



AutoCaddie

An AI-Driven Smart Coach

Louis Deal (EE), Rylan Simpson (CpE), Jarett Artman (CpE), Joshua Glaspey (CpE)

Department of Electrical and Computer Engineering

Table of Contents

Chapter 1: Executive Summary.....	2
Chapter 2: Project Description	3
2.1 Project Background and Motivation	3
2.1.1 Introduction	3
2.1.2 Motivation	4
2.1.3 Function of Project.....	4
2.2 Project Goals and Objectives.....	4
2.2.1 General Goals.....	4
2.2.2 Software.....	4
2.2.3 Hardware	5
2.2.4 Hyperparameters for AI Analysis	6
2.3 Overview of Planned System Design.....	7
2.3.1 General Process Flow	8
2.3.2 Hardware Control Flow	9
2.3.3 Software Control Flow	10
2.4 Engineering Requirements and Constraints.....	11
2.4.1 Engineering Requirements Overview	11
2.4.2 Engineering Requirements Table	12
2.4.3 Engineering Constraints.....	12
2.5 Quality Analysis	13
2.5.1 User / Marketing Specifications	13
2.5.2 House of Quality	14
Chapter 3: Project-Related Research.....	15
3.1 Existing Similar Projects and Topic Research.....	15
3.1.1 Smart Coaching	15
3.1.2 AI Golf Coaching.....	16
3.1.3 Real-Time AI Analysis	17
3.1.4 Inertial Measurement Unit Joint Kinematics Modeling	18
3.1.5 Effects of Distractions and Stimuli During Swing Action	19
3.1.6 Multi-Modal Feedback Training.....	20
3.1.7 Putting It All Together	20

3.2 Overall Structure and Relevant Technologies.....	22
3.2.1 System Design Overview for Technology Research.....	22
3.3 Relevant Technologies: Computer Vision and Neural Networks.....	25
3.3.1 Video Cameras.....	25
3.3.2 Live Video Analysis.....	27
3.3.3 Network Architecture	38
3.3.4 Database Selection.....	40
3.4 Relevant Technologies: Kinematic Data	44
3.4.1 Inertial Measurement Units.....	44
3.4.2 Wireless Data Transmission Components.....	46
3.4.3 Battery Power Source.....	49
3.4.4 Microcontroller Unit.....	51
3.4.5 Quaternion Data Processing.....	53
3.5 Relevant Technologies: Software Integration.....	56
Chapter 4: Related Standards and Realistic Design Constraints	71
4.1 Standards.....	71
4.1.1 Hardware Standards.....	71
4.1.2 Software Standards	74
4.1.3 Design Impact of Relevant Standards	80
4.2 Realistic Design Considerations and Constraints	82
4.2.1 Economics and Time	82
4.2.2 Environmental and Political.....	83
4.2.3 Health and Safety	84
4.2.4 Ethical and Social	84
4.2.5 Manufacturability and Sustainability.....	85
Chapter 5: Comparison of ChatGPT with other Similar Platforms.....	86
5.1 ChatGPT.....	86
5.2 Similar Platforms.....	87
5.3 Platform Comparisons.....	88
5.3.1 HIX.AI	88
5.3.2 Chatsonic.....	89
5.3.3 Microsoft Bing	89

5.3.4 YouChat.....	89
5.3.5 Claude	90
5.3.6 Google Bard.....	90
5.3.7 Auto-GPT.....	91
5.3.8 Copy.ai.....	91
5.4 Platform Selection.....	92
Chapter 6: Project Hardware and MCU Design Details.....	93
6.1 PCB Subsystem Overview	93
6.2 PCB Schematic Design and Methodology	94
6.2.1 PCB Schematics Overview	94
6.2.2 ESP32 Module.....	95
6.2.3 USB programming	96
6.2.4 Power Control.....	97
6.2.5 Connection Headers and Other Hardware.....	98
6.2.6 Main Backplane circuit.....	99
6.2.7 Voltage Converter Breakout.....	100
6.3 Peripheral Hardware Integration	100
6.3.1 IMU System Design and Integration: Arm Linearity Example	100
6.3.2 IMU System Integration Option: Back Rotation Example.....	103
6.3.3 IMU System Integration Choice: Shoulder Alignment to Ground	104
6.3.4 Camera System and Placement	105
6.4 PCB and Peripheral Control.....	105
6.4.1 MCU Control	105
6.4.2 IMU Component control.....	106
6.5 Modularity of Hardware Peripheral Sensors.....	107
6.6 Summary of Hardware and MCU Design.....	107
Chapter 7: Project Software Design Details	108
7.1 General Overview	108
7.2 Machine Learning Model.....	109
7.3 Software Organization.....	111
7.3.1 Software Design Overview.....	111
7.3.2 Low-Level Software Flow.....	113

7.3.3 Development Components	113
7.4 Communication	117
7.4.1 MCU	117
7.4.2 Cameras	118
7.4.3 Neural Network	119
7.4.4 Internal Communication	119
7.4.5 GUI	120
7.5 Data Processing Algorithmic Integration	120
7.5.1 Algorithmic Structure	120
7.5.2 Frame Extraction	123
7.5.3 Image Standardization	123
7.5.4 Kinematics	124
7.6 Data Management	124
7.6.1 Data Collection and Storage Requirements	124
7.6.2 Video Data Augmentation	125
7.6.3 Quaternion Data Augmentation	126
7.7 Neural Network and Kinematic Algorithm Interpretation.....	126
7.7.1 Neural Network Output Interpretation	126
7.7.2 Quaternion Data Output Interpretation.....	127
7.8 Graphical User Interface	129
7.8.1 Framework Selection: tkinter	129
7.8.2 Parameters and Data Output	130
7.8.3 Project Directory Structure.....	131
7.8.4 GUI Design	133
7.9 System Integration and Optimization	140
7.9.1 Overall Integration	140
7.9.2 Algorithmic Optimizations	140
7.10 Summary of Software Design	144
7.10.1 Introduction	144
7.10.2 Machine Learning and AI Subsystems	144
7.10.3 Software Implementation and Design	144
7.10.4 Technical Aspects of Software.....	145

7.10.5 Communication and Data Processing	145
7.10.6 Data Management and Analysis	145
7.10.7 GUI and System Integration.....	145
7.10.8 Optimization and Validation	145
7.10.9 Conclusion	146
Chapter 8: CAD Design and Fabrication	147
8.2.1 PCB Board Layout	147
8.2.2 PCB Fabrication.....	148
8.3 Kinematics system and hardware	149
8.3.1 PCB and IMU enclosures.....	149
8.3.2 Enclosure box mounting	150
Chapter 9: Testing Plan.....	151
9.1 Hardware Component Test and Verification	151
9.1.1 MCU Wiring Verification.....	151
9.1.2 IMU Component Testing	151
9.1.3 Wireless Transceiver Testing.....	151
9.2 Neural Network Testing.....	152
9.2.1 Neural Network Optimization	152
9.2.2 Neural Network Validation	154
9.3 Software Integration Testing	155
9.3.1 Introduction	155
9.3.2 Unit Testing with tkinter and Python unittest	156
9.4 Summary of Testing Plan.....	158
Chapter 10: Justification for Usage of Laptop as Central Processing Device.....	159
10.1 Selection of Processing Device.....	159
10.2 Storage Considerations.....	160
10.3 Cost Analysis	161
10.4 Processing Capabilities.....	162
10.5 Final Justifications.....	162
Chapter 11: Conclusion.....	164
Chapter 12: Administrative Content	165
12.1 Estimated Budget.....	165

12.2 Semester Milestones.....	166
12.3 Distribution of Responsibilities	167

Chapter 1: Executive Summary

Technology is reshaping every facet of our lives, and the domain of sports and physical training remains ripe for innovation. Addressing this gap, a project led by Group 22, stands as a transformative solution aimed at revolutionizing the way athletes and fitness enthusiasts train, improve, and excel.

AutoCaddie leverages cutting-edge Artificial Intelligence algorithms to serve as a smart coach, offering real-time, individualized feedback for enhancing performance. Unlike traditional coaching methodologies that may be subject to human biases and limitations, AutoCaddie employs data analytics and sensor technologies to provide precise and actionable insights.

The project encapsulates a comprehensive development framework that spans across multiple hardware integrations, software engineering, and user interface design. Modern sensors collect a wealth of performance metrics, which are then processed through optimized algorithms alongside machine learning models sequencing live camera data to generate tailored coaching feedback. This represents a leap in technological capability, converging multiple disciplines to deliver a product that not only stands on its own merit but sets new standards for smart coaching solutions.

This document serves as an exhaustive guide detailing every aspect of the AutoCaddie project—from initial research and design constraints to testing protocols and final implementation. It incorporates an in-depth examination of relevant existing technologies, meticulous part selection based on empirical research, and a discussion of design standards to ensure compliance with industry norms.

The AutoCaddie project is not merely a technological advancement; it is a paradigm shift. It promises to elevate the quality of training and performance optimization to unprecedented levels, offering tangible benefits to athletes, coaches, and sports organizations alike. Therefore, it is poised to make a significant impact in bridging the technology gap in today's sports training landscape.

For those who wish to delve deeper into the technical and administrative facets of the project, the document includes extensive appendices and references, providing a wealth of information for further study and research.

Chapter 2: Project Description

2.1 Project Background and Motivation

2.1.1 Introduction

One of the most challenging aspects of sports training and physical fitness is the lack of personalized, real-time feedback. Whether you're a professional athlete or a casual fitness enthusiast, the absence of immediate and accurate coaching can hinder performance and lead to a plateau in improvement. To address this problem, we introduce AutoCaddie, a cutting-edge AI-driven smart coaching system that aims to redefine the way individuals train and improve in their respective sports or fitness regimes.

AutoCaddie employs a comprehensive suite of sensors and data analytics tools to capture a wide range of performance metrics. These metrics are processed through a sophisticated machine learning algorithm, alongside other optimized computations, which then provide real-time personalized coaching feedback directly to the user. This system is designed to be integrated into existing training and can also function independently, offering a versatile solution adaptable to various sports and physical activities.

In addition to real-time feedback, AutoCaddie features an interface that provides a more in-depth analysis of the collected data. This includes the information of the current swing performed by the player and other data we collected from our camera footage. This feedback will be immensely valuable to the player.

Moreover, all the data captured by AutoCaddie can be securely stored and accessed remotely, serving multiple purposes. It can be used for further analysis, for more nuanced feedback, or even utilized for research and development in sports science.

By delivering a highly adaptable, precise, and user-friendly smart coaching solution, AutoCaddie aims to bridge the existing gap in personalized sports training. The system promises not only to enhance individual performance but also to contribute to the broader field of sports technology and analytics.

For potential future and additional various iterations, and to support a customizable experience to better assist in a user's improvement of golf swinging skills, a key approach in the design of the hardware subsystem of the AutoCaddie project is modularity. Modularity allows the user to customize their experience and train on specific areas of the golf swing. The user can choose to possibly use various types of sensors with appropriate software adjustments, or can use various types of sensor placements. For example, a user can choose to train their posture in swing preparation by placing inertial measurement units on their back, or measure their footwork by using feet pressure sensors.

2.1.2 Motivation

As engineers with a passion for golf, we understand firsthand the challenges and limitations of traditional training methods in the sport. The existing systems often lack real-time, personalized feedback essential for meaningful improvement. AutoCaddie emerged from our desire to integrate our engineering skills with our hobbies, aiming to address these gaps. By leveraging cutting-edge AI and sensor technology, we aspire to enhance not only our own golfing experience but also to make advanced, data-driven coaching accessible to golfers at all levels. Through AutoCaddie, we aim to elevate the standards of training, making the sport more engaging and rewarding for everyone involved.

2.1.3 Function of Project

The function of the AutoCaddie project is to capture and analyze the biomechanics of a golfer's swing using a combination of cameras and sensors. The system processes this data in real-time through machine learning algorithms to identify areas for improvement. The analyzed information is then immediately displayed on a computer screen in a user-friendly format, providing golfers with actionable tips to enhance their swing. This enables golfers to make instant corrections, fostering more effective practice sessions and accelerating skill development.

2.2 Project Goals and Objectives

2.2.1 General Goals

- Develop an artificial intelligence program utilizing machine learning to provide multiple modes of feedback for a user's golf swing (using primarily a driver).
- Design a printed circuit board (PCB) capable of performing wireless transmission of simple numeric data to a central computer.
- Interface kinematic and video data into a central data processor computer.
- Measurably improve the user's golf swing through the usage of the system.

2.2.2 Software

2.2.2.a Main Goal:

Develop a ML-driven AI to provide feedback to a user's swing that integrates both sensor and video data.

2.2.2.b Objectives:

- Find a database to train the AI.

GolfDB [1] is a repository of golf videos from a set number of camera perspectives and is freely available to the public. These videos will be utilized to train the machine learning program to evaluate a swing from two different camera angles, one from the side and one from behind.

- Integrate video feed data to AI program.

Two cameras (listed under budget) will be utilized to send video data to the laptop for AI analysis via a wired connection.

- Effectively train the AI to recognize swings and make comparisons.

The AI program will receive video data via a cable and will compare the active video feed to the videos gathered from GolfDB, returning audio and visual feedback based on that comparison.

- Program the AI to effectively provide feedback based on comparisons.

The machine learning program will be programmed such that it returns audio feedback (via a buzzer either on the player or the laptop to indicate proper arm straightness) and visual feedback via a graphical user interface (GUI) displayed on a screen.

- Integrate IMU sensor data with AI.

Absolute position and calculated shoulder straightness (IMU orientation vectors will be included in a vector calculation to determine the overall angle from shoulders to the ground plane and will be compared to a threshold of acceptable arm angles) will be fed to the AI program to assist it in providing feedback.

- Design and integrate visual and audio feedback queues.

An on-screen GUI designed by the machine learning team will be utilized to provide visual feedback based on the IMU data and analyzed active video feeds, while a buzzer will be used to indicate proper position of the arm.

2.2.3 Hardware

2.2.3.a Main Goal:

Design a system to measure the kinematics of the user's arm, and wirelessly relay the data via a transceiver to a central computer for processing.

2.2.3.b Objectives:

- Select and acquire hardware components.

The included hardware system of the AutoCaddie project should integrate IMU systems that interface with a wireless communication circuit utilizing wireless transceiver modules for communication with a computer. Furthermore, cameras that meet the requirements of an implementation of machine learning artificial intelligence should be chosen.

- Integrate I2C or UART communication of IMUs with MCU chip.

I2C or UART, communication protocols for technology, will be used to extract the measurements provided by the IMUs and send them to the MCU.

- Integrate wireless communication between MCU and main processor computer.

UART, another serial communication mode, will be utilized to send the calculations from the MCU chip to the wireless transceiver on the PCB, and will be used to extract the received data on the laptop to the AI analysis program.

- Design mounting system for IMU sensors on user.

The IMUs will be integrated into a flexible sport compression sleeve, set into the surface of it at the wrist, elbow, and shoulder. The wires connecting the IMUs to the MCU will be fed through flexible tubing which will prevent the wires from inhibiting the ergonomics of the wearable system.

- Design IMU housing components for mounting system

The IMUs will be mounted into a compression sleeve using an adhesion method (likely Velcro) and a compression method to prevent their motion (likely elastic bands which can be tightened from a central knot).

- Design data collector/data transmission PCB

A system model has already been developed, but a lower-level model of the PCB will not be completed for some time.

- Design PCB housing for mounting system

The PCB will be housed within a 3d-printed, filament-based plastic housing designed to the specific shape of the PCB board. The battery will be contained within and will be activated with a switch to ensure that power can be conserved. The design will be such that it does not impede the motion of the player.

- Integrate video camera feed into main processor computer.

2.2.4 Hyperparameters for AI Analysis

2.2.4.a Main Goal:

Accurately define and quantify the essential components of a golf swing for the AI to assess and provide feedback.

2.2.4.b Objectives:

- Analyze the arm straightness.

Monitor the relative straightness of the inner arm throughout the swing.

Instruct the AI to detect and highlight any variations from the ideal arm posture.

- Analyze the shoulder rotation.

Quantify the degree and fluidity of the shoulder rotation during the swing.

Guide the AI to assess shoulder mobility and compare it against ideal benchmarks.

- Analyze the hip rotation.

Measure the rotation of the hips and their synchronization with the rest of the body.

Enable the AI to determine if the hip rotation is optimal or requires adjustment.

- Analyze the initial body posture.

Quantify the relative movement of each shoulder to itself while setting up a swing.

Perform quaternion data calculations to determine axes rotations over time.

2.3 Overview of Planned System Design

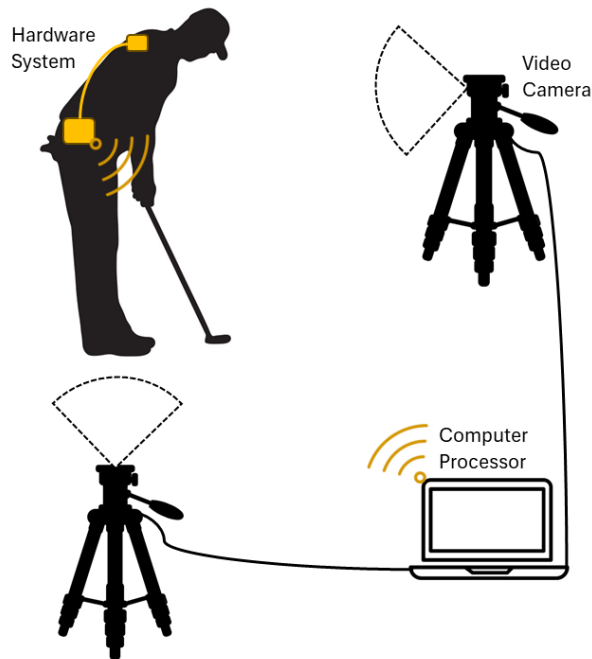


Figure 1: Graphic representation of physical system setup.

As seen by the above figure, the AutoCaddie system involves three separate streams of data. The system integrates two video camera feeds and a sensor data feed from IMU sensors on the user. The data is fed to a computer or laptop that can be placed in view of the user for visible and audible feedback. As each data feed source is portable, the devices that produce the data can be placed within the environment as necessary, while maintaining given requirements. The sensor data on the user should maintain consistent data streaming to the computer and should be both secure on the user and should be calibrated as necessary. The first of the two cameras should be placed at the front of the user, in order to record the full posture and straightness of the user's arm. The second camera should be placed on the side of the user, to record the swing pathing in relation to the ball. Finally, the computer where feedback is given can be placed by preference of the user.

2.3.1 General Process Flow

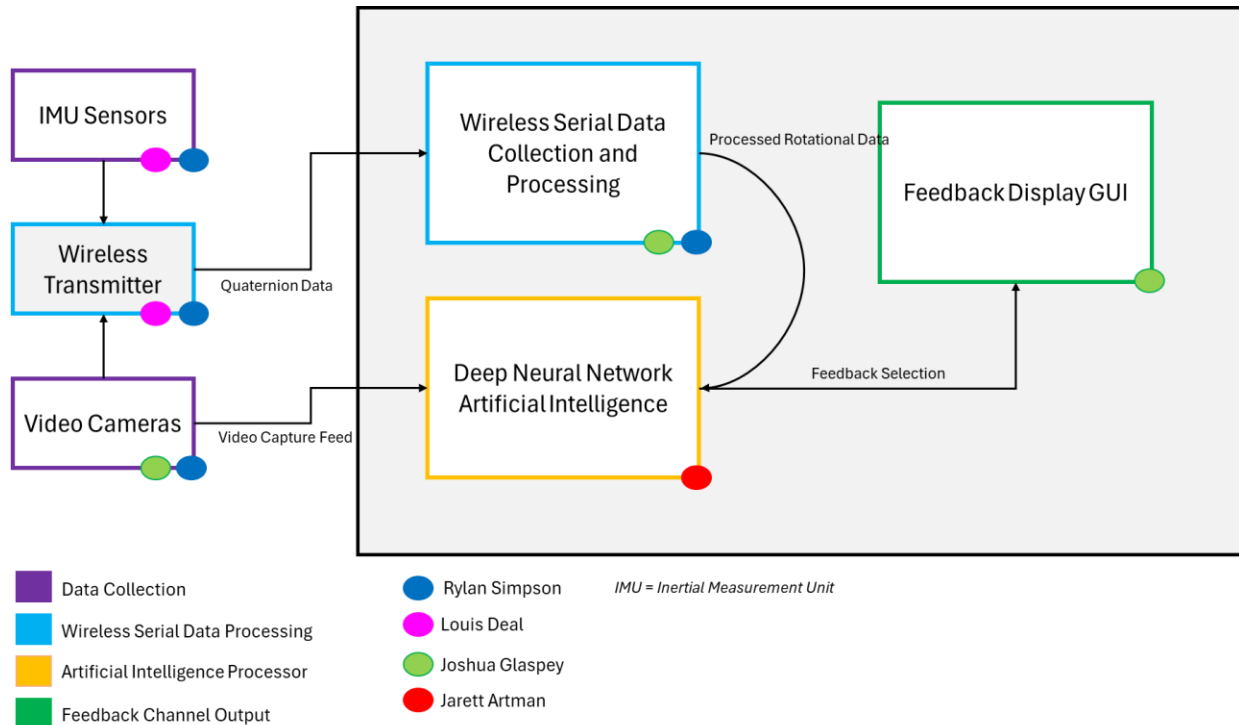


Figure 2: Block diagram showcasing the system overview and data path of the AutoCaddie system.

The related block diagram figure demonstrates the high-level overview of the AutoCaddie system and the flow of the data within the system during its function. Outside of the data processing subsystem, there are two channels in which external data is collected and is sent for processing. These two channels are the IMU sensor channel and the video camera channel. The IMU sensors collect quaternion data on the user's driving arm. This data is then sent to a wireless transceiver module so that the sensor data can be sent to a processor for analysis wirelessly, so as not to impede user movement. This sensor data can be cleaned and post-processed by artificial intelligence as necessary. The video cameras record video footage of the user, particularly during swing action. This video footage is transmitted to the central computer where the artificial intelligence analyses the footage against a database by which it was previously trained on. Finally, once the artificial intelligence's analysis and direct calculations are completed, the program generates an output sequence, where visual feedback is provided to the user.

2.3.2 Hardware Control Flow

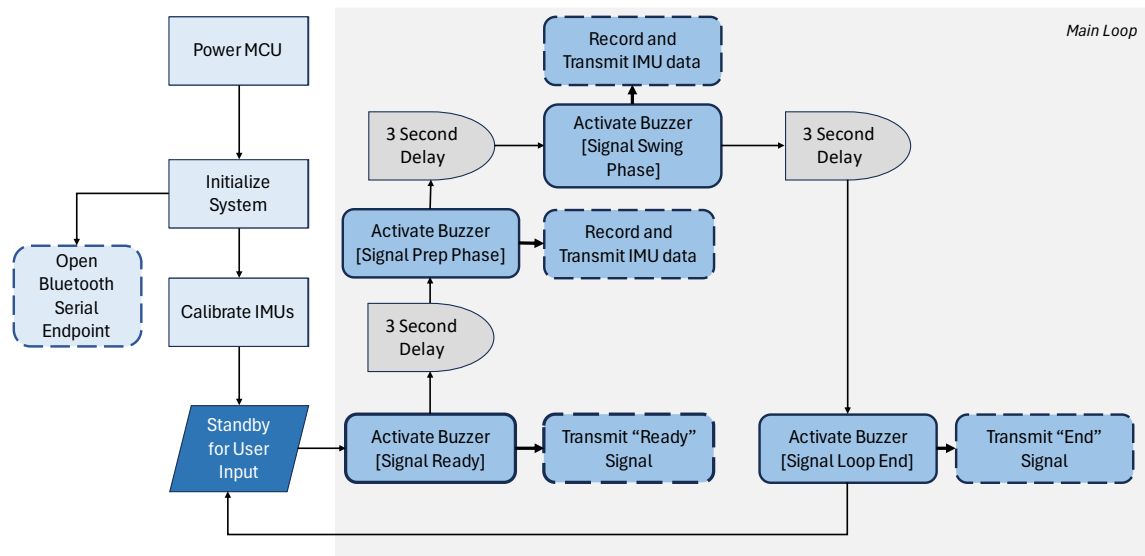


Figure 3: Block diagram demonstrating a general overview of the hardware control within AutoCaddie.

This block diagram provides a general overview of the hardware system directed by the hardware's microcontroller. The system begins when it is powered on. Once powered, the system initializes itself. During initialization, both the wireless communication link is opened, and the IMUs are calibrated. Once This is done, the system entered a standby mode, where it waits for user input to proceed with the main loop. When the user presses a "start" switch, the main loop of the system begins. First, the system will activate a buzzer to indicate to the user that the system has begun the main loop. At the same time, the system will transmit to the computer that it has started its sequence, and to begin with its own sequence and record data via the video cameras. After three seconds, giving the user time to get into position, the system will activate the buzzer again, indicating that the swing preparation phase has begun. During this time, the system will record data from the IMUs on the user, and transmits the data to the computer. After this three-second phase is complete, The system will activate the buzzer once more to indicate the user to swing. During the user's swing action, IMU will once again be collected and transmitted. After the last 3-second period in which the user should swing, the buzzer will be activated a final time to indicate to the user that the system has completed its process, and that they may return to the computer to finalize the collected data. The system will then re-enter the standby mode and wait or the user to once again activate the loop.

2.3.3 Software Control Flow

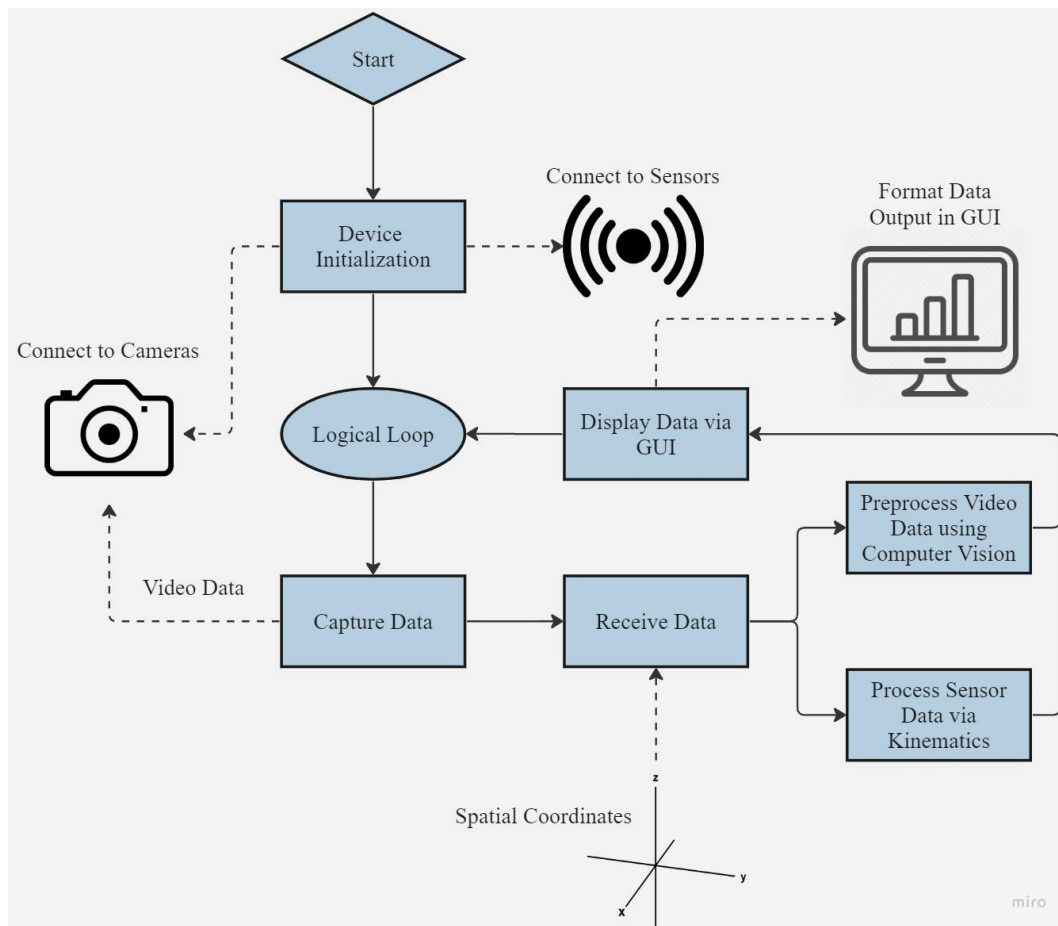


Figure 4: Block diagram showcasing a high-level overview of the software control within AutoCaddie.

This block diagram gives a high-level overview of the flow of the software system within AutoCaddie. The system begins by initializing the connected peripherals including the IMU sensors and two video cameras. Once all of these systems have been connected, the primary loop begins. First, the user is recorded taking a golf swing by both the physical data recording devices. The data from the IMUs are packaged and transmitted via wireless communication, and the cameras export the entire video broadcast to the computing device via a wired connection. Once the data has been collected, it is analyzed using two separate analysis systems. The first system extracts frames from the video data, preprocesses them, and sends them through AutoCaddie's deep neural network. The second system takes the IMU data and, through matrix operations, performs kinematic equations to deconstruct the data. At this point, the results are formatted and displayed to the user through a GUI, and the system can loop back to the data capturing state. All of these internal software systems combine to provide AutoCaddie's comprehensive AI feedback.

2.4 Engineering Requirements and Constraints

2.4.1 Engineering Requirements Overview

The AutoCaddie project is an interdisciplinary initiative that converges the fields of electrical engineering, computer science, and data analytics to construct a sophisticated, real-time golf swing analyzer. This senior design project aims to solve a specific problem in the realm of sports technology: providing immediate, actionable feedback to golfers to improve their swing.

The hardware setup involves a strategically positioned high-resolution frontal camera along with additional sensors. These devices are meticulously engineered to capture crucial metrics of a golfer's swing, such as speed, angle, and body posture. One unique feature of this system is the incorporation of a buzzer mechanism. This buzzer sounds at the appropriate moment, signaling the golfer to start their swing and thereby ensuring a standardized initiation point for all swing data collected.

The raw sensor and IMU data will be processed by a dedicated computing unit located near the golfer. This unit is equipped with both machine learning algorithms and hyper-specific algorithms designed to analyze the biomechanical aspects of a golf swing in real time. The algorithms will offer immediate coaching advice based on the analyzed data displayed on a nearby computer screen.

The screen's user interface is designed for simplicity and ease of use, ensuring that golfers can effortlessly understand and apply the real-time feedback provided. The feedback is tailored to highlight areas requiring immediate attention, allowing for quick adjustments and improved swings in subsequent attempts.

An essential hardware requirement for the project is the creation of a 3D-printed housing unit that attaches to the golfer's shoulders. This housing will serve as a secure mount for the circuitry and sensors. Additionally, another 3D printed mount will be attached to the user's hip through the use of a belt, allowing for the safe keeping of the central MCU.

The AutoCaddie system is built to be adaptable, capable of functioning both as an integrated part of existing training setups and as a standalone unit. While the system does not allow for adjustments based on the training environment, it is engineered to be reliable and effective in a variety of standard golf training settings.

In essence, the AutoCaddie project embodies a harmonious blend of hardware and software engineering components. The project serves not only as a practical, real-world application for immediate, data-driven golf swing improvement but also as an invaluable educational experience for the team. It equips team members with the hands-on experience necessary for multidisciplinary engineering roles in their future professional endeavors.

2.4.2 Engineering Requirements Table

Requirement	Description	Reasoning
Cost	Total costs less than \$500	System must be affordable for market entry and accessible for all players
Responsiveness	(Expected) Feedback within 10 seconds	Prevents long waiting time for feedback, including connection feedback systems for IMUs
Connection range	Stable connection from at least 20 meters of signal source	Requirements align with wireless data communication restrictions to send valid data for accurate feedback.
Accuracy	Feedback is accurate at least 90% of the time	Requirement sets standard for AI generated feedback, requiring reliable and desirable feedback.
Battery Life	2 hours battery life minimum of wireless transceiver	Requirements establish usability standards, enabling users to practice with the system for a meaningful duration.
Ergonomic	Wearable system is nonrestrictive and mountable in less than 5 minutes	Requirement establishes a usability standard for user-friendly system mounting.
Mounted Dimensions	Wireless transmission PCB designed in less than a 10cm-by-10cm frame	Requirement facilitates ergonomic design, following non-obstructing sizing for placement on user.
Weight	Wearable system weighs at most 5 pounds	Requirement facilitates ergonomic design, avoiding user fatigue of a weightier system.

Table 1: Engineering requirements table that outlines the relevant requirements set for the engineering design of the AutoCaddie System.

2.4.3 Engineering Constraints

The design of AutoCaddie seeks to involve both direct-conditional and user-resultant AI-driven feedback. To achieve this design endpoint, the system is required to integrate both a sensor system to measure and analyze user kinematics, and a video feed of the user for a machine learning algorithm to analyze the user's motions. The primary driving force behind the AutoCaddie system is the machine-learning based AI. This machine-learning based AI system must have some form of existing data to extrapolate off, and due to time constraints, a pre-existing database will be used. For these purposes, GolfDB shall be utilized. For accurate and consistent comparison of live data to the database, a video feed of the user must support at least a 30 frame-per-second video stream of at least a 420-pixel resolution. Furthermore, since the user should practice their golf swings in a proper outdoor environment, the software environment must be constrained to a portable computer system, such as a laptop, which must be plugged in or have some form of steady charge available, as to not overly limit a user's use time.

Regarding the hardware, there are several constraints that should be put in place. First of all, to promote potential accessibility for users in-market, the total costs of the device should remain under 500 dollars. Equally as important, the design of the hardware sensor systems must be designed so as not to impede the movement of the user while being as accurate as possible. The IMU sensors must be secured on the user and must be immobile with respect to the user's shoulders. Additionally, IMU sensors are subject to drift, so the IMU sensor system must be periodically calibrated, or else data inaccuracies may accumulate and produce incorrect results. With regards to the physical setup of the IMU sensor system, direct cable connections to the computer for data processing can interfere with user swings, so the data communications must be constrained to wireless communications that can be received by the laptop or other data processor's technology. These communications must also be constrained to a given data transmission speed floor (chosen to be $\geq 100\text{kbps}$), so that the AI can utilize a sufficient amount of data within the period of a user's swing action. Finally, in parallel to how the software's computer environment must be portable, the cameras for a video stream of a user must also be portable, as to allow a user to take the system to a desired location for training. The full list of constraints is as follows.

Software:

- Existing database as training reference.
- Video quality of $\geq 420\text{p}$ at $\geq 30\text{fps}$.
- AI must be containable in a portable computer unit.
- Computer must have a constant energy source.

Hardware:

- Total costs must be $< \$500$.
- IMUs must be secured on user and immobile with respect to user arm.
- User-mounted sensor system cannot restrict user swings.
- Wireless communication standards must be used and are receivable by computer.
- Video cameras must be portable and can directly connect to a computer.

2.5 Quality Analysis

2.5.1 User / Marketing Specifications

Affordability	$< \$500$ price point
Battery Life	2 Hour Lifespan
Feedback speed	$< 10\text{s}$ response generation time
Weight	< 5 pounds (On user)
Accuracy	$> 90\%$ feedback accuracy

Table 2: User/Marketing specifications of the AutoCaddie system that can be used to endorse the AutoCaddie system.

