Media Engineering and Technology Faculty
German University in Cairo

Machine Learning based analysis of sports movement

Bachelor Thesis

Author: Abdelrahman Mohamed Elsayed Darwish
Supervisor: Prof. Dr. Mohamed Ehsan Ashour

Submission Date: 19 May, 2024

Media Engineering and Technology Faculty
German University in Cairo

Machine Learning based analysis of sports movement

Bachelor Thesis

Author: Abdelrahman Mohamed Elsayed Darwish
Supervisor: Prof. Dr. Mohamed Ehsan Ashour

Submission Date: 19 May, 2024

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Abdelrahman Mohamed Elsayed Darwish
19 May, 2024

Acknowledgments

Above all, I am deeply grateful to Allah for His abundant blessings in my life, particularly for granting me the opportunity to pursue my education with success, as He is the best of planners.

I extend my sincerest thanks to my supervisor, Dr. Mohamed Ashour, whose unwavering support, guidance, and willingness to invest time in mentoring me have been invaluable. His advice has greatly enriched the thesis.

I also want to express my profound appreciation to my family for their unwavering support throughout this endeavor. Their encouragement and belief in me have been a constant source of strength.

Furthermore, I am grateful to my friends for their support and assistance. Their contributions have significantly enhanced the quality of my work.

Abstract

The integration of wearable sensors and machine learning techniques in sports data analysis has revolutionized the way we understand and enhance athletic performance. This thesis aims to develop a wearable sensor capable of collecting sports-related data, with a focus on tennis and various human activities.

The project utilizes an ESP32-CAM module and an MPU-6050 sensor. We begin by programming these components to ensure their functionality. Following successful programming and testing, we proceed to design a printed circuit board (PCB) and integrate the components onto it. Subsequently, the constructed wearable sensor system undergoes testing by collecting data from various human and tennis activities, successfully recording all data.

The collected human activity data is then subjected to analysis using a decision tree machine learning algorithm to recognize different activities. Decision trees are chosen for activity recognition due to their interpretability and effectiveness. The results of the analysis are promising.

These promising results underscore the potential of our wearable sensor system in providing reliable and detailed sports data collection. The successful implementation and testing of this system pave the way for further advancements in the application of wearable technology and machine learning in sports data analysis.

Contents

List of Figures

List of Tables

List of Abbreviations

AAR	Animal Activity Recognition
AI	Artificial Intelligence
CAD	Computer-Aided Design
CSV	Comma-Separated Values
DTW	Dynamic Time Warping
dws	Going Down Stairs
EDA	Electronic Design Automation
GND	The Ground
HAR	Human Activity Recognition
IMUs	Inertial Measurement Units
IoT	Internet of Things
k-NN	K-Nearest Neighbors
PCB	Printed Circuit Board
PCA	Principal Component Analysis
RF	Random Forest
sit	Sitting
std	Standing
SVM	Support Vector Machines
TCP/IP	Transmission Control Protocol/Internet Protocol
ups	Going Up Stairs
wlk	Walking

Chapter 1

Introduction

In recent years, the integration of wearable sensors and machine learning techniques has revolutionized sports data analysis, providing unprecedented insights into athletic performance. This convergence of technology and sports science has not only enhanced training methodologies and injury prevention strategies but has also elevated overall athletic proficiency. Against this backdrop, the development of a wearable sensor system capable of collecting sports data and utilizing machine learning algorithms, specifically decision trees, for analysis has become increasingly important[2].

1.1 Thesis Objective

The primary objective of this thesis is to develop a wearable sensor system capable of collecting sports data and utilizing machine learning algorithms, specifically decision trees, for analysis. By doing so, we aim to have made a usable wearable sensor and a method for analyzing the data collected from this sensor.

1.2 Thesis Methodology

The methodology employed in this thesis encompasses several key steps, including programming and testing the ESP32-CAM module and MPU-6050 sensor for data collection, designing a printed circuit board (PCB) for integrating the components into a wearable device, testing the wearable sensor system by collecting data from tennis activities and various human activities, visualizing the collected data to identify patterns and trends, and implementing activity recognition using decision trees for human activity analysis.

1.3 Thesis Outline

The thesis is structured as follows:

- **Background (Chapter 2)**: This chapter gives an overview of athletic performance analysis, sensor technology in sports, machine learning in sports analytics, internet of things(IoT) applications in sports, data visualization techniques,Transmission Control Protocol/Internet Protocol (TCP/IP) Protocol,(comma-separated values) CSV files, and PCB design.
- **Related Work (Chapter 3)**: This chapter discusses existing research on wearable sensor technologies, the integration of ESP32-CAM and MPU6050, decision trees in classification, collection of accelerometer and gyroscope data, and machine learning-based activity recognition.
- **Methodology (Chapter 4)**: This chapter details the methodology used for developing the wearable sensor system and analyzing sports data, including system components, connections, data recording modes, data collection coding, PCB design, data collection attempts, machine learning analysis, and data analysis for recorded human activities.
- **Conclusion (Chapter 5)**: In this chapter, the findings and contributions of the thesis are summarized, and implications for future research are discussed.
- **Future Work (Chapter 6)**: This chapter explores potential avenues for further research and development in this area.

Chapter 2

Background

This chapter provides a comprehensive overview of several key areas essential for understanding the research context. It begins with athletic performance analysis, covering the evaluation of physical and technical capabilities, optimization of training regimens, injury prevention, and performance enhancement. Following this, the focus shifts to sensor technology in sports, discussing various types of sensors used for data collection. The chapter then explores the application of machine learning in sports analytics, introducing fundamental concepts and specific applications. Additionally, it delves into the Internet of Things (IoT) applications in sports, highlighting how IoT enhances data collection and analysis. Key data visualization techniques for sports analytics, such as average window with time, principal component analysis (PCA), confusion matrix plot, and decision tree plot, are also examined. Finally, the chapter discusses the Transmission Control Protocol/Internet Protocol(TCP/IP) Protocol, the use of Comma-Separated Values (CSV) files for data handling, and the basics of printed circuit board (PCB) design, which are crucial for developing wearable sensor technologies.

2.1 Athletic Performance Analysis

Athletic performance analysis involves the comprehensive assessment of athletes' physical and technical capabilities to optimize training programs, mitigate injury risks, and improve overall performance. Utilizing quantitative measures such as speed, agility, strength, and endurance, coaches and sports scientists can identify areas for improvement and tailor training strategies accordingly. By integrating data from diverse sources like sensors, video analysis, and performance evaluations, athletic performance analysis facilitates informed decision-making in training methodologies, competition tactics, and injury rehabilitation protocols.

Figure 2.1: Player Performance Analysis and Injury Prevention in Football.

2.1.1 Evaluation of Physical Capability

Evaluating athletes' physical attributes is fundamental to understanding their capabilities on the field or court. Speed, often considered a defining factor in many sports, determines how quickly an athlete can cover distances or react to opponents' movements. On the other hand, strength showcases an athlete's power and ability to exert force, crucial in activities like grappling sports and collisions in team sports. Agility measures an athlete's ability to change direction swiftly and maintain control of their body movements, essential in sports that demand sudden changes in direction, such as football and basketball. Endurance reflects an athlete's stamina and ability to sustain physical activity over prolonged periods, critical for sports that demand prolonged exertion such as long-distance running and swimming.

2.1.2 Evaluation of Technical Capability

Technical capability encompasses a wide range of skills that are specific to each sport. Precision in technique execution involves executing movements with accuracy and consistency, essential in sports such as basketball and gymnastics. Tactical understanding refers to an athlete's ability to make strategic decisions and anticipate opponents' moves, vital in team sports like football and rugby. Mastery of technical skills not only enhances individual performance but also contributes to overall team success.

2.1.3 Optimization of Training Regimens

Optimizing training regimens involves a comprehensive analysis of performance data, physiological metrics, and individual training needs. By understanding each athlete's strengths and weaknesses, coaches can design tailored training programs that address specific areas for improvement while maximizing overall athletic development.

2.1.4 Prevention of Injuries

Injuries are an unfortunate reality in sports, but proactive measures can significantly reduce their occurrence. Biomechanical analysis helps identify potential imbalances and movement deficiencies that may predispose athletes to injury. Implementing corrective exercises and movement patterns can address these issues and minimize injury risk.

2.1.5 Performance Enhancement

Through targeted training interventions, feedback mechanisms, and performance monitoring, athletes can achieve incremental improvements in their performance, reaching their full potential on the field or court.

2.2 Sensor Technology in Sports

Sensor technology plays a pivotal role in modern sports by enabling the measurement and analysis of biomechanical parameters, movement patterns, and physiological responses in athletes. Sensors can be integrated into wearable devices, equipment, or sports facilities to provide valuable feedback to athletes and coaches.

Figure 2.2: A brain sensor designed to optimize mental performance for football players, particularly during critical moments like penalty shootouts.

2.2.1 Types of Sensors

In sports, various sensors are utilized to capture a comprehensive range of data related to athletes' movements, biomechanics, and physiological responses. Among the most commonly employed sensors are accelerometers, gyroscopes, and inertial measurement units (IMUs), each serving distinct purposes in sports performance analysis:

2.2.1.1 Accelerometers

Accelerometers are sensors that measure acceleration forces exerted on an object along multiple axes. In sports, accelerometers are used to quantify the rate of change of an athlete's velocity and acceleration during movement. These sensors can detect linear acceleration, including both forward/backward, lateral, and vertical movements, providing valuable insights into an athlete's speed, agility, and explosive power. Accelerometers are often integrated into wearable devices, equipment, or sports apparel to monitor athletes' movements during training sessions and competitive events.

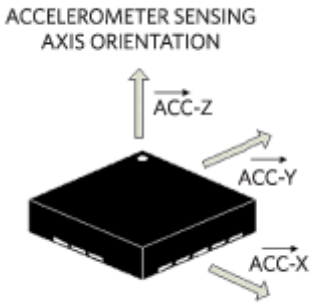


Figure 2.3: Accelerometer.

2.2.1.2 Gyroscopes

Gyroscopes are sensors that measure the rate of rotation or angular velocity around multiple axes. In sports, gyroscopes are employed to detect rotational movements, changes in orientation, and angular velocity of athletes' body segments or sports equipment. Gyroscopes provide crucial information about the rotational dynamics of movements such as spinning, twisting, or turning, offering insights into balance, coordination, and technique. Gyroscopes are commonly integrated into wearable devices, sports equipment, and specialized motion capture systems to analyze complex movements in sports like gymnastics, and diving.

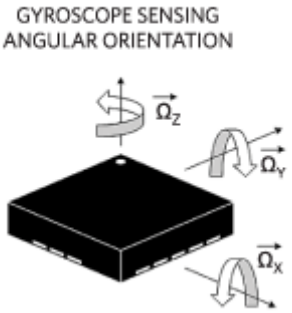


Figure 2.4: Gyroscope.

2.2.1.3 Inertial Measurement Units (IMUs)

IMUs combine accelerometers and gyroscopes into a single sensor package to measure both linear acceleration and angular velocity simultaneously. IMUs provide comprehensive motion tracking capabilities, enabling the capture of complex movements and kinematic data in sports. By integrating multiple sensors, IMUs offer a more complete understanding of athletes' movements, including linear and rotational components, which is essential for biomechanical analysis, motion assessment, and performance optimization. IMUs are widely used in sports science research, athlete monitoring systems, and wearable technology applications to quantify motion patterns, assess techniques, and prevent injuries.

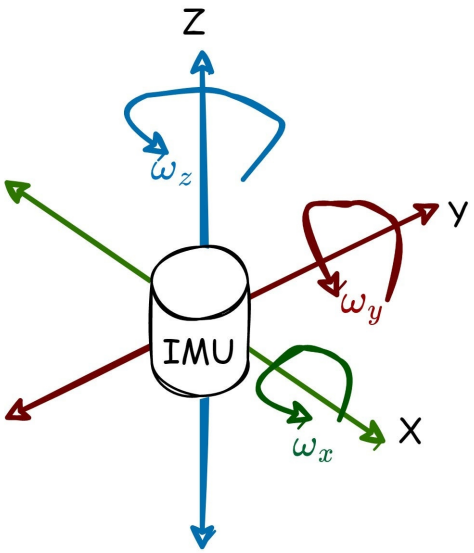


Figure 2.5: IMU.

2.2.1.4 Other Types of Sensors

In addition to accelerometers, gyroscopes, and IMUs, various other types of sensors are employed in sports to capture a comprehensive range of data. Heart rate monitors are extensively utilized to monitor athletes' cardiovascular responses during training and competition, providing insights into exertion levels and recovery rates. GPS trackers enable precise location tracking and movement analysis, facilitating performance evaluation in sports such as running, cycling, and team sports where positional awareness is crucial. Pressure sensors, on the other hand, are deployed to measure environmental factors such as air pressure, altitude, and barometric pressure, which can impact athletic performance, particularly in outdoor sports and high-altitude training scenarios. By integrating multiple sensor modalities, sports scientists can obtain a holistic view of athlete performance, encompassing biomechanical, physiological, and environmental dimensions. These sensor technologies play a crucial role in sports performance analysis by enabling the precise measurement, tracking, and interpretation of athletes' movements and biomechanical parameters. By leveraging data from accelerometers, gyroscopes, IMUs, and other sensor types, coaches, sports scientists, and athletes can gain valuable insights into performance metrics, technique quality, injury risks, and training effectiveness, ultimately enhancing athletic performance and competitive outcomes.

2.3 Machine Learning in Sports Analytics

2.3.1 Introduction to Machine Learning

Machine learning is a subfield of artificial intelligence (AI) that focuses on developing algorithms capable of learning from data to make predictions or decisions without being explicitly programmed. It encompasses various techniques such as supervised learning (using labeled data for prediction/classification), unsupervised learning (identifying patterns in unlabeled data), and reinforcement learning (learning through interaction with an environment). Machine learning algorithms learn patterns and relationships from datasets, allowing them to generalize and make predictions on new, unseen data. Applications of machine learning span across various domains, including finance, healthcare, marketing, and sports analytics.

2.3.2 Machine Learning in Sports

Machine learning has emerged as a powerful tool in sports analytics, enabling the analysis of large datasets to uncover patterns, trends, and insights that traditional statistical methods may overlook. By training models on historical data, machine learning algorithms can learn complex relationships between input features and output variables, allowing for the development of predictive models and decision support systems. This has led to more informed decision-making processes and has facilitated innovation in various aspects of sports management and performance analysis.

2.3.2.1 Machine Learning Approaches in Sports

Machine learning techniques are applied to various tasks in sports analytics, each serving specific purposes and offering unique advantages. Here are some common machine learning approaches used in sports analytics along with examples:

Decision Trees

Decision trees operate by recursively partitioning the input space based on specific parameters or features, a process crucial for effective classification tasks. These partitions create distinct branches in the tree structure, with each branch representing a different attribute or condition. In sports analytics, decision trees utilize various parameters such as player attributes, game situations, and environmental conditions to divide the data into subsets that are more homogeneous in terms of the target variable, facilitating accurate classification of player actions, game outcomes, or performance indicators.

Linear Regression

Linear regression, a statistical method, is utilized to establish a mathematical relationship between a dependent variable and one or more independent variables. For example, in basketball analytics, the dependent variable could be the number of points scored by a player in a game, while the independent variables could include minutes played. In the realm of sports analytics, linear regression finds application in projecting continuous outcome variables, such as player performance metrics. This predictive modeling relies on various input variables, including historical data, environmental conditions, and the strength of opponents.

Support Vector Machines (SVM)

SVM are a class of supervised learning algorithms used for classification and regression analysis. In sports analytics, SVM can be utilized for player classification tasks, such as categorizing athletes into skill levels or performance categories based on attributes like speed, strength, and skill proficiency.

Neural Networks

Neural networks are computational models inspired by the structure and function of the human brain. In sports analytics, neural networks can be applied to various tasks such as player tracking, image recognition, and game outcome prediction. For example, a neural network can be trained to recognize specific player movements or gestures from video footage, aiding in tactical analysis and performance evaluation.

2.3.2.2 Examples of Machine Learning Applications

Machine learning has found numerous applications in the realm of sports analytics, revolutionizing the way athletes are trained, games are analyzed, and strategies are devised. In this part, we explore several examples of machine learning applications in sports, ranging from player tracking and activity recognition to performance prediction and game strategy

optimization. These innovative applications harness the power of advanced algorithms and data-driven insights to enhance athlete performance, optimize team strategies, and achieve competitive success.

Activity Recognition

Activity recognition focuses on identifying and classifying different actions or activities performed by athletes during training or competition. Machine learning algorithms analyze sensor data, video feeds, or motion capture recordings to recognize specific movements, gestures, or exercise routines. For instance, activity recognition can be used to classify different types of sports actions such as dribbling, passing, and shooting in basketball or different swimming strokes in swimming. By accurately recognizing activities, coaches and sports scientists can monitor athletes' training loads, technique quality, and performance improvements over time.

Player Tracking

Player tracking involves the analysis of players' movements and positions during games or training sessions. Machine learning algorithms are employed to track players in video footage or using wearable tracking devices, allowing coaches and analysts to extract valuable insights into player performance, positioning, and tactical decisions. For example, tracking data can be used to assess players' running patterns, speed profiles, and spatial interactions on the field, enabling coaches to optimize team formations, strategy, and player substitutions.

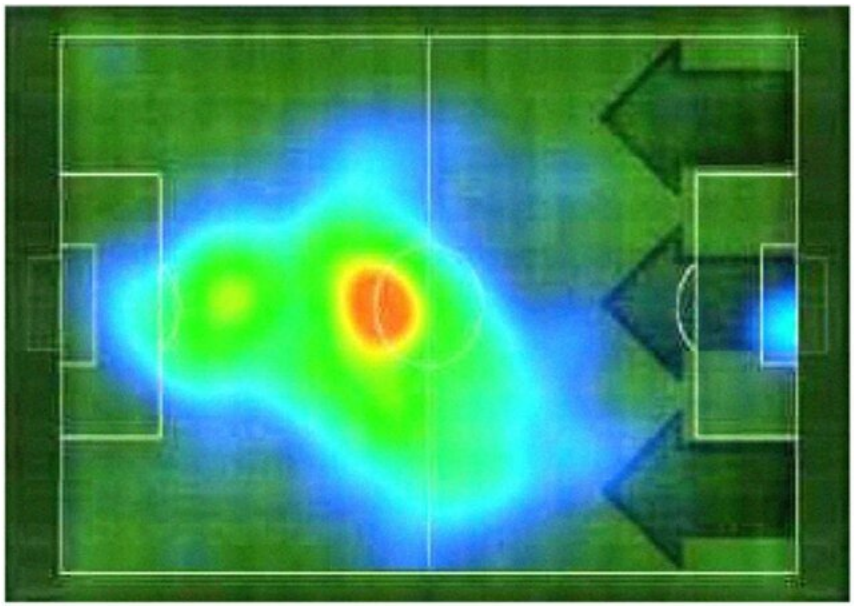


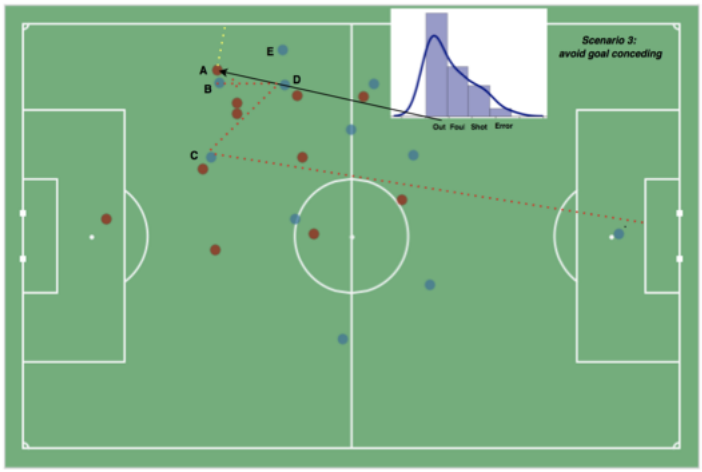
Figure 2.6: Heat map of a player movement in a football match.

