Delft University of Technology
Master of Science Thesis in Computer and Embedded Systems Engineering

IAmMuse: A signal-processing-based method to estimate arm positions with mmWave radar

Brendan Rudolf Mesters

IAmMuse: A signal-processing-based method to estimate arm positions with mmWave radar

Master of Science Thesis in Computer and Embedded Systems Engineering

Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Brendan Rudolf Mesters

December 9th 2025

Author
Brendan Rudolf Mesters
Title
IAmMuse: A signal-processing-based method to estimate arm positions with mmWave radar
MSc Presentation Date

December 16th 2025

Graduation Committee

Marco Zúñiga Zamalloa	Delft University of Technology
Koen Langendoen	Delft University of Technology

Abstract

The broad adoption of integrated computing systems through the Internet of Things has led to a need for seamless, low-latency interfaces. Deploying these installations in public spaces, such as gaming areas in airports, presents unique challenges. Traditional input modalities are ill-suited for these environments. Cameras compromise user privacy, and wearables are susceptible to theft, damage or loss. Millimetre wave (mmWave) radars are an ideal sensor for such an interface, as they can operate through non-conductive materials and independently of lighting conditions. Most human sensing solutions for mmWave radars use Deep Learning models, which rely on large data sets for training and have a high computational overhead, limiting their feasibility on resource-constrained edge devices. To address these limitations, we present *IAmMuse*, a lightweight, signal-processing-based interpretation pipeline for human sensing using mmWave radar technology. We filter and enhance the raw point cloud data, using spatio-temporal density information, as well as kinematic context acquired through a few-shot online learning step. IAmMuse uses this pre-processed data to generate a stabilised prediction, classifying the user's arm position. We implemented a musical system, controlled through these predictions, as an example application for the technology. The user selects musical notes by moving their arms to either a *low*, *middle*, or *high* position, similar to a conductor. To assess the efficacy of this method, we present a comparative analysis with a State-of-the-Art Human Pose Estimation model. This comparison shows that IAmMuse achieves a classification accuracy 50% higher than the State-of-the-Art model, while using less than 1% of the training data. This thesis validates the viability of non-deep-learning-based interpretation algorithms for human sensing with mmWave radars through a fully functional prototype.

Preface

I want to thank a few people by name for their help with this thesis.

Firstly, I want to thank Girish Vaidya, who was my daily co-advisor up until the summer, for his patience and technical and academic assistance.

I want to thank Marco Zuñiga, my thesis advisor, for his assistance during the thesis, but in particular for his helpfulness and friendliness towards me during the most stressful parts of this thesis.

Lastly, I want to thank my parents, Erika and Jack, for supporting me in so many ways throughout my whole education (and throughout my whole life).

Brendan Rudolf Mesters

Delft, The Netherlands
9th December 2025

AI Disclosure

During the writing of this thesis, generative AI (Gemini 3 Pro) has been used to proofread the paper. I reviewed all AI-generated suggestions before dismissing or implementing them, and I take final responsibility for the contents of this thesis. All AI-generated suggestions have been reviewed
The generative AI was used to:

- provide feedback on the tone of the text.
- point out logical leaps in explanations.
- detect grammar and spelling errors.
- suggest terminology, e.g., suggesting to change "A small, real-time initial training step" to "An online few-shot learning approach".
- solve LaTeX-related formatting issues.

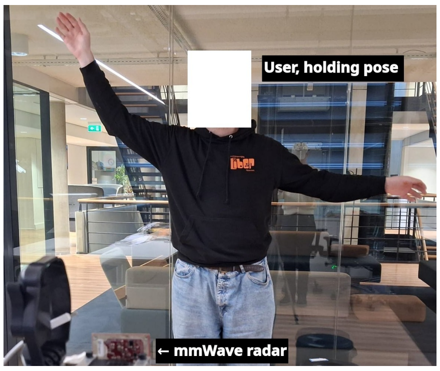
The generative AI was **not used to**:

- write any sections.
- rewrite entire sentences or paragraphs.
- generate figures.

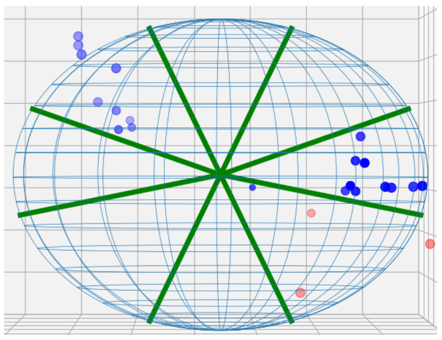
Contents

Chapter 1

Introduction



(a) **User using the system**



(b) **Visualisation of the point cloud and internal state**

Figure 1.1: **IAmMuse in use, with the user using the system (a) and a visualisation of the radar output (b)**

1.1 Motivation

The landscape of Human-Computer Interactions (HCI) is shifting as our computers are becoming more powerful and more integrated. Standalone systems, such as the home computer, are becoming increasingly rare in favour of integrated Internet of Things (IoT) systems. Using traditional input methods, such as a mouse, keyboard or touchscreen, to control these IoT environments is intrusive and impractical. As such, these IoT environments require new and alternative input methods. Smartphone apps and voice-controlled systems do fill some of this demand, but a broader set of input modalities would be beneficial.

A sufficiently unintrusive interface would not require any physical device to interact with it. Simple arm movements should be sufficient to control the underlying system. These types of Human Pose Estimation (HPE) systems have already been explored, but primarily used cameras for the recognition task. Cameras have a few problems; firstly, they are dependent on advantageous lighting conditions, and suffer in under- or over-illuminated environments.

Secondly, cameras are privacy-intrusive, which could cause issues with new privacy-focused regulations, such as the GDPR [30]. Millimetre wave (mmWave) radars, on the other hand, offer a privacy-preserving sensing method that is well-suited for this exact purpose.

Limitations in the State of the Art

Research into HPE using mmWave radar point clouds already exists; however, these systems are predominantly built with a Deep Learning (DL) framework. MARS [2], mmDiff [10], and mmPose [32] are prime examples of state-of-the-art (SoA) DL models for HPE. Although these DL models have an exceptional ability to generalise over unstructured data, they also suffer from some limitations, namely their computational intensity, functional opaqueness and the difficulty of tuning [6, 15]. Interpretation methods built on a more traditional signal processing basis would be able to run on more affordable hardware. In addition, slight scenario changes would only incur a retuning in more traditional systems, as opposed to the expensive retraining of DL systems.

Applications

To effectively validate the capabilities of our proposed architecture, we implement it within a specific application context. HPE solutions could fill an existing gap in low-latency and high-precision interfaces for IoT systems; thus, our chosen application should require those features. Existing solutions, such as voice assistants, fail to address this niche, as they do not provide immediate feedback to the user due to the time it takes to interpret the user's request. We require a suitable low-latency benchmark to validate the viability of any specific HPE interface. A well-known class of interactive systems which rely on low-latency feedback is musical instruments. In addition to their low-latency requirement, musical instruments also provide instant auditory feedback to the user. Therefore, we identified musical applications as an ideal benchmark for HPE-powered computer interfaces. A video of the final system in action, with the musical interface, can be found here [26].

1.2 Research Challenge

Using mmWave radars over alternative sensors, such as cameras, introduces unique problems. Unlike cameras, which provide dense, rich, and low-noise data, mmWave radar point clouds are sparse (20-30 points per frame at 10 frames per second) and noisy. Some frames are also exceptionally sparse (~5 points) or contain data only of either the right or left half of the user, which makes those frames less valuable. This sparseness poses a significant challenge when it comes to *real-time* systems with a *low-latency* requirement. In order to have a tight response time, the system needs to generate an accurate response in a single frame or a few frames at most, which becomes challenging on partially incomplete frames.

The avoidance of DL methods poses another challenge, as they do provide an impressive inherent ability to generalise, even if this is achieved through a *black box*. The move to an algorithmic solution requires domain-specific prior knowledge to parse the sparse point cloud and infer the actual events that generated the input. This is a challenging problem, certainly considering the poor quality of mmWave point clouds.

However, the problem that DL models often try to solve, creating a full-body skeletal estimation, is often superfluous and computationally wasteful. We therefore explicitly constrained the problem space for this thesis to avoid this additional latency and any dependency on large data sets. Instead of a skeletal estimation, we perform an *arm angle estimation*, classifying the user's arms into discrete angular zones (see fig. 1.2). This prioritises focus on a fast response time with low overhead while allowing for quick retuning, as opposed to costly retraining. Therefore, our final challenge is to reach competitive efficacy in the proposed problem space, with State-of-the-Art mmWave radar HPE solutions.

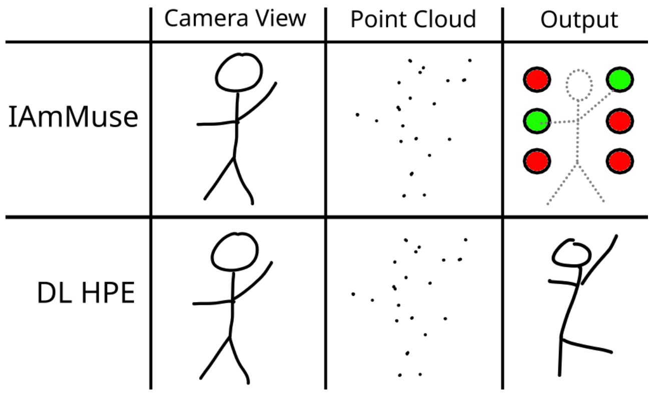


Figure 1.2: **Differences between IAmMuse and Deep Learning HPE**

1.3 Contributions

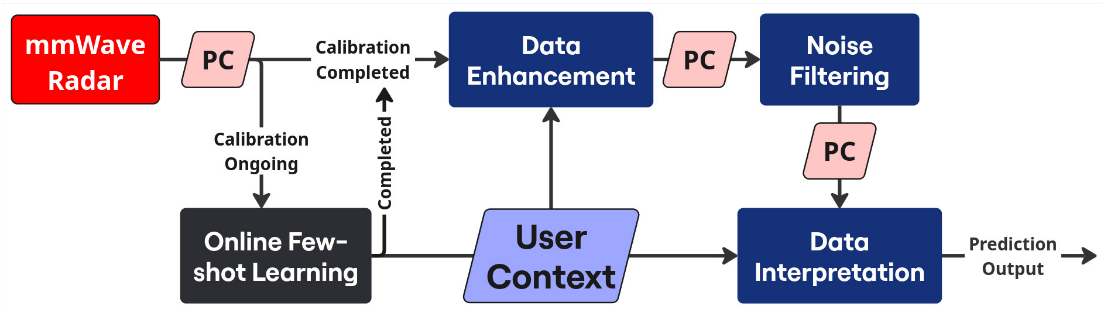


Figure 1.3: **Diagram of the interpretation framework.**

This thesis makes three distinct contributions to the field of mmWave sensing for interactive interfaces: a theoretical interpretation framework, a real-time software implementation of this framework, and a comparative evaluation against a state-of-the-art Deep Learning model.

1. Theoretical Interpretation Framework for Sparse Point Cloud Interpretation
Our first contribution is a framework for the cleaning, enhancing and interpretation of sparse mmWave radar point clouds.
This framework comprises:

- A *user context gathering* system, using a *few-shot online learning* approach. This system provides information about the position of the user's sternum and their arm span.
- A *spatio-temporal filter* for point clouds, which uses the local density and the prior frames to filter out both *dense single-frame noise spikes*, as well as more *consistent low-density noise*.
- A *data enhancement* technique to identify points produced by the user's hands. This uses the user context to add an extra dimension of information to the point cloud.
- A *stabilising angle estimation* method, which uses the context gathered by the data enhancement system to estimate which discrete arm angle range the user's arms are in. Stabilisation methods are employed to counteract jitter and noise spikes.

2. Real-Time Software Implementation: 'IAmMuse'
Our second contribution is a multi-threaded full pipeline implementation of this framework, called 'IAmMuse'. It is designed with robustness and fast response times in mind, to be able to run on more resource-constrained hardware.
For this contribution, we present:

- A custom high-performance driver for the radar communication protocol, implemented in Rust. This development yields a platform-agnostic solution, resolving the compatibility limitations of the vendor-supplied driver while providing a native, low-latency interface, enabling seamless integration into the rest of the pipeline.
- A fully asynchronous and multi-threaded implementation of the framework shown in fig. 1.3. This architecture ensures that data processing can start as soon as a data packet arrives from the mmWave radar. This enables optimal hardware utilisation, even on more resource-constrained hardware.
- A custom visualizer in Python, providing a direct view into the system's internal state. This is an essential tool, allowing us to tune and monitor the system through visual verification of many of the pipeline steps.

3. A Comparative Evaluation Against the State of the Art
Our third contribution is a quantitative comparison with an SoA HPE interpretation system. For this, we compared IAmMuse to the DL model MARS [2], to assess the efficacy of the IAmMuse framework.
The findings from this benchmark are that:

- IAmMuse achieved an accuracy 50% higher than the best-performing MARS model, for correct state predictions, on our evaluation data set. The output of MARS was often unstable, rapidly switching between different predictions.

- If MARS is trained using the data from the online few-shot training, then the performance is only slightly better than random guessing. In this scenario, IAmMuse had an accuracy 4 times higher than MARS.
- The training data for MARS also had to match the evaluation data very closely; using a different mmWave HPE dataset, for example, produced a model that was very prone to bias. These models predicted the same state, regardless of the input data, more than 90% of the time.

Thesis Outline

Chapter 2 presents the theoretical foundations required to understand the thesis and its challenges, as well as the current state of research in the field. Chapter 3 provides a detailed discussion of the interpretation framework. Chapter 4 shows the testing setup, lays out the characteristics of the collected data, and the baseline comparisons. Chapter 5 shows the results of this comparison, and lastly Chapter 6 will conclude this thesis and give some future recommendations.

Chapter 2

Background

This thesis presents an *explainable* and *tunable* algorithmic interpretation method for the point clouds generated by millimetre wave (mmWave) radars. This method will interpret data for human pose estimation (HPE), a task which has thus far predominantly been explored with the use of Deep Learning (DL) systems. This chapter establishes the necessary foundations for the rest of the paper.

2.1 Millimetre Wave Radar

Millimetre Wave Radars (mmWave radar) are a type of sensor which has been discussed for many decades already, with early research dating back to the 1970s. Commercial usage dates back to the 1990s for the telecommunication sector, with the automotive industry starting to use it at the turn of the century [25]. In recent years, mmWave radars have been used in a broad number of sensing applications, from human tracking and activity recognition, through to object inspection and classification [33, Fig. 2].

