questions and topics.

7 Abbreviations

A

ACF	Autocorrelation Function
ADAC	Advanced Driver Assistance Systems
ADC	Analog-to-Digital Conversion
ANN	Artificial Neural Networks

B

BP	Back-Propagation
----	------------------

C

CNN	Convolutional Neural Networks
CSV	Comma-separated values

D

DFT	Discrete Fourier Transform
DL	Deep Learning

E

ECG	Electrocardiogram signal
EMG	Electromyogram signal
EEG	Electroencephalography signal

F

FFT	Fast Fourier Transform
-----	------------------------

G

GUI Graphical User Interface

H

HOV High Occupancy Vehicle

Hz Hertz

I

IR Infra-red

K

Khz KiloHertz

L

LED Light Emitting Diode

LPF Low Pass Filter

M

ML Machine Learning

MLP MultiLayer Perceptron

mW milliWatt

MHz MegaHertz

N

NHTSA National Highway Traffic Safety Administration

O

OCS Occupant Classification System

P

PSD	Power Spectral Density
ppm	parts per million

T

TOF	Time of Flight
TSV	Tab-separated values

8 References

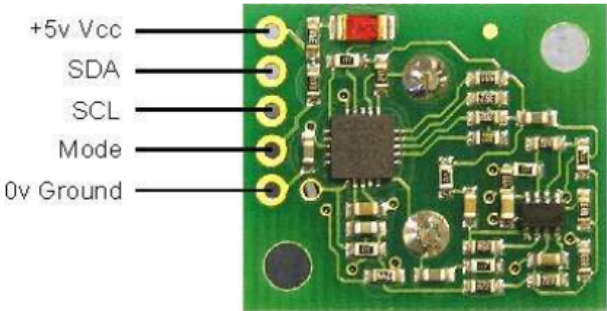
1. European Commission (2023, February 21). Road safety in the EU: fatalities below pre-pandemic levels but progress remains too slow [Press Release]. https://ec.europa.eu/commission/presscorner/detail/en/ip_23_953
2. National Highway Traffic Safety Administration, "Data Collection Study: Deaths and Injuries resulting from certain Non-Traffic and Non-Crash Events", May 2004
3. Chen Y, Luo Y, Ma J, Qi A, Huang R, De Paulis F, Qi Y. Non-Contact In-Vehicle Occupant Monitoring System Based on Point Clouds from FMCW Radar. *Technologies*. 2023; 11(2):39. <https://doi.org/10.3390/technologies11020039>
4. Kuchár P, Pirník R, Janota A, Malobický B, Kubík J, Šišmišová D. Passenger Occupancy Estimation in Vehicles: A Review of Current Methods and Research Challenges. *Sustainability*. 2023; 15(2):1332. <https://doi.org/10.3390/su15021332>
5. Nussbaumer, Henri J, 1982, Fast fourier transform and convolution algorithms (pp. 85-91)
6. Numpy Version 1.25 Manual, "Discrete Fourier Transform (numpy.fft)" <https://numpy.org/doc/stable/reference/routines.fft.html>
7. M. M. Richter, S. Paul, V. Kėpuska and M. Silaghi, Signal Processing and Machine Learning With Applications, Springer, 2022
8. Moradi, Shahpoor & Trad, Daniel & Innanen, Kristopher. (December 2019) , Canadian Journal of Exploration Geophysics, Vol 44, No.1 "When quantum computers arrive"
9. Bierig, S. & Jones, Anne. (2009). Accuracy and Cost Comparison of Ultrasound Versus Alternative Imaging Modalities, Including CT, MR, PET, and Angiography. *Journal of Diagnostic Medical Sonography*. 25. 138-144. 10.1177/8756479309336240
10. Bohn, D.A. (1988). Environmental Effects on the Speed of Sound. *Journal of The Audio Engineering Society*, 36, 223-231.
11. A. K. Sahoo and S. K. Udgata, "A Novel ANN-Based Adaptive Ultrasonic Measurement System for Accurate Water Level Monitoring," in *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 6, pp. 3359-3369, June 2020, doi: 10.1109/TIM.2019.2939932.
12. Garrett, S. L. Attenuation of Sound. In *Understanding Acoustics: An Experimentalist's View of Sound and Vibration*; Springer International Publishing: Cham, 2020; pp. 673–698.
13. A. Bonyár, A. Géczy, G. Harsanyi and P. Hanák, "Passenger Detection and Counting Inside Vehicles For eCall- a Review on Current Possibilities," 2018 IEEE 24th International Symposium for Design and Technology in Electronic Packaging (SIITME), Iasi, 2018, pp. 221-225. doi: 10.1109/SIITME.2018.8599285
14. A. Géczy, R. De Jorge Melgar, A. Bonyár and G. Harsányi, "Passenger detection in cars with small form-factor IR sensors (Grid-eye)," 2020 IEEE 8th Electronics

