Frequency Modulated Continuous Wave Radar Based Fall Risk Monitoring System

by

Daniel Ilan Copeland, M.D.

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Daniel Ilan Copeland, M.D. Department of Mechanical Engineering May 10, 2024
Certified by:	Brian W. Anthony Principal Research Scientist, Thesis Supervisor
Accepted by:	Nicolas Hadjiconstantinou Chairman Graduate Officer, Department of Mechanical Engineering

Frequency Modulated Continuous Wave Radar Based Fall Risk Monitoring System

by

Daniel Ilan Copeland, M.D.

Submitted to the Department of Mechanical Engineering
on May 10, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

ABSTRACT

Falls represent a significant health risk, especially for the elderly. Fortunately, interventions have been shown to decrease falls when clinicians identify at-risk patients. However, factors such as medication changes, illness, and injuries can rapidly increase fall risk, making timely clinical identification and subsequent interventions challenging to implement. Our study introduces a comprehensive approach to assessing fall risk using a frequency-modulated continuous-wave (FMCW) radar system, addressing the need for frequent, low-cost, long-term balance monitoring solutions. This technology is compared with ground-truth contact-based lab sensors like force plates and motion capture systems, establishing a foundation for accurate balance assessments in home settings. In our cross-sectional analysis, participants performed the one-legged stand test (OLST) with simultaneous data collection from FMCW radar, force plates, and motion capture systems. By integrating the FMCW radar with machine learning algorithms, we achieved a 98.4% accuracy in identifying OLST foot movements and an R-squared of 0.70 in predicting force plate patterns, demonstrating the system's nuanced capability for balance performance evaluation. Additionally, we examine the efficacy of combining radar technology with machine learning to identify movements similar to those performed in fitness, clinical, and rehabilitation settings. We also explore the use of simulations for optimizing radar system configurations. This thesis demonstrates the effectiveness of FMCW radar technology in laboratory settings and its potential for home-based health monitoring. The study highlights the transformative potential of integrating radar technology with machine learning through detailed experimentation and analysis, offering a versatile tool for health monitoring and fall risk assessment.

Thesis supervisor: Brian W. Anthony
Title: Principal Research Scientist

Acknowledgments

I would like to sincerely thank all the yogis who generously contributed their time and effort to this study. Their dedication, flexibility, and commitment to the research were invaluable, and their participation was essential to the success of this thesis.

I am particularly grateful to Evan Linton and Dr. Shawn Zhang for their help with the experimental set-up, data collection, and data processing.

I would also like to acknowledge Nurse Tatiana Urman for her enthusiastic assistance in obtaining informed consent from the participants and helping ensure data collection ran smoothly. Her professionalism and dedication ensured the study was conducted ethically and efficiently.

I want to give special thanks to my UROP, Hector Lugaro, for helping with the actuator and cueing videos. His technical skills and dedication greatly contributed to the project's progress.

I would also like to thank Dr. Praneeth Namburi, Professor Caitlin Mueller, Professor Faez Ahmed, Professor Alex Slocum, and Ous Abou Ras for their invaluable insights throughout this project's many stages.

I thank my principal investigator, Dr. Brian Anthony, for his guidance and expertise on FMCW radar technology and for providing general research direction. His support and mentorship were crucial to my and this project's development.

I would like to thank my MechE friends, the Makerworks community, and MIT's coaching cohorts. These communities made my masters a joy and imbued me with a sense of purpose and togetherness.

Finally, I would like to thank my family, and specifically, my wife, Isabelle, for sticking with me through a bit more grad school and my dog, Millie, for her pure joy and licks and for getting me off my computer and outside to touch the grass.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

1.1 Significance of Falls in the Elderly

1.1.1 Current Status Falls and Risk Monitoring

Falls among the elderly and those with balance impairments represent a significant public health issue, leading to increased morbidity and mortality and accruing over $50 billion in annual healthcare costs in the US alone [1]. The World Health Organization reports that approximately 28–35% of people aged $\geq$ 64 years experience at least one fall every year [2]. Fortunately, detecting an increase in fall risk and intervening early has been shown to reduce falls by up to 24% [3]–[6]. Traditional methods for assessing fall risk, such as the Timed Up and Go (TUG) test, the Berg Balance Scale (BBS), and force plate tests, require clinical settings and expert oversight, which can be resource-intensive and challenging to obtain more frequently than once a year. Frequent at-home assessments could identify increases in fall risk due to medication changes, injury, or illness between clinical visits [7].

As the global population ages, the demand for accessible, frequent, and non-intrusive solutions for early fall risk detection becomes increasingly urgent [8]. While self-reported questionnaires and in-home assessments are supplementary methods, their subjectivity and resource-intensive nature remain significant drawbacks. Addressing these challenges, this thesis introduces the innovative use of Frequency-Modulated Continuous-Wave (FMCW) radar as a precise, non-contact tool for proctoring the One-Legged Standing Test (OLST) [Figure 1.1].

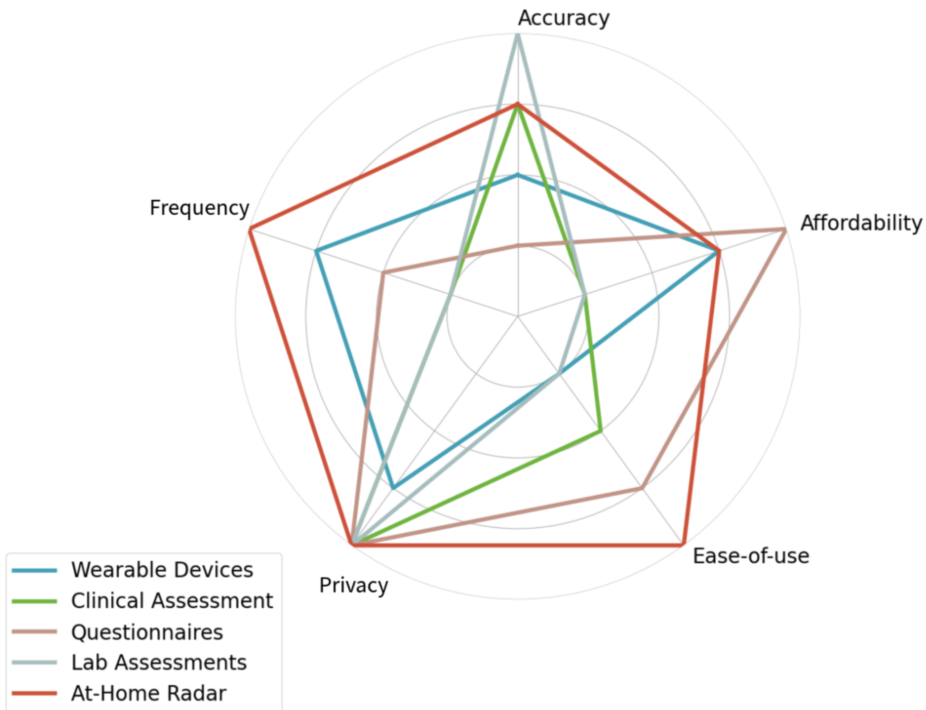

Figure 1.1: Spiderweb diagram illustrating the comparative analysis of different balance assessment tools across five key metrics: Accuracy, Affordability, Ease-of-use, Privacy, and Frequency. 'At-Home Radar' (red) is emphasized, showcasing its relative positioning against traditional tools like 'Wearable Devices', 'Clinical Assessment', 'Questionnaires', and 'Lab Assessments'. This visualization underpins the discussion on the viability and advantages of 'At-Home Radar' systems for fall risk assessment and monitoring in the paper.

Traditionally overseen by clinicians, the OLST provides crucial insights into an individual's balance, postural control, and ability to maintain equilibrium and correlates strongly with falls and mortality [Figure 1.2] [9], [10].

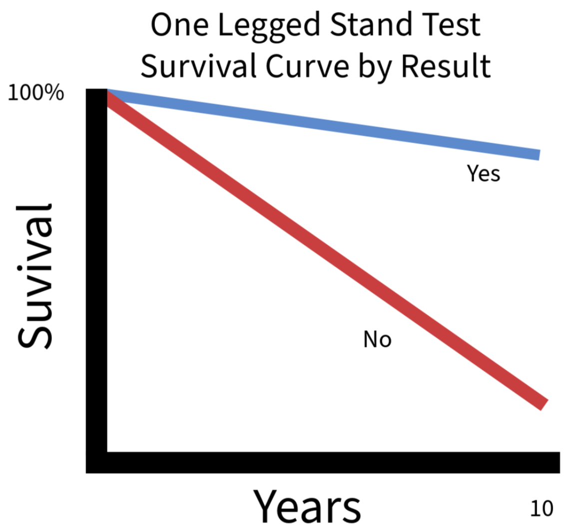

Figure 1.2: A Sketch of a Kaplan-Meier survival curve depicting the association between the ability to perform a successful 10-second OLST and long-term survival rates. Individuals who passed the OLST ('Yes') demonstrate a markedly higher survival probability over the course of 10 years, as represented by the blue line. Conversely, those unable to maintain the stance ('No') show a significantly decreased survival probability, illustrated by the red line. This graph highlights the OLST's prognostic value of a simple physical performance measure in predicting longevity in middle-aged and older adults. The test's predictive capability for mortality may stem from its implicit assessment of overall physical fitness, stability, muscle strength, and functional health status, which are crucial indicators of an individual's health span and resilience against age-related declines and fall risk. [9].

1.1.2 Proposal of Radar as a Fall Risk Monitoring Tool

Commonly utilized in the automotive and aviation industries [11], FMCW radar is an advanced sensing technology that is beginning to be leveraged for health monitoring, such as for vital signs [12]–[14]. This study compares a radar-based measurement approach to gold-standard balance assessment tools, such as force plates and motion capture (MOCAP) technologies, confirming its ability to accurately proctor an at-home OLST and track balance-related metrics [15]. Successfully implementing this technology to enhance early detection and intervention could result in significant healthcare savings and, more importantly, help preserve the health and independence of the world's increasingly geriatric population [8].

1.2 FMCW Radar

1.2.1 Basics of Radar

Principle of Operation

radar operates on the principle of emitting microwave-frequency electromagnetic waves, which then reflect off objects (often termed "targets") and return to the radar receiver.

By analyzing the properties of the returned signal, information about the target's position, speed, and other characteristics can be deduced.

Time of Flight and Determination of Distance

The time taken for the emitted radio wave to travel to the target, reflect off it, and return to the radar receiver is known as the "time of flight." Given that radio waves travel at the speed of light (c), the distance (d) to the target can be calculated using the equation

$$d = \frac{c \times \text{Time of Flight}}{2} \tag{1.1}$$

The division by 2 is to account for the round-trip travel of the wave. This equation gives a direct measure of the distance based on the time taken for the signal to return [Fig. 1.3].

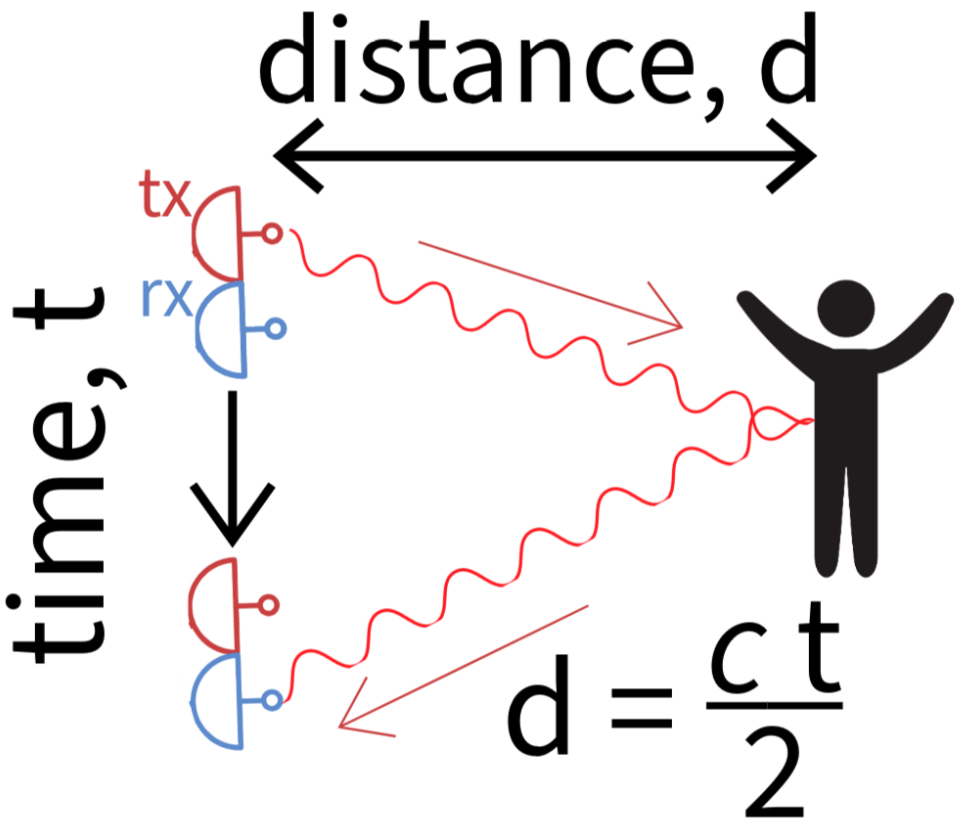

Figure 1.3: Illustration of the time of flight and distance determination using RADAR.

1.2.2 Doppler Effect

Introduction to the Doppler Effect

The Doppler Effect is a phenomenon observed in wave mechanics, where the frequency of a wave changes for an observer moving relative to the source of the wave. In radar systems, the waves reflected off a target are considered a new source, which causes a shift in the observed frequency based on the relative velocity between the radar and the target. This shift is crucial for determining the velocity of a moving target, making the Doppler Effect central to radar systems.

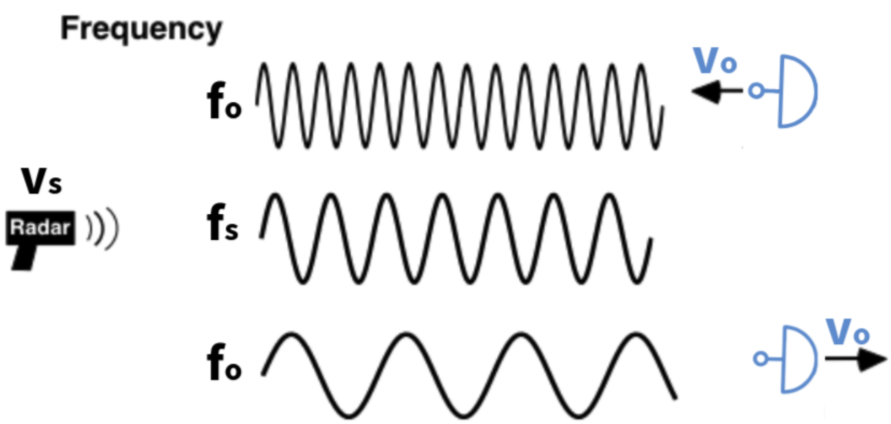

Figure 1.4: This image illustrates the Doppler Effect as observed by a moving receiver (right) moving relative to the radar emitting source (left). The radar emits waves at a certain frequency, denoted by f_s, in the middle waveform. In the top waveform, the receiver moves towards the emitter at velocity V_o, the waves are compressed, resulting in a higher frequency f_o observed by the receiver, and the waves are more closely packed. Conversely, in the bottom waveform, when the object moves away from the radar at velocity V_o, the waves are stretched, leading to a lower frequency f_o observed by the receiver. and the waves are more spread out. The change in frequency between the source and receiver due to the relative motion is known as the Doppler Effect, which is central to radar technology for determining the velocity of objects.

As illustrated in Figure 1.4, the radar detects and interprets the change in frequency of the returned wave to determine the relative motion of the target.

Mathematical Representation

- **Definition:** If a source of frequency f_s is moving with a velocity v_s relative to an observer (with velocity v_o), the frequency f_o observed by the observer is given by:

$$f_o = \frac{(c + v_o)}{(c + v_s)} f_s \tag{1.2}$$

  where c is the speed of the wave in the medium.

- **When both source and observer are moving towards each other:**

$$f_o = \frac{(c + v_o)}{(c - v_s)} f_s \tag{1.3}$$

- **When both source and observer are moving away from each other:**

$$f_o = \frac{(c - v_o)}{(c + v_s)} f_s \tag{1.4}$$

Application in RADAR

In RADAR systems, the Doppler Effect is employed to determine the velocity of a moving target. When the radar transmits a signal that hits a moving object, the reflected signal's frequency shifts. By analyzing this shift, the radar can calculate the object's velocity. This is especially crucial in applications like air-traffic control, and automotive RADAR where the vehicles' position and velocity need to be monitored.

The Doppler shift Δf in radar systems for a target moving directly towards or away from the radar is given by:

$$\Delta f = \frac{2v_r}{c} f_0 \tag{1.5}$$

where:

- Δf is the Doppler frequency shift.
- v_r is the radial velocity of the target.
- f_0 is the transmitted frequency.
- c is the speed of light (approximately 3×10^8 m/s).

1.2.3 Differences Between FFT, Fourier Series, and Fourier Transform

Fourier Transforms are mathematically foundational to FMCW Radar technology. However, similar terminology here can be confusing. In signal processing and mathematical analysis, the Fourier series, Fourier Transform, and Fast Fourier Transform (FFT) are three fundamental techniques used for representing and analyzing functions in terms of sinusoids. While they are closely related, there are important distinctions among them.

1. **Fourier Series**:
   The Fourier series decomposes a periodic function $f(t)$ with period T into a weighted sum of sines and cosines. Its representation is:

$$f(t) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t)] \tag{1.6}$$

   Where:

$$a_0 = \frac{1}{T} \int_{-T/2}^{T/2} f(t)\, dt \tag{1.7}$$

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos(2\pi n f_0 t)\, dt \tag{1.8}$$

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(2\pi n f_0 t)\, dt \tag{1.9}$$

   f_0 is the fundamental frequency equal to $1/T$.

2. **Fourier Transform**:
   The Fourier Transform extends the idea of the Fourier series to non-periodic functions, producing a continuous frequency spectrum. For a given function $f(t)$, its Fourier Transform $F(\omega)$ is given by:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt \tag{1.10}$$

   Its inverse, which retrieves $f(t)$ from $F(\omega)$, is:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t}\, d\omega \tag{1.11}$$

3. **Fast Fourier Transform (FFT)**:
   The FFT is not a distinct transform, but rather an efficient algorithm to compute the Discrete Fourier Transform (DFT) and its inverse. The DFT maps a sequence of N complex numbers to another sequence of N complex numbers and is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn} \tag{1.12}$$

   Where $x[n]$ is the signal in time domain and $X[k]$ is its frequency representation. The FFT reduces the computational complexity of naive DFT calculation from $O(N^2)$ to $O(N \log N)$, making it feasible for real-time processing and analysis of large datasets.

1.2.4 Fundamentals of FMCW Radar (Frequency Modulated Continuous Wave Radar)

Frequency Modulation

In FMCW radar, instead of emitting a continuous signal like Doppler radar or a short pulse and waiting for its return like pulsed radar, the system continuously transmits an EM signal whose frequency is modulated over time. This modulation can be linear, sawtooth, triangular, among other patterns. The primary reason for modulating the frequency is to encode the time of transmission into the transmitted signal frequency. This allows for the continuous measurement of both the distance and velocity of targets.

Signal Mixing and Intermediate Frequency in FMCW Radar

In FMCW radar, signal mixing is an indispensable mechanism combining the transmitted and received signals to derive the intermediate frequency (IF), encapsulating essential data concerning the range and velocity of detected objects. During the mixing process, the received signal, which carries a time delay due to traveling to and back from an object, as well as a potential Doppler shift from the object's velocity, is mathematically combined with the transmitted signal. The equation for the mixing can be given as:

$$x_1 = \sin(\omega_1 t + \phi_1) \tag{1.13}$$

$$x_2 = \sin(\omega_2 t + \phi_2) \tag{1.14}$$

$$x_{\text{out}} = x_1 \cdot x_2 \approx \sin((\omega_1 - \omega_2)t + (\phi_1 - \phi_2)) \tag{1.15}$$

Where:

- x_1 and x_2 are the two input sinusoids.
- x_{out} is the output after mixing, IF in FMCW radar.
- ω_1 and ω_2 are the angular frequencies of the input sinusoids x_1 and x_2 respectively.
- t is time.
- ϕ_1 and ϕ_2 are the phase offsets of the input sinusoids x_1 and x_2 respectively.

This IF signal is calculated for each unit of FMCW radar measurement, known as a chirp. The chirp contains range and velocity information that is then stored in a standard data structure known as a Radar Datacube.

Creating a Radar Datacube

The IF signal x_{out} produced by each mixed FMCW chirp is stored in a single "Fast Time" dimension of the radar data cube using the data processing shown in Figure 1.5. Successive IF signals from successive chirps are stored in a second "Slow Time" dimension. A set number of successive processed chirps are grouped together to create a frame. Many FMCW radars have multiple receiving antennae or channels (MIMO). Each receiving antennae's data is stored in another dimension such that fast and slow time arrays are stacked to make the 3-dimensional data storage object.

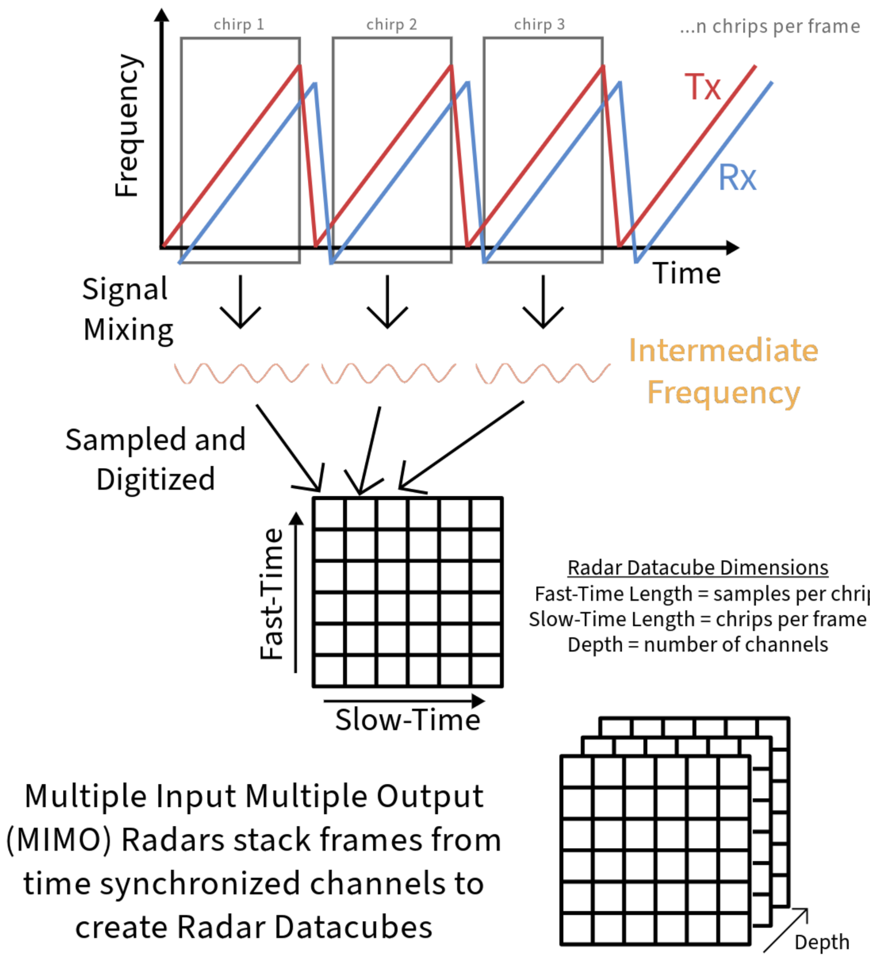

Figure 1.5: A depiction of the MIMO radar processing sequence to create a Radar Datacube. A series of chirps (Tx) are transmitted and their reflections (Rx) are received. The resulting IF signal from the mixing process is sampled and digitized along fast-time. These samples populate a data matrix with fast-time and slow-time dimensions, corresponding to individual chirps and chirp sequences, respectively. In MIMO radar systems, datacubes from multiple time-synchronized channels are then stacked to introduce a third dimension—'Depth'. This additional dimension represents multiple channels and is essential for enhanced spatial resolution and precise target location identification in MIMO radar applications.

Extracting Range and Velocity from a Radar Datacube

Radar systems process received signals to extract information regarding the range, velocity, and direction of detected targets. An important principle of signal processing is that all signals, regardless of their complexity, can be decomposed into the addition of simple sine waves of varying frequency and phase. The Fast Fourier Transform (FFT) [Equation 1.11] is an efficient algorithm that shows which frequencies comprise a given signal.

In radar systems, the FFT plays an instrumental role in converting the time-domain representation of received signals into the frequency domain, enabling the extraction of essential parameters such as target range and velocity.