This recent interest in mmWave-based sensing solutions over some of the alternative methods, such as LiDAR and camera-based sensing methods, is due to some of the inherent advantages present in mmWave-based systems. Cameras, for example, provide a detailed, high-resolution image, but they are susceptible to environmental changes in the light intensity. mmWave radars do not suffer from this limitation, since they are an active sensor, meaning that they observe the reflection of a signal which has been sent out by the sensor itself. Furthermore, most cameras do not include any depth information; they produce a 2-D projection of a scene. mmWave radars, on the other hand, provide a 3-D point cloud of the scene, which does have this range data. mmWave radars share many of the previously mentioned attributes with LiDAR and radio-frequency radars. The main differences between LiDAR and mmWave radars are the cost and point density. Lidar is significantly more expensive, but also produces significantly denser point clouds, see fig. 2.1. An essential difference between LiDAR and mmWave radars, however, is cost. The high cost of LiDAR is a serious barrier when it comes to broad adoption in IoT sensors. Other radio frequency radars suffer from lower accuracy due to the wavelength of the transmitted signal, which is longer than that of mmWave radars [35]. This results in a lower resolution As a result, mmWave radars are a cost-effective sensor for this job.

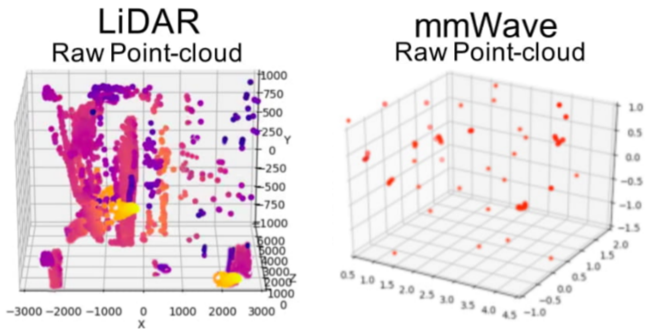


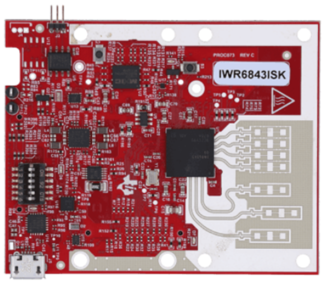
Figure 2.1: **Point density difference between LiDAR (left) and mmWave radar (right). Adapted from [1, fig. 2] (cropped and rearranged w.r.t. original figure)**

To achieve this, a mmWave radar transmits a millimetre wave of continuously increasing frequency, and by observing the backscatter, it can infer information about the scene. Since the frequency is continuously increasing, we know that the distance to the object which caused the backscatter is proportional to the difference in frequency between the outgoing and incoming signals. An alternative name for these sensors is thus a *Frequency Modulated Continuous Wave* (FMCW) sensor. The mmWave radar uses a component called a *mixer* to calculate the difference between the ongoing and outgoing signals, which takes two signals as inputs and provides the difference in frequency as well as the difference in phase between those two signals. This frequency difference holds information about the macro-scale distance, while the phase difference gives information on the micro-scale movements of an object, which is very useful in some specific applications.

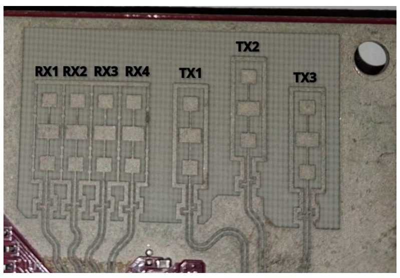
The mmWave radar has multiple receivers and multiple senders, which allows the sensor to calculate an elevation and an azimuth for each object. These different receivers are spaced half a wavelength apart from each other. This change in position means that the signal will have travelled a different distance, measurable in less than one wavelength, to each of the receivers. This phase difference can be used to calculate a difference in distance to the object in question, which in turn can be used to calculate the angle of arrival. The specific layout of Transmitters (tx) and Receivers (rx) on the mmWave radar (see fig. 2.2b) means that the mmWave radar can calculate the elevation, azimuth, distance, and Doppler information (which tells us the velocity towards or away from the sensor) of objects in the scene, which is given to the user as a 4-D Point cloud. A lot of technical details have been omitted in this explanation, as they are not very important to the workings of the thesis. For more detail, the mmWave Radar video course by Texas Instruments[17], or the introductory article by Iovescu and Rao [18], are a good starting point.

2.1.1 Data Characteristics

The data generated by these mmWave radars has some key characteristics. We already mentioned the ability to sense sub-millimetre movements using



(a) **Top down view**

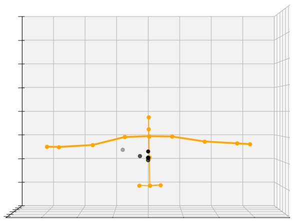


(b) **Antenna layout**

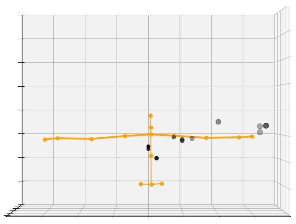
Figure 2.2: **The IWR6843ISK mmWave radar**

phase information, which is crucial for specific applications, such as vital sign monitoring [34] and acoustic reconstruction [23]. Another unique feature of mmWave radars is their ability to penetrate, and thus look through, specific materials. These materials include fabrics, such as cotton and leather [24], but also include solid materials, such as plastic and wood [20]. This allows for seamless installation behind otherwise solid structures (such as a plastic case), and can help in the detection of hidden objects and contraband.

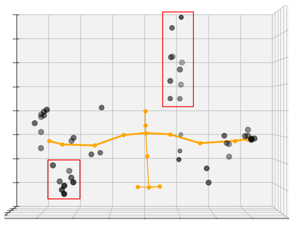
mmWave radar is also able to output several types of data, using different firmwares. The majority of these firmwares, however, produce point clouds as output, which are what this thesis aims to interpret. These point clouds generated by the mmWave radar are relatively sparse, generating around 10 frames per second, usually with no more than 30 points per frame, dropping to unusably low levels in some frames, see fig. 2.3a. Besides just sparse frames, the mmWave radar also regularly (~ 10%) produces *partial frames* where a part of the point cloud seems to be missing, see fig. 2.3b. The point clouds produced by the mmWave radar are not only sparse, but also quite noisy [5]. This noise occurs in two distinct manners: low-density consistent *background noise*, and *high-density single-frame* noise spikes (see fig. 2.3c). The above-mentioned issues mean that the consistency and stability across frames are very low, which poses a challenge to data interpretation.



(a) **Sparse point cloud**



(b) **partial frame, with only points on the right side**



(c) **Dense point cloud with noise (in the red boxes)**

Figure 2.3: **Difference between point cloud frames**

Setups which negate one of these issues, such as the low density of the point clouds, do exist, but they suffer in other aspects, such as creating significantly more noise points. Different firmwares have to make concessions with regard to point-density, accuracy and the signal-to-noise ratio. Furthermore, external factors also affect the output quality of the point cloud, such as the materials in the frame or the characteristics of the room in which the radar operates.

2.2 Human Pose Estimation

Human Pose Estimation (**HPE**) is a broad term covering systems that estimate a human's pose, usually in the form of a set of 3D coordinates. A well-known example of an HPE system is the Microsoft Kinect system, a camera-based movement tracker originally introduced for the Xbox game console. Though HPE can cover a broad number of applications, many of them aim to construct a *skeletal estimation*, a set of essential keypoints to represent a human skeleton. Microsoft Kinect, in particular, generates a skeletal estimate composed of 25 points, as depicted in fig. 2.4, and is used as a source of ground truth for several other HPE systems.

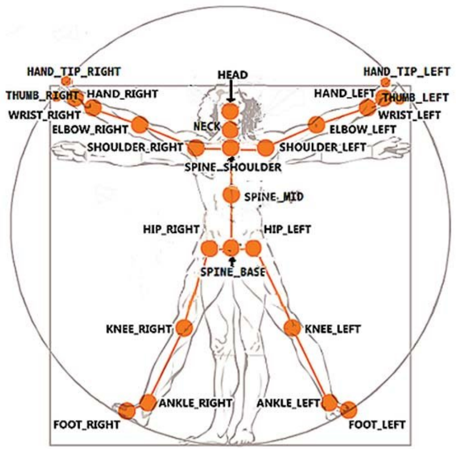


Figure 2.4: **Kinect's 25-point skeletal estimation [14]**

HPE methods have become increasingly prevalent as computer systems have become more integrated into our everyday environment, providing a non-invasive way to interface with them. These HPE systems can be used for a variety of purposes; the Kinect, for example, was originally built for gaming. Other fields have also started using HPE methods. Activity recognition, through kinematic skeletal estimations, can enhance existing security systems, allowing them to detect security threats autonomously. Action recognition and tracking methods can control IoT systems, and HPE systems have even seen use in automated sign language recognition, assisted living spaces, and driver assistance systems, as shown by Munea et al. in their review paper [28].

HPE systems exist for various input modalities. Vision-based methods [22] often use normal cameras or RGB-D cameras to construct pose estimations. Wearable-based methods [27] rely on sensors worn by the user to construct their estimations. And mmWave-radar-based methods often rely on Deep Learning methods to transform point clouds into post estimations.

2.3 Related Work

The majority of mmWave HPE methods utilise deep learning for its ability to generalise over unstructured data. MARS [2], for example, utilises a CNN to predict a set of *3D joint coordinates*, using a Kinect 2 skeletal estimation as the ground truth during training. This method uses the mmWave radar point cloud to output predicted skeletal estimations on a frame-by-frame basis. mmPose [32] also uses a CNN to produce a skeletal estimation from the mmWave radar. However, where MARS uses the point cloud directly, mmPose uses the 2D *range-azimuth* and *range-elevation* projection to predict the human pose, to mitigate the data sparsity issue. Lastly, mmDiff [10] uses a constrained diffusion model to perform the human pose estimation. To prevent unrealistic hallucinations, they explicitly constrain the diffusion model with kinematic and temporal bounds.

While the mmWave HPE State-of-the-Art solutions all leverage deep learning methods, several other point cloud and mmWave radar-based interpretation systems instead opt to employ more traditional machine learning and signal processing methods. DBSCAN [9] is a foundational clustering technique for point clouds that is used in many other data interpretation pipelines. In addition to the widely recognised DBSCAN, we also find exemplary domain-specific interpretation systems in the field of mmWave sensing. Li et al. achieved audio reconstruction [23], using classical signal processing techniques to extract vibrational information through the micro-doppler information of the mmWave radar. Similarly, Poux and Billen [31] created a system which uses a voxel-based segmentation pipeline for geometric labelling.

Lastly, since this thesis uses a musical system as a benchmark application to test the efficacy of our final mmWave-radar-based interface, we consider similar, sensor-driven, musical applications. A particularly relevant domain to consider is that of *full body* musical interfaces, given that our system uses arm tracking to control the musical output. One such system is Somi-1 [29], which uses motion sensors to control music with dynamic movement. MiMu gloves [13] is another similar system that needs to attach to the user's hands, which measures finger position as well as hand motion. Two existing systems already use mmWave radars for musical expression, which both work through Deep Learning: *WaveTune* [19], which uses gestures to select a combination of prebuilt beat-patterns; and *O Soli Mio* [4], which allows you to control specific musical effects, or switch between different samples, by performing gestures above a sensor. IAmMuse, on the other hand, aims to provide a more traditional interface, allowing the user to play specific notes through their arm movement.

Chapter 3

Tracking Method

The main goal of the thesis is to find a method for predicting *where* a user's arms are located, from a point cloud generated by a millimetre wave radar. More specifically, it should tell us in what general **zone** the arm is currently present, this can be either: "low", where the user's arm points at the ground; "middle", where the user's arm points out to the side; or "high", where the user's arm points up to the sky. These regions exist both for the left and for the right arm, and thus there are six different *zones* which the system tries to predict, see fig. 3.1. There are, of course, some physical bounds on this model. A person can, for example, only have each arm in *one zone*, that means the system will always output **one** *left zone* (left low, left middle, or left high) and **one** *right zone*.

To facilitate this zone detection, a standalone pipeline was built to handle both the communication with the FMCW and packet decoding, as well as the filtering, labelling, and interpretation of the data. This thesis discusses the latter part of this pipeline, which implements a noise filtering system and our interpretation algorithm. Section 3.2 discusses the data enhancement method used by IAmMuse, and section 3.3 presents IAmMuse's interpretation algorithm. The final pipeline for IAmMuse only arrived after a lot of different approaches had been explored and proved unfruitful.

Most of the explored approaches rely on a small *online few-shot training step*, at times referred to as a *calibration* step, to avoid confusion later on in the report. This training step provides context about the user, which is used during the *arm angle estimation*. Interpretation of mmWave data without this context is highly challenging due to the low point density, along with the noise percentage. This reliance on training data could be remedied in the future by utilising a greater understanding of these interpretation methods. However, this thesis still uses training data as it aims to create that *understanding of interpretation methods*.

3.1 Data filtering

One issue present with the mmWave Sensors is the fact that they are susceptible to both environmental noise and seemingly random noise. There are two distinct types of noise which we encounter. Low density *background noise*, which consists of a few random points scattered throughout a frame, and high density *noise spikes*, which only occur in a single frame. These noise spikes often

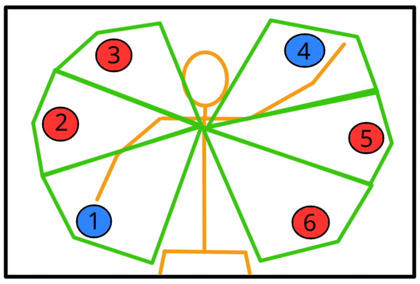


Figure 3.1: **Arm zone visualization**

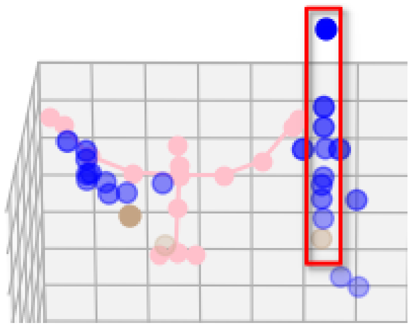


Figure 3.2: **A noise spike in the data (right-hand side of image)**

occur in *streaks*, lines of points through 3d space. These streaks are not very dense spatially and appear in only a single frame, see fig. 3.2. These noise points roughly take the shape of a line, though the points never line up exactly. The nature of this noise, combined with the unstable signal output by the mmWave radar, makes it challenging to filter out noise.

Many sophisticated systems for point cloud filtering already exist [9, 36]. However, these existing systems expect a high point density, since they are built for Lidar, which makes them unsuitable for mmWave radar point clouds. Even papers describing methods for "sparse point clouds" reason about thousands of points at a time, as opposed to the dozens of points encountered in mmWave data. This low density distinguishes mmWave point cloud denoising as a separate challenge, evidenced by the purpose-built dataset by Farella et al. [11]

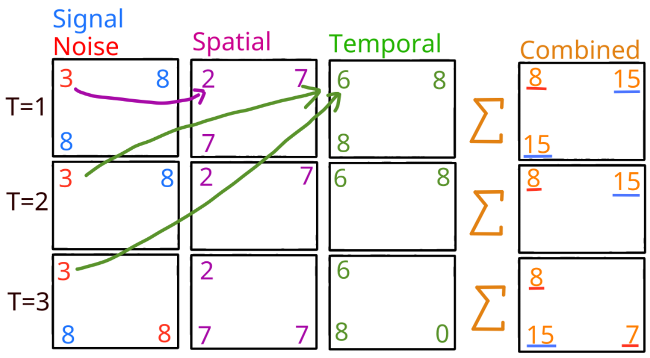


Figure 3.3: **Overview of the internal workings of the combined noise filter, using the spatial filter to filter out background noise (the red 3) and the temporal filter to filter out noise spikes (the red 8).**

For these reasons, we propose a novel noise filter for low-density mmWave point clouds. The primary objective of the filter is to label each point as either *signal* or *noise*. Figure 3.3 conceptually shows this process, with each row representing a different frame, and each column representing a different internal filtering step. The first column shows the input point cloud, where each quadrant represents a separate cluster with a number of points. This is a simplified version of the point clouds often encountered, where there are a few densely packed sets of points in distinct spatial regions. The second column shows the *spatial filter*, which considers the number of nearby points in **the same frame**. The third column shows the *temporal filter*, which considers the number of nearby points in **other frames**. Lastly, the fourth column combines these scores to filter out both ambient noise (clusters which consistently get a low score, e.g. 3) and noise spikes (clusters which are dense, but only present in a single frame). This is done by simply comparing the combined score to a pre-defined threshold; in this case, any threshold value between 9 and 14 would accurately filter the noise from the signal.

