Design & Innovation Project (DIP)

Project Report

Smart Touchless control with Millimeter-Wave Radar Sensor and Artificial Intelligence

Project Group: E047

School of Electrical and Electronic Engineering
Academic Year 2021/22
Semester 1

Table of Contents

1. Project Objectives

Technology has been improving over the years and with the coronavirus in this recent pandemic impacting our regular routine, sharing common handles/buttons/touchscreens has created huge inconveniences and unease when everyone is more concerned about hygiene, sanitation and overall more health-conscious. Advancement in touchless technology could be the solution to reduce physical contact and reduce the spread of COVID-19 and other surface-borne pathogens.

Touchscreens and physical human-computer interaction (HCI) devices like buttons have become ubiquitous in everyday life. The key to unlocking this issue could live in recent new advances in touchless technology, incorporating millimetre-wave (mmWave) radar and artificial intelligence (AI). Touchless gesture control is a natural, convenient, and hygienic way of HCI and it is considered the future HCI for the emerging Low Touch Economy. It should also be intuitive so that the end-user finds it second nature to utilise it (eg. swipe their palm to flip a menu page), and will not result in a huge learning curve, preventing widespread adoption.

The ability of the millimetre-wave (mmWave) radar is able to take advantage of the modern day advances in sensor and facts processing technology with suitable AI algorithms for simple and effective hand gesture sensing and control of a flexible HCI with unlimited applications. Furthermore, this technology also makes a suitable substitute due to its robustness in cases where the camera lens may possibly be stained with dirt, causing distortion or occlusion to the image captured. In addition, the mmWave radar is able to penetrate materials such as plastic, drywall and clothing which allows for the radar sensor to be invisibly integrated into applications.

The innovation is completely unique in that it has the ability to determine which interface element the user will choose from hands-free pointing behaviour. The product uses a low cost commercial mmWave radar sensor It is a contactless sensing technique that detects objects and provides information about their range and velocity, which therefore makes it the most common form of touchless technology where individuals able to perform easy gestures to govern or have interaction with gadgets without bodily touching them [1].

Our project utilises gesture-based interface, it is a form of touchless technology. Individuals can perform easy gestures to control or interact with devices without the need to touch them [2]. A gesture is a symbol of physical motion of the body or the hand that relay messages or intent [3]. Furthermore, it is able to be used as a form of communication between computers and humans. It is different from the old-fashioned hardware-based methods and able to accomplish HCI via the gesture-based interface [4]. The gesture-based interface determines the user's aim through the recognition of the gesture or movement of any parts of the body [5].

Ultimate objective for us will be to integrate our gesture recognition solution into an edge device like a Raspberry Pi so that it can be packaged as a whole independent solution without the need for an external laptop or computer. This is simply the beginning of what the generation can achieve. As lockdown regulations round the sector maintain to ease, we trust the generation may be especially beneficial in a post-COVID world [6]. The technology used doesn't need to be screen-primarily based totally either, it may be on the surfaces or objects and we may want to convert any screen (which includes non-contact enabled ones) right into a touchscreen [7].

2. Project Summary

Our group has gone ahead with using radar sensors named Acconeer which is a reference module that is intended for commercial use, evaluation and development of an application that encourages creativity and innovation. Being a high-performance radar sensor module that is user friendly, it gives users the flexibility of developing their own application with radar concepts such as the different radar signal processing techniques, i.e STFT as well as various AI algorithms used.

With all the knowledge and incorporation of various techniques, we will implement all these to be able to run on a Raspberry Pi micro-computer. The project has achieved the goal by using technologies based on radio frequency (RF) for active sensing to obtain a range profile, Doppler, which makes it worthy of the hand-gesture classification aim. Furthermore, the usage of radar structures for gesture sensing drives better robustness towards detrimental surroundings noises [8], along with light conditions, dirt and complicated backgrounds, and maintains an extraordinarily low processing price than different approaches [9].

We are therefore pleased to conclude we have managed to achieve our intended learning outcomes as a group. The project's intended learning outcomes include the following:

- Working with a command-line interface (CLI) and a Linux environment
- Radar signal processing and signal representations
- Python and MATLAB scripting, including popular frameworks like NumPy and TensorFlow
- Machine learning and deep learning
- Working with individuals from different backgrounds and fully utilising each others' strengths

In the end, we are able to achieve our project objectives to a satisfactory level, with a working prototype deployed with the Raspberry Pi 3B+. We discuss the benefits and some limitations of our solution.

3. Scope

Upon project completion, we expect a working machine learning (ML) classifier complete with a radar data processing pipeline such that the prototype can detect and distinguish between pre-set classes of hand gestures. This prototype will also be implemented on a Raspberry Pi to simulate the whole project as a possible HCI or potentially an Internet of Things (IoT) device which has many applications.

Stage	*Description*
1	a) Set up our devices and familiarise usage with them, including both Acconeer radar sensors and Raspberry Pis. b) Familiarise ourselves with radar signal processing, radar signal representations, Python, Raspberry Pi usage and conduct some literature review.
2	a) Parameter and gesture exploration on Acconeer radar sensors. Collect sample radar sensor data in IQ mode for further analysis. b) Explore multiple feature extractors, namely magnitude, and STFT. Ensure that the feature plots match expectations so that we may use them later. c) Use an external dataset to begin work on the machine learning pipeline.
3	a) Determine the gestures to be collected and record radar sensor data that captures the movements frame-by-frame, including the slices of the depth range specified, then export it as a .h5 file. b) Process .h5 files using Python using our Gesture Extractor GUI to form a large dataset made up of raw data store in .npz file format.
4	a) Perform STFT on radar data and export an image-based signature based on the hand gesture samples recorded earlier. b) Feed the signature file into a Machine Learning (ML/AI) Model to train it and perform classification to effectively classify several hand gestures in both seen and unseen cases. Different machine learning techniques or classifiers will be experimented with to yield the best results for the application.
5	Implement the prediction pipeline in real-time. This requires interfacing with the Acconeer radar sensor itself by using an SDK and calling our own API functions which we have finalised in earlier stages.
6	Implement on an edge device with limited computational abilities e.g. Raspberry Pi and integrate into a workable product with applications such as volume control, switching on/off etc.
7	Wrap up and finalise the project, including documentation.

Table 1: Outline of our project stages

Table 1 details the various stages in the project that we completed, with some stages being executed concurrently in different subgroups to best utilise our time in meeting the tight deadline. We also detail our project pipeline for training the ML model in Figure 1. For ease of use, we have encapsulated each block into our own usable API, which may be used in the future - eg. during real-time predictions. This will help streamline the individual components to build up towards the final script which requires all the building blocks.

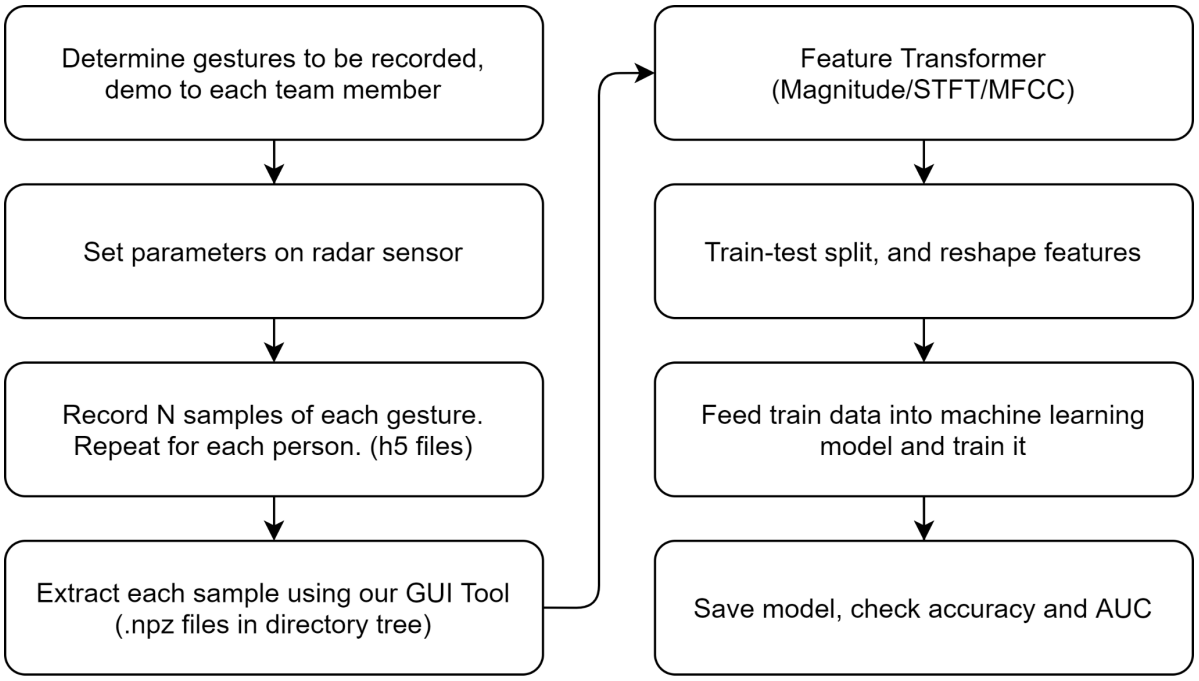


Figure 1: Our project pipeline for training the ML model

Some additions to our scope include (i) exploring another feature extractor called Mel frequency cepstral coefficients (MFCC), (ii) utilising deep learning for our application which is consistent with many recent papers, (iii) creating graphical user interfaces to automate repetitive tasks and showcase our project in an intuitive manner. These additions were included to make our project deliverable more robust and user-friendly, so as to use the resources that we have to the best ability that we are able to.

4. Schedule

Phase	*Planned Milestone Date*	*Actual Milestone Date*
Initiating Phase	Week 1	Week 1-2
Planning Phase	Week 2-3	Week 2-3
Execution Phase	Week 4-10	Week 3-10
Closing Phase	Week 11-12	Week 11-12
Project End Date	Week 13	Week 13

Table 2: Milestone dates for each phase

We largely adhered to our planned milestone dates in Table 2 and managed to achieve our weekly goals. Careful planning had to be done to ensure that each week had small deliberate goals so that our group's vision is always towards the end goal in mind. There were reasonable moments in which the group's activity was decreased due to quizzes or assignment deadlines. However, we always made up for the 'missed' progress in the following week. In the end, we managed to complete the project with its relevant documentation before the targeted deadline of Week 13.

5. Costs and Component List

PHASE	*Planned Costs (S$)*	*Actual Costs (S$)*
Initiating Phase	-	-
Planning Phase	660	654.25
Execution Phase	20	9
Closing Phase	20	-
Project Total Costs	700	663.25

Table 3: Budget list for the project

In the initiating phase, we were only given $1000 for the whole project, so budgeting will be crucial to ensure that our project will have enough money for the devices needed. We planned our expenses before the start of the project to avoid over-expenses. However, since we haven't had a clear solution during the initiating phase, the only thing we can do is to do some research on the costs of different kinds of millimetre-wave radar sensors and Raspberry Pis in Singapore.

In the planning phase, the project solution has been developed in detail to meet the project's objective. The estimated budget for the equipment and material costs is listed in the graph above which is estimated to be around $650. We planned to purchase 2 different models of radar sensors and Raspberry Pi to have a comparison of the accuracy for hand gesture recognition in our project.

As we have a limitation of $1000 for the whole project, the budget list was given to the project supervisor to get approval before we purchased the order. The budget is used to monitor and control cost expenditures during the execution phase.

Component(s)	*Qty*
Acconeer High Performance Module: XM112, XB112	1 set
Acconeer IoT Module: XM122, XB122, XA122	1 set
Raspberry Pi Model 3B+; 5.1V/2.5A PSU Micro-USB B Universal Plug	1 set
Raspberry Pi Model 4B 8GB RAM; 5V/3A PSU USB C UK Plug	1 set
64 GB microSD card	2
Micro-USB Fast Charging Cable	3

Table 4: Component list

We placed the orders for the Acconeer Radar Sensor module, XB112 and XB122, Raspberry Pi 3 Model B+, Raspberry 4 Model B 8 GB RAM and 64 GB micro-SD card after getting approval from our supervisor. We then found out that there is a need to buy some cables and wires for the devices that are not included in the budget list during the project implementation phase. Besides, we also forgot to estimate the shipping cost of the purchase platforms. Fortunately, we have already allocated some extra budget for small items like these so we still remained within our budget.

Table 4 shows the final component list that was used in our project. 2 sets of Acconeer sensors and Raspberry Pi with different specifications were purchased so that we may make comparisons of their performances.

6. Methodology

This section discusses our methodologies and intermediate experimental results before achieving our final project goal of running a real-time gesture predictor on a Raspberry Pi with satisfactory accuracy. We describe our project sections in detail and conclude certain sets of parameters or methods which led to the success of our project.

6.1 Literature Review

Touchless era is a department of gesture manipulation era that specializes in organising manipulation among computer systems and customers without the want for any shape of touch or bodily input [2]. These are, in turn, transmitted to the computer as valid commands or prompts. Tables 5 and 6 spotlight the benefits of radar over different gesture popularity methods, namely camera and infrared-based ones.

Limitations	*Description*
Cameras can infringe on users' privacy	Users often do not feel comfortable being watched by a camera. Radar gives an opportunity to such visible surveillance that may be a higher suit for privacy-risking user related devices.
Illumination variation	The precision of pores and skin shade segmentation strategies is commonly encouraged via means of illumination variation while radar is less prone to lightning conditions.
Background complexity	Cameras based only support static gestures, with a suitable department of skin-shaded items (e.g., hands, face) towards a complicated background. while radar does not care about the shade of the items.
Poor vision under extreme weather events	If the environment that the camera is in is poor, it may not detect anything hence the greatest advantage of radar is that the transmission of radio waves isn't suffering from visibility and lighting Therefore, radar overall performance is steady throughout all environmental conditions.

Table 5: Camera vs Radar [13]

However, using deep learning usually requires a lot of data so it might be unattainable given the limited timeframe that we have. As an example, the train dataset used in [12] contained 368,000 gesture recordings. We shall see later that we did in fact explore deep learning and found the results to be on par if not more robust than traditional ML classifiers.

For radar signals, the authors in [10] detail various hand-gesture radar signal representations. An example in [11] uses an STFT based spectrogram to extract micro-doppler signatures. We also found that most recent literature employs deep learning techniques for gesture recognition. Complex architectures like LSTM and Convolution layers were used in [12] to form RadarNet to achieve this goal.

Limitations	*Description*
Unreliable and potentially endangered in harsh weather conditions	PIR sensor video display units alternate withinside the infrared radiation of an item. This way that if the item is stationary, PIR sensors cannot hit upon the immobile item. Environmental elements additionally have to be considered. If the sensor is meant for an outside application, it is able to degrade the general overall performance of the PIR sensor.
A well-known limitation, heat	At an ambient temperature of approximately 36 or 37 degrees (° C), the body normally records 36.7 °C, so no human movement is detected.