- System-Integration Technology Conference (ESTC), Tønsberg, Norway, 2020, pp. 1-6, doi: 10.1109/ESTC48849.2020.9229693.
15. Mubina Toa, Akeem Whitehead, "Ultrasonic Sensing Basics" , SLAA907D – Revised, December 2021, Texas Instruments
 16. Siddharth Mukherji, Encyclopedia of General Science for General Competitions, March 2021, Arihant Publications India limited
 17. Berg, R. E. (2023, June 23). sound. Encyclopedia Britannica. <https://www.britannica.com/science/sound-physics/Impedance> (accessed 24 June 2023)
 18. A. R. Selfridge, "Approximate Material Properties in Isotropic Materials" in IEEE Transactions on Sonics and Ultrasonics, vol. 32, no. 3, pp. 381-394, May 1985, doi: 10.1109/T-SU.1985.31608.
 19. H. Azhari, Basics of Biomedical Ultrasound for Engineers, Wiley-IEEE Press, Appendix A: Typical Acoustic Properties of Tissues, pp.313-314, 2010, <https://doi.org/10.1002/9780470561478.app1>
 20. S. A. Schelkunoff, "The impedance concept and its application to problems of reflection, refraction, shielding and power absorption," in The Bell System Technical Journal, vol. 17, no. 1, pp. 17-48, Jan. 1938, doi: 10.1002/j.1538-7305.1938.tb00774.x.
 21. Martínez, Milagros & Benet Gilabert, Ginés & Blanes, Francisco & Perez, Pepita & Simo, Jose. (2003). Using the Amplitude of Ultrasonic Echoes to Classify Detected Objects in a Scene. Proc. of the 11th International Conference on Advanced Robotics.
 22. Donald P. Massa, (2018). Understanding How Frequency, Beam Patterns of Transducers, and Reflection Characteristics of Targets Affect the Performance of Ultrasonic Sensors, Massa Products Corporation <https://www.massa.com/wp-content/uploads/2018/06/Massa-Whitepaper-3-DPM-160621.pdf> (accessed 13 June 2023)
 23. Hernández, Mario. (2016). A tutorial to extract the pitch in speech signals using autocorrelation. Open Journal of Technology & Engineering Disciplines (OJTED). 2. 1-11.
 24. Andreas Pech, 2021, Chapter 4 : Random Processes, Lecture Notes, Stochastic Signals and Systems, Frankfurt University Of Applied Sciences.
 25. Lyons, R.G., (2011). Understanding Digital Signal Processing (3rd ed.), Prentice Hall
 26. A. V. Oppenheim and R. W. Schaffer, Discrete-Time Signal Processing. Pearson, 2010.
 27. STEMLab 125-14 Red Pitaya Processor Board ,Technical Specification & Datasheet, <https://go.redpitaya.com/comparison-table> (accessed 17 May 2023)
 28. Saman Razavi, (2021) Deep learning, explained: Fundamentals, explainability, and bridgeability to process-based modelling, Environmental Modelling & Software, Volume 144, 105159, ISSN 1364-8152, <https://doi.org/10.1016/j.envsoft.2021.105159>.
 29. Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). <https://doi.org/10.1186/s40537-021-00444-8>
 30. Ultrasonic Range Finder, SRF02, Technical Specification & Datasheet, Devantech Ltd. <http://www.robot-electronics.co.uk/html/srf02tech.htm> (accessed 1 March 2023)
 31. Child Car Seat Safety, Courtney Medical Group (July, 2018) <https://courtneymedicalgroupaz.com/2018/07/23/car-seat-safety/>