Range Estimation

Beat frequency refers to the dominant frequency found in the intermediate frequency due to the reflection of an object. The relationship between the beat frequency and range R to the target can be expressed as:

$$R = \frac{f_{b_{range}} c}{2Bt_{chirp}} \tag{1.16}$$

Where:

- $f_{b_{range}}$ = Beat frequency due to range [Hz]
- t_{chirp} = Length of the chirp [s]
- B = Modulation bandwidth [Hz]
- R = Distance to the target [m]
- c = Speed of light in a vacuum [m/s]

Velocity Estimation using Phase Shift

For a target moving at a velocity, V, between two consecutive chirps, the change in the received phase can be calculated as:

$$\Delta\phi = \frac{4\pi\Delta d}{\lambda} \tag{1.17}$$

Where:

- $\Delta\phi$ = Phase difference between the received signals of two consecutive chirps
- Δd = Change in distance due to velocity between those chirps
- λ = Transmitted signal's wavelength

For the object's movement between the chirps:

$$V = \frac{\Delta d}{t_{chirp}} \tag{1.18}$$

By combining the two equations, the velocity due to the noticed phase shift between two chirps can be calculated as:

$$V = \frac{\Delta\phi\lambda t_{chirp}}{4\pi} \tag{1.19}$$

Range Resolution

Range resolution is the radar's ability to discern between two objects in proximity regarding distance. A radar system with high range resolution can effectively differentiate between two closely spaced targets. Mathematically, the range resolution (Δr) is inversely related to the bandwidth (B) of the transmitted signal:

$$\Delta r = \frac{c}{2B} \tag{1.20}$$

In FMCW RADAR, a substantial bandwidth is achieved by modulating the frequency across a broad range, linearly increasing range resolution.

Range-Doppler Map Generation

Range-Doppler Maps (RDM) are grid-like heat maps with x and y axes binned by received signal intensity across ranges, and velocities. The resolution of these bins is based on the fundamentals of the radar described above. RDMs are generated in the steps described in Figure 1.6. Because RDMs encode important position and velocity information and change over time, they have been used as input data for machine learning algorithms that analyze FMCW radar data [16], [17].

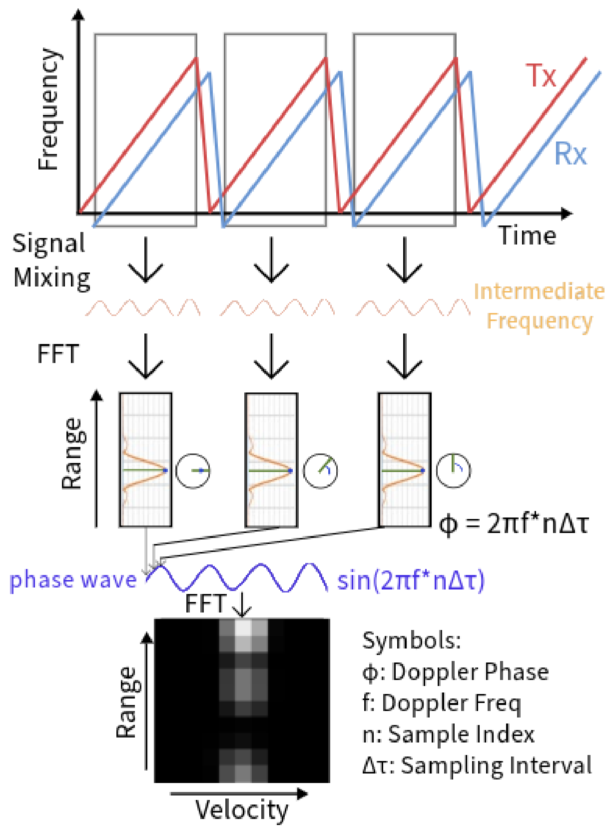

Figure 1.6: Processing steps for RDM generation using FMCW radar. The process begins with the transmission (Tx) of a linear frequency-modulated chirp and the reception (Rx) of the echo from the target, depicted over time. The received signal, which exhibits a frequency shift relative to the transmitted signal due to the round-trip delay and the Doppler effect from target motion, is mixed with the Tx signal to produce an Intermediate Frequency (IF) signal. The frequency of the IF signal is proportional to the target's range. Applying a Fast Fourier Transform (FFT) to the IF signal yields the range spectrum for each chirp. Subsequent FFT analysis across chirps reveals the Doppler frequency shift, indicated by changes in phase, which corresponds to the target's velocity. The final 2D FFT output provides a Range-Doppler map, where the peak positions within the respective range and velocity bins identify the target's range and velocity. The Doppler phase shift (ϕ) is related to the Doppler frequency (f) by the equation $\phi = 2\pi f \cdot n\Delta\tau$, where n is the sample index and $\Delta\tau$ is the sampling interval. This phase shift, represented by the sinusoidal function $\sin(2\pi f \cdot n\Delta\tau)$, encodes the velocity information of the target. [18].

1.2.5 Human Body as a Radar Reflector

Radar Reflectivity of Humans

The Radar Cross Section (RCS) measures how detectable an object is with a radar and has units $[m^2]$ ([19]). A larger RCS indicates that an object is more easily detectable. Considering a scenario where an individual stands directly in front of a radar system indoors with an unobstructed line of sight, the Radar Cross Section (RCS) primarily depends on the clothing material and the posture of the human. RCS can only be computationally calculated for simple geometric bodies.

Basic RCS Equation

The radar cross-section (RCS) is commonly given by the equation:

$$P_r = \frac{P_t \cdot G_t \cdot \sigma \cdot A_r}{(4\pi)^2 \cdot R^4} \tag{1.21}$$

where:

- P_r is the power received by the radar.
- P_t is the transmitted power of the radar.
- G_t is the gain of the transmitting antenna.
- σ is the RCS of the target (in this case, a human).
- A_r is the effective aperture of the receiving antenna.
- R is the range from the radar to the target.

Worst Case Scenario: Minimized RCS

- **Clothing Material**: Non-reflective and absorbent materials can minimize the RCS.
- **Posture**: Turning sideways might present a smaller cross-sectional area than facing the radar head-on.
- **Estimated RCS Value**: For a typical human without reflective clothing, given the various factors, an RCS might be around $\mathbf{1\ m}^2$.

Best Case Scenario: Maximized RCS

- **Reflective Clothing**: Metallic or reflective materials can drastically increase the RCS.
- **Posture**: Facing the radar directly can maximize the RCS.
- **Estimated RCS Value**: With highly reflective clothing and a direct orientation towards the radar, the RCS might approach $\mathbf{2\ m}^2$.

1.2.6 FMCW Radar in Healthcare

Applying FMCW radar in healthcare is a relatively new frontier. Recent studies have begun exploring FMCW radar's potential in monitoring vital signs, detecting falls in real-time, recognizing gestures, and assessing gait characteristics [16], [20]–[24]. In addition to being non-contact, radar can penetrate non-metallic objects, making it appealing for unobtrusive monitoring in home environments [22], [25].

1.2.7 Machine Learning and FMCW Radar

Raw and preprocessed FMCW radar data, despite their inherent noise and volume, are highly conducive to machine learning (ML) techniques [17]. Particularly, the utilization of Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs) has advanced the accuracy of radar monitoring systems [17], [26]. LSTMs, designed to analyze time-series data, are ideally suited for interpreting the temporal patterns of vital signs and movements detected by radar [24], [27], while CNNs, designed to process spatial data, are widely used for classifying human activities [28]–[30]. These ML building blocks lay the foundation for the development of health monitoring systems that can analyze evolving patterns over time. By harnessing the complementary strengths of LSTM and CNN models, the fusion of ML with FMCW radar technology stands poised to revolutionize patient care, ushering in innovative approaches to remote health assessments and elderly care management, as evidenced by recent studies [12], [20], [31], [32].

1.3 Force Plates and Motion Capture

Force plates have been the gold standard in objectively assessing balance and postural control. Researchers can infer information about the person's balance and risk of falling by measuring the ground reaction forces generated by a person's movements [33]. Center of pressure (COP) analysis, derived from force plate data, allows for a nuanced understanding of balance and postural control by providing insights into weight distribution and shifting indicative of instability [34]. Similarly, MOCAP systems provide detailed kinematic data by tracking body movements in three dimensions. These technologies have contributed substantially to our understanding of human balance and have been used extensively in research to identify fall risk factors [35]. For example, force plate-based COP tracking has been used to detect postural stability decline and increased fall tendency in patients with Parkinson's disease [36]. However, their application is typically confined to labs and clinics, with highly specialized equipment and staff [15].

1.4 Study Motivation

This thesis leverages the integration of radar, motion capture, and force plate technologies to provide a sophisticated approach to health monitoring and movement analysis. Motivated by the necessity for non-invasive, accurate assessments within healthcare, particularly for elderly fall risk detection, this research explores the multifaceted applications of these technologies through three focused analyses.

1. The first study investigates a robust system capable of proctoring the One-Leg Stand Test (OLST), a critical measure of balance and stability among elderly populations that provides insights into fall risk and prevention.

2. The second analysis classifies movement types, a central task for activity recognition in real-time health monitoring systems.

3. The third and final study focuses on optimizing radar positioning through simulation to maximize data accuracy and system efficiency.

These interconnected analyses address significant gaps in current monitoring practices and help pave the way for future advancements in personalized healthcare technologies. The ensuing chapters will delve into each study in detail, illustrating RADAR technology's expansive potential to transform health monitoring.

Chapter 2

Data Collection

2.1 Participants

Fifteen participants (age range: 18-31 years, 10 males, 5 females) with varying levels of self-reported balance were recruited for the study. Participants were required to be physically healthy and capable of performing the yoga tree pose, a one-legged standing balance pose, without assistance. The Institutional Review Board approved the study protocol (#1911000055), and written informed consent was obtained from all participants before data collection commenced.

2.2 Sensing Modalities

Data was collected in a controlled laboratory environment with a MOCAP system (Qualysis, Inc. Göteborg, Sweden) with integrated force plates (Bertec Corporation, Columbus, Ohio) and a 4-channel 24GHz FMCW DemoRad radar (Analog Devices, Inc. Wilmington, Massachusetts). We used a 24GHz FMCW radar because it provides high-resolution detection through clothing and non-metallic objects, and emits low-power, non-ionizing radiation, deemed safe by international health standards [37]. The MOCAP system is configured with 12 infrared cameras positioned around the laboratory and 2 RGB cameras on movable tripods. Eighteen reflective markers were placed on the participants' major joints to track their movements [Figure 2.1]. The radar system was positioned 5 meters in front of the participants at a height of 1 meter to allow the entire body to be within the radar's field of view.

MOCAP and force plate sensors were calibrated before each data collection to ensure alignment with the global coordinate system and consistency in representing force vectors and moments.

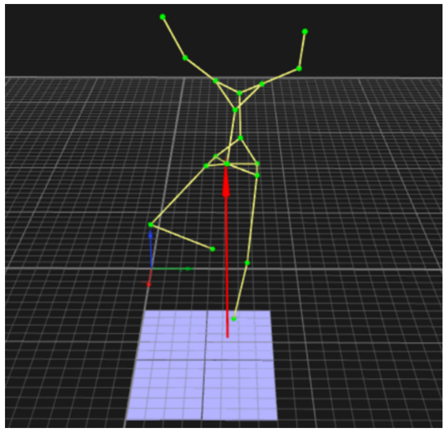

Figure 2.1: Example of 18 Motion Capture Markers and Force Plate Vector During a Participant's Left One-Legged Tree Pose

2.3 Study Procedure

2.3.1 Study Activities

Upon arriving at the laboratory, participants were briefed on the study and the data collection procedures. A brief warm-up routine was conducted to minimize injury risk. After the warm-up, we instructed participants to perform the yoga tree pose by placing the sole of one foot against the opposite leg's inner thigh, with arms remaining raised above their heads [Figure 2.2].

Figure 2.2: Example of a participant on force plates, standing and in left one-legged tree pose. These were the two states the participants were asked to transition between for the OLST.

This version of the OLST was chosen to challenge the healthy participant's balance skills.

The participants did not use their hands to help position their raised leg to avoid adding noise to the radar data. Six data captures were performed by each participant, three times on each leg (L/R).

During each of the six data captures, the participants challenged their balance by executing two short tree poses, holding stable for two seconds, and executing a more prolonged tree pose for five seconds while positioned on two force plates, one under each foot. This sequence was chosen to challenge the dynamic and static components of the postural control.

In addition to OLST Mountain to Tree (L/R), five other transition movements were performed by the participants: Cat to Cow, Cresent Lunge to Warrior II (L/R), and Forward Fold to Mountain. They were instructed to perform the moves to the best of their abilities.

Visual and audio cues were provided to guide the participants through the pose sequences and to ensure approximate temporal alignment between participants' performances.

2.3.2 Study Hardware

At the beginning and end of each data capture segment, we placed and cycled a radar reflective linear actuator equipped with a MOCAP marker to precisely synchronize the radar and the combined MOCAP/force plate system [Figure 2.3].

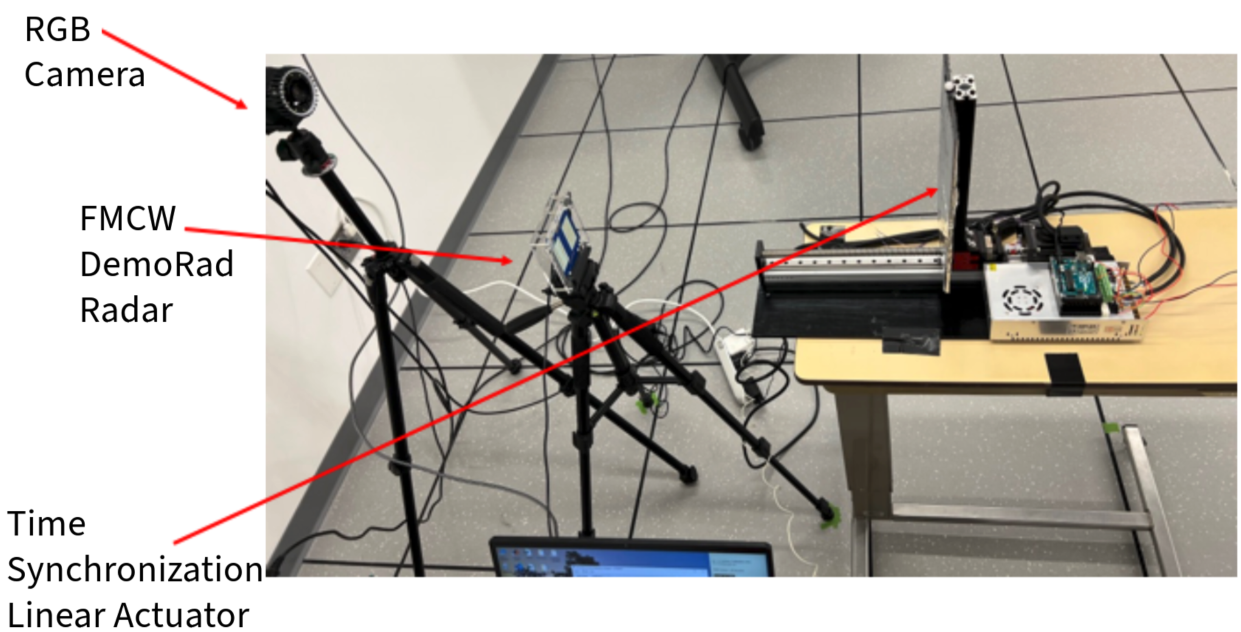

Figure 2.3: FMCW RADAR, RGB Camera, and MOCAP to Radar Time Synchronization Linear Actuator Setup [A.1].

The MOCAP, force plate, and radar systems recorded data simultaneously, allowing for a robust reference analysis of balance and stability.

Chapter 3

Study Analyses

3.1 At Home Radar-Based Fall Risk Monitoring

3.1.1 Motivation

The existing literature thoroughly documents the use of force plates and MOCAP systems for fall risk assessment in clinical settings, highlighting their efficacy and reliability [15]. Similarly, FMCW radar technology has been explored for various healthcare applications, including detecting gestures [17], [19] and vital signs [23], [25], [32], demonstrating its versatility and potential in non-invasive patient monitoring. However, a comprehensive synthesis of these technologies—utilizing the precision of contact sensors to train radar-based algorithms for fall risk evaluation in non-clinical, domestic settings—remains underexplored [12], [13]. This analysis showcases FMCW radar's capability in characterizing fall risk within domestic settings, coupled with the prognostic power of gold standard methodologies [38].

3.1.2 Methods

Radar Data Preprocessing

Raw data from each sensor system underwent the following preprocessing steps. We processed our radar data to suppress noise using a Hanning Window and generated Range-Doppler Maps (RDMs) the steps described in Figure 1.6, [A.2] [39].

We cropped the four-channel RDMs to remove range data beyond 6 meters and velocity data faster than $\pm$ 20 meters/seconds. We separated each 4-dimensional RDM (channels, frames, height, width) by channels into four 3-dimensional RDMs (frames, height, width) [Figure 3.1].

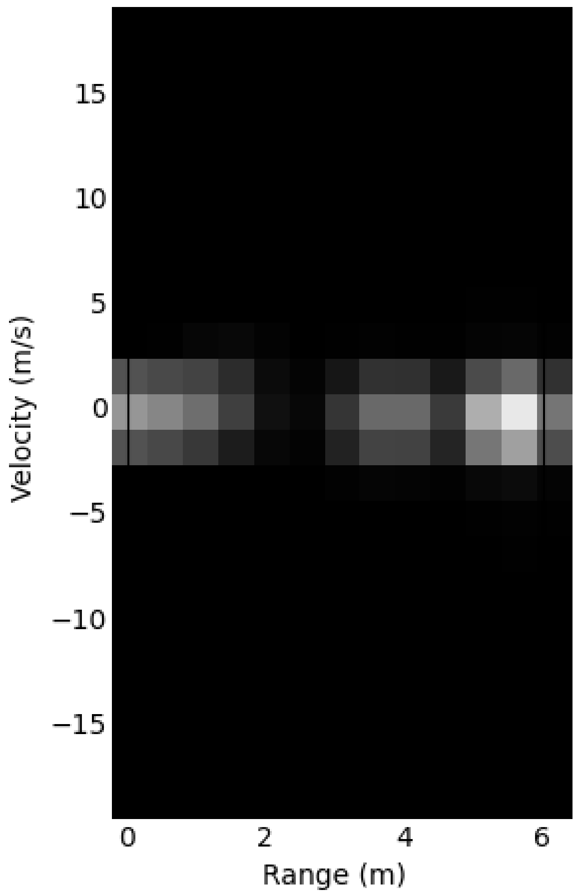

Figure 3.1: Range-Doppler Map (RDM) showcasing a stationary human target detected at approximately 5 meters. The map encodes intensity in grayscale values, with lighter areas indicating stronger radar returns. The x-axis represents the target range from the radar system, and the y-axis indicates the radial velocity, where stationary targets appear along the zero velocity line. The depicted data points around the 5-meter mark on the x-axis and near-zero velocity on the y-axis suggest the presence of a person standing relatively still with respect to the radar's position.

Multimodal Data Acquisition System Synchronization

We used an FMCW radar system alongside a combined MOCAP and force plate system for data acquisition. Both systems independently recorded and timestamped their own data streams. To synchronize the data streams, we employed feature detection by convolving the distinctive movement signature of the radar-reflective linear actuator at the commencement and conclusion of each data capture session [Figure 2.3].

One-Leg Standing Sequence Tagging

After synchronizing the data, we annotated our one-legged standing dataset with ground-truth motion characteristics. This involved identifying the precise times and corresponding frames for key events such as foot-lift, start-of-stability, end-of-stability, and foot-touchdown, allowing us to discern foot-up (FU) and foot-down (FD) movements and characterize participants' balance during the stability phase [Figure 3.2]. The time intervals between foot-lift and start-of-stability mark the initiation of the FU movement, while those between end-of-stability and foot-touchdown signify the conclusion of the FD movement. Additionally, the stability phase encompasses the duration between the start-of-stability and end-of-stability.

Methodological Frameworks for Identifying Foot-Lift and Foot-Touchdown Times in Force Plate Data

We designed a preprocessing, semi-automated labeling framework utilizing synchronized MOCAP and force plate data to obtain the FU and FD movements' base truth start and end times. This effort was critical for labeling the FMCW radar dataset to train models to recognize foot movement patterns. Two force plates, one under each foot, pinpointed foot-lifts and foot-touchdowns. The data labeling process involved:

Importation After loading, we denoted the role of each force plate data capture as the lifted foot or the foot on which the participant was standing.

Foot Lift Identification We applied dynamic thresholding to the foot-lift force plate data to discern changes in force plate values indicative of foot-lifts (non-zero to zero transitions) and foot-touchdowns (zero to non-zero transitions).

Event Validation Logic We validated foot-lift and foot-touchdown times to ensure a sequence of meaningful events, such as ensuring that a foot-touchdown time followed after a foot-lift time.

Manual Review We reviewed each foot-lift and foot-touchdown time using tracked marker positions and force plate data, verifying the timing and accuracy of detected events.

This structured, multimodal approach to preprocessing ensured correct base-truth foot-lift and foot-touchdown times for training models capable of discerning foot movements within radar data.

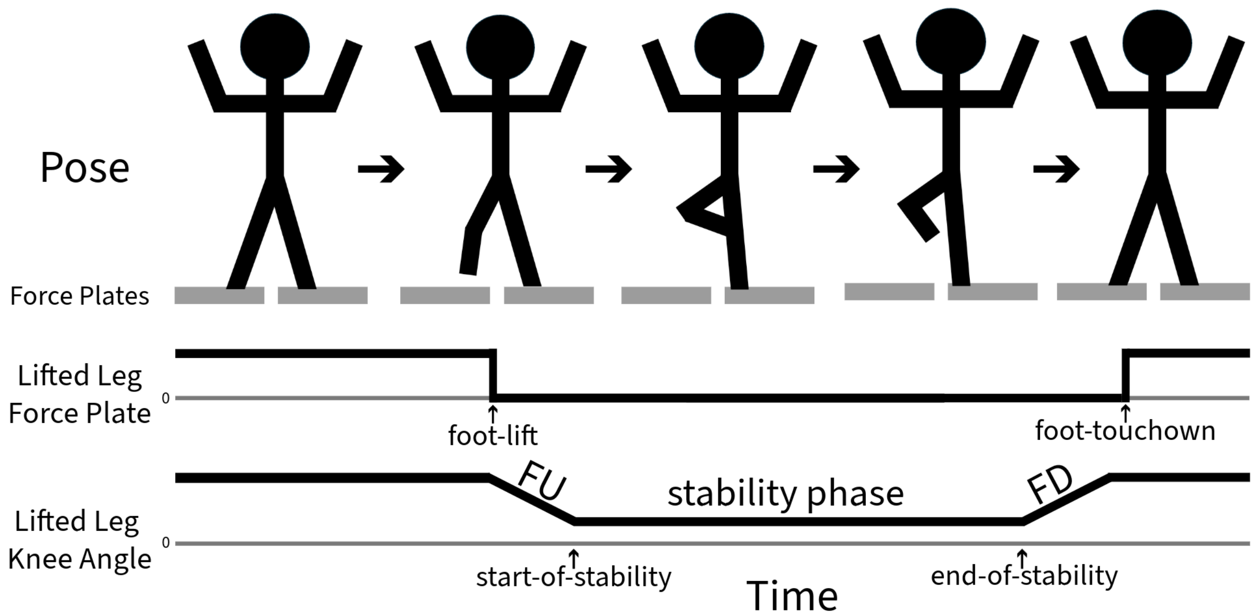

Figure 3.2: Sequential representation of a complete OLST movement task performed by subjects in front of the FMCW radar. Participants transition from a starting pose to a one-legged stance, followed by a return to the initial pose. MOCAP and force plate data are synchronized with radar signatures to identify key temporal events: 'foot-lift' marks the initiation of the one-legged stance, 'start-of-stability' indicates when the subject achieves balance, 'end-of-stability' designates the moment just before the lifted foot descends, and 'foot-touchdown' signals the foot's return to the force plate. Foot-Up (FU) is when the subject is lifting their foot. Food-Down (FD) is when the subject is lowering their foot. The 'stability phase' is when the subject maintains a one-legged stance. The knee angle of the lifted leg is calculated to identify the start and end of the stability phase.

Methodological Frameworks for Identifying Stability Phase in MOCAP Data by Analyzing Lifted Leg Knee Angle

Knee Angle Calculation Using 3d coordinate marker data provided by MOCAP, we calculated the lifted leg knee angle time series to identify when the knee angle minimized and plateaued (end of FU movement) and began straightening (start of FD movement) [Figure 3.2].

Stability Phase Identification We used dynamic thresholding with hysteresis to smooth the knee angle time series, account for knee angle variation across captures, and identify the start-of-stability and end-of-stability times.

MOCAP Time to RDM Frame Conversion After identifying the foot-lifts, foot-touchdowns, starts-of-stability, and ends-of-stability, we converted these MOCAP times into RDM frames. These time/frame pairs define the FU, FD, and stability phases in MOCAP, force plate, and RDM datasets.

3.1.3 Machine Learning Models

In the following subsections, we detail two CNN-LSTM models.