2.5.2 House of Quality

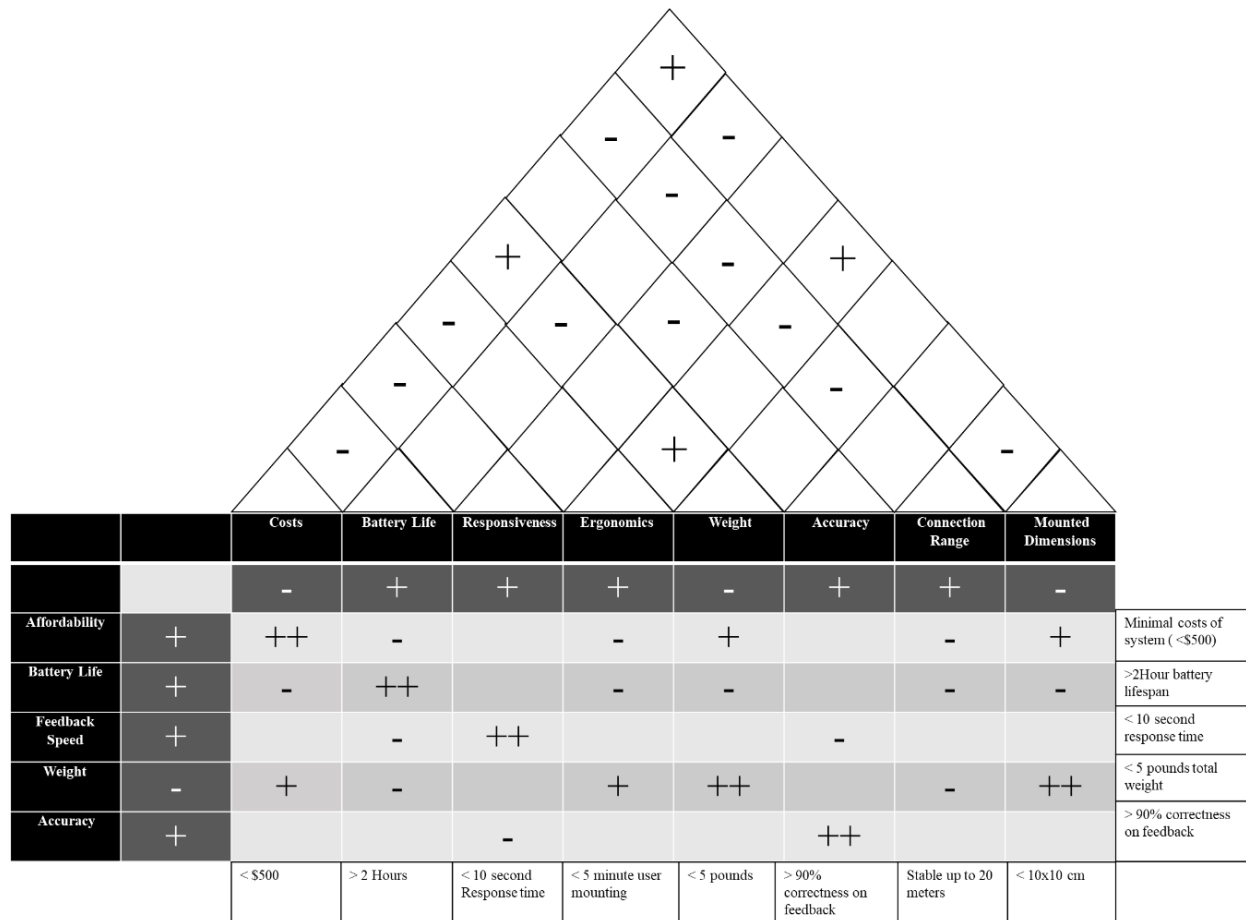


Figure 5: The house of quality analysis figure, demonstrating the relationships and overlaps between the engineering and user requirements and specifications.

As seen by the house of quality diagram, both the user/market specifications share some conditions, including that of battery life, weight, and accuracy. Regarding individual conditions of both user/market specifications and engineering requirements, both battery life and weight are the most influential. Increasing battery life requires either less powerful systems that draw less power, or requires the integration of a larger battery, thus increasing conditions such as weight and costs. Furthermore, reducing the weight of the system restricts the selection of both parts and overall design, possibly requiring the integration of more costly condensed application parts. Overall, most of the requirements or specifications are influenced by the hardware design.

Chapter 3: Project-Related Research

AutoCaddie is a project that demands knowledge of data acquisition's technical and physical aspects. Now that the goals, constraints, and high-level analysis are determined, it is possible to move forward with research toward the technology to be utilized by this product.

Before diving into specific research, it paints a stronger basis to reference previous examples of similar projects. By observing the methods and implementations of previous endeavors, we can shed light on some of the problems that may arise throughout the development process. Not to mention, it can help give a bit of creativity by introducing new perspectives to a similar topic.

As an overview, there are a few areas of interest that are worth exploring. AutoCaddie's primary selling point is automating the coaching process of golf and allowing the process to be more convenient. For the purpose of general information, it is worth investigating existing AI golf coach programs to determine different methods of implementation. For example, it is necessary to investigate if other products rely exclusively on software or introduce other physical means of recording data. Additionally, it is necessary to investigate technology that utilizes IMU data via physical body tracking to determine linearity or other gyroscopic data. Other useful examples of projects that can help gain primarily fall into the realm of neural networks, which will be explored later.

3.1 Existing Similar Projects and Topic Research

This project is not the first of its kind. There are other existing AI golf coaches that provide user feedback to help improve technique. Many of these products are sold at high prices, which takes away from the convenience of the system's design. To explore this topic in general, it is important to reference previous examples that have investigated similar topics.

3.1.1 *Smart Coaching*

Coaching is the method of providing feedback to a user for improvement in various tasks. This can be done with verbal instruction, replication, or, effectively, statistical analysis. The high-level explanation of AutoCaddie is having a virtual AI coach that provides useful feedback to the user in the form of processed data from a live video feed.

Generally speaking, a program, controlled by some governing method, is trained on data that is directly related to an area of improvement that a user is requesting. For golf, this can relate to driving down a field. For tennis, this can relate to an initial service motion. There are infinitely many examples of where it is applicable, but some of these products in action are as follows.

One example, coach2vec, [2] uses AI to understand the style of play for professional coaches in soccer. Its design gathers data from ball possessions, or periods where the ball is being controlled by the observed team and gathers a list of statistics collectively from the players to map similarities to specific coaches. For example, observing the

duration of possessions, on average, might signify a specific context of coaching style. The impressive aspect of this design is the usage of “k-Means” similarity to map soccer attributes to a matrix table and query specific details from coaches. In AutoCaddie, we need to establish a direct mapping between a specific training database to a live video feed. However, this process will not directly call for matrix decomposition due to the nature of the parameters. We can observe the behavior in a linear fashion and do 1-to-1 comparisons rather than query for related areas of interest.

Another example, Singularity-PA, [3] utilizes a large database of baseball games to enhance predictive modeling. The use of neural networks, trained on extensive pitch-by-pitch Statcast data from several years, allows for a more accurate and versatile approach to predicting plate appearance outcomes. The purpose of this program is to counter the limitations of previous calculation-based models by accounting for large amounts of history data that can be overlooked by mere equations. As a result, it ends up outperforming previous result-guessing techniques. The purpose of AutoCaddie does not stray far from this concept. It utilizes a neural network to produce more accurate data than a linear algorithm could produce by a frame-by-frame analysis. Due to the sheer amount of training data, it has access to a broader range of references to classify input data, as well as has stronger predictive capabilities towards the input actions of a swing.

Taking topics from both of these projects, it is important to utilize a neural network to achieve fast and accurate analysis of a video feed. Where algorithms can be limited by an absence of edge-case factors in a specific equation, neural networks shine by referencing large quantities of training data to predict results. Having the GolfDB database of videos allows for accurate coaching in golf that would be limited by normal analysis alone.

3.1.2 AI Golf Coaching

There are many existing smart golf coaching products, mostly composed of software-based assessments. Some examples of well-established products are ALFA Swing, [4] 18birdies, [5] and Sportsbox, [6] which are all mobile app golf coaches. Additionally, there are many examples of research papers pertaining to the topics of computer vision, data analysis, and neural networks which are used throughout these programs. Of the three listed, notably all of them utilize solely video capture to produce feedback. Many of these products rely on the convenience of recording and advising through the same app. Although mobile app development is a consideration, for the sake of the neural network in question, AutoCaddie is planned to connect to a computer program with stronger cameras positioned around the user for precise data acquisition. Something that is lacking in these mobile programs is the reliability of the data, as different cameras present different framerates and image quality. ALFA Swing requires an iPhone X [7] or better to get accurate information, whereas AutoCaddie does not require anything outside of itself.

However, AI Golf [8] is unique in its research on self-training. As mentioned earlier, coaching can be executed in a variety of ways, replication being one of the primary methods. AI Golf establishes this connection and expands by directly comparing the user's swing to a professional's swing. The main difference between it and AutoCaddie

is that it establishes its training data by combining a professional input motion sequence with output human poses to map to the user for specific feedback. AutoCaddie will preprocess incoming data into a specific form to match the training data for a direct comparison and provide predictive output based on the comparisons made to the database. However, an important topic worth referencing is discrepancy detection, or the ability of a program to identify differences between two separate feeds. In AutoCaddie, the preprocessing step needs to be capable of not producing discrepancies, but if so, it needs to be capable of recognizing that the video is of a different style rather than observed values being incorrect. Otherwise, false negatives can be generated.

In terms of hardware-based golf coaching, not as many products exist. SMARTGOLF AI Coach [9] is an AI golf coaching system that creates feedback by observation of data through sensors attached to the head of a golf club. It simulates a swing through the use of an interactive video game and provides personalized feedback in real time in terms of ball trajectory, club face, and advice for specific swings such as slices and hooks. Although this technology is useful for coaching with spin and club face, we believe that this is accomplishable with a rear-facing camera and computer vision analysis. Therefore, this type of hardware is unnecessary to the finished product.

3.1.3 Real-Time AI Analysis

On another note, it is necessary to utilize the correct neural network model in order to provide proper real-time analysis. There are multiple ways to train a system such that it is hyper-specific to recognizing select parameters, which is necessary to create a smart golf coach AI.

One example of this is the usage of “Reservoir computing” [10] with image pattern recognition, but it is useful in its property of short-term memory. Projects have been instigated under this concept, such as experimentation with low voltage dynamic memristors, which naturally responds to low-voltage electrical pulses to exhibit short-term memory effects. However, due to the nature of AutoCaddie, exhibiting a short-term memory system is not something that is necessary, since it will be observing all aspects of the data acquisition process and comparing it to a database. Nevertheless, understanding appropriate models and examples as to why specific use cases may yield unwanted results is important for the aspect of this project.

An application of real-time neural network processing is the Guitar Amplifier Emulation with Deep Learning. [11] This product explores the development and evaluation of deep neural network models for simulating audio distortion within guitar amplifiers. Notably, it explores two different models of interest, one being the recurrent neural network that was introduced in the last paragraph. The benefit of using this model is its boost in processing speed. However, a version of “WaveNet” was also utilized, which is a deep generative neural network used specifically for waveforms - which was developed by DeepMind. [12] It is designed to optimize hyper-parameters such as layer depth and activation functions. This model worked well when a large layer version was implemented. AutoCaddie aims to follow a similar model structure as WaveNet in that it can focus on specific hyper-parameters of a golf drive and provide convincing feedback. However, generative AI is a

stretch of a topic since it generates new data given input feeds, whereas the goal of this project is to evaluate the given data.

Closer related, another project introduces a deep convolutional neural network for profile analysis of big powder diffraction data - known as the Parameter Quantification Network (PQ-Net). [13] It can analyze live data from powder X-ray diffraction patterns from multi-phase systems and create predictions for relevant information such as scale factors, lattice parameters, among others. Most notably, the system's performance was evaluated across different test datasets, and it was concluded that the PQ-Net provides faster and more accurate analyses than previous algorithmic methods. The network itself is designed as a deep convolutional neural network, which is designed well for image-related tasks. AutoCaddie mirrors this concept very closely with analyzing live image data and providing insightful feedback, so this architecture is one that can be referenced during the design of the AI. Specifically, PQ-Net is trained via patterns with known parameters and learns to predict the expected behavior. Then, when given live data, the same process is used to create conclusions about the data. Many other low-level concepts from this project, such as minimizing the mean absolute error (MAE) loss function when training, incorporating dropout layers to prevent overfitting, and implementing deep ensembles are all important characteristics of improving performance in an algorithm as large as AutoCaddie's algorithm aims to be.

Overall, understanding differences between neural network models and their applications can help narrow down similar concepts to be used within generating the design. There is no correct method of creating a deep neural network, so long as it is trained correctly. That is where models such as PQ-Net shine, in that they are designed to optimize hyperparameters, and the mathematical methods can be reused within the AutoCaddie project.

3.1.4 Inertial Measurement Unit Joint Kinematics Modeling

Regarding the use of IMUs for the modeling of joint kinematics and angles between limbs as a supplement to AutoCaddie's artificial intelligence, there are several existing research articles and applications that exist, which highlight the benefits and existing methodologies for such practices. In the article by Versteyhe et al., [14] the focus lies on the utility of quaternions in biomechanics, offering a four-variable representation of orientation that avoids singularities and simplifies numerical procedures. The article outlines the background of the development of quaternions, which found themselves in use due to their improvements over Euler orientation vectors. Particularly applied in modern biomechanics, quaternions aid advancements in computer technology and motion capture by providing more robust means of calculations.

Versteyhe et al. sets the stage for the article by Challis et al. [15] where the development of a method using IMUs for knee joint kinematics is discussed. IMUs provide a portable and user-friendly alternative to traditional motion analysis methodologies. The method involves sensor calibration, placement on limb, and the application of quaternion math. Overall, Challis et al. showcases the practicality of IMUs in capturing joint kinematics

accurately. The validation experiments demonstrate the method's potential for clinical and scientific evaluations.

Adding to this narrative, Rahman et al. [16] introduces a wireless wearable sensor system utilizing IMUs and the Madgwick filter-based algorithm for joint angle measurement, focusing on the arms. The algorithm proves effective in trials, outperforming other systems and showcasing resilience to external acceleration. The study acknowledges limitations and points towards future directions, including real-world assessments and the development of a comprehensive wearable system for total body joint monitoring.

Together, these papers paint a comprehensive picture of the intersection between quaternions and IMUs in biomechanics. Quaternions, as highlighted by Versteyhe et al., provide a robust way to represent orientation, addressing challenges posed by other methods. IMUs, as discussed in the work by Challis et al. and Rahman et al., offer a practical solution for capturing joint kinematics, making these technologies a powerful combination.

The existence of these methods and technologies using IMUs to measure joint angles is driven by the need for accuracy, portability, and user-friendliness, crucial factors in applications such as rehabilitation or other applications requiring kinematic analysis. The advantages of quaternions, as mentioned by Versteyhe et al., align with the need for clear and unambiguous orientation representation. The IMUs, showcased by Challis et al. and Rahman et al., address the limitations of traditional motion analysis labs, providing a solution that can be easily applied in clinical settings.

The promising results from the wireless wearable sensor system designed by Rahman et al. further affirm the potential of these technologies for joint angle measurement in rehabilitation scenarios. The convergence of these advancements underscores the continuous evolution of methods and technologies, offering innovative solutions for precise and practical biomechanical measurements with applications ranging from rehabilitation to scientific research. Overall, the articles provide a good foundation for the intention in using IMU kinematics analysis using quaternions for measuring user movements of the arm.

3.1.5 Effects of Distractions and Stimuli During Swing Action

The research conducted by Leo Roberts et al. [17] on skilled golfers in this study sheds light on the impact of distraction and off-task thoughts on golf performance. While expert athletes have been shown to manage irrelevant thoughts during motor actions, this study focused on how such distractions during the preparation phase affect later performance. The results indicated that, generally, the golfers were able to maintain their performance levels. However, a notable finding was the control of distance for the shortest shots deteriorated when preparation was disrupted by secondary tasks. The analysis revealed that this interference led to both cognitive mistakes, such as poor decision-making, and execution mistakes, like poor timing. This suggests that off-task thoughts during the preparatory phase can lead to a range of errors by disrupting the organization of thought processes essential for effective action. Overall, this study uncovers an intricate

relationship between off-task thoughts and motor skill failure than previously recognized, emphasizing the importance of mental focus and preparation in golf performance.

Additionally, the findings of Timothy Fulton et al. [18] highlight the significant impact of visual distractions on golf putting performance and the protective role of a successful execution of Quiet Eye (QE) behaviors. The study revealed that longer QE periods were associated with more successful putting, indicating that anxiety can potentially undermine both QE and overall performance. However, the research showed that training in QE can act as a buffer against anxiety's negative effects on performance. Notably, the study also unveiled an additional advantage of effective QE behaviors – it mitigated the detrimental impact of camera flash distractions on putting performance. The elongation of the QE period during distracted conditions suggests that QE plays a pivotal role in the planning of motor actions, as supported by previous research. It is plausible that maintaining similar levels of putting performance in trials with and without distractions was due to the extended time provided to the brain for planning the putting action when distractions were present, underscoring the importance of cognitive focus in the face of visual interruptions during golf putting.

3.1.6 Multi-Modal Feedback Training

The project “Optimal Multi-Modal Sensory Feedback to Maximize Motor Learning Efficiency” led by Saba M. Hoesseini [19] integrates several forms of sensory input to analyze and provide feedback to a user and provides a collection of multi-modal feedback forms in order to study the effectiveness and response by a user. Intended for rowing sports applications, this system collects sensory data of user kinematics and speed of motion, as well as user force application on the machine during training. Following the processing of this data, the given metrics are used to analyze user performance and give multi-modal feedback of spatial, temporal, and force types.

This project serves as an example of sensor data collection for the provision of multi-mode feedback. As the project seeks to optimize motor learning efficiency during training, the outcomes and discoveries of this project can be applied to AutoCaddie. By incorporating relevant insight of the effects and means of multimodal feedback, whilst accounting for the optimal time for feedback of AutoCaddie users, more effective feedback and guidance can be designed for the users of AutoCaddie.

3.1.7 Putting It All Together

Overall, there are many existent technologies that give insight to some of the goals AutoCaddie aims to accomplish. Whether it be in terms of general coaching with AI assistance, custom-tailored golf feedback, or even optimizations via deep neural networks, there are many applicable examples. However, AutoCaddie will combine the best aspects of all of these areas in order to deliver the highest performance result.

Having the correct feedback given by an AI is necessary for the system to function properly. Creating the illusion that a system is personalized to the same level of feedback as a professional coach will help draw out the best results. A similar concept to this is the Turing Test [20] which is a physiological study proposed by Turing in 1950 in which a

machine passes if it can convince a person that it is not a program, but rather a real person. If AutoCaddie can establish accurate feedback on major components of the golf stroke, then the effectiveness of the product will rise.

This accuracy will be determinant on the effectiveness of the data analysis system. To boost results, AutoCaddie will implement a deep neural network that is trained on a collection of proper golf swing technique videos. By training a model on correct data, and developing consistent evaluation metrics, we can feed live data into the model and have it smartly calculate the given statistics and improvements.

As an additional means of feedback to supplement the feedback provided by the machine learning artificial intelligence model, IMU systems can be used to provide feedback regarding arm kinematics via quaternion data. Existing research and applications suggest that quaternion data from IMUs can be used to determine the effective kinematics and joint angles between limbs. These applications extend into fields such as rehabilitation and training examinations. As such, IMU quaternion data analysis can be used as a valid means of providing additional feedback to the users of AutoCaddie.

With respect to the implementation of the given feedback, the previous research in **Section 3.1.5** indicates that sustained cognitive focus is critical for performance, and that distractions can inhibit a golfer's ability during their swing preparation and swing action. As such, it can be inferred that forms of feedback, such as continuous stimulus to indicate correctable behavior during swing preparation may overall degrade the user's performance during training. Given this information, feedback should be reserved for intermediary stages between individual swings.

The strongest means of implementation can be via both hardware and software. Mainly, cameras can be used to capture action sequences of the user during a swing, in a similar fashion to existing phone coaching applications. However, having sensors also positioned on the user can help with synchronizing the data, as well as capture temporal data that may be less accurate taken from a 30-fps video stream. Therefore, AutoCaddie is targeted towards providing accurate data using the correct means of implementation.

3.2 Overall Structure and Relevant Technologies

3.2.1 System Design Overview for Technology Research

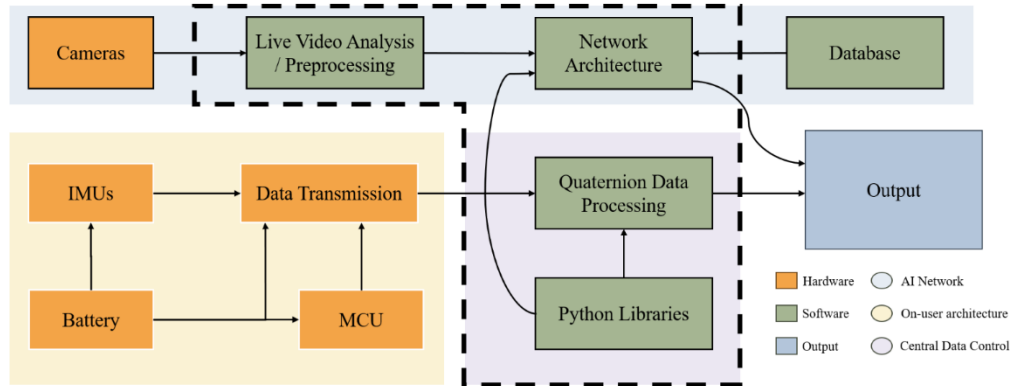


Figure 6: Diagram demonstrating the relationship between all researched components.

AutoCaddie comprises two main subsystems: the camera system and the IMU system. Each of these systems contains components of hardware and software and conjoin to provide the user with feedback pertaining to their golf swing.

The camera system captures live video data, which is then transmitted to a computer for further processing. The data undergoes frame extraction and preprocessing to prepare it for utilization within a neural network. This neural network is trained using a comprehensive database of proper golf technique swings, enabling it to analyze and evaluate the user's golf swing captured by the camera.

Simultaneously, the IMU system, which is affixed to the user, records spatial and rotation data in the form of quaternions. This system operates on battery power and transfers the collected data to a MCU. The MCU acts as an intermediary, forwarding the quaternion data to the computer with specific labeling. Within the computer, sophisticated quaternion data processing takes place, analyzing every timestamp to gain insights into the user's posture and kinematic data during the swing process.

The integration of quaternion data processing, neural network operations, and frame preprocessing is facilitated through various Python libraries, providing a cohesive and efficient environment for computational tasks. These libraries enhance the connectivity of the different components within the system.

Ultimately, the results derived from each system are compiled and presented to the user through a friendly graphical user interface. This serves as a comprehensive feedback mechanism, which offers insights for the user's golf swing performance and provides valuable information for skill improvement.

The overall architecture of the AutoCaddie system can be tailored in various ways to accommodate different user feedback mechanisms. Since AutoCaddie is designed with a machine learning interface for user feedback, the architectural variations primarily

revolve around the system's observation of the user and the impact of the user's actions. The primary architectural modifications involve the inclusion of specific components, particularly video cameras and user guidance components. Based on these considerations, three potential architectures for the AutoCaddie system are as follows:

3.2.1.a Single-Camera Video Feed

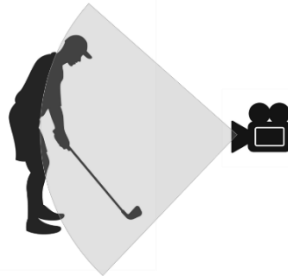


Figure 7: Graphical representation of single camera view target.

One of the most straightforward architectures for the AutoCaddie system involves utilizing a single camera input for capturing video footage of the user. This architecture, along with all subsequent architectures, will retain the IMU sensory system as is, employing wireless communication to interface with the artificial intelligence. The single-camera approach streamlines the process and reduces the volume of data to be processed, thereby enhancing processing speeds by reducing the volume of data that must be processed. However, the scope of data that can be analyzed by machine learning artificial intelligence is correspondingly reduced. In general, this architecture design prioritizes lower costs and optimal portability.

The prioritization of lower costs and optimal portability maximizes the potential scope of users that may seek to utilize AutoCaddie. By reducing costs, the financial barrier to entry is reduced, allowing more users to afford the system. Furthermore, the increase in portability reduces the necessary setup time for the system and allows the users to more easily take the system where they desire.

3.2.1.b Dual-Camera Video Feed

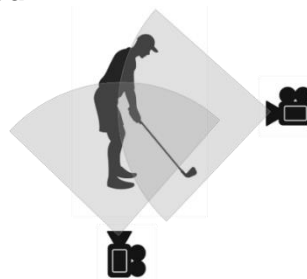


Figure 8: Graphical representation of dual camera view target.

The second architecture of the AutoCaddie system introduces two cameras for capturing video footage of the user. This enhanced setup provides a more comprehensive perspective of the user's swing motion, enabling the artificial intelligence to analyze the swing with greater depth and accuracy. However, this doubling of data inevitably increases the computational demands on the system, potentially leading to slower

processing speeds. Despite this drawback, the two-camera architecture strikes a balance between data availability and portability, offering a cost-effective solution for those seeking a more detailed analysis of their swing mechanics.

To further explore the advantages of this architecture, it's important to recognize the enhanced data acquisition capabilities. With two camera inputs, the system can capture a wider range of motion at different angles, allowing for a more thorough examination of the user's swing technique. This comprehensive data set enables artificial intelligence to identify patterns of movement and areas of feedback relevant to each viewport of the user, which a single camera setup could not achieve. This deeper level of analysis can provide users with more specific and actionable feedback, ultimately improving their swing performance.

3.2.1.c Dual-Camera Video Feed with Golfer's Training Ring

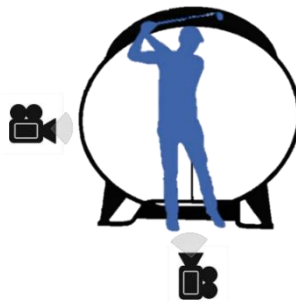


Figure 9: Graphical representation of a dual-camera setup with a training ring.

The third architecture of the AutoCaddie system builds upon the two-camera system, incorporating a training ring that serves both the user and the artificial intelligence. The user can utilize the training ring to ensure their swing pathing is correct and to develop proper muscle memory for swing path form. Simultaneously, artificial intelligence can analyze the user's path in relation to the ring and calculate variances to provide feedback. This architecture, while offering the most comprehensive user training, comes at the expense of increased cost and reduced overall portability.

Despite the advantages of a more robust training methodology using the training ring, this architecture does come with some drawbacks. The inclusion of a training ring increases the overall cost of the system, making it less accessible to some users. Additionally, the additional hardware component reduces the system's portability, making it less convenient for users who require a portable training solution.

3.2.1.d Architecture Selection

For the purposes of the AutoCaddie system design, the dual-camera video feed system will be selected. This system will provide the best balance of data quality to portability. The dual video feed system will provide the system with two perspectives of the user for data analysis, allowing for the system to provide analysis for multiple aspects of the user's performance. Regarding the lack of the implementation of the golfer's training ring, while the ring does assist the user in developing positive swinging habits, the ring only interacts

with the software as an additional mode of comparative analysis and does not produce any form of unique feedback. Furthermore, its inclusion would also increase costs, and would require the transportation of more materials. Regarding the inclusion of video cameras, the selection of video camera technology must be made in order to maximize the efficiency of the machine learning processing schema, as well as to best fit the video data to matched data.

3.3 Relevant Technologies: Computer Vision and Neural Networks

Video data streams play an important role as essential inputs for machine learning algorithms, particularly in applications involving real-time analysis of dynamic events. The establishment of a robust connection between frames extracted from live video data and frames sourced from a separate database enables the seamless integration of real-time information through an artificial intelligence system. In this context, cameras leverage the capability of comparing data keyframes extracted from an ongoing video data stream available in a training dataset. The feedback generated by artificial intelligence is a result of these intricate comparisons, facilitated by the pre-existing knowledge gained through prior training experiences. This process allows for the continuous and automated analysis of live events, showcasing the practical application of machine learning in extracting insights from video data.

3.3.1 Video Cameras

3.3.1.a Overview, Comparison, and Selection of Component Type

Video cameras have been chosen as the primary motion capture system for the AutoCaddie project's machine learning analysis. While lidar and ultrasonic sensor technologies are available and can be utilized for capturing user motion and posture at discrete moments, these technologies are not aligned with the overall requirements of AutoCaddie's design systems. Lidar systems rely on point-based 3D imaging, but the lidar scanning process is considerably slower when compared to video imaging, which does not meet the timeliness requirements of machine learning applications.

Ultrasonic sensors can be employed to determine the general movement and positioning of objects within a 3D space; however, the resolution of potential spatial imaging provided by ultrasonic sensors does not meet the precision demands for data analysis. Additionally, both lidar and ultrasonic sensor data will necessitate extra processing and produce data formats that are not supported by existing databases. Consequently, the preferred choice for data acquisition in the context of machine learning systems is video cameras that allow for generic video streaming, as they align more closely with the project's requirements for real-time, data capture and compatibility with existing databases, where keyframing must be used.

The most critical characteristic for the cameras to meet is live data streaming capabilities. The video data stream must be able to be streamed directly to the machine learning AI program for it to process the video stream data. For the video data streams to be effective, the video data streams should be at a minimum of 420-pixel (p) resolution at a 30 frames per second (fps) speed. The portability of the cameras must also be a feature, as the AutoCaddie system must be able to be set up at a region of the user's preference. Furthermore, the costs of the cameras must be minimized to reduce the market barrier costs for user access.

3.3.1.b HP 320 FHD

The HP 320 FHD digital webcam device with CMOS photosensor technology. With a 0.25in lens, the camera supports a 1080p resolution at a 30fps speed. Furthermore, the camera uses USB-A connectivity, so wired streaming is stable and limited power consumption is a non-issue. This camera is intended for webcam streaming purposes, so it is low-profile, and can be retrofitted for live streaming to an AI program. The total cost of this camera comes to \$28.51.

3.3.1.c RunCam 2 FPV

The RunCam 2 FPV camera is a digital sports action camera. The camera uses CCD photosensor technology and uses a 1x optical zoom magnification. The camera supports a 1080p resolution at 60fps. As the camera is marketed for mobile robotics purposes, it is inherently portable. Further, the camera features Wi-Fi connectivity, allowing for a wireless video stream to a computer. The total cost of the camera comes to \$99.99.

3.3.1.d WOLFGANG GA100

The WOLFGANG GA100 is a digital sports mounting camera. The camera uses CMOS photosensor technology and uses a 1x optical zoom magnification with a 170-degree viewport. The camera supports a 1080p resolution at 60fps. As the camera is marketed for mobile dashcam and webcam purposes, the camera features Wi-Fi connectivity, allowing for a wireless video stream to a computer. The total cost of the camera comes to \$49.99.

3.3.1.e Hardware Comparison Table

Factor	HP 320 FHD	RunCam 2 FPV	WolfGang GA100
Connectivity	Wired USB-A	Wi-Fi	Wi-Fi
Resolution	1080p	1080p	1080p
Speed	30fps	60fps	60fps
Portable	Yes	Yes	Yes
Cost	\$28.51	\$99.99	\$49.99

Table 3: Hardware comparison table that compares video camera candidates.

3.3.1.f Conclusion

The selected video camera for the AutoCaddie system is the HP 320 FHD. This video camera was chosen due to its connectivity and cost characteristics of the device. The HP

320 FHD is roughly one-fourth the price of the RunCam 2 and is roughly one-half the price of the WolfGang GA100. This optimization of price reduces the total costs of the system, making it more available at market entry price for the user. Furthermore, regarding connectivity, a USB connection to a computer processing the video guarantees a more stable and faster data stream, as well as the fact that the camera does not need to rely on limited battery power. The expected video resolutions of each video camera are identical, and the limited 30fps video speed quality does not pose an issue for the AI processing algorithm, as it the machine learning based AI will be trained off of an existing database of videos of similar frame speeds. Should more keyframes be needed, frame interpolation algorithms can be implemented.

The video camera systems should both feed into the central computer where the primary data processing will occur. The computer system should be able to interface with both of the camera video feeds. These two video feeds must then be consolidated to a single processing schema of the machine learning algorithm, from which the video feed is analyzed with respect to an established analyzation process.

3.3.2 Live Video Analysis

To extract valuable insights from the video streams, real-time video analysis plays a crucial role in AutoCaddie. The system relies on capturing a continuous stream of live data covering a user's golf swing, which is then segmented into distinct actions. This segmentation is a fundamental step, as it allows for focused analysis of specific moments within the swing sequence.

Following the segmentation process, various preprocessing techniques are applied to the data. These techniques are essential for preparing it for transmission to a neural network, ensuring that the input data is in a suitable format for accurate analysis. The preprocessing steps may include tasks such as noise reduction, image enhancement, and normalization. By refining the data through these methods, the system can enhance the quality and reliability of the subsequent analysis.

While the intricate details of the neural network architecture will be discussed in **Section 3.3.3**, it is important to emphasize the methods used to ensure the accuracy of its calculations. This accuracy is pivotal in generating precise insights from the golf swing data. Notably, these efforts are essential to guarantee that the coaching and feedback provided by AutoCaddie are based on dependable analyses of the user's swing, contributing to an enhanced overall user experience.

3.3.2.a Frame Extraction

For AutoCaddie to achieve successful video analysis, it needs to be capable of frame extraction. In digital video processing, a video is a sequence of individual frames displayed in rapid succession, which creates the illusion of continuous motion. Each frame is an image that captures a specific instance of time. In the case of AutoCaddie's live video feed, these frames capture images that represent the user across all instances of time during a golf swing.

The process of converting videos into frames involves breaking down the video file into individual images. Each frame represents a distinct snapshot of some action, enabling precise examination of the golfer's movements. These frames are then processed and analyzed using various techniques to extract information on swing trajectory, form, among other things. By dissecting the video stream into frames, AutoCaddie can precisely pinpoint the moments that matter, which helps for accurate assessment of the motion.

The process of capturing each frame separately from a video stream facilitates the application of preprocessing algorithms on visual data. Each frame within a video can be represented as a 2D matrix of pixels, with each pixel assigned a numerical value corresponding to its color intensity. This representation enables algorithms to work with visual data in a format conducive to manipulation.

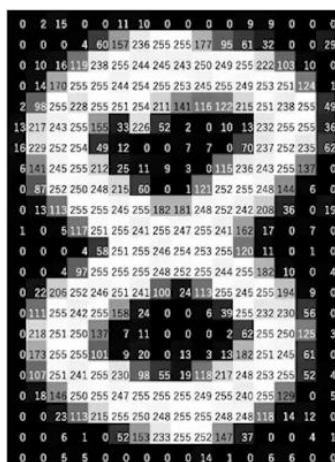


Figure 10: An image represented as a 2D array of numbers. The value in each matrix position represents the grayscale equivalent on a scale of 0 to 1. Referenced via [21].

In this 2D matrix, the rows and columns correspond to the height and width of the frame respectively. Each element in the matrix represents the exact hexadecimal color equivalent of the corresponding pixel in the image. Hexadecimal is represented in a 16-bit format, where every group of 4 bits (or 1 byte) are equivalent to a specific color value. The first 3 bytes compose the levels of Red, Blue, and Green, where the last byte represents the transparency of the image.

By breaking down the visual information into these individual pixels and numerical values, algorithms can perform various operations on the data. In the context of AutoCaddie, these operations include object recognition, and feature extraction.

3.3.2.b Computer Vision

The term given to breaking down visual information from pixels within images in regard to pattern detection is computer vision. Computer vision is pivotal for the success of AutoCaddie, as it can drastically increase the efficiency of the software. At its core, it involves the extraction of meaningful information from frame data. Expanding, many different algorithms and techniques can be utilized to interpret entire video streams. In

this section, the fundamentals of computer vision will be elaborated upon, as well as its transformational potential.

Object detection is a primary research direction in the field of computer vision. Literally, object detection is the process of applying algorithms to an image to distinguish a particular object against its background. This process used to involve traditional detection algorithms, involving pre-processing, window sliding, feature extraction, selection, classification, and post-processing. However, it yielded high complexities and time investments to complete this process and produced results with no pertinence.

In 2012, Krizhevsky [22] proposed an image classification model harnessing a convolutional neural network (CNN). This model, known as ImageNet, was subjected to rigorous evaluation in an image classification competition, where it demonstrated its superiority over traditional object classification methods by surpassing them by over 11% accuracy. This served as a catalyst for extensive research into deep CNNs for classification tasks.

Moving on, in recent years, the field of computer vision has shifted to this deep-learning approach. Three prominent categories of deep learning models have played large roles in reshaping this industry, namely CNNs, the “Boltzmann family” which includes both Deep Belief Networks (DBNs) and Deep Boltzmann Machines (DBMs), and Stacked Autoencoders (SdAs). Each category exhibits distinct characteristics, advantages, and limitations.