Performance Prediction

Performance prediction involves forecasting athletes' future performance based on historical data, contextual factors, and environmental conditions. Machine learning models are trained on past performance data, including player statistics, match outcomes, and environmental variables, to predict future performance metrics such as scoring rates, game outcomes, or injury risks. For example, predictive models can estimate a soccer player's likelihood of scoring a goal based on factors such as their position on the field, opponent's defense strategy, and weather conditions. Performance prediction models assist coaches in making informed decisions about team selection, game strategies, and training interventions to optimize performance outcomes.

Game Strategy Optimization

Game strategy optimization focuses on using machine learning techniques to develop optimal game plans, tactics, and strategies for sports teams. By analyzing historical data, opponent statistics, and situational contexts, machine learning algorithms can identify patterns and trends in gameplay, enabling coaches to devise effective strategies to outperform opponents. For example, optimization algorithms can recommend lineup changes, substitution strategies, or tactical adjustments based on real-time performance data and opponent analysis. Game strategy optimization empowers coaches to make data-driven decisions that maximize their team's chances of success on the field.



(c) Scenario3: (performed action: error/bad ball control), (optimal action: out)

Figure 2.7: In a scenario where a player's ball control in a football match was bad, machine learning analysis suggests that the optimal course of action is for the player to kick the ball out of play[1].

In each of these machine-learning approaches, the utilization of advanced algorithms and data-driven insights revolutionizes the way sports are analyzed, coached, and played.

By leveraging the power of machine learning, sports organizations can gain a competitive edge, enhance athlete performance, and achieve success on and off the field.

2.4 Internet of Things (IoT) Applications in Sports

2.4.1 Introduction to IoT

IoT is a transformative technology that connects physical devices and objects to the Internet, enabling them to collect, exchange, and analyze data autonomously. IoT extends the power of the internet beyond traditional computing devices to a wide range of everyday objects, from household appliances and industrial machinery to vehicles and infrastructure. By embedding sensors, actuators, and communication capabilities into physical objects, IoT systems can monitor, control, and optimize processes in real time, leading to increased efficiency, productivity, and innovation across various domains.

2.4.2 IoT in Sports

IoT has transformed the sports industry by connecting physical objects and devices to the Internet, enabling seamless data collection and analysis. IoT applications in sports span athlete performance monitoring.

Figure 2.8: IoT in sports is depicted through athletes wearing an assortment of wearable devices.

2.4.2.1 Enhancing Performance and Decision-Making

IoT technology plays a crucial role in optimizing sports equipment and leveraging data analytics to drive performance improvements and strategic decisions in sports. By integrating IoT sensors into sports equipment, athletes receive real-time feedback on their performance, enabling them to enhance their skills and gameplay. Simultaneously, IoT-generated data provides a wealth of information that can be analyzed to extract actionable insights for athletes, coaches, and sports organizations.

2.5 Exploring Key Data Visualization Techniques for Sports Analytics

In this section, we explore essential data visualization techniques widely used in sports analytics. These techniques serve a pivotal role in interpreting and communicating analytical findings effectively. We delve into specific visualization methods, such as the average window with time, Principal Component Analysis (PCA), confusion matrix, and decision tree plot. Each visualization method offers unique insights into sports data, aiding in the interpretation of complex patterns and trends.

2.5.1 Average Window with Time

The average window with time plot is a visualization tool commonly used in sports analytics to analyze trends and variations in data over a specific time frame. This plot calculates the average values of data points over consecutive time intervals (windows) and displays them graphically. Each data point on the plot represents the average value within a particular time window.

This visualization technique helps in identifying long-term trends in data, allowing analysts to track changes over time and gain insights into consistency and variability.

Figure 2.9: This plot shows that using the average window made the data pattern clearer.

2.5.2 Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique used to visualize high-dimensional data in a lower-dimensional space while preserving the most important information. PCA identifies the principal components (orthogonal axes) that capture the maximum variance in the data and projects the data onto these components, effectively reducing the dimensionality.

PCA enables visualization of complex data in a lower-dimensional space, helps in identifying patterns and relationships among variables, and facilitates data exploration and interpretation by focusing on the most significant components.

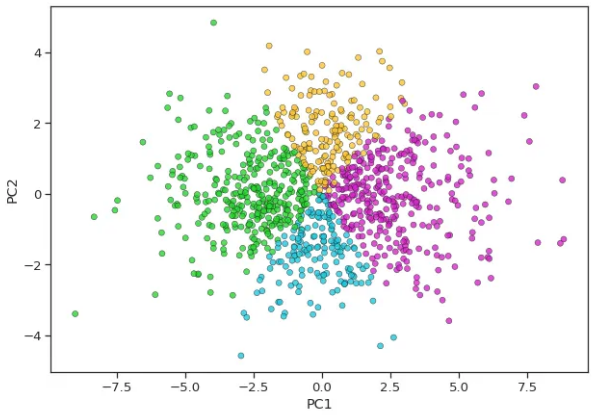


Figure 2.10: This plot represents three-dimensional data that has been reduced to two dimensions using PCA.

2.5.3 Confusion Matrix plot

A confusion matrix plot visually represents the performance of a classification model by comparing predicted labels with true labels. This plot consists of a table containing four key metrics: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

The confusion matrix plot provides a comprehensive evaluation of model performance, offering a clear visualization of the distribution of correct and incorrect predictions. It aids in understanding the types of errors made by the model.

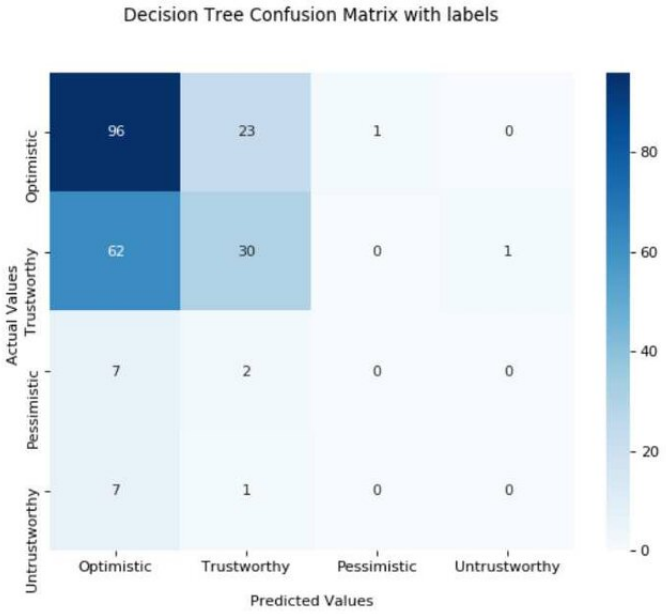


Figure 2.11: An Example of a confusion matrix

2.5.4 Decision Tree Plot

A decision tree plot illustrates the hierarchical structure and decision-making process of a decision tree model. Each node in the plot represents a decision based on a specific feature, with internal nodes indicating the decision criteria and leaf nodes representing the final class label or regression output.

Decision tree plots offer a transparent and interpretable visualization of the decision-making process, enabling stakeholders to understand how the model generates predictions. By visually depicting the logic behind the model, these plots facilitate communication and interpretation of model results in an intuitive manner.

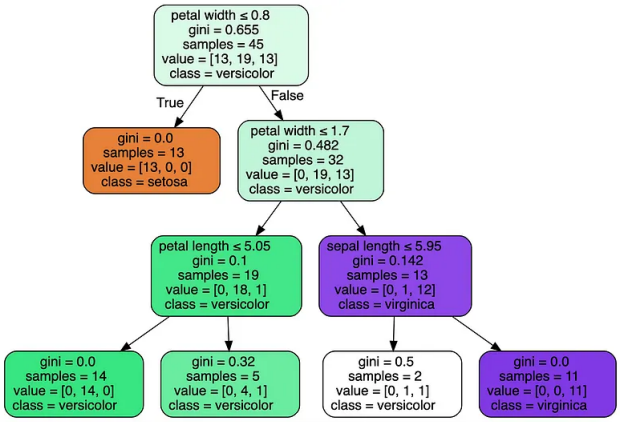


Figure 2.12: An Example of a decision tree.

2.6 TCP/IP Protocol

In computer networking, the Transmission Control Protocol (TCP) and Internet Protocol (IP) form the backbone of communication between devices on a network. TCP/IP is a suite of protocols that ensures reliable, end-to-end communication over interconnected networks.

2.6.1 Transmission Control Protocol (TCP)

TCP is a connection-oriented protocol designed to deliver data packets reliably and in order between devices. It establishes a connection between the sender and receiver before transmitting data, ensuring packets arrive intact, in sequence, and without duplication. Additionally, TCP manages flow control and congestion control mechanisms to optimize network performance.

2.6.2 Internet Protocol (IP)

IP is responsible for addressing and routing data packets across networks. It assigns unique IP addresses to devices and determines the most efficient path for data transmission using routing algorithms.

2.6.3 Port Numbers

Port numbers are used to identify specific endpoints or services on a device within TCP/IP communication. They facilitate the coexistence of multiple network applications on the same device and enable communication between them. Port numbers are categorized into well-known ports (0-1023), registered ports (1024-49151), and dynamic or private ports (49152-65535), with each range serving different purposes.

2.6.4 Sockets

Sockets represent endpoints for bidirectional communication across a network. They allow processes on different devices to exchange data, enabling seamless communication. In the TCP/IP context, sockets are identified by a combination of IP address and port number. Applications utilize socket programming interfaces to create, configure, and manage sockets for efficient network communication.

2.7 Comma-Separated Values (CSV) Files

CSV is a simple and widely used file format for storing tabular data. In a CSV file, each line represents a row of data, and each field within a row is separated by a delimiter, typically a comma. However, other delimiters such as semicolons or tabs may also be used. CSV files are plaintext, making them easy to create, edit, and parse using various programming languages and software applications. CSV files find extensive use in data interchange between different applications, as they provide a lightweight and universally accepted format for representing structured data.

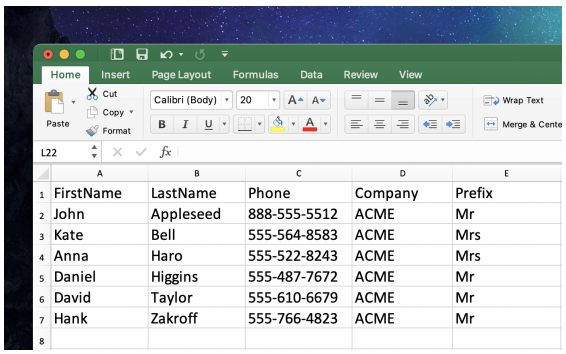
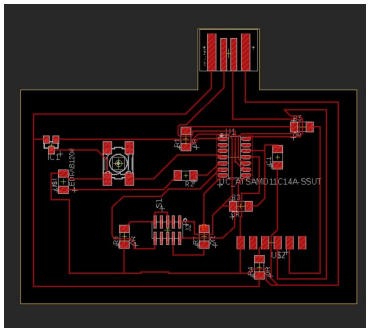


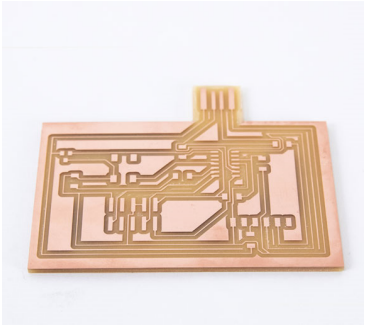
Figure 2.13: An example for a CSV file.

2.8 Printed Circuit Board (PCB)

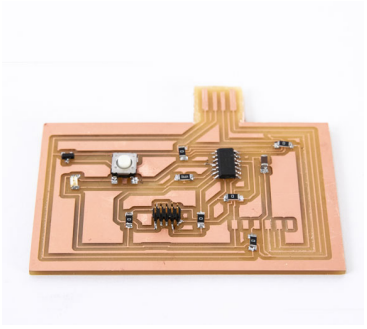
PCB design involves creating printed circuit boards that connect electronic parts, enabling the functionality of complex electronics. Designers utilize Computer-Aided Design (CAD) software, such as Fusion 360, to strategically place parts and manage signal pathways. Ensuring the integrity of signals is vital for reliable performance. Effective PCB design is essential for the functionality and reliability of electronic devices.



(a) A design for a PCP.



(b) The PCP after it was manufactured.



(c) PCB after components were soldered.

Figure 2.14: An example for a PCP.

Chapter 3

Related Work

In this section, we review several papers relevant to the thesis. We begin with the application and significance of wearable sensors in sports, followed by the integration of an ESP module with the MPU6050 sensor. Next, we discuss data collection methods, focusing on how data is gathered and labeled. Finally, we explore papers related to machine learning, specifically those addressing activity recognition and the use of decision trees.

3.1 Wearable Sensor Technologies in Sports

The paper by Morais [3] discusses the increasing trend in sports towards using wearable technology for real-time acquisition of physiological, kinematic, and kinetic data crucial for enhancing performance and reducing injury risks in athletes. Traditionally, video-based analysis has been used, but it is time-consuming and lacks real-time output. Wearables offer a practical alternative, allowing the monitoring of various parameters such as kinematics, kinetics, and physiological variables. The paper discusses five research explorations and their results:

Contributor 1 aimed to explore the measurement and monitoring of athletic performance, injury prevention, and rehabilitation using body wearable sensors. Key findings include that wearables enable the measurement and monitoring of athletic performance and facilitate injury prevention and rehabilitation.

Contributor 2 conducted a systematic review focusing on wearable technology applications in assessing physical performance among volleyball players. The review highlighted the prevalence of wearables in measuring key metrics like vertical jump height, crucial for actions such as blocking and attacking. However, it noted limitations, including correlations influenced by sample homogeneity and variations in underestimation due to factors like surface type. The study also identified a gap in comprehensive wearable use in volleyball, emphasizing the need for further research.

Contributor 3 assessed the external training load of elite goalkeepers using wearables equipped with GPS and accelerometers. Findings revealed peak training load periods and similar loads between starting and non-starting goalkeepers, aiding in training optimization. This study showcased wearables' practical use in understanding training responses and enhancing performance while minimizing injury risks.

Contributor 4 investigated the effects of body segment exclusion or inclusion on the execution of a reverse punch using wearables. The study demonstrated that wearables allow for the study of specific movements, such as the reverse punch in martial arts, revealing insights into execution variations.

Contributor 5 aimed to understand the variation of kinematic and kinetic variables in swimming using a wearable system. The study showed that wearables in swimming can measure propulsive force and kinematic variables, offering insights into stroke mechanics and performance optimization.

Overall, the paper highlights that wearable technology offers an affordable and accurate means of obtaining real-time data in various sports contexts, complementing or even replacing traditional methods like video-motion analysis. It emphasizes the importance of sports scientists in interpreting wearable data for coaches and athletes to enhance performance and prevent injuries. It concludes by stating that while wearables are still primarily used for movement-based parameters, there is growing interest in understanding athletes' biophysical profiles using these technologies.

3.2 Using an ESP module and MPU6050 together

The section provides a brief overview of the study titled "Low-Cost Intelligent Child Safety Wearable IoT Device for India" by Khan et al. (2020)[4], focusing on the integration of an ESP module and MPU6050 for child safety. The device aims to address the growing concern of child safety by utilizing wearable IoT technology.

The authors integrate several components, including a GPS module, Node MCU microcontroller (ESP 8266), MPU-6050 gyroscope, and an alarm buzzer. These components work together to track the child's real-time location and send alerts to parents if the child moves outside predefined safe zones or enters unknown locations.

The study details the experimental setup and results, demonstrating the device's effectiveness in accurately tracking children's movements. By utilizing smartphone applications and cloud-based storage, the device ensures seamless communication between parents and children, thereby enhancing overall safety and security in potentially risky environments.

In summary, the paper contributes to the field by presenting a novel approach to child safety in India through the integration of ESP modules and MPU6050, leveraging IoT and wearable devices to address this pressing issue.

3.3 Decision Trees Used in Classification

In their paper titled "Classification Based on Decision Tree Algorithm for Machine Learning" [5], Charbuty and Abdulazeez delve into the versatile realm of decision tree algorithms and their applications in classification tasks within machine learning. Renowned for their simplicity and efficacy, decision tree classifiers find utility across diverse domains, from medical diagnosis to text analysis. This paper offers a comprehensive exploration of decision tree algorithms, encompassing various types, and discusses their advantages, limitations, and practical applications in the field of machine learning.

The authors compare these decision tree algorithms, shedding light on their respective strengths in handling large datasets and achieving high accuracy in classification tasks. However, decision tree algorithms have certain limitations, including susceptibility to overfitting and difficulty handling complex relationships in the data. Despite these limitations, decision trees remain a powerful tool in machine learning with a wide range of applications. Moreover, the paper provides a detailed survey of recent research endeavors in the domain. Notably, one recent study highlighted in the paper achieved an exceptional classification accuracy of 99.93 percent using a machine learning repository dataset. Furthermore, another recent research endeavor showcased in the paper employed decision tree algorithms, yielding remarkable results with an accuracy of 99.61 percent.

In essence, this paper serves as an invaluable resource for practitioners aiming to deepen their understanding of decision tree algorithms and harness their potential in a wide range of machine learning applications.

3.4 Collection of Accelerometer and Gyroscope Data

The paper "Horsing Around–A Dataset Comprising Horse Movement" by Jacob W. Kamminga et al. [6] presents a comprehensive dataset designed for studying Animal Activity Recognition (AAR) using motion data from horses and ponies. This dataset is significant due to the limited availability of large-scale datasets in this field, particularly those concentrating on activities involving horses and ponies.

The dataset includes 1.2 million 2-second data samples collected from 18 individual horses and ponies over a period of seven days. The data was gathered using sensor devices equipped with accelerometers, gyroscopes, and magnetometers.