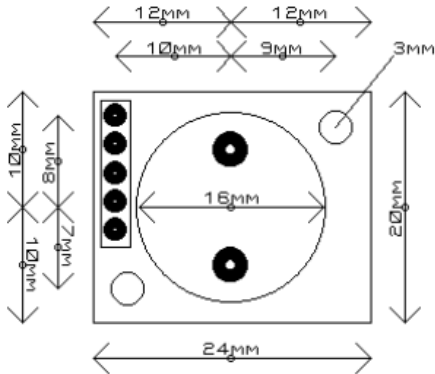
32. A. H. Pech, P. M. Nauth and R. Michalik, "A new Approach for Pedestrian Detection in Vehicles by Ultrasonic Signal Analysis," IEEE EUROCON 2019 -18th International Conference on Smart Technologies, 2019, pp. 1-5, doi: 10.1109/EUROCON.2019.8861933.
33. P. M. Nauth, A. H. Pech and R. Michalik, "Research on a new Smart Pedestrian Detection Sensor for Vehicles," 2019 IEEE Sensors Applications Symposium (SAS), 2019, pp. 1-5, doi: 10.1109/SAS.2019.8705978.
34. Phillip McKerrow, Kok Kai Yoong, Classifying still faces with ultrasonic sensing, Robotics and Autonomous Systems, Volume 55, Issue 9, 2007, Pages 702-710, ISSN 0921-8890, <https://doi.org/10.1016/j.robot.2007.05.006>
35. Ahmet Taspinar, Machine Learning with Signal Processing Techniques (April, 2018), ML Fundamentals Web blog post <https://ataspinar.com/2018/04/04/machine-learning-with-signal-processing-techniques/> (accessed 10 May 2023)
36. Haritha G, MLPModelPythonKeras, (2023), GitHub repository, <https://github.com/harytha6/MLPModelPythonKeras>

9 Appendix

Appendix 8.1 Pin Layout of Ultrasonic SRF02 sensor - Source [30]



Appendix 8.2 Physical Dimensions of Ultrasonic SRF02 sensor - Source [30]