The first model identifies and classifies RDM frames containing FU/FD movements within radar recordings. By accurately identifying when these movements occur, we can bring the powerful OLST beyond the clinic and into the home while maintaining patient privacy. The OLST's clinical utility stems from its prognostic power and simplicity; it can be performed in a family medicine doctor's office without equipment or much training. While clinicians can make qualitative assessments during the test, it is primarily used as a binary test that classifies fall risk at a single threshold.

Due to the limitations of the clinical OLST, we designed the second model to predict postural control during the stability phase. This model accepts RDM data as input and predicts a commonly studied force plate metric, the standing leg's time-normalized COP travel distance [Equation 3.1] [15], [40]. This metric's assessment of postural control is more granular than that of the OLST's, meaning it can identify a significant elevation in fall risk in a person whose binary OLST result does not change. Currently, this metric can only be obtained in a highly equipped lab or specialist clinical settings [34]. Accurate radar-based prediction and tracking of this metric would similarly bring this powerful test beyond the lab and into the home.

Foot-Up Foot-Down Detection Model

Generating Training, Validation, and Testing Datasets To assess the model's generalizability and reduce the risk of overfitting, we divided the participants into training, validation, and testing subsets using a fixed ratio of 67% for training, 13% for validation, and 20% for testing. This division allocated 10 participants to the training subset, 2 to the validation subset, and 3 to the testing subset.

The input data for the FU/FD model were blocks of 100 sequential RDM frames called windows. We generated these windows with a sliding window mechanism to ensure that the full FU/FD movements, the longest of which was 45 consecutive frames, are kept together in at least some windows. For each approximately 700-frame capture within the dataset, we generated fixed-size windows (100-frames) with overlap (90-frames) between consecutive windows. Each frame was labeled as FU, FD, or Neither according to FU/FD event frames determined from the MOCAP and Force Plate data preprocessing, laying the groundwork for supervised learning.

We applied a simple collate function to ensure uniformity and efficiency in data processing, padding shorter sequences to ensure all windows were 100 frames.

FU/FD Model Architecture Our model's hybrid CNN and LSTM architecture classifies each RDM within the 100-frame window. This design exploits spatial features within individual RDMs and temporal patterns across sequences of RDMs. Initially, the model employs a CNN layer to extract spatial features from the RDMs, applying a convolution operation followed by a ReLU activation function and max-pooling to reduce dimensionality while preserving relevant spatial characteristics. The spatially processed data then feeds into an LSTM layer, which captures temporal dependencies and dynamics over time by processing the sequence of feature-extracted RDMs. Finally, a fully connected layer serves as the classifier, taking LSTM's output and mapping it to the probabilities that each frame's label is FU, FD, or Neither.

Test Data Prediction Aggregation After the model generated prediction probabilities for each frame within the sliding test data windows, we had to map these overlapping predictions to the original full-length sequence of RDMs, resulting in one final class prediction for each RDM in the capture. Since the class prediction for each frame is a probability, we aggregated the overlapping predictions by choosing the highest probability prediction for each unique frame, ensuring the retention of the highest confidence prediction per frame. Following this aggregation, we smoothed the final predictions and consolidated blocks of similarly classified sequential frames into FU/FD events to improve the model's clarity.

Postural Control Prediction Model

Data Collection and Preparation Our postural control prediction model's dataset is also based on the processed radar RDM sequences. RDMs capturing each stability phase were extracted.

Dataset Division and Labeling RDM sequences were labeled with the stability phase time-normalized COP travel distance, indicating balance efficacy and postural control [Eq. 3.1]. The time-normalized distance (TN Dist) of the Center of Pressure (COP) is calculated as the sum of the square root of the squared deltas of COP in both x and y coordinates over consecutive time steps from the start (t_{ss}) to the end (t_{es}) of the stability phase. This value is then normalized by the total duration of the stability phase, providing a standardized measure of balance and postural stability.

$$\text{TN Dist} = \frac{\sum_{t=t_{ss}}^{t_{es}} \sqrt{(\Delta COP_x)^2 + (\Delta COP_y)^2}}{t_{es} - t_{ss}} \tag{3.1}$$

After labeling the RDM sequences with a TN Dist, we applied a collate function to ensure uniformity and efficiency in the data processing. By padding shorter sequences, we ensured all sequences within a batch were the same length.

Postural Control Prediction Model Architecture Like the FU/FD model, the Postural Control Prediction model also integrates CNN and LSTM networks to analyze RDM sequences for balance assessment. Initially, the model employs two CNN layers that process the input RDM sequences through filters, enhancing feature depth while maintaining spatial dimensions. This step is followed by max pooling to reduce the spatial dimensions of the feature maps, emphasizing significant features while reducing computational load. The sequential processing of CNN layers, coupled with spatial dimension reduction, prepares the data for temporal analysis.

Subsequently, the processed features are reshaped and fed into LSTM layers designed to capture temporal dependencies within the radar data sequences. The LSTMs analyze the dynamics across time, culminating in a fully connected layer that outputs the predicted standing leg's time-normalized COP travel distance.

Cross-Validation Training, Fine-Tuning, and Testing We used a 15-fold Leave-One-Out (LOO) cross-validation method, creating 15 models withholding one participant's data in each case. This way, each participant's data was used as a unique validation set for fine-tuning and testing the models. This approach tests the models' generalizability on unseen data.

For each model, we split the excluded participant's data 25%/75% for fine-tuning and testing, respectively. We iteratively fine-tuned each model using from 0% to 25% of the fine-tuning data, then evaluated performance on the test data, which constituted the remaining 75% of the participant's data. This process enabled us to determine the optimal amount of participant-specific data required to adapt the model effectively to new individuals.

3.1.4 Results

The FU/FD Detection Model Results

The FU/FD Detection Model combined CNN and LSTM layers to effectively classify RDM sequences [Figure 3.3 and Figure 3.4]. After consolidating sequential similarly labeled base-truth and predicted RMD frames into events, the model demonstrated exceptional accuracy in event detection on the test dataset, correctly identifying 98.4% of FU/FD events. Specifically, the model achieved a sensitivity of 0.99 and a specificity of 0.99 for FU events. For FD events, the sensitivity and specificity were 0.987 and 1.0, respectively [Figure 3.5].

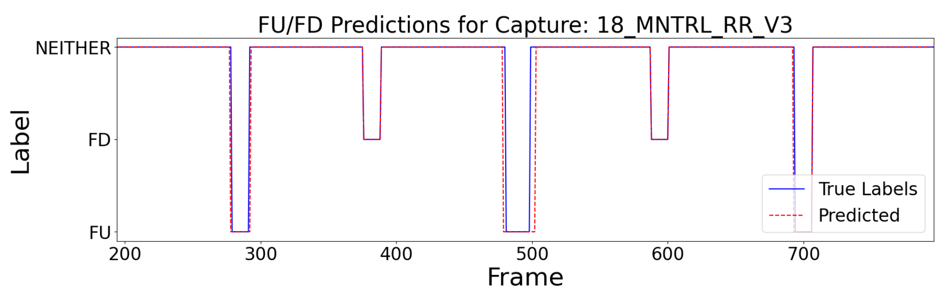

Figure 3.3: Test Participant 18's Left-Side Third-Capture FU/FD events, followed closely with the audio and video cues. All FU and FD events were correctly labeled, with no false positives. The solid line represents the true labels, while the dashed line represents the FU/FD model's prediction. All FU/FD events were accurately located, with no false positives.

An event prediction was considered correct if the True Label fell within 10 frames or approximately 0.36 seconds. This allowance accommodates minor temporal discrepancies between the model's prediction and the labeled event, stemming from the inherent variability in human motion and slight synchronization differences between radar and MOCAP systems.

Postural Control Prediction Model Results

The Postural Control Prediction Model assesses the OLST stability phase by predicting the force-plates time-normalized COP travel distance [Eq. 3.1]. For each model, we used the

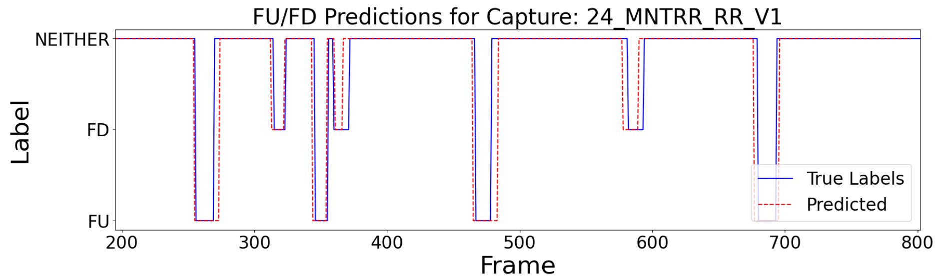

Figure 3.4: Test Participant 24's Right-Side First-Capture FU/FD events included a quick fallout during the first stability phase. The solid line represents the true labels, while the dashed line represents the FU/FD model's prediction. The additional FD and subsequent FU occurred in quick succession; however, all FU/FD events were accurately located, with no false positives.

excluded participant's data 25%/75% for fine-tuning and testing, respectively. The fine-tuning quickly increased the model's predictive accuracy with only a small percentage of the test participant's data. This finding was reflected in the overall R-squared scores that increased from 0.40 to 0.63 with an increase in fine-tuning stability from 2% to 5% [Figure 3.6]. Although not all data points align perfectly with the ideal prediction line (dashed black line), the proximity of the fit line to this ideal underscores the model's capability to accurately evaluate stability.

Our analysis indicates that fine-tuning significantly impacts model performance. As the fine-tuning data increased from 0 to 2 fine-tuning stability phase captures, a stark improvement in R-squared scores was observed [Figure 3.7]. However, the incremental benefit diminished progressively, and the R-squared scores plateaued after 6 fine-tuning stability phase captures, 11%, were used.

3.1.5 Discussion

The FU/FD Detection Model Discussion

The FU/FD Detection Model identified FU/FD events with an accuracy of 98.4%, validating using FMCW radar and machine learning for this home health activity recognition task. The model can efficiently monitor the 10-second OLST by calculating the time between FU and FD events. Before deployment, the system's accuracy could be further improved by incorporating logic. For example, following a high-confidence FU event, the system would only look for FD events.

Using the 10-frame buffer for calculating FU/FD prediction accuracy helped demonstrate the model's usefulness in real-world scenarios without compromising clinical utility.

The high accuracy, sensitivity, and specificity levels achieved in detecting FU/FD events illustrate the model's capacity to identify subtle movement patterns in radar data, which could be applied to other human motion tracking and characterization tasks, such as gait or movement disorders.

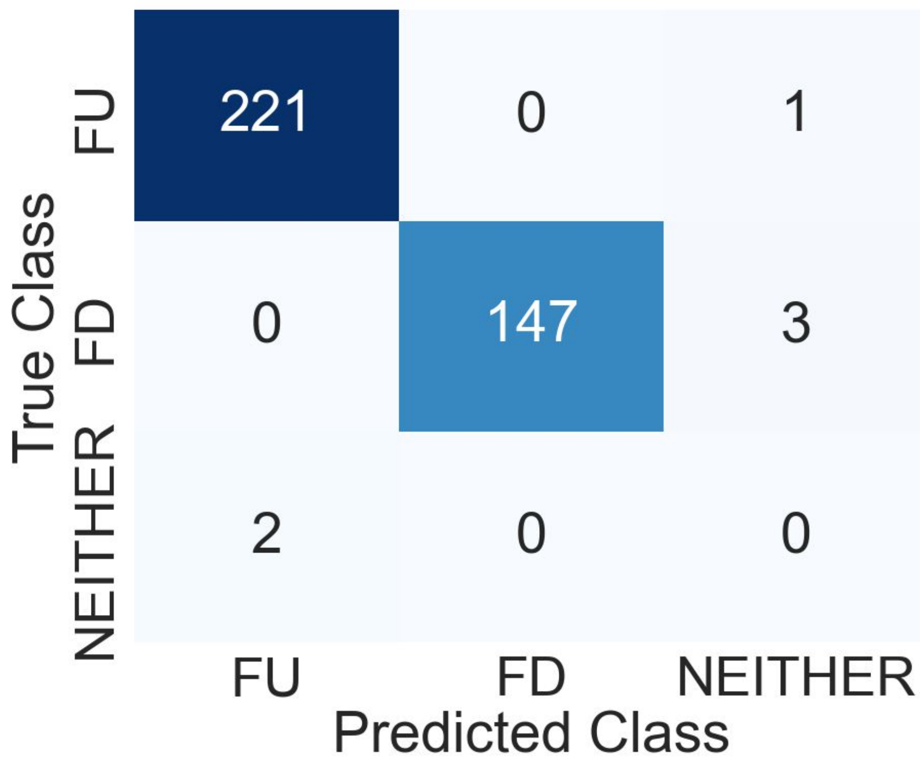

Figure 3.5: Confusion matrix representing the classification performance of the FU/FD model. The matrix displays the number of correctly and incorrectly predicted instances for each class. The True Class denotes the actual category of the movement as labeled in the test data, while the Predicted Class signifies the algorithm's prediction. The matrix diagonal represents accurate predictions, with 221 instances correctly identified as FU, and 147 as FD. Off-diagonal elements indicate misclassifications: 2 instances of NEITHER (no significant movement) were incorrectly predicted as FU, and 3 instances of FD were misclassified as NEITHER. There were no instances where FU was incorrectly predicted as FD, demonstrating a high classification accuracy for these movements by the algorithm.

Postural Control Prediction Model Discussion

The results of our Postural Control Prediction model with 15-fold LOO cross-validation provide compelling evidence that fine-tuning with a minimal subset of participant-specific data can significantly enhance predictive performance. This finding is critical for the practical deployment of personalized systems, where capturing an individual's unique movement patterns can lead to more accurate and reliable predictions. Each model requires only a small amount of fine-tuning data from a new participant to predict the time-normalized COP travel distance with high accuracy on the unseen 75% of the test data set.

The models' ability to identify stability nuances, even among young, healthy participants, suggests broader applicability in distinguishing between individuals with expected larger variations in balance ability and associated fall risk. Automatic notifications to the patient's care team about sudden or gradual decreases in performance could facilitate timely interventions. Additionally, longitudinal outcome data could offer new insights into pre-fall

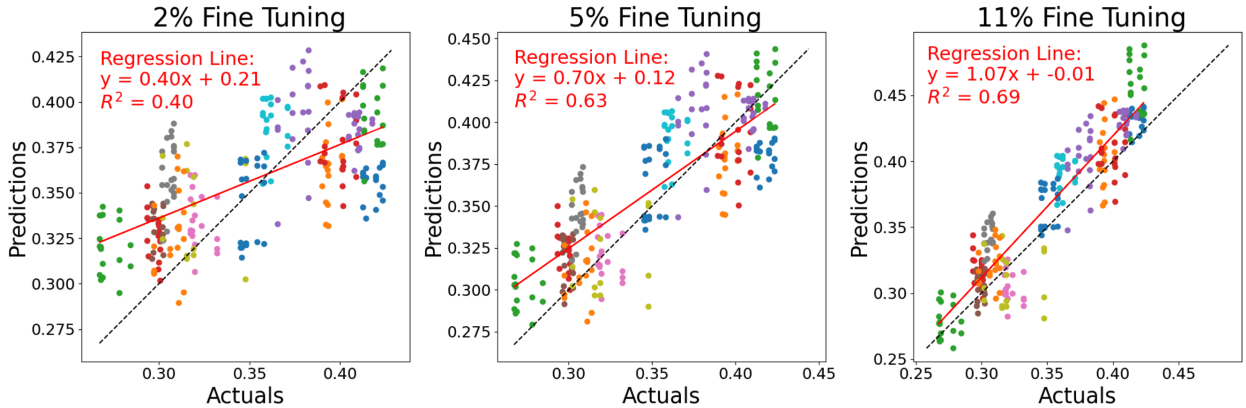

Figure 3.6: Comparison of model predictions versus actual values for the Center of Pressure (COP) time-normalized distance using different amounts of fine-tuning data. Each LOO model was tested on 75% of the excluded participant's data. Each subplot corresponds to results from models fine-tuned with a distinct percentage of the test data: 2%, 5%, and 11%, respectively. The scattered dots represent individual predictions for each test participant, with different colors indicating different participants. The solid red line depicts the regression line for each fine-tuning set, with the corresponding equation and R-squared value annotated. The dashed black line indicates the ideal scenario where predictions match the actual values perfectly. The trend demonstrates improved model accuracy and better generalization with an increase in the percentage of data used for fine-tuning.

movement patterns and lead to enhanced fall risk identification and prevention strategies. The Postural Control Prediction Model's strong performance in predicting the standing leg's time-normalized COP travel distance during the OLST highlights its potential as a valuable balance and fall risk assessment tool.

While the study's results are promising, certain limitations must be acknowledged. The fine-tuning process, although not requiring extensive data, does assume the availability of high-quality and representative movement sequences from future users. In scenarios where such data are challenging to obtain, model performance might be impacted. Additionally, this study was cross-sectional and, therefore, does not have data from a single individual whose movement patterns changed over time.

Data Preprocessing Discussion

We used single-channel data as input to reduce the FMCW radar hardware complexity requirements. Similarly, the decision to selectively focus on known range and velocity parameters further demonstrates a strategic approach tailored to the OLST application, streamlining the radar data processing workflow. This data processing method significantly reduces the computational burden by filtering out irrelevant signal data, increasing memory and processing efficiency in deployed products.

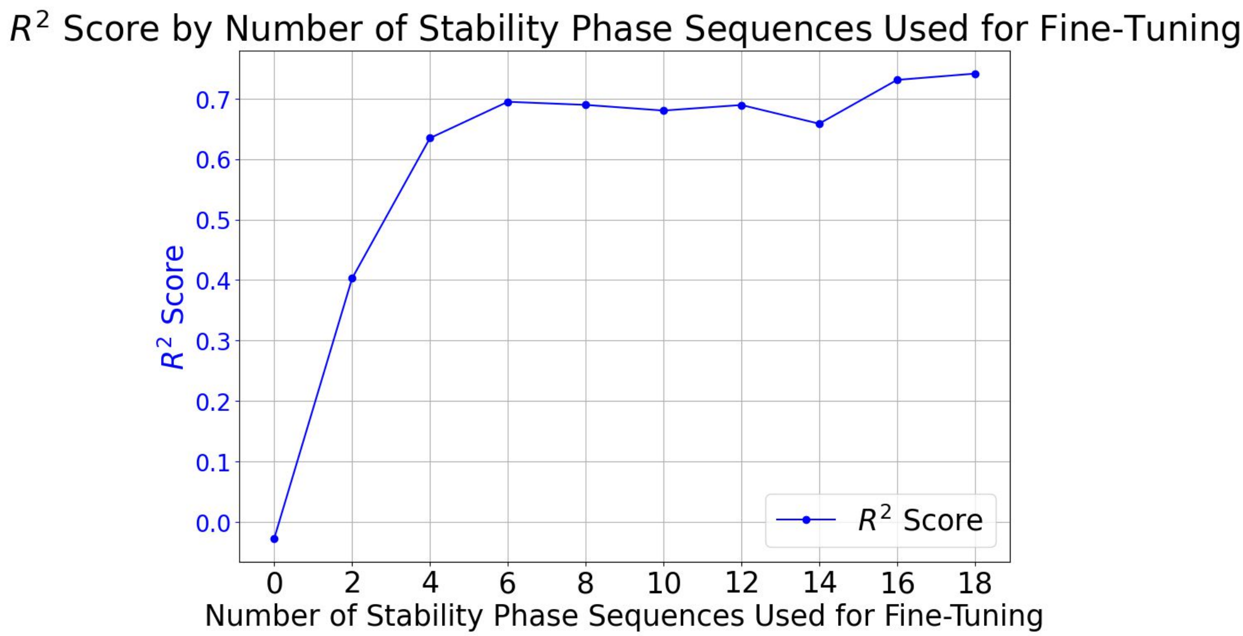

Figure 3.7: Graph illustrating the relationship between the number of stability phase sequences used for fine-tuning and the corresponding R-squared score of the model's predictions. The X-axis displays the count of standing phase RDM sequences, ranging from 0 to 18. This corresponds to 0 to 100% of the fine-tuning dataset. The Y-axis quantifies the R-squared score, indicating the model's prediction accuracy. The blue line and markers highlight the trend of R-squared score improvement as more sequences are used for fine-tuning, plateauing as the number of sequences increases, which suggests diminishing returns on prediction accuracy beyond a certain point of fine-tuning data inclusion.

3.2 Yoga Pose Transition Analysis Using FMCW Radar

3.2.1 Motivation

The increasing popularity of yoga as an in-home workout underscores its numerous physical and mental health benefits, including improved strength, flexibility, balance, and reduced stress [41]. This study investigates the potential of combining MOCAP, FMCW radar, and ML technologies to enhance yoga practice through precise, non-contact monitoring of movements and transitions.

This analysis aims to develop two ML models; the first model will be trained to identify a variety of yoga movements, and the second model will be trained to characterize human movement quality. The application of these technologies reaches beyond yoga with implications for sports science, in-home health monitoring, fall detection, and physical therapy, thereby contributing to the fields of human movement analysis and automated health support systems.

3.2.2 Methods

Motion Capture Data Preprocessing

In this study, MOCAP was used for base truth measurement. We trained preliminary models to identify yoga positions based on the MOCAP data. We analyzed these preliminary models to ensure that the base truth measurement data was able to identify the movements before proceeding to the more abstract radar data. Various ML techniques were applied to ensure the 13 static yoga positions at the beginning and ends of the transitions were identifiable within this base truth measurement.

MOCAP data were processed to obtain 18 joint positions and 11 joint angles for each yoga pose. The raw radar data were synchronized with the MOCAP data to ensure temporal alignment and identify each pose and transition. Noise and artifacts in both MOCAP and radar data were removed using appropriate filtering and trajectory-filling techniques. The resulting MOCAP data were then sliced into individual poses and transitions for further analysis. Five observations per participant per pose were randomly selected for the t-distributed stochastic neighbor embedding (tSNE) plot and parallel coordinates plot (PCP). Fifty observations per participant per pose were randomly selected for a Random Forest (RF) model. The RF dataset was subdivided into training (80%), validation (10%), and testing (10%) datasets.

Radar Data Preprocessing

For movement quality labeling, we chose to investigate the classic yoga movement, Crescent Lunge to Warrior II. Participants' Crescent Lunge to Warrior II transitions [Table 3.1] were evaluated by the author, a long-time yoga enthusiast, and scored based on transition smoothness, from 0 (very poor) to 4 (excellent).

Score	Transition Description
0	Very poor control; jerky, unsteady, or unstable transition without smoothness or grace.
1	Poor control; transition with some unsteadiness or abrupt movements, lacking overall smoothness.
2	Moderate control; mostly smooth transition with occasional flow interruptions.
3	Good control; smooth and well-coordinated transition with minor room for improvement.
4	Excellent control; exceptionally fluid, graceful, and controlled transition.

Table 3.1: Yoga Transition Scoring System.

The overall scoring metrics are in [Table 3.2].

Metric	Value
Mean	2.44
Median	2
Std Dev	1.04

Table 3.2: Crescent Lunge to Warrior II Transition Metrics.

Similar to the At Home Balance Monitoring Analysis, we processed our radar data to suppress noise using a Hanning Window and generated Range-Doppler Maps (RDMs) following the steps in Figure 1.6 [39]. We cropped the four-channel RDMs to remove range data from beyond 6 meters and velocity data faster than ± 20 meters/seconds. We separated each 4-dimensional RDM (channels, frames, height, width) by channels into four 3-dimensional RDMs (frames, height, width) [Figure 3.1]. The transition windows were extracted from the full capture's RDMs based on the transition timing videos that defined the expected pose and transition time intervals.

The MOCAP and radar datasets were tagged based on the transitions and poses they encoded. These datasets were then used to train machine learning models to accurately identify yoga poses and transitions, and evaluate the quality of transitions between poses.

3.2.3 Results

Visualization Techniques

The t-distributed stochastic neighbor embedding (tSNE) and parallel coordinate plots (PCP) were used to visualize the MOCAP data patterns and relationships between poses and joint angles. The t-SNE plot [Figure 3.8] demonstrates a distinct separation among the different yoga poses, suggesting that these poses are in close proximity when represented in a high-dimensional Euclidean space. For example, the Cat (baby blue) and Cow (pink) are in similar poses and close together in the tSNE plot; however, they are also generally able to be delineated even in tSNE's lower-dimensional representation.

Figure 3.8: tSNE Plot of Yoga Poses. Grouping visualization technique which shows a distribution in 2 dimensions of 11 dimensional joint angle observations colored by yoga pose. Even projected onto lower dimensional space, patterns amongst different poses can be identified.

The PCP [Figure 3.9] provided insights into the joint angle relationships and allowed for identifying specific features that contributed to the classification of each pose. For example, the two tree poses, Right Tree (yellow) and Left Tree (orange), are chiral and have distinctly mirrored knee angles.