The authors detail the sensor setup and data collection process, describing the positioning and configuration of the sensors on the animals to capture precise movement data. The data labeling procedure is also thoroughly explained, involving the segmentation of the data into meaningful activity categories such as walking, trotting, and grazing. The paper provides insights into the dataset structure and file organization, making it accessible and usable for researchers.

One of the key challenges highlighted is the accurate labeling of activities due to the complexity of animal movements. The authors discuss these challenges and the methods

employed to address them, including machine learning techniques and algorithms suitable for activity recognition. They emphasize the importance of precise data labeling for training and evaluating AAR classifiers, particularly supervised and unsupervised representation learning algorithms. The dataset is positioned as valuable for advancing AAR research, providing opportunities for developing and refining algorithms for recognizing and analyzing animal activities.

In conclusion, the dataset presented is a valuable resource for the AAR research community, providing a rich source of motion data that can be used to develop and refine algorithms for recognizing and analyzing animal activities.

3.5 Machine Learning-Based Activity Recognition

3.5.1 Human Activity Recognition using Accelerometer and Gyroscope Data

In their paper titled "Human Activity Recognition Using Accelerometer and Gyroscope Sensors" [7], Souza and Rajamohan delve into human activity recognition, focusing on the efficacy of accelerometer and gyroscope sensors. Their study entails a thorough analysis of classification accuracies using various machine-learning algorithms and sensor configurations.

Conducting experiments in two distinct settings, the authors evaluate classifier performance based on data collected from sensors positioned on the right lower arm and left ankle. Employing techniques such as K-Nearest Neighbors (KNN), Naïve Bayes (NB), SVM, Conditional Inference Tree (C-Tree), J48, and Random Forest (RF), they highlight significant fluctuations in accuracy across classifiers and sensor placements.

In the first setting, accuracies for the right lower arm range from 86.26 percent to 98.58 percent, and for the left ankle, they span from 81.01 percent to 98.10 percent. Similarly, in setting 2, accuracies vary between 85.31 percent and 93.36 percent for the right lower arm and between 75.83 percent and 90.05 percent for the left ankle.

A notable highlight of the paper is the RF classifier, which emerges as the most accurate for both accelerometer and gyroscope data.

In summary, this research offers valuable insights into human activity recognition through accelerometers and gyroscope sensors. It provides a comprehensive analysis of classification accuracies and the effectiveness of diverse machine learning algorithms, emphasizing the importance of sensor placement and the potential benefits of sensor combinations in enhancing recognition performance.

3.5.2 Classifying Human Activities Using Decision Trees

In their paper titled "Decision Trees for Human Activity Recognition in Smart House Environments" [8], Sanchez and Skeie explore the domain of human activity recognition

(HAR) within smart house environments, focusing on developing a model capable of discerning abnormal activities, particularly to aid individuals living alone. Grounded on the premise that individuals adhere to specific activity patterns in their daily routines, the authors utilize a decision tree classifier algorithm trained on an open-source database, conducting both training and testing phases using MATLAB.

The significance of HAR in augmenting the functionality of smart environments, particularly in supporting independent living among older adults, is underscored in the paper. Decision trees are chosen for their predictive capabilities, enabling the modeling of user behavior based on activity recognition data. Their results showcase a notable accuracy rate of 88.02% in activity detection, demonstrating the effectiveness of decision trees in this context.

Despite the promising nature of decision trees, the authors candidly acknowledge challenges, particularly in scenarios where activities occur within the same room, leading to prediction errors. In conclusion, while decision trees offer an effective approach to activity recognition, the authors advocate for further research to refine anomaly detection and mitigate prediction errors.

3.5.3 An Example of Activity Recognition in a Specific Sport

The paper "Activity Recognition of a Badminton Game Through Accelerometer and Gyroscope" by Md. Ariful Islam Anik et al. [9] addresses the challenge of recognizing various movements in a badminton game—such as serve, smash, backhand, forehand, and return—using motion sensors, specifically accelerometers, and gyroscopes.

The methodology involved several key steps. First, the authors defined specific badminton activities and gathered corresponding data using the MPU-6050 sensor. The data preprocessing step included cleaning and normalizing the data to prepare it for analysis. They then extracted relevant features from the raw sensor data, focusing on RMS values. For classification, they applied k-NN and SVM algorithms to classify the activities. This process included integrating the sensor with the racket, ensuring proper initial positioning, and using the selected features for activity recognition.

The study's results highlighted the effectiveness of different classifiers. Root mean square (RMS) values provided a basic means of distinguishing activities but resulted in similar values for different actions. The k-NN classification achieved an average accuracy of approximately 58 percent. The SVM classification showed a significantly higher accuracy of 88.89 percent, with better performance in recognizing smash and backhand activities compared to serving.

The authors concluded that their method holds promise for accurately recognizing badminton activities using accelerometer and gyroscope data. They suggested future improvements, including expanding the dataset size, enhancing accuracy with online classifiers like Dynamic Time Warping (DTW), and utilizing recognized activities to train AI-based applications for virtual reality games.

Chapter 4

Methodology

The Methodology chapter outlines the system components. It details the interconnections between these components and describes two modes of data recording: offline and online. Additionally, the chapter provides insights into the coding implementation for both modes and explores the PCB design process for the system. Furthermore, it outlines the methodology for machine learning analysis, focusing on decision trees for activity recognition.

4.1 System Components

4.1.1 ESP32-CAM

The ESP32-CAM is a robust microcontroller board that leverages the ESP32 chip to provide multiple functionalities, including Wi-Fi connectivity, Bluetooth, and data storage via an SD card. Serving as the cornerstone of the system, it enables wireless communication in online mode via Wi-Fi and offline data storage through the SD card. Its compact design and versatile features make it well-suited for applications requiring data collection through Wi-Fi or SD card, offering efficiency and flexibility to cater to diverse needs.

Figure 4.1: ESP32-CAM

4.1.2 Programming ESP32-CAM

The ESP32-CAM microcontroller cannot be programmed directly. Instead, we use the FT232RL FTDI module for programming. This module acts as a bridge between the ESP32-CAM and a PC or laptop, facilitating programming and communication.

Figure 4.2 illustrates the connections between the ESP32-CAM and the FT232RL FTDI module. Once connected, the FT232RL FTDI module is linked to the PC or laptop via a mini USB cable. This configuration enables us to program the ESP32-CAM microcontroller and perform necessary configurations.

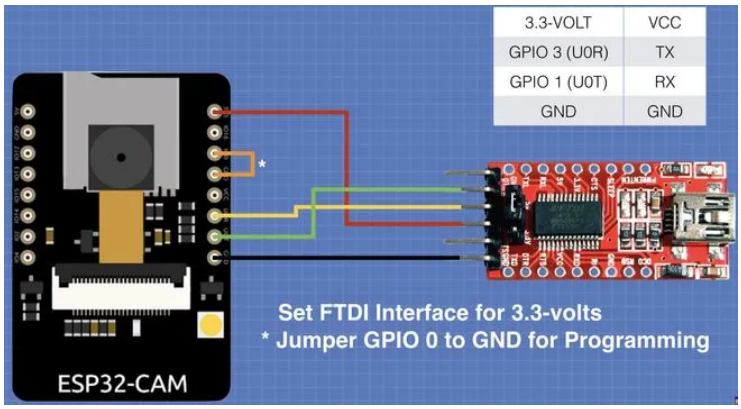


Figure 4.2: ESP32-CAM connection with FT232RL FTDI module.

4.1.3 MPU6050 Sensor

The MPU6050 is a widely used sensor module featuring a 3-axis accelerometer and a 3-axis gyroscope. It provides precise measurements of acceleration, rotation, and orientation, essential for motion analysis and gesture recognition. The MPU6050's small size, low power consumption, and high accuracy make it well-suited for our system.

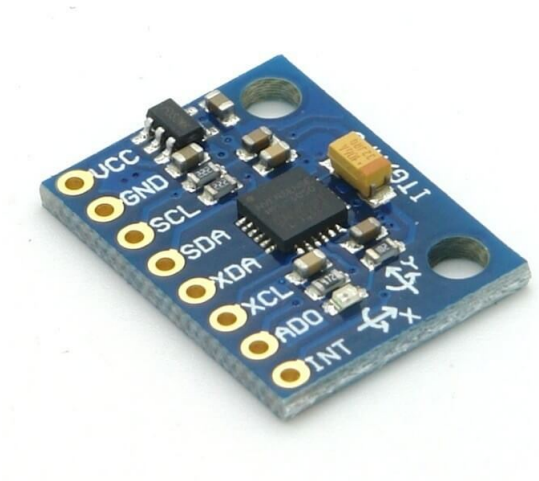


Figure 4.3: MPU6050

4.1.4 Touch Sensor TTP223

The TTP223 is a touch-sensitive switch module that detects human touch on its surface. It offers a simple and reliable way to trigger actions in our system, such as starting or stopping data recording. With its compact size and low power consumption, the TTP223 enhances user interaction and control without requiring complex input devices.

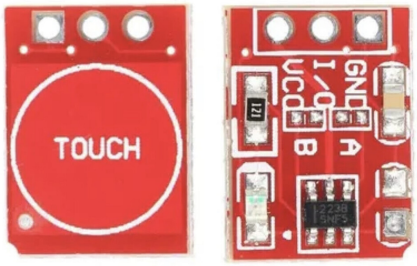
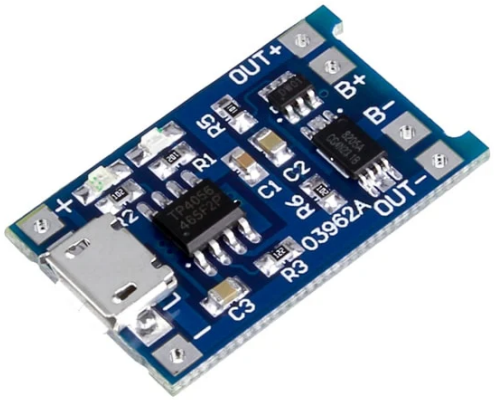


Figure 4.4: TTP223

4.1.5 TP4056 Charger and LiPo Battery

The TP4056 is a specialized lithium-ion battery charging module designed to efficiently and safely charge single-cell rechargeable LiPo batteries. When coupled with a high-capacity LiPo battery, it serves as a dependable power source for our system, guaranteeing uninterrupted operation during data collection sessions. Featuring built-in protection mechanisms such as overcharge and over-discharge protection, the TP4056 module actively safeguards the battery, thereby extending its lifespan and enhancing reliability.

(a) LiPo Battery.



(b) TP4056 Charger.

Figure 4.5: Images of a LiPo battery and TP4056 charger.

4.2 Connections

The components in our system are interconnected as follows:

- The ESP32-CAM is connected to the MPU6050 sensor as follows:

  GPIO14 of the ESP32 is connected to the SDA pin of the MPU6050, and GPIO13 of the ESP32 is connected to the SCL pin of the MPU6050. This enables communication between the ESP32 and the MPU6050 for data acquisition.

- The ESP32-CAM is also connected to the touch sensor TTP223 via GPIO12. This connection allows the ESP32 to detect touch inputs from the TTP223 sensor, enabling user interaction with the system.

- All components, including the ESP32-CAM, MPU6050 sensor, and touch sensor TTP223, are powered by the LiPo battery through the TP4056 charger. The positive terminal of the LiPo battery is connected to the input terminal of the TP4056 charger. The output terminal of the TP4056 charger is then connected to the 5V pin of the ESP32-CAM and the VCC pins of the MPU6050 sensor and the touch sensor TTP223. This setup ensures that all components receive power from the battery and can also be charged via the TP4056 charger.

- The ground (GND) pins of all components are connected together via the TP4056 charger, ensuring a common ground reference for the entire system and completing the electrical circuit.

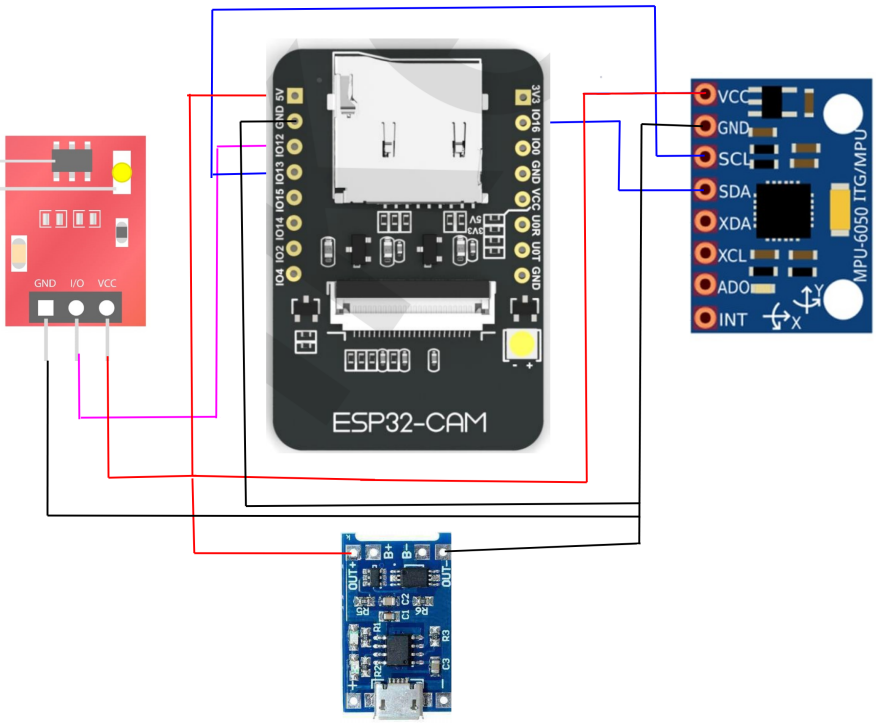


Figure 4.6: System connection

4.3 Data Recording Modes

Our system is able to collect data in two modes: offline and online.

4.3.1 Offline SD card Mode

In offline mode, the system functions autonomously without requiring an active internet connection. Data collection and storage occur locally on the device itself using an SD card. The process in offline mode unfolds as follows:

4.3.1.1 Sensor Data Recording

When triggered by the touch sensor, the system initiates the recording of sensor data from the MPU6050 accelerometer and gyroscope sensor. The recorded sensor data is then saved onto the SD card in a CSV file. This file serves as a repository for the collected data, allowing for subsequent retrieval and analysis.

4.3.1.2 Autonomous Operation

With all data processing and storage handled locally, the system operates independently of external dependencies. This makes offline mode suitable for scenarios where continuous data logging and real-time transmission are not prerequisites, such as standalone data collection in remote locations or environments with limited connectivity.

4.3.2 Online Data Transmission over WiFi

In contrast, online mode enables real-time data transmission and remote access to the collected sensor data. It relies on an active internet connection. In our system, online mode operates over a local network, ensuring data transmission within a confined environment. The workflow in online mode encompasses the following steps:

4.3.2.1 Network Connectivity

Initially, the system establishes a local network connection via Wi-Fi, to enable communication within the network. This connectivity allows the system to interact with other devices and access local resources.

4.3.2.2 Data Transmission Protocol

When the touch sensor is activated, indicating the start of data collection, the system switches its communication protocol to send data to a remote server. Utilizing the TCP/IP suite, the system packages the sensor data into data packets suitable for transmission over the local network.

4.3.2.3 Remote Data Storage

Upon establishing a connection to the server on the local network, the system sends the sensor data to the designated endpoint for storage and further processing. The server receives and stores the incoming data in a CSV file.

Our system offers both offline and online modes of data collection to accommodate different use cases and scenarios. This versatility provides flexibility and adaptability to varying requirements, allowing users to leverage the system's capabilities according to their specific needs and objectives.

4.4 Data Collection Coding

In this section, we provide an overview of the coding involved in both offline and online modes of data collection.

4.4.1 Programming Languages Utilized

4.4.1.1 Offline Programming (ESP32-CAM)

For offline programming of the ESP32-CAM module, we used the Arduino IDE, which is based on a simplified version of C++. This platform offers a convenient way to write, compile, and upload code to the microcontroller. With its user-friendly interface and comprehensive libraries, the Arduino IDE streamlines the development process, aligning well with our requirements.

4.4.1.2 Online Programming (ESP32-CAM and Server)

For online programming, we employed both Arduino(C++) and Python programming languages. Arduino was used for programming the ESP32-CAM board, enabling us to implement sensor data collection and transmission functionalities. Python was utilized for programming the server responsible for receiving data from the ESP32-CAM board, processing it, and storing it in a CSV file. Python's versatility, ease of use, and extensive libraries made it an excellent choice for implementing network communication and data processing functionalities. We utilized the Spyder IDE for Python development, which offers features like code highlighting, debugging, and variable exploration, enhancing the development workflow and productivity.

4.4.2 Libraries Used

Arduino and Spyder libraries play a crucial role in facilitating hardware interfacing and data collection, respectively.

4.4.2.1 Arduino Libraries

Arduino libraries provide a convenient way to interface with hardware components and peripherals in Arduino-based projects. The `WiFi.h` library allows Arduino devices to connect to wireless networks, enabling communication with other devices or the internet. The `Wire.h` library provides support for I2C communication protocol, allowing our microcontroller to communicate with the sensor. The `MPU6050.h` library specifically interfaces with the MPU6050 accelerometer and gyroscope sensor, providing functions to read sensor data and configure sensor settings. Additionally, the `SD_MMC.h` library facilitates communication with SD memory cards, enabling data logging and storage on external storage devices.

4.4.2.2 Spyder Libraries

Spyder libraries are commonly used in Python-based data analysis and scientific computing projects. The `socket` library provides a low-level interface for networking operations, allowing Python scripts to create network connections, send and receive data packets, and communicate with other devices over a network. The `csv` library is used for reading and writing CSV files, a common format for storing tabular data. It provides functions to parse CSV files, extract data, and perform data manipulation.

4.4.3 Offline Mode Coding

The provided code snippets demonstrate the implementation of the offline mode functionality, where sensor data is stored locally on an SD card. The code is designed to run on an ESP32-CAM microcontroller board, utilizing the MPU6050 sensor for data acquisition and an SD card module for data storage.