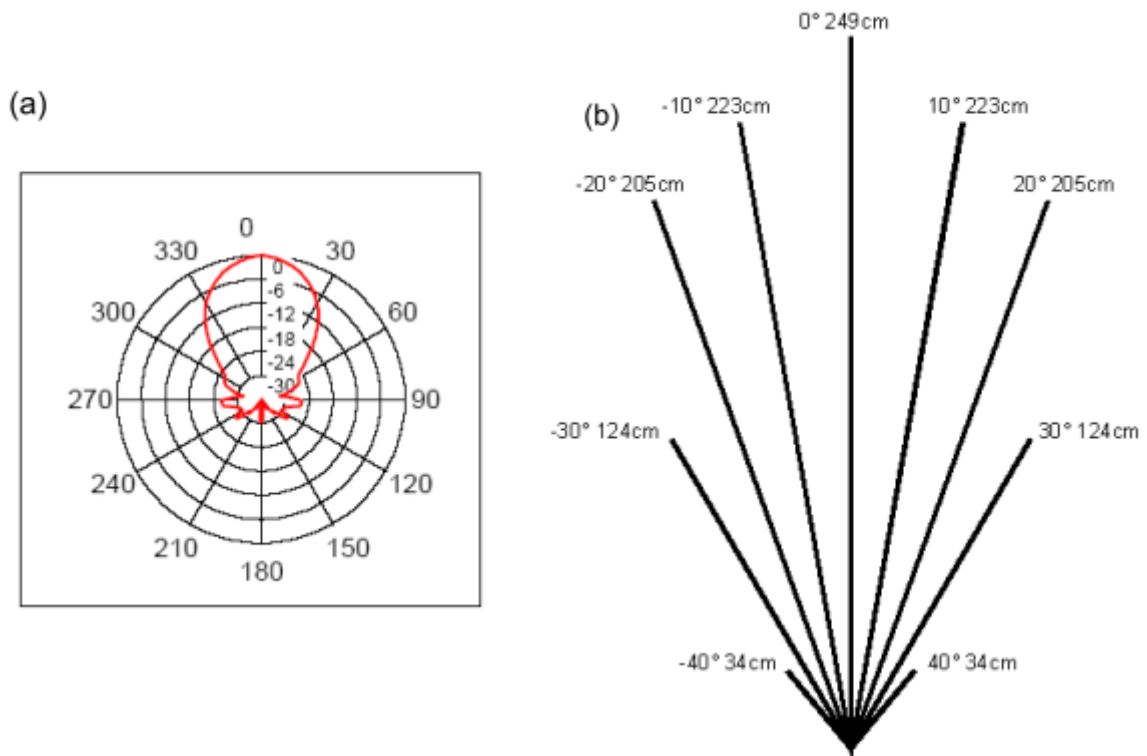


Appendix 8.3 Technical Specifications of Red Pitaya STEMLab 125-14 - Source [27]

Properties	STEMLab 125-14
Basic	
Processor	Dual-Core ARM Cortex-A9 MPCore
FPGA	Xilinx Zynq 7010
RAM	512MB (4Gb)
System memory	Micro SD up to 32GB
Connectivity	
Ethernet	1 Gbit

USB	USB 2.0
WIFI	using Wi-Fi dongle
RF inputs	
Channels	2
Sample rate	125MS/s
ADC resolution	14 bit
Full-scale voltage range	$\pm 1V / \pm 20V$
Input coupling	DC
Bandwith	DC-60MHz
Input impedance	1M Ω
RF outputs	
Channels	2
Sample rate	125MS/s
ADC resolution	14 bit
Full-scale voltage range	$\pm 1V$
Load impedance	50 Ω
Shortcut protection	Yes
Typical raising/falling time	2V / 10ns
Bandwith	DC - 60MHz
Extension connector	
Digital IOs	16
Analog inputs	4 channels 0-3.5V 12bit
Analog outputs	4 channels 0-1.8V 12bit
Communication interfaces	I2C, UART, SPI
Available voltages	+5V, +3.3V, -4V
Synchronization	
Trigger input	through extension connector
Daisy chain connection	over SATA connection
Ref. clock input	N/A

Appendix 8.4 Beam Pattern of Ultrasonic SRF02 sensor - Source [30]

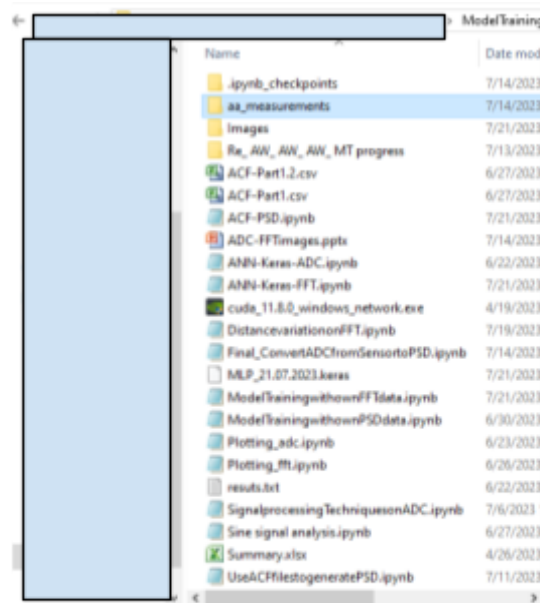


- (a) Manufacturers' Beam pattern, showing the sensitivity of the transducer in db.
- (b) Measured beam pattern for the SRF02, showing the maximum detection range of a 55mm diameter plastic pipe.