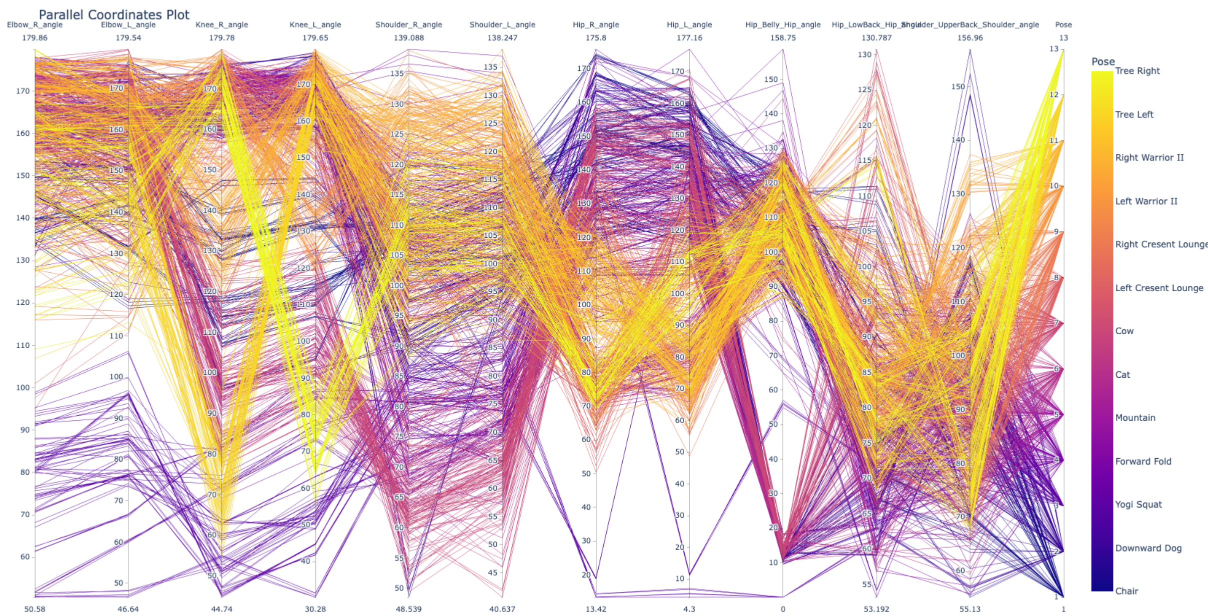

Figure 3.9: Parallel Coordinate Plot of Yoga Poses. Each vertical axis represents a different joint angle in the body. Each color represents a different static pose. Each line across is a single observation at a moment in time. The observation lines intersect the vertical axes at the joint angle value for that observation and are colored by the pose.

Pose Classification

A Random Forest (RF) model was trained to classify the 13 yoga poses based on joint positions and angles extracted from the MOCAP data. The most important features to the RF model in identifying the poses were the knee angles followed by the hip angles [Table 3.3].

Feature	Importance
Knee_R_angle	0.159263
Knee_L_angle	0.144546
Hip_L_angle	0.125505
Hip_R_angle	0.110485
Shoulder_L_angle	0.092234
Shoulder_R_angle	0.089760
Elbow_L_angle	0.067758
Elbow_R_angle	0.067507
Hip_Belly_Hip_angle	0.065061
Hip_LowBack_Hip_angle	0.044947
Shoulder_UpperBack_Shoulder_angle	0.032935

Table 3.3: Feature Importances

The model's performance [table 3.4] was assessed using precision, recall, and F1 score. On the validation dataset, the overall accuracy of the RF model was 98.6%, with a precision of 99%, a recall of 99%, and an F1 score of 99%.

Pose	**Precision**	**Recall**	**F1-score**	**Support**
Cat	1.00	0.99	0.99	82
Chair	1.00	1.00	1.00	83
Cow	0.99	1.00	0.99	69
Downward Dog	0.99	1.00	0.99	83
Forward Fold	0.97	0.97	0.97	71
Left Crescent Lunge	0.94	0.97	0.95	65
Left Warrior II	0.97	1.00	0.99	69
Mountain	0.99	0.94	0.96	78
Right Crescent Lunge	1.00	0.96	0.98	78
Right Warrior II	0.96	1.00	0.98	74
Tree Left	1.00	1.00	1.00	81
Tree Right	1.00	0.98	0.99	65
Yogi Squat	1.00	1.00	1.00	77
Cross-validation mean accuracy		0.9814		
Validation set accuracy		0.9856		
Accuracy (total)		0.99		975
Macro avg		0.99	0.99	975
Weighted avg		0.99	0.99	975

Table 3.4: RF Classification Results for Yoga Poses.

These results indicate that the RF model was able to effectively recognize and differentiate between the 13 types of yoga poses in the dataset.

CNN + LSTM Transition Stability Classification

A combination CNN-LSTM model was trained on time-distributed FMCW radar RDMs of yoga transitions, labeled from 0 (very poor) to 4 (excellent). The CNN model's performance was assessed using Mean Squared Error (MSE), accuracy, and top-two accuracy on validation data. The model achieved an MSE of 0.51, an accuracy of 60.2%, and a top-two accuracy of 85.2% [Table 3.5].

Metric	**Value**
Mean Squared Error (MSE)	0.51
Validation Accuracy	60.2%
Validation Top-2 Accuracy	85.2%
Min Validation Loss	1.07

Table 3.5: Stability Classification CNN, LSTM Combination Model Results.

CNN + LSTM Transition Classification

A similar combination CNN-LSTM model was trained to identify which transition was performed based on tagged time-distributed RDMs of yoga transitions captured in FMCW radar. The performance of the CNN model was assessed using accuracy and top-two accuracy, as many of the transitions are chiral, which may be difficult for an anterior radar to differentiate. The model achieved a validation accuracy of 67.4% and a top two accuracy of 83.5% [Table 3.6].

Metric	**Value**
Validation Accuracy	67.4%
Validation Top-2 Accuracy	83.5%
Loss	0.12

Table 3.6: Transition Identification CNN, LSTM Combination Model Results.

These results suggest that the combination CNN-LSTM model could identify which yoga transition was performed with a high degree of accuracy and, more often than not, distinguish between chiral transitions.

3.2.4 Discussion

This study aimed to develop machine learning models that accurately identify yoga poses and evaluate yoga transitions using MOCAP and FMCW radar technology. Visualization techniques, such as tSNE [Figure 3.8] and PCP [Figure 3.9], provided valuable insights into the patterns and relationships between joint angles, allowing for a better understanding of the features contributing to pose classification. The RF model effectively classified yoga poses based on joint angles. The combination CNN-LSTM model trained on labeled time-distributed 3D RDMs of yoga transitions characterized stability with an accuracy of 60.2%, a top two accuracy of 85.2%, and MSE of 0.51 [Table 3.5]. Top-two accuracy is crucial because it reflects the model's capability to identify the most likely two categories in transition smoothness ratings. Considering the subjectivity in rating yoga transitions, top-two accuracy provides a more nuanced evaluation of the model's performance.

The combination CNN-LSTM model demonstrated its effectiveness in identifying yoga transitions, achieving a testing accuracy of 67.4% and a top-two accuracy of 83.5% [Table 3.6]. This latter metric was particularly valuable given the chiral nature of several transitions that are difficult to differentiate from an anterior radar perspective. Despite these challenges, the high top-two accuracy indicates the model's ability to correctly identify one of the two most likely transitions, particularly between closely related chiral classes. These results demonstrate the model's robustness in recognizing yoga transitions and highlight the value of using additional performance metrics, such as top-two accuracy, in evaluations, especially when the classes are closely related.

This study highlighted the power of machine learning in transforming raw motion data into meaningful insights. These insights could guide future efforts in training and evaluating similar models.

3.3 Optimization of Radar Systems for Human Movement Characterization

3.3.1 Motivation

When designing this human movement-radar study, we encountered the challenge of determining where to place the radar sensors. This task presented an intriguing optimization problem that we believed warranted deeper exploration, particularly because it's an issue likely to be faced by future designers of radar systems for human movement detection within varying enclosed spaces. Optimizing radar systems based on simulated data offers crucial advantages for efficiently integrating these technologies into existing environments, particularly for precise, noninvasive monitoring in healthcare settings.

Simulations would allow for effective designing and retrofitting of spaces by testing various sensor configurations and placements without physical modifications, ensuring optimal setup before actual installation. This approach facilitates a thorough cost-benefit analysis, determining the most effective deployment strategies to balance cost with performance, and potentially reducing the number of sensors needed while maintaining high functionality. Furthermore, simulated data enables the fine-tuning of system specifications, including sensor frequencies and signal processing algorithms, to enhance detection accuracy, particularly for critical applications like fall detection and health monitoring. By exploring different operational scenarios and environmental conditions, simulations ensure radar systems are adaptable and scalable, ready for diverse real-world applications, and minimize the need for expensive physical testing. This method significantly accelerates the development and implementation of radar technologies, promoting innovation in non-invasive monitoring solutions.

Extensive public online libraries are available for motion capture data, yet similar resources for radar data are lacking. By adapting a radar simulator to accept motion capture data, it becomes possible to generate a wealth of simulated radar data, which could then be utilized across a multitude of radar applications.

3.3.2 Methods

Data Preprocessing

MOCAP data of the 18 joint positions were processed to obtain x, y, and z coordinates for both location, and velocity at each time step. The raw radar data were synchronized with the MOCAP data to ensure temporal alignment and identify each pose and transition. Noise and artifacts in MOCAP and radar data were removed using appropriate filtering and trajectory-filling techniques. The resulting MOCAP data were then sliced into individual poses and transitions. For each transition, the position and velocity of each motion capture marker were imported into the modified open-source radar simulator, RadarSimX [42]. Each label was given the worst-case scenario RCS value of 1. This simulator was tuned to the settings of the 1 Transmission/4 Receive (1Tx/4Rx) Multi-Input Multi-Output (MIMO) DemoRad, Analog Devices radar, used during data collection.

Range Doppler Maps (RDMs) were generated from the radar data [Figure 3.10]. Each transition generates 4 RDM sequences due to the MIMO device architecture. This built-in data augmentation adds noise and accounts for the multiple Rx antenna positions. We cropped the four-channel RDMs to remove range data from beyond 15 meters and velocity data faster than ± 5 meters/seconds. The tagged time-series RDMs were used as the inputs to the ML models.

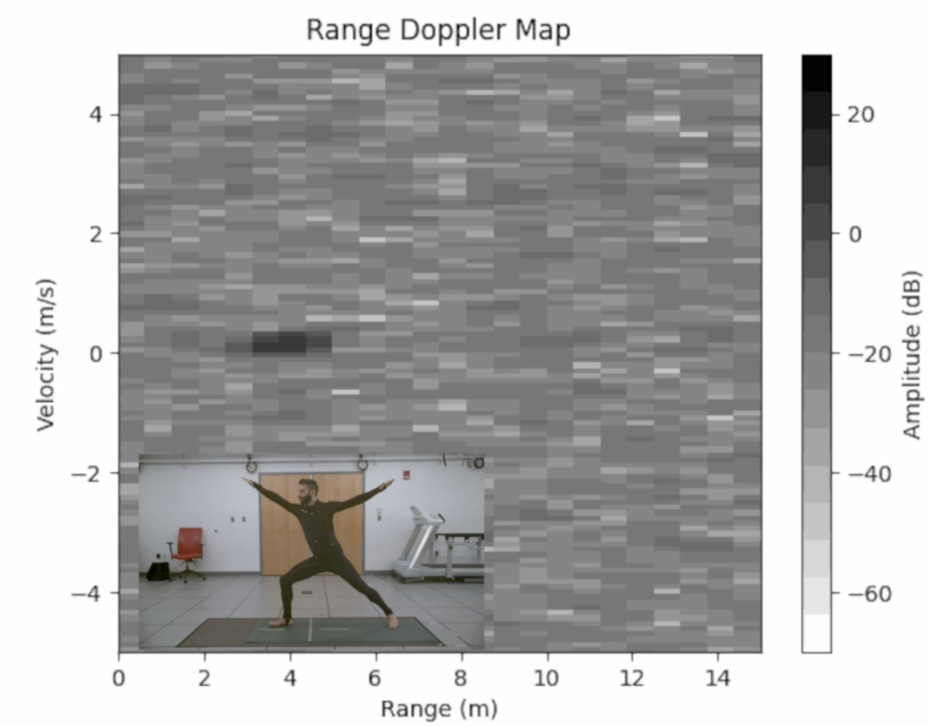

Figure 3.10: Example Range Doppler Map with Time-Synchronized Pose Overlay. The high-amplitude, dark section represents the signal from the participant.

Radar Set-Up Optimization

Optimizing the set-up of the simulated radar system was crucial for enhancing the performance of our ML models. We approached the optimization process by considering three main aspects: the location, number, and orientation of the radar units. To determine the most effective arrangement, we sampled the design space heuristically.

Given a radar placed at a height of $h = 1$ meter and an elevation angle of $\theta = 12.5$ degrees, the horizontal distance d from the radar to the point directly below the target can be calculated using the tangent of the elevation angle:

$$d = \frac{h}{\tan(\theta)}$$

Substituting the given values:

$$d = \frac{1 \text{ m}}{\tan(12.5^{\circ})} \approx 4 \text{ m}$$

This calculation suggests that the radar should be placed approximately 4 meters away from the target for optimal detection. Therefore, initially, we sampled four locations in the design space 4 meters away (front, back, left, and right) from the participant, directed at their approximate center of mass (x = 1, y = 0.5, z = 1).

3.3.3 Motion Characterization Results

Distinguishing between different types of yoga movements

Originally, the study intended to explore a wide design space to determine the optimal sensor placement for accurate movement characterization. However, during preliminary testing, it was discovered that the first two-radar system tested (anterior plus lateral sensors) yielded exceptionally high accuracy. Specifically, the models achieved a 98.2% accuracy and a 100% top-2 accuracy on test data from participants that they had never encountered before. The model also performed well on a single simulated radar system, with an 87.1% accuracy and a 97.7% top-2 accuracy. While relatively close, this result was notably better than the accuracy of the model trained on the real radar data, which had a validation accuracy of 67.4% and a top-two accuracy of 83.5%. The relatively close alignment between the real and simulated accuracies of the single radar system was particularly useful as it validated the effectiveness of the simulated environment for preliminary testing and optimizations. If put into practice, this outcome would reduce the need for extensive real-world data collection while allowing for reliable model development and refinement.

3.3.4 Sensitivity Analysis

Since the challenge of optimizing a two-radar system turned out to be fairly straightforward for this human movement classification task, we decided to perform a comprehensive sensitivity analysis on a single simulated radar system. We chose to evaluate a single-radar system's performance as a function of varying test participant positions in terms of distance and angular displacement from the training setup. This investigation aims to understand the limits within which the radar system can operate without significant loss of accuracy in the task of identifying different human movements. The analysis involved systematically modifying the simulated radar's position and orientation for the test data generation. Such an analysis is essential for establishing the system's robustness in real-world applications, where maintaining an ideal sensor-to-human alignment is not always possible.

Figure 3.11 illustrates the spatial relationship between elevation and azimuth angles that were explored, while Figure 3.12 provides insight into the antenna's gain pattern. The antenna's gain pattern is particularly critical in determining how accurately the system can detect and predict movements at various angles and distances.

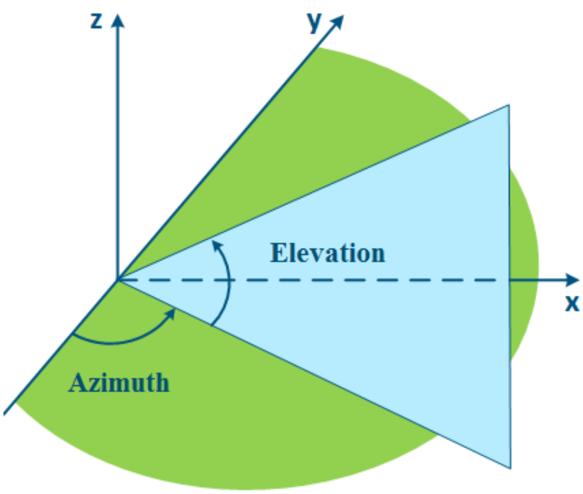

Figure 3.11: Conceptual diagram illustrating radar elevation and azimuth angles.

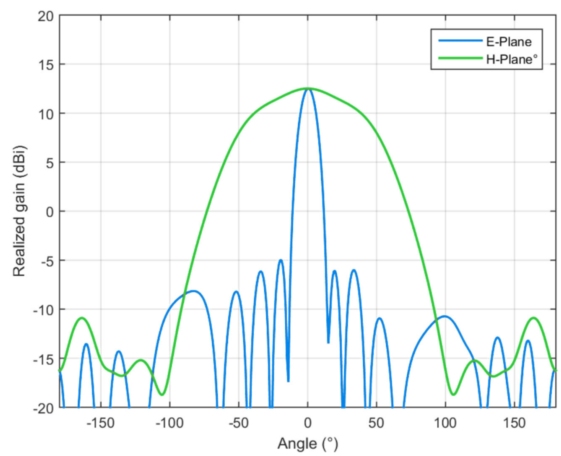

Figure 3.12: Measured gain of a single DemoRad antenna across E-plane and H-plane angles.

Combined Elevation and Azimuth Sensitivity Analysis

The angular sensitivity analysis, as depicted in Figure 3.13, illustrates the radar system's differential response to elevation and azimuth angle changes. Horizontal lines demarcating 'Pure Chance Accuracy' and 'Pure Chance T2A' are included to establish baselines for performance evaluation. The graph indicates a pronounced decline in model accuracy as the elevation angle increases beyond 4.5 degrees. Conversely, the azimuth sensitivity shows a more gradual decrease in accuracy. This suggests that the radar system's performance is more robust to azimuth changes than elevation.

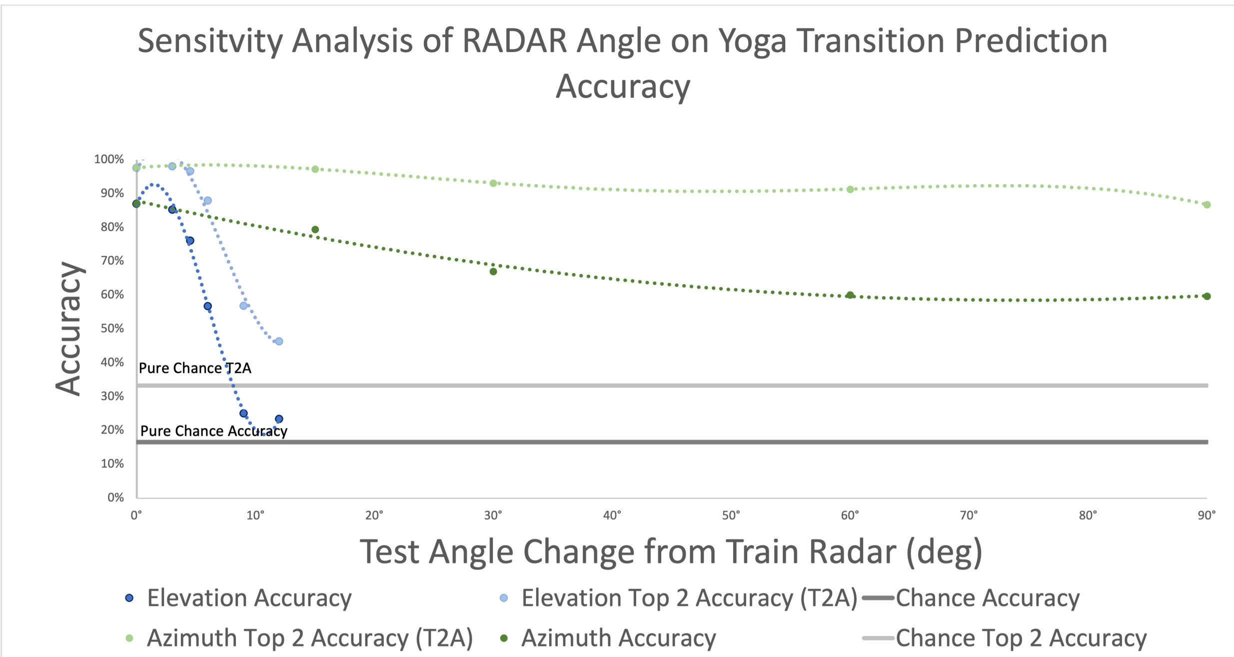

Figure 3.13: Combined Sensitivity Analysis of Elevation and Azimuth Angles on Yoga Transition Prediction Accuracy. The data reflects the model's higher sensitivity to elevation changes as opposed to azimuth changes.

Distance Sensitivity

Figure 3.14 graphically conveys the relationship between the training radar location distance and the corresponding accuracy of yoga transition predictions model trained on the RDM time series. The graph delineates two datasets: the accuracy and the Top 2 Accuracy (T2A). Once the distance exceeds 10 wavelengths, approximately 12.5 cm, prediction accuracy noticeably decreases, indicating a spatial constraint in the effectiveness of the radar's training algorithm. This visual analysis underscores the accuracy reduction beyond the 12.5 cm mark, a critical observation for understanding the model's limitations in spatial sensitivity.

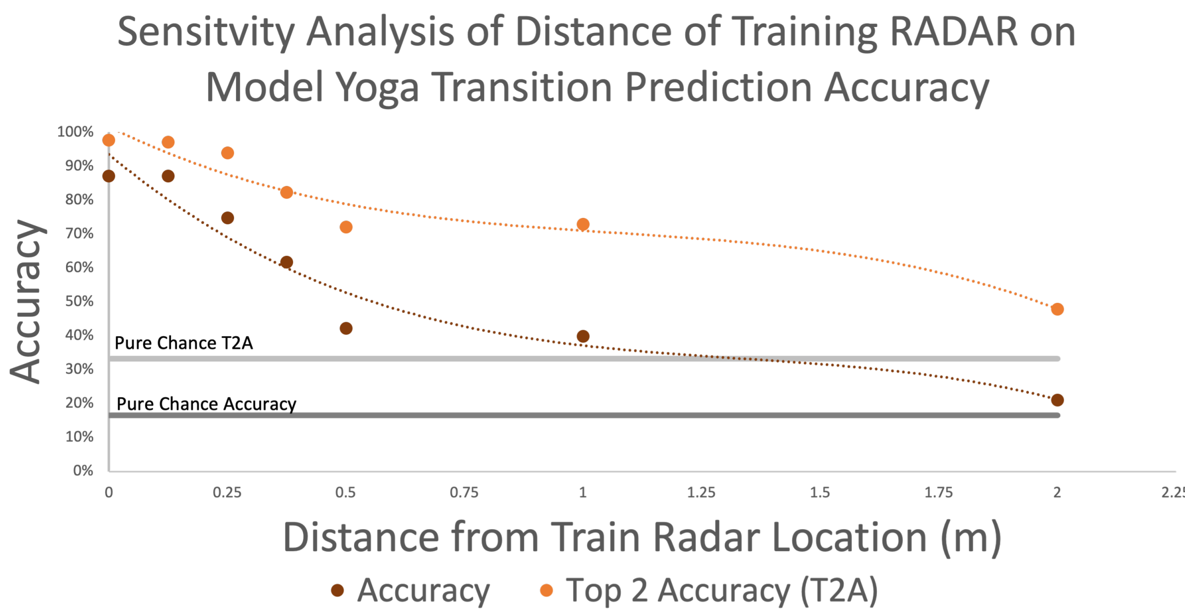

Figure 3.14: Sensitivity Analysis of Distance from Training Radar on Model Yoga Transition Prediction Accuracy.

3.3.5 Discussion

The sensitivity analysis of the simulated single radar system provides insights into the effects of relative positioning, both angular and distance, on movement identification accuracy. The aim was to plot the system accuracy as a function of the difference in simulated position between the training data and the testing data. Such an understanding is vital for practical human motion tracking applications where ideal positioning conditions are seldom met.

The combined analysis of elevation and azimuth sensitivity revealed that the radar system was notably more sensitive to elevation changes than azimuth. As depicted in Figure 3.13, there was a marked decrease in accuracy when the elevation angle exceeded 4.5 degrees. However, changes in azimuth showed a more gradual impact on performance. This finding is in alignment with the directional sensitivity of the radar system, which is more accommodating of azimuthal shifts—a likely consequence of the design and operational mechanics of the antenna gain pattern, as seen in Figure 3.12.

This disparity in angular sensitivity is a key consideration when designing radar systems for monitoring human activity, which predominantly occurs at a constant elevation, such as walking around a room. In such scenarios, optimizing radar systems to be more robust to azimuth changes than elevation changes could enhance detection and tracking accuracy and reduce the number of needed sensors.

Moreover, the analysis of distance sensitivity, illustrated in Figure 3.14, indicated a distinct threshold for accurate predictions. Beyond a critical distance of 12.5 cm, roughly equivalent to 10 wavelengths in this context, model performance declined. In real-world settings, this range would likely be too tight of a window to be effective. The most likely way to increase this range would be to collect training data from a wider range of distances. Additionally, when distance and angular displacements are combined, there may be compounding impacts on system accuracy. This relationship between spatial positioning and accuracy guides the setting of operational parameters for radar systems to ensure maximum efficacy and the need for more robust data collection and algorithms.