Table 6: Infrared vs Radar [14]

6.2 Acconeer Radar Sensors

6.2.1 Introduction

The Acconeer radar sensor is a mmWave pulsed coherent radar that transmits radio waves in short pulses. It can perform a high precision estimation measurement with a low power consumption which is less than 1 milliwatt. Additionally, the Acconeer radar sensor has the ability to monitor fast and small movements due to the fast pulsating of the system. Another advantage of the pulse coherent radar is that it can be hidden behind plastic or glass materials and hence do not need an open space, which is optimized for real-life application.

We have purchased Acconeer radar sensors because of their precise and robust distance and depth measurements. Radar can be found in many different applications like cars and aeroplanes etc. However, most radar systems available are huge, have heavy power consumption and are expensive, therefore the Acconeer radar sensor is manufactured in a way to overcome these drawbacks. Furthermore, it is coherent that it is able to transmit signals in a stable time and phase reference to as small as picosecond scale, which allows for high accuracy measurements.

6.2.2 Setting up the Sensors

Setting up the Acconeer radar module required a USB virtual COM port to allow the USB interface to communicate with peripherals of the Acconeer radar module. Without a USB virtual COM port flashing of the module will be challenging. Hence additional installation of Virtual COM port driver is installed on the computer for communication to be established. USB1 is connected and used as a power supply, flashing and communicating port as shown in Figure 2. On the other hand, USB2 on the XB112 Sensor Module is not used because it is for data reading.

Before receiving sensor outputs and using the Acconeer module, we have to flash software onto the sensor modules [15]. Flashing is to boot the Acconeer radar module and update the software of the Acconeer radar module. For our purposes, BOSSA software was used for the flashing procedure as

it is a flash programming utility tool. After flashing is completed, the module is now able to be used to communicate with the model. Secondly, installation of the Acconeer Python Exploration Tool is required because of the Python library packages to start up the Acconeer GUI built upon Python. After the successful installation, Acconeer GUI can be operated to execute and run the radar module.

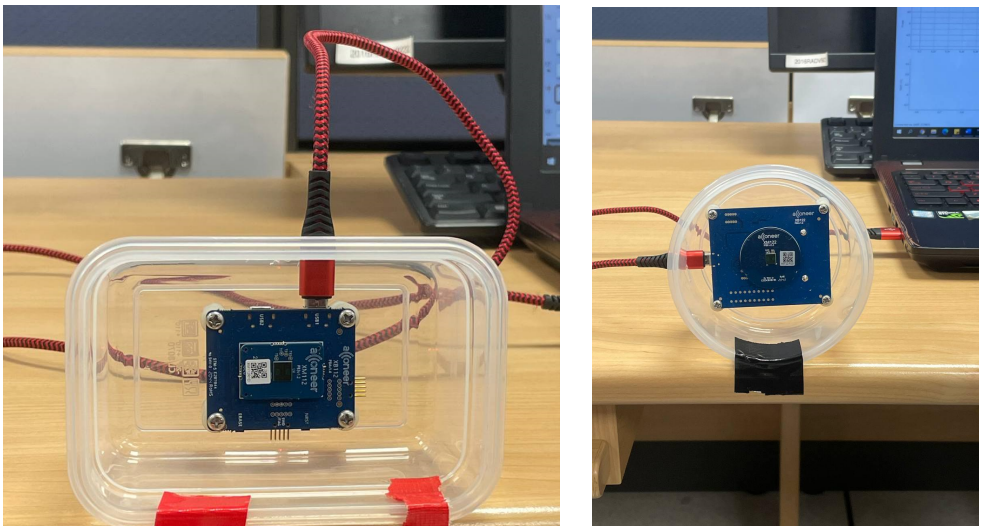


Figure 2: (i) XB112 sensor (ii) XB122 sensor

6.2.3 Parameter Exploration

The Acconeer Exploration Tool GUI is a comprehensive python-based graphical user interface (GUI) provided by Acconeer themselves to showcase their radar sensor products [16]. A screenshot of the GUI is shown in Figure 3. The tool is used in the initial stages to help us in testing and evaluating the sensor for our specific use case. The flexibility in changing certain key parameters like pulse duration, sampling frequency, and distance interval makes it our go-to tool of choice during this parameter exploration phase.

Firstly, it is necessary to decide what type of service to use. There are three options for the Acconeer radar: envelope, IQ, and sparse. The IQ service provides data in the closest form to raw data with both phase, amplitude and depth information. Furthermore, it utilises the Acconeer's pulse radar phase coherence produces stable in-phase and quadrature (IQ) components that can detect subtle movements in the target scene. When analyzing the stationary measurements, the IQ service is used to inspect the phase, amplitude and depth information of the different gestures displayed.

We opted for the IQ service to obtain complex IQ radar data, which allows for the detection of small gesture measurements. This allows us to utilise the full capabilities of the radar sensor hardware that we have since we have no computational limitations at hand.

The length of the radar pulse, which is determined by the profile setting, is a trade-off between depth resolution and SNR. It is highly undesirable to have direct leakage affecting the measurement. Direct leakage is when a portion of the transmitted radar pulse travels directly to the receiving antenna without reflecting off of an object. Furthermore, for lower profile numbers, the

reflectors when measuring depth will be stronger to avoid saturation of the received signal and the range of operation will be lower vice versa. On the other hand, for higher profile numbers, the SNR will be higher. Since the radar is located 20 cm to 50 cm away from the hand that gesture is performed and where the balance of SNR and depth resolution is acceptable, Profile 2 will be the optimal profile for the gesture measurement. Profile 1 is not chosen as the optimal profile because of the operating distance (< 20 cm) and higher profiles are not chosen because of the weak reflectors for gesture recognition. Hence, Profile 2 is selected.

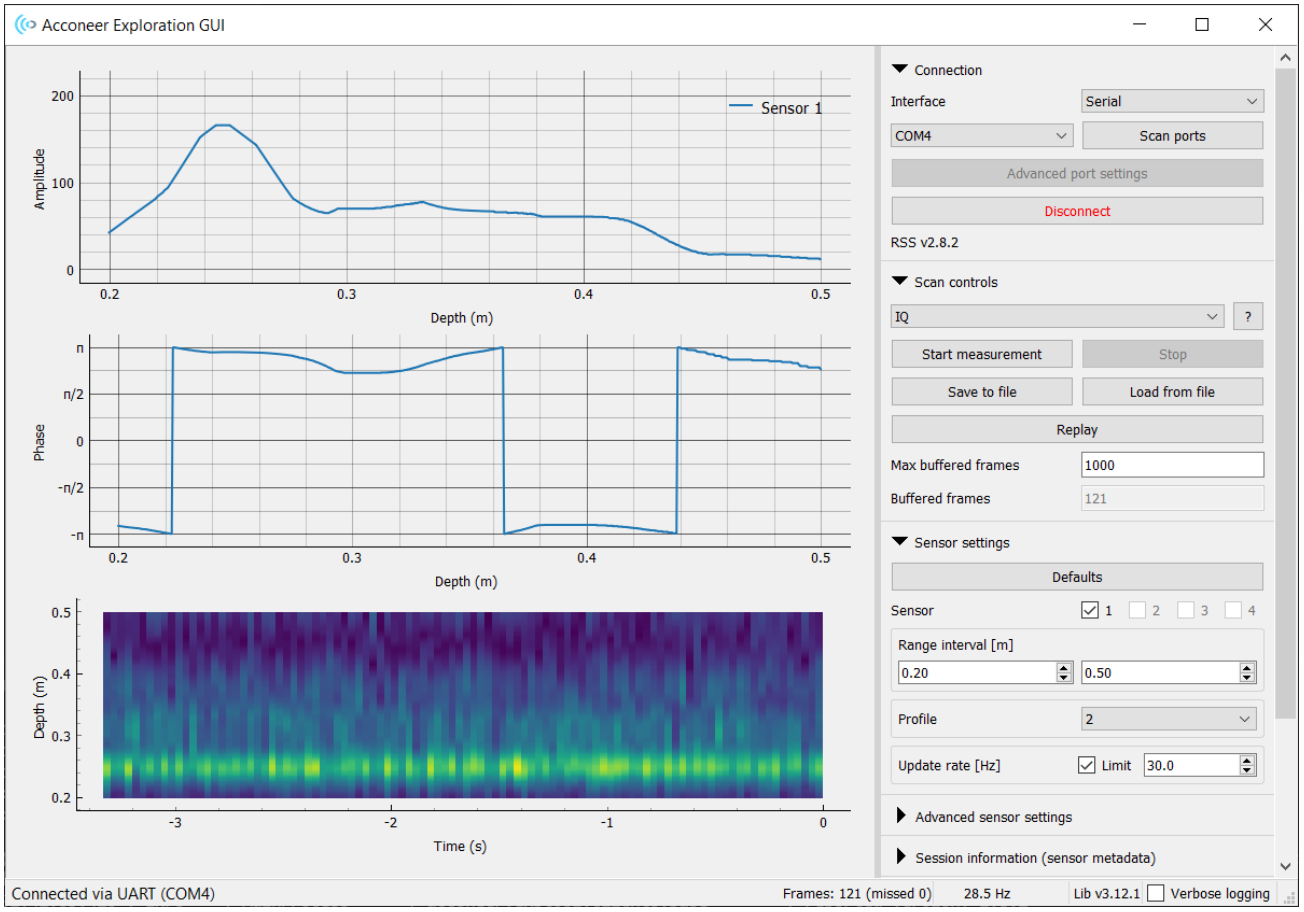


Figure 3: Acconeer Exploration Tool GUI

The range interval for which the radar listens to the received echo is set to the interval from 20 cm to 50 cm. The lower limit was set due to the Acconeer recommended lowest range start for Profile 2, the upper limit is set to 50 cm to detect small-scale hand gestures.

Recognizing waveform patterns

Before the actual collection of samples in data, we first have to recognize various patterns that can be found by gestures. Figure 4 shows the magnitude plot of the various gestures - we can clearly see the distinct waveform patterns.

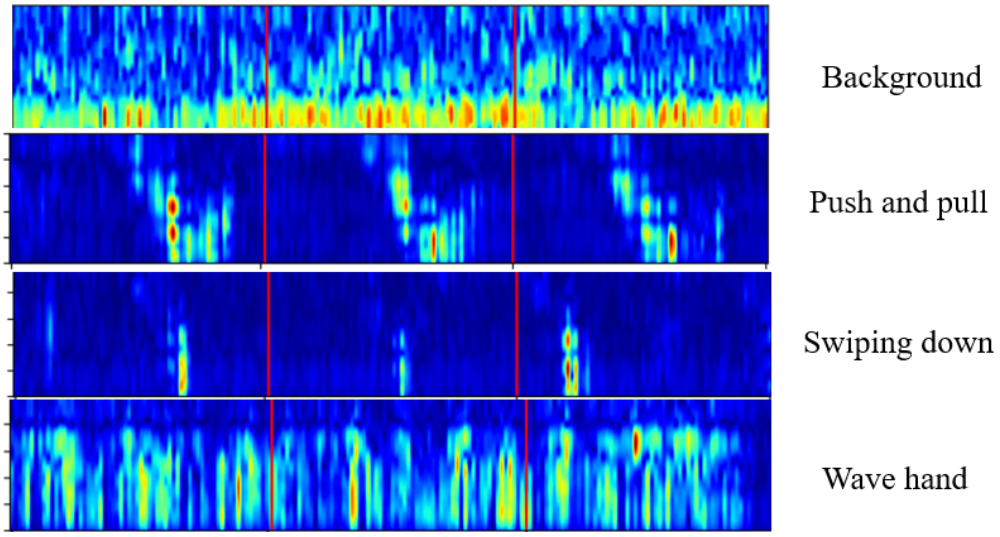


Figure 4: Magnitude plot of various gestures during testing

Optimal range setting

This segment of the report will elaborate on how calibration was done on the radar sensor and choosing the optimal setting. Prior to the experiment, the GUI was set to profile 2 with a range interval of 0.2 m to 1 m and a fixed sample frequency of 30 Hz. There are 2 ranges that were used for this experiment, namely 0.5 m and 1 m.

Figure 5: Calibration for (i) 0.5m and (ii) 1.0m

Figure 5 shows the set-up for this study where it was carried out by placing the XB112 radar sensor at different marked points on the ground to measure the distance from the radar sensor to the upright surface.

An error could be present in the obtained result that could be due to the radar sensor, set up error or the environmental limitation. Henceforth, to ensure the obtained results stay within the ± 5% tolerance rate, we calculate the error tolerance using the Chi formula $(O_i - E_i)/E_i$, where O_i is the obtained value and E_i is the expected value.

From Figure 6, we measured a distance of 50 cm from the radar sensor to the upright surface. However, the detected range obtained at a distance of 50.6 cm was observed instead of 50cm. Applying the chi formula, the error of 0.6 cm corresponds to 1.2%. This percentage error is kept within the tolerance level of 5% and the results are still acceptable.

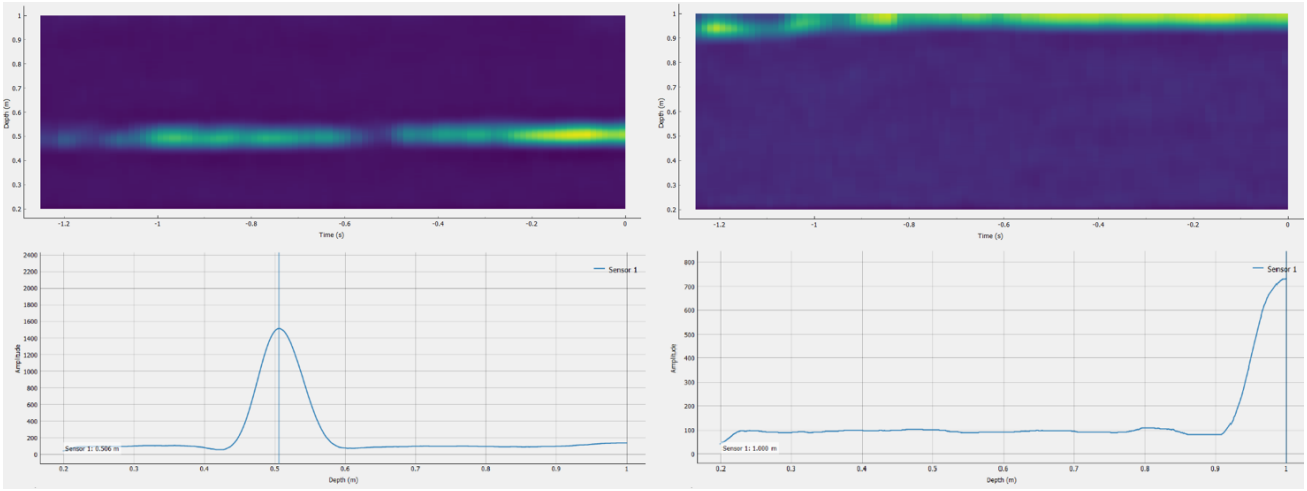


Figure 6: Radar sensor at (i) 50 cm, (ii) 100 cm from upright surface

In the following experiments, the range will be increased to 1 m with the same profile. The experiment included measuring the distance at its maximum of 100 cm and testing if the radar sensor was able to measure beyond its maximum distance. Figure 5 shows the distance of where the target is located. The distance of the target from the radar sensor was measured at 100 cm without any error detected.