4.4.3.1 Initialization

This snippet of code in figure 4.7 initializes the necessary libraries and variables for the operation of the system. It includes the inclusion of required libraries such as Wire.h, SD_MMC.h, and MPU6050.h. Additionally, it defines the GPIO pins for the I2C communication with the MPU6050 sensor (`SDA_PIN` and `SCL_PIN`), with `SDA_PIN` being GPIO16 and `SCL_PIN` being GPIO13, and the GPIO pin for the touch sensor (`TOUCH_PIN`), which is GPIO12. The state variable `x` is used to control the system's behavior, while `lastTouchTime` keeps track of the time of the last touch event for the touch sensor.

```
#include <Wire.h>
#include <SD_MMC.h>
#include <MPU6050.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
MPU6050 mpu;
#define SDA_PIN 16 // GPIO14 for SDA
#define SCL_PIN 13 // GPIO13 for SCL
#define TOUCH_PIN 12 // GPIO pin for touch sensor
File dataFile;
int x = 0; // State variable
unsigned long lastTouchTime = -1000;
```

Figure 4.7: Initialization Arduino code for offline mode

4.4.3.2 Setup

The setup function in figure 4.8 performs several essential tasks to initialize the system. It begins by establishing serial communication, configuring the I2C interface for communication with the MPU6050 sensor, and initializing the SD card module. To conserve GPIO pins required for other functions, the SD card module is set to operate in one-bit mode[10]. Additionally, the setup function prepares the touch sensor pin as an input, enabling it to detect touch events effectively. This comprehensive setup process ensures that the system is properly configured and ready to execute its intended functionalities seamlessly.

```
void setup() {
  Serial.begin(115200);
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Wire.begin(SDA_PIN, SCL_PIN); // Initialize SDA and SCL pins
  mpu.initialize();// Initialize MPU
  if (!SD_MMC.begin("/sdcard", true)) {
    Serial.println("SD Card initialization failed!");
    return;}
  Serial.println("SD Card initialized successfully!");
  pinMode(TOUCH_PIN, INPUT);// Set up touch pin
  }
```

Figure 4.8: Setup Arduino code for offline mode

4.4.3.3 Data Recording

This code segment in figure 4.10 controls the process of data recording from the ESP32-CAM board to an SD card in offline mode. The loop() function continuously monitors the state of the touch sensor and manages the data recording flow based on touch events. It employs a state machine with three primary states: idle, recording, and file closed (figure 4.9).

In the idle state (case 0), the code waits for the initial touch event. Upon detecting the touch, the code starts recording by opening a CSV file on the SD card. It then writes a header containing the data fields to the CSV file and transitions to the recording state. During the recording state (case 1), the code checks for another touch of the touch sensor after a specified duration to prevent successive touches by mistake. If a touch event occurs during the recording session, it signifies the end of the current recording session, the code closes the CSV file and transitions to the file closed state. Otherwise, it continues to read accelerometer and gyroscope data from the MPU sensor and writes this data to the CSV file. In the file closed state (case 2), the system awaits another touch event to initiate a reset. Upon detecting this touch event, the ESP32-CAM undergoes a restart. This restart is essential because when the SD card is removed for accessing recorded data, the microcontroller needs to be rebooted to properly initialize the SD card again.

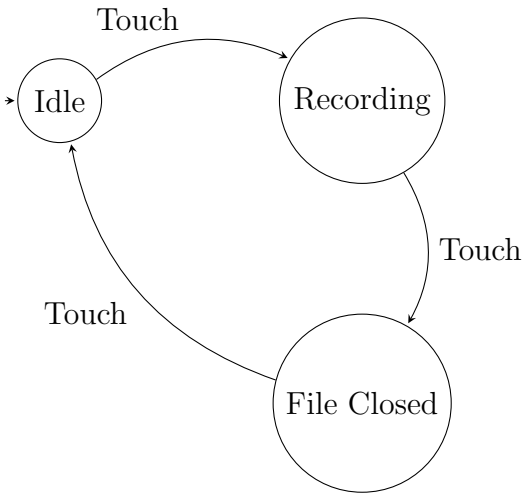


Figure 4.9: Finite State Machine (FSM) for System Operation.

```
void loop() { int touchState = digitalRead(TOUCH_PIN);unsigned long currentTime = millis();
 switch (x) {  case 0: // Idle state
if (touchState == HIGH && (currentTime - lastTouchTime) >= 1000) {
  Serial.println("Touch detected, starting recording.");   dataFile = SD_MMC.open("/data.csv", FILE_WRITE);
  if (!dataFile) {Serial.println("Error opening data.csv for writing!");return;}
  dataFile.println("Time,Accel X,Accel Y,Accel Z,Gyro X,Gyro Y,Gyro Z");
  x++; // Move to the next state
   lastTouchTime = currentTime;}  break;
                    case 1: // Recording state
if (touchState == HIGH && (currentTime - lastTouchTime) >= 2000) {
        Serial.println("Touched again, stopping recording.");
        dataFile.close(); // Close the file
        x++; // Move to the next state
         lastTouchTime = currentTime;
      } else {
int16_t ax, ay, az;int16_t gx, gy, gz;
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);unsigned long currentTime = millis();
dataFile.print(currentTime); dataFile.print(",");dataFile.print(ax);
 dataFile.print(",");dataFile.print(ay); dataFile.print(",");dataFile.print(az);
dataFile.print(",");dataFile.print(gx); dataFile.print(",");
dataFile.print(gy); dataFile.print(",");dataFile.println(gz);}
break;
          case 2: // File closed state
     if (touchState == HIGH && (currentTime - lastTouchTime) >= 1000) {
 Serial.println("Touch detected again, resetting.");  x = 0; // Reset state variable
  lastTouchTime = currentTime;delay(500); ESP.restart();}   break;}
  delay(10); }
```

Figure 4.10: Data record Arduino code for offline mode

4.4.4 Online Mode Coding

In this subsection, we demonstrate the Arduino code for the ESP32-CAM to transmit data and the Python code for the server to receive it.

4.4.4.1 ESP32-CAM Arduino Code

Initialization

This snippet of code in figure 4.11 initializes the ESP32-CAM microcontroller board, sets up the Wi-Fi connection, prepares the system for data transmission over TCP/IP protocol, and defines the GPIO pins used for various purposes in the ESP32-CAM board.

```
#include <WiFi.h>
#include <Wire.h>
#include <MPU6050.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
MPU6050 mpu;
const char* ssid = "ab";
const char* password = "09876542";
const char* serverIPAddress = "192.168.130.83";
const uint16_t serverPort = 12345;
#define SDA_PIN 16 // GPIO14 for SDA
#define SCL_PIN 13 // GPIO13 for SCL
#define TOUCH_PIN 12 // GPIO pin for touch sensor
WiFiClient client;
int x = 0; // State variable
unsigned long lastTouchTime = 0;
```

Figure 4.11: Initialization Arduino code for online mode

Setup Function

The setup function in figure 4.12 initializes the serial communication, configures the I2C interface, and connects to the Wi-Fi network. It also sets up the touch sensor pin as an input.

```
void setup() {
  Serial.begin(115200);
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Wire.begin(SDA_PIN, SCL_PIN); // Initialize SDA and SCL pins
  mpu.initialize();
  WiFi.begin(ssid, password); // Connect to Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);Serial.println("Connecting to WiFi...");}
  Serial.println("Connected to WiFi");
  pinMode(TOUCH_PIN, INPUT); // Set up touch pin
  }
```

Figure 4.12: Setup Arduino code for online mode

Data Transmission

This code segment in figure 4.13 demonstrates the data transmission process from the ESP32-CAM board to the server using the TCP/IP protocol. The loop() function continuously monitors the touch sensor's state and regulates the data transmission flow based on touch events. It employs a state machine with three distinct states: idle, connected, and reset, similar to offline mode.

In the idle state (case 0), the code awaits the initial touch event. Upon detection of the touch, it proceeds to establish a TCP connection with the server. If the connection is successful, the code transitions to the connected state. During the connected state (case 1), the code monitors for another touch event, signaling the end of data transmission. Upon detecting a touch event, the code moves to the reset state. If no touch event occurs, it continues to read sensor data and sends it to the server via the TCP connection. In the reset state (case 2), the code awaits another touch event to reset the system. Upon detection of the touch event, it initiates a restart of the ESP32-CAM. Figure 4.9 illustrates the state machine.

```
void loop() {
  int touchState = digitalRead(TOUCH_PIN);unsigned long currentTime = millis();
    switch (x) {
      case 0: // Idle state
      if (touchState == HIGH && (currentTime - lastTouchTime) >= 1000) {
        Serial.println("Touch detected, setting up TCP.");
        if (client.connect(serverIPAddress, serverPort)) {
  Serial.println("TCP setup successful.");x++; // Move to the next state
  lastTouchTime = currentTime; }
  else {Serial.println("TCP setup failed!");} }
      break;
     case 1: // Recording state
      if (touchState == HIGH && (currentTime - lastTouchTime) >= 2000) {
Serial.println("Touch released, stopping recording.");x++; // Move to the next state
        lastTouchTime = currentTime;}
        else {
  int16_t ax, ay, az; int16_t gx, gy, gz;
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
String data = String(currentTime) + "," + String(ax) + "," + String(ay) + ","
+ String(az) + ","+ String(gx) + "," + String(gy) + "," + String(gz);
        if (client.connected()) {
          client.println(data);   Serial.println("Data sent to server: " + data); }
      else {Serial.println("Connection to server lost!");} } break;

    case 2: // File closed state
      if (touchState == HIGH && (currentTime - lastTouchTime) >= 1000) {
  Serial.println("Touch detected again, resetting."); x = 0; // Reset state variable
  lastTouchTime = currentTime;client.stop(); // Close TCP connection
        delay(500);ESP.restart();}  break; }
  delay(50); }
```

Figure 4.13: Data record Arduino code for online mode

4.4.4.2 Server Python Code

Initialization

The code snippet in Figure 4.14 initializes the server, responsible for receiving data from the ESP32-CAM board via the TCP/IP protocol and storing it in a CSV file. The host's IP address is set to 0.0.0.0 to listen on all available interfaces, and a specific port number is defined to ensure exclusivity.

```
1  import socket
2  import csv
3
4  # Define server parameters
5  HOST = '0.0.0.0'  # Listen on all available interfaces
6  PORT = 12345       # Use the same port as in your ESP32 code
7
8  # Specify the path to the CSV file
9  CSV_FILE_PATH = r"C:\Users\Dell\Desktop\sensor_data.csv"
```

Figure 4.14: Initialization python code for online mode server

Creating a Socket, Receiving, and Storing Data

The snippet of code in figure 4.15 describes the setup of a TCP/IP server. It begins by creating a socket and binding it to a specified host and port. Once the server is set up, it listens for incoming connections. When a client connects, the server accepts the connection and creates a new CSV file to store the received data. The server then enters a loop to continuously receive and process data from the client. As long as the client remains connected and sends data, the server continues to receive and store it in the CSV file. However, if the client disconnects, the server closes the connection and the file.

```python
# Create a TCP/IP socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind((HOST, PORT))# Bind the socket to the address and port
    server_socket.listen() # Listen for incoming connections
    print(f"Server listening on {HOST}:{PORT}")
    client_socket, client_address = server_socket.accept()#Accept a client connection
    print(f"Connection from {client_address}")
    # Create a CSV file to store the received data
    with open(CSV_FILE_PATH, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(['Timestamp', 'Ax', 'Ay', 'Az', 'Gx', 'Gy', 'Gz'])  # Write header
        while True:
            # Receive data from the client
            data = client_socket.recv(1024).decode().strip()
            if not data:
                print("Client disconnected.")
                break
            entries = data.split('\r\n')# Split the received data into individual entries
            for entry in entries:
                if entry:  # Ensure entry is not empty
                    # Split the entry into values
                    values = entry.split(',')
                    if len(values) == 7:  # Ensure the entry contains all expected values
                        try:
                            timestamp, ax, ay, az, gx, gy, gz = map(int, values)
                            csv_writer.writerow([timestamp, ax, ay, az, gx, gy, gz])
                            print("Data received and stored:", timestamp, ax, ay, az, gx, gy, gz)
                        except ValueError:
                            print("Invalid entry:", entry)
                    else:
                        print("Invalid entry:", entry)
print("Server closed.")
```

Figure 4.15: Creating a Socket, Receiving, and Storing Data python code for online mode server

4.5 PCB Design

After confirming the data collection capability of our system, this section focuses on the PCB design and manufacturing process, crucial for integrating and interconnecting electronic components. Here, we delve into the design considerations, PCB design process, and the tools utilized to create the system's PCB.

4.5.1 Software Utilized

A range of software options exists for PCB design, each with its own distinct functionalities and advantages. However, we chose Autodesk Fusion 360[11].

4.5.2 Design Considerations

Several factors must be considered during the PCB design process to ensure optimal performance, reliability, and manufacturability. These considerations include:

- Component Placement: Strategically positioning components on the PCB to minimize route interference, optimize route paths, and facilitate ease of assembly and maintenance.
- Signal Routing: Carefully routing traces to maintain signal integrity, and avoid cross-talks.
- Power Distribution: Designing an efficient power distribution network to ensure stable voltage and prevent power-related issues.
- Thermal Management: Incorporating thermal vias and proper component placement to dissipate heat effectively and prevent overheating of sensitive components.
- Manufacturing Constraints: Considering fabrication and assembly constraints such as minimum trace width and minimum clearance to facilitate PCB fabrication and assembly processes.

4.5.3 PCB Design Process

The PCB design process involves the following steps:

1. Schematic Capture: Creating a schematic diagram of the circuit layout. This involves selecting the system components and connecting them with nets.

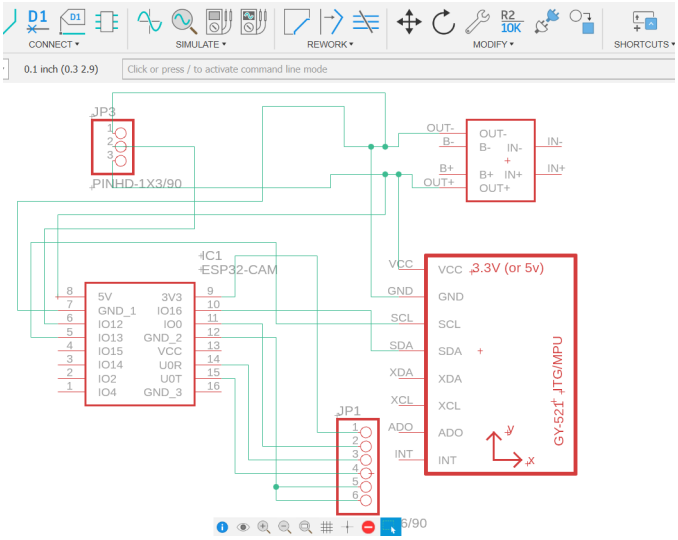


Figure 4.16: Schematic Diagram

2. Component Placement: Placing components on the PCB layout based on the schematic diagram. Considerations such as signal flow and thermal management, are taken into account during placement.

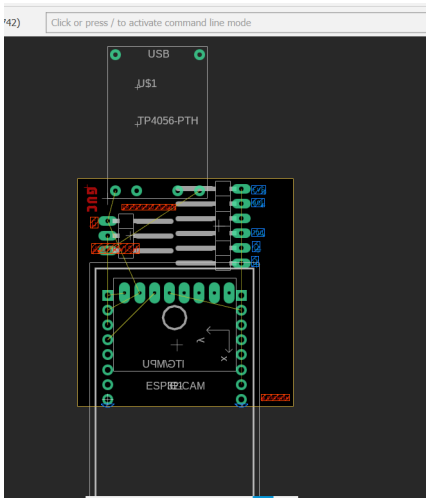


Figure 4.17: Component Placement

3. Routing: Routing traces to establish electrical connections between components while adhering to design rules and constraints.

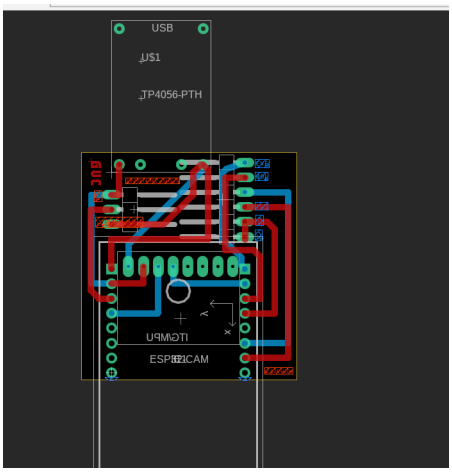


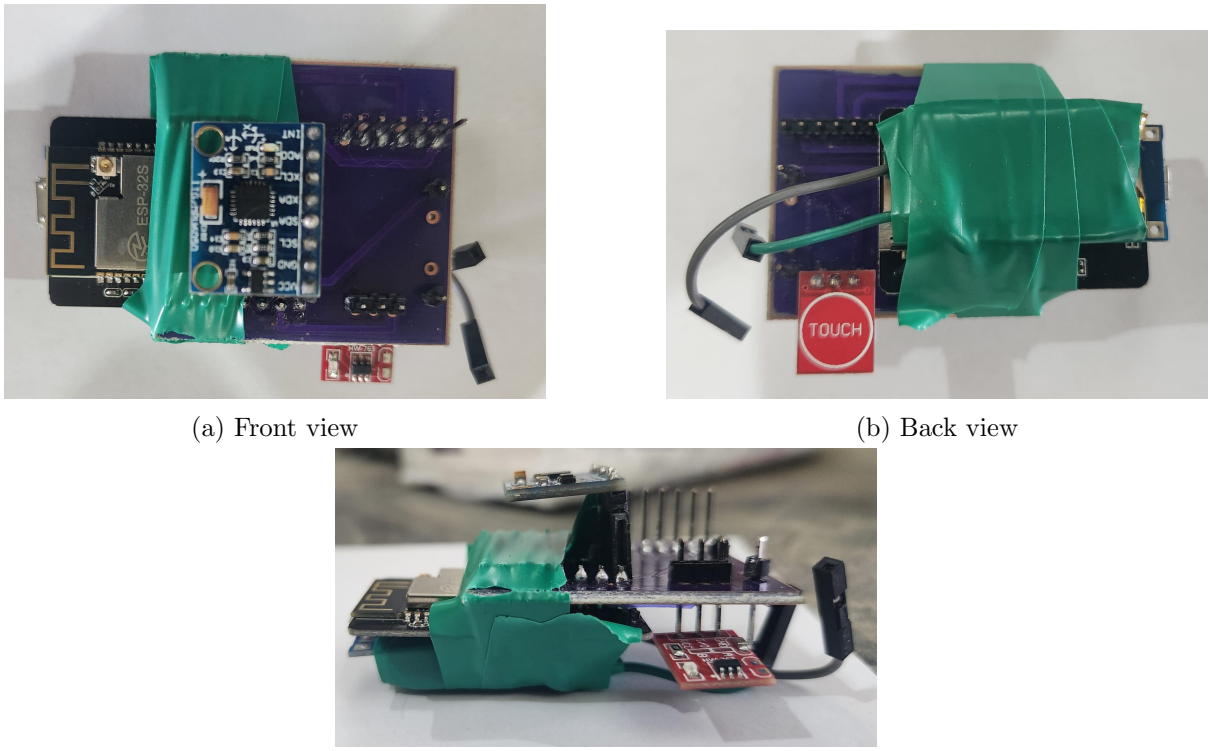
Figure 4.18: Routing

4. Gerber File Generation: Generating Gerber files, which contain the manufacturing data required for PCB fabrication..

5. Prototyping and Testing: Fabricating a prototype PCB and conducting thorough testing to verify functionality, performance, and compliance with design requirements.

(a) PCB front view. (b) PCB back view.

Figure 4.19: The PCB after it was manufactured.