Appendix 8.5 Names of files and terminology used to describe various measurement setups

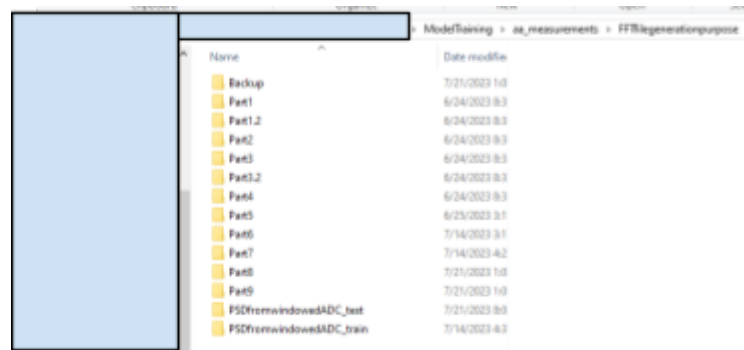
Storage of TSV Output text files	
ADC Measurements with dummy baby present in infant carrier seat	File named starting with "adc_baby....."
ADC Measurements without dummy baby in infant carrier seat	File named starting with "adc_empty....."
Leaning Variation in ADC Measurement	leaned_least, leaned_middle, leaned_back/morelieddownposition - are different suffixes added to the file names to indicate the varying levels of inclination of the infant carrier seat.
Important Python Notebooks	
ANN-Keras-PSD	MLP Model building and training
ACF-PSD	To generate PSD for new ADC data
Important Links	
https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/ - Visit the link to learn to generate other classification metrics like AUC-ROC,.....	

Appendix 8.6 Folder Structure



Name	Date modified
.ipynb_checkpoints	7/14/2023
aa_measurements	7/14/2023
images	7/21/2023
Re_AW_AW_AW_MT progress	7/13/2023
ACF-Part1.2.csv	6/27/2023
ACF-Part1.csv	6/27/2023
ACF-PSD.ipynb	7/21/2023
ADC-FFImages.pptx	7/14/2023
ANN-Keras-ADC.ipynb	6/22/2023
ANN-Keras-FFT.ipynb	7/21/2023
cuda_11.8.0_windows_network.exe	4/19/2023
DistancevariationonFFT.ipynb	7/19/2023
Final_ConvertADCfromSensortoPSD.ipynb	7/14/2023
MLP_21.07.2023.keras	7/21/2023
ModelTrainingwithoutFFTdata.ipynb	7/21/2023
ModelTrainingwithoutPSDdata.ipynb	6/30/2023
Plotting_adc.ipynb	6/23/2023
Plotting_fft.ipynb	6/26/2023
results.txt	6/22/2023
SignalprocessingTechniquesonADC.ipynb	7/6/2023
Sine signal analysis.ipynb	6/27/2023
Summary.xlsx	4/26/2023
UseACFfiletogeneratePSD.ipynb	7/11/2023

Home directory for running the python notebooks



Name	Date modified
Backup	7/21/2023 1:0
Part1	6/24/2023 9:3
Part1.2	6/24/2023 9:3
Part2	6/24/2023 9:3
Part3	6/24/2023 9:3
Part3.2	6/24/2023 9:3
Part4	6/24/2023 9:3
Part5	6/25/2023 3:1
Part6	7/14/2023 3:1
Part7	7/14/2023 4:2
Part8	7/21/2023 1:0
Part9	7/21/2023 1:0
PSDfromwindowedADC_test	7/21/2023 9:0
PSDfromwindowedADC_train	7/14/2023 4:3

Measurements folder with ADC Data stored in Part1, Part2.... Part9 folders and PSD data in test and train folders below that is used for training the MLP model

Appendix 8.7 Important Usernames & Passwords for Various Components

Component	Name	Password
RedPitaya Wifi access point	Red Pitaya AP 19	redpitaya

Appendix 8.8 Classification Python code with Keras MLP Model and to generate classification metrics.

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout,
GlobalAveragePooling1D
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor

gpus = tf.config.list_physical_devices('GPU')
#tf.config.experimental.set_virtual_device_configuration(gpus[0],
[tf.config.experimental.VirtualDeviceConfiguration(memory_limit=2048)])

# In[2]:

#Reading input data for training and testing

# Getting current working directory and storing it as a string variable and
reading all empty seat readings into one dataframe

directory_in_str =
```

```
"E:\\Haritha\\MasterThesis\\ModelTraining\\aa_measurements\\FFTfilegenerationpur  
pose\\PSDfromwindowedADC_train"  
Emptyfilelist = []  
empty = pd.DataFrame()  
print("Adding the following files : ")  
for file in os.listdir(directory_in_str) :  
    if file.startswith("PSD") :  
        print(file)  
        filepathempty = directory_in_str + "\\\" + file  
        df1 = pd.read_csv(filepathempty, engine = 'python')  
        empty = empty.append(df1)  
    else :  
        continue  
  