3.1.1 Spatial Filtering

The spatial filter is built on the idea that dense point clusters are likely to be signal. We give every point a score based on the weights of all points in the same frame, within a configurable distance (25cm). This point weight is assigned by the data enhancement system, described in section 3.2.3. This filter uses the fact that the encountered noise is not very spatially dense. This *density score* is compared with a (configurable) threshold, to determine if the point in question is *signal* or *noise*.

The main task of this spatial filter is to remove outliers, always let dense clusters pass, and quantify the significance of points in the context of the current frame. By constructing the *density score* from the weights of nearby points, we ensure that high-valued points propagate their importance. The spatial filter is effective against sparse noise, but struggles with dense, single-frame noise spikes. Mitigating these noise spikes requires a *temporal filter*.

3.1.2 Temporal Filtering

The main task of the *temporal filter* in IAmMuse is to provide stability *across* frames and to remove noise spikes. The temporal filter achieves this in a very similar way to the spatial filter; however, the temporal filter accumulates the weight of spatially nearby points in *previous frames* as opposed to in the same frame. A point's *temporal score* is thus defined as the cumulative weight of nearby (with a distance less than 20 cm) points in the three previous frames. Points in consistent clusters will thus be given a relatively high temporal score, while clusters that only occur in one frame will get a very low temporal score, even if they are very dense. This temporal score is then compared to a *temporal threshold* to determine if a point is *signal* or *noise*. In this way, the temporal filter can filter out any *inconsistent data*, such as sudden but dense noise spikes.

The temporal filter thus gives a low score to points in unstable and fleeting data, while giving high scores to points in temporally stable areas. The temporal filter will give higher scores than the spatial filter to points in consistent areas, since the temporal filter considers neighbours in multiple frames. Due to the inconsistent data issue of mmWave sensors (described in section 2.1.1), the temporal filter may miss significant amounts of data. That is why it is essential to use a combination of **both** the *temporal* and the *spatial* filter, in the form of a *combined filter*.

3.1.3 Combined Filtering and Tuning

IAmMuse uses a combination of the *spatial* and the *temporal* filters, so that it can get the benefits of both while mitigating their weaknesses. It does this by calculating both the *density score* and the *temporal score* for each point and combining them in accordance with a *combination factor*. Let S_{spat} and S_{temp} be the *spatial* and *temporal* scores, respectively. Let T_{spat} and T_{temp} be the *spatial* and *temporal* threshold respectively. We then introduce a **combination factor** F, where $0 \leq F \leq 1$, that determines how much we take into account each of the filters. A point is then considered to be *signal* if $F \times S_{spat} + (1 - F) \times S_{temp} > F \times T_{spat} + (1 - F) \times T_{temp}$. Otherwise, the point is considered to be *noise*.

The combined filter has five variables that can be tuned. The distance within which other points contribute to a score, for *both* the temporal and spatial filters. The threshold value for *both* the temporal and spatial filters. And the *combination factor*, which is used to combine the outputs of the two filters. It is crucial to have a clear strategy for how these variables will be tuned, since they all affect the output. Our tuning strategy first tunes the spatial filter such that it identifies most actual signal points correctly, even if some noise still comes through. Then, the temporal filter is tuned such that it labels most actual noise points correctly, even if some signal gets labelled as noise. Lastly, the combination factor is tuned to mitigate any noise spikes permitted by the spatial filter while avoiding the discarding of valid points by the temporal filter.

To tune the individual filters, you can only change the distance at which to consider other points or the score threshold. Increasing the threshold will increase the number of points that are labelled as noise, and decreasing it will do the opposite. Increasing the distance will increase the number of points that are labelled as signal, but it also decreases the importance of tightly coupled clusters. In this way, tuning the distance also affects how important the density of point clusters is to the output.

To tune the combination factor, we must ensure that the filters are already tuned individually. Then the combination factor starts at 0.5 during tuning, and gets tuned up if too many signal points are missed, and tuned down if too much noise gets through. This gives us a good starting point from which fine-tuning can be done. The fine-tuning was done based on visual inspection of the output. A researcher would inspect the current working, note an error in the output, and would tune the filters (in accordance with the logic described above) in such a way as to try to mitigate this error. The finetuning took around 15 minutes.

3.2 Data Enhancement

Data enhancement is the first application-specific step in the prediction pipeline. In addition to the data filtering discussed above, we also give more weight to points near the hands. These points are most indicative of the arm angle, as they move the furthest for a change in the arm angle. To find the points near the hands, a ball is constructed around the point cloud, where the ball's boundary follows the user's maximal reach. After noise filtering, *arm zones* can be specified as slices of the enclosing ball. An arm prediction is made based on the number of signal points in each zone.

Figure 3.4 shows a visualisation of the steps in the prediction pipeline.

- 1) A raw point cloud is received from the mmWave radar,
- 2) We consider the enclosing ball (see section 3.2.2) and calculate the point weight based on the distance to the edge (see section 3.2.3),
- 3) We apply the combined noise filter, taking into account the assigned point weight, as well as the previous frames (see section 3.1.3),
- 4) We count up the weight of all signal (blue) points within each arm zone (see section 3.3.1),
- 5) We make a position state prediction based on the weight of each arm zone, as well as the previous predictions (see section 3.3.2).
- gt) Image with the ground truth skeleton (in orange) as a verification.

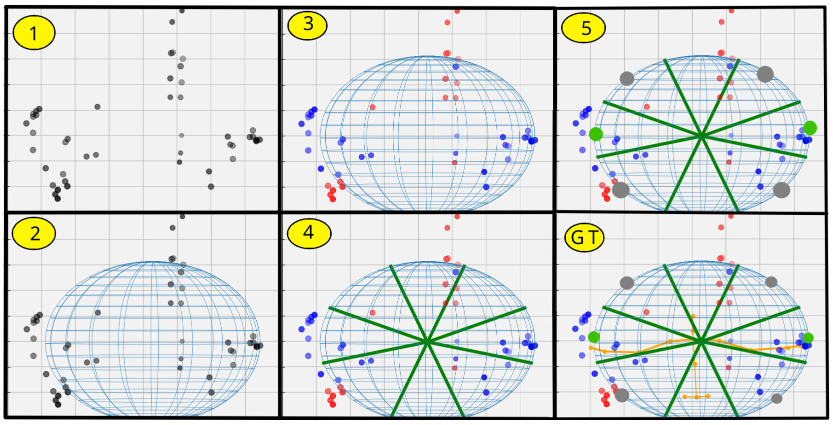


Figure 3.4: **Step by step zone prediction visualisation**

The data from the mmWave sensor is sparse, which makes it essential to use all information effectively, which can be done through *data enhancement*. Data enhancement broadly covers any method that increases the *quality* of the data sent to the classification systems. There are two distinct types of data enhancement methods. Physical enhancement, changing the physical setup in order to get better results, and software enhancements, where software is put in place to add extra information to our data. The first option, a change to the physical setup, can be effective; however, we should avoid this for the final product as it limits the flexibility of the system. Physical setup changes can help significantly during the design of a system, as high-quality data is needed to design a good classification algorithm, and a good classification algorithm is required to verify data enhancement methods. Software-based data enhancement methods add additional context to data which is already present. This is more challenging to create, but also more flexible, which makes it a preferable option for the final system.

3.2.1 Physical Enhancement

Many pose estimation systems rely solely on physical wearables [7, 12]. This is a logical choice, as it enables the direct tracking of specific parts of the user's body, at the cost of being more invasive and not easily deployable in non-private spaces. Our system aims to give more weight to points belonging to the hands.

We achieve this by utilising a *reflective glove*, which increases the point density near the hands (see fig. 3.5). This enhancement method was only used during the design of IAmMuse and not in the final system.

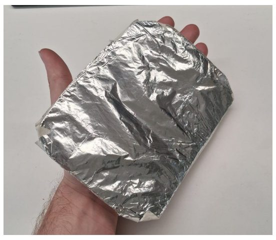


Figure 3.5: **A reflective wearable**

3.2.2 Enhancement Context: Enclosing ball

The primary goal of our data enhancement system is to make a distinction between points belonging to the hands, points belonging to the arms, and points belonging to neither. For this, we want to find the position of the user's sternum, as well as the arm span of the user. This will assist in the data enhancement, as any point emitted by the user's hand will have a distance to the user's sternum equal to half the user's arm span. To determine the sternum position, as well as the arm span, we first ask the user to move their arms up and down, tracing their kinematic range. We then accumulate these frames into a point cloud containing at least three thousand points. Finally, we fit an enclosing ball around the point cloud, allowing for some outliers outside of the ball, due to noise. The diameter of this ball should be equal to the user's arm span, and the centre of the ball should be equal to the user's sternum location.

For this, we used a process described by Ding, in *algorithm 1* of their paper [8]. This algorithm gives a stochastic estimation of a *minimal enclosing ball with outliers*, with bounds on the accuracy. The stochastic nature of this method results in an inaccurate fit in roughly 10% of the predictions. Though somewhat infrequent, this occurs often enough that a mitigation strategy is warranted. For this reason, the final enclosing ball estimation is constructed as the average of 100 different predictions to average out the noise and get a stable prediction. An in-depth explanation of the configuration of this algorithm can be found in Appendix E.

A current limitation of this system is that a user can only *calibrate* the system once, and this calibration dictates where the user has to stand, since the system expects the user's sternum to be stationary and at the centre of the enclosing ball. A more dynamic version of the calibration system was explored, where the enclosing ball would be recalculated each frame with the most recent 3000 points. In this approach, the enclosing ball would drift upwards if the user held their hands high for an extended period of time, invalidating the estimation. Similar systems, which allow the user to move slightly during execution, can be envisioned but were deemed outside of the scope of this research and are thus left for future research.

3.2.3 Programmatic Enhancement: Point Weight

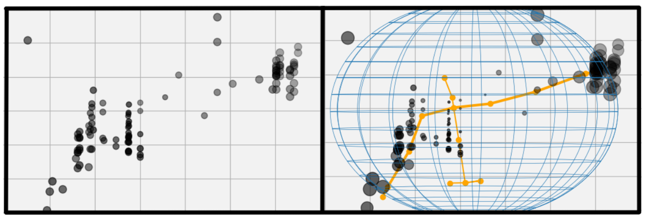


Figure 3.6: **Visualisation of the assigned point weights with the enclosing ball (blue) and ground truth skeleton (orange)**

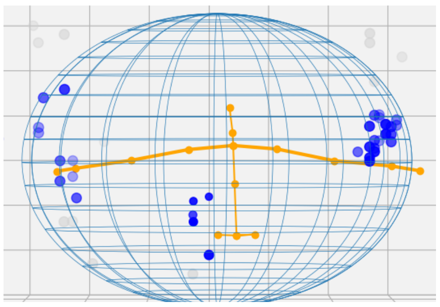
Once the context is acquired, the points can be given a weight based on the likelihood that they originated from the user's hands. The physical enhancement (section 3.2.1) generates a higher number of points of the hands to increase the prevalence of that region of the point cloud. The programmatic enhancement, on the other hand, adds the dimension of weight to represent the prevalence of these points.

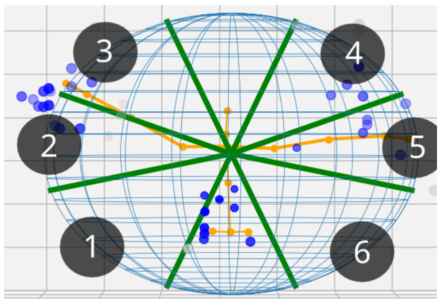
The enclosing ball gives a sufficient amount of context about the scene to be able to estimate which points are generated by the hands, arms, or neither. This context is not present in any one frame, which necessitates the online few-shot training step. Figure 3.6 shows an exaggerated visualisation of how the weights of the points change, using the enclosing ball. The points that are near the hands of the user (as visualised through the ground truth) receive an increased weight, while most other points receive a decreased weight. Notably, some noise points are also assigned an increased weight; however, these points are often removed in the noise filter.

This weight change is based on the distance to the surface of the enclosing ball, and the precise weight is determined using the *Gaussian Function*. The expected value of our Gaussian is equal to the radius of the enclosing ball, and the input to the function is the distance from the centre of the enclosing ball to the point in question. This ensures that the weight of a point is inversely proportional to the distance to the ball's surface. For the variance (σ^2), a value of 0.25 was used; this value ensured a high weight for any signal points belonging to either the hands or the arms, while reducing the weight of points further away by more than half. Other systems should take into account that the weights generated by this system are always smaller than one.

3.3 Data Interpretation

The data interpretation is built upon the specific enhancements of the input data provided by the data enhancement method. In our case, this entails a ball-shaped region that encloses the points of the user, the noise-signal labels, and the new point weight. The data interpretation method will use this information to determine which position state the user is currently in. Figure 3.7 shows a visualisation of the internal state of the system, with the ground truth in orange.


(a) **Internal data view**



(b) **Internal data view with zones**

Figure 3.7: **Internal data representation**

3.3.1 Angle Estimation

We want to predict a specific zone, *low*, *middle*, or *high*, for each hand. These zones are represented to be the volume within a *wedge* of the ball, as visualised in fig. 3.7b. Note that there are two regions (at the top and bottom) that are not a part of any zone. An alternative method defines these zones as angle ranges within the camera's projection plane. Here, the points are classified based on the angle of the vector extending from the projected sternum to the point in question. This produces an identical result, except for the fact that points which fall outside the ball can still be categorised into one of the position zones.

The point cloud is now divided into points contained within the six position zones and points not included in any position zone. We now assign a weight to each position zone, based on the weight of all the points contained in that specific zone. The weight of each position zone can be seen as the prediction likelihood of the user's arm actually being in that specific zone. Lastly, we find the right-hand position zone with the highest weight, and the left-hand position zone with the highest weight; these are now our zone predictions.

3.3.2 Output Stabilisation

This estimation system still has some issues, in particular when it comes to stability. The first issue we consider is that of the data sparsity and occasional intermittency observed in the mmWave radar. This lack of data results in some frames where a significant portion of the position zones has a low weight and, thus, a low certainty. These low certainties can lead to unsubstantiated, near-random predictions. To counteract this, we enforce a weight threshold that a zone must attain before we consider the prediction to be *valid*. If a frame does not contain any valid position zone predictions for the left or the right hand, then the previous prediction for that zone is used to avoid unsupported predictions.