Based on Figure 6, we observed that the signal increased up to its peak value when the range was situated at 100 cm. However, the signal remains situated at 100 cm and never returns back to its initial point. This shows that the radar was not able to pick up any point beyond the maximum range of 100 cm. This implied that the radar sensor was unable to detect any points beyond the maximum range of 100 cm and the radar sensor was valid in detecting up to its maximum distance and did not affect its performance.

Varying HWAAS

In order to improve the number of sample points to be collected, we have tried changing the hardware-accelerated average samples (HWAAS) which controls the number of pulses used to build one data point that is related to the number of pulses averaged in the radar to produce one data point [17]. For instance, if HWAAS is set to 10, it means that every data frame is built from 10 data frames averaged together. An excessive wide variety will grow the radar loop benefit however every sweep will take longer to collect and consequently restriction on the maximum update rate.

Acconeer recommended that in order to get better data from IQ service, HWAAS has to be set at least 20 and after many trials and errors in sample collecting the optimal HWAAS value chosen is

25. Figure 7 shows the various HWAAS settings that were being changed. We can observe that as the HWAAS setting was set higher, the signal got more distinct and less corrupt by additive noise.

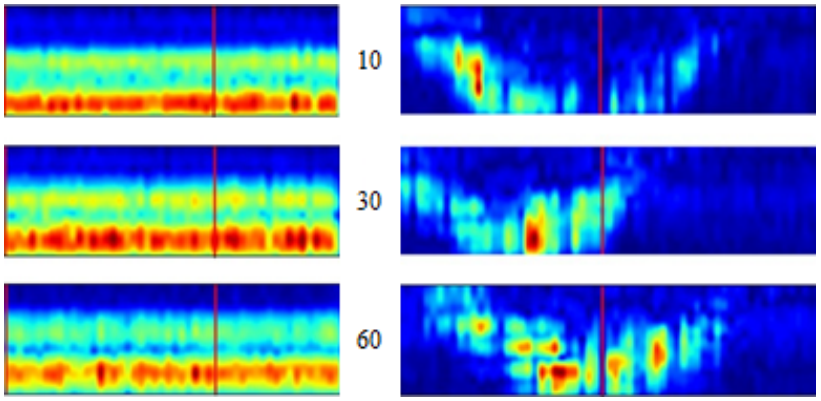


Figure 7: Magnitude of 2 gestures with varying HWAAS

Hardware Accelerated Average Samples (HWAAS) relies on the Depth resolution, where it is the ability to resolve reflections that are closely spaced. Optimizing depth resolution also means improving short-range performance. [17]

The time required to measure a sweep is approximately proportional to HWAAS. Therefore, if you need a higher sweep rate, you can reduce HWAAS [17]. HWAAS does not affect the amount of data transmitted from the sensor over Serial Peripheral Interface (SPI). Hence, our group has settled on a HWAAS value of 25, which is located in the middle of the minimum and maximum range of HWAAS such that it will not limit the maximum update rate. The setting of the update rate cannot be too high as it might result in missed data frames.

Varying downsampling factor

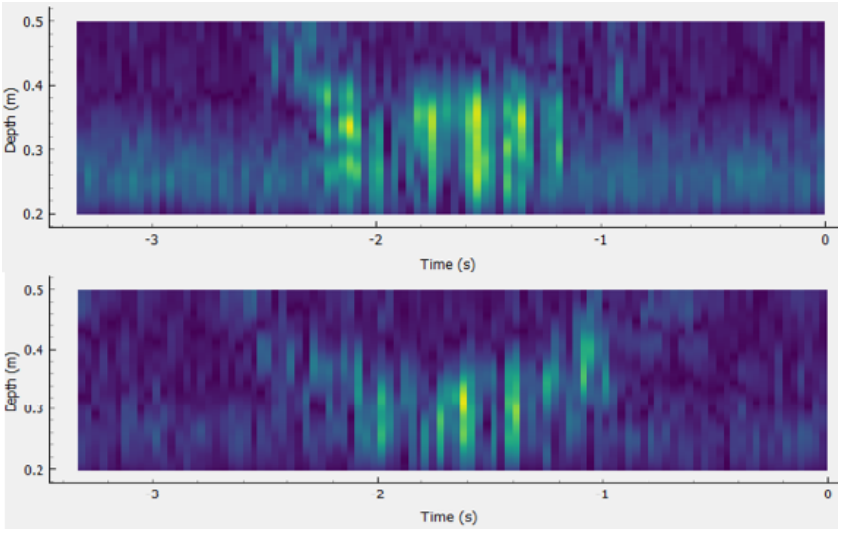


Figure 8: Downsampling factor of (i) ,1 (ii) 4

The following segments show the effect of changing the downsampling variable. Downsampling in Acconeer is an integer factor ranging from 1 to 4. The following experiment uses a factor of 1, which refers to no downsampling used and implies that it is the smallest possible depth interval that will be used. Lastly, a factor of 4 refers to samples at every other point.

Downsampling is performed by skipping measurements on the hardware itself, resulting in lower memory usage and lower power consumption. Figure 8 shows the magnitude plots of the same gesture push with downsampling factors of 1 and 4. An observation can be made that as the downsampling increases, the signal obtained is not as clear as the one with a downsampling factor of 1. Hence, we decide to leave the downsampling factor at 1 as we want to extract the maximum data we are able to collect from the radar sensor.

6.3 Data Collection

At the heart of any ML project is the data itself. Since ML is a statistical-based prediction based on the features extracted from the data, it is of utmost importance that the data that we collect is of high quality. We should also try to match the train set with the statistics/scenario of the application that we will be using the radar sensor in.

6.3.1 Sensor Parameters

Some parameters were also provided by Acconeer technical support and default values were used so that we have something to start with. After exploring the parameters in the previous section, we have determined the suitable parameters summarized in Table 7. Subsequent data collected used these parameters so that there is consistency in the dataset that we generate.

Service	Sampling Frequency (Hz)	HWAAS	Profile	Range Interval (cm)
IQ	30	25	2	20 - 50

Table 7: Parameters of Radar Sensor

6.3.2 Gestures

The dataset was defined in such a way in order to be able to perceive substantial dissimilarities in depth between individual gesture samples. An individual gesture sample comprises 64 data frames from the Acconeer GUI collected. This is equivalent to roughly 2 seconds of gesture ‘air’ time. To vary the train set, every gesture was performed and recorded by each of our 5 members to accommodate for different varieties of hand sizes and shapes. This step also serves to account for the different ways that a person might perform a gesture.

Figure 9: Example of Hold gesture in front of the radar sensor

Figure 9 shows an example of how the gestures were collected. A marker on the floor was used to ensure that our gestures were performed in the 0.2 - 0.5 m range. The gesture requires a change in depth so that the radar data obtained can be differentiated. It was observed that swiping left and right could not be differentiated, which confirms the theory and hypothesis. We experimented with some initial gestures for our first dataset. This will be brought up in later sections.

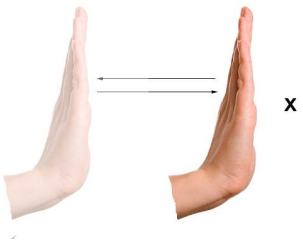
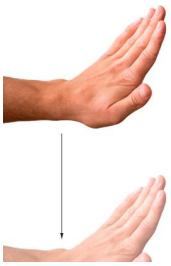
 x Hold It	 x Push
 x Swipe Down	 x Waving

Table 7: Finalised hand gestures excluding background

After thorough experimentation, we have finalised the hand gestures to the ones listed in Table 7. We found that including the pinching class in our dataset resulted in. We observed the magnitude plots to make sure that gestures are. One important note is that we also included the recording of background data/noise. This suffices as a fallback when no gestures are being performed during real-time later.

6.3.3 Gesture Extractor GUI

During the initial stages, we found that it was hard to gauge when to start recording the gestures due to the delay during radar sensor initialisation. This resulted in gestures being recorded that were not center-aligned in the frame. The magnitude plots of the samples can be used to gauge this and a sample is shown in Figure 10.

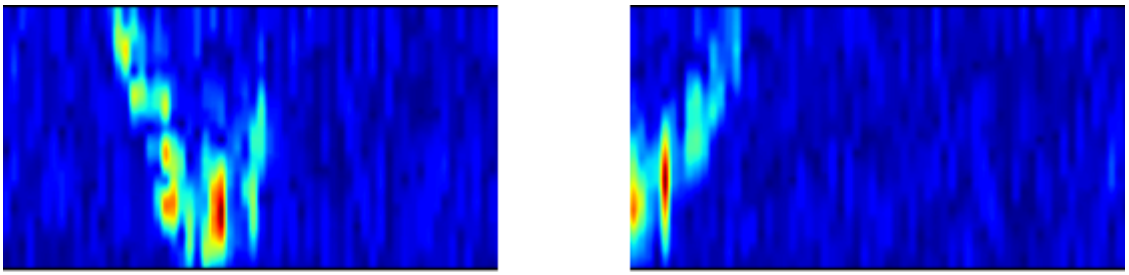


Figure 10: Example of push gesture (i) within frame (ii) out of frame

To fix this issue, we made a Gesture Extractor GUI which uses Tkinter in Python to help with two main functions. Firstly, we are able to record our gestures continuously eg. 30 at one go. This implies that the 'recording' will be shorter and postprocessing can be done later to extract the samples. On top of that, we have fine control of the position of the gesture within the frame, so we are able to center the gesture, thus fixing the alignment issue mentioned earlier.

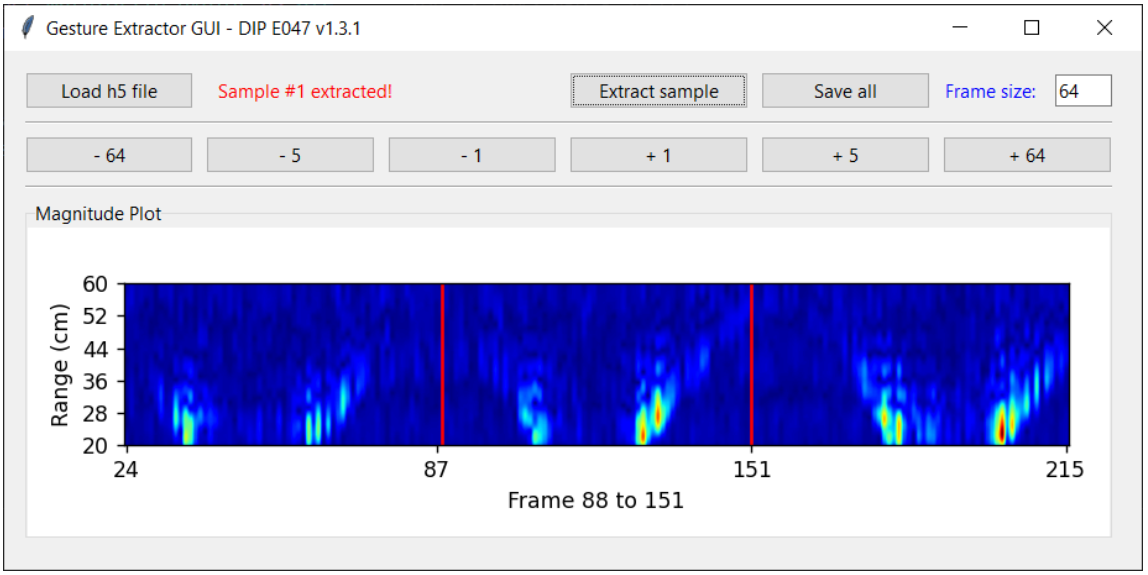


Figure 11: Gesture Extractor GUI

The motivation of the tool was inspired by the preparation of voice sample data for a keyword detection model [18]. A preview of the tool may be found in Figure 11, which was created from scratch solely for the project. It takes in .h5 files and outputs extracted samples in the form of .npz files, which is a compressed file format [19], chosen to reduce the size of our dataset in local storage.

6.4 Data Preprocessing

After obtaining the complex IQ data from the radar data, data preprocessing had to be performed using feature extractors. These feature extractors were used to transform the data into something usable before feeding it into the machine learning model. The feature extractors that we used include the magnitude, Short-Time Fourier Transform (STFT), and Mel Frequency Cepstral Coefficients (MFCC), which transforms the IQ data into a 2-dimensional array. Feature extractors are important as they serve to do the heavy lifting of a deep learning model and reduce the amount of redundant data from the data set. This allows non-deep learning models to learn complex non-linear relationships from the radar data.

6.4.1 Magnitude

The magnitude plot shows the time-varying distance information of the radar signal. It may be obtained by taking the absolute value of the complex IQ raw data and reshaping it, which were used heavily during the data exploration and data collection in earlier phases. Figure 12 shows the magnitude plots for our finalised gestures from Table 7.

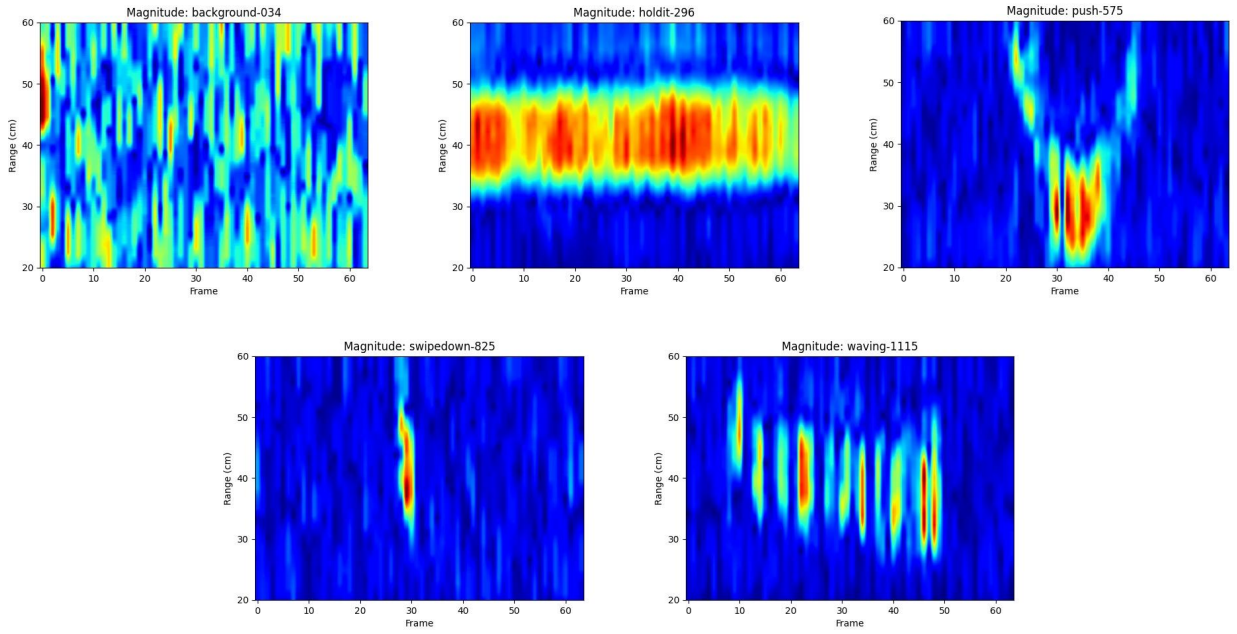


Figure 12: Magnitude plot of (i) background, (ii) holdit, (iii) push, (iv) swipedown, (v) waving

The plots look fairly distinguishable by gesture choice. However, since we have essentially discarded phase information of the IQ data, it might penalise the prediction accuracy when used in our ML models. We will investigate this further in later sections.