First, as referenced by A. Voulodimos et al., [23] CNN is a prominent topic in deep learning, especially for dealing with grid patterns in images. CNNs are designed to autonomously find spatial hierarchies of features, and further to distinguish patterns among them. The architecture of a CNN is constructed upon sophisticated mathematics, and features three fundamental building blocks: convolution, pooling, and fully connected layers.

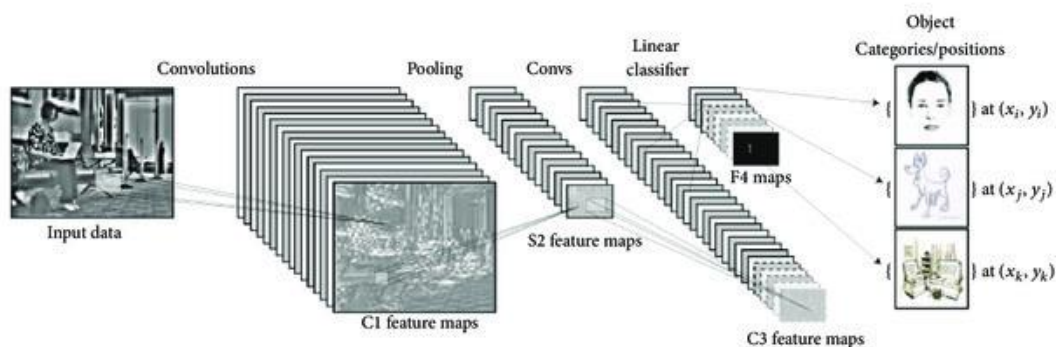


Figure 11: An example of a CNN used for object detection within an image. The image is split into different convolutions by extracting features. Then everything is pooled to refine the parameters. After a while, the features are classified. Referenced via [23].

The convolution layer takes on the task of feature extraction. By representing an image as a 2D grid, an additional small grid of parameters, known as a kernel, is applied at every position within the image using matrix multiplication. This operation renders CNNs adept at image processing because features can manifest at any location within the image grid.

The pooling layer works synergistically with the convolution layer to refine and abstract these features. The pooling layer operates by down-sampling the spatial dimensions of the input data, reducing its resolution. This is a critical step for preserving essential information while progressively simplifying the operation.

The fully-connected layer is the culminating phase of the CNN architecture. It is responsible for mapping the features into a conclusive output, often in the form of classification. Through this layer, the CNN transforms its hierarchical understanding of spatial patterns into a final decision or classification.

A primary benefit of CNNs is the fact that they learn their features from the dataset provided, removing the need for manually crafted features. Their significance is showcased through object detection. However, a drawback of these networks is that they have a heavy reliance on labeled data for training, which introduces data preprocessing steps.

Second, the “Boltzmann” family of models presents a different approach to hierarchical representation learning. DBNs excel in unsupervised learning scenarios, obviating the need for labeled data during the initial training. However, their computational cost and limitation in handling two-dimensional structures in input images pose challenges.

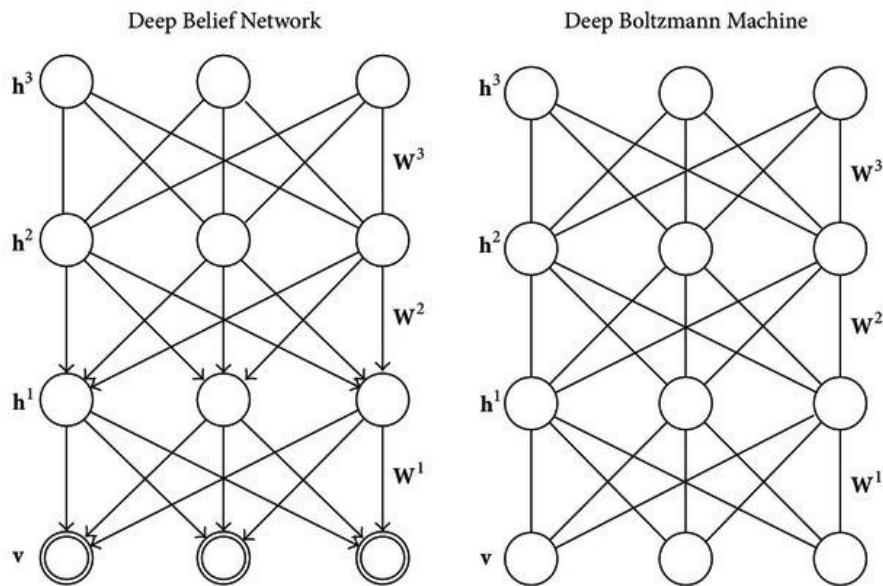


Figure 12: Deep Belief Network (DBN) and Deep Boltzmann Machine (DBM). In the DBN, the top two layers form an undirected graph. In a DBM, every connection is undirected. Referenced via [23] .

At the center of both DBNs and DBMs is the foundational Restricted Boltzmann Machine (RBM). RBMs are generative stochastic neural networks. They form bipartite graph, enabling more efficient training algorithms such as gradient descent. RBMs excel in extracting intricate hierarchical representations from data.

DBNs are probabilistic generative models that utilize RBMs as a layered structure. DBNs initialize their deep network efficiently by progressively refining hierarchical representations through stacking RBMs. This facilitates the extraction of complex features from the training data. Thus, this addresses the manual challenges of parameter selection, and eliminates the need for labeled data during the training process. However, DBNs encounter computational challenges and may overlook the two-dimensional structure of input images.

DBMs differentiate themselves by featuring undirected connections between all layers of the network. This is different from DBNs, where the top two layers form an undirected graph. The distinct advantage of DBMs lies in their ability to capture intricate representations of input data across multiple layers. Additionally, they are suitable for both unsupervised and supervised learning, making them flexible with diverse tasks. Notably, DBMs are set apart in the ability to handle uncertainty within inputs. However, similar to DBNs, a drawback of DBMs is their large computational cost.

Third, SdAs utilize autoencoders as their foundation. An autoencoder is trained to encode input data into a representation that facilitates accurate reconstruction. The training process minimizes reconstruction errors, ensuring that the output closely mirrors the input. The essence lies in learning features from the data, either by projecting inputs into principal components or, in the case of nonlinear hidden layers, capturing multimodal aspects of input distributions.

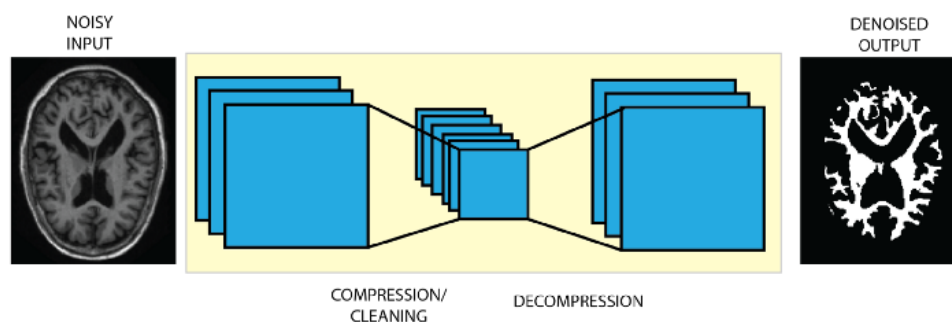


Figure 13: A diagram of denoising an image using a SdA. Referenced via [24].

Denoising autoencoders introduce a corruption process to the input, challenging the model to encode the uncorrupted input while undoing the effects of the applied corruption. This stochastic approach involves predicting corrupted values from uncorrupted ones after the data has been applied with noisy transformations. Thus, this becomes a powerful tool for unsupervised pretraining of deep architectures.

SdAs create a deep network by feeding the latent representation of the layer below as input to the current layer. The unsupervised pretraining occurs layer by layer, with each

layer trained as a denoising autoencoder. After the process is completed, the network undergoes fine-tuning, which is often supervised by the developer. While DBNs often exhibit superior performance, comparative studies reveal instances where SdAs prove to be competitive. Plus, the adaptability of autoencoders in layer parametrization stands out compared to the other models.

To directly compare these three deep-learning approaches to applying computer vision for object detection, the chart below summarizes each model's advantages and disadvantages.

Model	Advantages	Disadvantages
CNN	<ul style="list-style-type: none"> • Feature Learning: CNNs excel at automatically learning hierarchical features from data, making them highly effective for image-related tasks. • Spatial Invariance: The convolutional and pooling layers provide spatial invariance, allowing CNNs to recognize patterns irrespective of their position on the input. • Pretrained Models: Pretrained CNN models on large datasets offer transfer learning advantages for specific tasks, requiring less training data. 	<ul style="list-style-type: none"> • Computational Demands: Training CNNs can be computationally intensive, demanding substantial resources. • Interpretability: The internal representations learned by CNNs may lack interpretability, making it challenging to understand how and why certain features are detected.
DBN / DBM	<ul style="list-style-type: none"> • Unsupervised Learning: DBNs and DBMs can perform unsupervised learning, leveraging unlabeled data for feature extraction before fine-tuning for specific tasks. • Capturing Complex Representations: These models can capture multiple layers of complex representations, making them suitable for hierarchical feature learning. 	<ul style="list-style-type: none"> • Computational Cost: Inference in DBNs and DBMs can be computationally expensive, especially in joint optimization for large datasets. • Limited Applicability in Computer Vision: DBNs may struggle with the two-dimensional structure of images, affecting their performance in computer vision tasks. • Challenges in Optimization: Optimizing DBNs involves challenges, such as potential poor local optima and difficulties in further optimizing the network based on maximum likelihood training approximation.
SdA	<ul style="list-style-type: none"> • Flexible Parametrization: Autoencoders allow almost any parametrization of layers, offering flexibility in architecture design. • Unsupervised Pretraining: SdAs can be pretrained in an 	<ul style="list-style-type: none"> • Not a Generative Model: Unlike generative models like RBMs and DBNs, SdAs do not correspond to a generative model, limiting their ability to

	unsupervised manner, making them adept at learning hierarchical representations from unlabeled data. <ul style="list-style-type: none"> Competitive Performance: Stacked Denoising Autoencoders can demonstrate competitive performance, particularly in comparison with Deep Belief Networks. 	draw samples for checking learning outputs. <ul style="list-style-type: none"> Reconstruction Error Limitation: The optimization process in SdAs focuses on minimizing reconstruction error, which may not guarantee successful compression for all inputs.
--	---	--

Table 4: A direct comparison between three separate deep-learning models applied towards object detection using computer vision.

Overall, CNNs demonstrate superior performance in meeting the specific requirements of the AutoCaddie project. Notably, CNNs excel in image processing tasks, precisely suited for the analysis of video frames to identify users and track their golf club swing sequences. Their robustness in handling spatial features removes the requirement for precise camera placement, so long as the user remains within the image dimensions.

In contrast, the limitations of DBNs, DBMs, and SdAs render them unsuitable for the project's objectives. Both DBNs and DBMs exhibit suboptimal performance when analyzing 2-dimensional image data, a critical requirement for precise image analysis. Additionally, the non-generative nature of the SdA approach renders it useless for predictive calculations essential in tasks such as object detection.

On a separate note, the adoption of CNNs creates the necessity to implement data preprocessing algorithms for image labeling and optimization. The following sections explore comprehensive research on the video preprocessing algorithms integrated into this system. Thorough investigation of existing technologies is imperative for this project's success, ensuring that the collected data can be analyzed with precision and performed quickly.

3.3.2.c Depth Tracking

Firstly, by pivoting two distinct cameras around the user during the swing sequence, we can gather data comprehensively. However, as the cameras were selected, additional research has been conducted on the importance of depth tracking within the video feed. Depth tracking refers to the process of capturing the three-dimensional positions of specific points within data or a given space. However, within the scope of the AutoCaddie project, where computer vision algorithms can detect a person and golf club, this technology is not necessary. However, it still provides useful investigation to understand the technology.

Soonchan Park et al. [25] performed a depth tracking technique that involves the implementation of depth tracking cameras at a hardware level to establish key points throughout the recorded swing sequence. At a high level, this method utilizes pose analysis, where the depth tracking cameras precisely track the movements and positions of various body parts during the sequence. This technique utilizes the Swing Analyzer (SWAN) system for pose analysis in golf. It uses a depth camera to estimate the 3D

positions of various joints during the swing, which contributes to a novel human pose estimation algorithm. This incorporates regression techniques to accurately localize 14 joints within the golfer's body. However, a concern presented by the study is that the depth tracking within the camera contributed to slower recording speeds, which did not coincide well with the fast swing of golfers.

Within the scope of the AutoCaddie project, comprehensive recording of the entire swing sequence is an established requirement. **Section 3.2.2** specifies that the video camera needs to possess the capability to maintain precise synchronization between the recording speed, measured in frames per second, of the live video feed and the selected training video database. This is necessary to ensure consistency and accuracy in the analysis of the captured swing sequences.

Additionally, it is important to recognize the existence of other technologies that render this research obsolete in this scope. Particularly, the advancements in action sequencing and computer vision, which will be elaborated on in consequent sections, provide more accurate results when analyzing the user data. Therefore, after careful consideration, it is not necessary to provide depth tracking within the video feed, as it is an extraneous mechanism that does not contribute extra information to the system.

3.3.2.d Edge Detection

Another popular research topic with neural networks is edge detection. Edge detection involves identifying and locating edges or boundaries in images. In the context of AutoCaddie, this is important in the context of preprocessing data frames in order to increase performance of the neural network. Being able to detect the outline of a person and separate them from an image background can help reduce the number of pixels being compared through the convolutional process. According to B. Tian and W. Wei, [26] there are various algorithms and techniques used for edge detection, ranging from classical methods to advanced deep learning approaches. The following paragraphs expand on some of these techniques, and weigh factors that attribute each method to this project.

First-order and second-order edge detection operators play a pivotal role in identifying small changes within an image. Some examples of this include between objects and backgrounds, separate regions of the same background, and primitive elements. These operators fall into two categories, first-order and second order. First-order operators focus on finding the derivative of the image's gray change, which makes them highly sensitive to drastic changes. An example of this is the Canny algorithm, which offers versatility by giving resilience against noise interferences and has the ability to identify genuine weak edges.

Traditional edge detection is a sub method of image segmentation. This topic can be split into several subjects. Firstly, edge detection involves extracting edge pixels, whereas edge enhancement focuses on intensifying the contrast between the edge and its background, enhancing the visibility of the edge. Additionally, edge tracing is a technique employed to collect edge pixels into a coherent list. The definition of an edge can vary based on specific contexts, with the ideal step edge being one of the most used

definitions. The process of edge detection hinges on identifying grayscale changes within an image, rooted in physical processes such as geometric disparities or optical phenomena. These complexities make subsequent data extraction challenging, especially when the image data is marred by noise. The extensive nature of numerical differentiation lengthens this process, where minor input signal changes lead to significant output signal differences. Balancing the elimination of noise and accurate edge location becomes a delicate and contradictory task in edge detection. Therefore, using these traditional edge detection methods can jeopardize the accuracy of the results.

Separately, in 2017, Wang et al. developed an advanced method for detecting cracks in images. They overcame the limitations of existing techniques by utilizing a combination of downsampling and normalized cuts at multiple scales. This method employed a specialized wavelet transform to efficiently extract edge features. By integrating the strength and location characteristics of different scales, they created similarity matrices. These matrices were processed, leading to down-sampled eigenvectors (in the context of this project, it is only worth noting that these vectors allow for complex data operations to be highly simplified due to matrix properties). These vectors were then refined and transformed into a final result. The approach significantly improved accuracy while reducing computation time, as demonstrated through experiments on various datasets.

In 2020, Wang et al. further enhanced their method by integrating a full convolutional neural network with structured forests. They designed multiple neural networks to capture intricate crack patterns and introduced a strategy based on multiscale structured forests to differentiate small cracks effectively. This combined approach demonstrated impressive results in accurately detecting cracks in complex backgrounds. Through existing libraries, which will be explored in **Section 3.5.5**, these algorithms can be implemented through minimal lines of codes.

3.3.2.e Frame Standardization

On a separate note, for a CNN to maintain its effectiveness, the input data must be presented in identical format. CNNs' effectiveness is attributed to their ability to autonomously learn visual features without manual interventions. But, similar to an autonomous machine, it expects an input with specified dimensions in order to apply matrix multiplication and other algorithms to the data. Thus, the input data stream presented by the video cameras needs to be standardized.

It has already been researched that frames can automatically have bounding boxes applied over the golfer and club during the entire swing sequence. At this point, each of these images can be cropped to the bounding dimensions, which will be random by nature. Thus, to standardize the dimensions of each of these frames, images need to be enlarged. To address these filter border issues, padding techniques are employed. This section will compare the two most common methods: zero-padding vs. scaling with interpolation. [28]

Zero-padding involves enlarging smaller images to a fixed size by adding zero-value pixels around their borders. This approach offers two notable advantages. Firstly, it

prevents the risk of deforming patterns in the image during the resizing process, ensuring that the original features remain intact. This is critical for image classification where maintaining the integrity of visual patterns is necessary for accurate results. Secondly, zero-padding is less computationally expensive than scaling with interpolation. Zero-padding simply inputs zeros along the image grid's border rather than perform neighboring calculations on every missing pixel.

Additionally, possible concerns about this method can be debunked. There is a common assumption that the CNN won't adequately learn features from the border areas due to the zeros inserted along the border of the image. This is incorrect because CNNs contain a weight-sharing property, where the convolutional layers employ synaptic weights across all of the matrix multiplication windows, ensuring equal feature extraction capabilities for both the central and border areas. Another worry about this method is that the zeros will produce adverse effects on weight optimization. This is also debunked through a long mathematical proof that highlights that zero-value pixels remain inactive during the propagation techniques. In other words, the zeros do not directly affect the features being extracted and remain inactive until they are filtered.

Scaling with interpolation is the conventional approach to resize smaller images. This method involves increasing the resolution of the image to match the desired fixed size. However, this process carries inherent challenges. First, scaling poses the risk of deforming features or patterns across the entire image, which would directly impact the accuracy of the remaining classification tasks. Interpolation attempts to fix this problem by smartly assigning pixel values in relation to nearby ones, but the inherent sloppiness of scaling remains a concern. Second, scaling is a more computationally expensive process than zero-padding, as it requires a small algorithm to determine what a predicted pixel value will be. Higher-resolution images occupy more memory space as well, which amplifies the complexity of the process.

Therefore, it is wise to select zero-padding as the proper frame standardization technique. This way, there are no present deformations within the input images as they are presented to the network, and the data is only consistent of the golfer and a club during the swing sequence. Using this method, the precision of the program can be maximized.



Figure 14: A small image being enlarged using two separate algorithms. (a) represents the original image. (b) represents the image resized using zero-padding. (c) represents the image resized using interpolation. Referenced via [29] .

3.3.2.f Action Detection

The last important detail of image preprocessing with AutoCaddie is action detection (also known as feature extraction). Being able to preprocess the data so that it is presentable to the network to be as accurate as possible is one thing. However, being able to recognize specific actions in a swing is another. Action detection is the term given to computer vision algorithms that detect specific motions, sequences, or patterns across a collection of frames that denote a specific action. In terms of the golf swing, this can be attributed to the address, backswing, impact, among others.

The research conducted by [30] delineates potential actions analyzable within a parent swing, categorizing the golf swing into eight distinct parameters:

1. Address: The frame before the initial movement begins.
2. Toe-up: The backswing stage where the golf club shaft is parallel to the ground plane.
3. Mid-backswing: The stage where the golfer's arm is parallel to the ground plane during the backswing.
4. Top: The stage when the golf club changes directions from the takeback to the downswing.
5. Mid-downswing: The stage when the golfer's arm is parallel to the ground during the downswing.
6. Impact: The stage when the golf club head hits the golf ball.
7. Mid-follow-through: The stage when the golf club shaft is parallel to the ground plane during the follow-through.
8. Finish: The frame before the golfer finishes their swing.

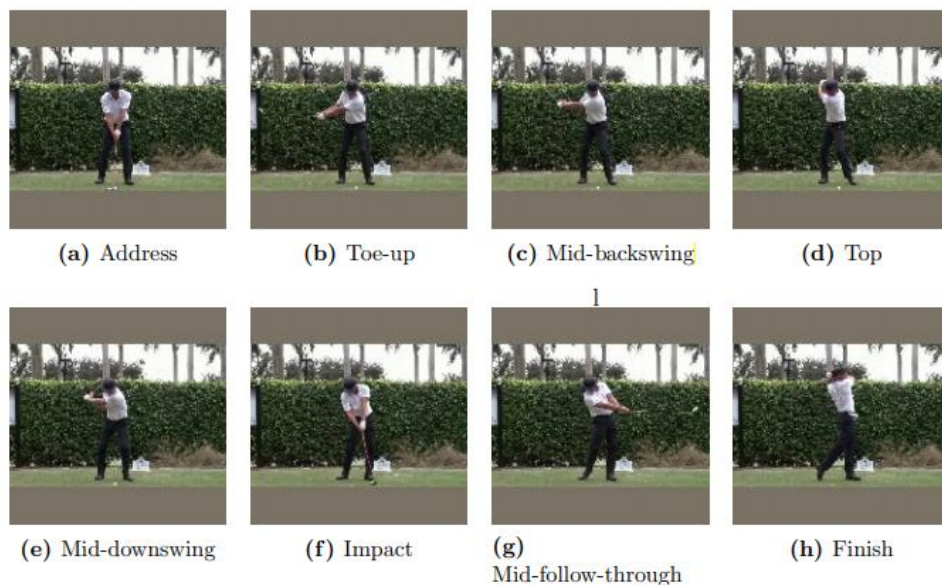


Figure 15: The eight actions within a complete golf swing. Referenced via [30].

Classification in the golf swing context involves labeling specific instances of time corresponding to predefined events. Due to the fact that there are eight distinct events

being labeled within this golf swing sequence, this problem is referred to as multi-class classification. CNNs, and deep learning neural networks in general, focus on accurately classifying input data to output features. AutoCaddie integrates this concept by offering coaching based on a user's swing, allowing for a more granular analysis of smaller, precise events.

3.3.2.g Summary

There is much consideration towards properly analyzing live video data from camera input streams to a computer. Analyzing live video data in AutoCaddie involves parsing the video data into image frames, where each frame represents a 2-dimensional grid of pixels, suitable for computer vision techniques. CNNs were identified as the most effective approach for image analysis in this context.

Standardizing input data is essential for error reduction. After marking a bounding box, images are cropped to match its dimensions and then expanded using the zero-padding algorithm to achieve a standardized dimension. This standardized format ensures that the CNN can accept images of proper dimensions, streamlining the process.

Overall, being able to differentiate different segments of a golf swing into separate actions is critical for the functionality of this neural network within AutoCaddie. The purpose of this project is to provide a helpful insight to the user about their stroke, which can be brought down to analyzing each individual action composing the entire swing. If these algorithms can effectively standardize the data being presented, we can eliminate external errors being introduced into the system. From this point, this data can be presented to the network, which will perform comparisons of each feature to a version with proper technique and present similarity scores to be formatted for the user.

3.3.3 Network Architecture

The network architecture for the AutoCaddie project is a critical component designed to process and analyze golf swing footage effectively. Our chosen architecture leverages the strength of convolutional neural networks (CNNs) [31] which are particularly adept at handling image and video data. The following sections detail the rationale behind selecting CNNs, their relevance to golf swing analysis, and how they integrate with the other components of our project.

3.3.3.a Selection of a Neural Network Trained on GolfDB

The AutoCaddie system will employ a neural network architecture trained specifically on the GolfDB dataset to classify the actions observed in golf swing footage. This choice is made with the intention to detect and highlight various aspects of a golfer's swing, comparing it against a repository of professional swings to identify potential areas for improvement. The GolfDB dataset provides a rich source of labeled video footage, offering a diverse array of swings from different angles and under various conditions. This depth and variety enable the training of a robust model that can generalize well across different golfers and swing styles.

3.3.3.b Why CNNs Excel in Video and Image Recognition

CNNs have been established as the de facto standard in the field of image and video recognition due to their ability to autonomously learn spatial hierarchies of features from visual data. [32] Unlike other neural network architectures, CNNs are designed to map image characteristics directly to outputs, making them highly efficient for tasks such as image classification, object detection, and video analysis. The layers within a CNN—convolutional layers, pooling layers, and fully connected layers—work in concert to process pixel data and extract meaningful patterns.

In the context of golf swing analysis, CNNs can recognize and learn from the intricate features of a swing, such as the posture of a player at address, the angle of the club at the top of the backswing, and the position of the body at impact. For example, the 'top of the swing' event is a critical phase where the golfer's coordination, club position, and balance converge. CNNs can discern this event by learning the commonalities found in the GolfDB footage, despite variations in the golfer's appearance, background, or lighting conditions. By focusing on these learned features, the network can provide feedback on whether a user's swing deviates from the ideal models in the database and suggest specific areas for improvement.

3.3.3.c Integration of CNNs with AutoCaddie Project Topics

The application of CNNs within the AutoCaddie project is not limited to simple classification tasks. It extends to a comprehensive understanding of golf swing mechanics, where the network must recognize and evaluate complex sequences of movements. The architecture is designed to correlate specific swing phases with the associated biomechanical outcomes, such as ball trajectory and speed.

The network's capability to analyze video data frame-by-frame allows for an in-depth temporal analysis of the swing. It will enable the AutoCaddie system to observe the golfer's motion across the entire sequence of the swing, from the initial stance to the final follow-through. This sequential analysis is crucial for identifying timing issues, such as early or late impacts, which can significantly affect the swing's efficacy.

Furthermore, the adaptability of CNNs to different data inputs makes them suitable for implementation in a real-world environment where swings are recorded under varying conditions. By training the network with GolfDB's comprehensive dataset, which includes diverse environmental settings and camera angles, our system will be robust against overfitting to a specific setting or view type.

3.3.3.d Exemplification of CNNs in Golf Footage Analysis

The utility of CNNs in the AutoCaddie project is exemplified through the analysis of golf swings using GolfDB. The database labels eight key events within a swing, allowing the network to target these events for a focused assessment. For instance, when analyzing the 'impact' phase, the CNN can precisely detect the club's position relative to the ball at the moment of contact, a factor directly tied to the shot's quality. The multi-layered

structure of a CNN can process the video data through successive filters, each designed to highlight different features such as edges which are fundamental in distinguishing between the nuanced positions of a golf swing.

3.3.3.e Balancing Precision with Performance in CNN Architecture

The architecture of the neural network for AutoCaddie is crafted to strike an essential balance between precision and performance speed. Precision is crucial in identifying the subtleties of a golfer's technique and providing accurate feedback. Our CNN architecture is designed to delve into the fine-grained details of each swing, capturing nuances that can influence the golfer's performance. However, the detailed analysis should not compromise the speed at which feedback is delivered. Golfers and instructors expect near-instantaneous responses from analytical tools to ensure they can quickly make adjustments and observe the outcomes of their efforts.

To achieve this, we have chosen a lightweight yet powerful version of the CNN, which maintains high accuracy without the computational lag often associated with deep, complex networks. Our architecture is optimized for the swift processing of video frames, enabling the analysis of golf swings in almost real-time. This optimization is particularly significant when considering the deployment of the AutoCaddie system on platforms. The decision to adopt a streamlined CNN model also stems from the need for operational efficiency, ensuring that our system can process data in reasonable time without a dip in performance. Thus, the chosen network architecture reflects a well-considered approach that addresses the project's core demands for fast, accurate, and reliable golf swing analysis.

3.3.3.f Conclusion

In sum, the network architecture of the AutoCaddie project is explicitly tailored to extract, analyze, and learn from the vast information presented in the GolfDB dataset. By utilizing CNNs, the project harnesses the power of deep learning to provide sophisticated, accurate, and practical swing analysis. The architecture not only meets the project's current analytical needs but is also scalable for future enhancements, such as integrating new data types or adapting to emerging analytical requirements in golf instruction.

3.3.4 Database Selection

3.3.4.a Rationale for GolfDB's Adoption

Upon defining the requirements for a swing analysis database within the AutoCaddie system, GolfDB was singled out as a prime candidate. A database, to be compatible with our system, needs to be comprehensive, specialized, and aligned with the project's high standards for accuracy and detail. GolfDB stands out in this regard, providing a dataset exclusively composed of professional golf swings, each methodically labeled across a spectrum of events essential for nuanced analysis. The database's structured approach to video labeling—encompassing the sequence of events from address to finish—ensures

a level of detail that is unparalleled by other datasets. Furthermore, it includes vital metadata on the golfer, which are imperative for the system's multifaceted analysis algorithm. GolfDB's alignment with AutoCaddie's goals is thus made it a prime candidate for our data base for this project.

3.3.4.b Comparison with Other Data Sources

The extensive survey of potential databases underscored the challenges associated with unlabeled and non-specialized golf footage. Although alternative online sources for free golf images provide a wealth of visual content, the absence of golf-specific annotations renders the footage less actionable for targeted swing analysis. The process of converting this raw footage into a structured and analyzable form is fraught with difficulties that echo across data integrity, labeling accuracy, and resource allocation. In the case of services which provide these images, the diversity and scope of content were promising, but the lack of consistent swing phase labeling posed significant barriers to direct application in swing analysis. The use of GolfDB, pre-labeled with event frames and defined by clear classification criteria, eliminates these barriers, thereby facilitating immediate integration and use within our project.

3.3.4.c Limitations of Unlabeled Data and General Annotation Services

Unlabeled data presents a significant challenge, primarily due to the depth of knowledge required to accurately categorize golf swing mechanics. Generic annotation services, while potentially cost-effective, often fall short of capturing the specific biomechanical and technical nuances of golf swings. The risk of incorrect annotations is amplified when dealing with complex actions such as the golf swing, where the distinction between amateur and professional techniques can be subtle and highly technical. The precision required to identify these subtleties is beyond the capabilities of nonspecialized services, which may not have the domain-specific training necessary for high-fidelity labeling. Consequently, the selection of GolfDB, which offers granularity and domain expertise in its annotations, was critical to avoid the pitfalls associated with generalized data services.

3.3.4.d Considerations Against Self-Recording of Swing Footage

In the development of AutoCaddie, we contemplated the feasibility of creating an independent database by recording and analyzing our own golf swings. This approach would have granted us complete control over the dataset, allowing for customization tailored to our system's specifications. However, this method was ultimately deemed unsuitable for several reasons. Primarily, none of our team members possess the level of expertise required to exemplify the 'perfect' golf swing, a benchmark necessary to establish a standard for comparison and improvement within the application. Moreover, the time required to learn, execute, record, and then analyze these swings to a competent standard was beyond our project's temporal scope. The undertaking of such a massive task would not only divert attention from the core objectives of system development and refinement but also risk introducing subjective bias and unverifiable accuracy in our analysis. Consequently, GolfDB's expert-labeled dataset, which is both time-efficient and

reliable, became the clear choice for integrating high-fidelity swing analysis into the AutoCaddie project.

3.3.4.e Compatibility of GolfDB with Convolutional Neural Networks

GolfDB presents itself as a well-suited repository for use with convolutional neural networks (CNNs), which are state-of-the-art in image and video recognition tasks. CNNs excel in identifying complex patterns in visual data and making sense of images and videos by learning hierarchies of features. Given that GolfDB comprises high-definition, annotated video data of golf swings, it fits seamlessly into the framework of CNN-based analysis. The database's structured format, with labeled events corresponding to different swing phases, provides a rich ground truth that aids in the efficient training of CNNs. This synergy is beneficial since CNNs require a substantial amount of pre-classified data to achieve significant accuracy. GolfDB's alignment with the requirements for CNN training ensures that AutoCaddie can capitalize on the power of deep learning for feature extraction and sequence prediction, leading to more accurate swing analysis. The high frame rate and quality of GolfDB videos also mean that the temporal resolution is sufficient for capturing the fast, nuanced movements of a golf swing, which are essential for producing a detailed and actionable swing analysis within AutoCaddie's user interface.

3.3.4.f Advantages of Video-based Databases Over Sensor-based Systems

The use of video-based databases for swing analysis presents several advantages over sensor-based motion capture systems, particularly in terms of cost-effectiveness and non-invasiveness. While sensor suites provide detailed biomechanical data, they require the golfer to wear a cumbersome array of sensors, which can alter the natural flow of the swing. Moreover, the setup for such systems is time-consuming and financially prohibitive, especially for a senior design project with limited resources. Cameras, on the other hand, offer a non-invasive and more practical method for capturing swings. They allow for the recording of a golfer's swing in a natural environment without the need for specialized equipment attached to the player's body.

Choosing a video-based approach aligns with the AutoCaddie project's objective to create a user-friendly and accessible tool for swing analysis. Using GolfDB, which is rich in video data, we can develop a system that provides accurate swing feedback by simply filming the golfer with standard cameras. This method significantly reduces the project's costs and eliminates the discomfort and behavioral alterations caused by wearing sensors. Thus, GolfDB not only provides a comprehensive and specialized dataset for CNN training but also promotes an unobtrusive and realistic golfing experience, enhancing the practicability and scalability of AutoCaddie.

3.3.4.g Consultations with Golf Experts

In the quest to build a system that stands up to professional scrutiny, we engaged with a PGA tour expert to understand the critical elements of a swing analysis tool. This expert, with years of experience in teaching and analyzing golf techniques, provided us with a

checklist of essential swing mechanics and the corresponding visual indicators that should be captured for effective analysis. They emphasized the importance of precision in capturing the 'top of the swing' and 'impact moment,' which are crucial for determining the quality of a stroke. GolfDB's dataset reflects these expert inputs, offering meticulously labeled swings that cover all phases identified by the professionals. The contribution of this expert was not just in affirming the choice of GolfDB but also in reinforcing the necessity for an extensive, multi-dimensional swing analysis model, which is now a cornerstone of the AutoCaddie project.

3.3.4.h Incorporating Multi-Angle Video Analysis

The assessment of a golf swing from multiple angles offers a composite view that is vital for a comprehensive analysis. An accurate swing diagnosis cannot be confined to a single perspective, as it often fails to capture the dynamic interplay of movements involved in a golf stroke. Recognizing the importance of this multi-dimensional requirement, GolfDB's varied video angles are ideal for our purpose. The database includes both face-on and down-the-line videos, providing the foundational data necessary for evaluating swings in a manner that reveals the depth and intricacies of each shot. The ability to scrutinize a swing from various angles is crucial in developing a robust analytical framework for AutoCaddie, ensuring our analysis is as close to real-world conditions as possible.

3.3.4.i Technical Advantages for AutoCaddie's AI Model Training

The pre-labeled nature of GolfDB videos offers a unique advantage in training the AI model underlying the AutoCaddie system. By utilizing this dataset, we can bypass the initial hurdles of data preprocessing and directly engage in the more critical tasks of feature extraction and algorithm optimization. This direct path to model development not only conserves resources but also enables a focus on enhancing the analytical capabilities of AutoCaddie. GolfDB's structured dataset provides a solid foundation for developing machine learning algorithms that can detect and analyze key swing events with high precision, thus streamlining the learning curve and enabling the delivery of a sophisticated, user-friendly analytical tool.

3.3.4.j Concluding on GolfDB's Selection

The strategic choice to integrate GolfDB into the AutoCaddie project is a reflection of our analytical and educational goals. By incorporating GolfDB, we leverage a dataset that was built with an understanding of the golf swing's complexity, as interpreted by experts. This decision aligns with our dedication to delivering an accurate and insightful analysis to golfers, enabling them to refine their swings effectively. GolfDB's use in the AutoCaddie project underscores our preference for a database that not only provides high-quality, detailed swing analysis but also enhances the learning experience for our target audience.

3.4 Relevant Technologies: Kinematic Data

Along with the primary source of feedback for the AutoCaddie system being the machine learning artificial intelligence, kinematic data of the user's swing action with their driving arm is also used as a supplementary source of data. This source of data is more direct and deterministic, as it uses set equations on collected data to determine the type of feedback given from the system. The kinematic data consists of quaternion vector data and acceleration data. This data is collected via the IMUs, packaged by the MCU, and is transmitted wirelessly to the central computer for processing by a wireless transceiver module.