A further challenge we have to solve is that of noise spikes, which throw off the predictions. Noise spikes are still at times permitted by the filter, for example, when two noise spikes happen in quick succession in the same region. These spikes can significantly influence the position zone weights, thus affecting the predicted zone. This manifests itself as incidental *random predictions*, which should be mitigated. We do this by introducing a temporal consistency constraint, since *valid predictions* are assumed to remain stable across frames. This constraint is enforced by only switching our output state if the same state has been predicted twice.

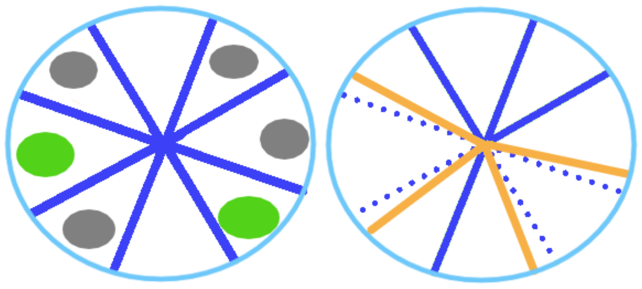


Figure 3.8: **Shows the zone size increase (orange) of the active zones (green).**

This is an incorrect behaviour as the user provides the same input (a stationary arm) and receives a different input on consecutive frames. This issue only occurs at the boundary between two position zones, and is caused by that boundary's specific position. Our mitigation strategy moves the boundary between the zones in such a way that points near or at that boundary are assigned to the currently selected zone, through a hysteresis mechanism, see fig. 3.8. With these mitigation strategies in place, the output of the prediction system is sufficiently stabilised.

Chapter 4

Experimental Setup

This chapter outlines the testing setup and introduces specific terminology for the collected data. A precise vocabulary and understanding of the data characteristics are essential to create effective test and training sets for the comparative SoA Deep Learning interpretation system. Therefore, this chapter discusses the testing setup (see section 4.1), the specific vocabulary for, and characteristics of, the collected data set (see section 4.2). Lastly, this chapter presents the training and test sets we used for the comparative model, as well as a reasoning for these choices (see section 5.1).

4.1 Testing Setup

The mmWave radar used in this thesis is the IWR6843ISK by Texas Instruments. The radar requires firmware to run, a diverse variety of which are provided by Texas Instruments, for varying sensing tasks. The main firmware characteristics that are important to this thesis are the number of points generated, the signal-to-noise ratio, and the discriminative value of each point, for arm angle estimation. We chose the *Mobile Tracker* firmware [16] as it meets those criteria. For a more detailed discussion of this firmware choice, see Appendix B. In addition to the mmWave Radar, a Kinect 2 was used to collect ground truth data. The skeletal estimation was generated through a code base published by Kharche on GitHub [21], which creates a smooth skeletal estimation and saves it, along with timestamps, to a CSV file. This skeletal estimation was transformed into an arm angle estimation by calculating the angle of the vector going from the *neck keypoint* to either *wrist keypoint*, with regards to the down vector, see fig. 4.1.

The experiments are performed in a large open room (7 $\times$ 10 m). The mmWave radar was mounted at a height of 125 cm and was tilted upwards at an angle of 5°. The Kinect was mounted at a height of 175 cm with an angle of 5° downwards. The user could see a real-time visualisation of the incoming point cloud, as well as the current arm angle estimation on a screen in front of them, as shown in fig. 4.2a. The user was standing at a distance of 2 meters from the sensors. Appendix C shows the experimental setup in detail. A large and open room was used, see fig. 4.2b, which meant that back wall reflections were spatially distinct from the user. This resulted in point clouds with relatively

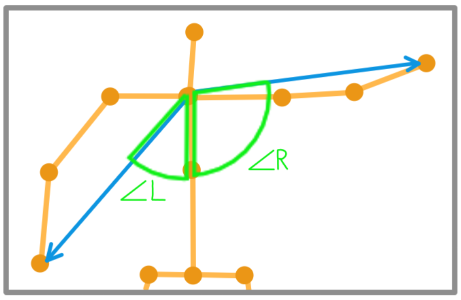
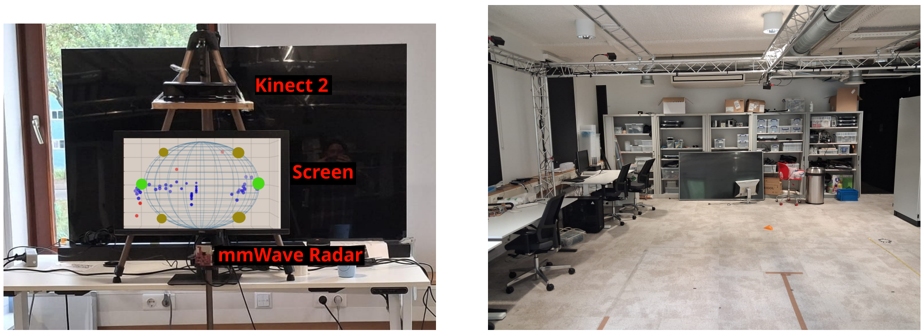


Figure 4.1: **Skeletal estimation into Arm Angle estimation visualisation.**



(a) **User view** (b) **View from the Sensor's perspective**

Figure 4.2: **The testing environment, providing the user view (a) and the sensor's perspective (b).**

little noise, producing a relatively high-quality dataset.

Six users took part in this experiment. For each user, we collected between 4 and 7 recordings; the average duration of a recording is 140 seconds. Each recording consists of two distinct parts: the *calibration* part, and the *interpretation* part. The *calibration* had an average duration of 20 seconds, during which the user moved their arms up and down. The remainder of the recording is part of the *interpretation*, and has an average duration of 120 seconds. During the *interpretation*, the user moves their arms to a specific position and holds the position for a few seconds before moving their arms to the next position. During the recordings, both the mmWave radar point cloud and a skeletal estimation produced by the Kinect 2 were recorded. The Kinect 2 recorded a video channel, but this video channel was only used to create the skeletal estimate and was never saved. While in use, the system produced musical notes in accordance with the pose that the system predicted at that given time. The users were given a consent form before the experiment, and were allowed to drop out of the experiment at any point. For privacy reasons, we chose only to record the height and arm span of the users, and no other information, see Appendix G. Before the experiment took place, we informed the users of how the system worked and what was expected of them through an explanatory PDF (see Appendix D). The specific actions the users performed are discussed in section 4.2.

4.2 Dataset

Frame specification

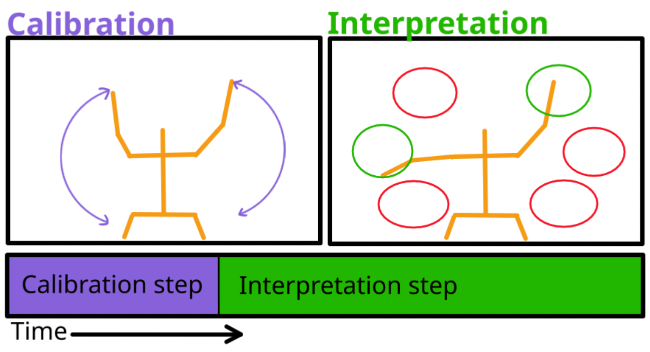


Figure 4.3: **Visualisation of the two frametypes present in each recording**

To properly compare IAmMuse to the state-of-the-art solutions, a clear definition of the available data is required. First, a distinction will be made between two types of frames within any singular recording of the system. At the start of each recording, users were instructed to calibrate the system, as is specified in section 3.2.2. Only after this calibration could the users actually use the system, since this calibration was required in order to do the data interpretation. We explicitly refrain from using the common machine learning terminology of *training* and *testing* data, for the frame types, as this would lead to confusion in section 4.3. Therefore, we refer to the data used for the *online few-shot training* as *calibration data* and the data recording during the system usage as *interpretation data*, as visualised in fig. 4.3.

Each recording contains **both** *calibration data* and *interpretation data* as the user recalibrates the system at the start of each recording. The distinction between the two types of data is critical due to the unique user behaviour in each step. In the calibration step, the user moves both their hands up and down at a steady rate. Conversely, in the interpretation step, the user intermittently moves their arms to specific positions and holds that static pose. Each recording has an average of 212 *calibration frames* and 1184 *interpretation frames*, resulting in an average duration of 140 seconds. The exact numbers for each recording can be found in Appendix F.

Recording types

The recordings are also split into two distinct groups, where the user follows different instructions. The *calibration data* of these recordings will be very similar, as the two types of recording require the same type of calibration. The *interpretation data*, however, will vary based on the recording type. Firstly, there is the **standard recording**, in which the researcher instructed the user on how they should move, in accordance with a script, which can be found in Appendix A. In this standardised script, the users hold various arm positions for 2-3 seconds. The script was designed to be simple to follow for the user by following a logical pattern while also covering every transition between distinct arm poses. These standard recordings were designed to provide a clear comparison between users and also to provide a stable training set for any Deep Learning models. For most users, there are *three recordings* of the standardised user set. However, due to data corruption, two users only have *two standard recordings*.

The second type of recording is called a **free play** recording. These recordings simulate a realistic usage scenario, offering a more accurate representation of the input data IAmMuse is expected to process. These recordings were made after the standard recordings, to ensure some familiarity with the system. The users were instructed to freely explore the musical interface, leading to more realistic behaviour patterns while they tried to make their own music. Different users made a different number of free-play recordings, due to the more open-ended nature of this recording type.

4.3 Baseline Analysis

For our comparison, we need a DL model that is a State-of-the-Art work in the field of mmWave radar HPE. Furthermore, the DL model has to evaluate the same data that IAmMuse evaluates, the point clouds of the mmWave radar. The output of the system should be a skeletal estimation, which can be transformed into an arm angle estimation in the same way as the ground truth. With these considerations, the MARS model [2], a State-of-the-Art model in the field of mmWave HPE and one of the first systems to generate skeletal estimations out of sparse mmWave point clouds, is selected as the ideal comparison model.

The MARS paper and this thesis both use a Kinect 2 for the ground truth and use mmWave radar point clouds as input. One notable difference in the setups is the fact that MARS uses an older radar model, the IWR1443, as opposed to the IWR6843, though their output should be comparable. Furthermore, the design of the MARS model has no temporal context and estimates each point cloud frame independently; therefore, it may struggle with sparse point cloud frames.

As a deep learning model, MARS's performance is directly linked to its training data. Therefore, we consider multiple versions of MARS in the evaluation, each trained on a different dataset. For the training of each of the models, we used a batch size of 128, with 150 epochs, in accordance with the example code provided on the GitHub repository of MARS [3]. The different MARS models we considered each had different training and testing data, as shown in table 4.1. For the evaluation, we considered five different models:

- **MARS baseline**: This model demonstrates the performance of the MARS model if it uses the same training and testing data as IAmMuse. Therefore, it trains on the *calibration data* of a **specific recording**, and it tests the *interpretation data* of **the same recording**. The performance of this model is discussed in section 5.1.

- **MARS pre-trained**: This model shows the ability of MARS to generalise over datasets. For this, it uses the weights provided by the original MARS paper [2], so it is trained on the MARS dataset, while it tests the *interpretation data* of all recordings. The performance of this model is discussed in section 5.2.

- **MARS full calib**: This model's training set consists of the *calibration data* of **all recordings**, increasing the size of the training set to roughly one fifth that of the MARS paper. This model is also tested on the *interpretation data* of all recordings. The performance of this model is discussed in section 5.2.
- **MARS partial standard**: This model shows the ability of MARS to predict standard recordings when trained on very similar data. To do this, the model is trained on two-thirds of the *interpretation data* of the standard recordings, and tested on the remaining one-third of the *interpretation data* of the standard recordings. In this manner, the testing and training data are as similar as possible, which should result in optimal performance. The performance of this model is discussed in section 5.3.
- **MARS standard to free**: This model tests the capacity of MARS to generalise from the standardised move set to a more realistic scenario. This model is trained with the interpretation data of *all standardised recordings*, and is tested on the interpretation data of *all free-play recordings*. The performance of this model is discussed in section 5.3.

By considering five distinct versions of MARS, we aim to provide a fair comparison, ensuring we do not disadvantage MARS with a single suboptimal training set. Furthermore, the consideration of multiple models allows various failure modes of the MARS model to be shown. Some of these failure modes are unique to the MARS models, but others are more generally found in Deep Learning models as a whole.

Model Name	**Training Data**	**Testing Data**
MARS baseline	Calibration data of *one specific recording*.	Interpretation data of *the same specific recording*.
MARS pre-trained	MARS data set.	Interpretation data of all recordings.
MARS full calib	Calibration data of all recordings.	Interpretation data of all recordings.
MARS partial standard	Interpretation data of 2/3 standard recordings.	Interpretation data of the other 1/3 standard recordings.
MARS standard to free	Interpretation data of all standard recordings.	Interpretation data of all free-play recordings.

Table 4.1: **The training and testing data for the various MARS models considered**

Chapter 5

Evaluation

To assess the effectiveness of the methods proposed in this thesis, a comparison should be made with a similar SOA model. For this thesis, the *MARS* system [2] was chosen, this choice is substantiated in section 4.3. A few terms should be defined before discussing the specific evaluation metrics used. The different positions a single arm can hold: "low", "middle", or "high" are referred to as "*arm positions*". The combined position of both arms, however, is called a "*pose*" or "*pose state*". With each arm being able to take one of three *arm positions*, that thus gives us **nine** total *pose states*.

The following evaluations will compare IAmMuse and the MARS models on several distinct metrics. Two metrics we consider are called the *likelihood* and the *accuracy* of a model to predict a certain *pose state*. This thesis defines the *likelihood* as the chance of a model predicting the given *pose state* **regardless** of the *pose state* indicated by the ground truth. While *accuracy* is defined as the chance of a model predicting a given *pose state*, while the ground truth indicates the same pose state. The likelihood graphs thus provide insight into the *bias* of a particular model, while the accuracy graph tells us what positions the model struggles or excels at. Since not all positions occur equally in the actual recordings, the likelihood graph also includes the ground truth class distribution.

Other visualisations may also be used, depending on the specific performance characteristics illustrated. Skeletal estimations depict the predicted output of the MARS model. *Time-Prediction plots* highlight the temporal characteristics of specific prediction models. Lastly, to illustrate the *data efficiency* of each model, we plot the final classification accuracy against the respective training set size.

The MARS models considered during this chapter have been discussed in section 4.3. There, we specify the training and testing data for each of the models. Furthermore, we provide the reasoning for why we consider those specific models.

We use specific terminology to refer to the data captured during recording. Data captured during the *online few-shot training* section is called *calibration data*, and the data recorded afterwards is called the *interpretation data*. This was done to avoid confusion when discussing the training sets for the MARS model, as discussed at the top of section 4.3

5.1 Baseline Analysis

The first comparison that will be considered is with *MARS baseline*, which is trained on the same data that IAmMuse is, for each recording. The IAmMuse system and this MARS model both train/initialise on the *calibration data* and evaluate the *usage data*, which means that these comparisons will be a fair "apples to apples" comparison.