6.4.2 STFT

The Short-Time Fourier Transform (STFT) plot shows the time-varying Doppler frequency information of the received signal [20]. It was obtained by moving the doppler window of the Fast

Fourier transform over the time domain radar signal. The resulting change of frequency content of the signal generates a 2-D time-frequency representation known as a spectrogram. In our case, we only extracted the variation of Doppler frequency from -20 kHz to 20 kHz since this region showed. Figure 13 shows the time-doppler plots for our finalised gestures from Table 7.

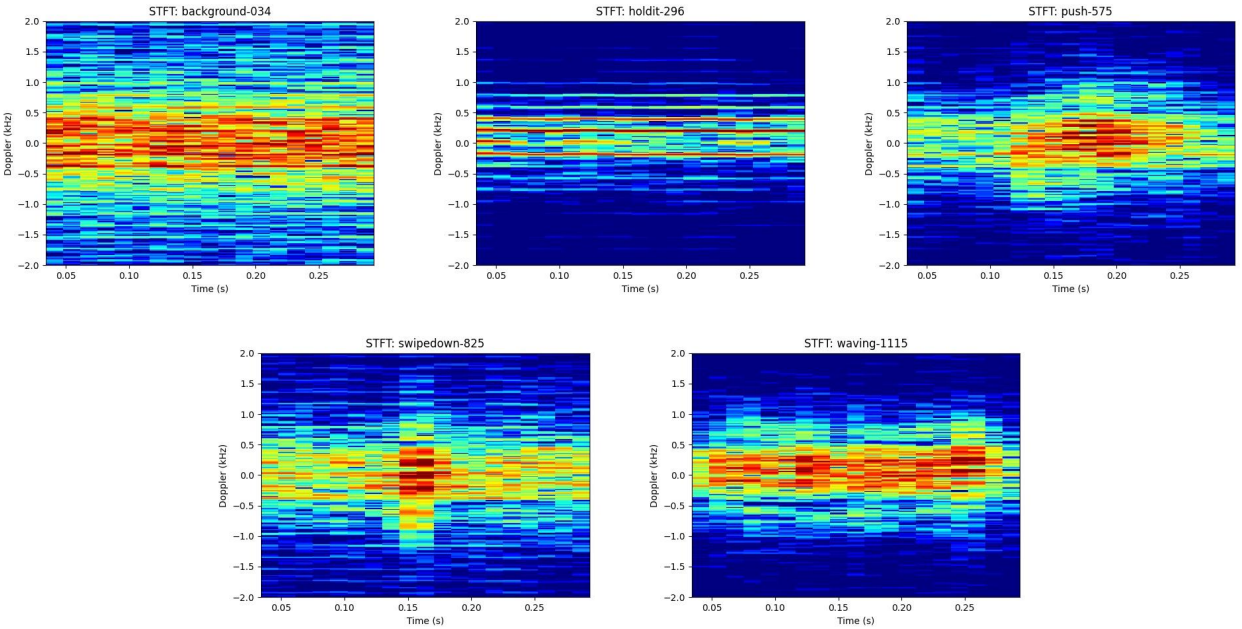


Figure 13: STFT plot of (i) background, (ii) holdit, (iii) push, (iv) swipedown, (v) waving

The STFT plots look fairly distinguishable and this carries the signature information of the gesture. It is expected that the STFT information will be a good feature to use during predictions.

6.4.3 MFCC

We also want to investigate an alternative feature extractor. So, we extract Mel Frequency Cepstral Coefficients (MFCC) features which are usually used in speech recognition systems. MFCC extracts a small set of features that describe the overall shape of the spectral envelope [21]. Figure 14 shows the power spectral density of one of our samples.

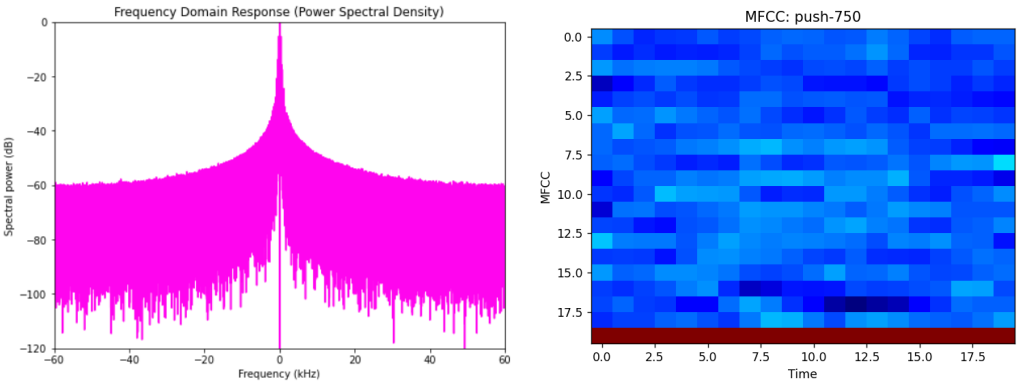


Figure 14: (i) Sample Power Spectral Density, (ii) Sample MFCC plot

Since the spectral density of the radar data is largely the same, we expect that the MFCC plots would be similar too. Figure 15 shows the MFCC plots for our finalised gestures from Table 7, which looks very similar in accordance with our expectations.

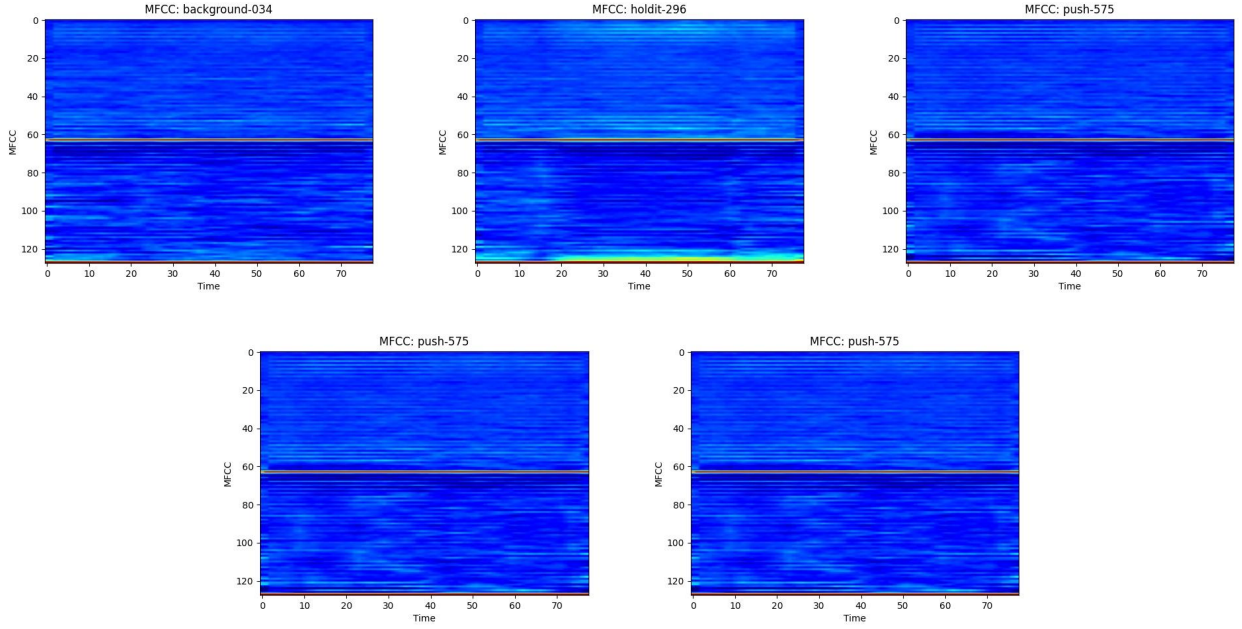


Figure 15: MFCC plot of (i) background, (ii) holdit, (iii) push, (iv) swipedown, (v) waving

We shall test these features with our ML models to see whether it gives a satisfactory prediction quality. The plots in Figure 15 look different from Figure 14 as we have experimented with the parameters and found that sr = 30 and n_mfcc = 180 yields the best prediction accuracy on the test set. A small n_mfcc of 10 to 30, which is typically used in the analysis of audio signals, led to unsatisfactory prediction results when tested in real-time.

6.5 Machine Learning

6.5.1 Machine Learning with scikit-learn

Machine learning is one area in computer science that is gaining massive attention as technology becomes more advanced during the current big data revolution. With the introduction of powerful CPUs and GPUs, coupled with the generation of large amounts of data, it is likely that many businesses utilise some sort of data-centric model to meet their business needs. A basic machine learning algorithm takes in input samples and makes predictions or decisions based on pattern recognition, self-learning and optimization without being programmed to do so explicitly [22]. Some classification models that are commonly used include the k-Nearest Neighbours (kNN), Random Forest Classifier, Gaussian Naive Bayes and Logistic Regression.

6.5.2 Deep Learning

Other than the machine learning models, deep learning models have also been explored and implemented in order to compare and contrast their performances. As technology becomes more rapidly increasing, deep learning has sparked the attention of many, especially in the Artificial Intelligence (AI) domain. Being a subset of machine learning, deep learning works best on dealing with unstructured data such as images, videos etc of which features are able to be self-learned by the model itself.

Since our feature extractor represents the output in a 2D array, they can be thought of as 'images'. The basic idea of 2D convolution, which forms the basis of a Convolutional Neural Network (CNN), involves an application of a filter or kernel to an input to generate feature maps. This is shown in Figure 16. Max-pooling has also been implemented by taking the maximum value in each region to reduce the number of parameters used for the neural network and hence removing the unnecessary features, as shown in Figure 17.

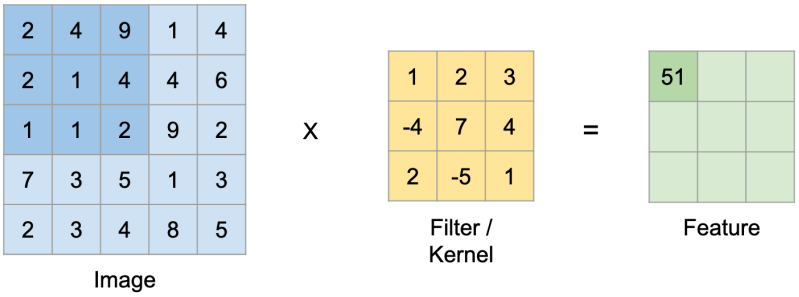


Figure 16: Convolution, the basis of a CNN [23]

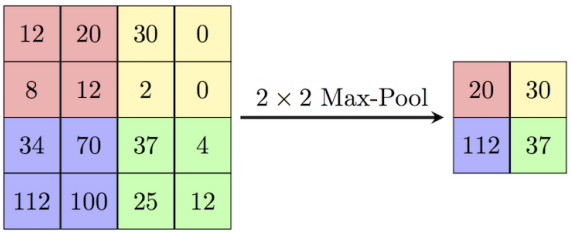


Figure 17: Max-pooling for reducing dimensionality [24]

With reference to basic ideas on the convolutional neural network (CNN), we implemented our own model architecture with inspiration from AlexNet [25]. Specifically, a Tensorflow Keras Sequential model with 20 epochs and a batch size of 8 is used. The model architecture is shown in Figure 18.

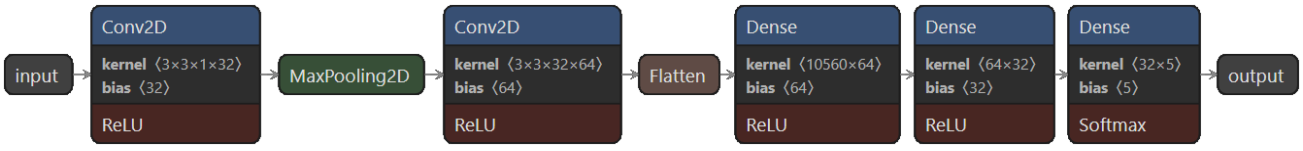


Figure 18: Deep learning model architecture

The categorical cross-entropy loss function and the adam optimizer is used to optimize the Keras model. The accuracy and the area under the Receiver Operating Characteristic (ROC) curve (AUC) are used as metrics to evaluate the performances of both the train and test datasets. Callbacks are instantiated to prevent overfitting of training data on the model. Early Stopping is implemented by monitoring the validation loss with a patience of 4 epochs.

6.5.3 External Dataset Performance

We also worked independently on the machine learning portion while data exploration was being conducted. A mock radar dataset of cars/drones/people obtained from Kaggle [26] was used to train and evaluate the performance of the machine learning models. Figure 19 shows two examples of Doppler matrices of the mock dataset of 17485 samples to be fed into the models. This is analogous to the STFT signatures that we will be using later. The performance of the K-Nearest Neighbors (kNN) model on the mock dataset is summarized in Table 8.