3.4.1 Inertial Measurement Units

3.4.1.a Overview, Comparison, and Selection of Component Type

Inertial measurement units serve as the primary motion sensor of the AutoCaddie system. The IMU units will be used to track the motion of the user's body, where one factor of feedback will be provided. For these purposes, there are several characteristics that IMU candidates are considered by. These most important characteristics include cost, degrees of freedom (DoF), data output modes, data output types, data output speed, and size dimensions.

In the context of motion tracking and analysis of the user's swing motion, IMUs emerge as the favored technology. IMUs amalgamate the functionalities of gyroscopes, accelerometers, and magnetometers, providing comprehensive data on the user's swing dynamics. This integrated approach allows for a holistic understanding of the user's motion, offering data for the user's swing control and arm straightness. IMUs often come in two configurations, a 6 DoF configuration utilizing only a gyroscope and accelerometer, and a 9 DoF configuration utilizing both a gyroscope and accelerometer, along with a magnetometer to stabilize and refine the data. As the 9 DoF is more accurate in circumstances void of magnetic interference sources, the 9 DoF configuration is preferable.

While Electromyography (EMG) devices can also be employed to measure muscle effort and strain during the swing action, determining the user's exertion level and the initiation of their swing, there are certain limitations to consider. EMGs often yield noisy data, potentially introducing inaccuracies in the analysis. Moreover, non-invasive EMG devices necessitate a significant amount of preparation before application on the user, which can be cumbersome and time-consuming. Given these considerations, the simplicity and ease of application of IMU devices make them the preferred choice for motion tracking and analysis, offering a robust and efficient solution for the AutoCaddie's needs of the user's arm kinematics.

3.4.1.b Adafruit BNO055/80

The Adafruit BNO055 (extending to the 080 series) IMU is a 9 degree-of-freedom (DoF) unit that features the following output data: absolute orientation (Euler and quaternion),

angular velocity, acceleration vector, magnetic field strength, linear acceleration vector, gravity vector, and temperature. The geospatial orientation vectors and orientations can be measured at 100Hz intervals. This system uses modified I2C communication standards as well as standard UART protocols. The overall dimensions of the BNO055 are 20mm x 27mm. To make the necessary kinematic motion assessments, inverse kinematics calculations can be performed using the Euler and quaternion orientation vectors. The overall package cost totals \$34.95.

3.4.1.c MPU-9255

The MPU-9255 is a 9 DoF multi-chip module IMU, which combines two die modules to form a singular system. The MPU-9255 combines a 3-axis gyroscope, 3-axis accelerometer, and a 3-axis magnetometer. The device features support for an “Automatic Activity Recognition” library, allowing for activity-induced measurements by the wearer from a low-power state without direct mode manipulation. This device supports both SPI and I2C communication standards. At normal low-power applications, the data output can output data at 500Hz. The overall dimensions of the MPU-9255 are 31.11mm x 16.84mm. Similar to the BNO055, inverse kinematics calculations can be performed using the Euler and quaternion orientation vectors to make the necessary assessments. The overall price of the IMU package is \$39.99.

3.4.1.d MPU-6050

The MPU6050 is a 6 DoF IMU device that combines a 3-axis gyroscope and 3-axis accelerometer onboard a digital motion processor. Along with the gyroscopic and accelerometer data, the unit provides a digital motion processor, which offloads some of the data processing workload of the central modules. This unit specializes in low-power, low-cost applications. Because of the device’s commonality, many libraries exist that facilitate the device’s integration into various systems. Some of these libraries also include digital motion processing algorithms, which can generate various extended types of data, such as quaternion data. Due to the reduced data vectors and processing offloading, the IMU unit can produce output at frequencies up to 1kHz. This IMU unit uses I2C communication standards. The overall dimensions of this unit are 21mm x 14mm. The overall price of the IMU unit is \$9.90.

3.4.1.e Hardware Comparison Table

Factor	BNO055/80	MPU-9255	MPU-6050
Unit cost	\$34.95	\$39.99	\$9.90
DoF	9	9	6
Output modes	I2C, UART	I2C, SPI	I2C, SPI
Output types	Absolute orientation (Euler and quaternion), acceleration vectors, gravity vectors, magnetic field vectors, and temperature.	Quaternion, Gyroscopic angular rotation vectors, and acceleration vectors	Gyroscopic angular rotation vectors, acceleration vectors, magnetic field vectors, and temperature.
Output Speed	100Hz	500Hz	1000Hz
Dimensions	20mm x 27mm	31mm x 17mm	21mm x 14mm

Table 5: Hardware comparison table that compares the IMU candidates.

3.4.1.f Conclusion

Based on the above research, the chosen IMU unit will be the MPU6050 unit. Regarding its output mode and data types, I2C is a commonly supported communication type, and requires a relatively low amount of connections to function. While the MPU6050 does only inherently support 6 DoF basic datatypes, digital motion processing libraries can provide advanced algorithms to make estimations on additional forms of data, such as quaternion data, which is preferable. The device's small profile also makes it easy to account for and incorporate on attachment systems on the user. Finally, the low-cost of the device is preferred as lowering the cost of the overall AutoCaddie project is a prominent goal.

As the IMU data is collected during the swing action period, the data must be formatted into a package that can be wirelessly transmitted to the central computer for further processing. This wireless transmission must be done as to reduce the presence of wires that extend off the user which may burden the user or may be caught on the user and cause inconveniences or damage.

3.4.2 Wireless Data Transmission Components

3.4.2.a Overview, Comparison, and Selection of Component Type

The wireless transceiver is the component responsible for tying together hardware and software; regardless of the variant chosen, the transceiver will communicate between the microcontroller and the target device, delivering the data gathered from the Inertial Measurement Units (IMUs) to the device performing analysis on the video feed (likely a laptop or desktop computer). For the purposes of this project, some important factors to consider are connectivity options, power consumption, output power, data rate, and maximum transmission distance.

In the selection of a wireless transceiver, many options exist which could effectively perform the tasks required by AutoCaddie. The options researched are Wi-Fi, Radio (RF), and Bluetooth.

Radio transceivers are a considerable option for the AutoCaddie project. However, when compared to Bluetooth and Wi-Fi transceivers, they present drawbacks which must be examined prior to selection. RF transceivers generally transmit over great distances, with high-power transmitters being capable of traversing hundreds of miles. The AutoCaddie project is unlikely to exceed 100 meters (about 328.08 ft) of range; therefore, this extreme range could present unnecessary power consumption and complexity. Bluetooth Low Energy, for instance, is designed to perform over ranges between 10 and 100 meters, which is exactly applicable for this project. RF transceivers generally offer low data rates, with the AM and FM encoding processes consuming time and power far exceeding those of Bluetooth and Wi-Fi. In projects where the data to be transmitted could be measured in bits, this is acceptable; however, AutoCaddie will be transmitting in the range of 13.6 KB, significantly larger than the data rate of RF transceivers could transmit in a reasonable time. Moreover, RF transceivers are often complex to configure and use for those who are unfamiliar with their applications. Therefore, in a project oriented towards accessibility and ease of use, Bluetooth and Wi-Fi are better options. RF transceivers

operate in the unlicensed 2.4 GHz ISM band, which is often crowded by other wireless devices. Without a frequency hopping feature, this can result in signal interference by other RF devices. RF transceivers do not have similar levels of universal compatibility like Wi-Fi and Bluetooth, and therefore is generally challenging to access for the common consumer.

Wi-Fi transceivers are a significant improvement over RF devices. However, in comparison to Bluetooth, there are still inferiorities to consider. Like RF transceivers, Wi-Fi transceivers offer significantly higher ranges of communication than Bluetooth. However, as has already been explored, the purposes of AutoCaddie do not require ranges exceeding 100 meters (about 328.08 ft) for reasonable applications, therefore Wi-Fi is a less than energy-efficient choice for a battery-operated project. Wi-Fi connections, while less complex than RF connections, are more challenging to configure than Bluetooth pairing. Many devices require complex network connections and configurations, reducing the accessibility of the project. Wi-Fi also operates in the 2.4 or 5GHz bands, which are shared by many other wireless devices. This opens the project to the possibility of external interference without the introduction of a frequency hopping feature (like that boasted by Bluetooth). WiFi offers data ranges which range to the extreme, which can be excellent for high-definition video streaming or the transmission of large files. However, with an average file size of 13.6 KB, this extreme data rate is not required and can result in unnecessary power consumption. Furthermore, Wi-Fi is less universally compatible than Bluetooth, especially when using legacy devices. Ensuring Wi-Fi compatibility across different platforms can be more challenging than an industry standard such as Bluetooth pairing.

Bluetooth offers excellent power efficiency, with numerous devices using Bluetooth Low Energy (BLE) which is optimized for low-power consumption. For a battery-operated project such as AutoCaddie, this is an extremely attractive feature. Additionally, Bluetooth utilizes output powers which are suitable for short-range connectivity, which is the most likely application of AutoCaddie. Bluetooth offers data rates which are more than adequate for the purpose of transmitting IMU data; the collection rate of data from the IMUs is approximately 100Hz, and with a general data rate of 9600 bps (configurable to higher or lower as needed), this transmission option is capable of exceeding a reasonable data rate for, for instance, a UART output which could range upwards of 13.6 KB. Moreover, Bluetooth boasts a maximum transmission distance of approximately 100 meters (about 328.08 ft), which is significantly more than would be required for a reasonable application of AutoCaddie, which is intended to be short ranged.

Bluetooth is also very accessible, with the majority of modern devices available in the market containing Bluetooth Serial chips, therefore communicating to a user's device is as simple as turning it on and selecting it from a pairing list. Bluetooth uses adaptive frequency hopping to mitigate signal interference, which, since it uses the 2.4 GHz ISM band, is useful in maintaining a stable connection in an environment crowded with other wireless devices. Bluetooth is well-established, with a wide range of chips, development tools, and resources available to facilitate development and ease of use.

With these factors researched, the most applicable options in the Bluetooth range of devices will be considered. Bluetooth boasts low power consumption, adequate range, excellent configurability, universally accepted connectivity options, and non-excessive data rates, as well as security features such as adaptive frequency hopping.

3.4.2.b HC-05

The HC-05 is a very popular wireless transceiver, most often used in Arduino-integrated projects (ATMEGA328P MCU) as a Bluetooth communication device. The use of Bluetooth serial as a communication mode is extremely applicable in this case, as, depending on the transmission distance of the transceiver, it can be used for virtually all modern devices, especially laptops. The device requires UART to properly receive data for transmission, something which the vast majority of MCU chips boast, and virtually all computers use. The transceiver consumes 30 mA of power while actively transmitting at a default data rate of 38400 bps (Baud), though this rate can be adjusted manually to match the default 9600 Baud rate of most computers. The output power is +4dBm. The transmission distance is at a maximum of 10 meters (about 10 adult steps).

3.4.2.c HC-06

The HC-06 transceiver is the sister device to the HC-05, boasting a greater output power of +6dBm at a default data rate of 9600 bpm (Baud), with a significantly greater maximum transmission distance of 100 meters (about the length of a football field). The device requires UART to receive data with a supporting option of USB connectivity, communicating across Bluetooth Serial. The HC-06 operates at 40mA while transmitting.

3.4.2.d nRF24L01+

The nRF24L01+ wireless transceiver is a very popular radio transceiver. It operates at 2.4 GHz ISM with an operating power consumption of 26 μ A. It boasts an output power of 11.3mA @ 0dBm, with a data rate (reception) of 13.5mA @ 2Mbps. The transmission distance for this device, as it operates via radio, will be significantly greater than the purposes of this project require, and thus has not been included in this section.

3.4.2.e Hardware Comparison Table

Factor	HC-05	HC-06	nRF24L01+
Connectivity	UART, Bluetooth Serial	UART, Bluetooth Serial, USB	RF (2.4 GHz ISM)
Power Consumption	~30mA Operating	40mA Operating	26 μ A
Output Power	+4dBm	+6dBm	11.3mA @ 0 dBm
Data Rate	Default 38400 bps (Baud Rate)	Default 9600 bps (Baud Rate)	13.5mA @ 2Mbps
Tx Distance	~10m	~100m	Irrelevant (RF)

Table 6: Hardware comparison table that compares wireless transceiver module candidates.

3.4.2.f Conclusion

Based on the above research, the transceivers to be employed on the AutoCaddie architecture will be the HC-06 wireless transceivers. These devices boast an excellent transmission range, nearing 100 meters (about 328.08 ft) which will allow for an excellent level of flexibility. The HC-06 device utilizes Bluetooth Serial communication, something

which is generally supported natively in the vast majority of modern devices, particularly laptops, which will be the primary target for data reception. The transceiver requires UART to be used to send the data to be transmitted via Bluetooth, which is supported by the MSP430FR6989IPZ, and is very easily configured on laptops and the MCU chip selected. The default data rate is 9600 bps, which will ensure that the feedback timing requirements are met and exceeded, allowing the user to view their individualized feedback without data bottlenecks significantly in the hardware. This component, however, like most data transmission modules, draws much more power as compared to other standard components of a PCB. As such, a battery power source must be selected to properly account for the use case of the transceiver's power draw.

3.4.3 Battery Power Source

3.4.3.a Overview, Comparison, and Selection of Component Type

Batteries are a critical part of the sensor system architecture that is worn by the user. Without the batteries, the IMU and the wireless data transmission systems cannot be powered and will not be functional, thus disabling a core part of the AutoCaddie system. The batteries implemented in the wireless data transmission PCB must provide enough charge to power the user-mounted system for at least 2 hours. Furthermore, the battery weight must be minimized so as to reduce the overall weight of the system mounted on the user. Additionally, rechargeable batteries are preferred to reduce battery waste from the system.

There are several available battery types suitable for use in AutoCaddie hardware systems, with alkaline, lithium-ion, and lithium-polymer batteries being commonly employed in PCB powering applications. Lithium-ion batteries are rechargeable, relying on lithium ions stored within a liquid electrolyte, offering relatively high energy densities and a long-term lifespan when used correctly. On the other hand, lithium-polymer batteries are similar to lithium-ion batteries but store ions within a polymer material, resulting in a slightly higher energy density while being less resilient to damage compared to their lithium-ion counterparts.

Alkaline batteries represent another battery type suitable for powering the hardware system. Alkaline batteries are disposable and generally more robust than other types, but they can pose risks if damaged and are not rechargeable due to their disposable nature. For the AutoCaddie project, lithium-ion batteries were chosen as the preferred battery type because they are rechargeable and generally more durable, a crucial consideration given that the battery will be integrated into a wearable circuit system subject to inertial impacts and movements that may cause wear.

3.4.3.b Tenergy 7.4v 2.2Ah Li-ion 18650

The Tenergy 7.4V lithium-ion battery pack is a rechargeable, high-capacity battery. With an energy capacity of 2200mAh and a continuous discharge capacity of 2.2A, this battery can provide multiple hours of charge should charge consumption be optimized. Since this

battery pack consists of two standard 18650 2.2Ah batteries, the approximate weight of the system comes to 100g. The price of this battery pack totals \$16.99.

3.4.3.c Adafruit 3.7V 2.2Ah Lith-ion 18650 (PKCELL)

The Adafruit 3.7V lithium-ion single-cell battery pack is a rechargeable, lightweight battery cell. With an energy capacity of 2200mAh and a continuous discharge capacity of 1A, this battery can provide about two hours of charge even at continuous standard draw. As a single 18650 size battery cell, the weight of this product is 46g. The price of this battery pack totals \$9.95

3.4.3.d Adafruit 3.7V 6.6Ah Lith-ion 18650 (PKCELL)

The Adafruit 3.7V lithium-ion dual-cell battery pack is a rechargeable, high-capacity battery pack. With an energy capacity of 6600mAh and a continuous discharge capacity of 1A, this battery pack can provide several hours of charge with charge consumption optimization. As this battery pack contains three 18650 battery cells, the weight of this product comes to 148g. The price of this battery pack totals \$ 29.95.

3.4.3.e Hardware Comparison Table

Factor	Tenergy 7.4v 2.2Ah Li-ion 1865	Adafruit 3.7V 2.2Ah Lith-ion 18650 (PKCELL)	Adafruit 3.7V 6.6Ah Lith-ion 18650 (PKCELL)
Energy Capacity	2.2Ah	2.2Ah	4.4Ah
Discharge Capacity	2.2A	1A	1A
Weight	~100g	46g	120g
Rechargeable	Yes	Yes	Yes
Cost	\$16.99	\$9.95	\$29.95

Table 7: Hardware comparison table that compares battery candidates.

3.4.3.f Conclusion

Comparing the battery pack candidates, the Adafruit 3.7V 6.6Ah Lithium-ion battery was chosen as the best candidate. This rechargeable battery pack provides the most charge capacity of the other candidates. Furthermore, this battery pack is comparable in weight to the Tenergy battery pack, which has only half the charge capacity. The high capacity of the battery pack was the most influential factor in the selection of this component, as it guarantees the sensor and wireless data transmission systems will be able to be powered for a sufficient amount of time. The 1A discharge capacity is an acceptable amperage, as the system should not exceed this amperage as a requirement. Should more amperage be required, current regulators can be implemented. Regarding the price of the battery, though it is the most costly selection, the length of charge supplied to the system is a critical factor, and thus must provide large “safety” margins of available power. Further, it remains within a similar price index to the other selections (maximum \$25.00 price range).

Along with the data transmission systems, which are the primary power draw of the hardware system, the batteries must also supply power to the IMUs and the Microcontroller unit. The MCU can also control the general power draw of the external components, such as the wireless transmission unit. By idling the given components, their overall power draw can be reduced.

3.4.4 Microcontroller Unit

3.4.4.a Overview, Comparison, and Selection of Component Type

The MCU (Microcontroller Unit) chip will be the lynchpin of the entire project's sensory systems. This processor will collect data from the sensors (IMUs) attached to the user, perform arithmetic upon the data, and finally send the packeted data to a wireless transceiver unit. Integrated onto the PCB, without the MCU chip, nothing is collected or sent, therefore selecting the proper device is paramount to the success of the project. The most important factors considered when evaluating MCU chips are cost, power consumption, processor frequency, number of GPIO pins (analog and digital), connectivity options (UART, I2C, SPI, Phillips I2C), memory (type and size), low-power modes, and wireless transmission options.

Before concluding with the use of a microcontroller, other options such as FPGAs may be considered. An FPGA (Field-Programmable Gate Array) is an extremely versatile and configurable device, excelling at highly specialized tasks requiring high speed, parallel processing, and custom digital logic design. However, the AutoCaddie hardware will very simply receive data from the IMU (Inertial Measurement Units) components, format them to an easily processed file, and output the formatted data to a wireless transceiver. The versatility and configurability of an FPGA is not an applicable advantage in this context. Furthermore, FPGAs are typically significantly more expensive than microcontrollers, therefore projects that do not require their particular upsides incur unnecessary costs via their use. While FPGAs excel at highly specialized tasks, they are less suited for general-purpose processing, such as formatting I/O data and outputting it. FPGAs are generally fixed in terms of their hardware configuration; projects which implement them typically limit their scalability to the limited reach of their FPGA, while a microcontroller has the capability to receive data from I/O devices added to the project later. Microcontrollers, such as the MSP430, generally have intrinsic hardware support for common communication interfaces such as UART and I2C, making them more applicable for interfacing with simple I/O devices such as IMUs. Finally, FPGAs typically consume greater power than microcontrollers, enabling their advantages but in the context of a simple I/O data transmission, results in extraneous and unnecessary power consumption.

3.4.4.b MSP430FR6989

The MSP430FR6989 is an extremely versatile MCU (Microcontroller Unit) chip which easily satisfies the computation and transmission requirements of this project. The chip is compatible with all input and output connectivity requirements demanded by the I/O devices included in the PCB, those being UART and I2C, with an added peripheral option of SPI. As was specified in the PCB requirements, this chip has no native wireless transmission capacities, requiring external transceivers to be employed. There are 16 analog inputs and 83 digital GPIO pins. The power consumption in Standby mode is 100 μ A/MHz, which is sufficiently low as to avoid unnecessary draw on the battery. The MSP430FR6989IPZ also boasts 3 low-power modes, each reducing the power consumption by a factor of 100 A when certain components are not required. The cost is a conservative \$10.21 per unit.

3.4.4.c ATMEGA328P

The ATMEGA328P chip is used most frequently in Arduino development boards, making it easily one of the most popular and documented MCU chips on the market. It boasts a processor frequency of 20 MHz and achieves 20 MIPS, 32KB of flash memory, USART, SPI, I2C connectivity, and more. As was delineated in the PCB requirements, this chip has no innate wireless transmission capabilities, requiring the use of external transceivers. The chip boasts 32x8 GPIO registers with an active power usage of 200 μ A/MHz. Also extremely important, the chip has 2 low-power modes which allow the user to conserve energy when certain features are not required.

3.4.4.d ESP32

The ESP32 chip is extremely unique among MCU chips. Advertising an extremely fast processor with a frequency of 160 MHz, 384KB ROM and 400 KB SRAM, in terms of performance (excluding power) it vastly outperforms its counterparts. It boasts UART, SPI, I2C, I2S, USB/JTAG connectivity, making it the most versatile in terms of connectivity among those evaluated for the purposes of this project. In terms of wireless communication, it is capable innately of Wi-Fi, Bluetooth, and radio (RF) communication, making it capable naturally of what this project requires. It boasts 22 GPIO pins with a power consumption of 335 mA, significantly higher than the other chips on this list. However, it compensates for this with 4 low-power modes.

3.4.4.e Hardware Comparison Table

Factor	MSP430FR6989	ATMEGA328P	ESP32
Processor Frequency	16 MHz	20 MHz (20 MIPS)	160 MHz
Memory	128KB FRAM	32KB Flash, 1KB EEPROM, 2KB SRAM	384KB ROM, 400 KB SRAM
Connectivity	UART, SPI, I2C	USART, SPI, I2C (Phillips compatible)	UART, SPI, I2C, I2S, USB/JTAG
Wireless Transmission	None (PCB Req)	None (PCB Req)	Wi-Fi, Bluetooth, RF
GPIO	16 Analog In, 83 D I/O	32x8 GP Registers	22 GPIO
Power Consumption	Active 100 μ A/MHz	(@1MHz) Active 200 μ A	Active 335 mA
LPM	Standby (0.4 μ A), Real-Time Clock (0.35 μ A), Shutdown (0.02 μ A)	Power-Down (0.1 μ A) Power-Save (0.75 μ A)	Modern-Sleep (28 mA) Light-Sleep (130 μ A) Deep-Sleep (5 μ A) Power Off (1 μ A)

Table 8: Hardware comparison table that compares MCU candidates.

3.4.4.f Conclusion

From the above research and relevant testing, the selected MCU chip is the ESP32. The ESP32, being a more modern yet mature MCU chip means that its ease of use and general support is much greater than that of its competitors. The interfacing IDE support C and C++ languages as well as derivatives. Furthermore, due to its popularity, community support and available libraries are much greater than other chip types.

In terms of specific project requirements, the ESP32 maintains itself as a prime selection for the AutoCaddie project. As mentioned earlier, its relative modernity, maturity, and popularity is such that there are a number of easily accessible libraries for interfacing with

the IMU and Bluetooth transceiver modules required for the AutoCaddie project. Additionally, the ESP32 features enough serial connection modules to interface with all required peripherals of the proposed model of the AutoCaddie project. Finally, with regard to the hardware profile of given ESP32 models, the pin interfaces with the ESP32 can allow for more simpler solderability, further facilitating the design of the hardware sensory interfaces.

3.4.5 Quaternion Data Processing

Within AutoCaddie's demand for linear data processing, quaternion data plays a crucial role in capturing and interpreting the spatial and rotational dynamics in a golfer's swing. This section explores quaternion data, in terms of what separates it from Euler data, and the methods that can be used for real-time calculations. These insights contribute to the fundamental understanding of the procedures undermining AutoCaddie's ability to analyze the various fields of user kinematics during the swing action.

Euler angles represent a set of three angles used to describe the orientation of an object in a three-dimensional space. These angles represent rotations about the axes of a fixed coordinate system and are commonly used in various fields, such as robotics and aerospace engineering.

Specifically, according to J. Diebel, [34] we can define these angles as a rotation sequence, which is defined as a sequence of three rotations around different axes. The order of these rotations is important and can be represented by three integers: (i, j, k).

These angles are represented as a three-dimensional vector denoted by $u := [\phi, \theta, \psi]^T$. Here, ϕ , θ , and ψ are the angles of rotation about the specified axes. The Euler angles are used to compute a rotation matrix $R(\phi, \theta, \psi)$, which represents the orientation of an object after the specified rotations.

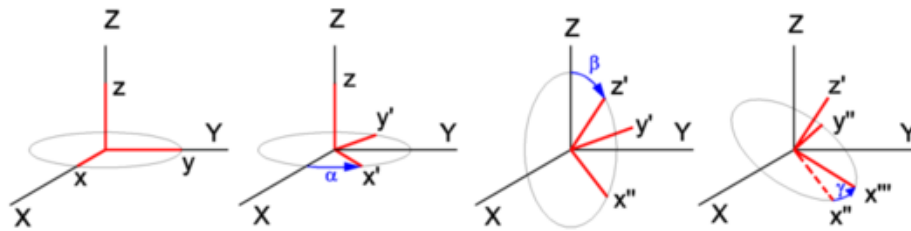


Figure 16: Euler angles. The far-left plot represents the original (x,y,z) axis. The second-leftmost plot represents applying angle ϕ across the z-axis. The second-rightmost plot represents applying angle θ across the y-axis. The far-right plot represents applying the angle ψ across the x-axis, completing the Euler angle. Referenced via [35] .

The time-derivative of the Euler angle vector gives the vector of Euler angle rates, representing how fast each angle is changing with time. Angular velocity, which describes how fast an object is rotating, is related to the Euler angle rates through specified matrices. In the context of AutoCaddie, this is important in determining the rate of change of the bend of the user's least dominant arm during the swing motion.

Specifically, the following process explains how to calculate the straightness between two points over time using Euler angles. First, each Euler angle can be converted into a 3x3 rotation matrix. The conversion process is highly dependent on the order of rotations. Next, interpolation can be utilized to obtain a series of intermediate rotation matrices that represent the rotation over time. A common methodology for this process is linear interpolation (LERP), with matrix operations shown below.

$$\begin{bmatrix} 1 & x_0 & y_0 & z_0 & x_0 y_0 & x_0 z_0 & y_0 z_0 & x_0 y_0 z_0 \\ 1 & x_1 & y_0 & z_0 & x_1 y_0 & x_1 z_0 & y_0 z_0 & x_1 y_0 z_0 \\ 1 & x_0 & y_1 & z_0 & x_0 y_1 & x_0 z_0 & y_1 z_0 & x_0 y_1 z_0 \\ 1 & x_1 & y_1 & z_0 & x_1 y_1 & x_1 z_0 & y_1 z_0 & x_1 y_1 z_0 \\ 1 & x_0 & y_0 & z_1 & x_0 y_0 & x_0 z_1 & y_0 z_1 & x_0 y_0 z_1 \\ 1 & x_1 & y_0 & z_1 & x_1 y_0 & x_1 z_1 & y_0 z_1 & x_1 y_0 z_1 \\ 1 & x_0 & y_1 & z_1 & x_0 y_1 & x_0 z_1 & y_1 z_1 & x_0 y_1 z_1 \\ 1 & x_1 & y_1 & z_1 & x_1 y_1 & x_1 z_1 & y_1 z_1 & x_1 y_1 z_1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} = \begin{bmatrix} c_{000} \\ c_{100} \\ c_{010} \\ c_{110} \\ c_{001} \\ c_{101} \\ c_{011} \\ c_{111} \end{bmatrix},$$

Figure 17: A matrix multiplication method to solve linear interpolation. $(x_0, y_0, z_0), (x_1, y_1, z_1)$ are defined, and vector c is defined as function values. Referenced via [36].

From this point, once a series of rotation matrices are calculated to represent the orientation over time, calculations can be made to find the straightness between two subsequent points using the Euclidian distance formula.

However, a large weakness of Euler angles is the presence of singularities. Singularities refer to specific angle configurations where the representation becomes problematic. This can lead to problems such as gimbal lock. Gimbal lock occurs when two of the three rotational axes become parallel, resulting in a loss of one degree of freedom. This makes it challenging to uniquely represent orientations near the singularity, leading to ambiguity in interpreting the rotations.

Depending on the sequence of rotations, singularities may occur, making certain orientations difficult to represent accurately. These singularities can result in discontinuities, making it challenging to interpolate smoothly during the calculation process. Since AutoCaddie requires accurate calculations, jeopardizing data during the swinging process is unacceptable. That is why, to mitigate these issues, alternative representations of rotational data are often used, primarily quaternions.

Quaternions represent a numerical way to represent complex numbers in three dimensions. As provided by J. H. Challis, [37] the complex number form of a quaternion is shown below (also known as Hamilton's quaternion formulation).

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

Equation 1: The complex number form of a quaternion data point.

In this equation, q_0 , q_1 , q_2 , and q_3 represent real numbers, and i , j , and k are imaginary components. Separately, we can say that the four-tuple form of a quaternion is shown as the equation below.

$$q = (q_0, q_1, q_2, q_3)$$

Equation 2: The four-tuple form of a quaternion.

Separately, the scalar and vector form of a quaternion is shown in the equation below.

$$q = q_0 + v, v = (q_1, q_2, q_3)$$

Equation 3: The scalar and vector form of a quaternion.

On a separate note, Hamilton's quaternion formulation allows direct visualization as a rotation about an axis, avoiding the ambiguity of Euler angles. The change in orientation can be visualized from its quaternion. Specifically, given a single quaternion, the rotation angle can be extracted from the equation below.

$$(a, b, c) = \left(\frac{q_1}{\sin\left(\frac{\theta}{2}\right)}, \frac{q_2}{\sin\left(\frac{\theta}{2}\right)}, \frac{q_3}{\sin\left(\frac{\theta}{2}\right)} \right), \theta = 2 \cos^{-1}(q_0)$$

Equation 4: The rotation angle θ , and the axis of rotation (a,b,c) .

Backtracking, the presence of singularities within Euler angles leads to ambiguities. The gimbal lock problem arises when certain orientations lead to a loss of one degree of freedom, which is the result of the event when two of the three rotational axes become aligned. However, quaternions represent rotations in a different way that avoids this issue.

By considering the set of all possible 3D rotations, this set forms a mathematical structure known as the rotation group (SO(3)). The unit quaternion space, when used to represent 3D rotations, acts as a double cover of the rotation group SO(3). In other words, the rotation space of unit quaternions forms a hypersphere in a four-dimensional space. For every point on this hypersphere, there are two antipodal points that represent the same rotation in the SO(3) space. Knowing this, when interpolation is introduced to find rotational data in this quaternion space, if there is a gimbal lock present, an equal rotation point will be available. Therefore, there will always exist smooth interpolation path, and singularity is entirely avoided using quaternion data.

By analyzing this information, it is safe to conclude that quaternions are a superior choice for rotational data recording in the context of AutoCaddie. By utilizing quaternions, there will not be a case where data is invalidated by discontinuities.

Additionally, there is a straightforward equation that can be used to find the angle between two quaternion points. It can be derived as follows.

Given two unit quaternions (the magnitude of a unit quaternion is equal to 1), $q_1 = (w_1, x_1, y_1, z_1)$ and $q_2 = (w_2, x_2, y_2, z_2)$, the dot product between them and the cosine of the half angle between them can be defined by the following equations.

$$q_1 \cdot q_2 = w_1w_2 + x_1x_2 + y_1y_2 + z_1z_2$$

Equation 5: The dot product between two quaternions.

$$\cos\left(\frac{\theta}{2}\right) = \frac{q_1 \cdot q_2}{||q_1|| ||q_2||} = q_1 \cdot q_2$$

Equation 6: The cosine half angle between two unit quaternions.

Therefore, by solving for the angle θ , the equation for the angle between two unit quaternions is shown below.

$$\theta = \cos^{-1}(2 \langle q_1, q_2 \rangle - 1)$$

Equation 7: The angle between two quaternions.

Furthermore, by utilizing this equation, AutoCaddie's several possible calculations that determine linearity can be easily performed. Like the frame extraction discussed earlier, the MCU will transmit a packet of quaternion data to the computing device, representative of the spatial and rotational data during every instance of time measured from the beginning to the end of the swing. Primarily observed between specific events of the golf swing, the non-dominant arm of the user is required to be hyperextended to align with proper technique. By performing this angular calculation between subsequent points throughout the dataset, it is simple to identify any significant arm straightness deviations during the swing process.

This will be implemented via the software system of AutoCaddie. Since the process can be simplified to a simple equation, it becomes also possible to simplify the required code to a simple function. In the next section, the programming language used for this calculation will be discussed as well as useful libraries that will facilitate the implementation of other subsystems, specifically the other data preprocessing techniques, the neural network, and the software design.

3.5 Relevant Technologies: Software Integration

3.5.1 Overview and Language Selection

Within AutoCaddie's domain, the choice of programming language plays one of the most pivotal roles in shaping the product's functionality, efficiency, and overall effectiveness. The focus of this section is the rationale behind the selection of Python as the dominant programming language, and the specific libraries that can be imported to improve many features of this project. First, by examining programming languages and analyzing benefits and drawbacks, the most informed decision can be made.

In the context of this project, optimal developmental efficiency is anticipated through the alignment of the selected programming language with the developers' existing familiarity. Additionally, through an exhaustive examination of popular languages supporting the creation of CNNs, four programming languages have been selected as candidates. These languages are Python, Java, C++, and JavaScript.

First, Python [38] is a dynamic and versatile programming language. It stands out for its unique characteristics that distinguish it from traditional languages. It embraces flexibility and clear coding design to enhance readability. It also benefits from its widespread usage, as it is known for its extensive library collections.

There are many advantages to Python. Its simple syntax makes it one of the most readable languages, especially in comparison to the other languages listed prior. It is also versatile, as it accommodates both procedural and object-oriented programming. It is efficient for rapid development, which is useful in the AutoCaddie project for the developmental procedure. Also, it showcases true portability by being able to run on any system that supports it. Additionally, it automatically performs memory allocations and garbage collection, which eliminates memory-related issues. Most importantly, it contains an extensive collection of built-in libraries, third party libraries, and open-source libraries that contain mostly every algorithm and optimization currently researched. A strong example of this is for developing CNNs, where multiple libraries currently offer support for development.

However, Python also presents disadvantages. It is not a very fast language due to the nature of it being dynamically typed. This also makes it memory intensive, which makes it difficult to use for memory-constrained environments. Additionally, it is not very popular for desktop or mobile applications, as other languages are stronger for this problem. It is also not optimized for custom database access, and it does not have multithreading support, which poses potential concern for neural network development.

Second, Java [39] stands as a versatile, high-level programming language celebrated for its robustness, security, and object-oriented nature. Notable, its syntax, designed for readability, resembles the English language, contributing to its accessibility. It is also platform-independent, which can allow for seamless execution across separate platforms.

Java contains many benefits. Its straightforward syntax and automated memory management simplify the learning curve for this language. It also utilizes object-oriented programming by containing classes, inheritance, encapsulation, polymorphism, and abstraction. Also, it excels in security with the absence of explicit pointers, execution within virtual machines, and dynamic class loading. Plus, it has strong memory management, exception handling, and type checking. Lastly, it has support for multithreading, which allows for tasks to be parallelly executed. This is an important concept for training certain neural networks, particularly random forests, where unrelated processes can be completed in parallel to achieve more results at the same time.

However, it also contains drawbacks. Java interpreted runtime may lead to slower performance when compared to other languages such as C++. It also relies on the Java virtual machine, which results in higher memory consumption. Also, due to the lack of the ability to explicitly declare pointers, Java has minimal machine interaction, which disables certain computational benefits. Lastly, there is no direct method for manual memory release, which may increase memory demands over time.

Third, C++ [40] is a programming language that is known as an extension of the basic C language. It stands out for being versatile and powerful while also containing object-oriented programming methods. It is portable and allows for low-level maneuverability. Overall, C++ has become essential in diverse applications over the years, including compilers and operating systems.

There are many advantages to C++. First, it facilitates efficient problem solving through object-oriented principles such as classes, data abstraction, etc. Next, it is highly portable, which can allow for seamless integration of code throughout multiple systems. It is recognized for low-level maneuverability, which allows for device-specific optimizations. Lastly, it has comprehensive memory management, multi-paradigm flexibility, and scalability.