In the comparison between IAmMuse and *MARS baseline*, a significant discrepancy was observed. The prediction accuracy of *MARS baseline* marginally exceeded random performance, as can be seen in fig. 5.1. This poor performance can be attributed to the limited size of the training set (ranging from 108 to 275 frames, depending on the specific recording), as opposed to the 24,066 frames used by the MARS paper [2]. IAmMuse avoids these limitations by using domain-specific knowledge in its design to avoid a dependence on large datasets.

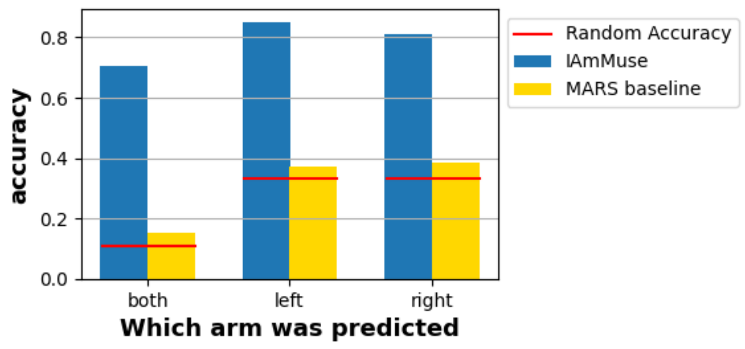


Figure 5.1: **Accuracy of predicting one or both arms correctly for the IAmMuse system and the MARS baseline (a2a) system.**

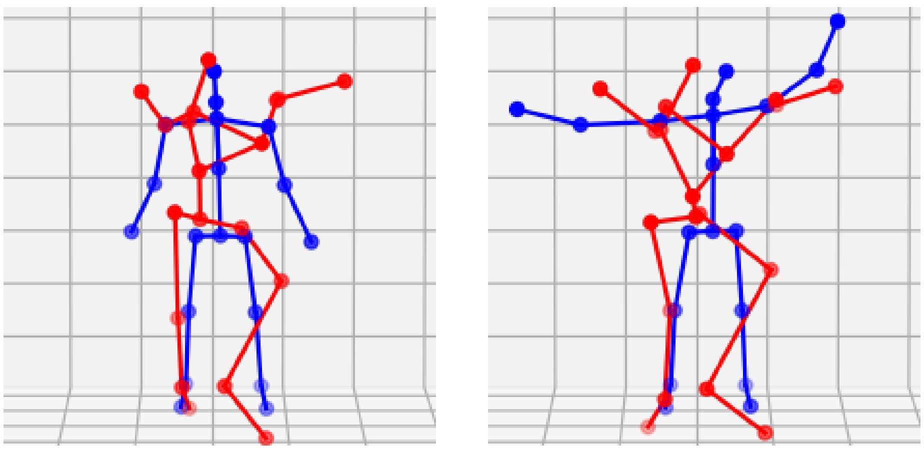


Figure 5.2: **Two skeletal predictions of MARS baseline (in red) with the ground truth as captured by the Kinect overlaid (in blue)**

The lack of training data can often produce low performance, or even unintelligible results with deep learning models. This is in large part because there are no inherent concepts of logical bounds built into many DL models, similarly so with MARS. This is even further illustrated by the fact that *MARS baseline* is not able to generate correct-looking human skeletons, see fig. 5.2. This does, however, result in MARS performing very poorly when provided with the same tools as IAmMuse. On the same training and testing data, the accuracy of single *arm position* predictions is 2 times higher for IAmMuse, and the accuracy of the full *pose state estimation* is more than 4 times higher for IAmMuse. One inherent advantage of Deep Learning models, however, is that they can be trained with a large data set *once* and used *many times*. This could alleviate the accuracy issues observed, at the cost of an increased training set size.

5.2 Larger Datasets

Given the poor performance of *MARS baseline*, at least in part due to the small training data set, we will consider only models with a significantly larger training set, from this point forward. For this purpose, two different models will be considered, *MARS pre-trained* and *MARS full calib*. *MARS pre-trained* is the model provided by the MARS paper [2], and is thus trained on the MARS dataset. *MARS full calib* is the model which is trained on the *calibration data* of **all** recordings, of all users. These models should test the generalisation capabilities of the MARS framework, as both are trained with a larger dataset consisting of users performing different kinematic actions while remaining stationary. *MARS full calib* uses 7,229 frames for training, while *MARS pre-trained* uses 24,066 frames, as shown in fig. 5.11.

Surprisingly, these models do not perform significantly better than the baseline model. Figure 5.3 shows how both the *MARS paper* and the *MARS full calib* model perform only slightly better than a random prediction model. This low accuracy is present for the *pose state* prediction ("both") as well as for the individual *arm positions* predictions ("left" and "right"). The poor performance of these two models, though seemingly similar to *MARS baseline*, has a different cause.

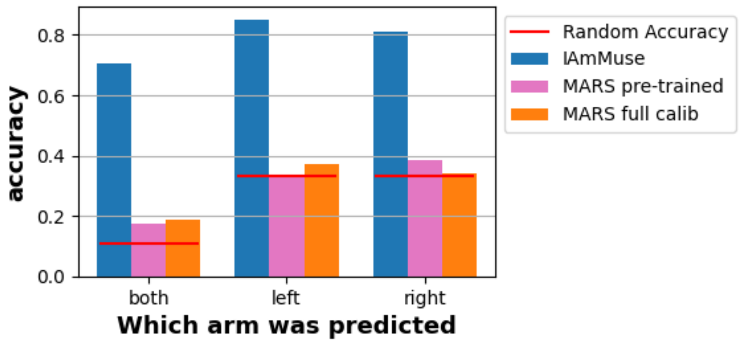


Figure 5.3: **Accuracy of predicting one or both arms correctly for the IAmMuse system, the MARS pre-trained, and the *MARS full calib* systems.**

The baseline model was not able to generate successful human skeletons, leaving the output of its arm zone prediction relatively random. The problem with *MARS pre-trained* and *MARS full calib*, however, is very different, as can be clearly seen in fig. 5.4. This figure shows the prediction likelihood of each model, as well as the ground truth state distribution (in red). *MARS baseline* (yellow) does not predict the various pose states with the same likelihood; however, all pose states are predicted sometimes (even if it's only very rarely). This behaviour is to be expected, as the output of this model is, at least in some part, random. The *MARS pre-trained* (pink) and *MARS full calib* (orange), conversely, seem only to predict one or two position states ever. IAmMuse (blue), on the other hand, follows the ground truth distribution very closely.

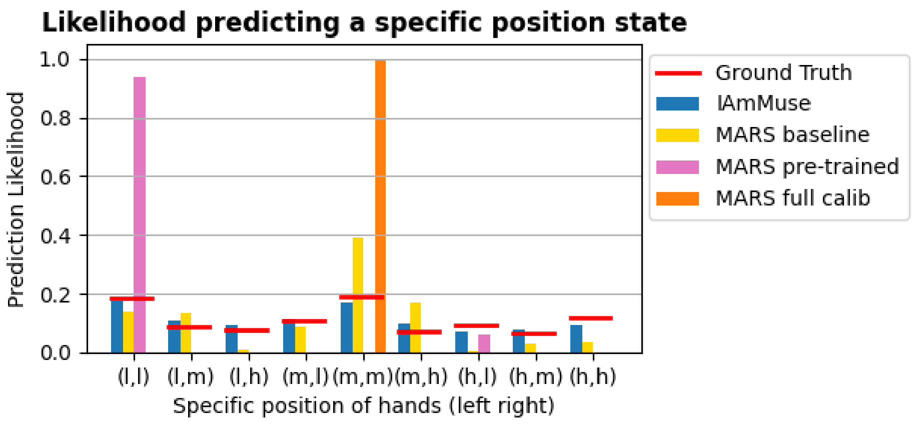


Figure 5.4: **Likelihood of position states for various MARS models and the ground truth**

This behaviour of *MARS pre-trained* and *MARS full calib* points to them suffering from something else. *MARS pre-trained* suffers from unrepresentative training data and potentially from overfitting. This overfitting can happen because mmWave radars produce slightly different point clouds in different rooms, as mentioned in section 2.1. These room changes can alter small secondary patterns in the point clouds, mainly with respect to the noise and reflections in the room. If these secondary patterns are being used in the recognition model of a DL system, then it will cease to function when moved to a new location. Furthermore, there are slight differences in the hardware setup between the MARS paper and IAmMuse. The MARS paper uses a different mmWave radar (the IWR1443BOOST, as opposed to the newer IWR6843ISK, which was used for IAmMuse) and firmware, which would make their data unrepresentative for our setup. In fig. 5.5a, it is shown how the skeletal estimate does not line up well; furthermore, the predicted skeleton almost always has its arms tilted slightly downwards, regardless of the input point cloud.

MARS full calib seems to suffer from a somewhat different issue, though, as its data collection used the same hardware and the same physical environment. The model always holds its arms out to the side, regardless of the input data, as can be seen in fig. 5.5b, the exact point positions in the prediction also rarely ever move. This indicates that this model has gotten stuck in a local minima during training, since the position where the skeleton holds both its arms out straight minimises the average mean squared error for *calibration data*. Since the calibration data consists of a user waving their arms up and down, as is described in section 4.2, the static position with minimal error is the one where a user holds their arms out to the side. This position causes the minimal squared error, as it minimises the distance between the predicted arm position and the set of all possible arm positions This shows that the model has stopped trying to dynamically interpret the input data, instead taking on a static position of minimal loss.

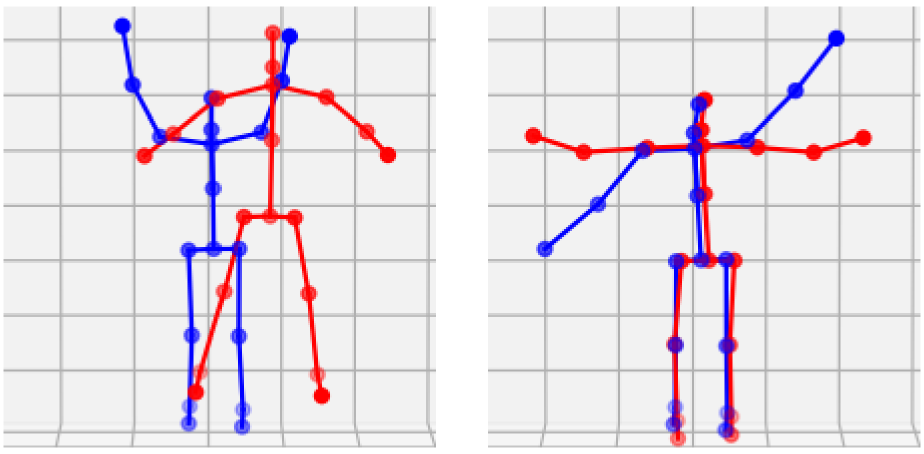

(a) **pre-trained: non-aligned** (b) **calib-trained: static "T-posing"**

Figure 5.5: **skeletal estimates of pre-trained (a) and calib-trained (b) MARS (red) with ground truth (blue)**

These results show some of the shortcomings that deep learning models may encounter. A lot of high-quality and diverse training data is needed to avoid degenerate solutions, such as minimising the error with a static mean pose, as *MARS full calib* did. Furthermore, one of the main advantages of DL models, being able to learn minute patterns in data, can become a downside when the specific environment changes, as the model may lose the scene-specific noise patterns it depends on. As such, even when MARS has a training set 28 times larger than what IAmMuse uses, we still find that IAmMuse's accuracy on the *pose state* estimation is 3 times higher than MARS's. The final comparison in our evaluation attempts to elicit a viable performance from the MARS system by training it on a significant portion of the *interpretation data*, in an effort to alleviate the encountered issues.

5.3 IAmMuse vs Best MARS

The final two MARS models considered are *MARS partial standard* and *MARS standard to free*. As discussed in section 4.3, these models are trained using a subset of the *interpretation data*, the rest of which is used for testing. These models will assess the accuracy of the MARS model when it is trained with highly representative data, representing a *best-case scenario* for the MARS architecture. Furthermore, the training set size for these models, 18,531 and 25,139 frames, respectively, is comparable to the 24,066-frame training set utilised by the MARS paper.

The accuracy of *MARS partial standard* and *MARS standard to free* is significantly higher than the previously considered MARS models, as shown in fig. 5.6. Prior models had an accuracy below 0.4 for individual *arm position* predictions, and an accuracy below 0.2 for *pose state* predictions. *MARS partial standard* and *standard to free* however, both have an accuracy above 0.65 for the *arm position* predictions, and above 0.45 for the *pose state* prediction. Even with this increase in accuracy, however, IAmMuse still outperforms these models by a significant margin.

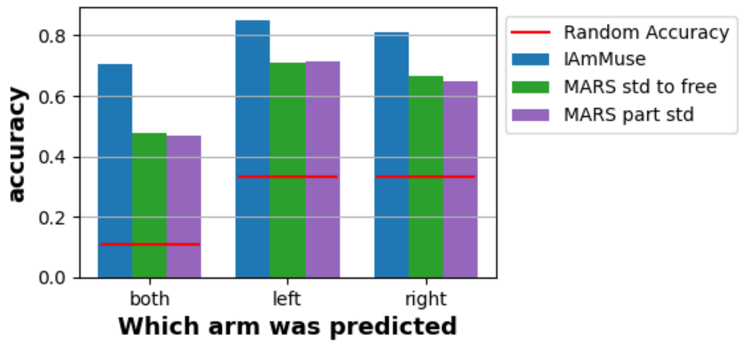


Figure 5.6: **Accuracy of predicting one or both arms correctly for the IAmMuse system, the MARS standard to free, and the MARS partial standard systems.**

The *accuracy graph* (fig. 5.7) shows a high accuracy for the MARS models, on frames where both arms hold the same position. However, the MARS models have very poor performance on frames where the two individual arm positions are dissimilar, such as when one arm is positioned *high*, and the other is positioned *low*. The *likelihood graph* (fig. 5.8) provides more insight into the cause of these results. *MARS partial standard* and *standard to free* predict the *pose state* where both arms are in the *middle* position disproportionately often; consequently, prediction position states where the arms are in very distinct positions are predicted significantly less frequently. This is a similar phenomenon as seen with *MARS full calib*, but to a lesser extent.