# Empty seat Data ingest and attach Label = 0  
empty.reset_index(inplace = True)  
  
#Index given is 16, as iloc uses n-1 as the end limit for 0:16  
empty.drop(empty.iloc[:,0:16] ,axis=1,inplace=True)  
empty["Label"] = 0  
empty = empty.T  
empty.head()  
  
# In[3]:  
  
empty.reset_index(drop=True, inplace = True)  
empty  
  
# In[57]:  
  
empty.iloc[5000,400]  
  
# In[4]:  
  
Fs = 1953125  
N = 32768  
f2 = np.linspace(600 * Fs/(N), 800* Fs/(N), 200)  
  
# In[26]:
```

```
plt.plot(f2, empty.iloc[0:100,0:200], label='PSD,empty')
plt.plot(f2, empty.iloc[16675,0:200], label='PSD,baby')
plt.legend()
plt.xlim([35000,45000])
plt.grid(True)
plt.ylabel("PSD")
plt.xlabel("Frequency (Hz)")
```

```
# In[54]:
```

```
#Plotting empty seats from same measurement batch
for i in range(16) :
    plt.plot(f2,empty.iloc[i,0:200])
#plt.plot(f2, empty.iloc[16675,0:200], label='PSD,baby')
plt.legend()
#plt.xlim([35000,45000])
plt.grid(True)
plt.ylabel("PSD")
plt.xlabel("Frequency (Hz)")
```

```
# In[50]:
```

```
#Plotting empty seats from different measurement batches
plt.plot(f2,empty.iloc[0,0:200])
plt.plot(f2,empty.iloc[100,0:200])
plt.plot(f2,empty.iloc[200,0:200])
plt.plot(f2,empty.iloc[5000,0:200])
plt.plot(f2,empty.iloc[5200,0:200])
plt.plot(f2,empty.iloc[5400,0:200])
plt.plot(f2,empty.iloc[5600,0:200])
#plt.plot(f2, empty.iloc[16675,0:200], label='PSD,baby')
plt.legend()
#plt.xlim([35000,45000])
plt.grid(True)
plt.ylabel("PSD")
plt.xlabel("Frequency (Hz)")
```

```
# In[31]:
```

```
train, test = train_test_split(empty, test_size=0.2, random_state=42,
                                shuffle=True)

# In[32]:

X_train = train.iloc[:,0:200]
#X_train = X_train/1000
y_train = train['Label']
X_test = test.iloc[:,0:200]
#X_test = X_test/1000
y_test = test['Label']

# In[49]:

len(y_test)

# In[50]:

sum(y_test)

# In[33]:

X_train.shape

# In[34]:

X_train_np = X_train.to_numpy()
X_train_np = np.reshape(X_train_np,(X_train.shape[0],X_train.shape[1],1))
X_train_np.shape

# In[35]:

X_test_np = X_test.to_numpy()
```

```
X_test_np = np.reshape(X_test_np,(X_test.shape[0],X_test.shape[1],1))
X_test_np.shape

# In[58]:

# It can be used to reconstruct the model identically.
import keras
reconstructed_model = keras.models.load_model("MLP_21.07.2023.keras")

# In[36]:

input_dim = len(X_train.columns)
input_dim

# In[40]:

neurons = 64
epochs = 150
batch_size = 50

# In[38]:

#Run this step to refresh the layers added to the model. You can rebuild the
layers in the model from the beginning.
model = Sequential()

# In[39]:

model.add(Dense(neurons, input_dim = input_dim, activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.summary()

# In[41]:
```

```
history = model.fit(X_train_np,y_train, epochs=epochs, batch_size=batch_size,
verbose=1, validation_split=0.2)
predictions= model.predict(X_test_np)
```

```
# In[43]:
```

```
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
# In[45]:
```

```
y_pred = (predictions > 0.5) *1
```

```
# In[45]:
```

```
output = pd.DataFrame({'ID' : test.index,
'PreferredScore':predictions.flatten()})
output.to_csv('preferred15032023.csv', index=False)
```