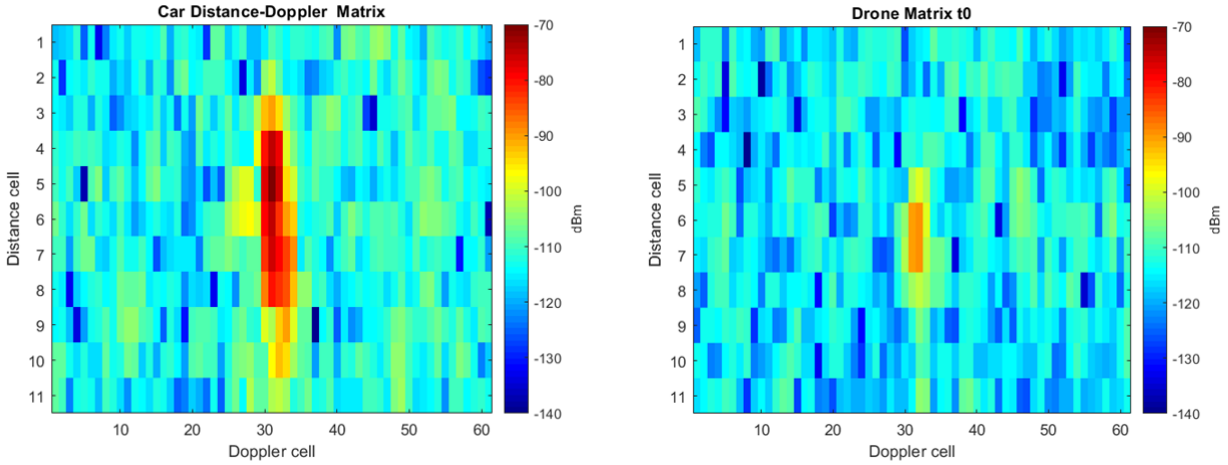


Figure 19: Sample Doppler matrices of the mock dataset - (i) Car, (ii) Drone

Train Accuracy	Train AUC	Test Accuracy	Test AUC
0.953	0.995	0.902	0.966

Table 8: Classification results for kNN on mock radar dataset

6.5.4 Integrating with our First Dataset

After collecting adequate radar data, the machine learning pipeline is implemented using our own dataset with a train-test-split ratio of 0.7. The first dataset size consists of 1000 IQ data samples of four unfinalized gestures (250 samples each) (i) background, (ii) pinching, (iii) push, and (iv) swing. The STFT feature extractor is used to match the mock dataset used earlier. The performance of the K-Nearest Neighbors (kNN) model on our dataset is shown in Table 9.

Since the model is unchanged from the mock dataset, we expect a similar accuracy. However, as observed in Table 9, the accuracy drops when the newly obtained dataset is used. Investigations were done to analyse the data and its features to gain insights and improve the pipeline based on a data-centric approach.

Train Accuracy	Train AUC	Test Accuracy	Test AUC
0.841	0.990	0.847	0.955

Table 9: Classification results for kNN on our first dataset

We realized that the first bulk dataset recorded earlier had some issues. Some of the STFT plots obtained from the dataset are plotted and visualized in Figure 20. Through discussion it was noted that there weren't sufficient time gaps between gestures so extracting samples would have some data from the next/previous gesture. The data was also not recorded well as it was hard to determine the beginning and end of a gesture.

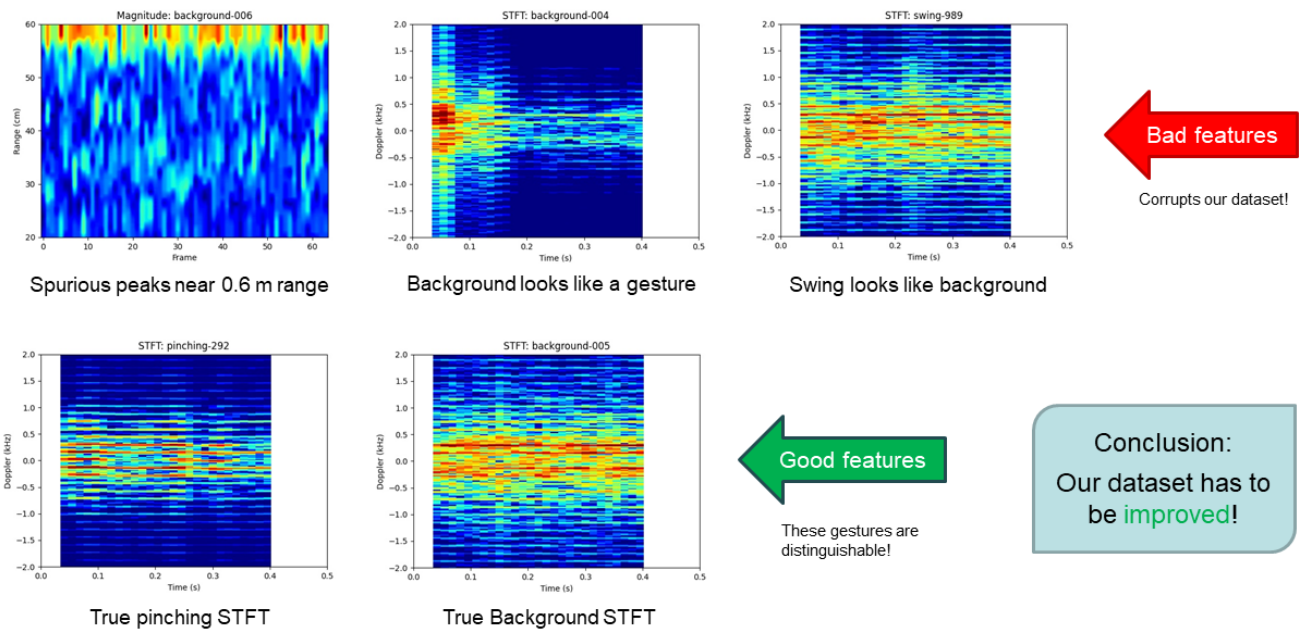


Figure 20: Visualisation of the STFT plots obtained from the first dataset

6.5.5 Data-Centric Improvements

We then decided to re-record our dataset and change our gestures to those in Table 7. The range was reduced to 0.2 - 0.5 m to remove the spurious peaks near 0.6 m. Our team members also advised the gesture usually does not exceed the 0.5 m range so it was safe to discard data beyond 0.5 m. Care was taken this time too to ensure that the gestures were sampled when they are in the middle of the frame ‘window’, with sufficient gaps between subsequent gestures.

The performances of the classification models are summarized in Table 10. Consistent with the earlier radar dataset with the unfinalized gestures in Section 6.6.2, the Linear SVM and Logistic

Regression models performed better than the other models. The Decision Tree Classifier, however, did not perform as well as the previous dataset. For the deep learning model, the train and validation accuracies and losses are plotted in Figure 21.

ML/DL Models	*Train Acc*	*Test Acc*	*Train AUC*	*Test AUC*
K Nearest Neighbours (kNN)	0.971	0.967	0.999	0.999
Random Forest Classifier	0.993	0.995	1.000	1.000
Gaussian Naive Bayes	0.837	0.807	0.905	0.886
Decision Tree Classifier	0.923	0.896	0.992	0.983
Linear Support Vector Machine (SVM)	1.000	1.000	1.000	1.000
Logistic Regression	1.000	1.000	1.000	1.000
Deep Learning	1.000	1.000	1.000	1.000

Table 10: Performance comparison of ML/DL models

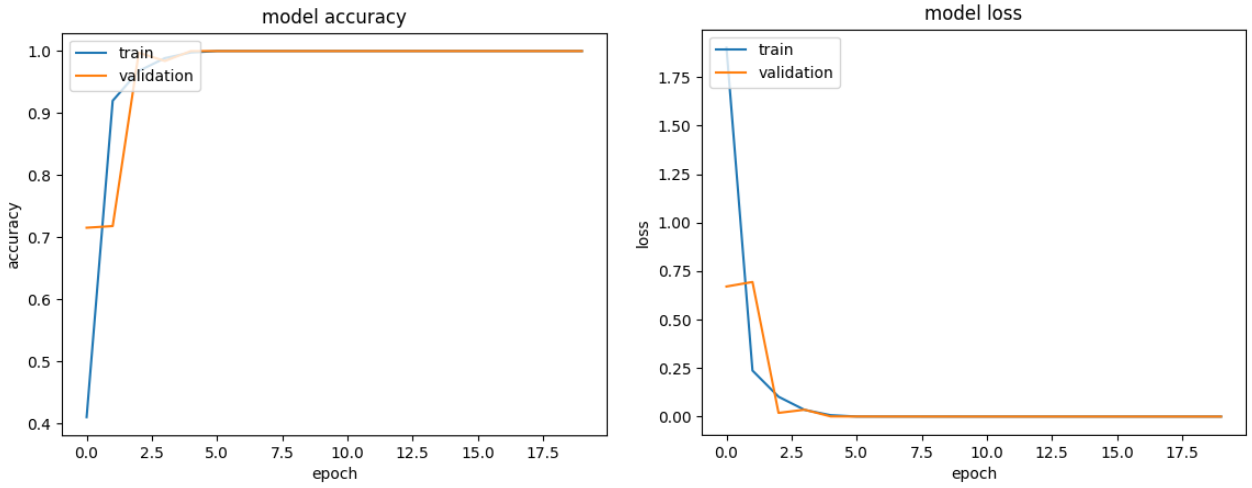


Figure 21: Train and validation (i) accuracy (ii) loss

The two finalized models are (i) Logistic Regression - because of its simplicity and (ii) the Deep Learning model - which is expected to perform better when working with images.

6.6 Real-Time Implementation

Acconeer also provides a software development kit (Acconeer SDK) so that we as developers may interface with the sensors directly by calling higher-level app programming interfaces (API), as supposed to interface as the device driver level. The SDK was also explored at an earlier stage so that we are familiarised with it.

When the machine learning pipeline is almost finalized, we may begin the real-time implementation. All of the individual Python scripts (eg. radar.py, preprocess.py, ml.py, etc) are saved in a “master” folder and are grouped into individual classes. These scripts form our API functions which we will use directly since the individual components have been finalised.

Figure 22: Summarized real-time implementation

Care must be taken when implementing the code to allow for real-time predictions. This will require special modifications so that the gesture prediction is satisfactory in terms of both accuracy and run-time. This effect will be apparent especially when migrating to the Raspberry Pi. The summarized real-time implementation pipeline can be found in Figure 22. Further methodologies will be discussed in this section to help improve the accuracy and robustness of our project’s use.

6.6.1 Run-time Analysis and TensorFlow Lite

Each of the blocks in Figure 22 is tested to find the average run-time. The functions were tested on a Windows machine by looping on 10000 iterations. Please note that the absolute times are not so much of concern but the relative times between each API function is more important.

Block	*API Function*	*Average Run-time*
Feature extractor	Magnitude	350 µs
	MFCC	7.5 ms
	STFT	24 ms
Reshape features	Reshaping data	10 µs
ML/DL model	Machine Learning (Logistic Regression)	400 µs
	Deep learning (non-optimized)	60 ms
	Deep Learning (tflite)	500 µs

Table 11: Comparison of timings for each API function

The deep learning model takes relatively longer than other API functions. This posed no issue when we ran our script on our laptops, but led to time synchronization issues when migrating to Raspberry Pi. As a result, our deep learning model needs some optimization and was converted to a TensorFlow Lite model with default settings to reduce the evaluation time. This saved model file is saved as a .tflite file which is a flat buffer file [27] so that fast inference can take place.

TensorFlow Lite is an open-source deep learning framework optimized for on-device inference [28] such as mobile and embedded devices which have limited computing resources. Many solutions have been successfully deployed using this framework, with some current applications in image classification and object detection. It is worthy to note that for our project, converting the deep learning model to TensorFlow Lite brought a run-time speed-up of about 120 times, which will be very significant when we run our scripts on the Raspberry Pi.

6.6.2 Special Techniques

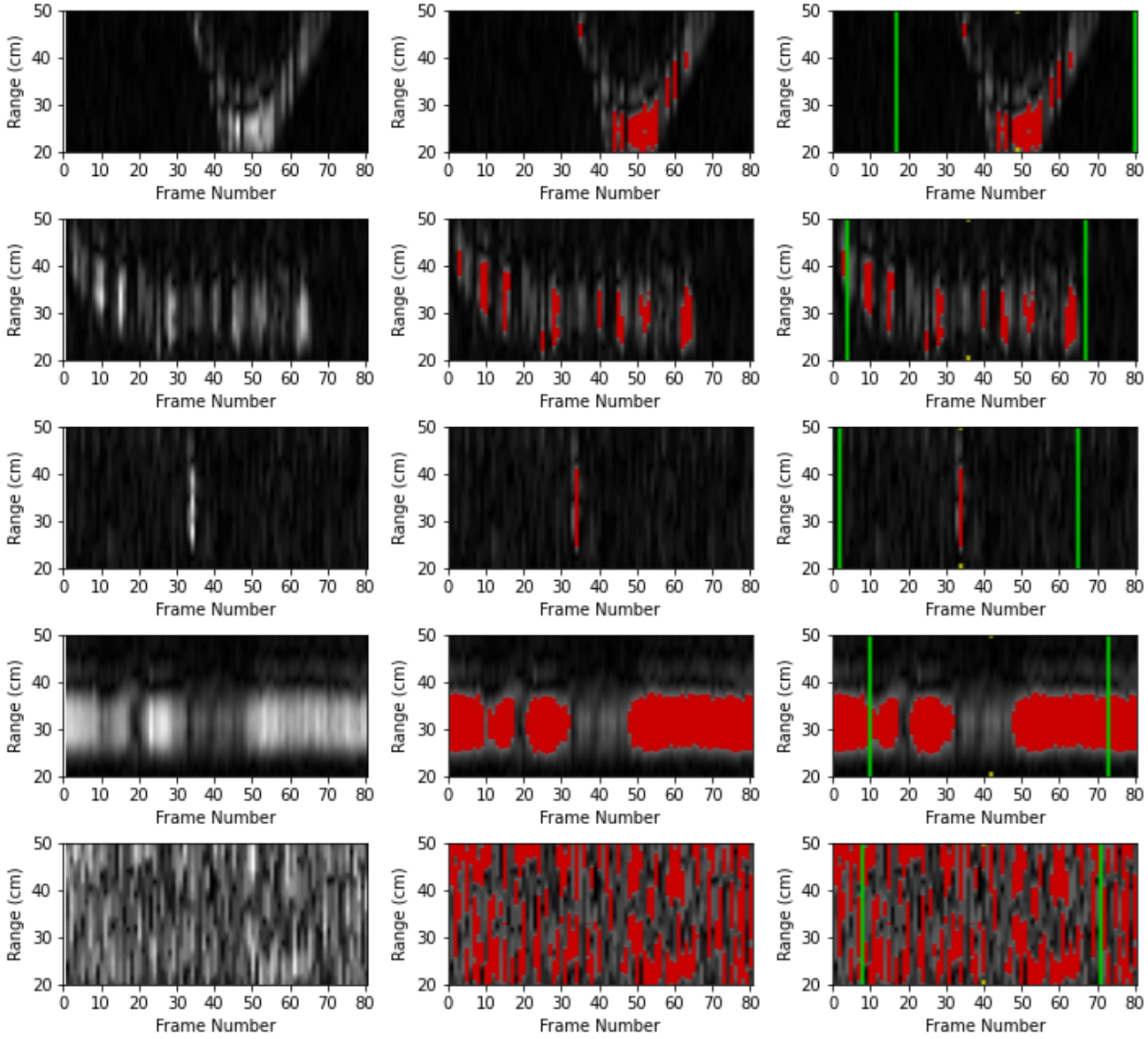


Figure 23: Centering Algorithm - (i) Magnitude plot (ii) Thresholding (iii) Center window found

Our initial fix was to sample the gestures in a way that is not centered within the 64 frames. However, this led to a worse test accuracy of 83% using the same parameters, with the real-time predictions performing even worse than earlier. We suspect that we might need a larger dataset that

is 5 times larger to accommodate for all the different positions that the gesture can be in. Seeing that this is unfeasible in the time span we have, we employ two other techniques to solve this issue.