However, there are many drawbacks to C++. First, the language's complexity poses a barrier to entry. Notably, C++ lacks support for certain modern programming concepts such as functional programming and automatic memory management. Also, the absence of built-in garbage collection, security vulnerabilities due to direct memory manipulation, and array limitations underscore this language's usability in the scope of this project.

Fourth, JavaScript [41] a lightweight and object-oriented scripting language, plays a large role in crafting dynamic HTML pages with interactive effects on the web. Primarily used for enhancing user interfaces and adding dynamic functionalities to websites, JavaScript's versatility extends to game development and mobile app creation. Its syntax is heavily influenced by C, and it allows for web tools to be integrated seamlessly into projects.

JavaScript has many advantages. First, it supports interfaces, modules, and classes, which covers a large range of programming objectives. It can operate in a relationship of both the client-side and server-side. Also, it empowers developers to create responsive interfaces, reacts to user interactions, and can run on any browser. It also facilitates the development of large-scale applications.

However, JavaScript presents crucial drawbacks in the scope of this project. AutoCaddie is a program that will be downloaded locally, so immediately the usage of JavaScript to make a local program is outweighed by the other three listed languages. Additionally, it lacks support for networking applications, which limits its utility in that domain. Further, JavaScript supports only single inheritance, but not multiple level inheritance like other object-oriented languages provide. For these reasons, without much comparison, JavaScript will not be the selected language for this project.

Putting everything together, Python, Java, and C++ all provide object-oriented programming that can run programs efficiently. C++ presents the largest learning curve, but in terms of practicality, that is not a limiting factor for development for this project. The primary selection criteria to be considered is the available tools provided for each language for developing the neural network, as well as tools that can automate data preprocessing. As the neural network is the driving force behind AutoCaddie, ensuring

that the CNN is constructed optimally is the main concern. Therefore, due to the overwhelming community support for neural network libraries being developed in Python (elaborated upon in **Section 3.5.6**), as well as the provided libraries for matrix operations to improve processing speeds (elaborated upon in **Section 3.5.7**), Python is the best choice for the programming language for AutoCaddie.

3.5.2 Introduction to Data Preprocessing Libraries

As covered earlier, there are extensive data preprocessing techniques that need to be incorporated into AutoCaddie to allow for the smooth flow of live data analysis. There are three separate data manipulation algorithms that need to be addressed: frame extraction, frame standardization, and quaternion data calculations. First, when the video stream is fed into the code, there needs to be logic for parsing it into a list of frames representative of every time stamp throughout the swinging process. Second, once the images have been reduced, they need to be resized to match a standardized format, which can be completed using zero-padding. Third, by performing basic arithmetic, it is possible to compute quaternion data calculations for rotational shifts over time. Each of these topics will be explored in the subsequent sections, with research towards potential Python libraries to be utilized by AutoCaddie.

3.5.3 Matrix Operations and Zero-Padding

One of the primary selections for AutoCaddie involves the handling of matrix operations. Optimal data management is critical for AutoCaddie's performance, so having access to tools to combine similar data operations into single variables such as matrices can reduce the time to complete complex computations. Not to mention, each frame from the video data can be represented as a 2D grid of pixels, which stresses the importance of having access to matrix operations within the code. The three most popular Python libraries for this task are NumPy, SciPy, and pandas.

NumPy [42] short for numerical Python, constitutes a powerful library encompassing multidimensional array operations. Numpy's utility extends to operations related to Fourier transforms, shape manipulation, and comprehensive support for linear algebra functions, along with random number generation capabilities. Thus, it significantly enhances computational efficiency when dealing with arrays and matrices.

NumPy offers an array of advantages. First, NumPy arrays are designed to be up to 50 times faster than traditional Python lists, ensuring efficiency in mathematical operations. The array object comes equipped with numerous supporting functions, simplifying operations on ndarray. Additionally, NumPy's compatibility with Python and straightforward installation further enhance its appeal.

However, there is also an array of disadvantages present with NumPy. First, NumPy has separate support for "not a number" values, which makes it difficult for the user to compare values in cross-platform settings. Additionally, it requires a contiguous allocation of memory.

SciPy [43] is an extension of the NumPy library. SciPy stands out as a comprehensive collection of mathematical algorithms and utility functions. It provides high-level commands and classes for efficient data manipulation and visualization. Overall, it can transform a Python session into a robust data-processing session rivaling MATLAB.

SciPy's integration with Python creates versatile equation creation using a readable design. The library's expansive collection of additional modules, contributed by developers globally, covers diverse areas of programming from parallel programming to web interaction. The high-level commands and classes in SciPy simplify complex mathematical operations in the same regard that NumPy does. However, SciPy has more flexibility than NumPy in that it has no constraints of homogeneity.

However, it does bring limitations. The primary difference of SciPy versus NumPy is the processing speed. SciPy was developed in Python while NumPy was developed in C, so inherently NumPy completes operations quicker. Not to mention, SciPy is a larger library, which presents more dependencies and lookups that may slow down specific functions. Also, SciPy's official documentation is still a work in progress, so there exists the possibilities of errors or incomplete information while referencing functions.

Pandas [44] is another highly popular Python software library that specializes in data manipulation and analytics. Pandas offers extended data structures for holding diverse types of labeled and relational data. It facilitates a range of operations, including merging, joining, reshaping, and concatenating data. It is also built off the foundation of NumPy, requiring its usage for optimal functionality.

Pandas also presents many benefits. Its ability to reshape and pivot datasets, along with merging and joining operations, makes it a powerful tool for handling complex data structures. The DataFrame object in Pandas adds another layer of functionality, enabling data manipulation and indexing. Other noteworthy features include robust support for data alignment, integrated handling of missing data, and extensive tools for reading and writing data in different file formats.

However, Pandas also presents certain drawbacks. Its memory consumption is relatively high compared to NumPy, making it less efficient for handling large datasets. The indexing of Pandas is slower than NumPy arrays, and the library may not be as performant as NumPy for datasets exceeding 500,000 rows. Additionally, pandas does not allow easy appending of data entries as quickly as regular Python.

The table below summarizes the advantages and disadvantages of each of these libraries. This way, each can be directly compared, and a result can be determined.

Library	Advantages	Disadvantages
NumPy	Efficient handling of numerical data. <ul style="list-style-type: none"> • Fast execution of operations on arrays. • Supports matrix multiplication and linear algebra. • Supports object-oriented approach using ndarray. • Efficient handling of arrays, matrices, and large datasets. • Broadcasting functions for uneven array shapes. • Consumes less memory compared to Pandas and SciPy. • Easy appending of data entries. 	<ul style="list-style-type: none"> • Primarily designed for numerical data. • SciPy and pandas extend NumPy's features. • No complex data structures. • Limited statistical functions. • Memory overhead for large arrays.
SciPy	<ul style="list-style-type: none"> • Efficient handling of numerical data. • Optimized numerical operations and algorithms. • Extensive linear algebra routines and solvers. • Object-oriented design with high-level commands. • Additional modules for various specific data domains. • Broadcasting functions for arrays. 	<ul style="list-style-type: none"> • Primarily designed for numerical data. • Memory consumption is higher than NumPy, and is dependent on operation and size. • Indexing may be slower for certain operations. • Appending data operations are slow.
Pandas	<ul style="list-style-type: none"> • Supports numeric, alphabetic, and heterogeneous data. • Efficient for data manipulation. • Object-oriented with DataFrame and Series. • Extensive tools for data manipulation and analysis. 	<ul style="list-style-type: none"> • Limited linear algebra support compared to NumPy and SciPy. • Does not support broadcasting. • Higher memory consumption compared to Numpy. • Indexing is slow for Series. • Limited support for appending data quickly.

Table 9: Comparison between three popular Python libraries with matrix operation utilities.

For the purpose of AutoCaddie, the selected library needs to balance processing speed with resource consumption. In terms of operations, it is important to maintain a list of frames, crop an array to the confines of a bounding box, and apply zero-padding to match predefined dimensions. Overall, NumPy stands out as the ideal choice for this project for several reasons. When considering processing speed and resource consumption, NumPy is the strongest in both subjects. Of the three libraries, NumPy excels the most with large datasets due to its nature of being programmed in C. It can also easily maintain a list of frames using its array object, ndarray, which is optimized for performance over convenience, which does not pose a threat to the developers. This also makes operations involving quick and parallel computations efficient, specifically in regard to the quaternion data calculations.

The quaternion data calculation can be summarized into a single equation. However, by conjoining all adjacent data points into vectors, which we can represent by the matrix below, this equation can be simplified to a single call. Then, iteratively, the resultant matrix

can be analyzed for large shifts in rotational movement, which satisfies possible linearity objectives in AutoCaddie.

$$\Theta = \cos^{-1}(2 \cdot \langle Q1, Q2 \rangle - 1_{m-1}),$$

$$\Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dots \\ \theta_{m-1} \end{bmatrix}, Q_1 = \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_{m-1} \end{bmatrix}, Q_2 = \begin{bmatrix} q_2 \\ q_3 \\ \dots \\ q_m \end{bmatrix}, 1_{m-1} = \begin{bmatrix} 1 \\ 1 \\ \dots \\ 1 \end{bmatrix},$$

$$\Theta, Q_1, Q_2, 1_{m-1} \in R^{m-1}$$

Equation 8: Using a 2D matrix of quaternion data points to determine a vector of linearity data.

Moreover, NumPy's array manipulation capabilities, including cropping arrays to bounding boxes and applying zero-padding, are implemented with high performance in mind. Specifically, using array slicing, portions of images containing the contents within a bounding box can be easily separated from the rest of the array. Also, NumPy provides `numpy.pad`, which allows for array to be padded with zeros efficiently. This function is flexible and can be applied dynamically to different image input dimensions.

It may be the case that SciPy and Pandas have similar methods for this logic, but it is also the case that both libraries contain extra resources which are not necessary for the scope of AutoCaddie. Combine this fact with their extra memory management due to the large number of resources, and overall, both libraries negatively impact the storage requirements of AutoCaddie. Therefore, due to its high performance and smaller package size, NumPy is the correct library to utilize.

3.5.4 Frame Extraction

Moreover, to correctly apply these matrix operations, the system needs image data. As the video cameras record the user completing a golf swing, the live broadcast is sent to the system, and stored as video data. From there, AutoCaddie must rely on another Python library to extract frames from these videos for the calculations. There are three packages that are considered for this project. Specifically, they are OpenCV, FFmpeg, and video2images. Each of these libraries aim to accomplish the same task, but provide subtle differences in their implementations.

OpenCV [45] or Open-Source Computer Vision Library, stands as a versatile and powerful tool for video editing, image processing, and computer vision tasks. Its capabilities extend to techniques such as image scanning and face recognition, making it a crucial component in various applications. With OpenCV, users can perform a multitude of operations on videos, including frame extraction and cropping.

Since OpenCV is the most popular computer vision library, it is available with many benefits. Firstly, it has extensive functionality, offering a wide array of functions for image and video processing. In the context of AutoCaddie, it has methods built-in for frame extraction and cropping, which is important for the data preprocessing. Additionally, OpenCV benefits from a large community of developers, ensuring regular updates, but fixes, and improvements. Lastly, it is cross-platform supported, making it available for use with Windows, Linux, and macOS.

However, several drawbacks of OpenCV are as follows. It has a steep learning curve due to its extensive feature set, which might prolong the development process of this project. Additionally, some advanced functionalities have complex implementations, leading to challenges in understanding and properly utilizing them.

FFmpeg-python [46] is a versatile tool for video processing. While numerous Python FFMpeg wrappers exist, what sets ffmpeg-python apart is its exceptional support for complex filter graphs, making it suitable for both simple and intricate signal graphs. This library simplifies video manipulation by offering operations like flipping, trimming, overlaying, and more.

The advantages of FFMpeg are as follows. It excels with supporting complex filter graphs, enabling users to create intricate video processing pipelines effortlessly. It also utilizes a Pythonic interface, which makes video editing more readable and accessible. Lastly, it has a flexible installation process, which allows users to choose platform-dependent methods.

However, there are several major flaws with FFMpeg in the context of this project. Similar to OpenCV, there is a large learning curve with this library. It utilizes command-line interfacing, which poses a challenge for developers. Also, users need to ensure proper installation by configuring the \$PATH environment variable, which can conflict with an existing environment variable. Lastly, FFMpeg is targeted towards complex video editing, which differs from the goal of AutoCaddie to analyze a short video of a golf swing. This library solves a problem entirely different than what is being sought after by this project, which makes OpenCV a stronger choice. Therefore, FFMpeg will not be used by AutoCaddie.

Videos2images [47] is another Python library designed to simplify the process of converting volumetric video data into image frames. The library addresses the need for customizable frame capturing rates during video annotation tasks, allowing users to extract frames from a video or a specific video segment efficiently. Videos2images proves to be beneficial in scenarios where specific frames are required for video annotation, and the flexibility it provides in terms of output format and output directory makes it a strong tool for video processing tasks.

There are several benefits for videos2images. Firstly, it aims to simplify the video-to-image frame conversion process, providing a user-friendly interface for efficient extraction. Next, it allows for users to define custom frame capture rates, enabling precise

control of the number of frames extracted per second. Also, it allows users to specify the output format and directory, offering flexibility in organizing and saving the extracted frames. Lastly, it also supports command-line usage.

However, there are also several drawbacks for `videos2images`. First, this library depends on external Python libraries such as `tqdm`, `imageio-ffmpeg`, and `moviepy`, which may introduce unnecessary dependency issues within AutoCaddie. Next, it has limited video format support by only covering `.mp4`, `.avi`, and `.mkv`. Lastly, the command-line interfacing can prove to be confusing, creating another learning curve necessary for the developers to overcome.

Between OpenCV and `videos2images`, OpenCV is the superior choice for AutoCaddie for a multitude of reasons. Firstly, OpenCV contains a more extensive documentation repository, which allows for the learning curve of the library to be quicker to overcome for the developers. Similarly, OpenCV has robust community support, which helps ensure that this library will have minimal errors during deployment. Secondly, OpenCV's code implementations are more straightforward in this context, which enhances the readability of AutoCaddie's codebase. Specifically, the code implementation of frame extraction of a video is shown below. [A1]

Moreover, the code implementation for cropping a frame to specified dimensions in the context of a bounding box is shown below [A2].

Furthermore, when comparing the two libraries, OpenCV stands out as a self-contained library, whereas `videos2images` introduces external dependencies. This distinction ensures that AutoCaddie remains more self-sufficient and less prone to dependency errors. Putting all of this together, OpenCV will be utilized within the AutoCaddie project.

3.5.5 Required Libraries for a CNN

Keras [48] is a deep learning API written in Python. It stands out as a critical component of the TensorFlow machine learning platform (TensorFlow is explained further in **Section 3.5.6**). Keras prioritizes facilitating rapid experimentation, emphasizing the importance of transitioning from ideas to results swiftly to enhance the research process. As fine-tuning the object detection model becomes a concern during AutoCaddie prototyping, it is beneficial to have quickly-adaptable model programming, which is what Keras provides. Keras has demonstrated industry-strength performance and scalability, with prominent users such as NASA, YouTube, and Waymo.

In the context of image classification, Keras introduces developers to fundamental data structures: layers and models. The Sequential model, a linear stack of layers, serves as a simple starting point, while the Keras functional API enables the construction of complex layer graphs. Keras facilitates model configuration through compilation, where learning processes are specified with loss functions, optimizers, and metrics. The library supports an iterative training process an evaluation on test data, making it accessible for users at various skill levels. It is worth mentioning that, at a high level, the iterative training process occurs before deployment on AutoCaddie, so users will not have to worry about object

detection taking extensive time periods to compute. Rather, this is a concern the developers need to factor into.

Scikit-learn (sklearn) [49] is a machine learning library in Python that provides a range of tools for working with various machine learning tasks. sklearn offers a comprehensive implementation, including support for classification, regression, and outlier detection.

Also, sklearn emphasizes the optimization of parameters during model training. Careful selection of parameters is important but can also be determined via brute force methods during development such as GridSearchCV, which is an algorithm that tests every possible combination of parameters and navigates to the best performing combination.

In short, Keras is a deep learning API that supports many deep learning operations, notably MobileNetV2's model integration. Also, sklearn is another machine learning library that defines a large range of tools for working with machine learning tasks. Overall, Keras and sklearn provide the necessary tools needed to implement these topics into AutoCaddie.

3.5.6 Network Libraries: TensorFlow vs. PyTorch

For the AutoCaddie project, TensorFlow [50] is an apt choice. TensorFlow provides extensive support for image preprocessing and augmentation, which can be beneficial for preparing golf swing data for object detection and subsequent analysis.

While the project has settled on using TensorFlow and Keras for the CNN implementation, PyTorch also offers equivalent functionalities that could have been considered. The Torchvision [51] module provides access to pre-trained models, including those optimized for mobile devices, with easy-to-use interfaces for computer vision tasks. PyTorch's dynamic computation graph could allow for more customized adaptations of MobileNetV2 if necessary, which may be advantageous for specific golf swing detection requirements. TensorFlow's comprehensive toolset which includes the TensorFlow Object Detection API and TensorFlow's data API can be directly applied to the AutoCaddie project. These tools streamline the model development pipeline, from data loading and preprocessing to model training and evaluation. The use of MobileNetV2 through Keras in TensorFlow offers the following specific advantages for AutoCaddie's development.

- **Rapid Development:** This is due to the ease of using Keras with TensorFlow. This accelerates the development of object detection models and enables rapid iteration.
- **Performance Tuning:** With TensorFlow's profiling and optimization tools this will be invaluable for tuning MobileNetV2's performance on the AutoCaddie laptop setup.
- **Deployment Readiness:** This is because TensorFlow's model serialization and deserialization functionalities facilitate the saving and loading of models, making the transition from development to deployment seamless.

By choosing TensorFlow integrated via Keras, the AutoCaddie project benefits from a streamlined, efficient workflow that is both powerful and flexible enough to meet the

demands of golf swing analysis on a laptop-based platform. This setup ensures that the project can maintain high performance without the need for extensive computational resources.

3.5.7 Graphical User Interface Libraries

Up to this point, all of AutoCaddie's internal processing has been defined through code. Python is the overbearing language of this project and wraps all the components into a centralized style for accessibility and compatibility purposes. In terms of simplifying complex mathematical operations, matrix calculations and image padding can be completed using NumPy. For extracting frame data from the live video feed, OpenCV provides the most overwhelmingly supported solution which will integrate high accuracy into this process. Also, Keras, sklearn, and Tensorflow are additional libraries that facilitate the usage of deep neural networks, whether it be smart preprocessing algorithms or the development of an internal CNN. However, the remaining piece of the puzzle to complete AutoCaddie's software implementation is to present the output to the user in a well-formatted method. That is where the importance of using a GUI is apparent.

In the realm of Python GUI development, various different frameworks offer distinct features and benefits. Each framework caters to a specific preference and requirement, making it essential to explore and compare strengths and weaknesses. For the purpose of AutoCaddie, five prominent Python GUI frameworks will be analyzed: PyQt5, Tkinter, wxPython, Kivy, and PySimpleGUI.

PyQt5 [52] is a powerful and versatile GUI framework, offering extensive capabilities for developers. Its integration with the Qt framework, active community support, and comprehensive documentation make it a compelling choice for building cross-platform desktop applications. However, developers need to remain aware of licensing considerations and be prepared for a large learning curve.

The strengths of PyQt5 are as follows. Firstly, it is powerful in the fact that it allows developers to develop visually appealing, as well as functional, applications. Also, it has cross-platform compatibility, making it able to be developed and deployed across various operating systems. Next, its Qt integration allows developers to leverage the speed of C++ while working with Python, which is a solution to the major drawback of the Python language. Additionally, PyQt5 benefits from a largely active community, ensuring continuous improvements and support. This feeds into a comprehensive documentation, making it easier for developers to utilize all of its features. However, PyQt5 shoulders certain weaknesses. Mainly, PyQt5 is licensed under the GPL license. Developers must be mindful of licensing considerations, especially in commercial projects. Also, creating standalone executables may be challenging, and the resulting files may be larger compared to other frameworks. This can impact the distribution potential of AutoCaddie on deployment.

Tkinter [53] is an open source Python GUI library which is renowned for its simplicity and user-friendly features. Tkinter comes pre-installed with Python, eliminating the need for additional installations and forwarding its accessibility. Tkinter's advantages lie in its ease

and speed of implementation, flexibility, stability, and the absence of the need for external downloads.

Some advantages of Tkinter are as follows. Firstly, Tkinter is one of the quickest GUI toolkits to implement compared to other popular choices. Also, it is flexible and stable. Next, it is largely accessible, as it is included in the installation of Python. Overall, it is referred to as the “beginner friendly” framework because, along with all of the accessibility and simple to implement code, it is very simple to understand, eliminating the large learning curve for developers.

However, Tkinter does have drawbacks. First, it does not include advanced widgets. To have access to advanced features and tools, external downloads are necessary, which defeats a large aspect of the accessibility of this framework. Additionally, it does not have a similar tool like the Qt Designer within PyQt5. Also, based on community feedback, sometimes Tkinter makes debugging difficult, and it does not fully have a reliable UI.

wxPython [54] stands as a cross-platform GUI toolkit for Python. It serves as an interface to the wxWidgets library, a popular cross-platform library written in C++. Notably, wxPython is open-source, aligning with Python free usage philosophy. Additionally, wxPython Phoenix is a project aimed at enhancing the speed while eliminating complexities of the original wxPython. Both versions of this GUI framework are available with stable cores.

The advantages of wxPython are as follows. Significantly, wxPython has a native look and feel across platforms, which allows for the GUI elements to integrate seamlessly across separate operating systems. This is attributed to its wrapping of native wxWidgets components. Additionally, wxPython offers flexibility in coding, allowing for specific adjustments to accommodate platform differences.

However, wxPython presents other weaknesses. Since this framework advertises heavily on cross-platform compatibility, the fact that its installation on macOS is difficult undermines its very principles. Additionally, despite it providing many features, it lacks the same level of flexibility as PyQt5, especially with customization options. Not to mention, the nuances with cross-platform development adds complexity that developers may stray away from.

Kivy [55] is a robust and open-source Python library that is also designed for building cross-platform applications. This library enables the creation of mobile applications using Python, emphasizing user-friendly and interactive UIs. Kivy’s framework supports the development of applications with distinct looks and feels across different platforms.

The strengths of Kivy are as follows. Kivy excels in cross-platform development, allowing developers to create applications that can run in various environments. It also provides built-in support for multi-touch interactions, making it well suited for mobile development. Kivy also introduces the Kv language, which facilitates the creation of customized widgets. This intermediate language streamlines the process of defining classes and

widgets, making it easier to configure and tailor applications. Lastly, Kivy holds the ability to have developers write code once and reuse it across different platforms, eliminating platform-specific implementations.

However, Kivy presents many weaknesses as well. Kivy's user interface is non-native, which poses challenges for users unfamiliar with the platform. Additionally, the package size of the Kivy framework is large in comparison to the other libraries, which is because of the inclusion of the Python interpreter within the Kivy package. Also, Kivy suffers from small community size, resulting in limited support for developers. Support requests may take longer to be addressed, and less overall debugging has been completed on the framework, suggesting a higher probability of roadblocks for development. Lastly, the documentation for Kivy is deemed improper and incomplete, which makes utilizing the entire framework difficult.

PySimpleGUI [56] is a Python package designed to simplify GUI development by transforming frameworks like Tkinter, Qt, and wxPython into more straightforward interface. It streamlines window definition using Python core data types such as lists and dictionaries. Also, it changes event handling from a callback-based model to a message passing one, allowing for a more intuitive approach. The package doesn't mandate an object-oriented structure, and its simplicity doesn't limit its usability to only basic problems.

Therefore, PySimpleGUI presents the following strengths. Primarily, it provides an extremely easy-to-learn interface compared to the other GUI frameworks, requiring less code for the same functionality. Also, it can be implemented using either pip install or the downloading of a single source code file. Similar to many other frameworks, it supports cross-platform compatibility, and allows the developers to run the same GUI by changing just one line of code. It also have extensive documentation, tutorial, example code, and videos, and can be integrated with various popular packages, notable OpenCV which is being utilized for bounding box annotation within AutoCaddie's live video feed. Some more features are as follows: it is highly customizable, open source, backwards compatible with support for both Python 2.7 and Python 3, and receives frequent updates.

However, PySimpleGUI also presents various weaknesses. Traditionally, PySimpleGUI deviates from the normal Pythonic approach, which can be tricky for veteran developers who are used to creating GUI applications prior to AutoCaddie's development. Additionally, the documentation, despite being extensive, lacks structure and adopts a case-by-case approach rather than a comprehensive walkthrough. PySimpleGUI also lacks specific tools that other frameworks possess, such as a forms designer. Lastly, this framework is considered best only for simple GUI development, and might not be suitable for complex commercial projects such as AutoCaddie.

In short, each of these five frameworks presents many strengths and weaknesses. To gain a better understanding of the big picture, each of the frameworks are presented in the table below.

Feature	PyQt5	Tkinter	wxPython	Kivy	PySimpleGUI
<i>Open Source</i>	Requires a commercial license.	Yes	Yes	Yes	Yes
<i>Cross Platform</i>	Windows, Mac, Android, Raspberry Pi.	Windows, Mac, Linux.	Windows, Mac, Linux.	Windows, Mac, Linux, Android, iOS, Raspberry Pi.	Windows, Mac, Linux, Raspberry Pi.
<i>Python Version</i>	Python 3 only.	Python 2 and 3.	Python 2.7 and 3.	Python 2 and 3.	Python 2.7 and 3.
<i>Installation</i>	Requires separate installation (not pre-installed).	Usually comes bundled with Python.	Can be installed using pip.	Requires installation, not pre-installed.	Can be installed via pip or single source file.
<i>Ease of Learning</i>	Moderate, QtGUI and QtDesigner modules assist.	Beginner-friendly, simple layouts and widgets.	Simple to use with good documentation.	Moderate, Kv design language, good community support.	Extremely easy, especially for beginners.
<i>Widget Collection</i>	Extensive set of modern and customizable widgets.	Vast collection, including standard and Ttk widgets.	Native look with standard widgets.	Customizable look with built-in widgets.	Provides a variety of widgets, may lack some features.
<i>Form Designer</i>	QtDesigner module provides a visual interface.	No built-in form designer.	No built-in form designer.	No built-in form designer.	No built-in form designer.
<i>Commercial Use</i>	Requires a commercial license.	Free to use.	Free to use.	Free to use.	Free to use.
<i>Community Support</i>	Strong community support and resources.	Well-established, abundant resources.	Active community.	Growing community, online resources available.	Active community, official Cookbook and Udemy course.
<i>GUI Design Focus</i>	Well-suited for both small- and large-scale applications.	Suitable for desktop apps, less focus on multimedia.	Suited for simple portable desktop applications.	Primarily designed for touchscreen-oriented interfaces.	Simplified GUI development, focus on simplicity.

Table 10: Comparison between five popular Python GUI frameworks.

By analyzing the table above, the following conclusions can be drawn. PySimpleGUI is suitable for beginners due to its simplicity and extensive documentation. WxPython is a strong choice for simple portable desktop applications as it provides native widgets and ease of use. For powerful desktop applications, PyQt5 is recommended due to its comprehensive support and modern interfaces, but developers need to recognize the licensing. For mobile application development, Kivy is recommended as it is designed for mobile support and touchscreen integration. Tkinter is another valid desktop application framework, and provides benefits over PyQt5 in being open source, easier to understand,

and more accessible. However, Tkinter does not have all of the available tools that PyQt5 provides, so for production it is less recommended.

In the context of AutoCaddie, a complex GUI design is necessary. The system needs to be able to interact with the user, provide calibration steps, format and present readable output, and showcase accessibility at a high enough level to maintain interest with the user. Therefore, PySimpleGUI is not suited for this application, as it does not possess enough features to justify its selection. Similarly, due to the nature of AutoCaddie's software deployment being a desktop application, Kivy is rendered unusable.

Overall, Tkinter is the best selection for AutoCaddie. It offers portability between different computing devices, provides tools that are comprehensive enough to support AutoCaddie's requests, and gives developers a non-stressful learning curve in comparison to wxPython and its widgets. The primary justification for selection of Tkinter over wxPython is the ease of access for the developers to ensure quick development of the software. Additionally, PyQt5 is not selected due to the licensing restraints, where the budget of this project outweighs the framework's benefits. Therefore, Tkinter is the framework that will be utilized for creating an intuitive and accessible GUI for communication with the user.

Chapter 4: Related Standards and Realistic Design Constraints

4.1 Standards

In this section, we delve into the usage of standards for maintaining consistency, safety, and compatibility throughout the AutoCaddie project.

The American National Standards Institute (ANSI), [57] functioning as a non-profit entity, orchestrates the intricate world of standards and conformity assessment. Its mission revolves around fostering global competitiveness and enhancing the quality of life in the U.S. It acts as a conduit for different stakeholders, including standards developers, government bodies, manufacturing entities, and various associations. Overall, ANSI ensures the integrity of standards development by accrediting procedures of over 240 organizations. The American National Standards (ANS) process mirrors this precision, involving consensus, public scrutiny, and a meticulous appeals system. ANSI's vigilant oversight ensures the entire standards framework operates seamlessly, akin to the careful observation of IMU data's linearity in a swing, delivering reliable user feedback.

Throughout this chapter, standards and constraints relevant to this project will be researched within ANSI and ANSI-accredited standards. From there, each topic will be explained in terms of significance and overall contribution.

4.1.1 Hardware Standards

Regarding the implementation of hardware, the sensor data collection and data transmission must adhere to various standards. Most particularly, data communication standards and circuit design standards must be accounted for. For circuit design standards, standards relevant to the designing of a PCB device must be accounted for. These standards may also relate to standards for data communication, which includes standards on communication means incorporated within the AutoCaddie project.

4.1.1.a IPC-2221

IPC-2221 is a widely recognized industry standard document that provides guidelines and requirements for the design of PCBs. Developed by the Institute for Interconnecting and Packaging Electronic Circuits (IPC), the standards document addresses a range of topics with auxiliary documents and standards categories. These topics of standards include board layout, conductor sizing, thermal management, and electrical characteristics. Overall, the IPC-2221 standards ensure that designed PCBs are developed with reliability and manufacturability in mind. Some relevant discussions of standards and guidelines relevant to the PCB design for the AutoCaddie system are as follows:

Signal Integrity: This section discusses the importance of maintaining signal integrity in high-speed digital and analog circuits. It covers topics such as controlled impedance, trace length matching, and signal routing techniques, which are crucial for reliable data transmission. Most importantly, this section outlines the placement of parts with respect to cycle frequencies and power sources, with lower-frequency parts being separated from

the power source and higher-frequency parts. This considered placement of elements maintains signal integrity between active elements and reduces chances of crosstalk. This section also discusses board spacing width with respect to applied voltage. For example, voltage levels from 0 to 15 require a minimum spacing of 0.05 to 0.1mm.

Power Distribution: Proper power distribution is essential for stable and noise-free operation, which is critical when dealing with data collection and transmission. This section provides guidelines for designing power planes, decoupling capacitors, and managing power delivery networks. A primary consideration that is encouraged by the document is to use decoupling with respect to power and ground traces.

Grounding: Proper grounding is crucial for reducing noise and maintaining signal integrity. The IPC-2221 document discusses grounding techniques and considerations that are relevant to data transmission circuits. As an extension of the guidelines for power distribution, the document outlines that if different grounds are used, particularly those for different measurement references, the grounds should be kept as separate as possible.

Thermal Management: In some cases, data collection and transmission circuits can generate heat, which can affect performance and reliability. This section addresses thermal management techniques, including heatsinking and proper component placement. Considerations of heat transfer are emphasized, including those of the mechanics of heat conduction and convection.

Environmental Considerations: Data collection and transmission devices may be deployed in various environments. The IPC-2221 document provides guidelines for selecting materials and design practices that can enhance the PCB's reliability under different environmental conditions. Considerations of environmental conditions should account for the application of the PCB and its potential environments, such as impact and environmental electromagnetic noise.

4.1.1.b I2C Standards

Standards for I2C Communication are written by NXT Semiconductor, originally Philips Semiconductor.

1. I2C Communication traditionally contains 2 bus lines: a unidirectional serial data line (SDA) and bidirectional serial clock line (SCL).
2. I2C boasts collision detection and arbitration to prevent data corruption if two or more controllers simultaneously initiate data transfer, making it a true multi-controller bus.
3. Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100kb/s in the Standard mode, up to 400kb/s in the Fast-mode, up to 1 Mb/s in the Fast-mode Plus, or up to 3.4 Mb/s in the High-speed mode.
4. Serial, 8-bit oriented unidirectional data transfers up to 5 Mb/s in the Ultra-fast mode.
5. On-Chip filtering rejects spikes (outliers) on the bus data line to preserve data integrity.

6. The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance. More capacitance can be allowed under the following conditions: lowered operating frequency, using devices with higher drive current (Fast-Mode Plus), bus buffers (divide the bus into segments so that each segment has a capacitance below the allowable limit), switched pull-up circuit (alternately accelerate rising edges by switching a low value pull-up in and out when needed).

I2C will be the communication mode which collects data from the IMUs and transmits it via soldered wires (2 communication lines per device) to the PCB. The speed capabilities of I2C ensure that it will not be the limiting factor in terms of latency, and the built-in anti-corruption features ensure that data is significantly less likely to be damaged through the delivery process.

4.1.1.c Bluetooth Communication Standards

The HC-06 Wireless Transceiver to be implemented on the PCB is characterized as Bluetooth Class 2, and thus the characteristics of Bluetooth Class 2 will be summarized below.

1. Bluetooth power class 2 generally has a safe operating range of approximately 30 meters but can range higher depending on the specific device.
2. Bluetooth operates between 2400 – 2480 MHz, 2.5 GHz, and 3.5 GHz, but generally at 2.4 GHz in a globally available ISM band.
3. Bluetooth power class 2 has a maximum output power between 0 and +4 dBm.

Bluetooth is intended to be the exclusive wireless communication mode employed on the AutoCaddie project. Its projected use is to transmit a standard packet of data, approximately 13.68 kB in size, from the PCB directly to an external transceiver in a laptop or similar device. With a range of approximately 30 meters, a safe distance from the golf swing can be achieved, ensuring that no peripheral devices are damaged through the data collection process. The +4dBm output power makes the possibility of losing data due to signal weakness very unlikely, especially if the distance standard is adhered to.

4.1.1.d UART Communication Standards

UART will be the communication mode which transmits data from the MCU to the wireless transceiver module (HC-06). UART is a very widely used serial communication option, one of the staples of peripheral communication in computing.

1. The transmission process consists of 1 START bit, data bits (5, 6, 7, 8), 1 PARITY bit (optional), and 1 STOP bit. The following is a visual representation of this arrangement:

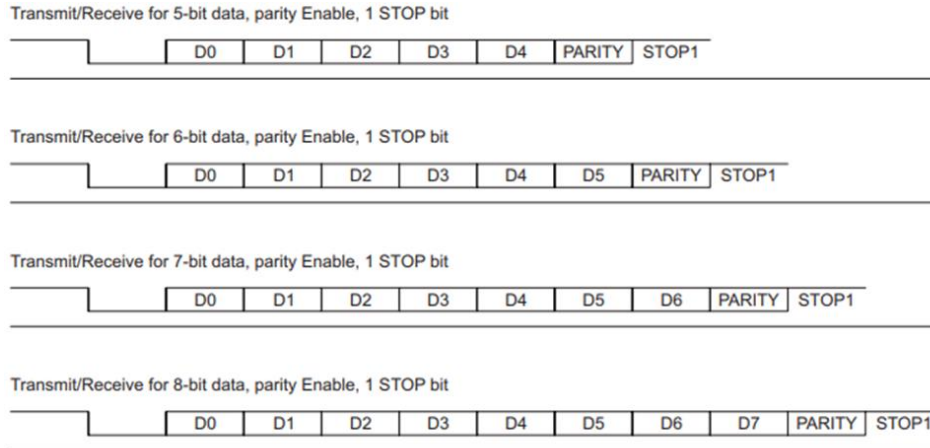


Figure 18: UART Protocol Formats.