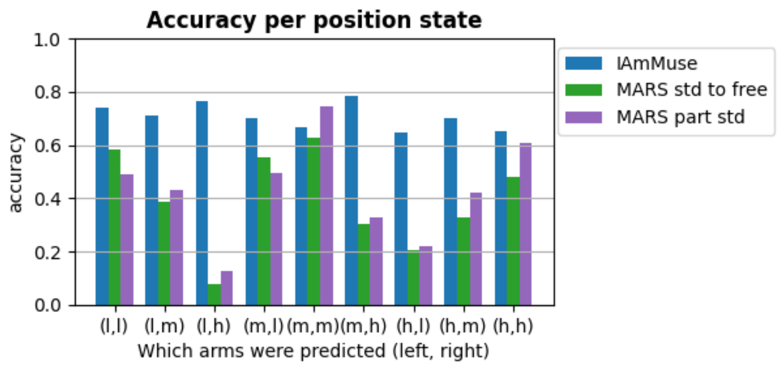


Figure 5.7: **Accuracy per state position for the IAmMuse system, the MARS standard to free, and the MARS partial standard systems.**

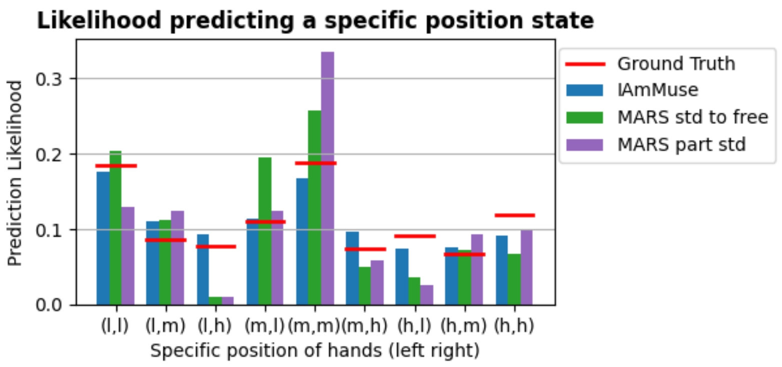
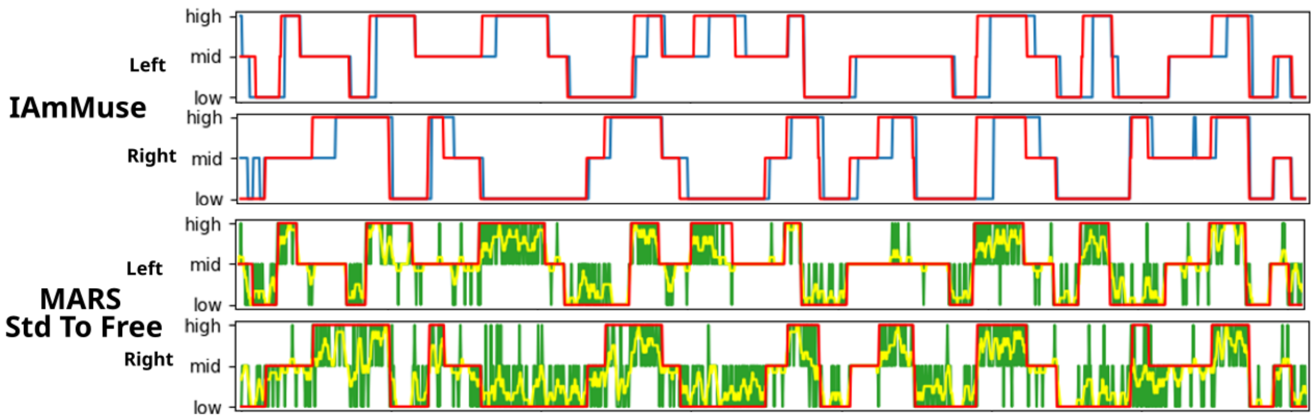
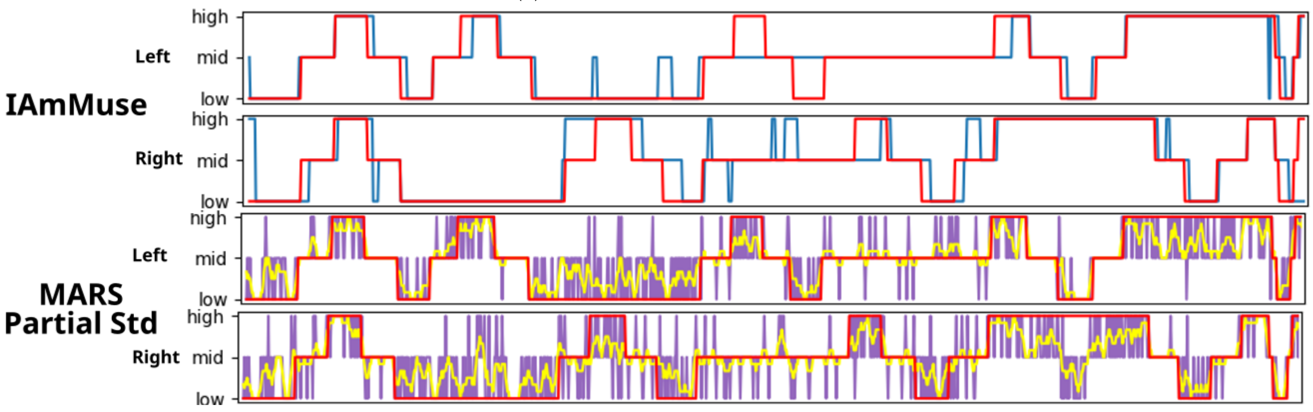


Figure 5.8: **Likelihood of position states for IAmMuse (blue), MARS standard to free (green), MARS partial standard (purple), and the ground truth (red)**



(a) **MARS standard to free**



(b) **MARS partial standard**

Figure 5.9: **time-prediction plots for IAmMuse (blue), MARS standard to free (green), MARS partial standard (purple), with the ground truth (red) and a temporal smoothing version of MARS (yellow)**

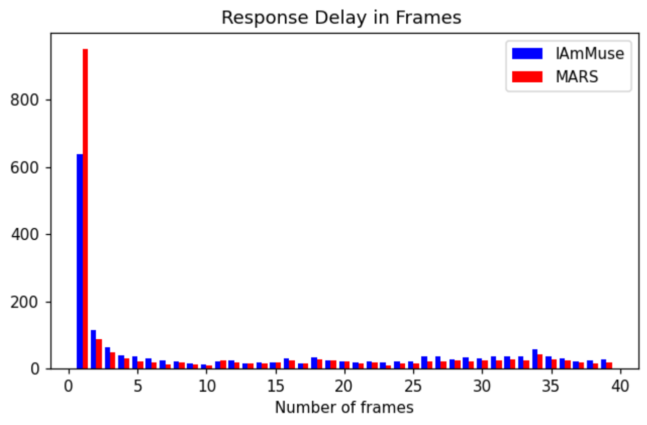
The *time prediction* plots also clearly depict this tendency to "jump back to the middle position" which the MARS models exhibit. Figure 5.9 shows the predictions of each model over time, as well as the ground truth. Here, the predictions of the particular MARS models keep rapidly jumping between "middle" and the actual position. We tried to solve this issue by applying an averaging smoothing filter with a window size of 7 (The yellow line), but even with this extra assistance, the error of the MARS systems was significantly higher than the error of the IAmMuse system. This issue is most likely in part because the MARS system does not take into account the temporal connections in the data; it simply predicts each frame on its own.

This shows that, even if MARS is trained on a dataset with fully representative data and a similar size to that of the MARS paper, IAmMuse still outperforms it with an accuracy that is 1.5 times higher. Although these MARS models do produce a reasonable estimation most of the time, they still suffer from a statistical bias. Furthermore, their output can not be stabilised by a smoothing algorithm due to the significant frequency at which incorrect predictions are made. Finally, these models would also be highly impractical to use in real-world scenarios if they have to be retrained with approximately an hour of training data every time they are installed in a new space. Therefore, we conclude that we were not able to produce a MARS model with comparable accuracy to that of IAmMuse in the problem space of *arm angle estimation*, even when using 25,000 frames of representative data as a training set.

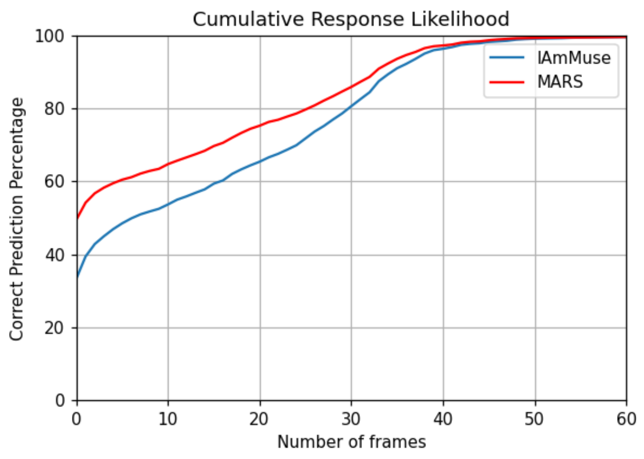
5.4 IAmMuse Response Times

This section analyses the *response time* of the different systems. The response time is the delay, in frames, between the ground truth changing and the prediction model correctly predicting the new state. The response times of the *MARS baseline*, *pre-trained* or *full calib* models have been excluded, as their low classification accuracy makes this information statistically uninformative. The response times displayed are thus for the *MARS partial standard* recording, as it had the highest classification accuracy, and because the results for *MARS partial standard* and *standard to free* were very similar. However, the high variance in the output of the *MARS partial standard* model artificially deflates its average response time. Since the metric considers the *first correct frame* as a successful response, the rapid change of predictions means that the model can at times achieve low latency scores, because it rapidly switches state, and not because it detected the changing input data correctly.

The response times of both IAmMuse and MARS are shown in fig. 5.10a. Each bar represents the number of state changes which were recognised in that precise number of frames. Figure 5.10b shows the cumulative response likelihood, the number of frames it took until a response was measured. These figures show that IAmMuse produces a correct response within a single frame in 36% of state changes, while MARS does so in 51%. After seven frames, IAmMuse will have produced a correct response in 50% of cases, while it takes until nearly 30 frames to produce a correct response in 80% of cases. MARS's response time is lower than that of IAmMuse, though this may in part be due to the high variance in its output. The rapidly changing predictions of MARS will allow it to receive *low response times* coincidentally, as opposed to achieving low response times through genuine state detection. This happens because the response time metric only considers the time until the **first** successful prediction.



(a) **Response delay of IAmMuse (blue) and MARS (red)**



(b) **Cumulative response likelihood of IAmMuse (blue) and MARS (red)**

Figure 5.10: **Response characteristics of IAmMuse and MARS**

5.5 Key Takeaways

In this chapter, we compared IAmMuse to various trained MARS models. IAmMuse outperformed each of these MARS models, with an accuracy that was between 50% and 300% higher than the MARS model, depending on the specific model considered. The highest performing MARS models also required a significant amount of training data (25,000 frames, 100 times the size of IAmMuse's training set), which had to match the testing data very closely. With insufficient training data, or data which did not match the test set exactly (*MARS pre-trained* and *MARS full calib*), the performance was only slightly better than random guessing, see fig. 5.11. This shows that a well-designed algorithm can outperform a DL system with a fraction of the training data.

The evaluated models exhibited distinct characteristics, which affected their performance. *MARS baseline*, trained on the calibration data of a single recording, performed poorly due to insufficient training data, resulting in unrealistic skeletal estimations (see fig. 5.2). *MARS pre-trained*, trained on the data provided by the MARS paper, also performed poorly, even though it had ample training data (see table 5.1. The model was not able to generalise from the training set in the MARS paper to the test set used in this thesis. *MARS full calib*, being trained on 7,229 frames, also had a very poor performance, as it produces a static, degenerative estimation. *MARS partial standard* and *MARS standard to free* performed the best of all the MARS models considered. However, these models produced a volatile result, often predicting the user's arm to be in the *middle position*, even when this was not the case.

While IAmMuse does outperform the MARS models on accuracy, on response times, it does exhibit higher latency than MARS. The response time is

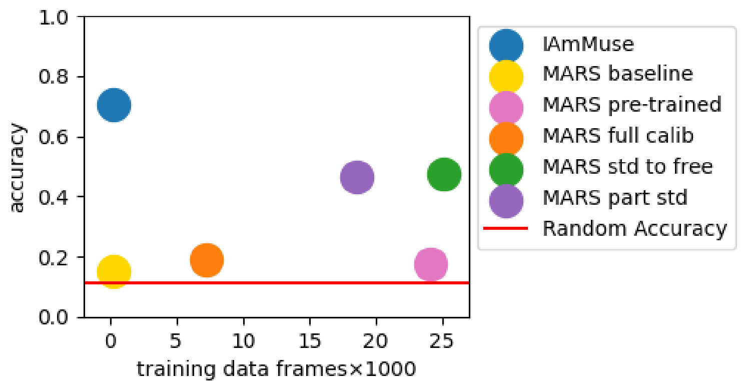


Figure 5.11: **The accuracy for each model considered (y-axis) plotted against the amount of data it used for training (x-axis)**

System	Training Data (in frames)	Accuracy of Pose State Prediction
IAmMuse	108-275	70%
MARS baseline	108-275	15%
MARS full calib	7,229	18%
MARS pre-trained	24,066	19%
MARS partial standard	18,531	48%
MARS standard to free	25,139	47%

Table 5.1: **The training set size and prediction accuracy for each of the considered systems**

measured as the number of frames it takes for a change in the ground truth to be correctly identified by the prediction model. We found that the *MARS partial standard* had a correct response in the first possible frame in 51% of cases, while this number is 37% for IAmMuse. Furthermore, IAmMuse requires approximately seven frames to achieve a response in 50% of state changes. However, the lower response times may in part be caused by the high variance of the MARS model. Because the response time metric measures the delay until the first successful prediction, the MARS model's tendency to rapidly switch states allows it to achieve low latency scores coincidentally, rather than through accurate tracking of the input data.

Nevertheless, the IAmMuse system achieves a significantly higher accuracy, of 70%, compared to the best MARS accuracy being 48%, all while using only around 1% of the amount of training data which the MARS system uses. IAmMuse addresses a more constrained problem, in *arm angle estimation*, as opposed to the *skeletal estimation* which MARS provides, but for many applications, this more focused data is sufficient. IAmMuse has succeeded in showing the potential of non-deep learning approaches to mmWave radar interpretation for use in computer interfaces. In addition to the increased accuracy, this approach also allows for the tuning of specific components, as opposed to retraining the system. Furthermore, IAmMuse does not function in a 'black box', as opposed to deep learning models. We were able to alleviate the issues regarding the low density of the mmWave radar point clouds by leveraging domain-specific knowledge, as well as user characteristics obtained through a *few-shot online learning* step, to enhance the raw point cloud.

Chapter 6

Conclusion

With the steady adoption of integrated computer systems, a need arises for seamless privacy-preserving and low-latency interfaces. A promising sensor for this task is the mmWave radar; however, its output is sparse and prone to noise. Existing interpretation methods primarily rely on Deep Learning (DL) methods, which are data-hungry, resource-intensive, and opaque, potentially limiting their wide-scale adoption. Therefore, we designed IAmMuse, an interpretation system exploiting known domain knowledge, through traditional signal processing methods. This interpretation method produces a *discrete arm-angle-estimation*, which predicts the position of both the user's arms as being either *low*, *middle*, or *high*. When compared to the DL baseline model, IAmMuse achieved a classification accuracy 50% higher than that of the baseline, while using approximately 1% of the amount of training data.