Despite these insights, the study has limitations that must be acknowledged. Using a controlled laboratory setting provided a stable environment and simulated data to assess the radar system's performance but did not fully capture the complexity and variability of real-world conditions. Moreover, the participants in this study were within a young age bracket, which does not represent the movement patterns and potential physical limitations of the elderly population—the demographic that could benefit significantly from enhanced radar detection in applications such as fall detection and activity monitoring.

Chapter 4

Conclusion and Future Work

4.1 At Home Radar-Based Fall Risk Monitoring

4.1.1 Conclusion

This analysis has demonstrated the viability of using FMCW radar and machine learning models to assess balance and fall risk in non-clinical settings. We have shown that FMCW radar can accurately detect FU and FD movements and predict force plate patterns. The high levels of accuracy, sensitivity, and specificity our models have achieved demonstrate the technology's potential for daily, noninvasive, low-cost, home-based monitoring of fall risk.

Successfully integrating FMCW radar with machine learning techniques has validated the feasibility of conducting detailed balance assessments outside the clinical environment and opened up new possibilities for enhancing the quality of life for at-risk populations. As we continue to refine non-contact technology and explore its full potential, we remain committed to improving the autonomy, safety, and well-being of individuals at risk of falls, ultimately contributing to the broader goal of helping the elderly age with dignity and improving public health outcomes.

4.1.2 Future Work

The promising results of these models suggest several pathways for future exploration. Future research should prioritize collecting data from populations more representative of those who will benefit from this technology, including older adults and individuals with balance disorders. Long-term studies would offer deeper insights into balance deterioration and the efficacy of treatments. It would be interesting to investigate whether frequently assessing one's fall risk actually reduces falls since frequent testing may increase balance practice and stability awareness.

The length of time a patient is able to stand on one leg has been shown to decrease with age [10]. In addition to the binary 10-second version of this test, a hold-as-long-as-possible test could easily be implemented with the same model, providing another longitudinal result worth tracking.

Integrating radar with other technologies could yield a more nuanced balance evaluation, while real-time monitoring systems may provide instantaneous fall-risk alerts, enabling prompt intervention. Finally, investigating radar's potential in other health metrics, such as gait and posture analysis, could expand its utility in preventive healthcare.

4.2 Yoga Pose Transition Analysis Using FMCW Radar

4.2.1 Conclusion

The yoga pose prediction models have significant potential for application in various fields, including sports science, in-home health, fall detection and risk stratification, and physical therapy.

This analysis has demonstrated the potential of machine learning models and visualization techniques to enhance yoga practice by accurately identifying poses and evaluating transitions. The resulting insights and applications could help pave the way for more meaningful yoga experiences and contribute to the growing field of non-contact human movement monitoring.

4.2.2 Future Work

Future research could explore incorporating additional data modalities, such as electromyography (EMG), ultrasound, video, and force plate data, to provide a more comprehensive understanding of the biomechanics involved in yoga practice. Furthermore, the models can be extended to include a broader range of yoga poses and different levels of expertise among participants. This would enable the development of more sophisticated and robust AI systems that cater to the needs of diverse yoga practitioners, enhancing the overall effectiveness and accessibility of yoga practice in various populations.

By integrating these models with computer vision methods like the Google Pose Project, it may be possible to develop a "yoga guru AI" mobile app that provides personalized, real-time feedback to yoga practitioners, helping them improve their practice. Similarly, an AI in-home physical therapy app could provide immediate feedback on patients' movement and posture, guiding them toward a safer and more effective rehabilitation process.

4.3 Optimization of a Radar System for Human Movement Characterization

4.3.1 Conclusion

Using FMCW radar to evaluate human motion involves a multidisciplinary approach, incorporating physics, biomedical studies, engineering, and computer science. Understanding the underlying physics is crucial for developing effective real-world systems.

Enhancing radar system sensitivity can lead to a robust home-monitoring system capable of accurately predicting and preventing fall incidents, thus promoting the safety and independence of the elderly. Future research should aim to refine these models, expand the dataset to cover a broader range of movements and locations, and assess the scalability of these systems for residential use. Retrofitting unique home environments is also essential for practical application and validation.

This analysis sets the stage for innovative radar technology applications in healthcare, leveraging machine learning to interpret complex data and enhance the well-being of vulnerable populations. The potential to scale up the use of simulations with extensive MOCAP data could transform how we monitor and analyze human movement, significantly impacting preventive healthcare and healthy aging.

4.3.2 Future Work

The outcomes of this analysis lay a solid foundation for future research and development in radar system design for motion detection and classification. Future work could expand in several promising directions, including extending sensitivity analysis to cover a broader range of elevation and azimuth angles and environmental factors like temperature, humidity, and multi-path effects, which could enhance the resilience of radar systems.

Integrating radar data with other sensor modalities such as LIDAR, IMUs, acoustics, and visual camera data could create a more robust detection system, providing complementary information that reduces uncertainties and improves system performance. Additionally, expanding the dataset to include more human movements could facilitate the development of comprehensive activity recognition systems, with significant implications for sports science, elderly care, and rehabilitative medicine.

Practical deployment of radar systems in real-world environments and addressing ethical and privacy concerns are also crucial. Future work should focus on developing guidelines and technologies to protect individual privacy while leveraging radar technology for public safety and welfare. Innovative solutions and societal acceptance of radar technologies will require developing a commercialization strategy for radar-based motion detection systems, ensuring systems meet user needs through feedback, and fostering transdisciplinary research across engineering, data science, healthcare, and ethics. These initiatives are poised to drive significant advancements in radar systems, making them more technically sophisticated, operationally efficient, socially responsible, and aligned with user needs.

Finally, this study's implications are limited by the controlled environment and the young participant age range, which may not fully represent the elderly's movement complexities. Future studies should include a more representative sample of the target population and consider real-life environmental challenges.

Appendix A

Appendix

A.1 Time Synchronization Linear Actuator

A.1.1 Hardware

The calibration device consists of an aluminum covered carbon fiber plate, that is mounted on a linear ball screw guide (FUYU FLS40, 10 mm/rev), and is driven by a NEMA 23 stepper motor. It was driven with a TB6600 Stepper Motor Driver, and controlled by an Arduino UNO [A.1]. The calibration device is used to synchronize the data collected by the motion capture equipment and the FMCW radar. This was achieved by placing a reflective ball on top of the aluminum plate, which is visible by the motion capture equipment.

Pins 2 and 3 were connected to the Dir+ (+5V) and Pul+ (+5V) pins on the driver. Pin 2 is used to set the direction of our rotation, and pin 3 deals with the rotation itself. The rest of the pins on the signal section of the driver were connected to ground. The B-, B+, A- and A+ pins on the high voltage section were connected to the motor in the respective slots, and the GND and VCC pins were connected to our 10W power supply. The driver was set to 6400 steps per revolution.

A.1.2 Software

The code written for the Arduino uses loops to rotate the motor an integer number of times, using a nested loop that makes 6400 steps a number of times specified by the Fcounter (forward rotation counter) and Bcounter (backwards rotation counter) variables. Each individual step is made by first setting the direction of the rotation using the digitalWrite() function, and then producing the step itself by setting pin 3 to low and then high using the same function. Then, a delay of 20 microseconds is applied between steps, which was the lowest delay that was found to work experimentally. The code uses a loop to actuate the plate forwards (Low V on pin 2), and another to actuate it backwards (High V). A 100 ms delay between switching directions was implemented, as it was the shortest delay that worked. This device is placed with the aluminum-covered side facing the radar, and upon pressing the reset button on the Arduino, the plate starts moving after a short delay. It is used at the start and at the end of the simultaneous data collection for the radar and motion capture device.

```
void setup() {
  // SUMMARY: Rotates motor 25 times forward, 25 times backward. Prints position+time data every loop.
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  Serial.begin(9600);

  Serial.print("\n");
  Serial.print("LINEAR CODE START"); // Prints start message through serial monitor. Use CoolTerm to print data to .txt
  Serial.print("\n");
  int Fturn = 25;    // Times motor rotates forward, max safe value = 25
  int Bturn = Fturn; // Times motor rotates backwards, make less than Fturn to avoid damage, but setting equal works fine
  int timeMS = 0;            // Defined later
  float loopTime25 = 5222;   // Time it takes motor to rotate 25 times in ms.
  float loopDist25 = 340;    // Distance covered by plate when motor rotates 25 times in mm.
  float loopSpeed25 = loopDist25 / loopTime25;
  float motorDist = 10000;   // Distance in mm of the end of the actuator with the motor from the radar
  float midTime = 0;         // Defined later
  float posMM = 0;           // Defined later
  float lastPosMM = 0;       // Defined later
  for (int Fcounter = 0; Fcounter < Fturn; Fcounter++) { // Sets up loop to rotate forward Fturn amount of times.
    for (int i = 0; i < 6400; i++) {// Loop that rotates motor once. It takes 6400 steps to rotate motor once.
      digitalWrite(2, LOW); // Sets direction to be forward (away from motor)
      digitalWrite(3, LOW);
      digitalWrite(3, HIGH);
      delayMicroseconds(20); // Lowest delay that works for each step in micros.
    }
    timeMS = millis();                                   // Records current time since code started running in ms.
    posMM = timeMS * loopSpeed25;                        // Uses this time and the speed of the plate to calculate current distance from initial position.
    Serial.print(String(timeMS) + "," + String(motorDist - posMM)); // Prints current time and the distance from the radar.
    Serial.print("\n");
  }
  delay(100); // Shortest delay that worked

  lastPosMM = posMM;  // Records last position of plate.
  midTime = millis(); // Records current time.
  for (int Bcounter = 0; Bcounter < Bturn; Bcounter++) { // Sets up loop to rotate Bturn times in backwards direction.
    for (int j = 0; j < 6400; j++) {   // Rotates the motor once backwards.
      digitalWrite(2, HIGH);           // Sets direction to be backwards (towards motor).
      digitalWrite(3, LOW);
      digitalWrite(3, HIGH);
```

```
      delayMicroseconds(20);
    }
    timeMS = millis();                                // Records time since code started running
    posMM = (timeMS - midTime) * loopSpeed25;    // Obtains distance covered from most forward position.
    Serial.print(String(timeMS) + "," + String(motorDist - lastPosMM + posMM)); // Prints current time and total distance from radar.
    Serial.print("\n");
  }
}
```

Listing A.1: Arduino Code for Motor Control

A.2 Data Processing Classes

A.2.1 FMCW Radar

```python

import h5py
import numpy as np
import os
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, correlate
from scipy.ndimage import convolve
from PIL import Image, ImageDraw

class FMCWRADARDataCapture:
    """
    A class to handle the capture, processing, and saving of FMCW RADAR data from a specified HDF5 file.

    Attributes:
        file_path (str): Path to the HDF5 file containing RADAR data.
    """

    def __init__(self, file_path):
        """
        Initializes the FMCWRADARDataCapture class with the file path.

        Args:
            file_path (str): Path to the HDF5 file to be loaded and processed.
        """
        if not os.path.isfile(file_path):
            raise FileNotFoundError(f"The file '{file_path}' does not exist.")
        self.file_path = file_path
        self.output_path = file_path.replace("_Data", "_Data_NP")  # Default output path

```

```
    def load_and_save(self, output_path=None, format='npy', save_npy=False):
        """
        Loads RADAR data from the HDF5 file, processes it, and optionally saves it to disk.

        Args:
            output_path (str, optional): Path to save the processed data.
            format (str, optional): Format to save the data ('npy' or 'npz').
            save_npy (bool, optional): Whether to save the data to disk.
        """
        if output_path is None:
            output_path = self.output_path
            output_path = os.path.splitext(output_path)[0]  # Ensure correct file extension

        if not os.path.exists(os.path.dirname(output_path)):
            os.makedirs(os.path.dirname(output_path))

        with h5py.File(self.file_path, 'r') as file:
            # Process RADAR data here
            dataCubes = self._process_file(file)

            if save_npy:
                if format == 'npy':
                    np.save(output_path, dataCubes)
                elif format == 'npz':
                    np.savez(output_path, dataCubes)
                else:
                    raise ValueError("Unsupported format. Use 'npy' or 'npz'.")

        return dataCubes

    def _process_file(self, file):
        """
        Internal method to process RADAR data from the HDF5 file.

        Args:
            file (h5py.File): Opened HDF5 file.

        Returns:
            np.ndarray: Processed RADAR data cubes.
        """
        # Configuration and initialization of RADAR parameters
        FreqStrt = file['/BrdCfg/FreqStrt'][()]
        FreqStop = file['/BrdCfg/FreqStop'][()]
        B = (FreqStop - FreqStrt) / 284 * 256  # Effective bandwidth calculation

        If = file['/If'][:]  # Read the If signal
        NrFrms, Nx = If.shape

```

```
        dataCubes = np.zeros((4, NrFrms, 256, 128))  # Initialize data storage

        for frame_idx in range(NrFrms):
            for channel_idx in range(4):
                start_idx = channel_idx * 256 * 128
                end_idx = (channel_idx + 1) * 256 * 128
                reshaped_data = If[frame_idx, start_idx:end_idx].reshape(128, 256)
                dataCubes[channel_idx, frame_idx, :, :] = reshaped_data.transpose()

        return dataCubes


    @staticmethod
    def rawDataToDataCube(rawData, numFrames, numChirpsPerFrame, numSamplesPerChirp, numAntennas):
        # Reshape and rearrange the rawData
        matrixData = rawData.T.reshape(numChirpsPerFrame * numSamplesPerChirp, numFrames * numAntennas)
        dataCubes = np.zeros((numFrames, numChirpsPerFrame, numSamplesPerChirp, numAntennas))

        for frame in range(numFrames):
            for antenna in range(numAntennas):
                chirps = matrixData[:, frame * numAntennas + antenna]
                chirpsMatrix = chirps.reshape(numSamplesPerChirp, numChirpsPerFrame)
                dataCubes[frame, :, :, antenna] = chirpsMatrix.T


        return dataCubes.transpose((3,0,1,2))

        def range_doppler_processing(self, dataCubes):
        """
        Processes data cubes to generate Range-Doppler Maps for each channel and frame.

        Args:
            dataCubes (np.ndarray): Data cubes with RADAR information.

        Returns:
            np.ndarray: Range-Doppler Maps for each channel.
        """
        n_channels, n_frames, n_bins, n_doppler = dataCubes.shape
        rdm_all_channels = []
        range_window = np.hanning(n_bins)
        doppler_window = np.hanning(n_doppler)

        for channel_idx in range(n_channels):
            rdm_list = []
            for frame_idx in range(n_frames):
                current_data = dataCubes[channel_idx, frame_idx, :, :]
```

```
                windowed_data = np.outer(range_window, doppler_window) * current_data
                rdm = np.fft.fft2(windowed_data)
                rdm = np.fft.fftshift(rdm, axes=1)
                rdm = np.abs(rdm)
                rdm_list.append(rdm)
            rdm_all_channels.append(rdm_list)

        return np.array(rdm_all_channels)

    def angle_of_arrival_processing(self, dataCube):
        """
        Processes data cubes to generate Angle of Arrival (AoA) heatmaps for each channel and frame.

        Args:
            dataCube (np.ndarray): The raw data cubes to be processed.

        Returns:
            np.ndarray: Array of processed AoA heatmaps for each channel.
        """
        n_channels, n_frames, n_bins, n_elements = dataCube.shape
        aoa_all_channels = []
        spatial_window = np.hanning(n_elements)

        for channel_idx in range(n_channels):
            aoa_list = []
            for frame_idx in range(n_frames):
                current_data = dataCube[channel_idx, frame_idx, :, :]
                windowed_data = current_data * spatial_window
                aoa_spectrum = np.fft.fft(windowed_data, axis=1)
                aoa_spectrum = np.fft.fftshift(aoa_spectrum, axes=1)
                aoa_spectrum = np.abs(aoa_spectrum)
                aoa_list.append(aoa_spectrum)
            aoa_all_channels.append(aoa_list)

        return np.array(aoa_all_channels)

    def generate_actuator_filter(self, dataCubes):
        """
        Processes each frame in the data cubes to generate actuator-specific filters for RADAR signal analysis.

        Args:
            dataCubes (np.ndarray): The raw data cubes.

        Returns:
            np.ndarray: Range-Doppler Maps filtered based on actuator movement.
        """
        n_channels, n_frames, n_bins, n_doppler = dataCubes.shape
        filtered_rdm = []

        for channel_idx in range(n_channels):
```

```python
            for frame_idx in range(n_frames):
                current_data = dataCubes[channel_idx, frame_idx, :, :]
                rdm = np.abs(np.fft.fft2(current_data))
                rdm = np.fft.fftshift(rdm, axes=1)
                filtered_rdm.append(rdm)

        return np.array(filtered_rdm)

    def plot_match_scores(self, match_scores):
        """
        Visualizes the match scores across different frames to identify significant matching events.

        Args:
            match_scores (np.ndarray): Array of match scores.

        """
        plt.figure(figsize=(10, 6))
        plt.plot(match_scores, marker='o', linestyle='-')
        plt.title('Pattern Match Score Across Frames')
        plt.xlabel('Frame Index')
        plt.ylabel('Match Score')
        plt.grid(True)
        plt.show()

    def create_gif(self, data, gif_filename, fp_data_capture):
        """
        Generates a GIF from radar data frames, annotating foot movement states and saving it to the specified path.

        Args:
            data (np.ndarray): 3D array containing the image data.
            gif_filename (str): Filename for the GIF, without path.
            fp_data_capture (FPDataCapture): Object containing information about foot lift and down frames.

        Returns:
            str: Path to the saved GIF file.
        """
        gif_dir = os.path.join(os.getcwd(), 'data/gifs')
        os.makedirs(gif_dir, exist_ok=True)
        gif_path = os.path.join(gif_dir, gif_filename)

        with imageio.get_writer(gif_path, mode='I', duration=fp_data_capture.seconds_per_frame) as writer:
            for i in range(data.shape[0]):
                frame = data[i, :, :].T  # Transpose for correct orientation
                img = Image.fromarray(np.uint8(plt.cm.viridis(frame) * 255))
                draw = ImageDraw.Draw(img)
                text = 'Foot Down' if i in fp_data_capture.foot_down_frames_after_actuator else 'Foot Up'
                draw.text((10, 10), text, fill='white')
```

```python
                writer.append_data(np.array(img))

        return gif_path

    def process_and_save_channels_tx_separately(self, data, output_folder_path, file_name):
        """
        Processes and saves individual radar channels and transmission periods to separate files.

        Args:
            data (np.ndarray): The raw data from all channels.
            output_folder_path (str): Base directory to save processed files.
            file_name (str): Base file name to use for saved files.

        """
        specific_output_folder_path = os.path.join(output_folder_path, file_name[:2])
        os.makedirs(specific_output_folder_path, exist_ok=True)

        num_channels, num_frames, _, _ = data.shape
        for channel_idx in range(num_channels):
            channel_data = data[channel_idx, :, :, :]
            np.save(os.path.join(specific_output_folder_path, f"{file_name}_channel_{channel_idx+1}.npy"), channel_data)

        print(f"Data for {file_name} processed and saved in separate channel files.")

    def sub_select_RADAR_DATA(self, data):
        """
        Subselects and processes radar data to focus on a specific region of interest.

        Args:
            data (np.ndarray): Full radar data array.

        Returns:
            np.ndarray: Processed and subselected radar data.
        """
        # Assume data dimensions are [channels, frames, height, width]
        processed_data = np.zeros_like(data)

        for channel_idx in range(data.shape[0]):
            for frame_idx in range(data.shape[1]):
                frame_data = data[channel_idx, frame_idx, :, :]
                # Example processing: select central part of the radar image
                center_y, center_x = frame_data.shape[0] // 2, frame_data.shape[1] // 2
                sub_frame = frame_data[center_y-50:center_y+50, center_x-50:center_x+50]
                processed_data[channel_idx, frame_idx, :, :] = sub_frame
```

```python

        return processed_data

    def visualize_data(self, data, frame_index):
        """
        Displays a single frame from radar data for visual inspection.

        Args:
            data (np.ndarray): Radar data array.
            frame_index (int): Frame index to visualize.
        """
        plt.figure()
        plt.imshow(data[frame_index, :, :], cmap='hot', interpolation='nearest')
        plt.colorbar()
        plt.title(f'Radar Data Visualization at Frame {frame_index}')
        plt.xlabel('Range Bins')
        plt.ylabel('Doppler Bins')
        plt.show()
```

Listing A.2: FMCW Radar Capture Python Class

A.2.2 MOCAP

```python
import numpy as np
import pandas as pd
import os
import csv
import re
from datetime import datetime
from scipy.signal import find_peaks, convolve
import matplotlib.pyplot as plt

class MOCAPDataCapture:
    def __init__(self, base_file_path):
        self.base_file_path = base_file_path
        self.sample_frequency = 100
        self.pos_file_path = base_file_path.replace(".tsv", "_pos.tsv")
        self.vel_file_path = base_file_path.replace(".tsv", "_vel.tsv")
        # Pattern to match "/##/" where ## are two digits
        self.participant_pattern = r"/(\d{2})/"
        match = re.search(self.participant_pattern, base_file_path)
        if match:
            self.participant_id = match.group(1)
            print(f"Processing File: {self.base_file_path.split('/')[-1]}")
        else:
            raise ValueError("Participant ID could not be extracted from the base file path.")
        self.position_data = None
        self.velocity_data = None
        self.start_actuator_time = None
        self.end_actuator_time = None
```

```python
        self.load_and_process_data()

    def load_and_process_data(self):
        """
        Loads and processes position and velocity data from TSV files.

        Args:
            pos_file_path (str): The file path to the position TSV file.
            vel_file_path (str): The file path to the velocity TSV file.
        """
        try:
            self.position_data = self.process_tsv(self.pos_file_path)
            self.velocity_data = self.process_tsv(self.vel_file_path)
            # print("Position and velocity data loaded and processed.")
            # print(self.position_data)
            # print(self.velocity_data)
        except Exception as e:
            print(f"An error occurred: {e}")

    def process_tsv(self, file_path, save_to_csv=False):
        print(file_path)
        if not os.path.isfile(file_path):
            raise FileNotFoundError(f"The file '{file_path}' does not exist.")
        with open(file_path, mode='r', newline='') as tsv_file:
            tsv_reader = csv.reader(tsv_file, delimiter='\t')
            # print(tsv_reader)
            first_5_rows_list = []
            remaining_rows_list = []

            try:
                for i, row in enumerate(tsv_reader):
                    if i < 5:
                        first_5_rows_list.append(row)
                    else:
                        if len(row) < 58:
                            row += [''] * (58 - len(row))
                        remaining_rows_list.append(row)
            except Exception as e:
                print(f"An error occurred while processing the file: {e}")
                return

            # Create Header pandas DataFrames from first 5 rows of lists
            df_header = pd.DataFrame(first_5_rows_list).set_index(0)
            try:
                df_header.columns = ["Value"]
            except:
                pass
            # print("Header for data frame")
            # print(df_header)

            # Create blank, correct shape pandas DataFrames from the remainder of the lists
            df = pd.DataFrame(remaining_rows_list)
```

```python

            # Shift row 6 to the left and remove cell 6,1
            df.iloc[2, 0:-1] = df.iloc[0, 1:].values

            #delete empty column
            df = df.iloc[:,:-1]

            # Remove rows 7 and 8 (originally 8 and 9)
            df = df.drop(df.index[0:2])
            df.columns = df.iloc[0]
            df = df.drop(df.index[0])

            ## Change data types of columns
            df = df.apply(pd.to_numeric, downcast='float')

            # Add 'frame', 'time' and participant columns
            df.insert(0, 'frame', range(0, len(df)))
            df.insert(1, 'time', [i * 0.01 for i in range(len(df))])
            df.insert(2,'participant_id', self.participant_id)

            # Reset index
            df.reset_index(drop=True, inplace=True)

            if save_to_csv == True:
                if df.shape[0] != 4000:
                    print(df.shape)
                    raise Exception("DATA Frame is the wrong size!!")
                else:
                    self.output_folder = "/Users/danielcopeland/Library/Mobile Documents/com~apple~CloudDocs/MIT Masters/DRL/LABx/RADARTreePose/data/csvs"
                    output_file_path = os.path.join(
                        self.output_folder, os.path.splitext(os.path.basename(file_path))[0] + ".csv"
                    )
                    print(f"Saved: {os.path.basename(file_path)}")
                    df.to_csv(output_file_path, index=False, header=True)
            return df


    def plot_convolution_result(self, actuator_vel_x):
        """
        Plots the convolution result along with a threshold line to help determine an appropriate threshold.

        Args:
            actuator_vel_x (np.array): The actuator velocity data.
        """
        # Generate the template signal
        template = np.concatenate([np.full(102, 50), np.zeros(10), np.full(102, -50)])

        # Convolve the template with the actuator velocity data
        convolution_result = convolve(actuator_vel_x, template, mode='
```

```
valid')

        # Find local minima in the convolution result
        local_minima_indices, _ = find_peaks(-convolution_result)

        # Define the threshold
        threshold = -4e5

        # Plot the convolution result
        plt.figure(figsize=(12, 6))
        plt.plot(convolution_result, label='Convolution Result')

        # Plot the local minima
        plt.plot(local_minima_indices, convolution_result[local_minima_indices], 'rx', label='Local Minima')

        # Plot the threshold line
        plt.axhline(y=threshold, color='g', linestyle='--', label=f'Threshold ({threshold})')

        plt.xlabel('Time Step')
        plt.ylabel('Convolution Value')
        plt.title('Convolution Result with Local Minima and Threshold')
        plt.legend()
        plt.show()


    def find_actuator_start_end_direction_changes(self):
        """
        Uses convolution to find the start and end times of transitions in the actuator velocity
        from around +50 to -50, ensuring that peaks are not within 2 seconds of each other.
        """
        if self.velocity_data is None:
            print("Velocity data not loaded. Please load data before running this function.")
            return

        # Generate the template signal
        template = np.concatenate([np.full(102, 50), np.zeros(10), np.full(102, -50)])

        # Extract the actuator X velocity data
        actuator_vel_x = self.velocity_data['Actuator_vel_X'].to_numpy()

        # Convolve the template with the actuator velocity data
        convolution_result = convolve(actuator_vel_x, template, mode='valid')

        # Find local minima in the convolution result as potential matches
        local_minima_indices, _ = find_peaks(-convolution_result)

        # Threshold for determining a strong match
```

```python
        threshold = -4e5  # Adjust based on your data's characteristics

        # Filter out matches that don't meet the threshold
        significant_matches = [idx for idx in local_minima_indices if convolution_result[idx] < threshold]

        # Ensure matches are not within 200 indices of each other
        filtered_matches = []
        for match in significant_matches:
            if not filtered_matches:  # If list is empty, add the first match
                filtered_matches.append(match)
            else:
                # Check if current match is more than 200 indices apart from the last added match
                if match - filtered_matches[-1] > 200:
                    filtered_matches.append(match)
                else:
                    # If within 200 indices, keep the one with the more significant peak (lower value in convolution result)
                    if convolution_result[match] < convolution_result[filtered_matches[-1]]:
                        filtered_matches[-1] = match  # Replace the last match with the current one

        if filtered_matches:
            # Set start and end times based on the filtered matches
            self.start_actuator_time = self.velocity_data.iloc[filtered_matches[0]]['time']
            if len(filtered_matches) > 1:
                self.end_actuator_time = self.velocity_data.iloc[filtered_matches[1]]['time']
            print(f"Start actuator time: {self.start_actuator_time}, End actuator time: {self.end_actuator_time}")
        else:
            print("No appropriate transitions found in the Actuator_vel_X data.")


    def get_time_normalized_length(self, start_time, end_time, markers):
        allowed_markers = ['Shoulder', 'Wrist', 'Chest', 'Belly']
        time_normalized_lengths = {}

        # Filter the position data for the given time range
        filtered_data = self.position_data[(self.position_data['time'] >= start_time) & (self.position_data['time'] <= end_time)]

        for marker in markers:
            if marker not in allowed_markers:
                print(f"Marker {marker} is not allowed.")
                continue

            sides = ['R', 'L'] if marker in ['Shoulder', 'Wrist'] else ['']
```

```
            for side in sides:
                marker_name = f"{marker}_{side}" if side else marker
                pos_columns = [f"{marker_name}_pos_X", f"{marker_name}_pos_Y", f"{marker_name}_pos_Z"]

                # Check if the necessary columns exist in the data
                if not all(col in filtered_data.columns for col in pos_columns):
                    print(f"Data for {marker_name} is incomplete or missing.")
                    continue

                # Calculate the distance traveled by the marker
                distances = np.sqrt(np.sum(np.diff(filtered_data[pos_columns].values, axis=0)**2, axis=1))
                total_distance = np.sum(distances)

                # Calculate the time normalization factor
                time_normalization_factor = (end_time - start_time)

                # Calculate the average speed (distance/time)
                average_speed = total_distance / time_normalization_factor if time_normalization_factor > 0 else 0

                # Store the results
                if side:  # For R or L markers
                    key = f"{marker}_{side}"
                    time_normalized_lengths[key] = average_speed
                else:  # For markers without side specification
                    time_normalized_lengths[marker] = average_speed

        return time_normalized_lengths
```