```
# In[74]:
```



```
scores = model.evaluate(X_test_np,y_test, verbose=1)
print(scores)

# In[54]:

scores2 = reconstructed_model.evaluate(X_test_np,y_test, verbose=1)
print(scores2)

# In[48]:

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Empty",
"Baby"])

disp.plot(cmap=plt.cm.Blues)
plt.show()

# In[ ]:

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, y_pred)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, y_pred)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred)
print('F1 score: %f' % f1)

# In[51]:

#Saving the model
model.save("MLP_21.07.2023.keras")
```

```
# In[105]:

#Testing on measurements from another day - 13.04.2023

directory_in_str_new =
"E:\\Haritha\\MasterThesis\\ModelTraining\\aa_measurements\\FFTfilegenerationpur
pose\\PSDfromwindowedADC_test"
Emptyfilelistnew = []
emptynew = pd.DataFrame()
print("Adding the following files : ")
for file in os.listdir(directory_in_str_new) :
    if file.startswith("PSDfromWindowedADC-Part8") :
        print(file)
        filepathemptynew = directory_in_str_new + "\\\" + file
        df1new = pd.read_csv(filepathemptynew, engine = 'python')
        emptynew = emptynew.append(df1new)
    else :
        continue

#emptynew.head()

# In[106]:

test_new, test_leftout = train_test_split(emptynew, test_size=0.001,
random_state=42, shuffle=True)

X_testnew = test_new.iloc[:,0:200]
y_testnew = test_new['Label']

# In[107]:

X_testnew = X_testnew.to_numpy()
X_testnew = np.reshape(X_testnew,(X_testnew.shape[0],X_testnew.shape[1],1))
X_testnew.shape

# In[108]:
```

```
type(X_testnew)

# In[109]:

predictions2 = reconstructed_model.predict(X_testnew)
y_pred2 = (predictions2 > 0.5) *1
scores2 = reconstructed_model.evaluate(X_testnew,y_testnew, verbose=1)
print(scores2)

# In[110]:

cm_unseendata = confusion_matrix(y_testnew, y_pred2)

disp = ConfusionMatrixDisplay(confusion_matrix=cm_unseendata,
display_labels=["Empty", "Baby"])

disp.plot(cmap=plt.cm.Blues)
plt.show()

# In[111]:

TN, FP, FN, TP = cm_unseendata.ravel()
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print('TPR: %f' % TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print('TNR: %f' % TNR)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print('FPR: %f' % FPR)
# False negative rate
FNR = FN/(TP+FN)
print('FNR: %f' % FNR)

# In[112]:

# accuracy: (tp + tn) / (p + n)
```

```
accuracy = accuracy_score(y_testnew, y_pred2)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_testnew, y_pred2)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_testnew, y_pred2)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_testnew, y_pred2)
print('F1 score: %f' % f1)
```

Appendix 8.9 Reading Testing Data from files in separate folder and Data formatting

```
#Testing on measurements from another day

directory_in_str_new =
"E:\\Haritha\\MasterThesis\\ModelTraining\\aa_measurements\\FFTfilegenerationpur
pose\\PSDfromwindowedADC_test"
Emptyfilelistnew = []
emptynew = pd.DataFrame()
print("Adding the following files : ")
for file in os.listdir(directory_in_str_new) :
    if file.startswith("PSDfromWindowedADC-Part8") :
        print(file)
        filepathemptynew = directory_in_str_new + "\\\" + file
        df1new = pd.read_csv(filepathemptynew, engine = 'python')
        emptynew = emptynew.append(df1new)
    else :
        continue

#emptynew.head()

# In[106]:

test_new, test_leftout = train_test_split(emptynew, test_size=0.001,
random_state=42, shuffle=True)
```

```
X_testnew = test_new.iloc[:,0:200]  
y_testnew = test_new['Label']
```

```
# In[107]:
```

```
X_testnew = X_testnew.to_numpy()  
X_testnew = np.reshape(X_testnew,(X_testnew.shape[0],X_testnew.shape[1],1))  
X_testnew.shape
```