We first extended the frames recorded to a total of 80 frames, compared to our trained data which was based on 64 frames. Figure 23 shows a centering algorithm that uses the magnitude plot to find the frame center of the 80 frames. The magnitude values are first normalised and pixels of interest are thresholded at more than 0.4 of the maximum value - these pixels are shown in red in Figure 23. The frame numbers of these pixels of interest are extracted and their mean is calculated. This value corresponds to the window center, of which a window of size 64 frames is 'cropped out' and fed into the subsequent pipeline. This centering algorithm proved to be very useful and robust as it contributed the most to the resolution of the issue described earlier.

We found that we faced a major issue during the real-time implementation - the accuracy of the predictions were not satisfactory and in fact mostly inaccurate. Upon further investigation, we discovered that it does work accurately, but only when the gesture is performed at a specific timing. This indicates that our gestures must always be synchronized specifically such that it is centered within the window frame. This is hard to execute in real life and leads to a decrease in robustness in our solution.

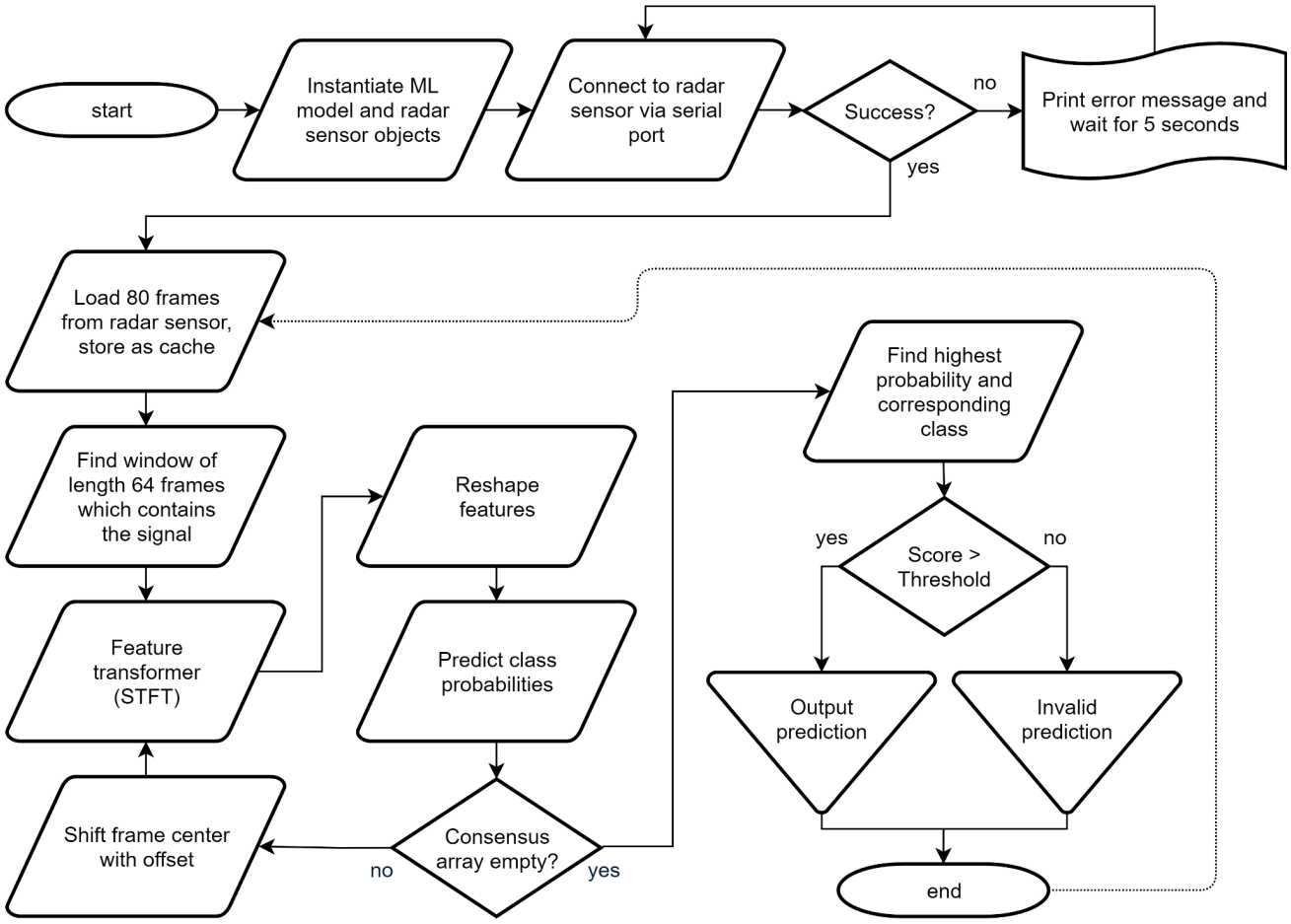


Figure 24: Algorithm for real-time implementation script

The second technique to complement the centering algorithm earlier is to draw consensus votes from frames by shifting the window center obtained earlier by {-k, -k+1, …, 0, …, k-1, k} indices. This is equivalent to taking k future and past samples each to form a total of 2k+1 samples. This improves the accuracy by taking the average votes of each consensus. At the same time, there would be improved robustness to the timing and synchronization issues faced earlier. The gesture predictions would only be valid if a gesture obtains a majority consensus of 0.6 or better.

6.6.3 Algorithm for Real-time Implementation

A considerable amount of time was spent to improve the real-time implementation to a satisfactory level. As such, the algorithm for real-time implementation can be found in Figure 24, which incorporates both the special techniques discussed earlier. The script runs in an infinite loop unless an Interrupt is used.

6.6.4 Hybrid Models

From the results in Table 10, it is evident that Logistic Regression is also a good model. Since the evaluation time for the Logistic Regression model is minor as seen in Table 11, we also include it as a predictor alongside the TensorFlow Lite DL model. The prediction probabilities are obtained from both models and averaged. This allows for voting from both models to ensure that the gestures predicted are more robust. For real-time predictions, a threshold of 0.6 was used to remove any predictions that were not ‘confident’. These predictions which score below the threshold are printed out as N/A.

6.6.5 Comparison of Feature Extractors

This section uses the DL architecture introduced earlier, while varying the feature extractors. All feature extractors attained a test accuracy of 98% and above on the validation set. The qualitative results for real-time predictions are summarised in Table 12. Since STFT is the most reliable feature and includes some form of robustness, we choose to use this as our finalised feature extractor.

Feature Extractor	*Prediction Quality*
Magnitude	All gestures can be distinguished except Swipe Down which was predicted as Push
STFT	Able to distinguish between all gestures, Reliable
MFCC	n_mfcc = 30 : Bad prediction results n_mfcc = 180 : All gestures can be distinguished except Background which was predicted as Swipe Down

Table 12: Prediction quality of various feature extractor

6.7 Raspberry Pi Integration

The Raspberry Pi, developed by Raspberry Pi Foundation in association with Broadcom, is a small single-board computer that is commonly used by computer science hobbyists because it is low cost and open design. The operating system of Raspberry Pi which is a 32-bit Linux-based OS is free to download and it promotes Python and scratch as its main programming languages.

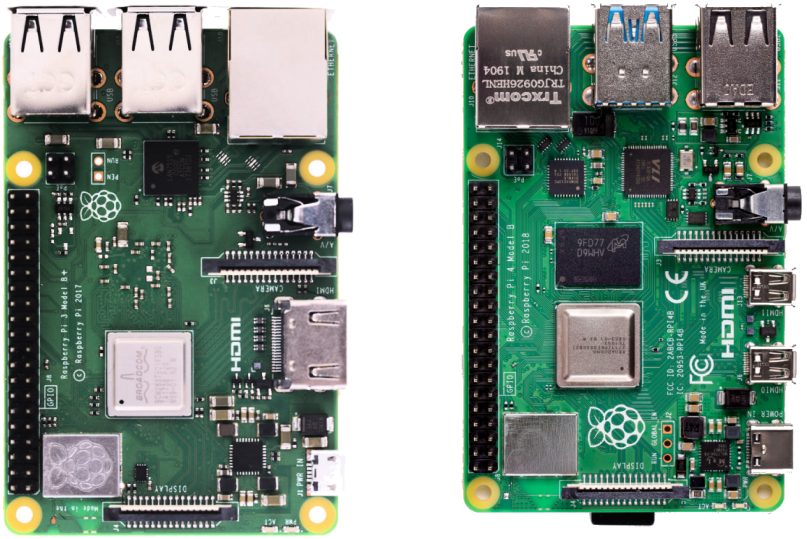


Figure 25: (i) Raspberry Pi 3B+, (ii) Raspberry Pi 4 [29]

For our project, we purchased the Raspberry Pi Model 3B+ and the latest Raspberry Pi 4B model with 8GB RAM shown in Figure 25. Both devices have a dimension of 85.6 x 56.5mm. The 4B model has a faster 1.5 GHz clock speed processor than the 3B+ with a clock speed of 1.4 GHz [29]. This might not spot a big difference, but the Raspberry Pi 4B has higher RAM compared to the entry-level 1GB for the model 3B+. Compared with Raspberry Pi 3B+, the performance of Raspberry Pi 4 has tremendously improved as shown in Figure 26. So, even if the Raspberry Pi 3B+'s performance may meet our requirements, we also purchased the latest model to explore the differences between the two models.

6.7.1 Setup and Installation

VNC Viewer is used as a remote desktop for the Raspberry Pi to reduce unnecessary costs of purchasing a monitor and HDMI cables. Connection to the Raspberry Pi can be done via static IP assignment [30], in which a physical Ethernet cable is connected from the Raspberry Pi to the computer. An alternative method is connecting the Raspberry Pi to a router, which was preferred as multiple users were able to access the Raspberry Pi at the same time and the access could be done over Wi-Fi. Since the router had Internet access, this enabled the Raspberry Pi to have Internet access too without much configuration.

Python package dependencies (e.g., tensorflow, opencv, numpy) were also installed on the Raspberry Pi using pip3. Some of the issues faced included the librosa/numpy library, which was not able to be installed via pip3. Extensive research was done to find alternative methods to install our required libraries.

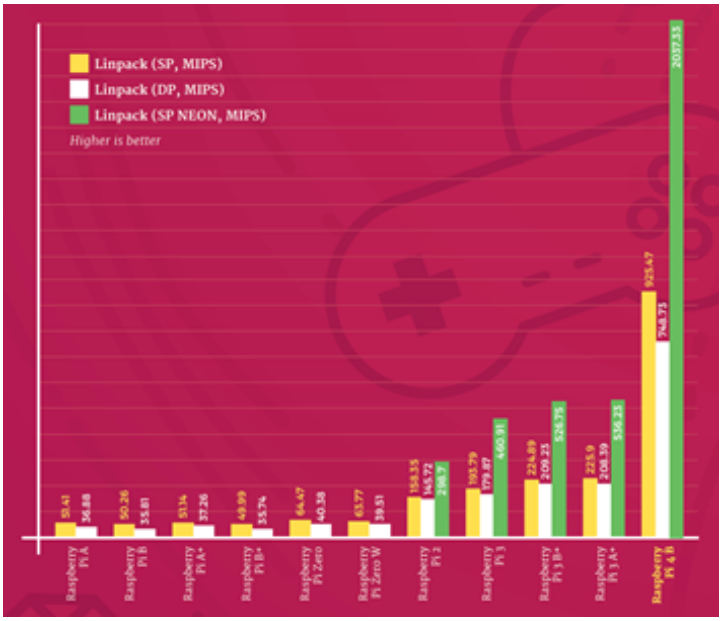


Figure 26: Linpack benchmark test for Raspberry Pi Models [29]

After pulling the code from our repository in GitHub, we also managed to run the real-time python code on the Raspberry Pi with some errors. After some debugging and fixing some compatibility issues of our script with a Linux system, which includes defining a specific port for the Raspberry Pi ('/dev/ttyUSB0'), we are able to run the script successfully.

6.7.2 Real-Time Inference

The real-time python code on the Raspberry Pi didn't run as well as the laptop. The real-time prediction we got in Raspberry Pi was wrong and it showed a hand gesture prediction which was different from our hand gesture movement. Through the error printed on the Raspberry Pi when we run the code, we found out that one of the issues is that TensorFlow takes up too much system memory.

We checked through intermediate magnitude and feature plots to ensure that they are correct. We realise that the script is able to correctly perform the prediction when the gesture is performed after waiting for 0.5 seconds after the previous result is printed. Upon analysis of the runtime of each API function on the Raspberry Pi, we realised there is significant processing time for some API functions which caused a large time delay. Optimization measures had to be performed to resolve this issue.

6.7.3 Resolution

TensorFlow Lite was introduced to optimize and reduce the run-time for the deep learning model inference. Besides that, the real-time accuracy drops due to the time delay in feature extraction, in particular the STFT code. The consensus voting technique from past and future frames we used to improve prediction results brought a negative effect when implemented on the Raspberry Pi. We observed that the evaluation time in Raspberry Pi is much slower than in the laptop which is shown in Table 13. This evaluation time includes all the API functions in the entire pipeline.

If the consensus buffer was left at k=2 as in the laptop, the average evaluation time would be 890 ms in the Raspberry Pi 3B+. This exceeds the tolerable delay for bidirectional telecommunications delays which is 300 ms [31], hence the penalty in accuracy that was observed earlier. It is also important to note that a delay of 890 ms is roughly equivalent to 27 frames (out of a total of 80 frames), which represents a significant delay in the gesture window.

Consensus buffer (k)	No. of samples (2k+1)	Average evaluation time in laptop	Average evaluation time in Raspberry Pi 3B+
0	1	25 ms	190 ms
1	3	71 ms	530 ms
2	5	107 ms	890 ms

Table 13: Evaluation time with varying consensus buffer

We conclude that using the consensus voting technique will help with the accuracy and robustness, but causes an undesired “talk over” to the real-time processing. However, this slight drop in accuracy and robustness would not harm the performance of the real-time predictions by a significant amount. Hence, the optimal consensus buffer values are k=0 for Raspberry Pi and k=2 for laptops after much experimentation.

After all optimizations, we found that the real-time script was able to run on the Raspberry Pi 4 with satisfactory accuracy. We also concluded that our optimized script was not computationally expensive and worked well running on the 3B+ model with lower specifications.

7. Outcomes / Benefits

This section showcases our project outcome, namely the deliverables and one specific use case of gesture recognition. We also compared the robustness and limitations of our solution, then compared the devices that we purchased to conclude that our project can be integrated independently using a Raspberry pi 3B+ and one Acconeer sensor.

7.1 Project Deliverables

7.1.1 Results

Our real-time script output is shown in Figure 28, which clearly shows predictions made when the gesture is performed in front of the radar sensor. To make the result more visually appealing and functional, we made a GUI which is showcased in the use cases subsection in Figure 30.