2. UART peripherals adhere to a data rate referred to as “Baud Rate,” in b/s. Most devices operate in a rate range of 2400 b/s to 3000000 b/s. The default reception rate is 9600 b/s.
3. UART adheres to a hardware implemented queue called FIFO (First In First Out) that forces data to be processed in order of reception.
4. UART can operate synchronously or asynchronously. Asynchronous communication transmits data without a shared clock signal, while synchronous communication uses a clock signal to ensure precise timing. The asynchronous case is most common for general-purpose data transfer, including the AutoCaddie project.

4.1.2 Software Standards

In terms of the software implementation, there are many aspects of the project to account for. Regarding device communication, there are necessary standards for data validation and device benchmarks. For utilizing Python, standard syntax is necessary. Regarding the machine learning model, there are training standards, data processing methods, and validation tests to reference. Lastly, for the user experience, there are guidelines for the design, accessibility, and interactivity of the feedback.

4.1.2.a Device Performance Benchmarking

In terms of measuring the complexity of the model, this will need to be addressed during implementation. However, PyTorch is implemented with a benchmark module [58] to measure and compare code performance.

The benchmark module is used to validate that code meets performance expectations by providing proper insights about storage, memory allocation, optimizations, among other relevant details for a solution. They often utilize call graph analyses to delve deep into the internal workings. Currently, benchmarking can be applied to evaluate a multitude of performance concerns. During testing, it can be used to measure execution time and suggest possible optimizations. It can also help de-noise the data in case potential bad readings skew the input stream.

4.1.2.b Syntax Standards

For readability purposes, it is important to maintain a standard syntax style. Therefore, by referencing official Python documentation, we are recommended to use PEP 8, [59] or Python Enhancement Proposal 8.

PEP 8 serves as the official style guide for writing clean, readable, and consistent Python code. It encapsulates the Python community's best practices for code formatting, ensuring that Python codebases are maintainable and comprehensible. PEP 8 advocates for a uniform coding style, with recommendations covering aspects such as indentation, variable naming, whitespace usage, and commenting.

In PEP 8, code should be indented with four spaces per level, ensuring consistency in code structure. Variable and function names should follow a “lowercase_with_underscores” naming convention, and classes should use CamelCase. Whitespace should be used judiciously, with particular attention to maintaining consistent spacing around operators and after colons. Comments should be clear and concise, providing helpful explanations where necessary. Adhering to PEP 8 not only enhances the readability of Python code but also promotes collaborative development, making it easier for developers to work on projects collectively.

4.1.2.c Model Training Standards

Google, a leader in technology, has taken up the mantle of defining common model training standards for developers. In a comprehensive document, [60] Google outlines essential rules and insights gathered from extensive research discussions. These rules serve as beacons, illuminating the often-intricate path of machine learning model training.

The research paper distills Google's extensive experience into a set of fundamental principles for model training. In the following paragraphs, a list of rules will be given – with proper explanations and justifications of importance.

Rule #1 emphasizes the significance of understanding the problem at hand deeply. It stresses the importance of clarifying objectives, a foundational step often overlooked. Rule #2 echoes this sentiment, underlining the need for precise definitions of success, ensuring that everyone involved shares a common understanding of what constitutes a successful outcome.

Rule #3 introduces the concept of the “no free lunch theorem,” emphasizing that there's no universally superior algorithm. Instead, the choice of algorithm should be tailored to the specific task and dataset. Rule #4 emphasizes the iterative nature of model development. It stresses the importance of prototyping and continuous refinement, highlighting that the first iteration is seldom the final solution.

Rule #5 sheds light on the critical role of human evaluation. Despite the rise of advanced algorithms, human judgment remains indispensable. Rule #6 champions the value of a strong feedback loop, ensuring that models continuously adapt to changing real-world scenarios.

Rules #7 and #8 underscore the significance of data quality. Garbage in, garbage out, as the saying goes – emphasizing the need for clean, reliable data to train effective models. Rule #9 delves into the crucial aspect of feature engineering, emphasizing the art of transforming raw data into meaningful inputs for machine learning algorithms.

Rules #10 and #11 emphasize the importance of regularizing models, preventing overfitting, and ensuring their generalizability. Rule #12 touches upon the vital concept of ensemble models, where combining predictions from multiple models often outperforms individual ones.

The paper introduces Rules #13 to #17, highlighting the challenges associated with online learning and emphasizing the need for robust, scalable systems that can handle vast amounts of data in real time. Rule #18 introduces the essential notion of fairness and bias, emphasizing the ethical responsibility associated with machine learning models. Rules #19 to #28 cover an array of topics, including feature selection, regularization, and the intricate balance between complexity and simplicity.

Rules #29 to #37 delve into the nuanced aspects of training-serving skew, importance weighting, and the challenges posed by changing data. Rules #38 to #42 explore the complexities that arise in later stages of model development, emphasizing the need to align objectives, simplify ensembles, and seek qualitatively new sources of information.

In essence, this research paper provides a comprehensive guide, distilling Google's wealth of experience into a set of actionable rules that illuminate the often-intricate path of machine learning model training. By emphasizing the importance of clear objectives, continuous iteration, human evaluation, and adaptability, Google's standards provide invaluable insights, shaping the future landscape of machine learning.

However, it is also important to note that, although there are important rules to follow, many of the rules provide ambiguity, which allows for creative freedom upon implementing a neural network.

4.1.2.d Data Processing Standards

A good benchmark to measure the network complexity of AutoCaddie is to observe SwingNet, [61] which is a network architecture deployed to analyze the golf swing using GolfDB.

The data processing standards for the SwingNet neural network architecture involve various hyperparameters that are essential to its effective training and deployment. These standards are crucial for determining the computational resources required for this task and optimizing performance.

One critical aspect of these data processing standards is the choice of hyperparameters. In this regard, an extensive ablation study was conducted to identify suitable configurations. The hyperparameters explored include the input size (d), sequence length

(T), batch size, number of LSTM layers (N), and the number of hidden units in each LSTM layer (H). The study also investigated the impact of initializing the network with pretrained ImageNet weights and the use of bidirectional LSTM layers. The goal was to identify a configuration that maximizes performance while considering limited computer resources.

Interestingly, the study found that the use of pretrained ImageNet weights was essential for the model's training success, suggesting that initializing with pre-trained weights is critical for domain-specific tasks where appearance variation is minimal. The sequence length (T) had a significant impact on performance, emphasizing the importance of temporal context in golf-swing sequencing. Moreover, increasing the batch size was shown to dramatically improve performance, although it may impact convergence speed. Bidirectionality in LSTM layers led to a 12.1-point improvement in performance. These findings provide valuable insights into the hyperparameters crucial for SwingNet.

The study also revealed that the correlation between input size and performance did not follow the expected pattern, as input sizes of 160 and 128 outperformed the input size of 192. This suggests that choosing an input size of 160 is more cost-effective. These insights help establish a solid foundation for defining data processing standards that balance computational efficiency and performance in SwingNet.

It's worth noting that the model's reliance on pre-trained ImageNet weights led to the exploration of freezing some of the ImageNet weights without significant performance loss, allowing for larger sequence lengths and batch sizes. This finding contributed to the development of a baseline SwingNet configuration with an input size of 160, a sequence length of 64, and a single-layer bidirectional LSTM with 256 hidden units. This configuration, combined with pre-trained weights and freezing some layers, facilitated training on a single GPU with 12GB of memory and faster convergence.

Therefore, it can be noted that a model for AutoCaddie can be trained sufficiently with 12GB of memory. However, it is possible to utilize smaller sequence lengths for testing to reduce the memory requirements of the network. But, since 12GB memory is such a low requirement, it is not necessary to do so.

4.1.2.e Validation Standards

In short, the purpose of validation data in a neural network is to test that it has learned how to analyze input rather than simply memorize it. An analogy for the validation process is passing test cases on a programming assignment. The validation process ensures that a coding solution is robust enough to solve any input case, rather than having hard-coded solutions for specific input cases. By completing validation on the neural network training, we can ensure that the model has been trained appropriately.

This is implemented by segmenting the database used to train a model into separate groups: training, validation, and testing. The training data is iterated over initially, which creates a matrix of features that are analyzed by the system. Then the validation data is presented to ensure that the network learned the hyperparameters rather than memorized them. This set is the one that is used repeatedly to ensure the given model is the best fit

and allows for tweaking of the hyperparameters to find the lowest error. Once that is found, the test dataset is used to provide a final unbiased evaluation of the system. Therefore, there are presented questions of what makes a strong validation set, and what the signifying factors are that a model is well fit.

To gain a proper understanding of these standards, we can reference the following IEEE/EIA Standard developed under the considerations of ANSI:

- 12207.0 - Standard for Information technology – Software Life Cycle Processes (March 1998). [62]

Additionally, we can reference “Verification and Validation of Neural Networks: A Sampling of Research in Progress” [63] to understand how the validation standards directly apply to neural networks.

The validation plan encompasses any items subject to validation. As referenced prior, the pre-trained neural network (PTNN) is the item subject to validation, and the performance needs to be measured to ensure it falls within an acceptable range. This can be observed through a simulated testing process.

The testing process involves several advanced techniques. These techniques include unit testing, incorporating both requirements-driven (black box) and design-driven (white box) approaches. Requirements-driven testing focuses on aspects like computational correctness, boundary conditions, stress scenarios, error handling, and recovery. Design-driven testing delves into internal logic paths, examining various elements such as interfaces, critical code elements, and system response. Additional tests involve frequency response, phase and gain margins, failure modes and effects testing, sensitivity analysis, timing tests, system resource evaluation, and adaptation time assessment. These thorough testing methods ensure the accuracy, stability, and reliability of neural networks in diverse scenarios.

4.1.2.f Data Receiving Standards

As defined by the Software Life Cycle Process Standard (March 1998), the data acquisition process needs to be outlined by the acquirer. In terms of a software standpoint, the system needs to be fully capable of acquiring data and utilizing it properly. It can be summarized into a 5-step process:

- 1) Initiation
- 2) Request-for-Proposal preparation
- 3) Contract preparation and update
- 4) Supplier monitoring
- 5) Acceptance and completion

As outlined by the hardware, we are utilizing Bluetooth Class 2 with UART communication to transmit data from the PCB to the computer. The data recording process is monitored via time steps, so after a short duration, the computer will begin receiving data, and the PCB will send packets over wireless transmission.

It is also important to mention that there is no direct request for proposals, as the systems work independently to conserve power. The frame recording system begins by observing live-action sequencing using CV algorithms, and the IMU recording begins via a predefined timer. Once the CV algorithm initializes the action sequence, the computer begins acquiring video data, and at the end of the sequence, becomes available to receive data.

4.1.2.g User Experience Standards

As outlined by “User Experience Design (UX) Standards and Guidelines” by Arizona State University, [64] there are many design standards and guidelines that contribute to a strong user experience. These can focus on design, accessibility, usability, navigation, information architecture, content, and search engine optimization. Accessibility is a large topic and will be covered in a separate standard. To ensure consistency and a positive user experience, the Unity Design System, launched in August 2020, serves as a foundational framework across all asu.edu pages and subdomains.

The Unity Design System encompasses various components such as global headers, primary navigation, content blocks, form fields, and page templates, including those for degrees, events, news, and directories. The system is continuously evolving based on community feedback and contributions, emphasizing a collaborative and adaptive approach. To facilitate adoption, the system provides resources like the Arizona State University UI kit, Adobe XD templates, and guidelines for design elements like color palette, typography, spacing, icons, and focus states. Additionally, the website encourages community engagement through platforms like the Unity Design System Slack channel, inviting users to propose new components and provide valuable input, ensuring the ongoing refinement and effectiveness of the UX standards.

The Unity Design System provides documentation for each of these aspects. When designing the software application for AutoCaddie, it will be important to reference this documentation. Some examples include color palette, typography, spacing, navigation, and content.

4.1.2.h Accessibility Standards

There are essential guidelines and tools for ensuring the accessibility of websites and digital tools, in adherence to the Revised 508 Standards and the W3C Web Content Accessibility Guidelines (WCAG) 2.0 Level AA. [65] Web designers, developers, and content managers are pivotal in this process, focusing on areas like design, navigation, and usability.

Additionally, the Interagency Trusted Tester Program promotes a unified testing approach to reduce redundancy and save resources across federal agencies. The program employs ANDI (Accessible Name & Description Inspector) [66] and Color Contrast Analyzer (CCA) [67] to assess accessibility. ANDI, developed by the Social Security Administration, is a bookmarklet tool used for testing websites and web-based

applications. CCA, created by the Paciello Group, assesses color contrast when it cannot be programmatically identified by ANDI.

Elaborating on CCA's color contrast analysis, the Web Content Accessibility Guidelines (WCAG) provide accessibility standards and benchmarks. These guidelines specify contrast ratios, which are crucial for readability. For WCAG 2.0 Level AA, the contrast ratio for normal text should be at least 4.5:1, ensuring clear visibility. Larger text, vital for those with visual impairments, should have a ratio of 3:1. In the updated versions, WCAG 2.1 and 2.2 Level AA, a contrast ratio of 3:1 is required for graphics and user interface components, enhancing accessibility for various elements. For the highest standard, WCAG Level AAA, normal text requires a contrast ratio of 7:1, ensuring maximum clarity, while large text should have a ratio of 4.5:1.

Moreover, Universal Design [68] is a standard in which products are designed for usability with all people. This extends to those with disabilities. By adopting universal design principles, information throughout the software becomes more available to the consumer. The principles are outlined as follows.

Equitable Use ensures designs are marketable to diverse abilities, promoting equal access and appeal. *Flexibility in Use* accommodates various preferences and methods, ensuring ease of use regardless of individual differences. *Simple and Intuitive Use* focuses on clarity, consistency, and effective feedback for users of diverse backgrounds and literacy levels. *Perceptible Information* emphasizes communication effectiveness, using multiple modes and maximizing contrast. *Tolerance for Error* minimizes hazards and errors through careful design and feedback mechanisms. *Low Physical Effort* prioritizes user comfort, employing reasonable forces and reducing repetitive actions. *Size and Space for Approach and Use* ensures elements are reachable, graspable, and usable for individuals of all body sizes and abilities, fostering universal accessibility.

These principles collectively promote inclusivity and usability, fostering environments that cater to everyone's needs. By incorporating them into AutoCaddie, the user will be able to utilize the resources more comfortably.

4.1.3 Design Impact of Relevant Standards

In the realm of product design, adhering to the provided standards holds large significance. The standards serve as guidance for the pathway of design to be established. For AutoCaddie, maintaining all of the relevant hardware and software standards will shape the product into one that is safe, efficient, and accessible.

In terms of following the specified hardware standards and specifications, the design of the hardware systems of AutoCaddie should especially account for standards that are directly related to the application and purpose of each respective part.

Regarding PCB design, since multiple instances of data are active in transit during sensory data collection and transmission, part and trace placement should account for noise and potential cross-talk, so signal isolation and integrity should be a foremost

consideration. Additionally, the PCB should be robust enough to withstand environmental forces such as impacts and temperature shifts.

Additionally, considerations of communication standards should be attended to. With regards to the requirements of the AutoCaddie systems, standard common implementations and specifications of communication methodologies will be selected that meet the specific needs of the application of the transmitted data. For example, a sufficient speed standard will be selected for the UART communication via Bluetooth that is fast enough to transmit all necessary data in the shortest period of time possible.

In terms of adhering to the software standards, AutoCaddie will implement these topics through its digital system. These ideas apply everywhere within the deployed neural network, processing, and the output application.

Firstly, it is important to measure computational efficiency throughout the application creation process. AutoCaddie will utilize TensorFlow, which includes large amount of training calls and data management. By embedding benchmarks into the code, we can quantifiably measure the efficiency of the system.

Furthermore, this can expand to device validation standards. By benchmarking the performance of the system, we can analyze the effectiveness of a current training model during the validation process. If the model is not deemed to be effective, we can tune the hyperparameters and retest until a satisfactory performance measurement is reached. Utilizing benchmarking with the validation standards can ensure that the neural network system designed is optimal in performance.

On a similar note, to increase the efficiency of production, PEP8 will be referenced for a common Python coding style. That way, as trouble arises during development, the code will be more accessible to the developers, and debugging will simplify.

While incorporating TensorFlow, proper model standards will be adhered to. By referencing a similar architecture to SwingNet, we can embed TensorFlow into the system while designing an optimal analysis program.

For the device output to be used, the computing device just needs to contain 12GB of memory. This will be essential in standardizing the benchmarks, and for observing the performance of the completed system.

Separately, making sure that the system is designed properly will maintain data receiving standards. The camera and PCB data transmission will be sent separately, so there needs to be accountability with the design. The computer will use CV algorithms to define action sequencing on live video feed, which will define a final action and terminate the frame input stream. From there, it will be open for receiving packet data from the PCB, which will be initiated via a separate time step counter. Once the timer finishes, the data will be transmitted to the computer, where it will begin its analysis.

In terms of software design, there are many accessibility and user experience standards to adhere to. The list of features has been defined previously, so making sure that each of these points is accounted for will make the final design more transparent to the user. Using the recommended testing programs such as ANDI and CCS will ensure that the result is comfortable for the user, which will improve the experience.

4.2 Realistic Design Considerations and Constraints

4.2.1 Economics and Time

4.1.2.a Economics:

Initial Budgetary Constraints: With an estimated budget of under \$450, AutoCaddie's development process must carefully allocate resources to ensure that each component and stage of development remains within the budget. Overstepping this budget could hinder the project's completion or compromise its quality.

Component Selection: The choice of the Adafruit 3.7V 4.4Ah Lithium-ion battery, while being the most effective in terms of performance, is also the most expensive battery option at \$19.95. It is essential to balance the need for optimal components against the overall budget, ensuring that the quality of other components isn't compromised due to overspending on any particular item.

Unexpected Costs: There may be unforeseen expenses in the development phase, such as software licensing, additional components, or replacement of any malfunctioning parts. An emergency fund, or at least some flexibility within the budget, is advisable to cater to such eventualities.

4.1.2.b Time:

Battery Life: The chosen battery, the Adafruit 3.7V 4.4Ah Lithium-ion, provides several hours of charge, ensuring that the AutoCaddie system remains operational for the duration of a standard golf training session. However, optimizing the system to maximize battery life and ensure efficient energy usage is vital. Any reduction in battery life could lead to incomplete sessions and impact the system's utility.

Testing and Debugging: While building AutoCaddie, there will likely be a need for multiple testing and debugging sessions, especially given the complexity of integrating hardware components with the machine learning algorithm. Allocating sufficient time for these processes is essential to ensure a fully functional and reliable product upon completion.

Finalization: As this is a senior design project, it's crucial to allocate time towards the end of the project timeline for comprehensive review, documentation, and any final modifications. This ensures that the project is not only complete but meets the set standards and objectives.

Dependency Management: Some tasks will be dependent on the completion of others. For instance, the hardware cannot be fully tested without the completion of certain software modules. Managing these dependencies efficiently and ensuring that no task becomes a bottleneck is critical to maintaining the project timeline.

In conclusion, while the AutoCaddie project might not face traditional market-driven economic and time constraints, the need for meticulous planning, budget adherence, and strict time management is paramount to its successful and timely completion.

4.2.2 Environmental and Political

AutoCaddie is an unobtrusive, safe, environmentally friendly, and politically noninteractive platform which seeks to improve the golfing abilities of all users. Therefore, the following constraints are minor and can be considered beyond the project's main concern.

4.2.2.a Environmental Concerns:

AutoCaddie employs a Lithium-Ion battery of small size and yield. Lithium Ion is generally sourced from mines which utilize large, energy inefficient mechanisms of extraction. However, due to the minute quantity of this metal interacting with the project, the environmental impact of the Lithium-Ion battery will be negligible even on a large scale.

AutoCaddie will be accessible via readily available electronics such as laptops and desktop computers, and thus the user interaction component of the project will cause no further environmental impact than the production of these devices has already caused. The hardware components utilized on the PCB attached to the user consist of easily sourced and acquired metals with little environmental footprint such as copper, aluminum, iron, and gold. These metals are all present in exceedingly small quantities, and due to their chemically inert nature pose no environmental hazards, even upon malfunction.

4.2.2.b Political Impact:

AutoCaddie is a platform directed towards the improvement of athletic ability in golf, and therefore has no political affiliation, interaction, or direction whatsoever. However, golf as a sport could cause political concerns when viewed through a social lens; golf is colloquially perceived as a sport directed towards wealth: to play, one must purchase golf balls, drivers, and clubs, as well as pay to use fields which can range to be unattainably expensive for many. AutoCaddie will be accessible to all but has the potential to be sequestered naturally to those who can financially justify playing golf. The components present in AutoCaddie are not costly, and therefore the training platform will be affordable and accessible to the majority of golf players interested in real-time, accurate feedback on their swings. AutoCaddie will not contribute to the financial barrier-to-entry present in the sport of golf. Moreover, as AutoCaddie has little environmental interaction and potential for harm, there will be no cause for political activity surrounding the project.

4.2.3 Health and Safety

4.2.3.a Electrical Safety Constraints:

The implementation of AutoCaddie introduces health and safety concerns, primarily revolving around electrical hazards. A significant aspect to address involves the careful routing of power wires within the athletic sleeve worn by the user. If any of these wires were to be exposed, there exists a potential danger. However, it is noteworthy that the system operates on low voltage, thereby minimizing the associated risks. The intentional design of the sleeve to function with low power plays a pivotal role in ensuring overall user safety. This design choice not only reduces the likelihood of electrical accidents but also establishes the product as inherently safe for skin contact.

4.2.3.b Environmental Safety Constraints:

To enhance the longevity of the device and prevent unintended circuit shorting, it is imperative to shield the system from excessive water exposure. Environmental constraints dictate that the system should not be used in heavy rain, as moisture infiltration can compromise its functionality. Furthermore, considerations must be made regarding user-generated sweat, a common occurrence during physical activities. While human sweat is unlikely to disrupt the system significantly, the presence of live wires within a protective sheath further safeguards against potential issues, ensuring the device remains operational even in sweaty conditions.

4.2.3.c Physical Safety Constraints:

Incorporating AutoCaddie into athletic activities necessitates careful physical preparation. While the athletic sleeve helps alleviate arm soreness during golf motions, users may still experience stress in other body parts such as shoulders, back, elbows, or legs. Proper stretching routines play a vital role in mitigating these challenges. It is essential for the AutoCaddie team to provide users with comprehensive knowledge on appropriate stretching exercises to perform before engaging in the activity. Additionally, guidelines addressing how to manage sore muscles are indispensable. Taking responsibility for users' physical well-being, the AutoCaddie team should offer expert recommendations, enhancing the overall user experience and ensuring safety during golfing activities.

4.2.4 Ethical and Social

With regards to ethics within societal context, the primary concern that is raised by the development of the AutoCaddie project is the fact that it may be used to completely phase out and replace human professional trainers. In a vacuum, the AutoCaddie system is intended to provide a mode of feedback for those who either do not have the resources or ability to otherwise receive feedback while they train. However, as the AutoCaddie system may become available to all Golfers intending to train their swings, some may seek to use the AutoCaddie to replace professional trainers.

Though this risk does exist to some degree, it should be emphasized that the AutoCaddie system should be used as a supplement to traditional training methods if they are available, rather than a complete replacement. Without the presence of a professional AutoCaddie serves as a mode of adaptive feedback as an objective observer of a user's

feedback, allowing the user to understand some avenues of improvement that should be pursued. With the presence of a trainer, the AutoCaddie system becomes a supplement that can serve as a reinforcement or addition to provide guidance and can exist to process real-time data and data analysis. As the current technology of the AutoCaddie stands, it cannot serve as a complete replacement of the ability of professional trainers, as they can more readily provide exact tailored feedback without needing to be exposed to a precise database like the AutoCaddie does. Should this concern of human replacement still exist, however, the scope of the AutoCaddie system can remain limited to not overstep into the overall coverage of a professional trainer.

4.2.5 Manufacturability and Sustainability

The manufacturability constraints of the AutoCaddie system primarily revolve around the selection of the components and parts. To maintain availability and due to limitations to costs, the design of the system should prioritize cost-efficiency and availability by selecting market-available parts, such as in-market IMUs and video cameras. This reduces the need for proprietary parts, which can require higher short-term manufacturing costs and manufacturing resources. The PCB, which is one custom-designed part, will be outsourced to a single manufacturer, centralizes and compartmentalizes external manufacturing costs. Other custom parts, such as PCB housings and mounts, can be created and fitted in-house, thus reducing manufacturing and resource costs.

Sustainability is another consideration in the design of the AutoCaddie project. With the present design of the project, there is no actively disposable part of the system that is repeatedly spent and replaced. As such material waste as a result of the use of AutoCaddie is not a primary concern. However, as with all physical systems, degradation and wear does pose an issue. As such, parts selection and design must account for longevity and robustness. With these design facets in mind, Part wear and breakage will be reduced, improving sustainability as well as reducing waste and long-term costs.

Chapter 5: Comparison of ChatGPT with other Similar Platforms

5.1 ChatGPT

In the process of designing a research paper, ChatGPT, [69] developed by OpenAI, stands out as a cutting-edge conversational AI model. Its capabilities have significantly influenced various domains, including academic research and project development. In the context of AutoCaddie, ChatGPT can be an invaluable resource, offering many different features that can streamline the reading and writing process.

One feature ChatGPT offers is creativity. During the report creating process, it can be difficult to brainstorm fully coherent thoughts and ideas and immediately transcribe them into work. ChatGPT can foster creativity by generating diverse ideas and suggesting other approaches to problems present with the project. Unlike traditional research tools, it doesn't just provide information; it engages in dynamic conversations, stimulating creative thinking and offering unique perspectives.

Due to the large quantity of open-source data that ChatGPT is trained from, it serves as a vast repository of knowledge – enhancing research capabilities. Its extensive database provides access to information before January 2022, which, outside of neural networks, covers every topic from within the AutoCaddie project. Furthermore, it can expand on information rather than simply provide it, which creates a more personalized method of understanding new material.

ChatGPT is also efficient at creating drafts and providing revisions. The model's ability to generate coherent paragraphs has alleviated the burden of repetitive tasks, which allows the report process to be focused on the core content. Moreover, ChatGPT's grammar and style suggestions have enhanced the overall readability and professionalism of the document.

The language model and capabilities of ChatGPT can also be compared to other platforms. Traditional search engines include vast amounts of information updated with present events, as well as other documents and pieces of work that can be accessed by special instruction (not open source). ChatGPT, however, goes beyond delivering static content such as a search engine would; it engages users with dynamic conversation and adapts responses based on context. This interactive nature fosters a deeper understanding of topics and facilitates collaborative exploration, setting it apart from traditional search engines.

Moreover, ChatGPT is not confined to rigid templates or structures as conventional writing tools are. It adapts to specific needs, generating content tailored to a request. Additionally, it can understand complex queries and provide nuanced responses, which yields more power than that of standard writing tools. ChatGPT serves as an intelligent writing assistant that can offer suggestions and insights that enhance the quality and depth of this paper.

On a separate note, its generative style is unique. Below is a block quote excerpt from ChatGPT that shows the transformative impact it has in elevating the quality of this report:

In the process of developing our Senior Design paper, ChatGPT proved instrumental in generating insightful content and refining our ideas (OpenAI, 2023). Its interactive nature, expansive knowledge base, and efficient drafting capabilities streamlined our research and writing processes, elevating the overall quality of our project documentation. [69]

5.2 Similar Platforms

ChatGPT is a staple for generative AI available to the public. Since its takeoff, there has been a surge in users seeking AI solutions. Primarily, ChatGPT has gained widespread popularity. However, it is not the sole contender for AI solutions; several alternatives offer similar or even superior services.

HIX.AI [70] offers a comprehensive AI experience, combining accessibility with a diverse set of functionalities. Its chatbot, HIX Chat, covers a large database of topics while also demonstrating remarkable adaptability by processing uploaded PDF documents and summarizing complex web content. The integration of HIX.AI with a Chrome extension further streamlines tasks, automating processes like drafting and revising text in Google Docs. This seamless integration makes HIX.AI a versatile tool for various professional and personal applications.

Chatsonic [71] stands out for its powerful AI capabilities. Using Google Knowledge, a graphical information base, it provides factual responses and offers a range of tones to suit different conversational contexts. This versatility makes it an asset for businesses and individuals seeking context-specific interactions. Microsoft Bing, [72] powered by GPT-4, adopts a multifaceted approach, generating responses in different styles. Its integration of AI into online searches makes it a valuable resource for users seeking accurate and comprehensive information. Despite its slightly slower response times, its commitment to precision and creativity positions it as a contender in the realm of AI platforms.

YouChat [73] introduces multimodal functionality, allowing information exchange in forms including and outside of text. This approach enables YouChat to process textual, app, and link input and output, enhancing its overall adaptability. Claude, [74] available as both a chat and an API, demonstrates efficiency in looking up information and providing creative responses.

Google Bard, [75] although experimental, benefits from the technological prowess of Google. It showcases exceptional processing speed but holds challenges such as plagiarism and lack of resource citations. Auto-GPT, [76] requiring some coding knowledge, offers semi-autonomous functionality, allowing users to set goals and let the chatbot handle subsequent prompts and responses. Finally, Copy.ai [77] specializes in

article summarization and content creation, providing a customizable platform for users seeking specific writing tones and organizational features.

5.3 Platform Comparisons

5.3.1 HIX.AI

HIX.AI embodies a robust platform that offers numerous advantages to AutoCaddie. Its notable AI capabilities, encompassing a comprehensive suite of diverse AI solutions catering to various areas such as article generation, advertising, social media, and text rewriting and formatting, render it versatile and functional across a wide range of tasks. Additionally, its free tier provides access to many advanced features.

The following excerpt is taken directly from HIX.AI, which condenses the contents from G. Oppy and D. Dowe. [20]

The summary discusses various aspects and objections to Turing's claim that machines can possess intelligence. It addresses the slow progress in creating machines that can pass the Imitation Game and the criticism of the test for only recognizing intelligence in conversation-sustaining machines. The author also explores objections to Turing's claim, including theological concerns and fears about the consequences of thinking machines. The text mentions Lady Lovelace's objection to machines originating anything and the Lucas-Penrose constraint. The author concludes that further exploration is needed to determine the validity of these objections. Additionally, the summary touches on different arguments regarding machine capabilities, consciousness, and the behavior and decision-making differences between humans and machines. Lastly, it discusses the standing and interpretation of the Turing Test and its role in determining intelligence. [70]

Being able to summarize links by providing nothing but a URL saves critical while conducting research relevant to the project. Rather than spending hours browsing multiple research articles on the same topic, HIX.AI can save time by providing comprehensive analyses on each article, helping the selection process narrow down the options.

Nonetheless, HIX.AI does have certain drawbacks. The integration with Google Chrome, one of its main selling points, is irrelevant to this particular project. Moreover, while some features are freely accessible, others require a subscription, making it not entirely free software. A primary example is GPT-4: its usage is locked unless the full tier is purchased. Furthermore, it occasionally produces generic responses, a contrast to the creative content generated by ChatGPT in response to input queries. At this point the main research of the project has been conducted, so having access to a breakdown of article information is relevant only for development purposes.

5.3.2 Chatsonic

Chatsonic is a powerful AI tool with a wide range of capabilities. Its integration with Google Knowledge enhances accuracy through cross-referencing information, ensuring reliable responses. The ability to mimic different tones enhances its adaptability, improving creativity in language usage throughout a report. In the realm of content revision, where attention to nuance is crucial, Chatsonic stands out by empowering users to experiment with word choices, effectively refining paragraphs.

However, the features of Chatsonic are unlocked through a subscription-based model. In contrast to HIX.AI, which offers certain advanced features for free, Chatsonic relies heavily on subscriptions, which can pose financial challenges, particularly for projects with limited budgets like AutoCaddie. This makes it a difficult choice to select for usage when HIX.AI provides the same features for free, and ChatGPT provides more robust conversational output.

5.3.3 Microsoft Bing

Microsoft Bing helps enhance its search engine capabilities by utilizing seamless integration of AI within its core functionality. In response to its user activity, the platform is constantly evolving, providing additional information alongside search results. Additionally, it can generate three different styles of responses: Creative, Precision, and Balanced, which can help narrow searches based on user preferences.

However, despite these advancements, Microsoft Bing has its limitations. One drawback is the speed of generating responses, which may be slower compared to traditional browsing methods. Users experience delays in receiving generated responses, whereas a direct search on traditional engines may provide faster results. In the context of AutoCaddie, having the report process streamlined does coincide with efficiency, and other platforms can provide more efficiency in extracting information than Microsoft Bing. Specifically, HIX.AI can browse through websites faster, and ChatGPT can help foster ideas directly.

5.3.4 YouChat

YouChat's point of interest is its multimodal input/output generation. The capability of interpreting textual, app, and link data for free gives it a competitive edge against ChatGPT.

For example, YouChat summarized the article by G. Oppy and D. Dowe, which is the same example referenced prior by HIX.AI. The results are shown below:

The article titled "The Turing Test" explores the concept and implications of the Turing Test, a thought experiment proposed by Alan Turing in 1950. The test aims to determine whether a machine can exhibit intelligent behavior indistinguishable from that of a human. It discusses the role of natural language processing, machine learning, and the limitations of the test. The article also examines the ethical considerations and potential impact of creating machines that can pass the Turing Test. Overall, it

provides a comprehensive exploration of the Turing Test and its significance in the field of artificial intelligence. [73]

As a comparison to HIX.AI, it can provide an equally thorough summary. Despite being comparable in analytical ability to HIX.AI, it does not possess as many features that can improve developmental efficiency.

Although it excels in managing diverse data formats, its application integration is somewhat restricted. Users might encounter difficulties with seamless integration of specific applications, which can adversely affect their overall experience. Moreover, despite its impressive multimodal capabilities, YouChat lacks the ability to generate AI images. In the case of projects like AutoCaddie that heavily rely on visual data, the absence of this feature could impede the planning and creation of diagrams.

5.3.5 Claude

Claude promises a state-of-the-art AI model that provides users with accurate responses. It offers users meticulous and dependable responses, which can meet the demands of data analysis projects such as AutoCaddie. Moreover, it is accessible via API, which allows developers to integrate its capabilities into many different applications directly.

However, Claude's brilliance is partly overshadowed by its limitations. While a free plan is available, the functionality within this tier is restricted and often falls short in delivering significant results. This limitation raises concerns regarding the feasibility of relying solely on the free version of this platform for extensive data analysis requirements. Additionally, Claude's functionality deviates from the interactive and user-friendly conversational style of ChatGPT. Its unique approach may present challenges, particularly when revising existing work. The distinctiveness of Claude's style necessitates a learning curve that could potentially affect user experience and slow down efficiency.

Furthermore, although the API functionality shows promise, it does not align perfectly with the specific requirements of AutoCaddie. The GUI primarily focuses on displaying similarity and angular calculations rather than generating text, which makes the API integration pointless in this context.

5.3.6 Google Bard

Google Bard contains the backing of the tech giant Google, providing access to large amounts of resources and expertise, similar to ChatGPT. Additionally, it excels with fast speeds for output generation, even for complex input queries. This yields a competitive edge to ChatGPT, as overall efficiency may improve.

However, Google Bard's is currently in an experimental phase, which brings notice of inconsistent output generation, which ultimately raises concerns about data reliability. In AutoCaddie, where accuracy and consistency are critical, relying on an experimental tool with unpredictable outcomes may jeopardize the integrity of the analysis.

Additionally, Google Bard's inability to provide resources or verifiable data sources for its responses raises questions about the authenticity of the generated information. Without traceable origins, the data becomes less credible, limiting its usability in research-intensive projects.

5.3.7 Auto-GPT

Auto-GPT serves as a ChatGPT alternative implemented via a Python interface. This gives freedom to developers who can incorporate this platform directly into their projects. Moreover, Auto-GPT is semi-autonomous, or it allows the user to provide goals and follow-up information as well as incorporate chatbot functionality. Alongside that, it uses GPT-4 without the need of a purchased tier.

Conversely, the platform being implemented via Python is also a drawback. It poses a necessary learning curve, which negatively impacts efficiency for streamlined development. Additionally, it does not integrate well with AutoCaddie, as only the software design section of the project is dependent on Python, and the final product will not require any chatbot functionality.