(a) Front view (b) Back view

(c) Side view

Figure 4.20: The PCB after components were soldered

4.6 Data Collection Attempts

After our system was ready, we tested its functionalities by recording data for several activities. We recorded some human activities including sitting (*sit*), standing (*std*), walking (*wlk*), going upstairs (*ups*), and going downstairs (*dws*). This data was recorded while holding the sensor in hand. Additionally, we recorded data using a tennis racket for forehand, backhand, and serve actions. The sensor was fixed in the space between the racket handle and the net. We plotted a sample of the data with time and also plotted the average window with a size of 20 to be able to visualize the data and make comparisons. The recording frequency was every 12 milliseconds.

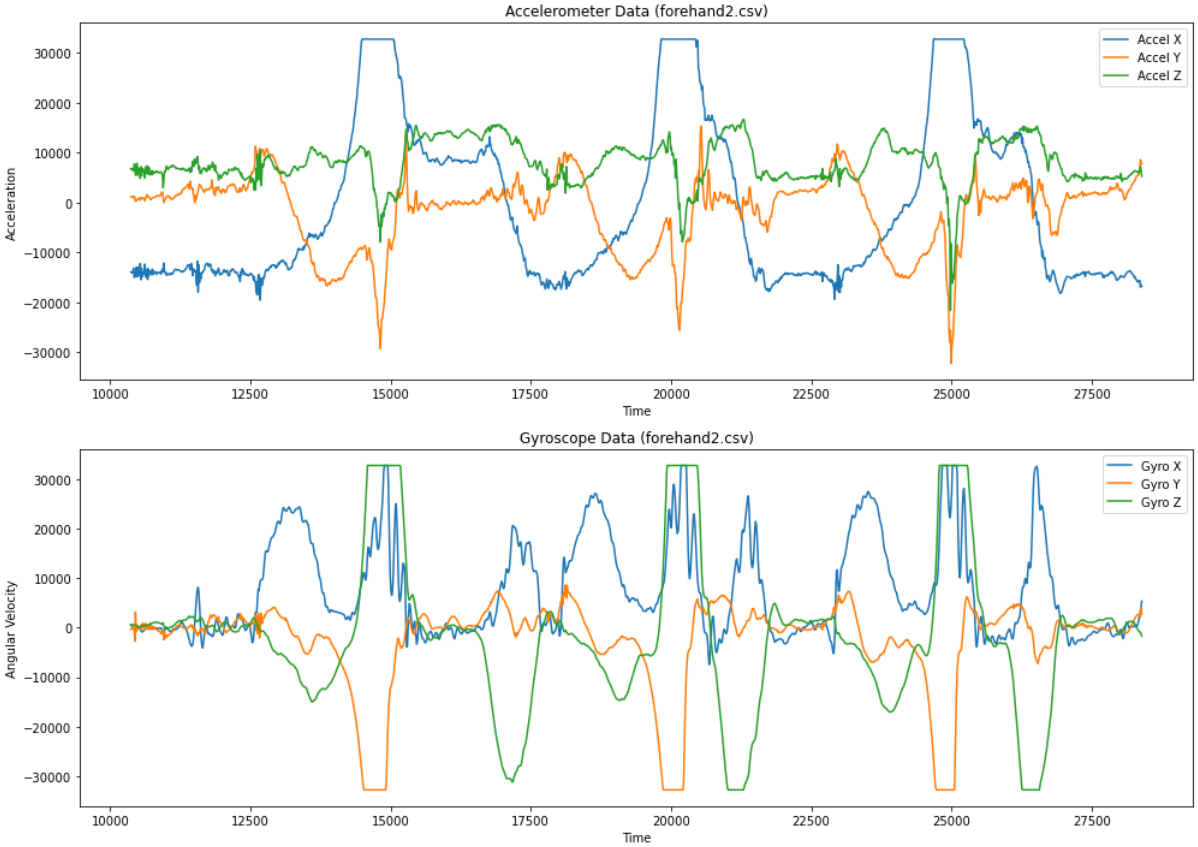
4.6.1 Tennis Data Collection and Visualization

We collected data for three tennis actions: forehand, backhand, and serve. For every action, we recorded 20 hits with no motion interval between each hit to distinguish them. Then we plotted a sample of data with time and the average window for the three actions in Figures 4.21, 4.22, and 4.23.

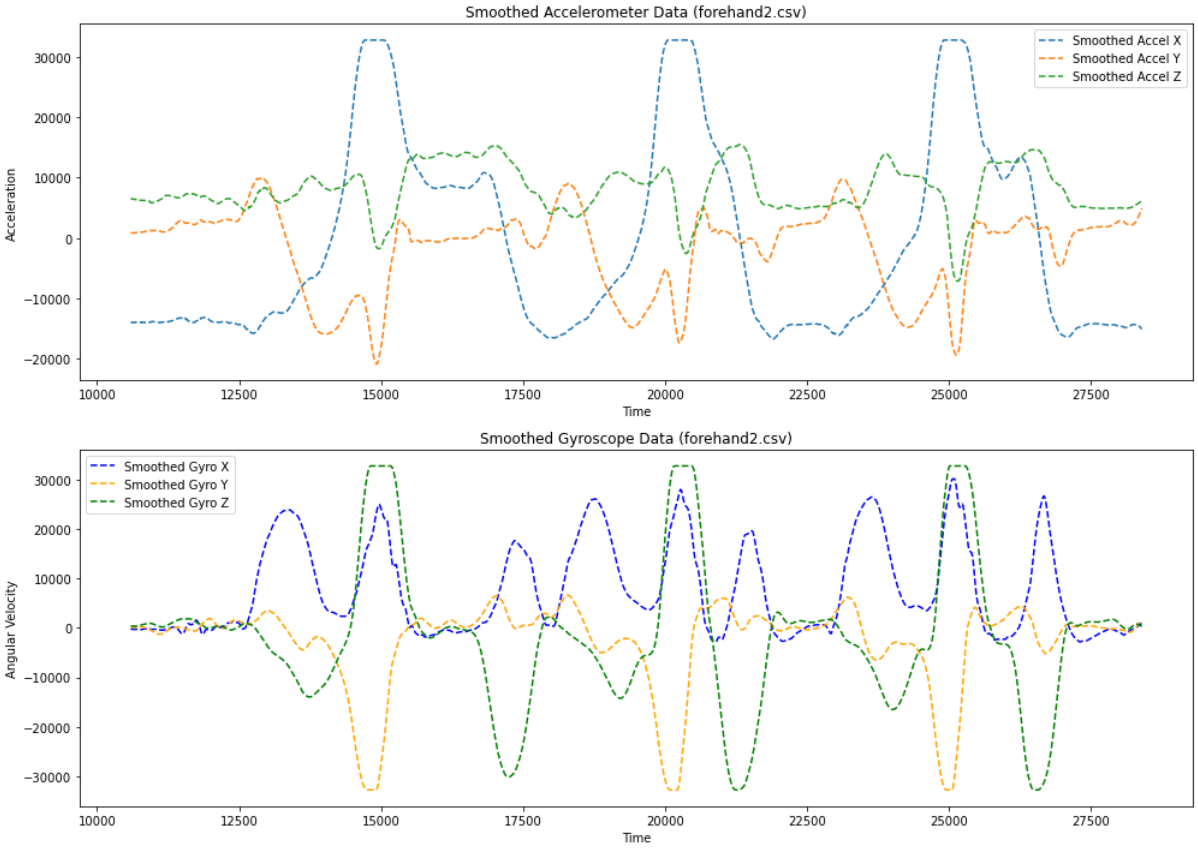
The following observations can be made:

- **Accelerometer Data:**
  - **Forehand:** Periodic peaks indicating the forehand strokes.
  - **Backhand:** Different peak patterns compared to the forehand, reflecting the mechanics of the backhand stroke.
  - **Serve:** High peaks, especially in the Z-axis, indicating the serve motion.
- **Gyroscope Data:**
  - **Forehand:** Significant changes in angular velocity, particularly in the X-axis.
  - **Backhand:** Different angular velocity patterns from the forehand, highlighting the different rotational movements.
  - **Serve:** High angular velocities, particularly in the Z-axis, corresponding to the overhead rotational motion.

From these figures, we see that the accelerometer data clearly show the shots, indicated by peaks on the X-axis, with intervals of no action between them. Since all actions are quite similar, it's challenging to distinguish them, but there are subtle differences. The main difference between backhand and forehand is in the Gyroscope X-axis, which is logical because they are the same actions but in different directions. The difference between backhand and serve is the acceleration in the Y-axis, and the difference between serve and forehand is also in the acceleration in the Y-axis.

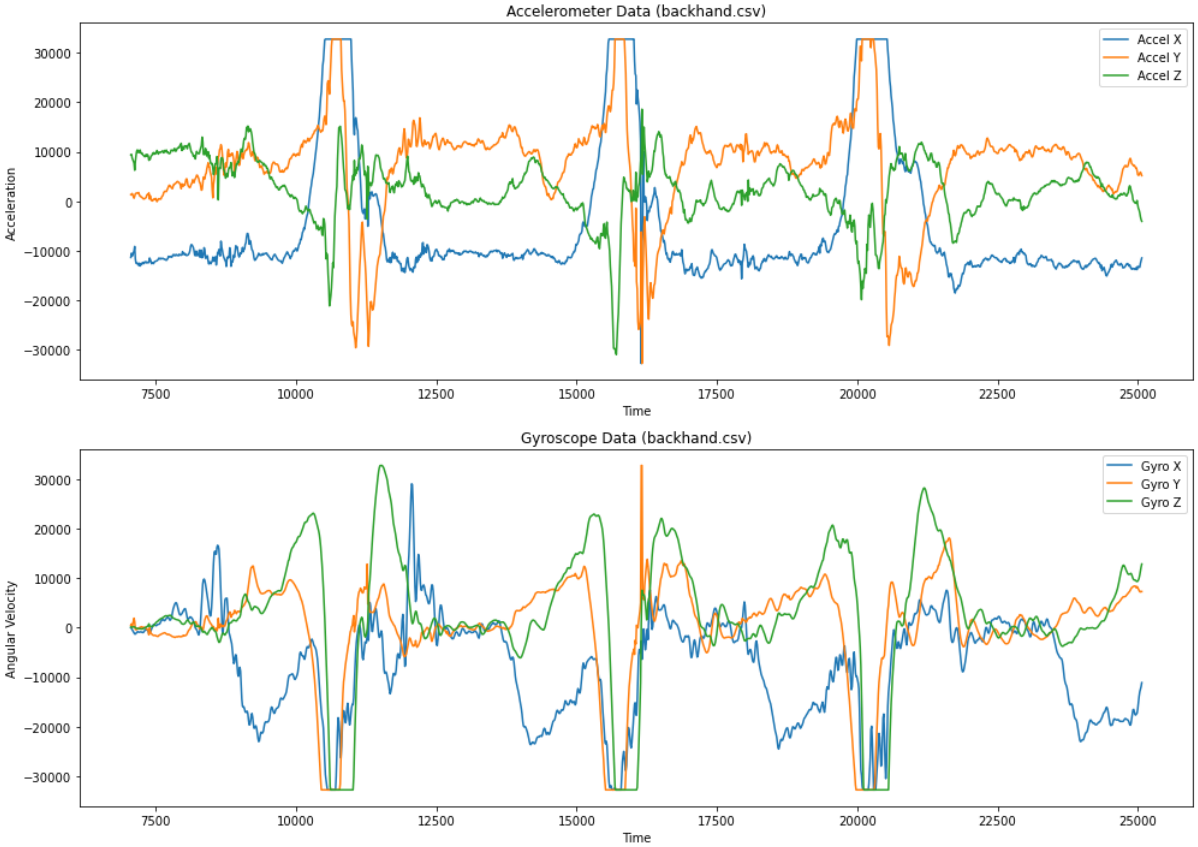


(a) Real data plot with time for forehand.

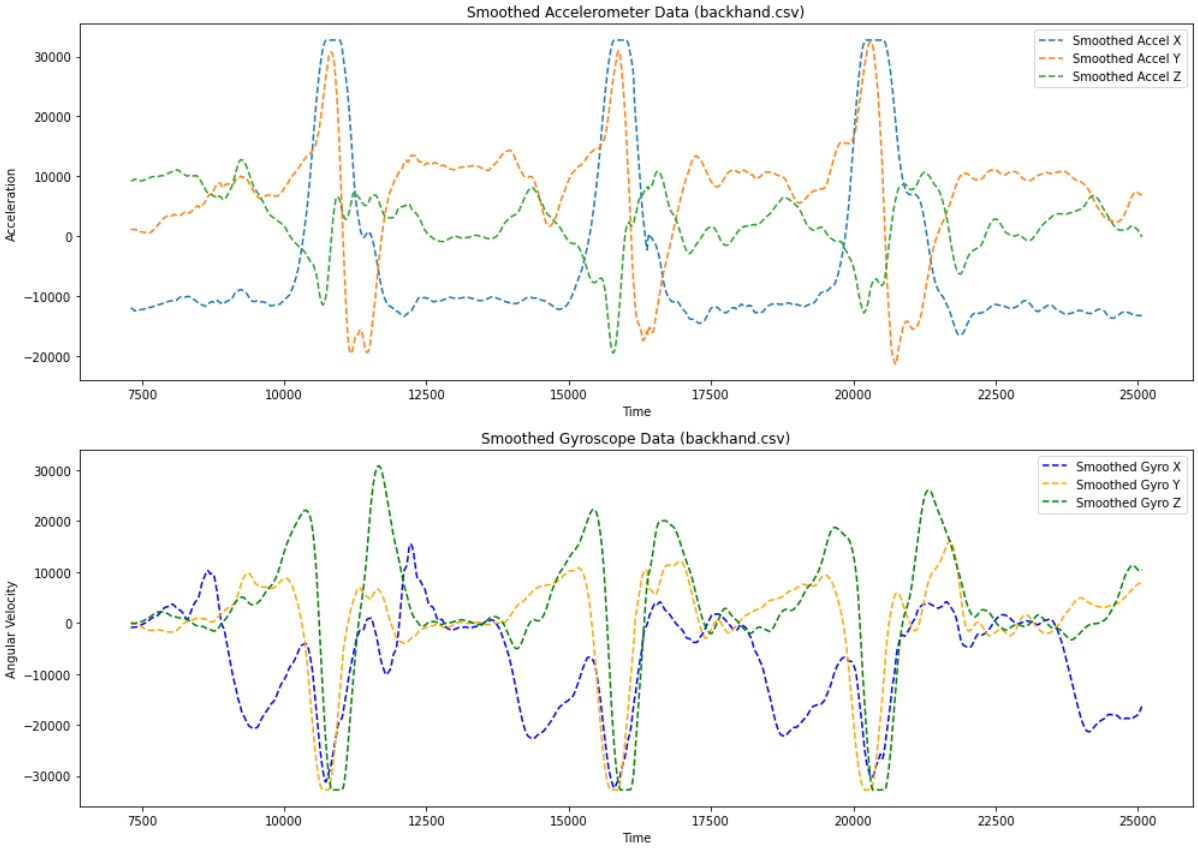


(b) Average window plot with time for forehand.

Figure 4.21: Forehand data visualization.

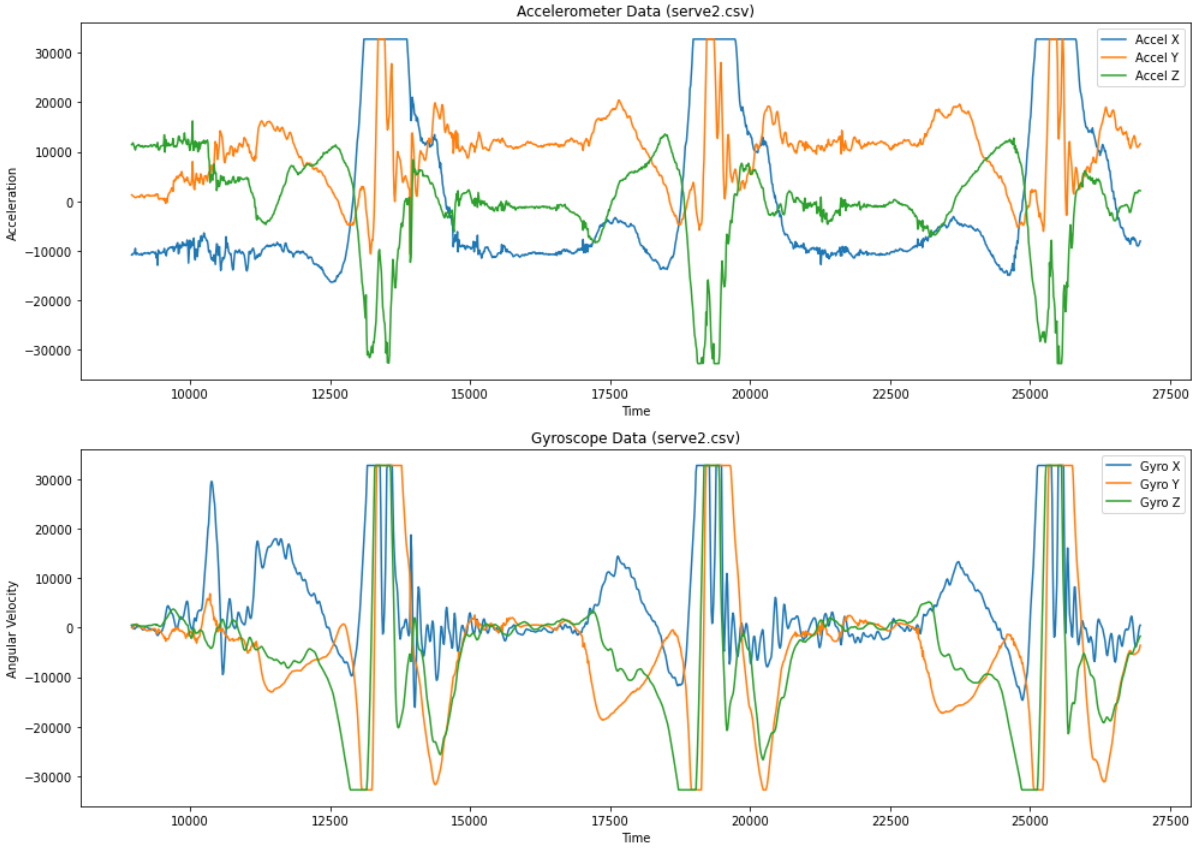


(a) Real data plot with time for backhand.