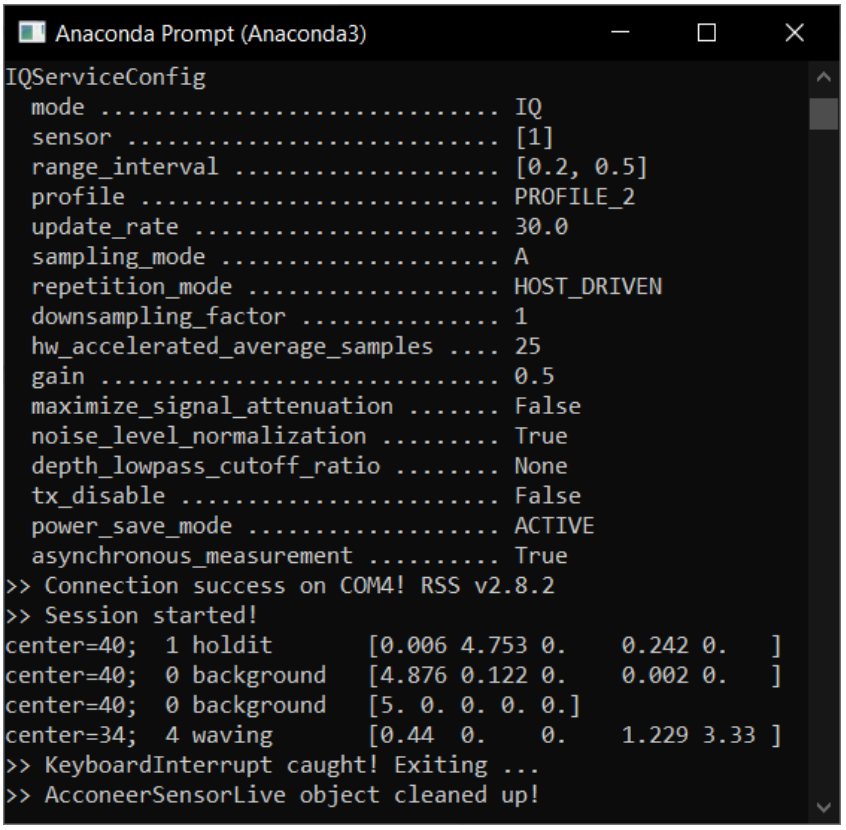


Figure 28: Screenshot of Predictions - Command Line

7.1.2 Robustness and Limitations

All four gestures are distinguishable from each other. In particular, one gesture, Hold it, faces some difficulties to be predicted accurately on the first try. However, as we allow some buffer time and wait for a while, the hold it gesture would see an accurate prediction, usually on the second try. This is because of the nature of the gesture in which we recorded the entire frame for the hand to be in

front of the radar sensor. This is hard to achieve in practice without first entering the ‘detection region’ by swiping in, which means that the synchronization has to be exceptionally precise for this gesture.

The other gestures (Push, Swipe down, Waving, Background) show relatively higher accuracy prediction results at the first try, with a real-time accuracy of up to 88%, based on random and independent tries of gesture predictions. We conclude that our results are extremely satisfactory considering that our dataset is many magnitude times smaller compared to datasets used in current literature. There might also be inherent biases of the gestures as they were only recorded from 5 of us in the team.

Figure 29: Radar sensor with lid

Since radar promises that it is able to penetrate materials with ease and yields robustness, we attach a lid to the container as shown in Figure 29. We found that all gestures were able to be predicted with high accuracy, except Background which is predicted as Hold It. However, this is not too surprising as the lid resembles a hand in front of the radar sensor. This issue is easily fixable by (i) recording more data with the lid in place to expand the train statistics such that it matches the radar sensor with lid statistics, or (ii) changing the Hold It gesture itself to something with more dynamic motion.

7.1.4 Conclusion on Sensor Modules and Raspberry Pi

Due to optimization steps performed and discussed earlier, our project runs well on the Raspberry Pi 3B+. The project is able to run independently as an integrated product with the Raspberry Pi.

There are some limitations with the Acconeer XB122 IoT sensor module, in particular, the update rate can only go up to 30 Hz without dropping too many frames. The Acconeer XB112 sensor module, on the other hand, could go up to an update rate of 80 Hz. However, the good news is that although the ML models were trained using the XM122 sensor, running it using the XB122 sensor also yielded good predictions. We then conclude that the XB122 sensor is desirable for IoT usage

since the XB122 sensor can be powered with a button battery (with lower power consumption) and supports wireless data transmission.

As for the Raspberry Pi 3B+, using it was slightly slower compared to the Raspberry Pi 4B. Hence, all exploration was done on the Raspberry Pi 4B as it was faster, but we concluded that the Raspberry Pi 3B+ is sufficient after we cloned the deployment steps onto the Raspberry Pi 3B+.

7.2 Use Cases and Benefits

Touchless Gesture Control could see huge potential growth in the following years, and with the advancement of machine learning and artificial intelligence incorporated into our daily lives, it would be inevitable for it to be considered the future of Human-Computer Interaction (HCI). Touchless gesture control could be implemented in a variety of scenarios. From ordering food on a restaurant menu to virtual reality games, the implementations are limitless just as a mouse and keyboard are used everywhere. Especially in the current post-Covid era, advancement in touchless technology could be the solution to reduce physical contact and reduce the spread of COVID-19 or any other similar diseases that could possibly occur in the future.

One useful application of touchless technology would be ordering food in a restaurant. With touchless gesture technology, it reduces physical contact between the customers and a menu kiosk, which minimizes the risk of transmission of COVID-19 or any other similar diseases.

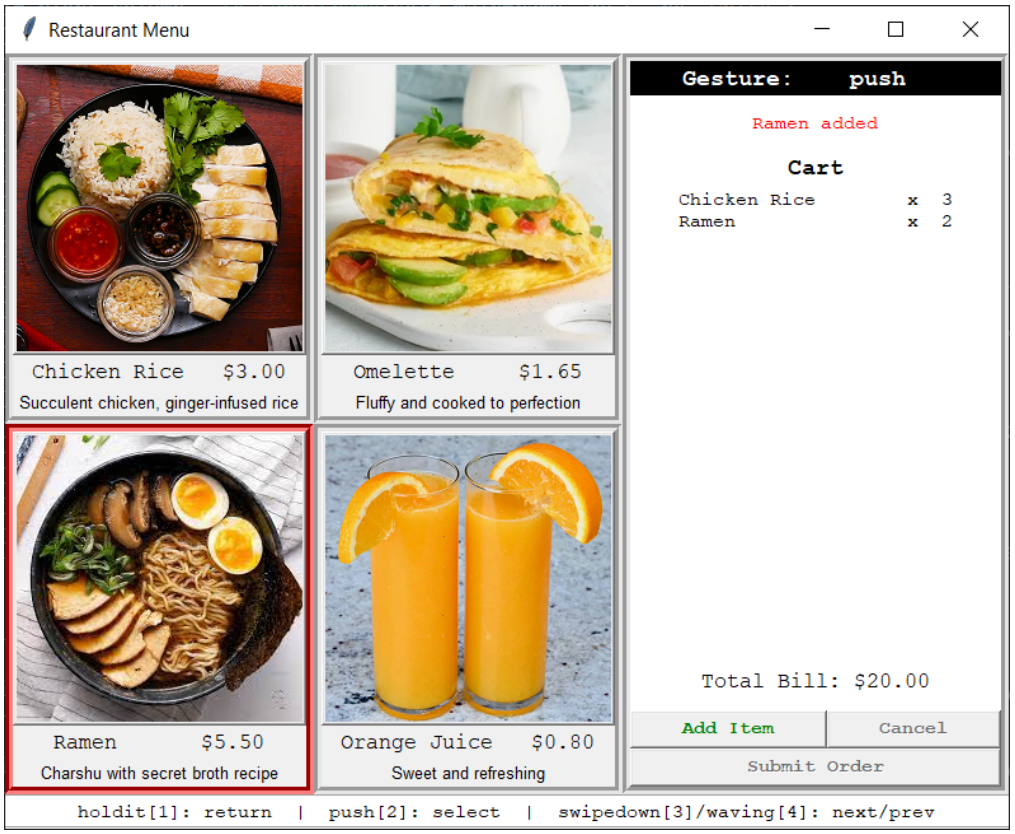


Figure 30: Menu Application

To demonstrate our project's use case, we developed a fully-functional menu application that accepts keyboard presses according to the following actions:

- [1] Hold It: enter the order cart on the right side of the menu
- [2] Push: select a food option from the menu, confirm the selection
- [3] Swipe Down: Move to the next option from the food selection menu
- [4] Waving: Go back to the previous option from the food selection menu, cancel the selection

A screenshot of the User Interface for our menu application, developed via Tkinter, is shown in Figure 30. We modify our prediction script to perform key-bindings to map the gesture prediction outputs to keyboard characters. This allows for each gesture to correspond to an action on the menu application. Table 14 shows a summary of the key-binding mappings.

Key	0	1	2	3	4	.	/
Function	Background	Hold It	Push	Swipe Down	Waving	Timing Sync	N/A

Table 14: Key-bindings and their mappings

8. Reflection

There were several challenges and hardships that occurred throughout the project. Whilst trying to solve the challenges faced, the whole team have gained both technical and soft skills along the way. For each problem detected, we performed a thorough analysis and suggested, designed and developed solutions to resolve the problems.

8.1 Data Collection

The first pipeline of the project is the data collection pipeline. Our team faced a number of challenges in collecting consistent and quality data. To develop ways to improve the collection of data. It is necessary to identify the barriers to consistent data collection qualitatively. The data collection challenges are summarized in Table 15.

Problem Analysis	*Design/Development of Solutions*
Different environment locations of data collection would result in inconsistent data samples, which would lead to bad prediction results.	Location of data collection has to be a controlled variable, which is fixed at Multimedia Technology Lab (S2.2-B4-02). The importance is that the space needs to be a large open area. The set-up of the radar sensors has to be consistent during the data collection process, as shown in Figure 2.
Inconsistent durations of data collection will result in distorted radar data collected. Some radar data would contain redundant information whilst others contain too little information.	Fix the duration of the data collection process for consistency and communicate the idea to the whole group.
Many similar depth-based gestures are detected for gestures that have similar depth differences. This is because we are using the variation in-depth measurement to differentiate various types of gestures.	Perform trial and error, and analyze the magnitude plots to detect any similarities that would be hard to be differentiated even with human capabilities. Choose and finalize 4 gestures that have obvious depth differences, shown in Table 7.
Fatigue can result from consistent data collection over long periods of time.	Take breaks during the data collection of each gesture, to ensure the quality and consistency of the radar data.
Manual data collection is too repetitive and tiring.	Develop a custom Data Collection GUI to automate the data collection process, shown in Figure 31, which is proved to be unsuccessful.

Recorded gestures using the custom Data Collection GUI in Figure 31 were not center-aligned in the frame due to delays during radar sensor initialisation, visualized in Figure 10.	Stop using the custom Data Collection GUI to collect radar data. Use the original Data Collection GUI developed by Acconeer to obtain radar data continuously. Develop a Gesture Extractor GUI, shown in Figure 11, to extract the samples and center them, thus fixing the alignment issue.

Table 15: Challenges faced and solutions implemented during the data collection pipeline

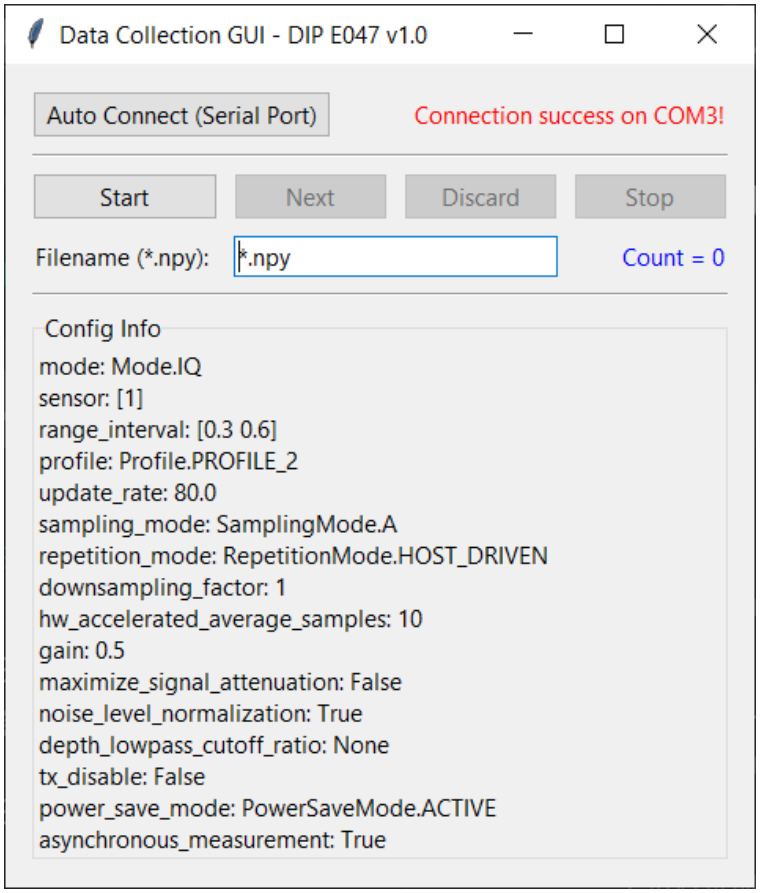


Figure 31: Custom Data Collection GUI in Python (unsuccessful)

8.2 Preprocessing

Upon completion of the data collection pipeline, we begin the preprocessing pipeline. This stage involves feature extraction on the radar data obtained earlier to be suitable as inputs for machine learning. The preprocessing challenges are summarized in Table 16. We also tried implementing the Principal Component Analysis (PCA) during feature extraction, which was used in some papers [32] but removed it at the end because there was not much increase in accuracy values.

Problem Analysis	*Design/Development of Solutions*
Feature extraction of STFT on radar data performed worse than magnitude features.	Visualized the STFT plots on radar data and realized that there were a lot of redundant regions. Set the y-limits of the STFT plots to contain Doppler frequencies only from -2kHz to +2kHz.
Limited knowledge of feature extraction techniques (e.g. STFT and MFCC).	Research and read up on relevant materials on preprocessing techniques to have a better grasp of the theories.

Table 16: Challenges faced and solutions implemented during the preprocessing pipeline

8.3 Machine Learning

The next stage is the machine learning pipeline. Its challenges are summarized in Table 17.

Problem Analysis	*Design/Development of Solutions*
Data collected is not sufficient to evaluate the performance of machine learning models.	Worked on a mock radar dataset of cars/drones/people and used the evaluation results as a baseline for the performance evaluation of the models.
Radar data obtained is different from the dimensions required for the machine learning/deep learning models.	Performed analysis on the shape and dimensions of the radar data and utilized tools (e.g. reshape function) to alter the dimensions required for machine learning/deep learning models.
The performance of the first bulk dataset performed poorly compared to the baseline metrics obtained via the mock dataset.	Visualized STFT plots obtained from the dataset and realized that there were insufficient time gaps between gestures so extracting samples would have some data from the next/previous gesture. The data was also not recorded well as it was hard to determine the beginning and end of a gesture. Re-recorded our dataset and changed our gestures to those in Table 7. The range was reduced to 0.2 - 0.5 m to remove spurious peaks near 0.6 m. Ensured that gestures were sampled when they are in the middle of the frame ‘window’, with sufficient gaps between subsequent gestures.