Several research challenges were addressed before and during implementation. The fragmented point cloud data was the first research challenge, as the mmWave radar only captured parts of the user in some frames. We circumvented this issue by creating a system that stabilises these predictions, even on sparse or partially incomplete frames. To achieve this, we applied a two-part smoothing algorithm over the prediction output. This algorithm first employs a confidence threshold to reject any low-confidence predictions. Secondly, it only changes the system output if a prediction occurs twice in a row.

Another challenge was the low-latency requirement. Low-latency response times could not fully be realised. Approximately 40% of the state changes were correctly identified on the following data frame. This detection probability increased to 50% by the 7th frame, and rose to 80% only after a latency of 30 frames.

The final research challenge for this thesis was to outperform, or perform comparably to, a State-of-the-Art DL model, MARS [2]. A comparison between IAmMuse and MARS showed that IAmMuse achieves superior accuracy in predicting the position of both the user's arms. IAmMuse had an accuracy of 70%, meaning that 70% of the frames were classified correctly, while MARS's accuracy ranged from 15% up to 48% depending on the training data used. To achieve an accuracy above 20%, MARS had to be trained with a subset of the interpretation data (with the remainder being used for testing), as opposed to the distinct *calibration dataset*, which IAmMuse used for training. Furthermore, the MARS models, which achieved an accuracy above 20%, also used roughly

100 times more training data than IAmMuse did (18,000-25,000 frames, as opposed to 108-275 frames). In the case where MARS used 100 times the amount of training data IAmMuse used, we still observed that IAmMuse gives an accuracy 50% higher than MARS.

Given these findings, we believe that more traditional signal processing-based interpretation systems for mmWave radar are a promising option for integrated HCI systems. Nevertheless, IAmMuse is currently not flexible enough to be deployed in real scenarios, as its systems are still too constrained. Constraining the scope to arm angle estimations is not an inherent issue; however, the low angular resolution remains a limiting factor. In addition, the dependence on a *calibration step* and the need for a stationary user limit the usability of the interface in its current state.

6.1 Future work

Unified Dynamic User Tracking

To simultaneously address the requirement of a stationary subject and the dependency on a calibration step, future research should investigate a dynamic user tracking model. To maintain compatibility with the current architecture, this tracking method would need to provide the same user model as IAmMuse, consisting of the user's *sternum location* and *arm span*. By tracking a user once they enter the Field of View of the radar, this system could maintain a continuous estimation of their sternum location. For the *arm span*, we can assume a default value at the start (e.g. 170 cm) and refine this online through analysis of the point cloud. Ensuring that these estimated variables match reality well allows for seamless integration into the existing IAmMuse pipeline.

Increased Angular Resolution

Future work should address the granularity limitation of the current system. We identified two potential approaches to solve this issue. Firstly, the *classification system* of IAmMuse could be replaced by a *regression model*, providing a real-valued angle estimation, as opposed to a discretised angle zone. Secondly, the granularity of the classification could be improved by subdividing the full angle range into a large number of smaller angle intervals. This latter approach can reuse the conceptual basis of the current classification system, but distribute the weight of a point over adjacent angular zones instead of using a binary hard-assignment. To support this new classification system, the stabilisation system could be replaced by a combined temporal median and IIR filter to remove outliers and ensure a smooth output. Consequently, this would also alleviate some of the latency issues observed, as this new stabilisation system can change its prediction with a single frame, as opposed to requiring at least two frames as IAmMuse did.

Latency Reduction

A current shortcoming in IAmMuse is its response time. The system does detect 40% of transitions immediately, but a detection rate of 50% is only achieved after 700 milliseconds, while an 80% detection rate only occurs 3 seconds after the state change. However, the fact that a significant percentage of frames are estimated with low latency suggests that the high latency issue is not inherent to the proposed framework, but rather an implementation-specific issue. Therefore, the first step is to identify and analyse the scenarios in which significant latency occurs. These findings can then be used to guide modifications to the existing method to alleviate this latency issue.

Bibliography

[1] Mohammad Arif Ul Alam, Mahmudur Rahman, and Jared Widberg. Palmar: Towards adaptive multi-inhabitant activity recognition in point-cloud technology, June 2021. URL https://doi.org/10.48550/arXiv.2106.11902.

[2] Sizhe An and Umit Y Ogras. Mars: mmwave-based assistive rehabilitation system for smart healthcare. *ACM Trans. Emb. Comput. Syst.*, 15(5s), September 2021. URL https://dl.acm.org/doi/abs/10.1145/3477003.

[3] Sizhe An and Umit Y Ogras. Mars. https://github.com/SizheAn/MARS/commits/main/, sep 2021. Last accessed: Nov. 7, 2025.

[4] Francisco Bernardo, Nicholas Arner, and Paul Batchelor. O soli mio: Exploviring Millimeter Wave Radar for Musical Interaction. May 2017. URL https://www.researchgate.net/publication/317285973_O_Soli_Mio_Exploring_Millimeter_Wave_Radar_for_Musical_Interaction.

[5] Walter Brescia, Pedro Gomes, Laura Toni, Saverio Mascolo, and Luca De Cicco. Millinoise: a millimeter-wave radar sparse point cloud dataset in indoor scenarios. In *Proceedings of the 15th ACM Multimedia Systems Conference*, MMSys '24, page 422–428, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704123. doi: 10.1145/3625468.3652189. URL https://doi.org/10.1145/3625468.3652189.

[6] Emmanuel Chris, Anita Johnson, and Grace Phonix. Deep learning vs. traditional machine learning: Key differences, November 2024. URL https://www.researchgate.net/publication/389991583_Deep_Learning_vs_Traditional_Machine_Learning_Key_Differences.

[7] Antonio I Cuesta-Vargas, Alejandro Galán-Mercant, and Jonathan M Williams. The use of inertial sensors system for human motion analysis. *Physical Therapy Reviews*, 15(6):462–473, 2010. doi: 10.1179/1743288X11Y.0000000006. URL https://doi.org/10.1179/1743288X11Y.0000000006. PMID: 23565045.

[8] Hu Ding. A sub-linear time framework for geometric optimization with outliers in high dimensions. Technical Report arXiv:2004.10090, School of Computer Science and Engineering, University of Science and Technology of China, He Fei, China, 2020. URL https://arxiv.org/abs/2004.10090. Preprint.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996. URL https://dl.acm.org/doi/10.5555/3001460.3001507.

[10] Junqiao Fan, Jianfei Yang, Yuecong Xu, and Lihua Xie. Diffusion model is a good pose estimator from 3d rf-vision. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 1–18, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-72640-8. URL https://doi.org/10.48550/arXiv.2403.16198.

[11] E. M. Farella, A. Torresani, and F. Remondino. Sparse point cloud filtering based on covariance features. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W15:465–472, 2019. doi: 10.5194/isprs-archives-XLII-2-W15-465-2019. URL https://isprs-archives.copernicus.org/articles/XLII-2-W15/465/2019/.

[12] Alessandro Filippeschi, Norbert Schmitz, Markus Miezal, Gabriele Bleser, Emanuele Ruffaldi, and Didier Stricker. Survey of motion tracking methods based on inertial sensors: A focus on upper limb human motion. *Sensors*, 17(6), 2017. ISSN 1424-8220. doi: 10.3390/s17061257. URL https://www.mdpi.com/1424-8220/17/6/1257.

[13] MiMu Gloves. MiMu gloves the world's most advanced wearable musical instrument for expressive creation, composition, and performance. https://mimugloves.com/. Accessed: 2025-11-21.

[14] Diego Guffanti, Alberto Brunete, Miguel Hernando, Javier Rueda, and Enrique Navarro Cabello. The accuracy of the microsoft kinect v2 sensor for human gait analysis. a different approach for comparison with the ground truth. *Sensors*, 20(16), 2020. ISSN 1424-8220. doi: 10.3390/s20164405. URL https://www.mdpi.com/1424-8220/20/16/4405.

[15] Jabbar Hussain. Deep learning black box problem. Master's thesis, Uppsala University, 2019. URL https://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Auu%3Adiva-393479.

[16] Texas Instruments. Mobile tracker user guide. https://dev.ti.com/tirex/explore/node?node=A__ASfgi0PR4fN023k7aC732w__radar_toolbox__1AslXXD__LATEST. Last accessed: Okt. 29, 2025.

[17] Texas Instruments. mmwave radar sensors. https://www.ti.com/video/series/mmwave-training-series.html, jun 2024. Last accessed: Nov. 12, 2025.

[18] Cesar Iovescu and Sandeep Rao. *The Fundamentals of Millimeter Wave Radar Sensors*. Texas Instruments, 2020. URL https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf?ts=1761576524490.

[19] Suchdeep Singh Juneja, Girish Vaidya, and Marco Zuniga. Wavetune: Harnessing radar sensors to compose music beats with body gestures. In *2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 138–145, 2024. doi:

10.1109/DCOSS-IoT61029.2024.00029. URL https://doi.org/10.1109/DCOSS-IoT61029.2024.00029.

[20] Boris Kapilevich, Yosef Pinhasi, Michael Anisimov, Boris Litvak, and Danny Hardon. Fmcw mm-wave non-imaging sensor for detecting hidden objects. In *2011 IEEE MTT-S International Microwave Workshop Series on Millimeter Wave Integration Technologies*, pages 101–104, 2011. doi: 10.1109/IMWS3.2011.6061847. URL https://doi.org/10.1109/IMWS3.2011.6061847.

[21] Swapnil Kharche. Kinect skeleton tracking. https://github.com/swapnil0908/Kinect-Skeleton-Tracking, June 2019. Last accessed: Nov. 7, 2025.

[22] Gongjin Lan, Yu Wu, Fei Hu, and Qi Hao. Vision-based human pose estimation via deep learning: A survey. *IEEE Transactions on Human-Machine Systems*, 53(1):253–268, 2023. doi: 10.1109/THMS.2022.3219242. URL https://doi.org/10.1109/THMS.2022.3219242.

[23] Wenhao Li, Riccardo Spolaor, Chuanwen Luo, Yuchao Sun, Huashan Chen, Guoming Zhang, Yanni Yang, Xiuzhen Cheng, and Pengfei Hu. Acoustic eavesdropping from sound-induced vibrations with multi-antenna mm-wave radar. *IEEE Transactions on Mobile Computing*, 24(8):7693–7708, 2025. doi: 10.1109/TMC.2025.3551317. URL https://doi.org/10.1109/TMC.2025.3551317.

[24] Dominik Meier, Christian Zech, Benjamin Baumann, Bersant Gashi, Michael Schlechtweg, Jutta Kühn, Markus Rösch, and Leonhard M. Reindl. Propagation of millimeter waves in composite materials. *IEEE Transactions on Antennas and Propagation*, 68(4):3080–3093, 2020. doi: 10.1109/TAP.2019.2955213. URL https://doi.org/10.1109/TAP.2019.2955213.

[25] Wolfgang Menzel. 50 years of millimeter-waves: A journey of development. *MWJ*, August 2008. URL https://www.microwavejournal.com/articles/6708-50-years-of-millimeter-waves-a-journey-of-development.

[26] Brendan Mesters. Iammuse usage example. https://youtu.be/OufP_pDxQt4, December 2025.

[27] Md Moniruzzaman, Zhaozheng Yin, Md Sanzid Bin Hossain, Hwan Choi, and Zhishan Guo. Wearable motion capture: Reconstructing and predicting 3d human poses from wearable sensors. *IEEE Journal of Biomedical and Health Informatics*, 27(11):5345–5356, 2023. doi: 10.1109/JBHI.2023.3311448. URL https://doi.org/10.1109/JBHI.2023.3311448.

[28] Tewodros Legesse Munea, Yalew Zelalem Jembre, Halefom Tekle Weldegebriel, Longbiao Chen, Chenxi Huang, and Chenhui Yang. The progress of human pose estimation: A survey and taxonomy of models applied in 2d human pose estimation. *IEEE Access*, 8:133330–133348, 2020. doi: 10.1109/ACCESS.2020.3010248. URL https://doi.org/10.1109/ACCESS.2020.3010248.

[29] Instrument of Things. Somi-1 the sound of me. https://instrumentsofthings.com/. Accessed: 2025-11-21.

[30] The European Parliament and the Council of the European Union. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (General Data Protection Regulation) (text with eea relevance). https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng, apr 2016.

[31] Florent Poux and Roland Billen. Voxel-based 3d point cloud semantic segmentation: Unsupervised geometric and relationship featuring vs deep learning methods. *ISPRS International Journal of Geo-Information*, 8(5), 2019. ISSN 2220-9964. doi: 10.3390/ijgi8050213. URL https://www.mdpi.com/2220-9964/8/5/213.

[32] Arindam Sengupta, Feng Jin, Renyuan Zhang, and Siyang Cao. mm-pose: Real-time human skeletal posture estimation using mmwave radars and cnns. *IEEE Sensors Journal*, 20(17):10032–10044, 2020. doi: 10.1109/JSEN.2020.2991741. URL https://doi.org/10.1109/JSEN.2020.2991741.

[33] Bram van Berlo, Amany Elkelany, Tanir Ozcelebi, and Nirvana Meratnia. Millimeter wave sensing: A review of application pipelines and building blocks. *IEEE Sensors Journal*, 21(9):10332–10368, 2021. doi: 10.1109/JSEN.2021.3057450. URL https://doi.org/10.1109/JSEN.2021.3057450.

[34] Haili Wang, Fuchuan Du, Hao Zhu, Zhuangzhuang Zhang, Yizhao Wang, Qixin Cao, and Xiaoxiao Zhu. Here: Heartbeat signal reconstruction for low-power millimeter-wave radar based on deep learning. *IEEE Transactions on Instrumentation and Measurement*, 72:1–15, 2023. doi: 10.1109/TIM.2023.3267348. URL https://doi.org/10.1109/TIM.2023.3267348.

[35] Boxun Yan and Ian P. Roberts. Advancements in millimeter-wave radar technologies for automotive systems: A signal processing perspective. *Electronics*, 14(7), 2025. ISSN 2079-9292. doi: 10.3390/electronics14071436. URL https://www.mdpi.com/2079-9292/14/7/1436.

[36] Faisal Zaman, Ya Ping Wong, and Boon Yian Ng. Density-based denoising of point cloud. In Haidi Ibrahim, Shahid Iqbal, Soo Siang Teoh, and Mohd Tafir Mustaffa, editors, *9th International Conference on Robotic, Vision, Signal Processing and Power Applications*, pages 287–295, Singapore, 2017. Springer Singapore. ISBN 978-981-10-1721-6. URL https://www.researchgate.net/publication/301846888_Density-based_Denoising_of_Point_Cloud.

Appendix A

Standardised Movement Set

In order to get similar data across users and to make sure that all movements were performed, a standardised movement set was devised.

Each position was held for 2-3 seconds. Each row in the table shows a *left hand position* and a *right hand position*, which can both be either "L" for low, "M" for middle, or "H" for high.