Listing A.3: MOCAP Capture Python Class Template

A.2.3 Force Plate

```

import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull

class FPDataCapture:
    """
    A class to handle the capture, analysis, and visualization of force plate data.

    Attributes:
        base_file_path (str): Base path to the TSV file containing the force plate data.
    """
```

```python

    def __init__(self, base_file_path, is_foot_always_up=False):
        """
        Initializes the FPDataCapture with paths to data and configurations.

        Args:
            base_file_path (str): Path to the base TSV file.
            is_foot_always_up (bool): Indicates if foot is always up, adjusting data processing.
        """
        self.base_file_path = base_file_path
        self.data_f_1 = self.import_data(base_file_path.replace(".tsv", "_f_1.tsv"))
        self.data_f_2 = self.import_data(base_file_path.replace(".tsv", "_f_2.tsv"))
        self.data = self.data_f_2 if "MNTRR" in base_file_path else self.data_f_1

    def import_data(self, file_path):
        """
        Imports data from a specified TSV file.

        Args:
            file_path (str): Path to the TSV file to import.

        Returns:
            DataFrame: A pandas DataFrame with the imported data.
        """
        num_metadata_lines = 26  # Number of initial lines with metadata.
        data = pd.read_csv(file_path, delimiter='\t', header=num_metadata_lines)
        data = data.apply(pd.to_numeric, errors='coerce')
        data.rename(columns=lambda x: x.strip().lower(), inplace=True)
        return data.reset_index(drop=True)

    def identify_foot_lift(self):
        """
        Identifies the times of foot lift events based on changes in Center of Pressure (COP).

        Returns:
            tuple: Two lists containing the times for foot lift and foot down events.
        """
        data = self.data_f_1 if "MNTRL" in self.base_file_path else self.data_f_2
        foot_lift_events = data[(data['cop_x'].shift(1) != 0) & (data['cop_x'] == 0)]
        foot_down_events = data[(data['cop_x'].shift(1) == 0) & (data['cop_x'] != 0)]

        filtered_lift_times = [time for time in foot_lift_events['time'] if time > 8]
```

```python
        filtered_down_times = [time for time in foot_down_events['time']]

        self.foot_lift_times = filtered_lift_times
        self.foot_down_times = filtered_down_times

        return self.foot_lift_times, self.foot_down_times

    def convert_time_to_frames(self, times):
        """
        Converts a list of times into corresponding frame numbers based on data frequency.

        Args:
            times (list): List of times to convert.

        Returns:
            list: List of corresponding frame numbers.
        """
        frames = [int(time * self.sample_frequency) for time in times]
        return frames

    def plot_cop(self):
        """
        Plots the trajectory of the Center of Pressure (COP) over time.
        """
        plt.figure(figsize=(10, 5))
        plt.plot(self.data['time'], self.data['cop_x'], label='COP X')
        plt.plot(self.data['time'], self.data['cop_y'], label='COP Y')
        plt.xlabel('Time (s)')
        plt.ylabel('COP Position')
        plt.title('Center of Pressure Trajectory')
        plt.legend()
        plt.show()

    def calculate_average_velocity(self):
        """
        Calculates the average velocity of the COP movements.

        Returns:
            float: The average velocity.
        """
        velocities = np.sqrt((np.diff(self.data['cop_x']) ** 2) + (np.diff(self.data['cop_y']) ** 2))
        average_velocity = np.mean(velocities)
        return average_velocity

    def generate_cop_trace_gif(self, gif_filename):
        """
        Generates a GIF animation showing the trace of COP movements.

        Args:
            gif_filename (str): Filename for the output GIF.
        """
        images = []
```

```python
        plt.figure(figsize=(6, 6))
        for i in range(0, len(self.data), 10):  # Sampling every 10 frames for simplicity
            plt.plot(self.data['cop_x'][:i], self.data['cop_y'][:i], color='blue')
            plt.xlim([self.data['cop_x'].min(), self.data['cop_x'].max()])
            plt.ylim([self.data['cop_y'].min(), self.data['cop_y'].max()])
            filename = f'temp_frame_{i}.png'
            plt.savefig(filename)
            images.append(imageio.imread(filename))
            os.remove(filename)

        imageio.mimsave(gif_filename, images, duration=0.1)
        plt.close()

        def plot_force_vectors(self):
        """
        Plots the force vectors (X, Y, Z) over time to visualize changes in force plate measurements.
        """
        plt.figure(figsize=(14, 7))
        plt.plot(self.data['time'], self.data['force_x'], label='Force X')
        plt.plot(self.data['time'], self.data['force_y'], label='Force Y')
        plt.plot(self.data['time'], self.data['force_z'], label='Force Z')
        plt.title('Force Vectors Over Time')
        plt.xlabel('Time (s)')
        plt.ylabel('Force (N)')
        plt.legend()
        plt.show()

    def calculate_convex_hull_area(self):
        """
        Calculates the area enclosed by the convex hull of the Center of Pressure (COP) points.

        Returns:
            float: The area of the convex hull.
        """
        cop_points = self.data[['cop_x', 'cop_y']].dropna().values
        if len(cop_points) < 3:
            return 0
        hull = ConvexHull(cop_points)
        return hull.volume  # In 2D, 'volume' is the area.

    def calculate_average_velocity(self):
        """
        Calculates the average velocity of the Center of Pressure (COP) based on its movement over time.

        Returns:
            float: The average velocity of COP.
        """
        dx = np.diff(self.data['cop_x'])
        dy = np.diff(self.data['cop_y'])
```

```
        dt = np.diff(self.data['time'])
        velocities = np.sqrt(dx**2 + dy**2) / dt
        return np.nanmean(velocities)

    def calculate_maximum_distance_from_centroid(self):
        """
        Calculates the maximum distance from the centroid of all Center of Pressure (COP) points.

        Returns:
            float: The maximum distance from the centroid.
        """
        cop_points = self.data[['cop_x', 'cop_y']].dropna().values
        centroid = np.mean(cop_points, axis=0)
        distances = np.sqrt(((cop_points - centroid)**2).sum(axis=1))
        return np.max(distances)

    def generate_cop_trace_gif(self, gif_filename):
        """
        Generates a GIF showing the trajectory of the Center of Pressure (COP) over time.

        Args:
            gif_filename (str): The filename where the GIF should be saved.
        """
        cop_x = self.data['cop_x'].values
        cop_y = self.data['cop_y'].values
        images = []

        plt.figure(figsize=(8, 8))
        for i in range(0, len(cop_x), 10):  # Adjust step for smoother animation
            plt.plot(cop_x[:i], cop_y[:i], color='blue')
            plt.xlim([np.min(cop_x), np.max(cop_x)])
            plt.ylim([np.min(cop_y), np.max(cop_y)])
            filename = f'temp_frame_{i}.png'
            plt.savefig(filename)
            images.append(imageio.imread(filename))
            os.remove(filename)

        imageio.mimsave(gif_filename, images, duration=0.1)
        plt.close()
        print(f"GIF saved to {gif_filename}")

    def plot_cop_velocity(self):
        """
        Plots the velocity of the Center of Pressure (COP) over time.

        Returns:
            matplotlib.figure.Figure: A plot of COP velocity over time.
        """
        dx = np.diff(self.data['cop_x'])
        dy = np.diff(self.data['cop_y'])
```

```
        dt = np.diff(self.data['time'])
        velocities = np.sqrt(dx**2 + dy**2) / dt

        plt.figure(figsize=(10, 5))
        plt.plot(self.data['time'][:-1], velocities, label='COP Velocity')
        plt.title('Center of Pressure (COP) Velocity Over Time')
        plt.xlabel('Time (s)')
        plt.ylabel('Velocity (mm/s)')
        plt.legend()
        plt.show()

    def save_data_summary(self, filename):
        """
        Saves a summary of the force plate data to a CSV file.

        Args:
            filename (str): The filename to save the data summary.
        """
        summary = {
            'Average Velocity': self.calculate_average_velocity(),
            'Max Distance from Centroid': self.calculate_maximum_distance_from_centroid(),
            'Convex Hull Area': self.calculate_convex_hull_area()
        }
        summary_df = pd.DataFrame([summary])
        summary_df.to_csv(filename, index=False)
        print(f"Summary data saved to {filename}")
```

Listing A.4: Force Plate Capture Python Class Template

A.3 Dataset Classes

A.3.1 Full Capture

```

import pandas as pd
import os
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms.functional import resize
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence
from torch import optim
from scipy.ndimage import uniform_filter1d
from scipy.stats import mode
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

class RdmFullCapture(Dataset):
    def __init__(self, root_dir, event_csv, included_folders, window_size=100):
```

```
        self.data = []
        self.full_capture_labels = []
        self.labels = []  # This will store labels for each capture, list of lists
        self.all_metadata = []  # Store metadata for each capture, list of dictionaries
        self.window_size = window_size
        self.current_index = 0

        # Load event labels and actuator frames
        self.event_labels_df = pd.read_csv(event_csv)

        # Iterate only over included folders
        for folder_name in included_folders:
            folder_path = os.path.join(root_dir, folder_name)
            for file in sorted(os.listdir(folder_path)):
                if file.endswith('.npy'):
                    filepath = os.path.join(folder_path, file)
                    radar_capture = "_".join(file.split('_')[:-1])  # Extract radar capture name

                    channel_number = filepath.split(".")[-2].split("channel")[-1]

                    # Ensure radar_capture matches one of the entries in the actuator CSV
                    if self.event_labels_df['RADAR_capture'].str.contains(radar_capture).any():
                        rdm_data = np.load(filepath)
                        rdm_data = torch.from_numpy(rdm_data).float()  # Convert numpy array to PyTorch tensor of type float

                        actuator_info = self.event_labels_df[self.event_labels_df['RADAR_capture'] == radar_capture].iloc[0]
                        actuator_start_frame, actuator_end_frame = actuator_info['RADAR_Start_Frame'], actuator_info['RADAR_End_Frame']
                        MOCAP_Start_Time = actuator_info['RADAR_Start_Frame']
                        MOCAP_End_Time = actuator_info['MOCAP_End_Time']
                        seconds_per_frame = actuator_info['Seconds_per_Frame']

                        # Create windows, label them, and add metadata
                        labels, goup_ranges, down_ranges = self.label_frames(radar_capture)
                        metadata = {
                            'channel_number': channel_number,
                            'frame_range': (actuator_start_frame, actuator_end_frame),
                            'MOCAP_time_range' : (MOCAP_Start_Time, MOCAP_End_Time),
                            'seconds_per_frame': seconds_per_frame,
                            'RADAR_capture': radar_capture,
                            'GOUP_ranges': goup_ranges,
```

```
                            'DOWN_ranges': down_ranges,
                            'window_start_frame': 0,
                            'window_end_frame': 0
                            }

                        self.all_metadata.append(metadata)
                        self.labels.append(labels)
                        self.data.append(rdm_data)

    def label_frames(self, radar_capture):
        num_frames = 1000
        labels = np.full(num_frames, 2)
        capture_events = self.event_labels_df[self.event_labels_df['RADAR_capture'] == radar_capture]

        goup_ranges = []
        down_ranges = []

        for _, event in capture_events.iterrows():
            if not pd.isna(event['frame_foot_up']) and not pd.isna(event['frame_stable']):
                start = int(event['frame_foot_up'])+1
                end = int(event['frame_stable'])+1
                labels[start:end] = 0  # GOUP
                goup_ranges.append((start, end))
            if not pd.isna(event['frame_break']) and not pd.isna(event['frame_end']):
                start = int(event['frame_break'])+1
                end = int(event['frame_end'])+1
                labels[start:end] = 1  # DOWN
                down_ranges.append((start, end))

        self.full_capture_labels.append(labels)

        return labels, goup_ranges, down_ranges

    def create_windows_for_capture(self, index, overlap):
        """
        Create windows for a specific capture given by index.

        Parameters:
        - index: Index of the capture to process.
        - window_size: The size of each window.
        - overlap: The overlap between consecutive windows.

        Returns:
        - A tuple containing windows, labels for each window, lengths of each window, and metadata.
        """
        self.overlap = overlap
        self.current_index = index

        if index >= len(self.data):
                raise ValueError("Index out of range.")
```

```python

        capture_data, capture_labels, _, capture_metadata = self[index]
        actuator_start_frame = capture_metadata['frame_range'][0]
        actuator_end_frame = capture_metadata['frame_range'][1]

        windows_ranges = []
        capture_windows_data = []
        windows_labels_data = []  # Collect labels data
        windows_lengths_tensor = []

        num_windows = 1 + (actuator_end_frame - actuator_start_frame - self.window_size) // (self.window_size - overlap)

        print(f"Creating windows {num_windows} windows for Radar Capture: {capture_metadata['RADAR_capture']}, channel: {capture_metadata['channel_number']}")

        for w in range(num_windows):
            start = w * (self.window_size - overlap) + actuator_start_frame  # Adjust for correct sliding window
            end = start + self.window_size
            window_range_dict = {'window_start_frame': start, 'window_end_frame': min(end, actuator_end_frame)}

            if end > actuator_end_frame:
                padding_length = end - actuator_end_frame
                window_data = torch.cat((capture_data[start:actuator_end_frame], torch.zeros(padding_length, *capture_data.shape[1:])), dim=0)
                window_labels = np.pad(capture_labels[start:actuator_end_frame], (0, padding_length), 'constant', constant_values=-1)
            else:
                window_data = torch.tensor(capture_data)[start:end]
                window_labels = capture_labels[start:end]

            capture_windows_data.append(window_data.unsqueeze(0))
            windows_labels_data.append(torch.tensor(window_labels).unsqueeze(0))  # Convert labels to tensor here
            windows_lengths_tensor.append(min(end, actuator_end_frame) - start)
            windows_ranges.append(window_range_dict)

        # Concatenate all windows and labels data after loop
        capture_windows_tensor = torch.cat(capture_windows_data, dim=0)
        windows_labels_tensor = torch.cat(windows_labels_data, dim=0)

        return capture_windows_tensor, windows_labels_tensor, torch.tensor(windows_lengths_tensor, dtype=torch.long), capture_metadata, windows_ranges

    def predict_on_windows(self, model, windows_tensor, lengths):
            model.eval()
            predictions = []
```

```python

            # Ensure lengths is a tensor
            lengths_tensor = torch.tensor(lengths, dtype=torch.long)
            windows_tensor = torch.tensor(windows_tensor, dtype=torch.float)

            with torch.no_grad():
                outputs = model(windows_tensor, lengths_tensor)

                # Correctly flatten output for subsequent operations
                outputs_flat = outputs.view(-1, 3)  # 3 classes

                # Apply softmax to get probabilities
                predictions = torch.softmax(outputs_flat, dim=1).numpy()

            return predictions

    def aggregate_predictions_sliding_windows(self, predictions, windows_ranges, smoothing_window_size=5):
        full_length = max(w_range['window_end_frame'] for w_range in windows_ranges) + 1
        num_classes = predictions.shape[1]

        # Initialize an array for the aggregated maximum likelihoods
        aggregated_predictions = np.zeros((full_length, num_classes))
        coverage_count = np.zeros(full_length)  # Track how many times each frame is covered by windows

        current_pred_idx = 0  # Track the current index within the flat predictions array

        for window_range in windows_ranges:
            start_frame = window_range['window_start_frame']
            end_frame = min(window_range['window_end_frame'], full_length)

            for frame_idx in range(start_frame, end_frame):
                # Extract the prediction for the current frame
                frame_prediction = predictions[current_pred_idx]
                current_pred_idx += 1  # Move to the next prediction

                # Aggregate by taking the maximum likelihood across overlapping predictions
                aggregated_predictions[frame_idx] = np.maximum(aggregated_predictions[frame_idx], frame_prediction)
                coverage_count[frame_idx] += 1

        # Handle frames not covered by any window (if any) to avoid division by zero
        coverage_count[coverage_count == 0] = 1

        # Normalize aggregated predictions by the number of windows covering each frame
        aggregated_predictions /= coverage_count[:, None]

```

```python
        # Let's say 'predictions' is your numpy array with shape (806, 3)
        smoothed_predictions = self.smooth_probabilities(aggregated_predictions)

        # Determine class predictions by selecting the class with the highest likelihood for each frame
        class_predictions = np.argmax(smoothed_predictions, axis=1)

        class_predictions[-1] = 2

        return class_predictions

    def smooth_probabilities(self, probabilities, window_size=7):
        # Check if probabilities array is 2D and has the correct shape
        if probabilities.ndim != 2 or probabilities.shape[1] != 3:
            raise ValueError("The probabilities array should be 2D with shape (n, 3).")

        # Apply a uniform filter to smooth each class's probability
        smoothed = np.apply_along_axis(lambda m: uniform_filter1d(m, size=window_size), axis=0, arr=probabilities)
        return smoothed

    def plot_predictions_with_time(self, index, smoothed_predictions, capture_name):
        """
        Plot smoothed predictions against the true labels and show time on the secondary x-axis.
        Adjusted to label the y-axis by classes 0 being FU, 1 being FD, and 2 being NEITHER.
        Labels and tick labels are made 2x larger.
        """
        labels = self.labels[index]
        metadata = self.all_metadata[index]
        correction_offset = 0.3

        fig, ax1 = plt.subplots(figsize=(20, 5))

        ax1.plot(labels, label='True Labels', color='blue')
        ax1.plot(smoothed_predictions, label='Predicted', color='red', linestyle='--')
        ax1.set_xlim([metadata['frame_range'][0], metadata['frame_range'][1]])
        ax1.set_xlabel('Frame', fontsize=34)  # 2x larger font size for X axis label
        ax1.set_ylabel('Label', fontsize=34)  # 2x larger font size for Y axis label
        ax1.set_yticks([0, 1, 2])
        ax1.set_yticklabels(['FU', 'FD', 'NEITHER'], fontsize=30)  # 2x larger font size for Y tick labels
        ax1.legend(loc='lower right', fontsize=24)  # Adjust legend font size if needed

        # Increase tick label size
```

```
        ax1.tick_params(axis='x', labelsize=14)  # Adjust X tick label size if needed
        ax1.tick_params(axis='y', labelsize=24)  # Adjust Y tick label size if needed

        frames = np.arange(metadata['frame_range'][0], metadata['frame_range'][1] + 1)
        times = (metadata['MOCAP_time_range'][0] + frames * metadata['seconds_per_frame']) - metadata['frame_range'][0] + correction_offset

        # Dynamically determine tick frequency to avoid zero step size
        tick_frequency = max(1, round(1 / metadata['seconds_per_frame']))
        tick_indices = np.arange(len(frames))[::tick_frequency]
        tick_frames = frames[tick_indices]
        tick_times = times[tick_indices]

        ax2 = ax1.twiny()
        ax2.set_xlim(ax1.get_xlim())
        ax2.set_xticks(tick_frames)
        ax2.set_xticklabels(["{:.2f}s".format(time) for time in tick_times], rotation=45, fontsize=14)  # Adjust secondary X tick label size if needed
        ax2.set_xlabel('Time (s)', fontsize=20)  # 2x larger font size for secondary X axis label

        plt.title(f'Predictions vs. True Labels for {capture_name}', fontsize=24)  # 2x larger font size for the title
        plt.show()

    def plot_predictions_without_time(self, index, smoothed_predictions, capture_name):
        """
        Plot smoothed predictions against the true labels and show time on the secondary x-axis.
        Adjusted to label the y-axis by classes 0 being FU, 1 being FD, and 2 being NEITHER.
        Labels and tick labels are made 2x larger.
        """
        labels = self.labels[index]
        metadata = self.all_metadata[index]
        correction_offset = 0.3

        fig, ax1 = plt.subplots(figsize=(20, 5))

        ax1.plot(labels, label='True Labels', color='blue')
        ax1.plot(smoothed_predictions, label='Predicted', color='red', linestyle='--')
        ax1.set_xlim([metadata['frame_range'][0], metadata['frame_range'][1]])
        ax1.set_xlabel('Frame', fontsize=34)  # 2x larger font size for X axis label
        ax1.set_ylabel('Label', fontsize=34)  # 2x larger font size for Y axis label
        ax1.set_yticks([0, 1, 2])
```

```python
        ax1.set_yticklabels(['FU', 'FD', 'NEITHER'], fontsize=30)  # 2x larger font size for Y tick labels
        ax1.legend(loc='lower right', fontsize=24)  # Adjust legend font size if needed

        # Increase tick label size
        ax1.tick_params(axis='x', labelsize=24)  # Adjust X tick label size if needed
        ax1.tick_params(axis='y', labelsize=24)  # Adjust Y tick label size if needed

        plt.title(f'FU/FD Predictions for Capture: {capture_name}', fontsize=30)  # 2x larger font size for the title
        plt.show()


    def find_consecutive_segments(self, predictions=[], min_length=5):
        if not isinstance(predictions,  np.ndarray):
            predictions = self.labels[self.current_index]
        segments = []
        current_segment = []
        last_label = 2

        for i, label in enumerate(predictions):
            if label == last_label and label in [0, 1]:  # GOUP or DOWN
                current_segment.append(i)
            else:
                if len(current_segment) >= min_length:
                    segments.append((current_segment[0], last_label))
                current_segment = [i] if label in [0, 1] else []
            last_label = label

        # Check the last segment
        if len(current_segment) >= min_length:
            segments.append((current_segment[0], last_label))

        return segments

    def generate_full_confusion_matrix(self, segments, true_segments, full_predictions, window=12):
        true_labels = self.labels[self.current_index]
        y_pred = []
        y_true = []
        for start, label in segments:
            # Look for the corresponding start in true_labels within a 10-frame window
            for i in range(max(0, start - window), min(len(true_labels), start + window)):
                if true_labels[i] == label:
                    y_pred.append(label)
                    y_true.append(label)
                    break
            else:
                y_pred.append(label)
```

```
                y_true.append(2)  # Neither

        # Second pass: look for false negatives using a window approach
        window = 10
        for start, label in true_segments:
            # Look for the corresponding start in true_labels within a 10-frame window
            for i in range(max(0, start - window), min(len(true_labels), start + window)):
                if full_predictions[i] == label:
                    # Not a false negative
                    break
            else:
                y_pred.append(full_predictions[i])
                y_true.append(label)  # Neither

        return confusion_matrix(y_true, y_pred, labels=[0, 1, 2])

    def generate_confusion_matrix(self, segments, window=12):
        true_labels = self.labels[self.current_index]
        y_pred = []
        y_true = []
        for start, label in segments:
            # Look for the corresponding start in true_labels within a 10-frame window
            for i in range(max(0, start - window), min(len(true_labels), start + window)):
                if true_labels[i] == label:
                    y_pred.append(label)
                    y_true.append(label)
                    break
            else:
                y_pred.append(label)
                y_true.append(2)  # Neither

        return confusion_matrix(y_true, y_pred, labels=[0, 1, 2])


    def generate_confusion_matrix_with_window_for_false_negatives(self, segments, window=12):
        true_labels = self.labels[self.current_index]
        y_pred = []
        y_true = []
        used_true_labels = []  # Keep track of which true labels have been matched

        # First pass: look for true positives and false positives
        for start, label in segments:
            found_match = False
            for i in range(max(0, start - window), min(len(true_labels), start + window)):
                if true_labels[i] == label and i not in used_true_labels:
                    y_pred.append(label)
                    y_true.append(label)
```

```python
                    used_true_labels.append(i)
                    found_match = True
                    break
            if not found_match:
                y_pred.append(label)
                y_true.append(2)  # Neither

        print(f"Used true labels are: {used_true_labels}")

        # Second pass: look for false negatives using a window approach
        for i, label in enumerate(true_labels):
            # Only consider labels that are GOUP or DOWN and haven't been used
            if label in [0, 1] and i not in used_true_labels:
                # Check if there's a sequence of similar labels within a window
                sequence_found = False
                for j in range(max(0, i - window), min(len(true_labels), i + window)):
                    # If a sequence is detected
                    if true_labels[j] == label:
                        sequence_found = True
                        break

                if sequence_found:
                    # If a sequence of the same event type is found within the window, consider it a false negative
                    y_pred.append(2)  # Neither (predicted)
                    y_true.append(label)  # Actual event type
                else:
                    # If no sequence is found, it's not considered a false negative
                    used_true_labels.append(i)  # Mark as used to avoid re-evaluation

        return confusion_matrix(y_true, y_pred, labels=[0, 1, 2])

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        data = self.data[index]
        label = self.labels[index]
        length = len(data)  # Or however you calculate the length of your sequence
        metadata = self.all_metadata[index]

        return data, label, length, metadata


    @staticmethod
    def collate_fn(batch):
        # Unzip the batch to separate sequences, labels, lengths, and metadata
```

```
        sequences, labels, lengths, metadata = zip(*batch)

        # Ensure sequences are tensors and pad them to have the same length
        sequences_padded = pad_sequence([torch.tensor(seq, dtype=torch.float) for seq in sequences], batch_first=True)

        # Similarly, pad labels if they are of variable lengths
        labels_padded = pad_sequence([torch.tensor(label, dtype=torch.long) for label in labels], batch_first=True, padding_value=-1)  # Use -1 as an ignore index if labels are of variable lengths

        # Convert lengths to a tensor
        lengths_tensor = torch.tensor(lengths, dtype=torch.long)

        return sequences_padded, labels_padded, lengths_tensor, metadata

    def plot_labels_and_ranges(self, index):
        """
        Plots the labels and frame ranges for a single capture.

        Parameters:
        - index: Index of the capture to plot in the dataset.
        """
        if index >= len(self.data):
            print("Index out of range.")
            return

        metadata = self.all_metadata[index]
        labels = self.labels[index]

        plt.figure(figsize=(20, 5))

        # Plot labels
        plt.plot(labels, label='Labels')

        plt.title(f'Labels and Frame Ranges for Capture: {metadata["RADAR_capture"]}')
        plt.xlabel('Frame Index')
        plt.ylabel('Label')
        plt.yticks([0, 1, 2], ['GOUP', 'DOWN', 'NEITHER'])
        plt.xlim([metadata['frame_range'][0], metadata['frame_range'][1]])
        plt.legend()

        plt.show()
```