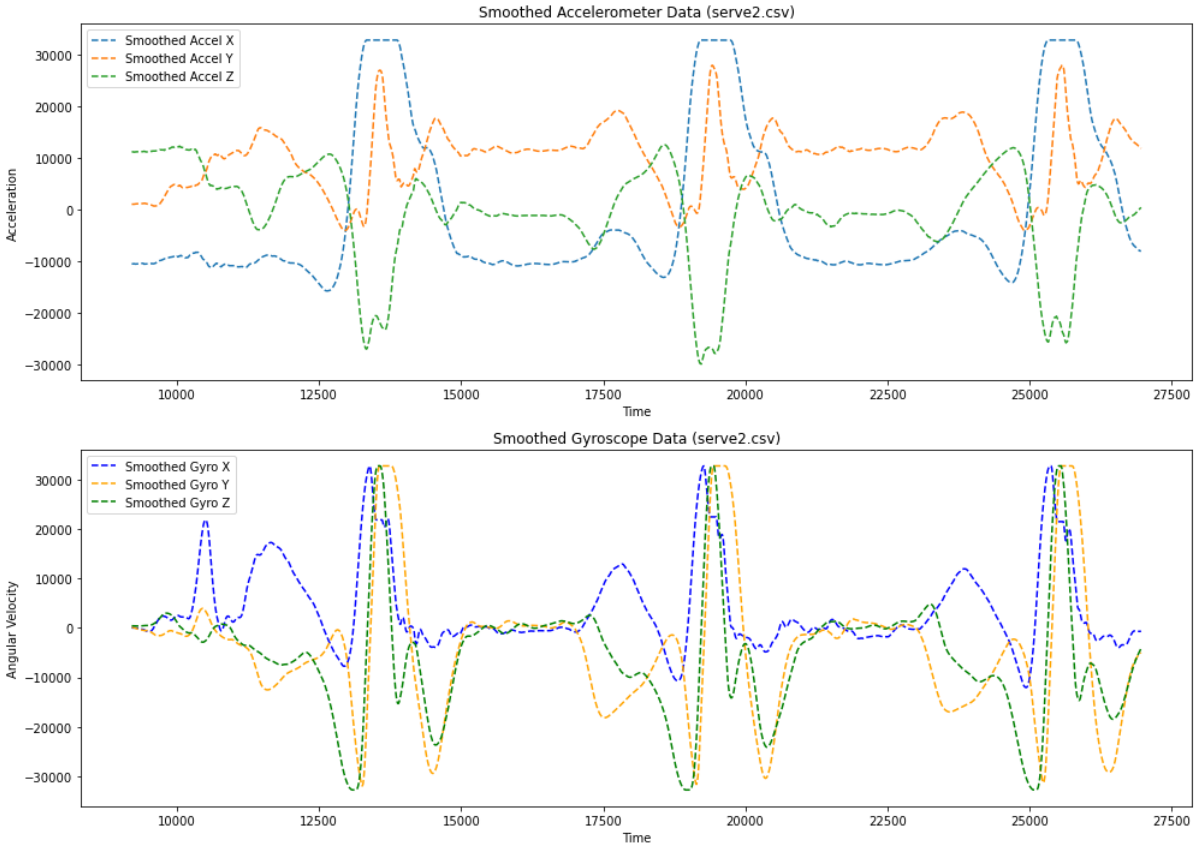


(b) Average window plot with time for backhand.

Figure 4.22: Backhand data visualization.



(a) Real data plot with time for serve.



(b) Average window plot with time for serve.

Figure 4.23: Serve data visualization.

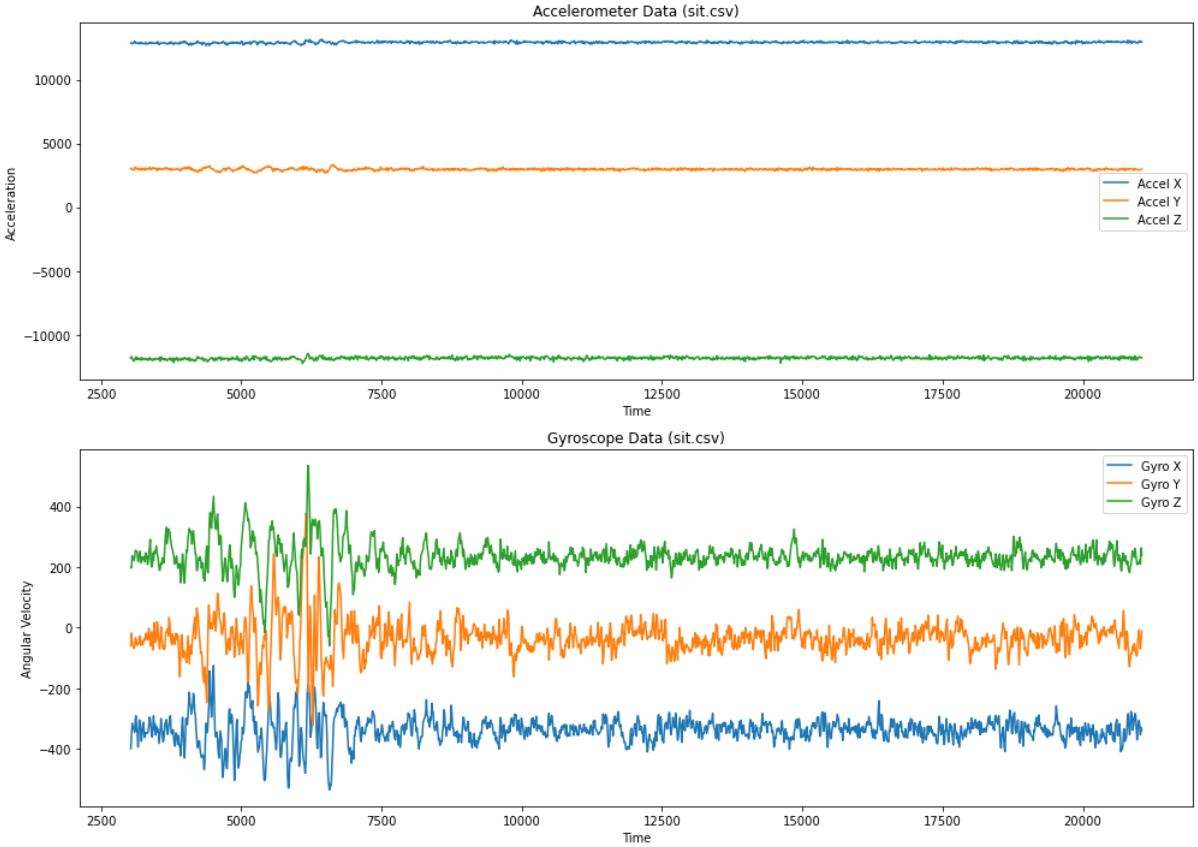
4.6.2 Sit, Stand, and Walk Data Collection and Visualization

We recorded 15 minutes each of sitting, standing, and walking.Then we plotted a sample of data with time and the average window for the three actions in Figures 4.24, 4.25, and 4.26.

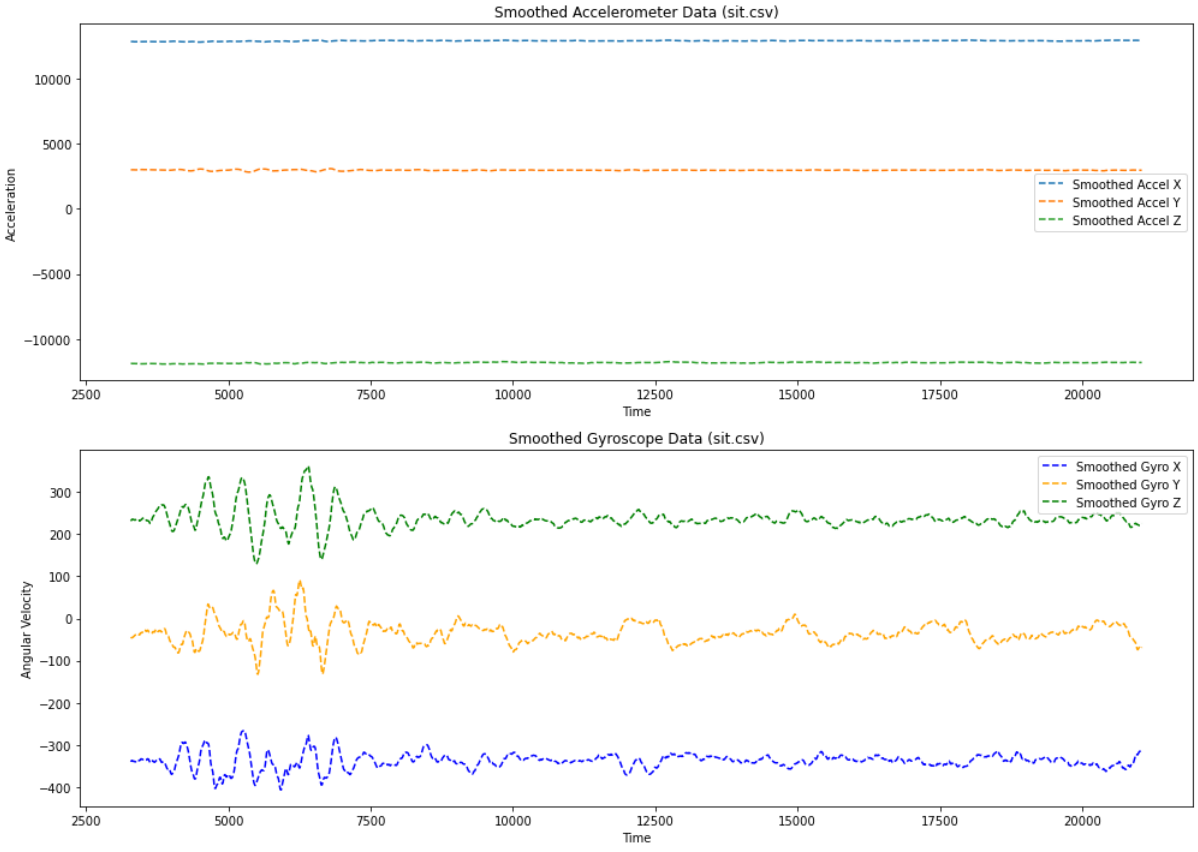
The following observations can be made:

- **Accelerometer Data:**
  - **Sitting:** Minimal variation, flat lines indicating little movement.
  - **Standing:** Low variation, slightly more noise than sitting due to minor body adjustments.
  - **Walking:** Regular, periodic peaks indicating the rhythmic motion of walking.
- **Gyroscope Data:**
  - **Sitting:** Low, consistent angular velocity.
  - **Standing:** Low angular velocity with minor variations to maintain balance.
  - **Walking:** Periodic changes in angular velocity corresponding to the swinging motion of arms and legs.

overall, the accelerometer data for sitting and standing are very similar in all three directions, which is expected because there is no motion in both activities. However, there are slight variations in the accelerometer data, reflecting minor body movements to maintain balance. Additionally, the gyroscope values are very distinguished, especially in the X and Z axes. Walking data is clearly distinguished from both sitting and standing.

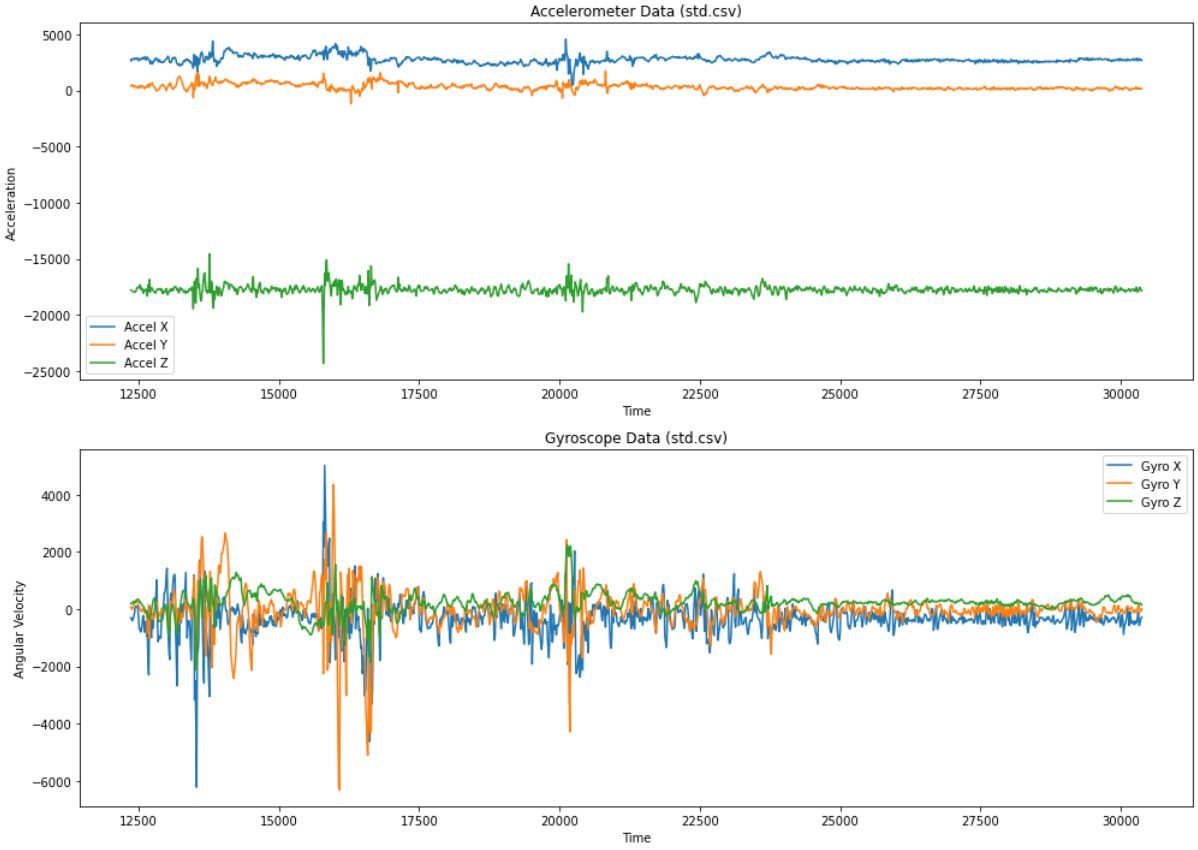


(a) Real data plot with time for sitting.

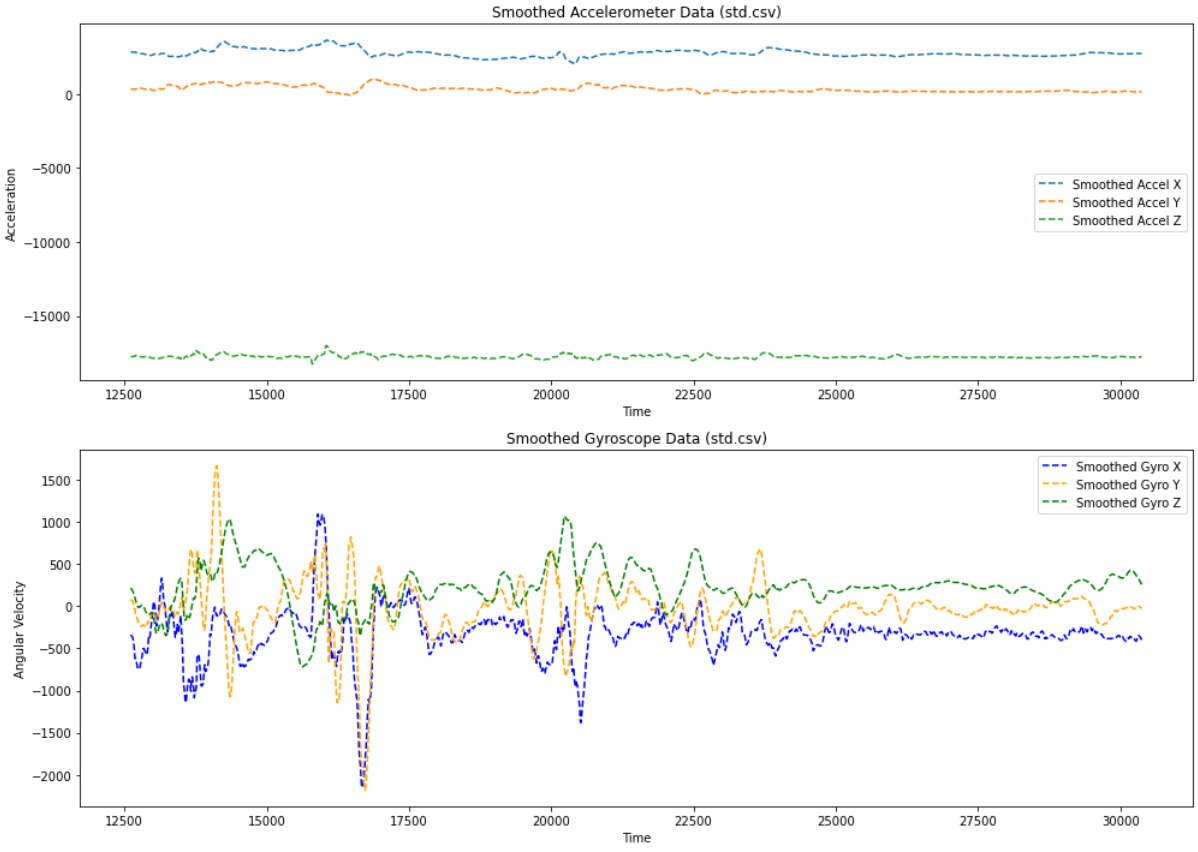


(b) Average window plot with time for sitting.

Figure 4.24: Sit data visualization.

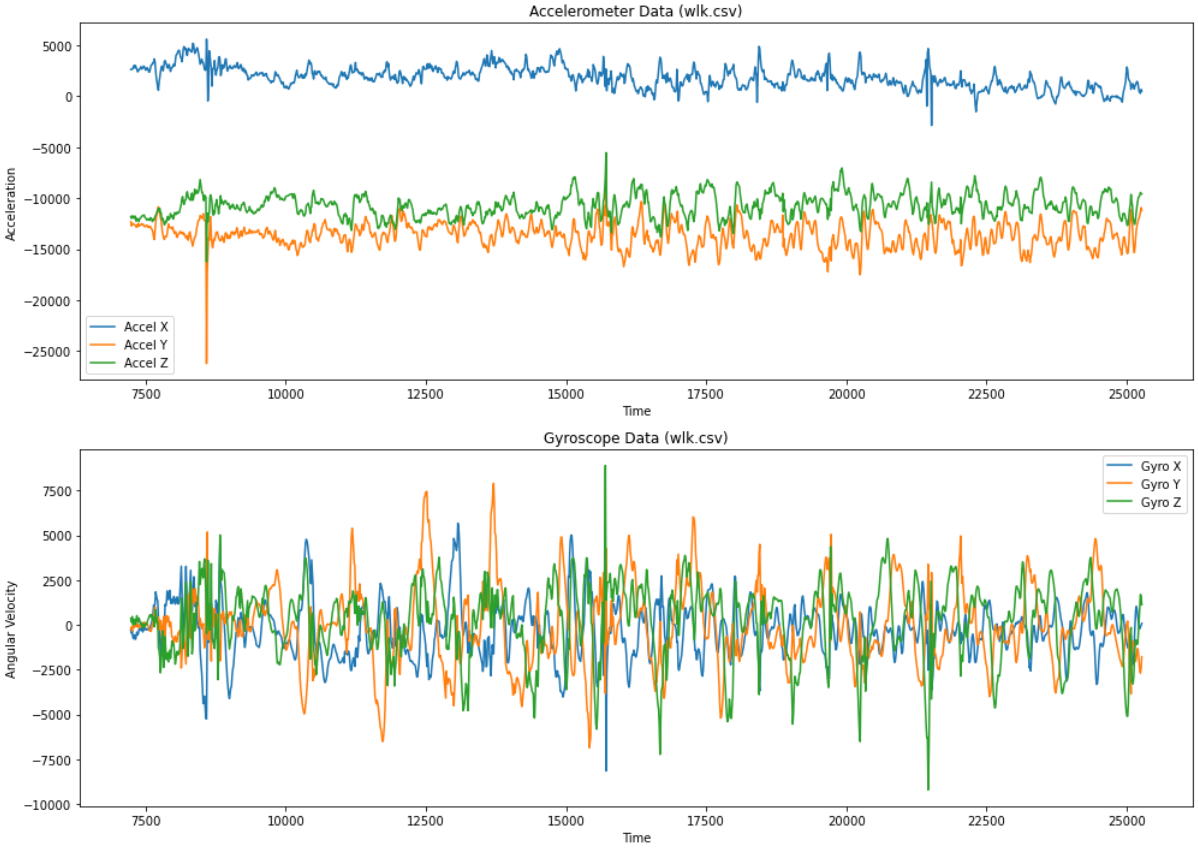


(a) Real data plot with time for standing.