Step	Left	Right
1	L	L
2	M	M
3	H	H
4	M	M
5	L	L
6	L	M
7	L	H
8	L	M
9	L	L
10	M	L
11	H	L
12	M	L
13	L	L
14	M	M
15	M	H
16	M	M
17	M	L
18	M	M
19	H	M
20	M	M
21	L	M
22	M	M
23	H	H
24	H	M
25	H	L
26	H	M
27	H	H
28	M	H
29	L	H
30	M	H
31	H	H

Table A.1: **Standardised movement set**

Appendix B

Configuration in Detail

A detailed look into the software and hardware configuration, including:
The specific firmware used:

- What are some of the specs of this firmware
- Why did we originally choose this firmware
- What are some potential downsides to this firmware

We chose the *Mobile Tracking* firmware of TI. This firmware was chosen as it did not produce too few points to analyze, it produced around 30 points per frame, as opposed to some other firmware, which produced lower point totals, at around 10 points per frame. Next to this, most points it produced seemed to be of high value, showing actual information about the scene. There were some firmwares that generated significantly more points (100+ points per frame), but only at active movement near the sensor (Vital Signs Tracking). During testing, it was found that these points were difficult to handle and not the most useful in our situation.

The *mobile tracking* firmware was the best one found during this thesis. There was not enough time to explore many of the potential firmware options, let alone write a custom firmware ourselves, as such, we landed on a "good enough" firmware.

Appendix C

Experimental Setup

Some figures to show the setup by which the experiments were performed. Note, the screen shown in fig. C.1 shows the same content as the screen visible in fig. C.4 while the system is running. The only person in the field of view of either the mmWave radar, or the Kinect during recordings was the user who was using the system at that moment.

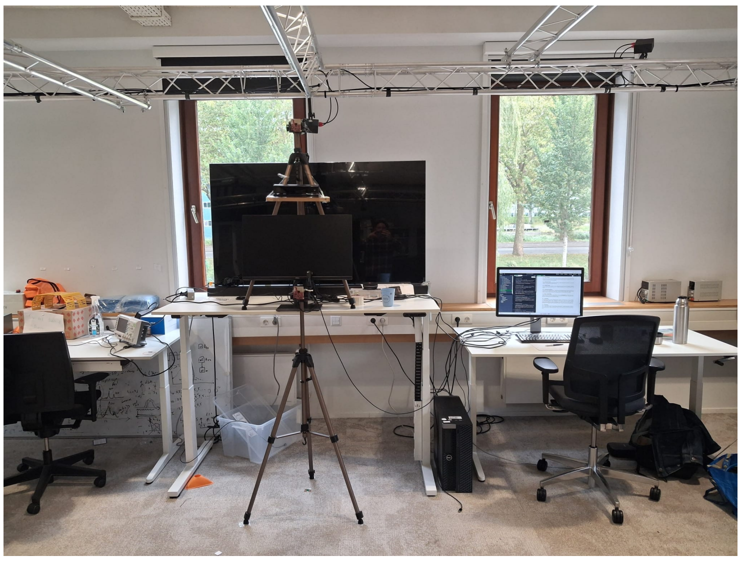


Figure C.1: **Sensor setup, with the mmWave radar at the bottom, screen in the middle, and Kinect at the top**

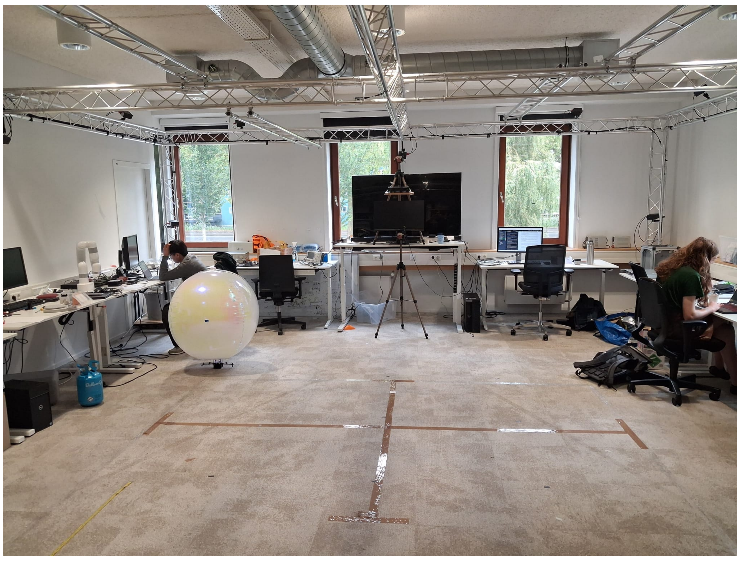


Figure C.2: **Wide view of the room, facing towards the sensors**

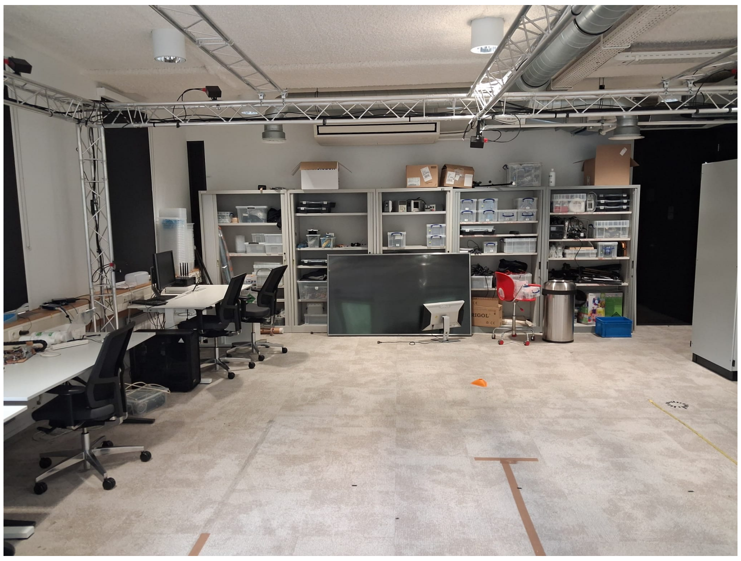


Figure C.3: **Wide view of the room, facing away from the sensors**

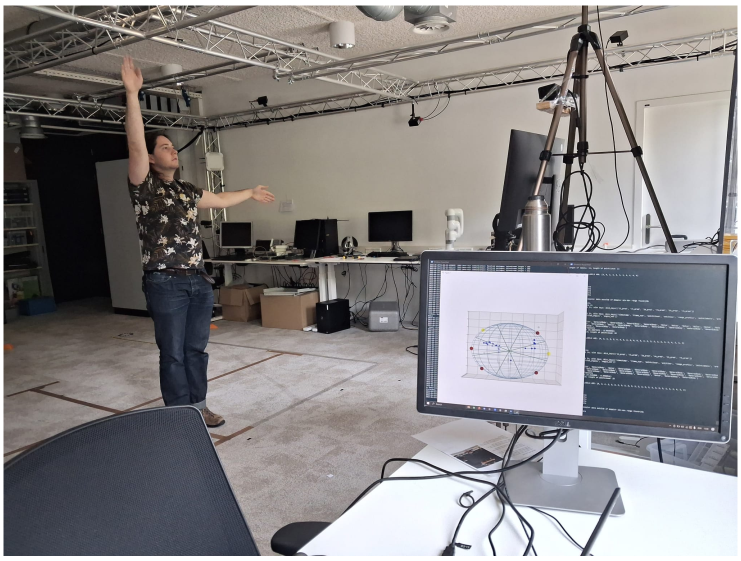


Figure C.4: **View of a researcher using the system.**

Appendix D

User Explanation

The users were given a small *explanation pamphlet* before using the system. This and the next page show that pamphlet.

MillimeterWaveRadar Test

What will you do?

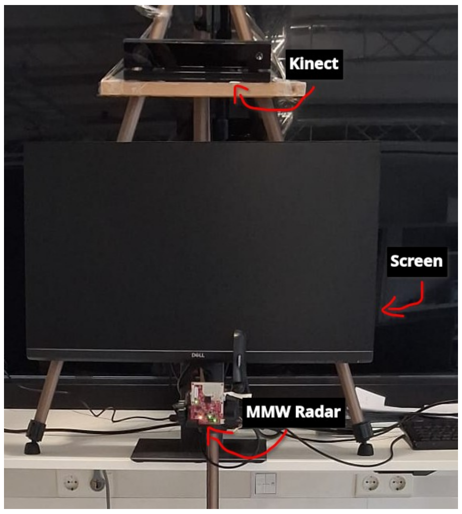
Dear participant, in the experiment you are about to undertake you will be asked to stand in front of a MilliMeterWave Rader (MMW Radar), a Kinect and a screen, and perform a few actions.

Before the system can be used we first need to do a *calibration step*, this will consist of you moving your arms around in circles.

First you will be asked to perform a predetermined set of actions, as dictated by the researcher.

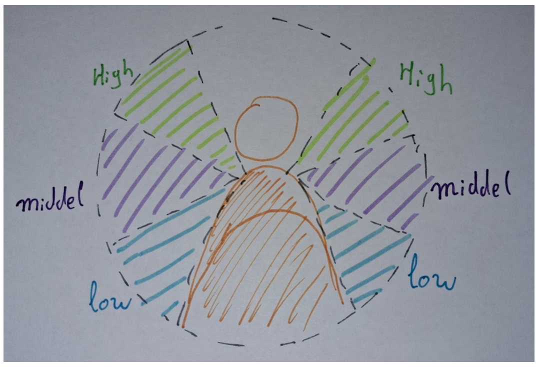
When these predetermined actions have been recorded, you will be free to experiment with the system, and to "make your own music".

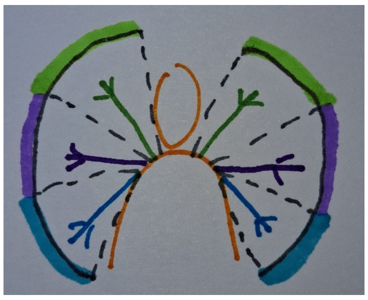
Note: The tuning of the system can be changed, but this does require a restart of the system. If you'd like to try out a different tuning please ask the researcher.



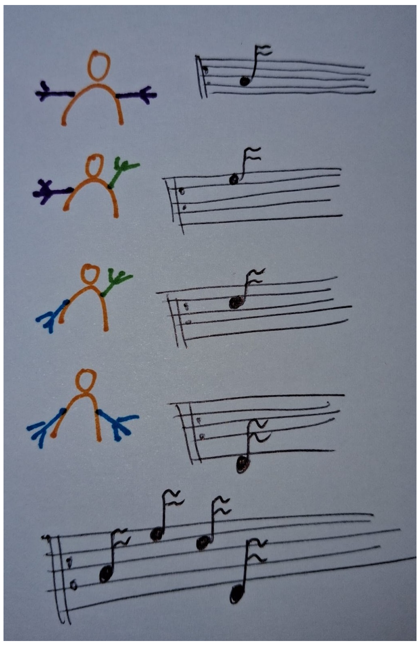
How to use the system

The system registers your movements the best if you “Fan/flap your hands” back and forth, towards and away from the camera.
The system tries to **generate** a *musical note* depending on **where your hands are**.
Specifically, your hands can be in **one** of **three** locations, *low*, *middle*, or *high*.





Depending on the *position* of **both hands** we generate a *music note*.



There are a total of **9 different positions** you can have, most will give different notes, but some might give the same note.

There are a few different “Tunings” available, during the “*free play*” period (after the standardized movements) the system can be put into a different tuning, simply ask the researcher.

tip: Its possible to switch a hand from *high* to *low* without it being *middle* in between, by pulling your hand to your body and extending it again towards the low position (as opposed to moving the stretched out arm down).

Appendix E

MEB configurations

The algorithm to calculate a Minimal Enclosing Ball (MEB) with outliers, also referred to as an *Enclosing Ball* in this thesis, takes three input parameters:

- δ, a tightness bound
- γ, a noise percentage
- z, the number of iterations for which the internal estimation system runs

The algorithm works by iteratively calculating a best guess for the MEB and adding more points to its estimate until it eventually arrives at a reasonable prediction. The *tightness bound* and *noise percentage* inform the system how lenient it can be when adding new points.

The precise values for these variables, which we used in this thesis, were arrived at by manual tuning of the system. If the system is deployed in a different environment, then this tuning may not be optimal anymore, and a retuning may be required. In general, δ should match the expected percentage of noise points. γ should be tuned to be as low as possible without including any obvious noise points. z should start low and be tuned up, until the resulting estimations do not significantly improve anymore.

For our purposes, we found that taking δ to be 0.08, γ to be 0.15, and z to be 50 results in reasonable estimations roughly 90% of the time. We then generate such an estimation 100 times, and average both the ball centre and the ball radius across these 100 trials to arrive at a final estimation. For a mathematical description of the algorithm see [8, Algorithm 1]

Appendix F

Recording Frame Distribution

The different recordings have a different number of frames in total, and will also use a different number of frames for the *calibration* step. The following table shows the exact number of *calibration* frames as well as the number of *interpretation* frames (interpretation frames + calibration frames = total recording frames). The duration of each recording is roughly $\frac{1}{10}$th of the number of frames in the recording, since the mmWave radar records at 10 frames per second, giving an average recording duration of 140 seconds.

The table will use a shorthand notation to refer to recordings. The user for whom the recording is will be specified by 'usr_xx', where 'xx' is a random, unique private identifier of the user. Standard movement set recordings will be specified with 'std', while free play recordings will be specified with 'free'. Each user has made multiple standard recordings, and some users have made multiple free play recordings. To distinguish between the recordings, each recording gets an incremental identifier when both the user and the type match, given as *rec_n* where n is the incremental identifier. Examples:

- "**standard movement set** recording from **user 07** number **2**" becomes "std usr_07 rec_2"
- "**free play** recording from **user 52** number **1 out of 1**" becomes "free usr_52 rec_1"

Recording	**# Calibration Frames**	**# Interpretation Frames**
total	7229	40272
average	212	1184
free usr_07 rec_1	168	853
free usr_07 rec_2	171	1014
free usr_07 rec_3	179	331
free usr_41 rec_1	235	1423
free usr_51 rec_1	187	1322
free usr_51 rec_2	206	1478
free usr_51 rec_3	208	1461
free usr_51 rec_4	179	1431
free usr_52 rec_1	223	668
free usr_52 rec_2	183	863
free usr_61 rec_1	194	1247
free usr_61 rec_2	180	769
free usr_94 rec_1	190	861
free usr_94 rec_2	230	1412
std experienced_user rec_1	171	1003
std experienced_user rec_2	210	1043
std experienced_user rec_3	162	917
std usr_07 rec_1	232	905
std usr_07 rec_2	246	1132
std usr_07 rec_3	211	1203
std usr_41 rec_2	206	1150
std usr_41 rec_3	223	1204
std usr_51 rec_1	191	1117
std usr_51 rec_2	212	1181
std usr_51 rec_3	182	1212
std usr_52 rec_1	204	989
std usr_52 rec_2	189	1147
std usr_52 rec_3	198	1028
std usr_61 rec_1	183	1118
std usr_61 rec_2	151	1291
std usr_61 rec_3	167	1017
std usr_94 rec_1	275	1067
std usr_94 rec_2	223	1142
std usr_94 rec_3	218	1044

Appendix G

User Information

Each user was asked to provide their height and arm span; the results can be seen in the following table.

User ID	Height (in cm)	Arm Span (in cm)
07	186	182
41	179	185
51	185	183
52	181	186
61	188	190
94	158	154