Listing A.5: Full Capture RDM Dataset Python Class Template

A.3.2 Stability Phase

```
import pandas as pd
import os
import numpy as np
```

```python
import torch
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from FPDataCapture import FPDataCapture

class StableRdmDataset(Dataset):
    """Dataset class for processing and loading radar and motion capture data for stability analysis in yoga poses."""

    def __init__(self, root_dir, event_csv, included_folders, label_type="avg_speed"):
        """Initializes the dataset with the directory of the data, an events CSV file, and the specific folders to include."""
        label_types = ['avg_velocity_squared', 'max_distance_from_centroid', "avg_speed", "sqrt_of_avg_speed"]
        if label_type not in label_types:
            raise ValueError(f"Invalid label type. Expected one of: {label_types}")

        self.data = []
        self.labels = []
        self.metadata = []
        self.force_plate_dir = "/Volumes/FourTBLaCie/Yoga_Study_FP_1and2_MNTR"
        self.num_channels = 4
        self.event_labels_df = pd.read_csv(event_csv)

        for folder_name in included_folders:
            folder_path = os.path.join(root_dir, folder_name)
            filtered_df = self.event_labels_df[self.event_labels_df['RADAR_capture'].str.startswith(folder_name)]
            for index, row in filtered_df.iterrows():
                radar_capture = row['RADAR_capture']
                frame_end = row['frame_end'] if np.isnan(row['frame_break']) else row['frame_break']
                t_end = row['t_foot_down'] if np.isnan(row['t_break']) else row['t_break']

                for i in range(self.num_channels):
                    capture_and_tx = f"{radar_capture}_channel{i+1}_tx{row['tx']}"
                    radar_file_path = os.path.join(folder_path, capture_and_tx + '.npy')
                    if os.path.exists(radar_file_path):
                        rdm_data = np.load(radar_file_path)
                        self.data.append(rdm_data)
                        metadata = {
                            'RADAR_capture': radar_capture,
                            'participant_id': radar_capture[:2],
                            "tx": row['tx'],
                            'channel': i+1,
                            "n_frames": rdm_data.shape[0],
                            'seconds_per_frame': row['Seconds_per_Frame'],
                            'frame_range': (row['frame_stable'], frame_end
```

```
),
                            'time_range': (row['t_stable'], t_end)
                        }
                        self.metadata.append(metadata)

                        force_plate_capture = self.create_fp_data_capture(radar_capture)
                        filtered_force_plate_df = force_plate_capture.isolate_rows_by_time(row['t_stable'], t_end)

                        label = force_plate_capture.calculate_label(filtered_force_plate_df, label_type)
                        self.labels.append(label)

    def create_fp_data_capture(self, radar_capture):
        """Creates a data capture object for force plate data based on the radar capture name."""
        participant = radar_capture[:2]
        MOCAP_FP_capture_name = radar_capture.replace('_RR_', '_MC_')
        base_file_path = os.path.join(self.force_plate_dir, participant, MOCAP_FP_capture_name + '.tsv')
        return FPDataCapture(base_file_path=base_file_path, is_foot_always_up=True)

    def __len__(self):
        """Returns the total number of samples in the dataset."""
        return len(self.data)

    def __getitem__(self, index):
        """Fetches a sample and its associated data from the dataset."""
        data = self.data[index]
        label = self.labels[index]
        metadata = self.metadata[index]
        length = metadata['n_frames']

        return data, label, length, metadata

    @staticmethod
    def collate_fn(batch):
        """Custom collate function to manage data batching."""
        sequences, labels, _, metadata = zip(*batch)
        sequences_padded = pad_sequence([torch.tensor(seq, dtype=torch.float32) for seq in sequences], batch_first=True)
        labels_padded = torch.tensor(labels, dtype=torch.float32)
        lengths = [md['n_frames'] for md in metadata]

        lengths_tensor = torch.tensor(lengths, dtype=torch.long)

        return sequences_padded, labels_padded, lengths_tensor, metadata
```

Listing A.6: Stability Phase Dataset Python Class Template

A.4 Model Classes

A.4.1 Full Capture RDM Classifier

```python

import pandas as pd
import os
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms.functional import resize
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence
from torch import optim

class RdmClassifier(nn.Module):
    def __init__(self, num_classes, hidden_size):
        super(RdmClassifier, self).__init__()
        self.num_classes = num_classes
        # Define CNN architecture
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1),  # RDMs have a single channel
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),  # Flatten the output of the convolutional layers
        )
        cnn_output_size = self._get_conv_output_size()

        # Define the LSTM layer
        self.lstm = nn.LSTM(cnn_output_size, hidden_size, batch_first=True)

        # Define the fully connected layer for classification
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x, lengths):
        x = x.float()  # Ensure input is float type
        batch_size, seq_len, _, _ = x.size()
        # Apply CNN to each RDM in the sequence
        c_out = self.cnn(x.view(batch_size * seq_len, 1, *x.size()[-2:]))

        # Reshape for LSTM input
        r_out = c_out.view(batch_size, seq_len, -1)

        # Pack the sequence for LSTM
        packed_input = pack_padded_sequence(r_out, lengths, batch_first=True, enforce_sorted=False)
        # Instead of using just the last hidden state
        packed_output, (hidden, cell) = self.lstm(packed_input)
        # Decode the packed output
```

```python
        lstm_out, _ = torch.nn.utils.rnn.pad_packed_sequence(packed_output, batch_first=True)
        # Apply the fully connected layer to all time steps
        out = self.fc(lstm_out)
        return out

    def _get_conv_output_size(self):
        with torch.no_grad():
            dummy_input = torch.zeros(1, 1, 23, 13)
            dummy_output = self.cnn(dummy_input)
            return dummy_output.size(-1)
```

Listing A.7: RDM Classifier Python Class Template

A.4.2 Stability Phase Predictor

```python

import torch
import torch.nn as nn
import torch.nn.functional as F

class RdmCNNLSTMModel(nn.Module):
    def __init__(self, num_channels, hidden_dim, lstm_layers=1, bidirectional=False):
        super(RdmCNNLSTMModel, self).__init__()
        self.num_channels = num_channels

        # Convolutional layers
        self.conv1 = nn.Conv2d(in_channels=num_channels, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.cnn_output_size = self._get_conv_output_size()

        # LSTM layers
        self.hidden_dim = hidden_dim
        self.lstm_layers = lstm_layers
        self.bidirectional = bidirectional
        self.lstm = nn.LSTM(input_size=self.cnn_output_size, hidden_size=hidden_dim, num_layers=lstm_layers, batch_first=True, bidirectional=bidirectional)

        # Linear layer for output
        direction_multiplier = 2 if bidirectional else 1
        self.fc = nn.Linear(hidden_dim * direction_multiplier, 1)  # Predicting a single value

    def forward(self, x):
        # Reshape output for LSTM layers
        batch_size, time_steps, height, width = x.shape
        x = x.view(batch_size * time_steps, 1, height, width)
```

```

        # Apply convolutional layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))

        # Reshape x back to [batch_size, time_steps, features] for LSTM processing
        x = x.view(batch_size, time_steps, self.cnn_output_size)

        # LSTM layers...
        lstm_out, _ = self.lstm(x)

        # Take the output of the last LSTM layer
        if self.bidirectional:
            lstm_out = lstm_out[:, -1, :]
        else:
            lstm_out = lstm_out[:, -1, :]

        # Linear layer
        out = self.fc(lstm_out)
        outputs = torch.squeeze(out)
        return outputs

    def _get_conv_output_size(self):
        # Create a dummy input to pass through the CNN layers to calculate output size
        # spatial dimensions of your input radar data are 23x13
        dummy_input = torch.zeros(1, self.num_channels, 23, 13)
        x = self.pool(F.relu(self.conv1(dummy_input)))
        x = self.pool(F.relu(self.conv2(x)))
        # Multiply the dimensions of the output feature map to get the total feature size
        return x.numel() // x.shape[0]  # Use numel() to get total number of features and divide by batch size (1 in this case)
```

Listing A.8: Stability Phase Predictor Python Class Template

Appendix B

Appendix

B.1 Technical Contributions to Sekisui House at MIT

B.1.1 Goals of Sekisui House at MIT

My research was primarily sponsored by Sekisui House at MIT, a joint venture between MIT's Institute for Medical Engineering and Science (IMES) and Sekisui House, one of Japan's leading homebuilders. This collaboration is dedicated to developing technologies that cater to the needs of an aging population through innovative in-home wellness monitoring and Early Detection Systems (EDS). By enabling individuals to stay healthy and independent in their own homes for as long as possible, Sekisui House at MIT aims to address the growing demands on healthcare systems and caregivers worldwide. The partnership leverages the capabilities of MIT's Clinical Center for Research Trials (CCTR) and HealthLab facilities, promoting educational and global exchanges among diverse communities. This initiative enhances medical and observational research to improve the quality of life for the elderly on a global scale.

B.1.2 Design and Implementation of a SQL Database

In collaboration with MIT, Sekisui House built two houses fully instrumented with a dense network of continuous wave (CW) radar and infrared sensors. Over a two-and-a-half-year period, Sekisui House collected over 40 TBs of supervised and unsupervised data on the occupants. This data was labeled by date and radar used; however, a relational database did not exist that could effectively connect the activities performed, occupants, places, and sensors. This connection was essential for researchers to identify, pull, and tag the correct data.

Conceptualization

The conceptualization phase began with identifying the primary objectives of the database system, which included the ability to track and correlate the diverse data streams from various sensors and interaction points within the houses. The goal was to create a framework that would facilitate complex queries involving multiple data types and support large-scale data analytics for ongoing research in aging and in-home care technologies. Key considerations were clarity, data integrity, scalability, and accessibility.

Design

The design of the SQL database was structured to support the complex needs of the Sekisui House research project. The database schema was developed to include tables for Activities, Subjects, Houses, Devices, Sensors, and Environmental Conditions, among others [Figure B.1]. Each table was designed to ensure relationships that would allow for efficient querying and analysis. For instance, the ‘Activities‘ table connects with the ‘Subjects‘ table through a foreign key that links each activity to an individual subject. Similarly, the ‘Devices‘ table relates to the ‘Sensors‘ table, enabling tracking of the effects of radiofrequency generating devices across different instrumentation.

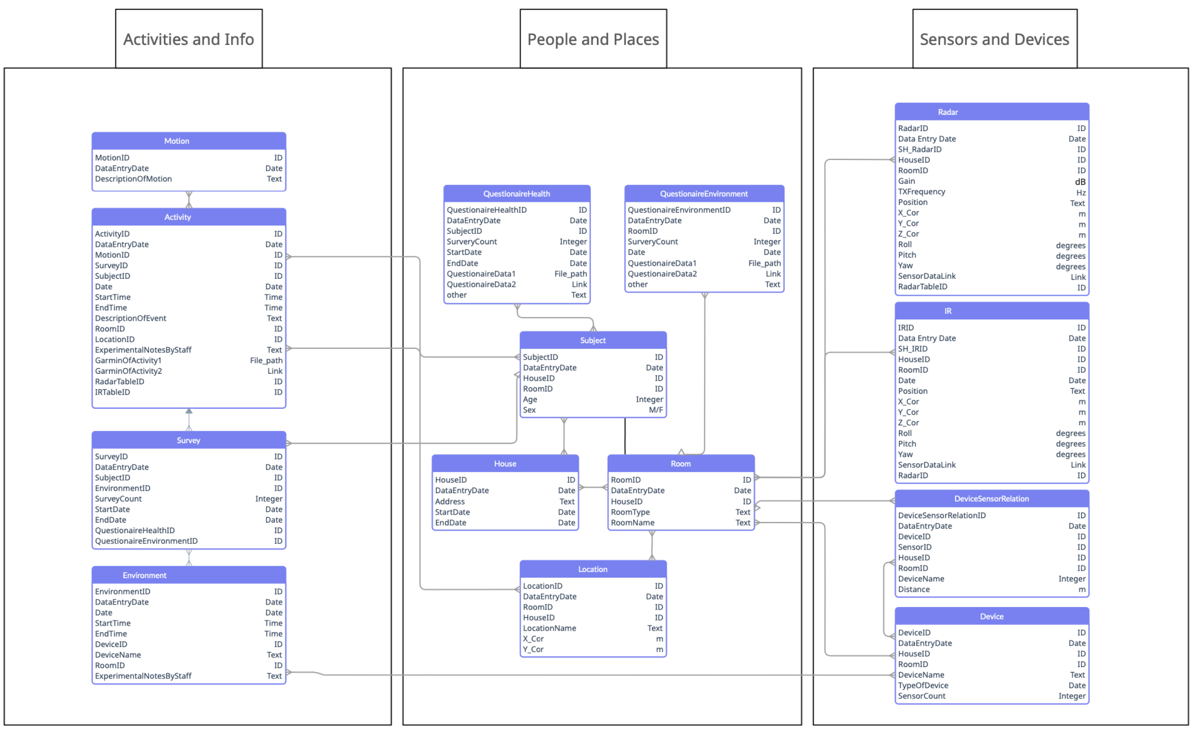

Figure B.1: Comprehensive SQL database schematic for Sekisui House at MIT: This schema integrates various modules, including Surveys, Environment, Activities, Radar, IR Systems, Questionnaires, Devices, Subjects, and Houses, detailing the relational structure and data types employed for effective in-home wellness monitoring and early detection systems within aging populations. Each table is outlined with attributes such as IDs, data entry dates, and specific device and subject identifiers to ensure precision in data collection and analysis.

Implementation

The implementation phase involved setting up the SQL database on a robust server infrastructure to handle the expected data load. Database indexing strategies were employed to optimize performance for frequent queries, such as those involving temporal data correlations between sensor readings and occupant activities. Procedures for data ingestion were established, with scripts developed to automate the parsing and loading of data from various sources directly into the database.

Maintenance and Usage

Post-implementation, the focus shifted to maintenance and ensuring the database's continuous operation. Researchers extensively use the database to generate customized reports, conduct statistical analyses, and develop machine-learning models that predict vital signs and health trends based on the collected data. An ongoing review process helps identify and rectify any inefficiencies in the database to improve response times and extend its capabilities as new types of sensors and data streams are introduced into the research environment.

B.1.3 Development of a Box Data Scraping Tool

Purpose and Design

The primary objective in developing the Box Data Scraping Tool was to streamline the process of retrieving large datasets from Box storage, a common repository for the immense volumes of data generated by Sekisui House's sensor networks. The design of the tool focuses on automating the extraction of files using metadata stored in the SQL database and an interface that allows for easy querying and identification of files of interest [Figure B.2].

Functionality

The functionality of the tool is best understood by its capability to handle and process structured CSV files that contain Box file IDs and filenames. This process is facilitated by a web-based interface, where users can upload a CSV file, input their Box access token, and specify a folder path where the files should be saved. The system is designed to offer researchers a seamless experience, enabling them to efficiently download the necessary files for their analysis without having to search through the Box directories [Figure B.3].

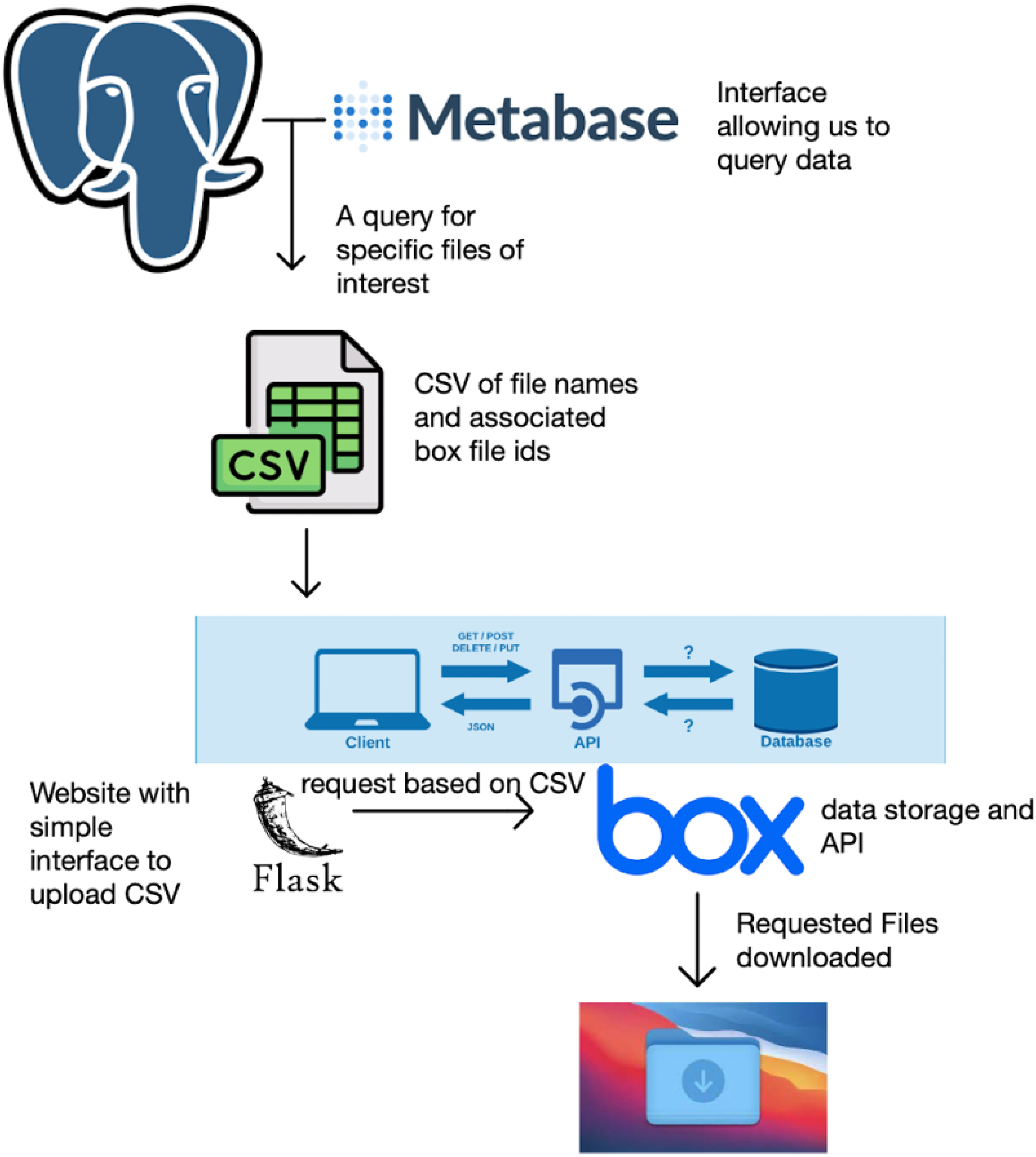

Figure B.2: Workflow diagram showing the process of data extraction from the SQL server, querying via Metabase, obtaining a CSV of file names and IDs, and interfacing with the Box API through a Flask-based client to download requested files.

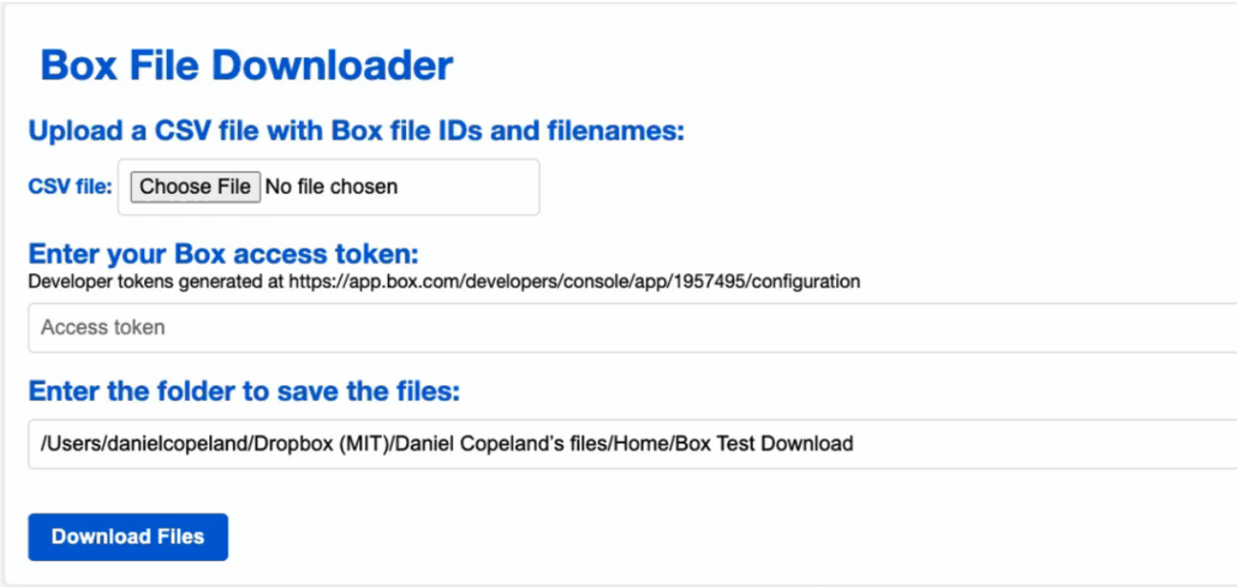

Figure B.3: Screenshot of the web interface used for the Box File Downloader tool. Users can upload the CSV file, input their access token, and specify the destination folder for the downloaded files.