(b) Average window plot with time for standing.

Figure 4.25: Stand data visualization.



(a) Real data plot with time for walking.

(b) Average window plot with time for walking.

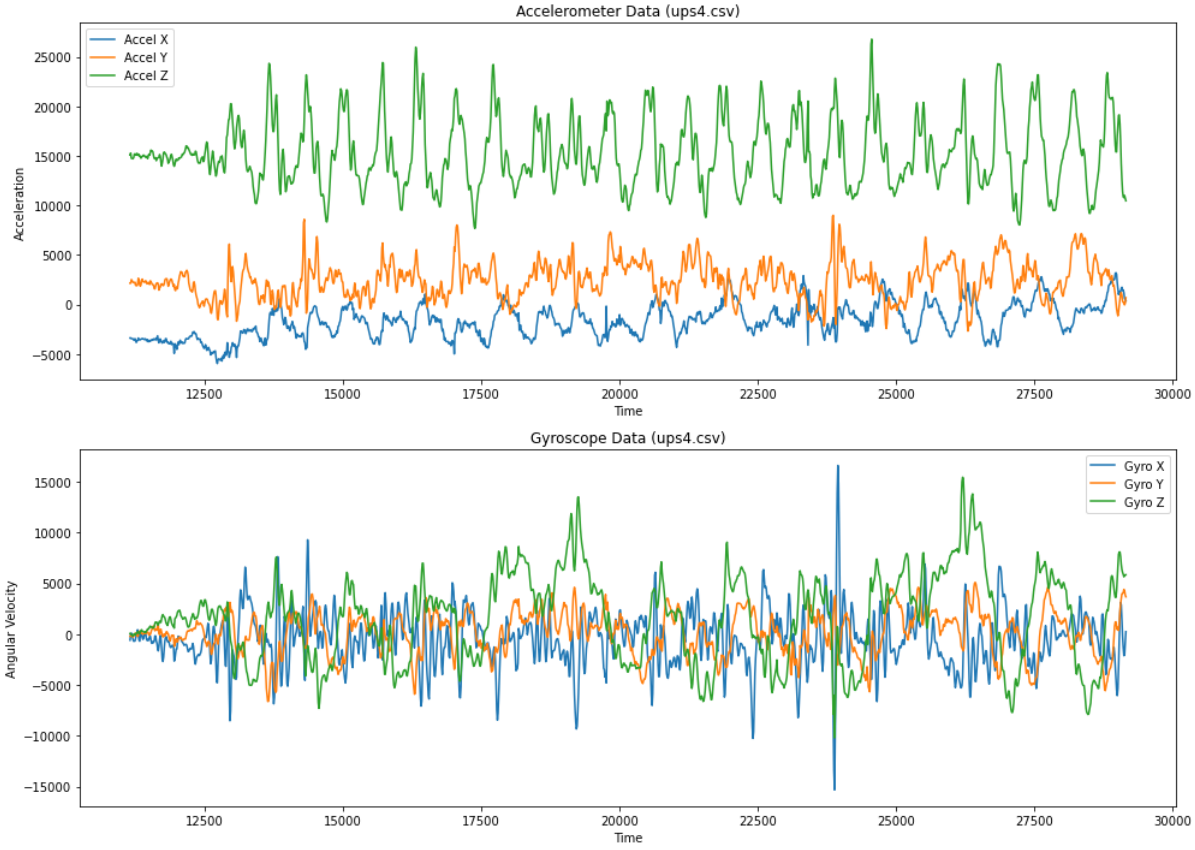
Figure 4.26: Walk data visualization.

4.6.3 Up and Down Stairs Data Collection and Visualization

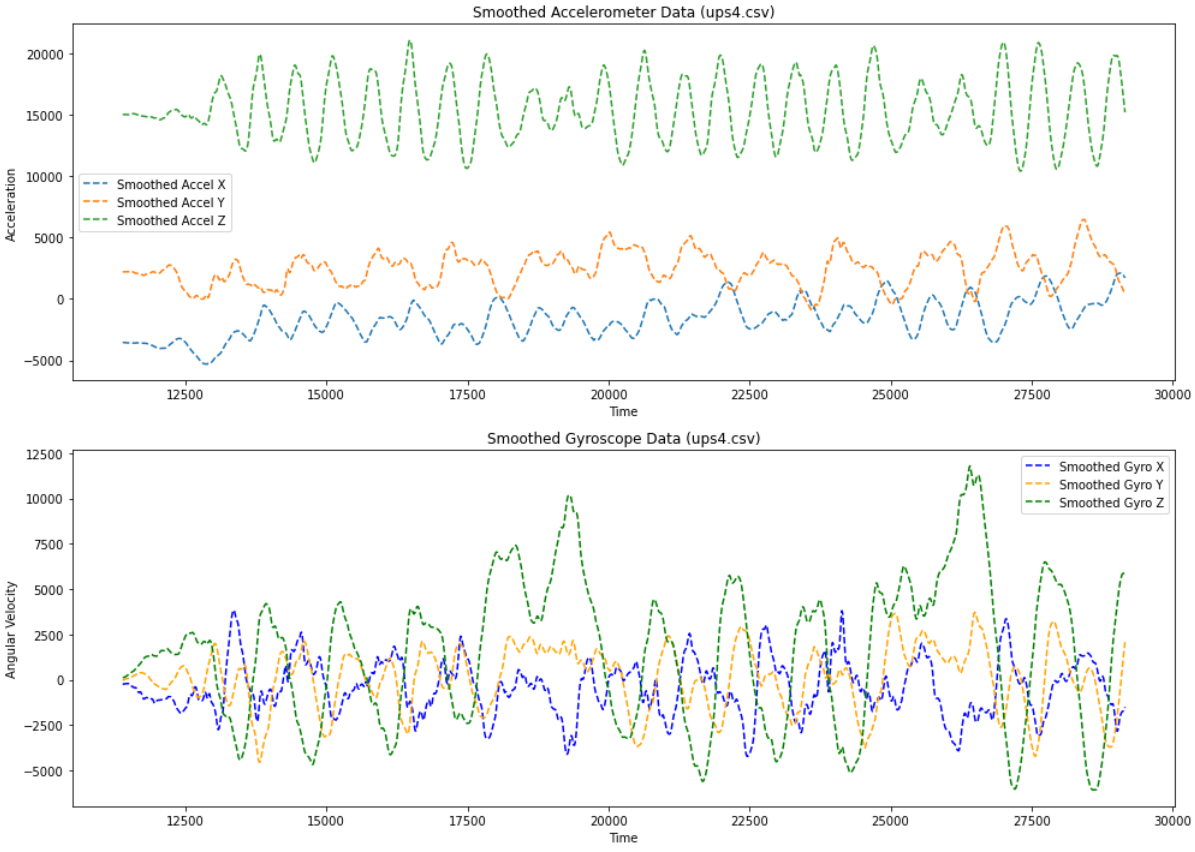
We recorded going upstairs and downstairs for the same 12 floors(240 staircases). From Figures 4.27 and 4.28, The following observations can be made:

- **Accelerometer Data:**
  - **Up Stairs:** Similar patterns to downstairs, making it hard to distinguish based on accelerometer data alone.
  - **Down Stairs:** Similar to upstairs with minor variations.
- **Gyroscope Data:**
  - **Up Stairs:** Unique patterns in the Z-axis, indicating the upward motion.
  - **Down Stairs:** Unique patterns in the Z-axis, indicating the downward motion.

from these figures,we see that the accelerometer data for going up and down stairs are nearly identical because they involve similar movements. Additionally, the X and Y gyroscope data are very similar. What really differentiates them is the gyroscope Z-axis, which is logical because one activity involves going up and the other involves going down.

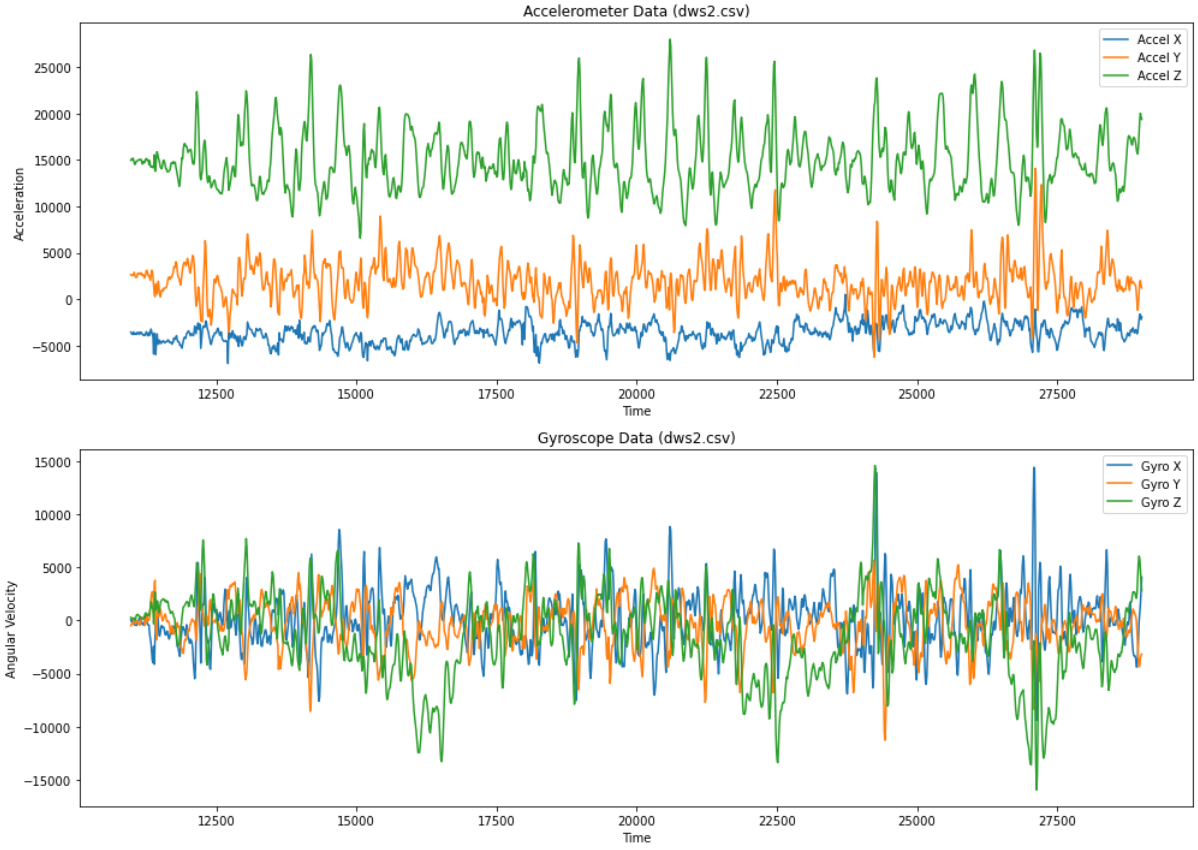


(a) Real data plot with time for going upstairs.

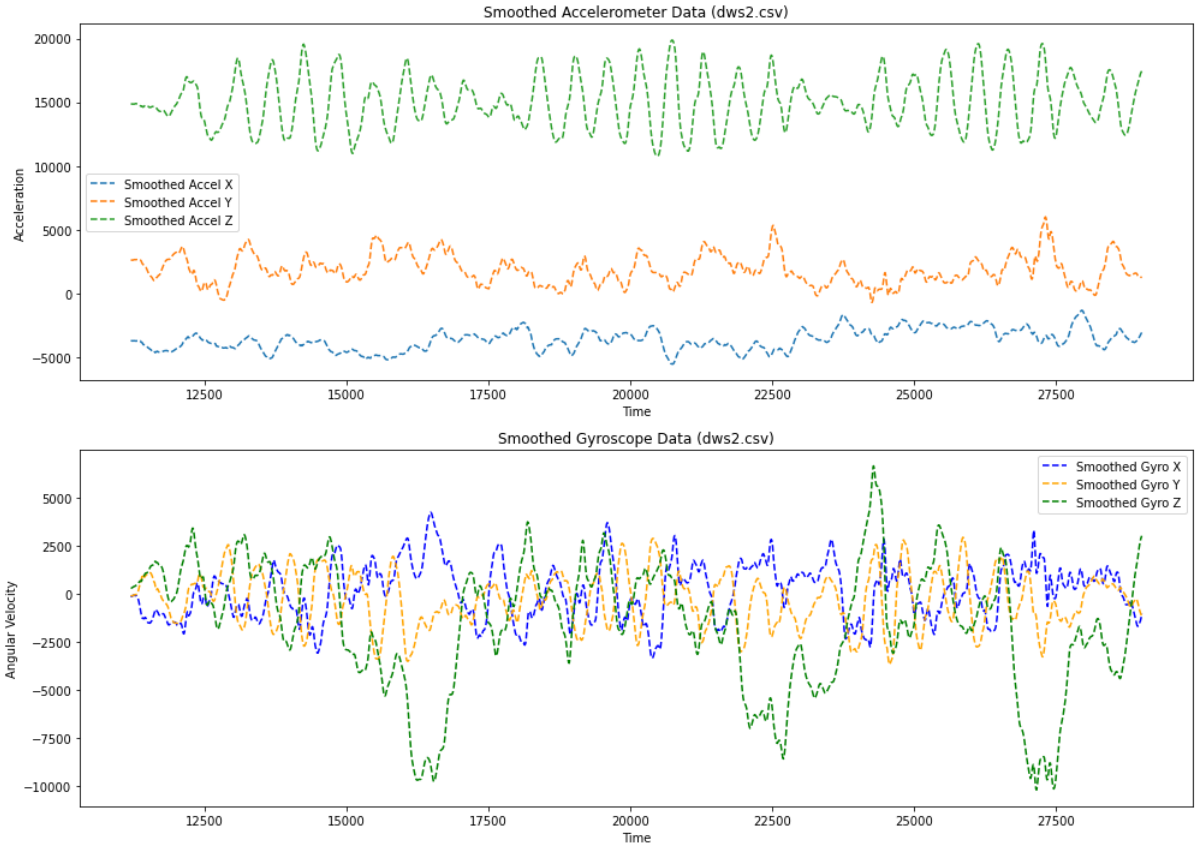


(b) Average window plot with time for going upstairs.

Figure 4.27: Ups data visualization.



(a) Real data plot with time for going downstairs.



(b) Average window plot with time for going downstairs.

Figure 4.28: Dws data visualization.

4.7 Machine Learning Analysis for Activity Recognition

After successfully recording data from several activities (sitting, standing, walking, going upstairs, and going downstairs), we conducted further analysis using machine learning techniques to recognize these activities. In this section, we detail the methodology employed for this analysis, focusing on decision trees as the primary classifier.

4.7.1 Utilization of Decision Trees

Decision trees were chosen as the classifier due to their simplicity and interpretability in handling numerical data effectively.

4.7.2 Coding

4.7.2.1 Programming Language Used

Python, particularly with the Spyder IDE, was selected as the programming language for conducting data analysis and machine learning tasks. Python provides an extensive array of libraries and tools tailored for data manipulation, preprocessing, modeling, and visualization, rendering it an optimal choice for diverse data science endeavors.

4.7.2.2 Library Imports

The code snippet in figure 4.29 imports various Python libraries and modules necessary for data manipulation, preprocessing, modeling, visualization, and evaluation. Here's a breakdown of the imports:

- `import pandas as pd`:

  The pandas library aliased as `pd`, is used for data manipulation and analysis. It offers powerful data structures like DataFrames and tools for handling missing data, reshaping, merging, and more.

- `from sklearn.preprocessing import StandardScaler`:

  The `StandardScaler` module standardizes features by removing the mean and scaling to unit variance. This preprocessing step ensures that features are on the same scale, which is crucial for many machine-learning algorithms.

- `from sklearn.tree import DecisionTreeClassifier`:

  The `DecisionTreeClassifier` module implements a classification algorithm based on decision trees. Decision trees recursively split the data based on features, making them simple and interpretable models for classification tasks.

- `from sklearn.metrics import accuracy_score, confusion_matrix`:

  These modules provide functions for evaluating classification models. `accuracy_score` computes the accuracy classification score, while `confusion_matrix` generates a confusion matrix, which is a table that describes the performance of a classification model.

- `from sklearn.decomposition import PCA`:

  The `PCA` module from implements Principal Component Analysis, a technique used for dimensionality reduction. PCA transforms the data into a new coordinate system to reduce the number of dimensions while preserving the most important information.

- `import matplotlib.pyplot as plt`:

  The `pyplot` module from the library provides a MATLAB-like interface for creating plots and visualizations. It is commonly used for generating static visualizations in Python.

- `import seaborn as sns`:

  The `seaborn` library is built on top of matplotlib and provides additional functionality for creating attractive and informative statistical graphics. It enhances the visual appeal of matplotlib plots and simplifies the process of creating complex visualizations.

- `import numpy as np`:

  The NumPy library, aliased as `np`, is a fundamental package for scientific computing in Python. It provides support for arrays, matrices, mathematical functions, random number generation, linear algebra, and more, making it essential for numerical computations.

```
1  import pandas as pd
2  from sklearn.preprocessing import StandardScaler
3  from sklearn.tree import DecisionTreeClassifier, plot_tree
4  from sklearn.metrics import accuracy_score, confusion_matrix
5  from sklearn.decomposition import PCA
6  import matplotlib.pyplot as plt
7  import seaborn as sns
8  import numpy as np
```

Figure 4.29: Library Imports for data analysis code

4.7.2.3 Data Loading and Preparation

The code snippet in Figure 4.30 demonstrates the process of loading data from CSV files into pandas DataFrames, preparing the data for analysis. Here's a breakdown of the steps involved:

1. **Loading Data**: The `pd.read_csv()` function is used to load data from CSV files into pandas DataFrames. Each CSV file contains data for a specific activity.