Table 17: Challenges faced and solutions implemented during the machine learning pipeline

8.4 Real-Time Integration and Raspberry Pi

The final stage would be the real-time integration and Raspberry Pi pipeline. The integration challenges are summarized in Table 18.

Problem Analysis	*Design/Development of Solutions*
Massive destruction that changed the environment path variables ensued when installing Python package dependencies on the Raspberry Pi and we decided to use Conda for some fixes.	Temporarily add the correct system path to obtain sudo command privileges, and go into the bashrc file and modify the path variables to uninstall conda from the Raspberry Pi.
Tried installing PyQt5 on the Raspberry Pi for our final GUI but the usual pip3 method does not work on the Raspberry Pi as it is a 32-bit ARM architecture and no other alternatives were found.	Utilized the Tkinter module library instead of the PyQt5 library for our final GUI for the Restaurant Menu.

Table 18: Challenges faced and solutions implemented during the final integration pipeline

8.5 Conclusion

We created a Github repository to store all of our source codes and documents, for the ease of sharing these files with each other. This is where we learnt about version control, which is important to avoid conflicting files. All programming exercises are done via Visual Studio Code, which is a great application to work with Python scripts. Some of us have not worked in a Linux environment prior, and we took some time getting used to the command-line interface. Working with environment variables and paths was also a challenge.

As much as complete parallelization was wished for, some parts still needed to be serialised. But this helped us work together as a group to help meet each other's needs. Our time was maximised as we are able to work independently during our flexible hours, and together during the group meetings on Wednesday. This enabled us to discuss our findings and progress for the week so that we are able to actively contribute in meetings.

In a nutshell, this Design and Innovation Project (DIP) on Smart Touchless control with Millimeter-Wave Radar Sensor and Artificial Intelligence has been an eye-opener for all of us. We have definitely gained many transferable skills which would be useful for our future, both technical and soft skills. We were also able to learn other skills not defined in our scope by extending it with some additions, in particular our two user interfaces for Gesture Extraction and Menu Application. We welcome anyone who would like to improve our dataset and techniques used to increase the robustness and use cases for our project.

Acknowledgement

We would like to acknowledge the commitment that our supervisor Prof. Lu Yilong has shown throughout this project. His flexibility in timing to approach him with any difficulties or questions enabled us to fully understand tough radar concepts which were very foreign to us. His suggestions were also very enlightening and helped us see the project in a bigger picture. We also want to thank him, for providing some sample references in MATLAB for us to port into Python. Finally, we would like to acknowledge the funding from Nanyang Technological University - School of Electrical and Electronic Engineering for this design and innovation project.

References

[1] N. Magrofuoco, J.-L. Pérez-Medina, P. Roselli, J. Vanderdonckt, and S. Villarreal, “Eliciting Contact-Based and Contactless Gestures With Radar-Based Sensors,” *IEEE Access*, vol. 7, pp. 176982–176997, 2019, doi: 10.1109/ACCESS.2019.2951349.

[2] “What Is Touchless Technology? | No-Touch Visitor Management System.” https://www.greetly.com/blog/what-is-touchless-technology (accessed Nov. 05, 2021).

[3] “Gestures.pdf.” https://web.mst.edu/~toast/docs/Gestures.pdf (accessed Nov. 06, 2021).

[4] D. Sarma and M. K. Bhuyan, “Methods, Databases and Recent Advancement of Vision-Based Hand Gesture Recognition for HCI Systems: A Review,” *SN Comput Sci*, vol. 2, no. 6, p. 436, 2021, doi: 10.1007/s42979-021-00827-x.

[5] “TurkVEChapter.pdf.” https://sites.cs.ucsb.edu/~mturk/pubs/TurkVEChapter.pdf (accessed Nov. 06, 2021).

[6] “The territorial impact of COVID-19: Managing the crisis across levels of government.” https://www.oecd.org/coronavirus/policy-responses/the-territorial-impact-of-covid-19-managing-the-crisis-across-levels-of-government-d3e314e1/ (accessed Nov. 06, 2021).

[7] “Is the future of touchscreens touchless?,” *IT PRO*. https://www.itpro.co.uk/hardware/357609/is-the-future-of-touchscreens-touchless (accessed Nov. 06, 2021).

[8] J. Ryde and N. Hillier, “Performance of laser and radar ranging devices in adverse environmental conditions,” *J. Field Robotics*, vol. 26, pp. 712–727, Sep. 2009, doi: 10.1002/rob.20310.

[9] S. Song, B. Kim, S. Kim, and J. Lee, “Foot Gesture Recognition Using High-Compression Radar Signature Image and Deep Learning,” *Sensors (Basel)*, vol. 21, no. 11, p. 3937, Jun. 2021, doi: 10.3390/s21113937.

[10] S. Ahmed, K. D. Kallu, S. Ahmed, and S. H. Cho, “Hand Gestures Recognition Using Radar Sensors for Human-Computer-Interaction: A Review,” *Remote Sensing*, vol. 13, no. 3, p. 527, Feb. 2021, doi: 10.3390/rs13030527.

[11] A. Alzogaiby, “Using Micro-Doppler Radar Signals for Human Gait Detection,” p. 98.

[12] E. Hayashi *et al.*, “RadarNet: Efficient Gesture Recognition Technique Utilizing a Miniature Radar Sensor,” in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, Yokohama Japan, May 2021, pp. 1–14. doi: 10.1145/3411764.3445367.

[13] K. Li, J. Cheng, Q. Zhang, and J. Liu, “Hand Gesture Tracking and Recognition based Human-Computer Interaction System and Its Applications,” in *2018 IEEE International Conference on Information and Automation (ICIA)*, Aug. 2018, pp. 667–672. doi: 10.1109/ICInfA.2018.8812508.

[14] Infineon Technologies AG, “Infineon-Radar vs PIR BGT24LTR11-PI-v01_00-EN.pdf.” https://www.infineon.com/dgdl/Infineon-Radar%20vs%20PIR%20BGT24LTR11-PI-v01_00-EN.pdf?fileId=5546d462576f34750157d2b1d6d27370 (accessed Nov. 05, 2021).

[15] “Setting up your XM112 — acconeer-python-exploration documentation.” https://acconeer-python-exploration.readthedocs.io/en/latest/evk_setup/xm112.html (accessed Nov. 04, 2021).

[16] “Working with the GUI — acconeer-python-exploration documentation.” https://acconeer-python-exploration.readthedocs.io/en/latest/gui/index.html (accessed Nov. 03, 2021).

[17] “Radar sensor introduction — acconeer-python-exploration documentation.” https://acconeer-python-exploration.readthedocs.io/en/latest/sensor_introduction.html (accessed Nov. 06, 2021).

[18] “Simple audio recognition: Recognizing keywords | TensorFlow Core,” *TensorFlow*. https://www.tensorflow.org/tutorials/audio/simple_audio (accessed Nov. 03, 2021).

[19] “numpy.savez — NumPy v1.21 Manual.” https://numpy.org/doc/stable/reference/generated/numpy.savez.html (accessed Nov. 03, 2021).

[20] G. Boultadakis, K. Skrapas, and P. Frangos, “Time-Frequency Analysis of Radar Signals.”

[21] “mfcc.” https://musicinformationretrieval.com/mfcc.html (accessed Nov. 04, 2021).

[22] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, “Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14413–14423, Dec. 2020, doi: 10.1109/TVT.2020.3034800.

[23] K. Patel, “Convolution Neural Networks — A Beginner’s Guide [Implementing a MNIST Hand-written Digit…,” *Medium*, Oct. 18, 2020. https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022 (accessed Nov. 04, 2021).

[24] “Max-pooling / Pooling - Computer Science Wiki.” https://computersciencewiki.org/index.php/Max-pooling_/_Pooling (accessed Nov. 04, 2021).

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, 2012, vol. 25. Accessed: Nov. 04, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[26] “Real Doppler RAD-DAR database.” https://kaggle.com/iroldan/real-doppler-raddar-database (accessed Nov. 03, 2021).

[27] R. Khandelwal, “A Basic Introduction to TensorFlow Lite,” *Medium*, Jun. 15, 2020. https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292 (accessed Nov. 03, 2021).

[28] “TensorFlow Lite | ML for Mobile and Edge Devices,” *TensorFlow*. https://www.tensorflow.org/lite (accessed Nov. 03, 2021).

[29] “Raspberry Pi 4 vs Raspberry Pi 3B+,” *The MagPi magazine*. https://magpi.raspberrypi.com/articles/raspberry-pi-4-vs-raspberry-pi-3b-plus (accessed Nov. 06, 2021).

[30] “How to give your Raspberry Pi a Static IP Address - UPDATE,” *The Pi Hut*. https://thepihut.com/blogs/raspberry-pi-tutorials/how-to-give-your-raspberry-pi-a-static-ip-address-update (accessed Nov. 04, 2021).

[31] “Fingerprinting for Solving A/V Synchronization Issues within Broadcast Environments.” https://ieeexplore.ieee.org.remotexs.ntu.edu.sg/document/7269993 (accessed Nov. 06, 2021).

[32] A. K. Mishra and B. Mulgrew, “Radar Signal Classification Using Pca-Based Features,” in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, May 2006, vol. 3, p. III–III. doi: 10.1109/ICASSP.2006.1660851.

Appendix A - Project Members Information

	Name	Project contributions	Report Contribution
1	Goh Jun De	Flashing the radar sensor, Testing different types of gestures where depth measurements are of concern, Setup static IP configuration for the Raspberry Pi, Hyperparameter tuning for the radar sensor, Building physical casing for the radar sensor and data collection	Sections 1, 2, 6.1-6.2, 8.1
2	Lee Wai Yeong	Machine learning & deep learning pipeline, Visualisation of ML/DL models, Frontend for GUI (Data collection, Restaurant menu), Preprocessing exploration and visualisation, Debugging of Linux environment paths and variables	Sections 4, 6.5-6.7, 7, 8
3	Philip Lee Hann Yung	Group leader, Planning the project pipeline, Gesture extractor GUI, Data preprocessing, Run-time implementation and algorithm optimizations, Menu application, Python scripting and Jupyter notebook exploration for all components	Sections 3, 6.3-6.7, 7, Acknowledgement, References, Appendix B, Cohesiveness
4	Tan Wen Han Davis	Building physical casing for the radar sensor. Finding out the limitation the Acconeer can handle by playing with various parameters and observing differences it can make. Finding the optimal setting that is suited for this project. Comparing other gesture reading devices in the market against radar. Matlab analysis prior to machine learning.	Sections 1, 2, 6.1-6.2, 8.1
5	Thum Yi Wen	Treasurer, Planning budget list for the project, Flashing the radar sensor, Exploring Machine Learning models, Setup Raspberry Pi and installing package dependencies in Raspberry Pi	Sections 5, 6.2, 6.4, 6.7

Appendix B

All codes and files relevant to the project may be found in our GitHub repository:
https://github.com/leephilipx/smart-touchless-control-radar

In particular, our `final-realtime-hybrid.py` script is shown here to showcase our API functions, which wrapped up all sub-blocks in our ML pipeline into their respective `radar.py`, `preprocess.py` and `ml.py` scripts stored in the `master` directory.

```
import warnings
warnings.filterwarnings("ignore")

from master import radar, preprocess, ml
from time import sleep
import numpy as np
import argparse
from pyautogui import press

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description='DIP E047: Final Real-time Gesture Prediction')
    parser.add_argument('-rpi', '--rpi', action='store_true', help='Raspberry Pi mode to fix port')
    parser.add_argument('-kb', '--kb', action='store_true', help='Keyboard presses')
    args = parser.parse_args()

    model_dl = ml.DeepLearningModel(model_path='stft-final.tflite')
    model_ml = ml.MachineLearningModel(model_path='log-reg.pkl')
    X_shape, Y_shape, class_labels = radar.getDatasetInfo(source_dir='2021_10_20_data_new_gestures')

    radarSensor = radar.AcconeerSensorLive(config_path='sensor_configs_final.json')
    port = '/dev/ttyUSB0' if args.rpi else radarSensor.autoconnect_serial_port()
    radarSensor.connect_serial(port)
    radarSensor.start_session()

    X_frame = np.zeros((1, 80, X_shape[2]), dtype=np.complex128)
    preds_threshold = 0.6
    frame_buffer = 0
    consensus_buffer = 0 if args.rpi else 2
    y_probs_buffer = np.zeros((len(class_labels), ))
    np.set_printoptions(suppress=True, precision=3)

    while True:

        try:
            X_frame[:, :-1, :] = X_frame[:, 1:, :]
            X_frame[:, -1, :] = np.expand_dims(radarSensor.get_next(), axis=0)
            frame_buffer += 1

            if frame_buffer == 80:

                if args.kb: press('.')
                x_center = preprocess.get_frame_center(X_frame, consensus_buffer)

                for offset in range(-consensus_buffer, consensus_buffer+1):
                    X_input = X_frame[:, x_center+offset-32:x_center+offset+32, :]
                    X_input = preprocess.get_batch(X_input, mode='stft')
                    X_input_dl = preprocess.reshape_features(X_input, type='dl')
                    X_input_ml = preprocess.reshape_features(X_input, type='ml')
                    y_probs_dl = model_dl.predict_tflite(X_input_dl)
                    y_probs_ml = model_ml.predict_proba(X_input_ml)
                    y_probs_buffer = y_probs_buffer + y_probs_dl + y_probs_ml
```

```
            frame_buffer = 0
            y_consensus = np.argmax(y_probs_buffer)
            if y_probs_buffer.max() > (preds_threshold * (2*consensus_buffer+1) * 2):
                print(f'center={x_center};  {y_consensus} {class_labels[y_consensus].ljust(12)}',\
                    np.squeeze(y_probs_buffer))
                if args.kb: press(str(y_consensus))
            else:
                print(f'center={x_center};  - {"n/a".ljust(12)}', np.squeeze(y_probs_buffer))
                if args.kb: press('/')
            y_probs_buffer = np.zeros((len(class_labels), ))

    except KeyboardInterrupt:
        print('>> KeyboardInterrupt caught! Exiting ...')
        break

    except Exception as e:
        try:
            print(f'\n>> Exception caught! {e}')
            print('>> Connection to sensor failed, trying again in 5 seconds ...')
            sleep(5)
            radarSensor.stop_session(verbose=False)
            radarSensor.disconnect_serial(verbose=False)
            port = '/dev/ttyUSB0' if args.rpi else radarSensor.autoconnect_serial_port()
            radarSensor.connect_serial(port)
            radarSensor.start_session()
        except KeyboardInterrupt:
            print('>> KeyboardInterrupt caught! Exiting ...')
            break
```