5.3.8 Copy.ai

Copy.ai is different from its competitors in that it is not directly a fully-capable chatbot. Conversely, it is powerful with regards to condensing text when provided a website URL. It performs rapid article summarization, as well as advanced writing tools for revising existing paragraphs.

As an example, Copy.ai's summary when given the article by G. Oppy and D. Dowe, which was also analyzed by HIX.AI, is shown below:

The article discusses the concept and history of the Turing test, a proposal made by Alan Turing in 1950 to assess the question of whether machines can think. The article covers various aspects of Turing's proposal, including his description of the "Imitation Game," which involves a person, machine, and interrogator trying to identify which is the person and which is the machine. The article also explores the conceptual and empirical questions raised by Turing's predictions and the current state of research into artificial intelligence. Finally, the article briefly discusses the Chinese Room argument and the question of whether the Turing test is an appropriate goal for researching artificial intelligence. [77]

Seemingly, this summary coincides with the format of HIX.AI. On a negative note, Copy.ai provides fewer generative AI features overall. Thus, HIX.AI can complete the same generative AI tasks as Copy.ai with more overall results, which makes selecting this platform difficult.

Moreover, it requires a paid plan to access the writing tools. As free alternatives exist, there is no necessity to purchase a tier of a platform that has the same features as its rivals. It does not compete in total functionality with ChatGPT's conversational AI.

5.4 Platform Selection

For AutoCaddie's development process, ChatGPT emerges as the strongest choice. It highlights a blend of strong qualities but excels with interactivity with the user. In the context of this project, where accuracy, creativity, and user engagement are necessary, ChatGPT is the strongest choice.

In the realm of accurate data comprehension, ChatGPT's prowess is unmatched. Its ability to decipher intricate queries is a skill that many of these platforms hold, but being able to support the output with examples and references is crucial for data validity.

Creativity is another domain where ChatGPT is powerful. As mentioned previously, ChatGPT has the power to generate beyond standard responses and provide narratives and explanations. This can help expand the project developers' perspective on relevant problems within this project.

There are other powerful platforms. HIX.AI stands out as one of the most powerful online tools against ChatGPT, yet suffers from its own limitations. For one, having the requirement of purchasing tiers, along with usage credits, limits its availability. Many of the other models fall to this dilemma, including YouChat, Chatsonic, Claude, and Copy.ai. Not to mention, HIX.AI does not foster the same creativity as ChatGPT as it is not as generative with its standard output.

Overall, ChatGPT is the most effective choice as an AI platform to streamline the process of documenting AutoCaddie.

Chapter 6: Project Hardware and MCU Design Details

6.1 PCB Subsystem Overview

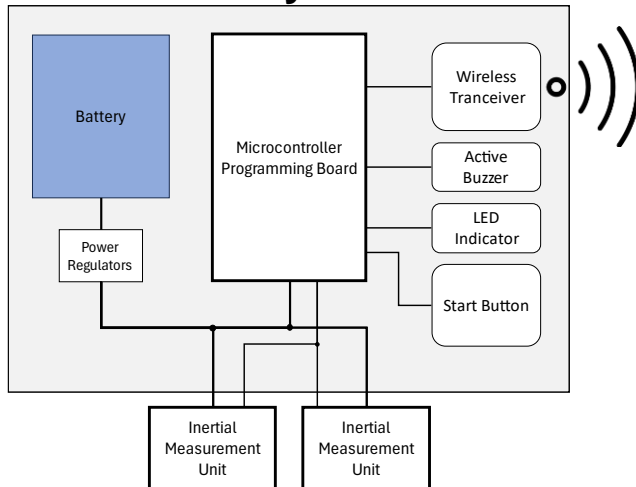


Figure 19: Block diagram showcasing the physical implementation of the wireless transmission subsystem.

The above diagram outlines the overview of the wireless transmission and IMU circuit. The data that is collected and is to be transmitted to the central computer system originates from the IMU devices. The IMUs collect absolute orientation quaternion vectors and acceleration vectors from kinematics of the user's driving arm. This data is collected by a microcontroller unit (MCU), which then packages the data and passes the data packet onto a wireless transceiver module. Finally, the data packet is transmitted to the central computer from the wireless transceiver module. A connected battery powers all components of the system, including the IMUs, a button that serves as an action trigger for the MCU to begin collecting IMU data for a period of time. An LED and a buzzer module serves to notify when the MCU is ready to use and when certain phases of the main system loop are active. As can be seen, the central MCU has its own dedicated board. This design choice will be discussed in the next section.

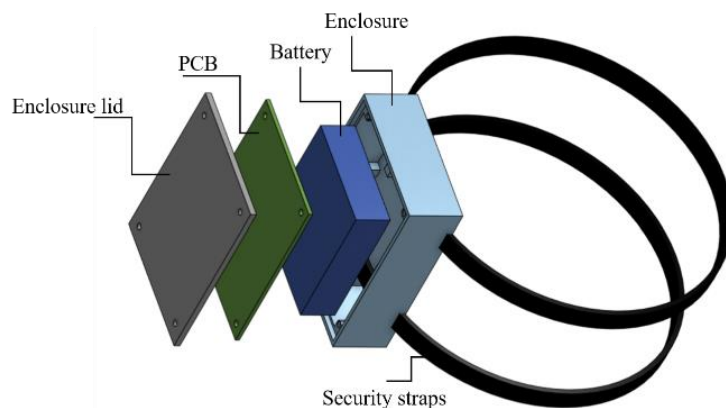


Figure 20: Explosion diagram of PCB and battery within enclosure, including security straps.

As shown in the above explosion diagram, the PCB and battery will be enclosed within a containing box. This box will ensure that the PCB is nonmobile with respect to the user. The using secure straps, the enclosure box can be mounted to the user's body. The battery is placed to the bottom of the enclosure box, such that it is placed closed to the user. This will minimize the moment of inertia for the weight of the enclosure should it shift when placed on the user. The enclosure box will be placed on the user's waist as to avoid getting in the way of the user's swing.

6.2 PCB Schematic Design and Methodology

The MCU has been broken out into its own programming board, to which this board plugs into the main “backplane” PCB. This was done in order to make troubleshooting easier, and to allow for reusability in the logical components of the hardware. The “backplane” PCB contains the connectors for the IMUs, as well as the LED, buzzer, and button peripherals.

6.2.1 PCB Schematics Overview

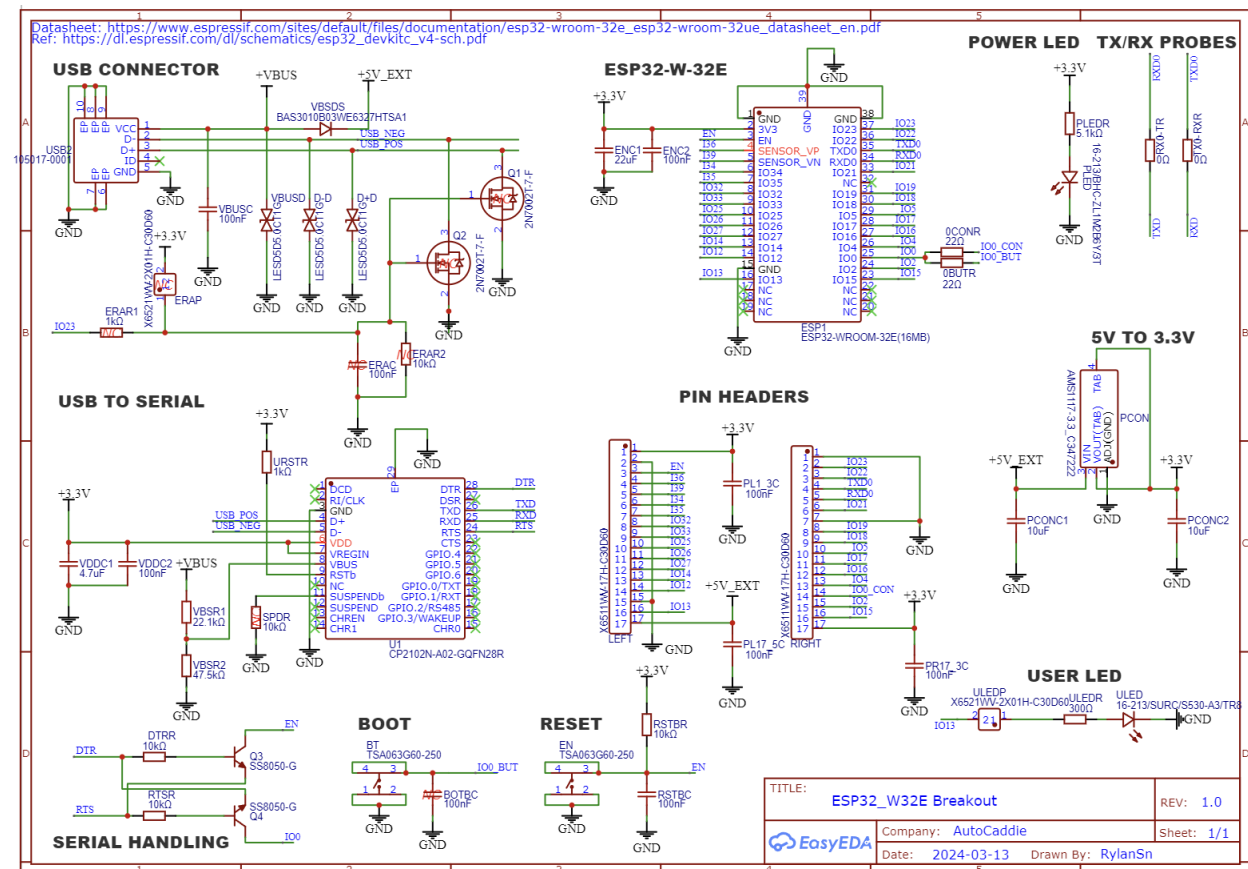


Figure 21: General Circuit Schematic of MCU programming Board

6.2.3 USB programming

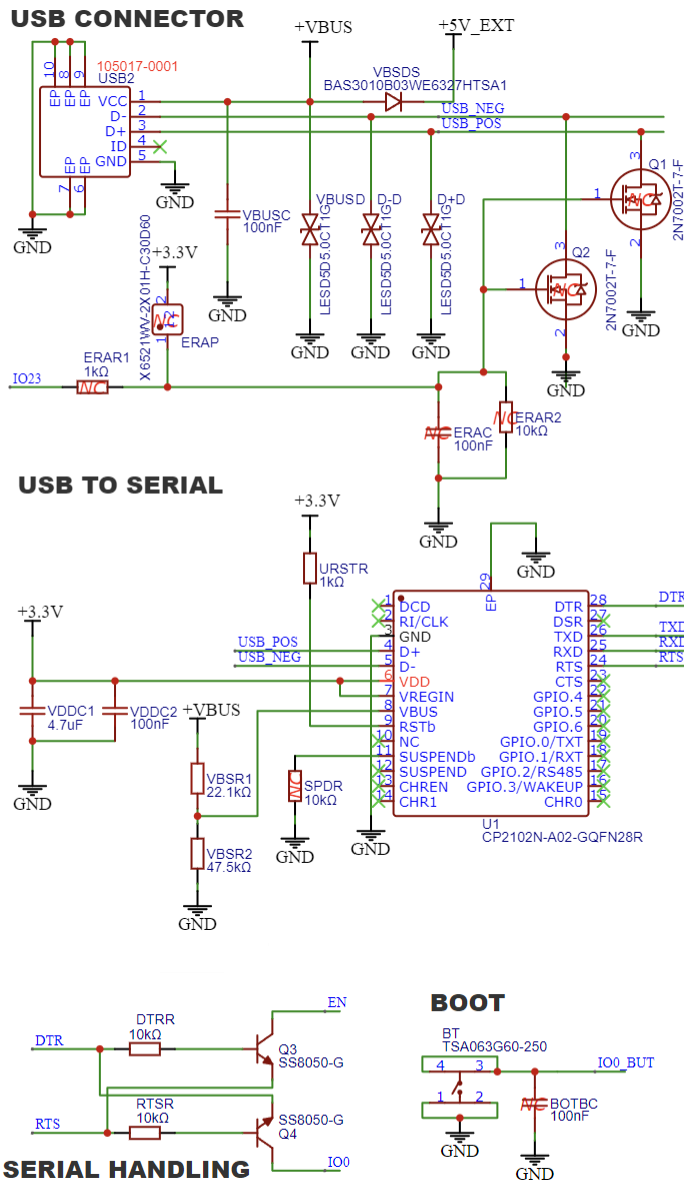
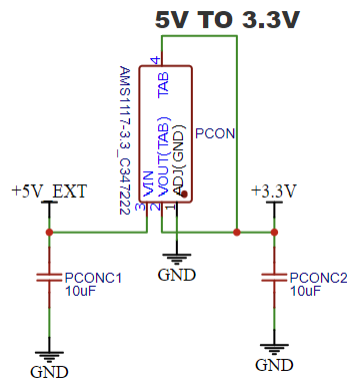


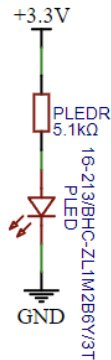
Figure 24: Circuit diagrams of program control functionality

The shown diagrams show the necessary hardware to program the ESP32 module. USB to serial programming was selected due to its faster interfacing for programming and testing.

6.2.4 Power Control



POWER LED



RESET

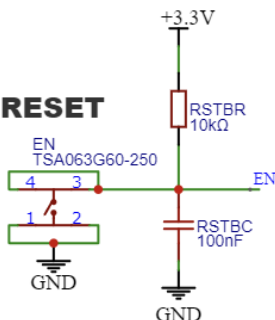
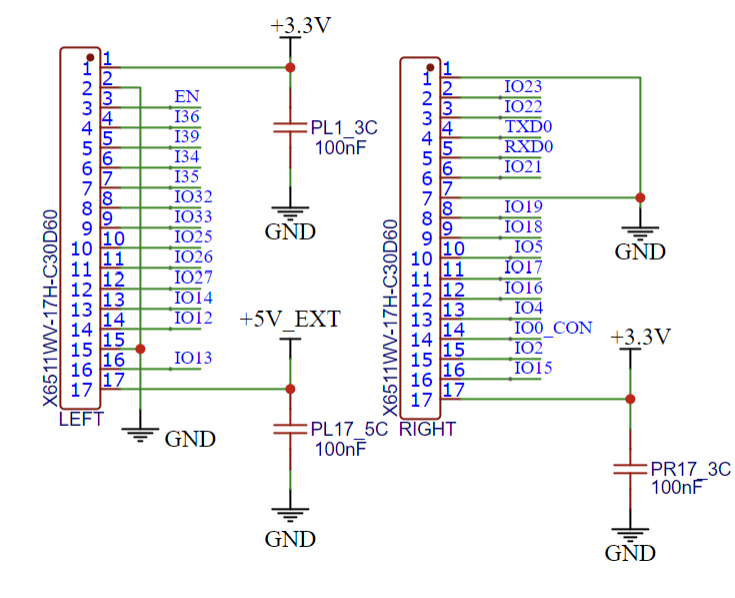


Figure 25: Circuit diagram of power regulation, power monitoring, and ESP32 power control

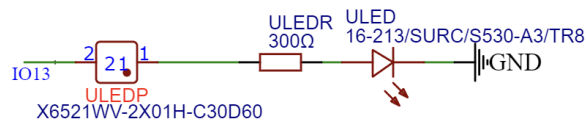
The as the ESP32 MCU will be powered from a 5v source, and will be programmed with 5v USB lines, the appropriate power conversion and power handling circuits must be implemented. As per recommended designs, the AMS regulator will serve as a 5v to 3.3v step-down regulator. The power LED indicates if the circuit is receiving sufficient power supply, and the reset button controls the EN pin to the ESP32. When pulled down, the ESP32 will not take power, and thus will reset upon receiving power once more. This button effectively serves as a restart button.

6.2.5 Connection Headers and Other Hardware

PIN HEADERS



USER LED



TX/RX PROBES

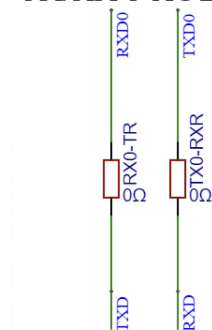


Figure 26: Circuit diagram of power regulation, power monitoring, and ESP32 power control

As the programming board has been separated from the general peripherals required for this specific iteration of the hardware system due to an aim of modularity, pin headers will be used to interface with the backplane board. Further, for the purposes of troubleshooting and development support, resistor probes and a user LED has been placed on the board

6.2.6 Main Backplane circuit

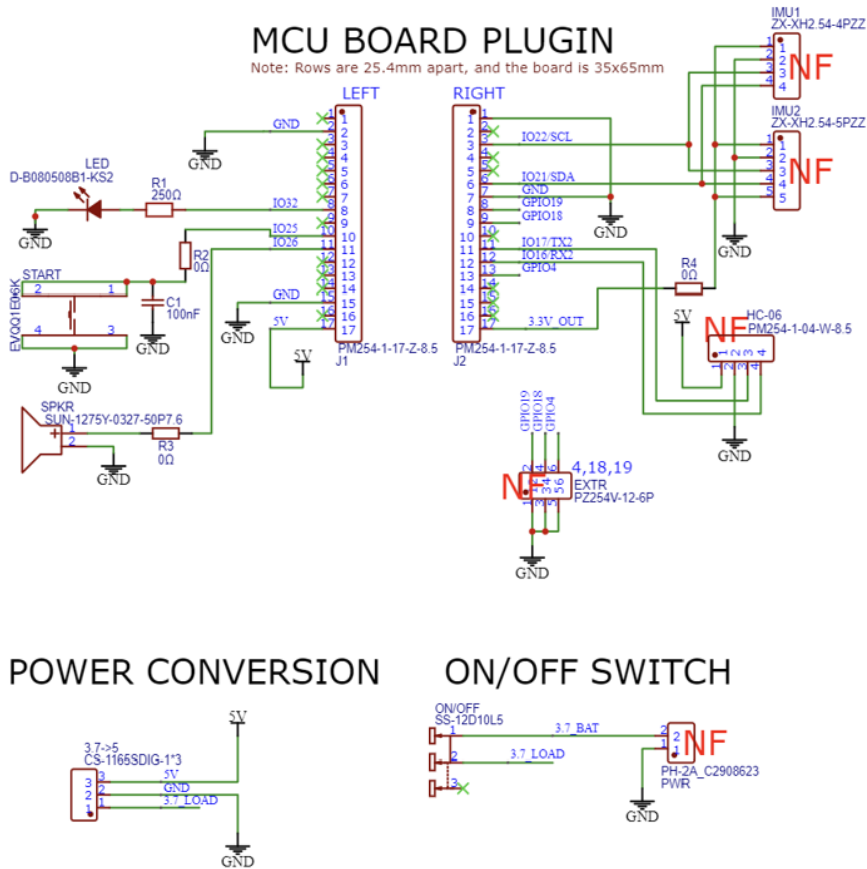


Figure 27: Circuit diagram of power regulation, power monitoring, and ESP32 power control

The main backboard for the hardware peripherals houses the key components which are involved in the functionality of the hardware system loop. This includes a start button, LED, buzzer, IMU connectors, and the HC-06 connection terminal. Pin headers are used to also break out a voltage converter for battery power regulation, and an on/off switch is used to allow the user to turn the hardware system on and off as they need to. As discussed before, Future iterations or configurations of the AutoCaddie system may need different peripherals, so the modularity of the backboard and programming board interface allows for easily swapping of functional boards.

6.2.7 Voltage Converter Breakout

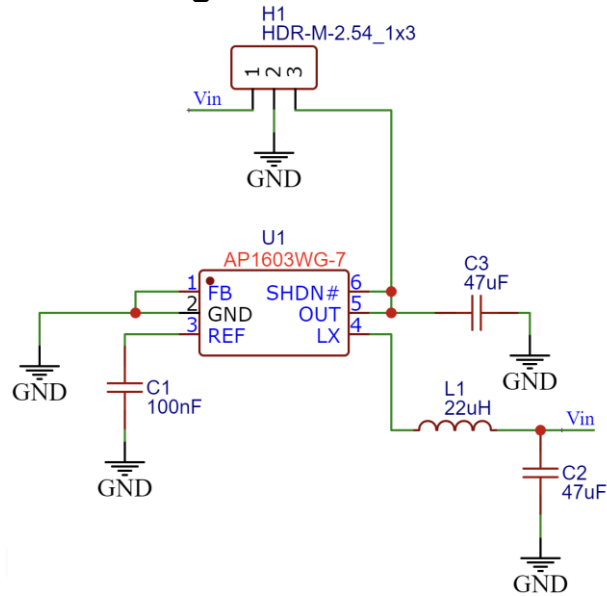


Figure 28: Circuit schematic of the boost regulator for battery power regulation

The power regulation circuit that has been designed for the Li-ion to MCU board interface has been designed to take the 3.7v power supply from the battery, and boost it to 5v. This 5v power source will serve to power both the MCU board, as well as the HC-06 wireless transceiver module. The wireless transceiver module will use this higher power to sustain a stable and strong connection to the central computer, whereas the MCU board will take that power and manage it as it requires.

6.3 Peripheral Hardware Integration

6.3.1 IMU System Design and Integration: Arm Linearity Example

The following section is relevant to a potential use of the hardware sensor system of AutoCaddie. In this example, IMUs are placed on the arm and are used to provide supplementary linearity data to compare to calculations performed by the artificial intelligence. Relevant to other discussion within this paper, this section introduces the specifics of quaternion usage. This example is not developed for the functional prototype.

The IMU sensor system data serves as a critical reference of real, direct-environment data that serves as an anchor for data analysis of the full AutoCaddie system. The primary reference frame that the IMU sensor system data provides is joint angle data for the user's driving arm. As arm straightness is a common point of feedback for golfers during swinging, the IMU sensors can provide direct data that models the linearity of the user's arm. This data can be used as its own condition for feedback, as well as used to further supplement the AI's models. The main types of data that are desired from the IMU sensors are absolute orientation quaternion vectors, as well as linear acceleration vectors.

The absolute orientation quaternion vector data serves as the primary key in the calculation of joint angles of the user's arm. Due to limits of sensor technology, relational position data cannot be directly detected or determined via instrument measurements without extensive data integration and estimation. As such, orientation data will be used to estimate rotational angles of a soft-joint model. At a high level, the quaternion rotation similarities are compared using quaternion dot product or inner product manipulations (as listed below). These rotational similarities are used to determine the overall projection difference of the vectors, thus allowing a calculation of a relative difference in angle. Quaternion vectors are preferred over Euler orientation vectors, as Euler orientation vectors suffer from singularity errors. Quaternions do not suffer from these singularity errors but come at the cost of more complex calculation requirements. The IMUs and quaternion calculations are not perfectly stable, however, so calibrations will be required between each use session of the AutoCaddie system.

$$\theta = 2 * \arctan 2(|q1 - q2|, |q1 + q2|)$$

Equation 9: Quaternion angle difference using arctan2.

$$\theta = 2 * \arccos(|q1 \cdot q2|)$$

Equation 10: Quaternion angle difference using arccos.

$$\theta = \cos^{-1}(2\langle q1, q2 \rangle^2 - 1)$$

Equation 11: Quaternion angle difference using inverse cosine.

Equations 9, 10, and 11 describe several options for estimating joint angles. Equations 9 and 10, though simpler in terms of calculations requirements, do suffer from the “double cover issue”, where \mathbf{q} and $-\mathbf{q}$ describe the same quaternion rotation. Generally, though, this should not pose a major issue in-use, as the human elbow joint has an average maximum flexion range from 0° to 180°, thus crossover into mirror vectors should not be generally possible. Equation 11 is generally understood to be a more stable and complete description of quaternion angle separations, describing the required rotation to achieve one quaternion rotation to get another. Though it is more computationally complex, Equation 11 will be chosen for preliminary testing and integrations.

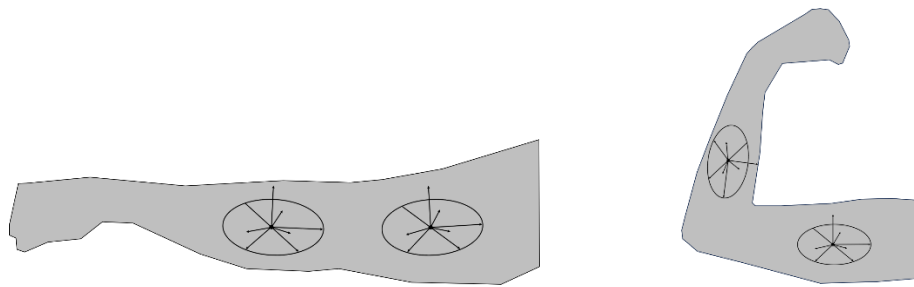


Figure 29: Demonstration of quaternion placement on user arm, in “straight” (left) and “bent” (right) arm states.

Of course, due to issues with IMU sensors systems, such as drift, calibration is required. As the quaternion angle difference calculations are not perfect representations of limb transpositions, errors, including those induced by drift, the required calibrations should be performed between each use of the AutoCaddie system. In order to reduce unwanted

variation in calibration results, a standardized procedure should be used to perform the calibration. For example, the user may initiate calibration and hold out their driving arm, which the IMU system is mounted on, at a 45° angle and keep the arm as straight as possible for a given period of time. With this pose during the calibration sequence, the AutoCaddie system can take calibration measurements to set an arm linearity threshold value, as well as “zero” in the reference quaternions, so that the necessary calculations are as accurate as possible with respect to the true position and rotations of the user’s arm.

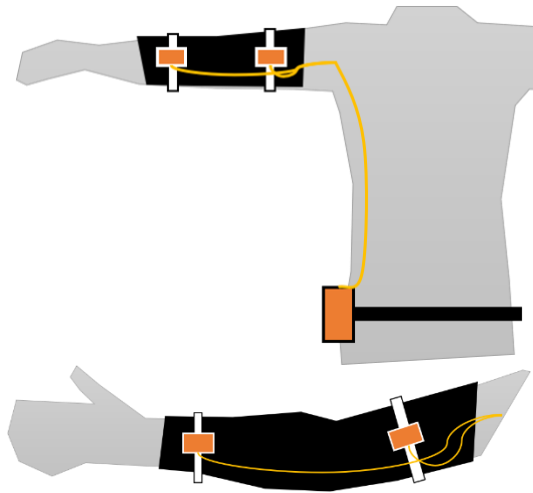


Figure 30: Demonstration of IMU system mounting on user.

As the IMU sensor data is relevant to the user’s driving arm, the IMUs must be securely placed on the user’s arm in a manner such that the IMUs maintain their relative position on the arm, while maintaining user mobility. For this purpose, the IMUs will be secured to an elastic sleeve worn by the user. This will allow for the system to maintain a relatively secure position, while not overly impeding user movement. The given wires of the IMUs will be fed through a flexible sheath, which will contain enough slack as to not catch or pull on the user during swing action. The IMU wires will feed into the PCB enclosure box, from where the data is wirelessly transmitted to the computer for calculation. The PCB enclosure is secured on the user’s waist for unobtrusive placement, as extra weight on the user’s arm or shoulders will impede user’s swings due to the extra inertial weight.

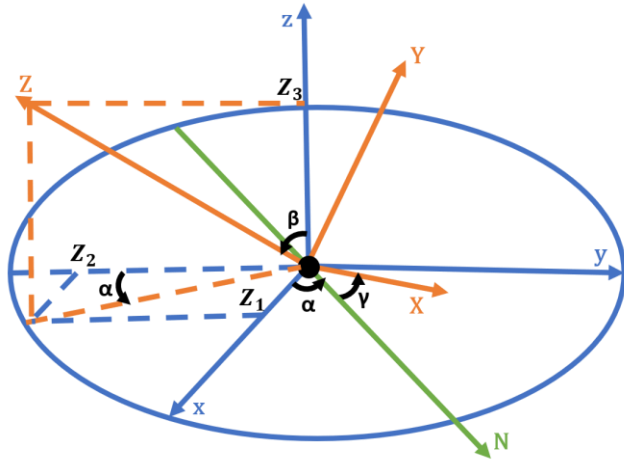


Figure 31: Diagram of Quaternion vector data

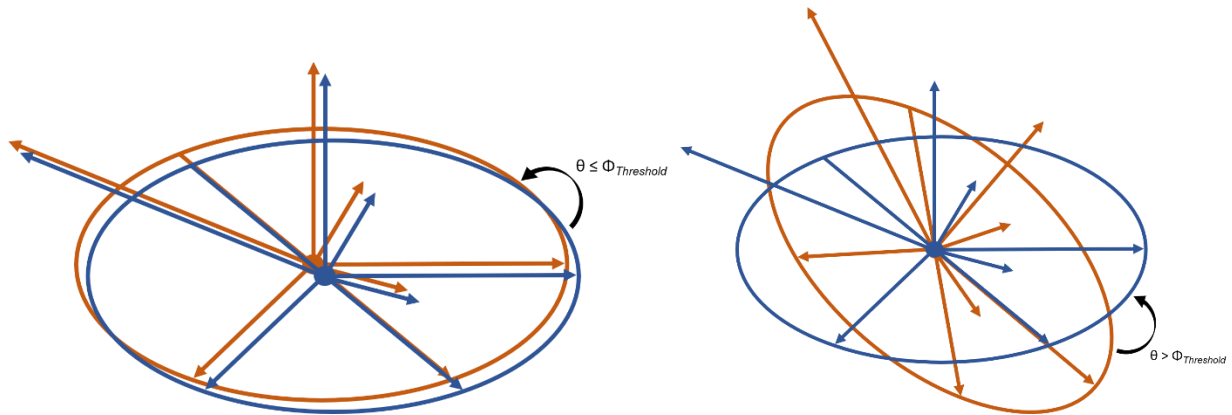


Figure 32: Demonstration of quaternion rotation difference calculation. Low dissimilarity of rotation (left) suggests linearity, whereas high dissimilarity(right) suggests nonlinearity

The IMU systems will collect data throughout the swing action following a signal to prompt the user to begin their swing. Following the swing, the collected data will be formatted into a package, and the package will be wirelessly transmitted to the central computer for processing. The calculation processing will be done on the central computer and not on the PCB MCU to save processing resources on the MCU system, as well as allowing the AI to interface with the pre-processed data. From here, feedback from the quaternion calculations can be generated, and the AI can use the data and calculation results as necessary. As discussed earlier, equations such as Equation 11 can be used to determine arm linearity from the measured quaternion vectors. The given angle calculation of the quaternion rotation difference can be compared to a given threshold value determined during calibration. If the calculated angle is less than the threshold, the arm is deemed straight, and if the calculated angle is greater than the threshold, the arm is deemed bent, and feedback to straighten should be given to the user.

6.3.2 IMU System Integration Option: Back Rotation Example

As with what will be discussed in chapter 6.6, modularity of the AutoCaddie system is an important focus on the design of the hardware subsystems. Especially with the use of IMU sensors, the user of the AutoCaddie system can chose to use one of various IMU

placement setups for the training of specific areas of their golf swinging abilities. As discussed previously, the IMU placement on the user's arm can be used to provide feedback and supplemental data of the user's arm linearity for the artificial intelligence model. In this IMU setup serves as supplemental source of data which is used to verify the accuracy of the artificial intelligence's feedback.

Another primary setup that the placement of the IMUs which can be used is a placement that focuses on the IMUs measuring the rotation of the user's back – specifically the shoulders and waist, and even the straightness of the user's back. For example, the shoulders should ideally turn 90 degrees from the target line and the waist should drive the rotation and pitch of the user's body during the follow-through of the swing. The first IMU can be placed in the center of the user's upper back between their shoulders, while the second IMU is placed in the center of the lower back, in line with the user's waistline. Both IMUs can be secured using clothing pins and Velcro pads. In terms of potential interference with the kinematics of the user, this setup provides less risk of obstruction than the arm linearity model. This IMU setup provides core data in this field of feedback, as the artificial intelligence model struggles to make these calculation provided video footage of the user.

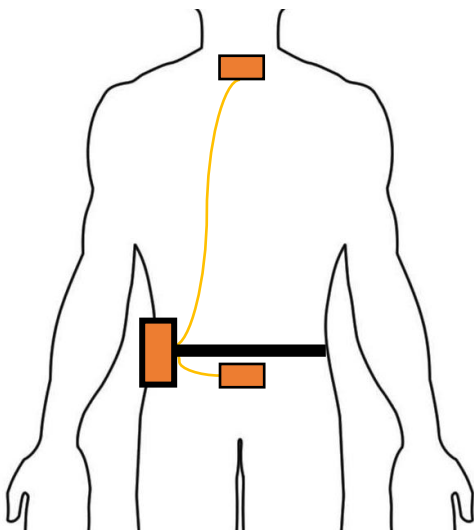


Figure 33: Demonstration of IMU setup measuring the shoulder and waistline rotation of the user.

6.3.3 IMU System Integration Choice: Shoulder Alignment to Ground

As shown in figure 33, a third form of IMU integration for feedback is a placement on the user's shoulder blades. Though simple, this setup can be used to glean valuable information in how a user holds their posture before their swing. According to reports from direct contacts, the stance and preparation one has in the swing is just as important as the swing itself. Given this information, AutoCaddie can provide feedback regarding this matter. The exact point that will be analyzed is the user's shoulder plane, and if it is parallel, or "square" with the ground. This can be performed by measuring the quaternion values of both IMUs, and determining if the related axis of rotation is within a given

threshold value. If so, the user's posture is correct. Otherwise, if the quaternion values of the respective axis go beyond a specific threshold, the AutoCaddie system can deem the user to need to improve on their posture before their swing.

This choice of IMU usage will be the central configuration utilized within the AutoCaddie project due to multiple factors. These factors include both ease of integration, and the extension of separate areas of feedback, which can greatly improve the fields in which AutoCaddie can provide feedback. The AutoCaddie system will also be programmed to collect data during the swing as well, to further supplement the AI model for during-swing analysis.

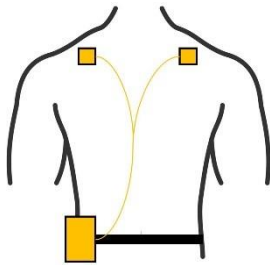


Figure 34: Demonstration of IMU setup measuring the shoulders' rotation of the user.

6.3.4 Camera System and Placement

Overall, the camera setup for the AutoCaddie system is fairly simple. The cameras used in the system will interface with the central computer via USB connections, and due to the portable nature of the selected cameras, they can be placed and have their positions adjusted as necessary. The positions and orientations of the cameras must be placed in such a way that they fully capture the entire figure of the user. To better assist in the placement and orientation of the cameras, tripods can be used as necessary. Furthermore, if the reach of the USB cables is not sufficient, length extension adapters can be utilized.

The first camera should be positioned to face the front of the user while they are postured for a swing. From this angle, the camera can capture visual data of the user such the stance posturing of the user, as well as their general swing path and hip rotation. The second camera should be positioned to face the side of the user while they are postured for a swing. From this additional angle, the camera can capture more visual data of the user, such as the overall back posture of the user, as well as the swing plane of the golf club. Together, these camera angles provide a complete set of possible data that can be analyzed for the central computer's artificial intelligence.

6.4 PCB and Peripheral Control

6.4.1 MCU Control

The MCU is the central unit of the PCB and IMU subsystems. The microcontroller manages the I2C communications with the IMUs and oversees the requests to and collection of data from the IMUs. Furthermore, the MCU manages the data collection

phase initiation through an action trigger, such as a dedicated button, and interfaces with a wireless communication module to wirelessly transmit the data collected during the collection phase. The data transmission protocols will be discussed in **Section 6.5**.