2. **Data Preparation**:

   - **Concatenation**: Data for each activity is concatenated into one data frame.
   - **Labeling**: Each data frame representing an activity is labeled accordingly to identify the activity it represents. This labeling step is crucial for supervised learning tasks, where the model needs to learn the relationship between features and activity labels.
   - **Trials Division**: Each activity is divided into 90 trials.
   - **Final Dataset Concatenation**: Finally, all individual datasets (representing different activities) are concatenated into one final dataset. This creates a comprehensive dataset containing data for all activities, facilitating further analysis and modeling.

```
# Load activity data: sit, std, wlk
sit = pd.read_csv('sit.csv')
std = pd.read_csv('std.csv')
wlk = pd.read_csv('wlk.csv')
# Load ups data: ups1, ups2, ups3
ups1 = pd.read_csv('ups4.csv')
ups2 = pd.read_csv('ups5.csv')
ups3 = pd.read_csv('ups6.csv')
dws1 = pd.read_csv('dws1.csv')
dws2 = pd.read_csv('dws2.csv')
dws3 = pd.read_csv('dws3.csv')
# Concatenate ups data into one DataFrame
ups = pd.concat([ups1, ups2,ups3])
dws=pd.concat([dws1,dws2,dws3])
# Assign activity labels to each dataset
sit['act'] = 0
std['act'] = 1
wlk['act'] = 2
ups['act'] = 3
dws['act']=4
# Calculate the size of each part to ensure nearly equal numbers of rows
sit_part_size = -(-sit.shape[0] // 90)  # Ceiling division to handle odd numbers
std_part_size = -(-std.shape[0] // 90)  # Ceiling division to handle odd numbers
wlk_part_size = -(-wlk.shape[0] // 90)  # Ceiling division to handle odd numbers
ups_part_size = -(-ups.shape[0] // 90)  # Ceiling division to handle odd numbers
dws_part_size = -(-dws.shape[0] // 90)  # Ceiling division to handle odd numbers
# Add trial column to each dataset
sit['trial'] = [i // sit_part_size + 1 for i in range(sit.shape[0])]
std['trial'] = [i // std_part_size + 1 for i in range(std.shape[0])]
wlk['trial'] = [i // wlk_part_size + 1 for i in range(wlk.shape[0])]
ups['trial'] = [i // ups_part_size + 1 for i in range(ups.shape[0])]
dws['trial'] = [i // ups_part_size + 1 for i in range(dws.shape[0])]
# Concatenate all datasets into one final dataset
final_data = pd.concat([sit, std, wlk, ups,dws])
#Reset the index
final_data.reset_index(drop=True, inplace=True)
```

Figure 4.30: Data Loading and Preparation for data analysis code

4.7.2.4 Data Splitting, Feature Engineering, Model Training, and Evaluation

The code snippet discussed in Figure 4.31 outlines a series of steps for data processing and model training. First, the data is divided into training and testing sets, with 18 trials allocated for training and 72 for testing. Subsequently, the training and testing data are grouped by both the 'trial' and 'act' columns, and features expressing each trial are aggregated. These features include mean, standard deviation, minimum, maximum, median, sum, variance, and skewness. Notably, the 'trial' and 'act' columns are excluded from feature scaling and fitting, as they have been separated into distinct data frames (y_test and y_train). Next, the features are scaled using StandardScaler to ensure uniform impact during fitting and prediction. Following this, a DecisionTreeClassifier is initialized, trained using the training data, and subsequently used to predict outcomes for each trial in the test set. Finally, the model's accuracy is evaluated using accuracy_score.

```
train_indices = [] # Split data into training and testing sets
test_indices = []
num_train_trials = 18
for activity_id in final_data['act'].unique():
    train_trials = np.random.choice(np.arange(1, 91), size=num_train_trials, replace=False)
    train_indices.extend(final_data[(final_data['act'] == activity_id) &
                                    (final_data['trial'].isin(train_trials))].index)
    test_indices.extend(final_data[(final_data['act'] == activity_id) &
                                   (~final_data['trial'].isin(train_trials))].index)
train_data = final_data.loc[train_indices]
test_data = final_data.loc[test_indices]
train_data.reset_index(drop=True, inplace=True)
test_data.reset_index(drop=True, inplace=True)
train_data.drop('Time', axis=1, inplace=True)
test_data.drop('Time', axis=1, inplace=True)
# Group the training data by both 'trial' and 'act' columns and aggregate the features
X_train_grouped = train_data.groupby(['trial','act']).agg(['mean', 'std', 'min', 'max',
                                                           'median', 'sum', 'var', 'skew'])
X_train_grouped.reset_index(inplace=True)
y_train_grouped = X_train_grouped['act']
# Group the test data by both 'trial' and 'act' columns and aggregate the features
X_test_grouped = test_data.groupby(['trial','act']).agg(['mean', 'std', 'min', 'max',
                                                         'median', 'sum', 'var', 'skew'])
X_test_grouped.reset_index(inplace=True)
y_test_grouped = X_test_grouped['act']
# Exclude 'trial' and 'act' columns from feature scaling and fitting
X_train_grouped_features = X_train_grouped.drop(['trial', 'act'], axis=1)
X_test_grouped_features = X_test_grouped.drop(['trial', 'act'], axis=1)
# Feature scaling
scaler = StandardScaler()
X_train_grouped_scaled = scaler.fit_transform(X_train_grouped_features)
X_test_grouped_scaled = scaler.transform(X_test_grouped_features)
# Initialize the decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_grouped_scaled, y_train_grouped)
# Predictions for each trial in the test set
y_pred_grouped = clf.predict(X_test_grouped_scaled)
# Evaluate the model
accuracy_grouped = accuracy_score(y_test_grouped, y_pred_grouped)
print("Accuracy:", accuracy_grouped)
```

Figure 4.31: Data Splitting, Feature Engineering, Model Training, and Evaluation for data analysis code

4.8 Data Analysis Results for Human Activities Recognition

After describing the analysis method in the previous section 4.7, we applied it to the human activities recorded by the system. As a result, we obtained specific findings along with corresponding plots that illustrate these results.

4.8.1 Accuracy Table

We ran our code five times to check the consistency of our results. The accuracy results for the different trials and the overall accuracy are presented in Table 4.1. we see that the worst accuracies are the ups and dws which were predicted due to their similarities as explained in subsection 4.6.3

Activity	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**All Trials**
Sit	1.0	1.0	1.0	1.0	1.0	1.0
Std	1.0	1.0	0.975	1.0	1.0	0.995
Wlk	1.0	1.0	1.0	1.0	1.0	1.0
Ups	0.9375	0.7625	0.9125	0.9375	0.925	0.895
Dws	0.8987	0.8734	0.8974	0.8974	0.9241	0.898
Overall Accuracy	0.9674	0.9273	0.9573	0.9673	0.9699	0.9579

Table 4.1: Accuracy of activities over five trials

4.8.2 Data Visualization

We visualized the data to gain more insights.

4.8.2.1 Decision Tree

The decision tree (Figure 4.32) illustrates how the model in one of the trials learned the data and correctly classified the activities. We observe that the accelerometer's y-axis and gyroscope's z and x axes were the main features for classification.

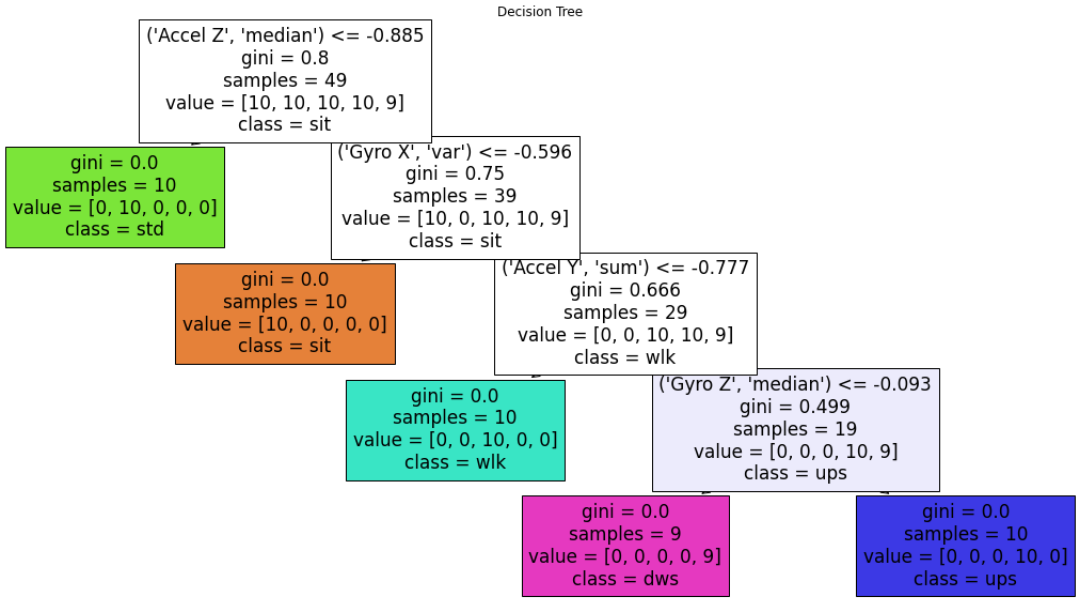


Figure 4.32: Decision tree

4.8.2.2 PCA Visualization

We plotted PCA for one of the trials to visualize the data in 2D, first with the predefined labels and then with the predicted labels. The results are shown in Figures 4.33 and 4.34, respectively. We observed that our model performed well in prediction, with minor errors. Additionally, it was evident that there is a similarity between the ups and dws data, as they exhibited the lowest accuracy in classification. The wlk data appears to be closely related to them, which may have contributed to some misclassifications.

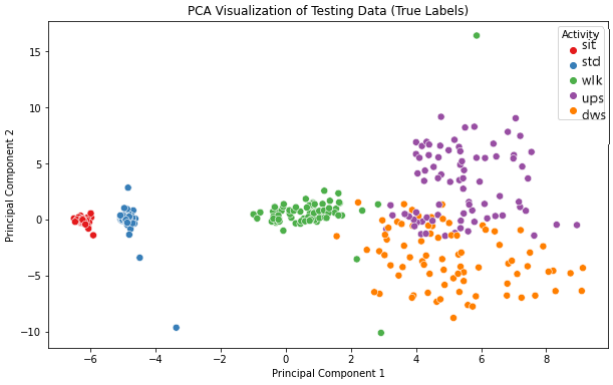


Figure 4.33: PCA visualization with true labels

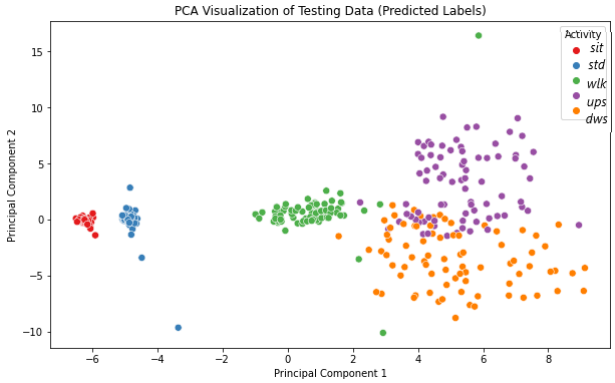


Figure 4.34: PCA visualization with predicted labels

4.8.2.3 Confusion Matrix

The confusion matrix (Figure 4.35) shows that ups and dws activities are somewhat confused with each other, while sit,wlk,std are well distinguished.

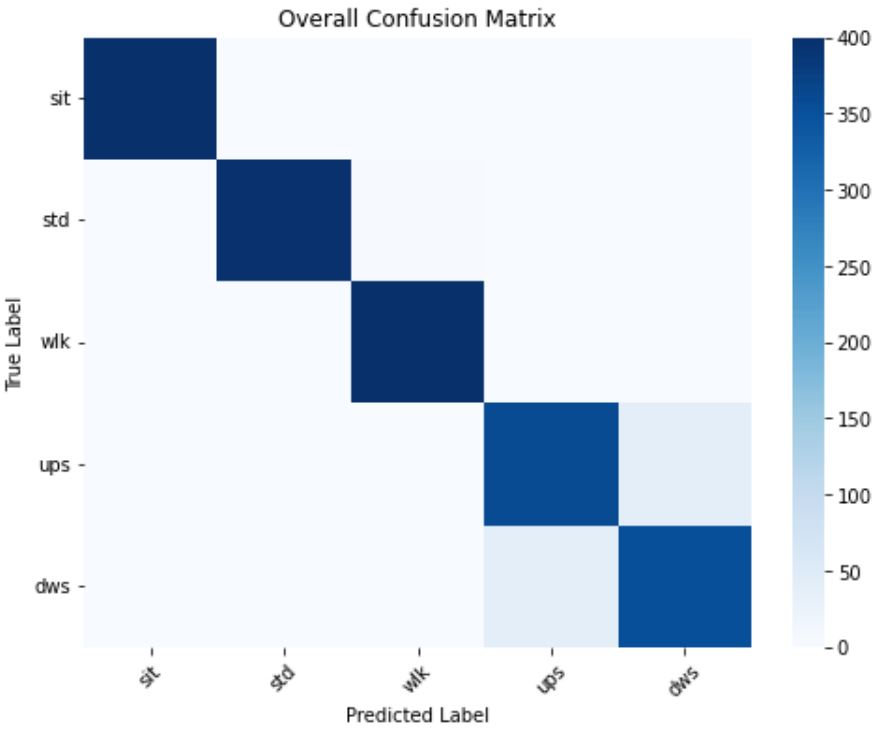


Figure 4.35: Confusion matrix

Chapter 5

Conclusion

In this thesis, we have developed a wearable sensor system capable of collecting sports-related data and utilizing machine learning algorithms for analysis. The integration of wearable sensors and machine learning techniques has shown promising results in enhancing our understanding of athletic performance and providing valuable insights for athletes and coaches.

Throughout the development process, we successfully programmed and tested the ESP32-CAM module and MPU-6050 sensor, designed and implemented a printed circuit board (PCB), and conducted thorough testing by collecting data from various human and tennis activities. Our system demonstrated a high overall accuracy of 95.79% in recognizing different human activities, highlighting its potential for reliable and detailed sports data collection.

The findings of this thesis contribute to the growing body of knowledge in the field of sports data analysis and wearable technology. By developing a wearable sensor system with robust data collection and analysis capabilities, we have laid the groundwork for further advancements in the application of wearable technology and machine learning in sports.

Chapter 6

Future Work

In this chapter, we outline potential avenues for future research and development based on our current findings.

6.1 Enhancement of Machine Learning Analysis

Future work involves conducting machine learning analysis on specific actions in sports, like football or tennis. By comparing multiple techniques, we can assess their effectiveness for activity recognition and improve the system's predictive accuracy.

6.2 Improving Sensor System Stability

To address stability issues, we'll design a 3D case for the system, preventing interruptions during data collection, especially during high-impact actions like powerful tennis shots. We'll also optimize power consumption for uninterrupted data recording.

6.3 Miniaturization and Optimization

We'll explore miniaturization options like the ESP32 D1 Mini Bluetooth+WiFi Board instead of the ESP32-CAM, reducing size and weight for easier wearability. Optimizing the system's design without compromising functionality is crucial.

6.4 Optimizing Data Transmission Over WiFi

To achieve real-time data transmission over WiFi, further optimization of the system's communication protocols is necessary. Future work could focus on finding ways to minimize latency and maximize data throughput, to help achieve the goal of sending data every 12 milliseconds, as demonstrated in offline mode.

Bibliography

[1] Pegah Rahimian, Afshin Oroojlooy, and Laszlo Toka. Towards optimized actions in critical situations of soccer games with deep reinforcement learning. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–12, 2021.

[2] Tao Wang, Jiajia Cui, and Yao Fan. A wearable-based sports health monitoring system using cnn and lstm with self-attentions. *PLOS ONE*, 18:e0292012, 10 2023.

[3] Jorge E. Morais. Editorial: Advances in wearable devices for sports. *Applied Sciences*, 13(24), 2023.

[4] Firoz Khan, Yashas S, Shivangowda Patil, Nandini J, and Greeshma S. Low cost intelligent child safety wearable iot device for india. *International Journal of Recent Technology and Engineering (IJRTE)*, 9:2367–2370, 05 2020.

[5] Bahzad Charbuty and Adnan Mohsin Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2021.

[6] Jacob W. Kamminga, Lara M. Janßen, Nirvana Meratnia, and Paul J. M. Havinga. Horsing around—a dataset comprising horse movement. *Data*, 4(4), 2019.

[7] Warren souza and Kavitha Rajamohan. Human activity recognition using accelerometer and gyroscope sensors. *International Journal of Engineering and Technology*, 9:1171–1179, 04 2017.

[8] Veralia Sanchez and Nils-Olav Skeie. Decision trees for human activity recognition in smart house environments. pages 222–229, 11 2018.

[9] Ariful Islam Anik, Mehedi Hassan, Hasan Mahmud, and Md Kamrul Hasan. Activity recognition of a badminton game through accelerometer and gyroscope. 12 2016.

[10] https://www.best-microcontroller-projects.com/esp32-cam.html .

[11] https://www.autodesk.ae/products/fusion-360/overview .

[12] Aislan Gomide Foina, Rosa M Badia, Ahmed El-Deeb, and Francisco Javier Ramirez-Fernandez. Player tracker - a tool to analyze sport players using rfid. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 772–775, 2010.

[13] Majid Dadafshar. Accelerometer and gyroscopes sensors: Operation, sensing, and applications. 2019.

[14] https://imacimi.wordpress.com/2019/11/12/how-has-big-data-analytics-affected-the-sports-industry/.

[15] https://www.dailymail.co.uk/sport/football/article-10853887/Liverpool-players-wear-sensors-BRAIN-cutting-edge-technology-German-neuroscientist.html.

[16] https://towardsdatascience.com/what-is-imu-9565e55b44c.

[17] https://celestialinfosoft.com/iot-in-the-sports-industry/.

[18] https://fab.cba.mit.edu/classes/863.22/CBA/people/Wedyan/week4.html.

[19] Khalid Mahboob, Raheela Asif, Syed Muhammad Nabeel Mustafa, and Humaira Rana. Predicting students' behavior towards their degree using machine learning techniques. In *2022 3rd International Conference on Innovations in Computer Science Software Engineering (ICONICS)*, pages 1–6, 2022.

[20] https://towardsdatascience.com/decision-trees-explained-d7678c43a59e .

[21] https://mins.space/blog/2020-06-29-moving-average-effect-window-size/ .

[22] https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d .

[23] https://dronebotworkshop.com/esp32-cam-intro/ .