B.1.4 Radar-Based Analysis Tools

Clutter-Cancel Canceller Tool for CW Radar Data

During the calibration of the large multi-radar system, it was noticed that there was a significant amount of drift in the I and Q channels of the CW radars. This drift was addressed with frequent clutter cancellation resets every two minutes across the multi-radar system. These resulted in a spike in the radar data across all frequencies. This raised the need for a robust outlier detection and spike removal tool. This tool employs a rolling 4.5 standard deviation outlier detection algorithm, which is instrumental in identifying anomalies within the CW radar data [Figure B.4].

The radar's ability to detect and measure vital signs and motion accurately was dependent on this tool's ability to remove these spikes. The implemented rolling standard deviation algorithm scans the radar data for spikes indicative of outliers and substitutes them with the local mean calculated within a dynamic window. This approach ensures a smoother data set, devoid of extreme variations that could lead to false readings or inaccuracies in vital sign detection and motion algorithms.

The tool has since become a cornerstone for researchers who rely on precision and accuracy when working with this radar data for vital sign detection and motion algorithms. Its ability to efficiently preprocess data ensures that subsequent analysis is based on high-quality and reliable data sets.

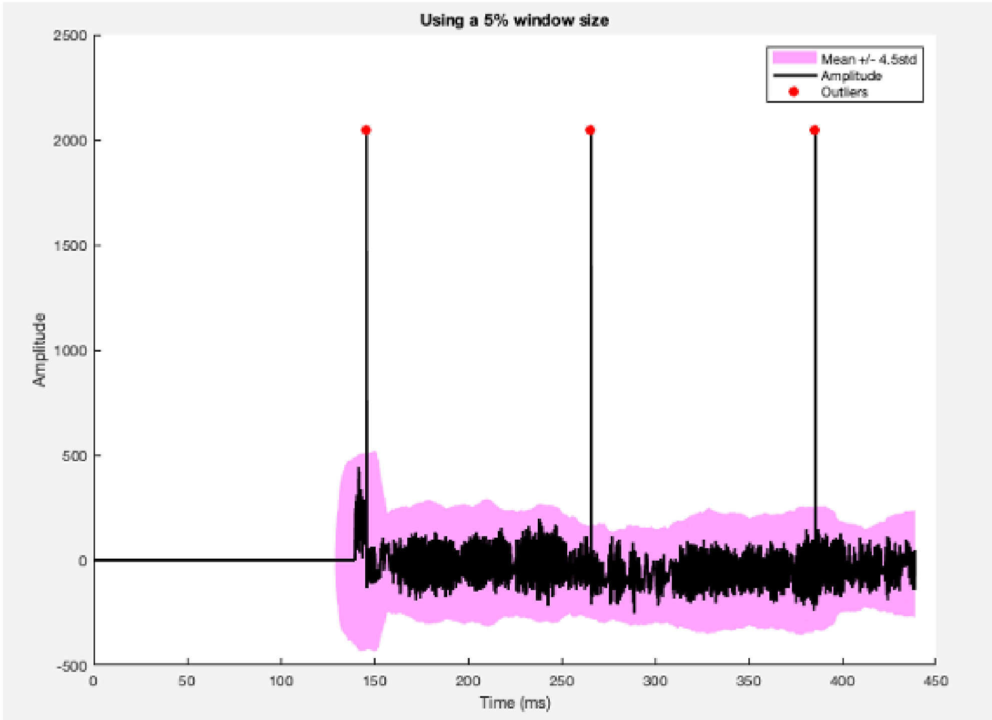

Figure B.4: Illustration of the rolling standard deviation outlier detection and spike removal technique applied to CW radar data. Spikes representing outliers are identified and replaced with the local mean to maintain data integrity for advanced vital sign detection and motion analysis.

Supine Respiratory Rate

A key application of the Clutter-Cancel Canceller tool is in the domain of vital sign monitoring. To test this tool, I applied the algorithm to radar data and analyzed it to detect the respiratory rate of a subject in a supine position. The respiratory rate is a crucial vital sign that indicates various medical conditions and the overall well-being of a patient.

A manual calculation of the respiratory rate was initially conducted to validate the reliability of the analysis. This manual approach involved counting the number of breathing cycles over a period, as represented by the fluctuations in the radar signal. In the specific case analyzed, a total of 65 cycles were counted over 3.93 minutes, resulting in a calculated respiratory rate of 16.5 breaths per minute Figure B.5.

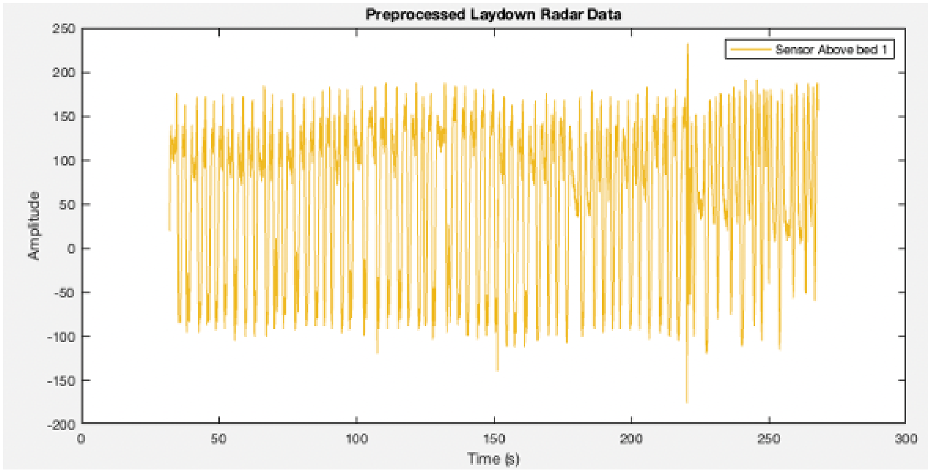

Figure B.5: Manually Calculating Respiratory Rate: The radar data, post application of the Clutter Cancel algorithm, showing 65 respiratory cycles over 3.93 minutes, indicating a respiratory rate of 16.5 breaths per minute.

Further analysis was conducted using Fast Fourier Transform (FFT) followed by smoothing to provide a more automated and precise measurement. The FFT analysis, followed by a Gaussian convolution, allowed for the identification of the dominant frequency component corresponding to the respiratory rate. The frequency of 0.28 breaths per second (equivalent to 16.8 breaths per minute) observed in the FFT analysis confirms the manual count, thereby validating the efficacy of the Clutter-Cancel algorithm for preprocessing CW radar data for monitoring vital signs [Figure B.6].

This synergy between manual methods and advanced signal processing techniques underpins the robustness of the radar data analysis, ensuring reliable vital sign monitoring in non-invasive settings.

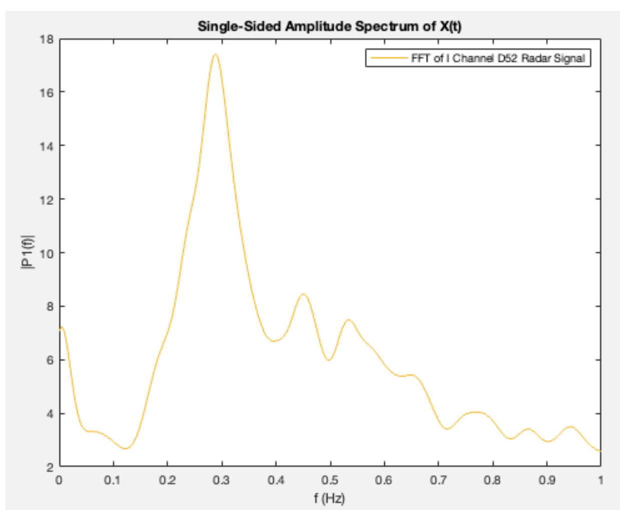

Figure B.6: FFT Analysis of Respiratory Rate: Smoothing through Gaussian convolution of the FFT reveals the primary frequency component of the radar signal, correlating to a respiratory rate of 16.8 breaths per minute.

Human Tracking Algorithm

The Human Tracking Algorithm represents the culmination of the diverse toolset developed for analyzing the Sekisui House dataset. This algorithm retrieves relevant radar data corresponding to specific activities and accurate timeframes by integrating an SQL database wrapper. Coupled with the Box Data Downloader, researchers can efficiently download the radar data of interest.

The human tracking algorithm refines the data by removing noise and spikes using the Clutter-Cancel tool, collates the multiple radars using recursion, and then applies a smoothed rolling standard deviation. This processing across multiple radar inputs allows for the precise tracking of human movement within the two-dimensional plane of the room, delineating both the x and y coordinates of individuals [Figure B.7].

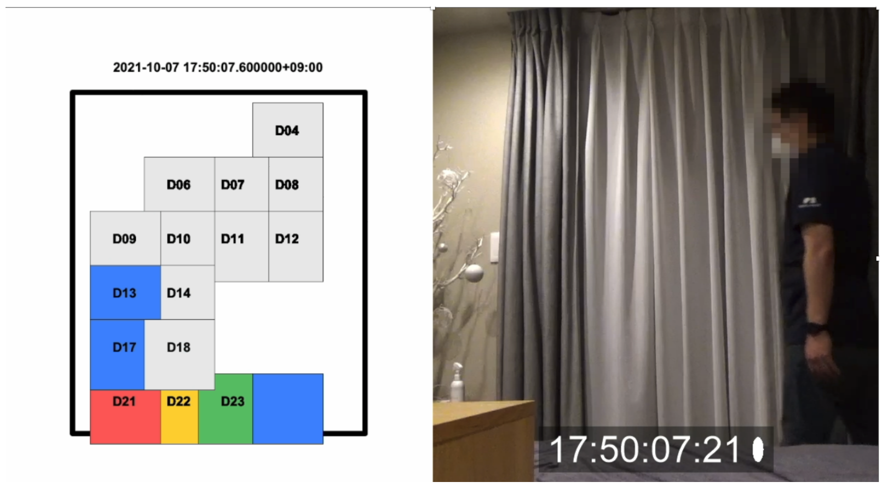

Figure B.7: A visual representation of the Human Tracking Algorithm in action. On the left, the grid overlay represents the radar segments, with colored blocks indicating active radar zones corresponding to human movement. On the right, the real-time video feed corroborates the radar data, with time stamps ensuring synchronicity between the two data sources.

This approach enhances the accuracy of human tracking in a controlled environment and extends the potential for non-intrusive monitoring in applications such as elderly care, security, and smart home systems. By mapping the detected movements to physical space, researchers can analyze patterns of life and draw significant conclusions about the behaviors and well-being of the subjects within these environments.

B.1.5 Discussion

The collaboration with Sekisui House at MIT has resulted in important technical contributions to the field of in-home wellness monitoring and early detection systems. Through this joint venture, we have developed a comprehensive suite of tools that have enhanced the ability to collect, process, and analyze data in innovative ways that cater to the needs of an aging population.

The creation of a robust SQL database has been foundational in structuring and analyzing the data collected from sensor-equipped homes. The database has enabled researchers to track a wide array of data points while maintaining its integrity and accessibility.

The Box Data Scraping Tool efficiently retrieves large datasets for researchers. By streamlining this process, we have enabled researchers to focus more on analysis rather than data management, thereby accelerating the pace of discovery and innovation.

The Clutter-Cancel Canceller Tool has addressed the critical challenge of data integrity in radar signal analysis. Its ability to remove noise and spikes from the radar data ensures that vital signs and movement are monitored with the highest level of accuracy. This tool's precision is evidenced in the manual and FFT analysis of respiratory rates, where it has demonstrated its efficacy in filtering out irrelevant data and spotlighting the true vital signals.

The Human Tracking Algorithm has showcased the full potential of our integrated tools. By providing a two-dimensional tracking capability, it has paved the way for advanced studies in human behavior and health monitoring, which are essential in the context of non-invasive elder care and smart home systems.

The success of these tools underlines the strength of the interdisciplinary approach that Sekisui House at MIT embodies. It also showcases the value of academic-industrial partnerships in pushing the boundaries of technology for social good. Due in large part to Sekisui House's vision, this project's outcomes extend beyond its business objectives. They provide critical tools and methodologies that advance the scientific understanding of healthy aging, ultimately contributing to the well-being and quality of life of elder populations globally.

References

[1] B. H. Alexander, F. P. Rivara, and M. E. Wolf, "The cost and frequency of hospitalization for fall-related injuries in older adults.," *American Journal of Public Health*, vol. 82, pp. 1020–1023, 7 Jul. 1992, ISSN: 0090-0036. DOI: 10.2105/AJPH.82.7.1020.

[2] W. H. O. Ageing and L. C. Unit, *WHO global report on falls prevention in older age*. World Health Organization, 2008.

[3] L. D. Gillespie, M. C. Robertson, W. J. Gillespie, C. Sherrington, S. Gates, L. Clemson, and S. E. Lamb, "Interventions for preventing falls in older people living in the community," *Cochrane Database of Systematic Reviews*, vol. 2021, 6 Sep. 2012, ISSN: 14651858. DOI: 10.1002/14651858.CD007146.pub3.

[4] M. F. Ong, K. L. Soh, R. Saimon, M. W. Wai, M. Mortell, and K. G. Soh, "Fall prevention education to reduce fall risk among community-dwelling older persons: A systematic review," *Journal of Nursing Management*, vol. 29, pp. 2674–2688, 8 Nov. 2021, ISSN: 0966-0429. DOI: 10.1111/jonm.13434.

[5] L. D. Ott, "The impact of implementing a fall prevention educational session for community-dwelling physical therapy patients," *Nursing Open*, vol. 5, pp. 567–574, 4 Oct. 2018, ISSN: 2054-1058. DOI: 10.1002/nop2.165.

[6] C. S. Colón-Emeric, C. L. McDermott, D. S. Lee, and S. D. Berry, "Risk assessment and prevention of falls in older community-dwelling adults," *JAMA*, vol. 331, p. 1397, 16 Apr. 2024, ISSN: 0098-7484. DOI: 10.1001/jama.2024.1416.

[7] N. Salari, N. Darvishi, M. Ahmadipanah, S. Shohaimi, and M. Mohammadi, "Global prevalence of falls in the older adults: A comprehensive systematic review and meta-analysis," *Journal of Orthopaedic Surgery and Research*, vol. 17, p. 334, 1 Jun. 2022, ISSN: 1749-799X. DOI: 10.1186/s13018-022-03222-1.

[8] M. Steverson, *Https://www.who.int/news-room/fact-sheets/detail/ageing-and-health*, Aug. 2022.

[9] C. G. Araujo, C. G. de Souza e Silva, J. A. Laukkanen, M. F. Singh, S. K. Kunutsor, J. Myers, J. F. Franca, and C. L. Castro, "Successful 10-second one-legged stance performance predicts survival in middle-aged and older individuals," *British Journal of Sports Medicine*, vol. 56, pp. 975–980, 17 Sep. 2022, ISSN: 0306-3674. DOI: 10.1136/bjsports-2021-105360.

[10] B. A. Springer, R. Marin, T. Cyhan, H. Roberts, and N. W. Gill, "Normative values for the unipedal stance test with eyes open and closed," *Journal of Geriatric Physical Therapy*, vol. 30, pp. 8–15, 1 Apr. 2007, ISSN: 1539-8412. DOI: 10.1519/00139143-200704000-00003.

[11] A. Srivastav and S. Mandal, "Radars for autonomous driving: A review of deep learning methods and challenges," *IEEE Access*, vol. 11, pp. 97 147–97 168, 2023, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3312382.

[12] X. Li, Y. He, and X. Jing, "A survey of deep learning-based human activity recognition in radar," *Remote Sensing*, vol. 11, p. 1068, 9 May 2019, ISSN: 2072-4292. DOI: 10.3390/rs11091068.

[13] B. van Berlo, A. Elkelany, T. Ozcelebi, and N. Meratnia, "Millimeter wave sensing: A review of application pipelines and building blocks," *IEEE Sensors Journal*, vol. 21, pp. 10 332–10 368, 9 May 2021, ISSN: 1530-437X. DOI: 10.1109/JSEN.2021.3057450.

[14] N. Mandischer, I. Koop, A. Granich, D. Heberling, and B. Corves, "Radar tracker for human legs based on geometric and intensity features," IEEE, Aug. 2021, pp. 1521–1525, ISBN: 978-9-0827-9706-0. DOI: 10.23919/EUSIPCO54536.2021.9616134.

[15] B. Chen, P. Liu, F. Xiao, Z. Liu, and Y. Wang, "Review of the upright balance assessment based on the force plate," *International Journal of Environmental Research and Public Health*, vol. 18, p. 2696, 5 Mar. 2021, ISSN: 1660-4601. DOI: 10.3390/ijerph18052696.

[16] B. Jokanovic and M. Amin, "Fall detection using deep learning in range-doppler radars," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, pp. 180–189, 1 Feb. 2018, ISSN: 0018-9251. DOI: 10.1109/TAES.2017.2740098.

[17] P. Zhao, C. X. Lu, B. Wang, N. Trigoni, and A. Markham, "Cubelearn: End-to-end learning for human motion recognition from raw mmwave radar signals," *IEEE Internet of Things Journal*, vol. 10, pp. 10 236–10 249, 12 Jun. 2023, ISSN: 2327-4662. DOI: 10.1109/JIOT.2023.3237494.

[18] M. M. Vázquez, *Basics of fmcw radar*. Sep. 2021.

[19] P. Hugler, M. Geiger, and C. Waldschmidt, "Rcs measurements of a human hand for radar-based gesture recognition at e-band," IEEE, Mar. 2016, pp. 259–262, ISBN: 978-3-9812-6687-0. DOI: 10.1109/GEMIC.2016.7461605.

[20] S. Z. Gurbuz and M. G. Amin, "Radar-based human-motion recognition with deep learning: Promising applications for indoor monitoring," *IEEE Signal Processing Magazine*, vol. 36, pp. 16–28, 4 Jul. 2019, ISSN: 1053-5888. DOI: 10.1109/MSP.2018.2890128.

[21] X. Yang, J. Liu, Y. Chen, X. Guo, and Y. Xie, "Mu-id: Multi-user identification through gaits using millimeter wave radios," IEEE, Jul. 2020, pp. 2589–2598, ISBN: 978-1-7281-6412-0. DOI: 10.1109/INFOCOM41043.2020.9155471.

[22] B. Vandersmissen, N. Knudde, A. Jalalvand, I. Couckuyt, A. Bourdoux, W. D. Neve, and T. Dhaene, "Indoor person identification using a low-power fmcw radar," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, pp. 3941–3952, 7 Jul. 2018, ISSN: 0196-2892. DOI: 10.1109/TGRS.2018.2816812.

[23] X. Huang, Z. Ju, and R. Zhang, "Real-time heart rate detection method based on 77 ghz fmcw radar," *Micromachines*, vol. 13, p. 1960, 11 Nov. 2022, ISSN: 2072-666X. DOI: 10.3390/mi13111960.

[24] E. Turppa, J. M. Kortelainen, O. Antropov, and T. Kiuru, "Vital sign monitoring using fmcw radar in various sleeping scenarios," *Sensors*, vol. 20, p. 6505, 22 Nov. 2020, ISSN: 1424-8220. DOI: 10.3390/s20226505.

[25] Z. Li, J. L. Kernec, Q. Abbasi, F. Fioranelli, S. Yang, and O. Romain, "Radar-based human activity recognition with adaptive thresholding towards resource constrained platforms," *Scientific Reports*, vol. 13, p. 3473, 1 Mar. 2023, ISSN: 2045-2322. DOI: 10.1038/s41598-023-30631-x.

[26] A. Sengupta, F. Jin, R. Zhang, and S. Cao, "Mm-pose: Real-time human skeletal posture estimation using mmwave radars and cnns," *IEEE Sensors Journal*, vol. 20, pp. 10 032–10 044, 17 Sep. 2020, ISSN: 1530-437X. DOI: 10.1109/JSEN.2020.2991741.

[27] G. Paterniani, G. Paterniani, D. Sgreccia, A. DAVOLI, G. Guerzoni, P. D. Viesti, A. C. Valenti, and et al., "Radar-based monitoring of vital signs: A tutorial overview," DOI: 10.36227/techrxiv.19212918.v1. URL: https://doi.org/10.36227/techrxiv.19212918.v1.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 8 Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[29] T. T. Niemirepo, M. Viitanen, and J. Vanne, "Binocular multi-cnn system for real-time 3d pose estimation," ACM, Oct. 2020, pp. 4553–4555, ISBN: 9781450379885. DOI: 10.1145/3394171.3414456.

[30] "Deep learning models for yoga pose monitoring," *Algorithms*, vol. 15, p. 403, 11 Oct. 2022, ISSN: 1999-4893.

[31] B. Erol, S. Z. Gurbuz, and M. G. Amin, "Motion classification using kinematically sifted acgan-synthesized radar micro-doppler signatures," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, pp. 3197–3213, 4 Aug. 2020, ISSN: 0018-9251. DOI: 10.1109/TAES.2020.2969579.

[32] M. S. Seyfioglu and S. Z. Gurbuz, "Deep neural network initialization methods for micro-doppler classification with low training sample support," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, pp. 2462–2466, 12 Dec. 2017, ISSN: 1545-598X. DOI: 10.1109/LGRS.2017.2771405.

[33] E. A. Wikstrom, M. D. Tillman, A. N. Smith, and P. A. Borsa, "A new force-plate technology measure of dynamic postural stability: The dynamic postural stability index.," *Journal of athletic training*, vol. 40, pp. 305–9, 4 2005, ISSN: 1062-6050.

[34] F. Quijoux, A. Nicolaï, I. Chairi, *et al.*, "A review of center of pressure (cop) variables to quantify standing balance in elderly people: Algorithms and open-access code*," *Physiological Reports*, vol. 9, 22 Nov. 2021, ISSN: 2051-817X. DOI: 10.14814/phy2.15067.

[35] N. Eichler, S. Raz, A. Toledano-Shubi, D. Livne, I. Shimshoni, and H. Hel-Or, "Automatic and efficient fall risk assessment based on machine learning," *Sensors*, vol. 22, p. 1557, 4 Feb. 2022, ISSN: 1424-8220. DOI: 10.3390/s22041557.

[36] J. W. Blaszczyk and R. Orawiec, "Assessment of postural control in patients with parkinson's disease: Sway ratio analysis," *Human Movement Science*, vol. 30, pp. 396–404, 2 Apr. 2011, ISSN: 01679457. DOI: 10.1016/j.humov.2010.07.017.

[37] F. Fereidouni, "Human health risk assessment of 4-12 ghz radar waves using cst studio suite software," *Journal of Biomedical Physics and Engineering*, vol. 12, 3 Jul. 2022, ISSN: 22517200. DOI: 10.31661/jbpe.v0i0.1272.

[38] A.-K. Seifert, M. G. Amin, and A. M. Zoubir, "Toward unobtrusive in-home gait analysis based on radar micro-doppler signatures," *IEEE Transactions on Biomedical Engineering*, vol. 66, pp. 2629–2640, 9 Sep. 2019, ISSN: 0018-9294. DOI: 10.1109/TBME.2019.2893528.

[39] F. D. Enggar, A. M. Muthiah, O. D. Winarko, O. N. Samijayani, and S. Rahmatia, "Performance comparison of various windowing on fmcw radar signal processing," IEEE, Nov. 2016, pp. 326–330, ISBN: 978-1-5090-3840-4. DOI: 10.1109/ISESD.2016.7886743.

[40] C.-H. Lee and T.-L. Sun, "Evaluation of postural stability based on a force plate and inertial sensor during static balance measurements," *Journal of Physiological Anthropology*, vol. 37, p. 27, 1 Dec. 2018, ISSN: 1880-6805. DOI: 10.1186/s40101-018-0187-5.

[41] A. Ross and S. Thomas, "The health benefits of yoga and exercise: A review of comparison studies," *The Journal of Alternative and Complementary Medicine*, vol. 16, pp. 3–12, 1 Jan. 2010, ISSN: 1075-5535. DOI: 10.1089/acm.2009.0044.

[42] Z. Peng, *Radar sim py*, https://github.com/radarsimx/radarsimpy, 2023.