As the IMUs use I2C communication protocols, the MCU must be programmed with this communication protocol in mind. During the “zeroing” calibration of the IMU systems, the status of the magnetometers, accelerometers, and gyroscopes can be determined, and appropriate calibrations or adjustments can be made. Once the IMU system is ready for data collection, the MCU can access the I2C IMU devices during the data collection phase. Once the data collection phase is initiated, the MCU will repeatedly request and collect data from the IMUs at a frequency the IMUs are capable of producing ($\leq 100\text{Hz}$). While the MCU is collecting the IMU data, the MCU consistently provide the information to the wireless transceiver module, so that the data will be transmitted to the central computer. The collected data will be formatted in a CSV (comma-separated value) format, where each element from the quaternion and acceleration vectors is a unique entry in the CSV package, and each instance of data is separated by a newline character. This simple data formatting method will reduce data packaging overhead and will be simple to parse. Furthermore, a header with the expected number of data entries will be provided, in order for the central computer to know if the data is a “start” or “data” signal, and know how much data is to be expected. The basic CSV format pattern for the data package is as follows:

```
HEADER X
q_x1,q_y1,q_z1,q_w1,q_x2,q_y2,q_z2,q_w2,a_x,a_y,a_z
q_x1,q_y1,q_z1,q_w1,q_x2,q_y2,q_z2,q_w2,a_x,a_y,a_z
q_x1,q_y1,q_z1,q_w1,q_x2,q_y2,q_z2,q_w2,a_x,a_y,a_z
... //Etc
q_x1,q_y1,q_z1,q_w1,q_x2,q_y2,q_z2,q_w2,a_x,a_y,a_z
```

For the preservation of the hardware system’s power supply, when the hardware devices are not in an active transmission loop, the system will enter a light sleep mode. This mode will disable all unnecessary systems, while pausing the CPU and preserving memory. The device can be woken up to begin its recording and transmission cycle when the user presses the “start” button.

6.4.2 IMU Component control

The MPU6050 devices are controlled via the ESP32’s I2C ports. This connection method allows for simple and effective data transmission, and is widely supported and provided for by many libraries. Since both IMUs have the same base address, one of the IMU’s AD0 pin must be pulled high to change its relevant address.

The currently utilized MPU6050 library, designed by Jeff Rowberg, uses digital motion processing (DMP) algorithms to generate more advanced data such as quaternion data from the general accelerometer and gyroscope data. For correct measurements however, the DMP functionality and IMU calibration must be performed when the IMU system is powered on. For correct calibration, the IMUs must be kept flat relative to the ground

during the calibration phase. Once calibration is complete, the system's LED will turn on to indicate that the system is ready to begin. Should calibration or DMP initialization fail, the LED will flash to indicate that the system must be restarted.

6.5 Modularity of Hardware Peripheral Sensors

A key factor of note regarding the hardware system sensor integration with the larger AutoCaddie system is its modularity. Current designs of the AutoCaddie system utilize IMUs as a basis for hardware-sourced data. However, due to the use of standardized connectors and modifiable control, the sensor control system of the AutoCaddie hardware can be modified to utilize different sensor hardware, such as foot pressure sensors. Should different sensors be used in place or in conjunction with other sensors, the main AutoCaddie software that reports feedback to the user can be adjusted for such changes. From such potential of modularity for the AutoCaddie design, further iterations of the AutoCaddie system can be customized to focus on specific areas of improvement that the user chooses, further boosting the AutoCaddie's application domain for users.

6.6 Summary of Hardware and MCU Design

Overall, the hardware system of the AutoCaddie project serves as a functional and modular platform for data collection and further iterative development. The PCB subsystem's main responsibility in the current AutoCaddie system model manages the wireless transmission of collected IMU data, as well as the source for user prompting and stage control. The PCB and battery are mounted on the user's body via a printed enclosure, which can be secured to the user without large impedance via any type of strap.. The MCU is segregated into its programming board, aiding troubleshooting and reusability, with the main backboard housing IMU connectors and peripherals. The ESP32 module's standard connections and USB programming facilitate faster interfacing, while power regulation circuits manage voltage conversion and monitoring, with a boost regulator for battery power. IMU integration options include arm linearity, back rotation, and shoulder alignment, each offering specific feedback for swing improvement. Camera placement captures user posture and swing path for AI analysis. MCU control oversees IMU data collection and transmission, with the system modular for potential integration of different sensors, expanding AutoCaddie's application domain and customization possibilities. Overall, the design emphasizes functionality, modularity, and user feedback for enhancing golf swing performance.

Chapter 7: Project Software Design Details

7.1 General Overview

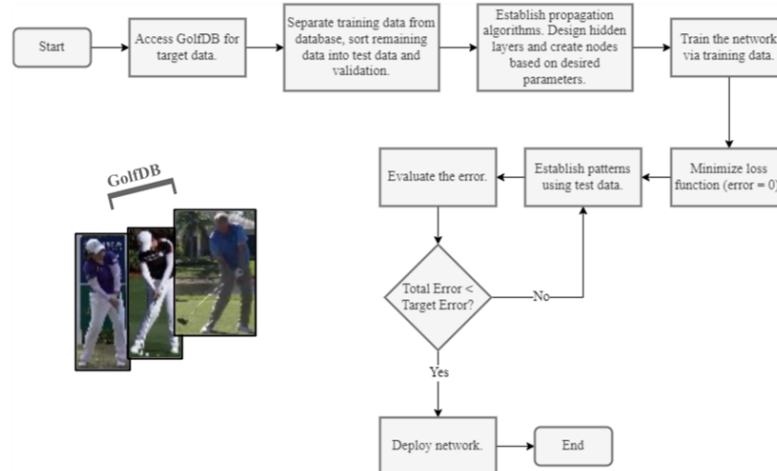


Figure 35: Block diagram outlining the machine learning training model. Accessed via [1].

This diagram represents a high-level breakdown of the training process for the neural network. First, videos from GolfDB will be referenced, with a large quantity (not all) of the data separated and designated as “training data.” Next, internal mapping networks will be designed to reflect the hyperparameters being studied by the user. These make up a series of “internal layers” for the network and will utilize backward propagation to map actions within the swing data to their respective parameters. Then, through iteration, the system will reference training data, and try to predict the correct feedback with a calculated error rate. If the error is small enough, it can be concluded that the system is successful with analyzing golf swings, and the network can be deployed.

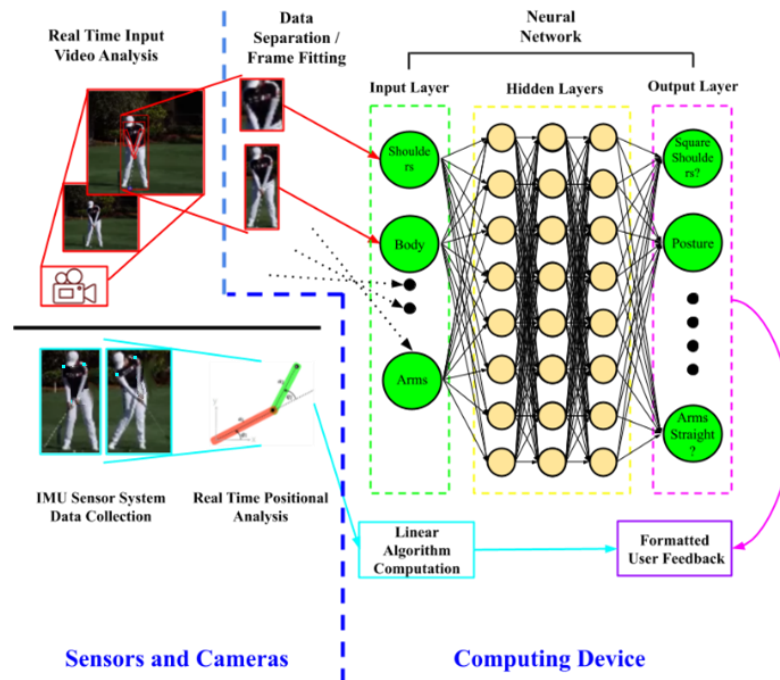


Figure 36: Diagram depicting a high-level overview of the Artificial Intelligence subsystem using a machine learning model. Accessed via [61].

The associated diagram represents a low-level breakdown of the data process. There are two separate systems operating under the same supervision: the video system and physical sensor system.

The video system records the user's swing across a short duration and uploads it via a standardized package to the computing device. From there, the video is preprocessed, including separating actions into frame sequences and matching the dimensions to the training data. Then the data is fed into the trained neural network, which compares the input layer of data to hyperparameters via advanced feature matrix mapping in the hidden layers, then produces a predicted swing referencing the user's initial velocity and direction of their motion. Finally, the results are fed as formatted user feedback to be displayed.

Similarly, the sensor system starts acquiring data from two IMUs which record quaternion data. From there, we can send the data, packaged with the video feed, to the computing device, which observes the linearity of the two IMU while prepping before the swing. This data is also fed as formatted user feedback to be displayed.

7.2 Machine Learning Model

At the core of AutoCaddie's innovative approach lies a sophisticated machine learning model that seamlessly integrates the powerful capabilities of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This model is meticulously crafted to analyze and interpret the intricate nuances of golf swings, providing a cornerstone for the project's cutting-edge sports training solution.

The CNN component of AutoCaddie's model is strategically employed to extract crucial angle information from the user's video data. Leveraging Google's state-of-the-art MediaPipe [87] framework, the CNN excels at processing visual inputs with exceptional accuracy and efficiency. This aspect is of paramount importance for a mobile-based application like AutoCaddie, where real-time performance and computational constraints are paramount considerations.

Once the angle data is obtained, it is seamlessly fed into the LSTM network, which serves as the temporal analysis component of the model. The LSTM architecture is specifically designed to capture and interpret the nuanced sequential patterns inherent in golf swings. This capability is crucial for accurately analyzing the intricate coordination and progression of a golfer's swing throughout its various phases, including the backswing, downswing, and follow-through.

The integration of these two neural network architectures enables AutoCaddie to perform a comprehensive analysis of golf swings, incorporating both spatial and temporal dimensions. Initially, the CNN layers meticulously extract crucial angle information from the video frames, capturing the spatial relationships between different body parts. Subsequently, these extracted angles are fed into the LSTM layers, which leverage their unique ability to process sequential data to identify and interpret the temporal patterns across the entire swing sequence.

The strength of AutoCaddie's machine learning model lies in its ability to generate a recommended swing trajectory based on the user's initial movements. By analyzing the first 30 frames of the user's swing, which capture the crucial initial momentum and body positioning, the LSTM network can predict and generate the optimal swing trajectory for the remaining frames. This predictive capability empowers AutoCaddie to provide golfers with personalized, real-time feedback, highlighting areas for improvement and offering insights to refine their technique based on their unique swing dynamics.

The model is meticulously tailored to align with AutoCaddie's software architecture, ensuring seamless interaction with its various subsystems. The video frames undergo preprocessing to enhance relevant spatial features, and the system leverages pre-trained weights on large datasets to enhance the model's proficiency in domain-specific tasks. The optimization process involves utilizing advanced techniques, such as the Adam optimizer, with an initial learning rate set at an appropriate value. Fine-tuning various hyperparameters, such as input size and sequence length, is an integral part of cultivating the most effective and efficient model configuration.

In the context of AutoCaddie, this machine learning model transcends its role as a mere technical component; it is the driving force behind a revolutionary coaching system. By leveraging advanced CNN and LSTM architectures, AutoCaddie can provide immediate, tailored feedback to golfers, identifying key improvement areas and offering insights to

refine their technique. This AI-driven approach, combined with a user-centric design, promises to significantly enhance the training process and overall experience for golfers, empowering them to unlock their full potential on the course.

7.3 Software Organization

The software implementation in AutoCaddie will be encompassed entirely within a laptop, or other computing device. It essentially acts as the main processor and feedback controller of the system. For each of the data acquisition subsystems, the software needs to be able to receive the results, apply preprocessing algorithms, feed it through either linear algorithms or the machine learning model, then display it back to the user. Additionally, the software needs to be packaged as a finalized product, including proper user design, additional features, and extensive testing. Each of these topics will be elaborated upon in this section.

7.3.1 Software Design Overview

The software design of the AutoCaddie project encompasses a comprehensive approach to developing an intelligent system that provides real-time coaching feedback for golfers. This overview outlines the key components, methodologies, and considerations involved in the software design process, aiming to create a seamless and user-friendly experience for athletes seeking to improve their golf swings.

The primary objective of the software system is to create a sophisticated, yet intuitive, AI-driven coaching system capable of analyzing golf swings in real time. The software aims to provide personalized feedback to users, helping them identify areas for improvement and refine their techniques. To achieve this, the software integrates data from two separate systems, processes it through machine learning and linear calculations, and presents feedback to users through a GUI.

The first section of the software design is the integration of data streams from IMUs and video cameras. These data streams provide real-time information about the user's swing, including initial posture, arm straightness, hip rotation, and shoulder straightness. These are features defined in a previous section. The software processes this data concurrently, utilizing algorithms to extract relevant features and patterns.

At this point, the machine learning component is employed, which plays a crucial role in this project. These machine learning algorithms are used to compare the user's swing data to reference videos from a comprehensive database (GolfDB). Specifically, these comparisons identify discrepancies and deviations, allowing the system to provide precise and actionable feedback to the user.

The software design coordinates closely with the hardware components, which creates the effect of seamless integration and communication. The software receives data from the IMUs to obtain precise sensor data, and uninterruptedly streams data from two video cameras to capture video information.

After the algorithms generate similarity data, it is formatted and sent to a GUI display for the user to interpret. These will be targeted directly at the features in question given by the machine learning model features it has been trained on. The design of the system allows for large replay value, where an infinite number of iterations can be performed as each output gives feedback for that specific instance of a swing.

The system undergoes rigorous testing, including unit testing, integration testing, user-experience testing, and accessibility testing. Unit testing validates individual software modules, ensuring their functionality. Integration testing assesses the seamless interaction between software and hardware components. User experience testing evaluates the effectiveness of feedback mechanisms and the comfortability of the layout. Many of the processes have been outlined in the standards of validation and user-experience, where online tools can assist in the refinement of the user program.

Overall, the software design of AutoCaddie is a meticulous process that integrates cutting-edge technology, machine learning algorithms, and user-centered design principles. By seamlessly combining data integration, machine learning, intuitive user interfaces, and efficient hardware communication, the software aims to revolutionize golf training. Through continuous refinement and testing, the software design ensures that AutoCaddie provides golfers with personalized, real-time feedback, enhancing their skills and overall golfing experience.

7.3.2 Low-Level Software Flow

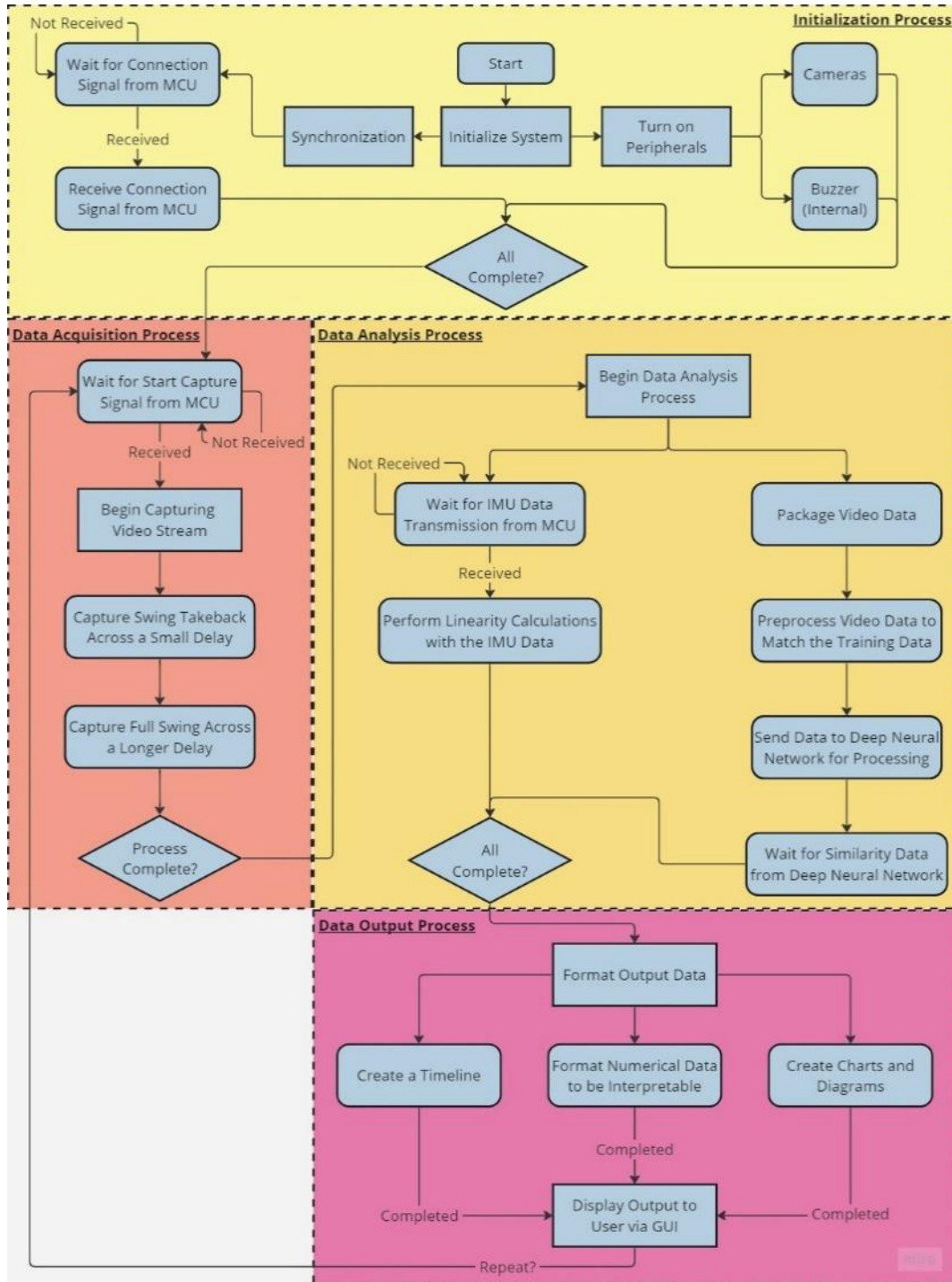


Figure 37: General software flow diagram.

7.3.3 Development Components

In this section, we explore the key components essential for building and deploying AutoCaddie as a robust local software solution. From the choice of programming languages and frameworks optimized for AI and machine learning tasks to the selection of powerful Integrated Development Environments (IDEs) that facilitate efficient coding

and debugging, every aspect is tailored to ensure the system's seamless operation within a local environment. Version control systems are employed to track changes and maintain code integrity, while rigorous testing and debugging tools guarantee the system's reliability and accuracy. Additionally, robust documentation practices are essential for comprehensive insights into the system's architecture and functionality.

7.3.3.a Machine Learning Model Programming Language

For the specific software domain of AutoCaddie, utilizing specific programming languages is necessary for performance. However, it is wise to reference the open-source community and understand commonly used languages for the tasks being tackled by this project. Specifically, for developing the Machine Learning model, Python is the most widely used language for this purpose. Therefore, for development, Python is the correct choice.

For further elaboration, Python is revered for its simplicity and versatility. For example, when comparing the popular language Java [79] to Python, the same task can be accomplished in significantly less lines. Here is the necessary code required in Java to display text to the user in console, generated via ChatGPT. [A7]

Similarly, here is the necessary code to accomplish the same task in Python, generated via ChatGPT. [A8]

Apparently, Python provides a simpler approach to development. There are more examples of simplification of programs, but overall Python allows for an intuitive development process. This clean syntax can allow for development in AutoCaddie to be swiftly completed and translated for debugging purposes.

Additionally, with the inclusion of NumPy, complex mathematical functions can be encoded as simple algorithms. Both the IMU system and the Machine Learning model will incorporate matrix operations, which – depending on the number of variables – can quickly become resource intensive. For the IMU sensor data, quaternion data is being recorded, which provides a 4-dimensional vector at every instance of time. Thus, for each of these calculations, the linearity can be determined using matrix operations, which each matrix can be defined and manipulated using a single variable in NumPy. For example, the equation below Quaternion angle difference, which was defined as Equation 11.

$$\theta = \cos^{-1}(2\langle q1, q2 \rangle^2 - 1)$$

Equation 11: Quaternion angle difference using inverse cosine.

Here is the Python integration using NumPy, generated via ChatGPT. [A9]

As shown by the function defined in the code example above, the quaternion angle difference calculation can be defined by a single line of code, and each quaternion data point can also be represented by a singular variable. Being able to simplify complex equations into a readable form in combination with both linear algebra techniques and NumPy allows for the development process to be easily readable and translatable for debugging.

Moreover, every image collected by AutoCaddie can be represented by a 2D matrix of pixels. Each pixel, furthermore, can be swiftly compared using matrix operations defined by NumPy. This streamlined approach not only ensures accuracy in assessing similarities but also optimizes the entire process, making AutoCaddie's feedback system astoundingly efficient and precise. This is the rationale behind why utilizing Python for machine learning model development is as important as it is.

7.3.3.b Graphical User Interface

Moving forward, since the machine learning model utilizes Python, it falls into place that Python will be the language for software development. Specifically, the software needs to be capable of understanding user logic, interpreting data in accordance with the software flow diagram, and present information in a GUI.

Python is revered for its large community support, including many open-source libraries for everything. In terms of GUI design, tkinter [54] is one of the most popular.

tkinter simplifies the complex task of displaying the swing analysis data generated via both the linear calculations and machine learning algorithms. Its extensive set of features allows for the creation of clear and intuitive graphical elements, which ensures that the user can easily interpret the real-time coaching feedback. As discussed by the standards, having a comfortable and intuitive GUI can help the user maximize their experience with AutoCaddie.

7.3.3.c Integrated Development Environment (IDE)

To follow the economic pattern of the project, there is no necessity to extend the budget of the project to an IDE when polished software is readily available. Whether the project is employed on Virtual Studio Code (VS Code), Google Colab, Spyder, or any other IDE, there are no major benefits outside of the comfortability to the programmer and the convenience of deploying a machine learning model. As a final decision, VS Code is the preferred choice due to the local nature of the software application, as well as the extensive community support with plugins that can improve the efficiency of the development process.

Visual Studio Code, commonly known as VS Code, is a free and open-source code editor developed by Microsoft. It offers a lightweight yet powerful platform for software development, providing features like syntax highlighting, debugging, Git integration, and extensions support. VS Code is highly customizable, allowing developers to tailor their environment with various extensions and themes. Its versatility and user-friendly interface make it a popular choice among programmers for a wide range of programming languages and frameworks. For the integration in AutoCaddie, it can serve as an optimal choice for developing both the machine learning model and software application.

7.3.3.d Version Control

In the context of AutoCaddie's software development, version control is paramount. It allows multiple developers to work collaboratively, ensuring a systematic and organized approach to coding. With the project being coded in VS Code, version control becomes

even more crucial. VS Code simplifies coding tasks and integrates seamlessly with version control systems, streamlining the process of tracking changes and managing codebase modifications.

GitHub, a web-based hosting service for version control using Git, emerges as an ideal platform for AutoCaddie's version control needs. Git is a distributed version control system that allows developers to collaborate efficiently while maintaining a complete history of code changes. GitHub enhances Git's functionality by providing a user-friendly web interface, making it easier to manage repositories, track issues, and merge code changes. Its collaborative features, including pull requests and code reviews, enable seamless teamwork and facilitate efficient communication among developers. GitHub's robust ecosystem of tools and integrations, combined with its accessibility and reliability, makes it a perfect choice for version control in the AutoCaddie project. With GitHub, developers can ensure code integrity, collaborate effectively, and maintain a well-organized codebase, contributing significantly to the project's success and streamlined development process.

A diagram of the source control flow as moderated by GitHub is shown below. Each developer can work on independent sections of the program, and push the changes to the overall project repository. So long as communication is maintained, and frequent pushing and pulling is employed, the version control is efficient using this system.

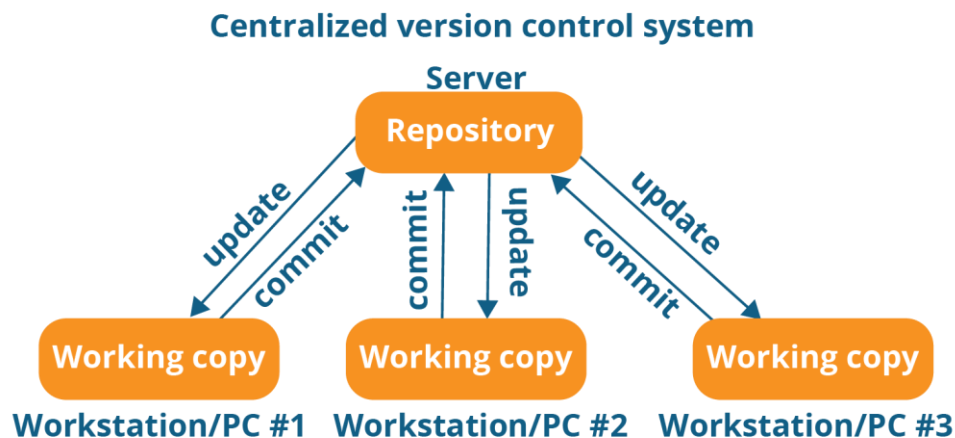


Figure 38: Version control diagram of GitHub. Each workstation can update (pull) from the main repository, then commit and push any changes back. Referenced via [80] .

7.3.3.e Documentation

Overall, clear and thorough documentation is indispensable. Documentation stands at the core of AutoCaddie's neural network and software application, offering vital insights for developers, users, and investors. It functions as a comprehensive reference, ensuring that everyone involved comprehends the project's intricacies and functions.

In the context of AutoCaddie, well-documented neural networks help simplify the extensive algorithms at a high level. It facilitates an understanding of the model between developers as changes are made or debugging is employed. Additionally, documentation accelerates the onboarding process for future expansions, enabling swift integration into

the project. For end-users, documentation serves as a manual, guiding them through AutoCaddie's functionalities, from capturing golf swings to interpreting feedback effectively.

Markdown, a lightweight markup language, will be the language used for AutoCaddie's documentation. Its simplicity ensures clear and organized content, which can boost productivity during its creation. Headers, subtext, sections, graphics, and other features can easily be introduced using a short character sequence. An example of this is shown below.

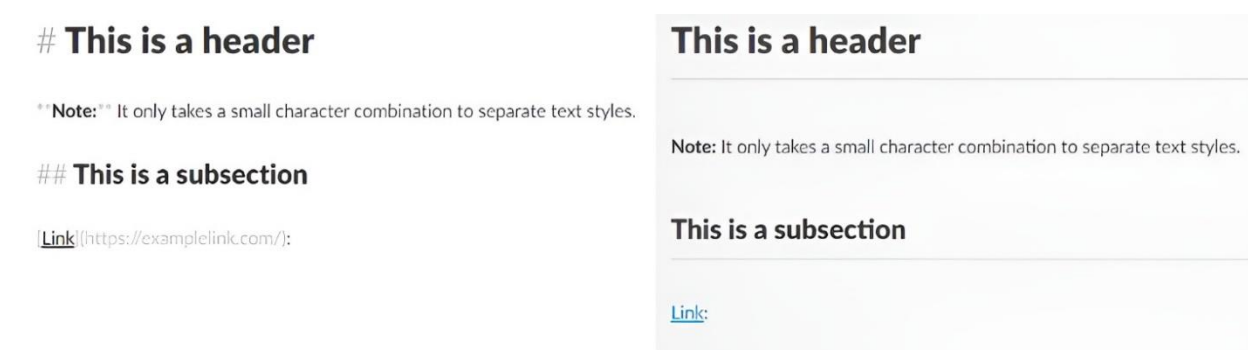


Figure 39: On the left is the source code for the markdown file. On the right is the resultant text displayed to the user. Created via [81] .

Markdown files are easily version-controlled using GitHub, enabling seamless collaboration among the developer team. Visual Studio Code, with its built-in Markdown support, becomes the primary tool for crafting AutoCaddie's documentation. Its intuitive interface allows developers to write and preview content in real-time, ensuring accuracy before publication.

By harnessing Markdown and Visual Studio Code, AutoCaddie can grow its documentation as a dynamic resource. This documentation becomes the source of knowledge for development by guiding users and contributing to AutoCaddie's success.

7.4 Communication

In the context of AutoCaddie, seamless communication between all its components is essential for its functionality. The communication pathways are structured to ensure data flow is smooth and uninterrupted. Utilizing standardized protocols and APIs, AutoCaddie establishes connections between its key elements: the Microcontroller Unit (MCU), cameras, neural network, intercommunication modules, GUI, and the end-user.

7.4.1 MCU

The MCU plays a pivotal role in data processing and exchange. It interfaces with IMUs, gathers input data, and communicates this information to the computer. AutoCaddie's MCU relies on Bluetooth technology to facilitate wireless communication. Bluetooth offers a balance between low power consumption and efficient data transfer, enabling real-time interaction with the cameras and the neural network.

During the swing recording process, the computer system will remain in a state of readiness, anticipating the reception of Bluetooth packets. The MCU will transmit four distinct signals, following a “Ready, Set, Go, Stop” format. The first signal, sent as a single beep, signifies the user to begin getting ready for their swing. The second signal, sent as another single beep, starts recording data from the IMU units. This lasts for three seconds. The third signal, sent as a quick series of two beeps, signifies that it is time for the user to begin their swing sequence. This phase also lasts for three seconds, which has been tested to contain ample time for the sequence to complete. The last signal, sent as two long beeps, signifies the end of the swing, and the user can return to the computer and start the data processing process.

Both the “Ready” and “Set” signals will serve as notifications, prompting the program to prepare for data acquisition. During runtime, the MCU continuously sends a stream of packets to the computer. Each transmitted packet will include a header, indicating the specific type of signal that has been sent.

Examples of code snippets that can anticipate Bluetooth packets over a socket connection are shown below. It utilizes the PyBluez library to establish the port connection. First, to establish a server configuration between the MCU and the computer, the code below can initialize a socket over a designated RFCOMM port, generated via ChatGPT. [A10]

There are several important pieces of information to gather from this code. The server socket is established via Bluetooth RFCOMM. Then the socket is bound to a specific port number, which will be the transmitting port of the MCU.

Additionally, waiting for data from the MCU can be demonstrated by the code below, generated via ChatGPT. The system can infinitely wait for data to be received, and, once valid data is received, the result can be used for processing. [A11]

In this code, data is infinitely trying to be received by the system. If the data received is not a valid package sent by the MCU, the system retries. If it is valid, then the data is stored into a data variable. A similar method is implemented into AutoCaddie, where the data variable is transcribed to a comma-separated-value (CSV) file, for easy reference during data processing.

7.4.2 Cameras

AutoCaddie's camera system captures all the necessary visual data of the golfer's swing. The system will consist of two cameras, one positioned behind the user, and another to the side of the user. By capturing both perspectives of the golfer, the hyperparameters in question can be comprehensively tracked throughout the swing process. In addition, both perspectives are supported by GolfDB's video collection.

Each camera can record at 30 frames per second, which provides ample data for attributing every action within the swing sequence. Additionally, it corresponds to the provided video framerate of GolfDB. The cameras will be tethered to the computer via

USB, which will allow for rapid transmission of data without reliance on signal receiving. Additionally, since the data will be stored as it is captured, there is no direct need for ensuring synchronization between the two streams, especially when the data preprocessing algorithms will attribute the actions between the two independently. The synchronization that occurs is to ensure frames are captured properly from both streams and, if so, only store on that success.

In terms of coding implementation, we can utilize OpenCV [45] a popular computer vision library in Python. Below is an example code snippet demonstrating capturing video frames from two cameras simultaneously, generated via ChatGPT. [A12]

In this code, the camera objects are initialized to separate indices representative of the USB ordering. From there, each camera is read, and if valid frame data is captured, it is stored in the array of frames. If one of the cameras does not record frame data, that instance of time is skipped. That way, both cameras capture the same sequence of frames.

7.4.3 Neural Network

The neural network serves as the core analytical engine within AutoCaddie's framework. Its primary function is to process data obtained from the cameras, using extracted angle data to create an educated analysis on how the user should complete a swing sequence. The neural network employs complex algorithms to decipher this information, identifying specific patterns within the golfer's movements from training. These patterns are crucial to the output data, as a prediction swing is generated referencing GolfDB's video collection of proper swings.

The neural network relies on effective communication with the cameras. This data is sent through preprocessing algorithms before being read, which helps prepare the data to be as transparent to analyze and boost the accuracy of the system. From here, due to the nature of a neural network, the entirety of the communication process can be encapsulated by a single function call. Similar to how Java creates classes, the complex analysis of the neural network can be abstracted to a separate section and called via a function. Therefore, data communication with the network is simple.

An example of this is shown below. The function is defined separately from the main body of code, generated via ChatGPT, and will send the camera data to be analyzed and return float data representative of the score of each feature. [A13]

7.4.4 Internal Communication

Beyond the neural network, the primary internal communication involves the MCU transmitting data for processing. This process mirrors the neural network setup, where the core logic is isolated from the main code, and outcomes are relayed through a function. The primary distinction lies in verifying that the data from the MCU corresponds to the correct signal before entering the interpretation algorithms. As detailed in Section 5.4, the MCU data is tagged with a header, indicating the specific signal, ensuring accurate data examination and understanding.

An example of deciphering the MCU data in csv format is shown below, generated via ChatGPT. In this case, the header “data” signifies the end of the swing sequence. [A14]

7.4.5 GUI

AutoCaddie’s GUI serves as the user’s window into the analysis. It receives the processed data after the swing has been completed and both the quaternion data analysis and neural network systems have completed.

This interface acts as the means of communication for conveying the swing data from the system to the user. The effectiveness of AutoCaddie lies upon the complete and coherent presentation of this information. Unlike needing to engage with peripherals, the GUI’s role lies in transforming the data into visually comprehensible graphics such as timelines and plots. This transformative process, detailed further in **Section 7.7 Graphical User Interface**, is a pivotal step, ensuring that users receive a rich, meaningful, and insightful overview of their swing performance.

7.5 Data Processing Algorithmic Integration

Moreover, the exact data processing algorithms can be integrated into AutoCaddie’s software to ensure the seamless transfer of data throughout the system. Specifically, this refers to frame extraction, image standardization, zero-padding, and kinematic equation algorithmic integration. The implementation strategy involves segregating these algorithms into discrete logic units, thereby delineating them from the overarching software design. This approach facilitates a modular and object-oriented execution of the software, allowing for efficient referencing and utilization of specific algorithms as required in different components of the system.

7.5.1 Algorithmic Structure

AutoCaddie leverages the tkinter framework, which is a cross-platform GUI toolkit for the Python programming language. It has been referenced that this project will utilize Python for all software development, and thus, through thorough research outlined Chapter 3, tkinter provides enough support for intricate data formatting while also providing portability and not demanding a large learning curve for development. By utilizing its tools, which will be highlighted in **Section 7.8**, AutoCaddie will be capable of providing the best possible user experience.

By utilizing the tkinter framework, it follows the modularity design principles that every algorithm is extracted into its own container of logic. In other words, each algorithm is encapsulated into separate .py files. This organizational strategy enhances code readability, facilitates testing and maintenance, and provides simpler integration into the entire system.

An example of the software hierarchy is shown below. Every Python (.py) file contains logic critical to execute the program, whereas *main.py* contains the primary controls. Through file referencing rooting back to *main.py*, it is possible to utilize code through abstracted files, such as each algorithm being implemented.


```
AutoCaddie/  
|-- main.py  
|-- algorithms/  
|   |-- __init__.py  
|   |-- frame_extraction.py  
|   |-- image_standardization.py  
|   |-- zero_padding.py  
|   |-- kinematic_equation_algorithm.py
```

Figure 40: A folder representation of the AutoCaddie software. Every indent is equivalent to the number of folders containing each file, where the `algorithms/` folder contains every algorithm Python script. The controlling logic is contained within `main.py`.

As shown by Figure 40, the AutoCaddie software hierarchy can be explained through abstraction principles. The `algorithms/` folder contains the Python scripts containing each algorithm that has been mentioned prior. Additionally, the `__init__.py` file signifies the `algorithms/` directory as a Python package, which enables imports between algorithm modules. By utilizing these design principles, it is possible to modularize the logic of each algorithm.

A primary consequence of separating every algorithm into separate files is to enhance unit testing. Unit testing is critical to ensure the reliability of each algorithm implemented, as it is the process of implementing software checks within specific sections of code to ensure overall functionality. With every algorithm residing in its dedicated `.py` file, unit testing becomes a streamlined process, allowing for the meticulous examination of individual components in isolation. This approach, overall, expedites the development lifecycle and reduces the burden of debugging. By subjecting each algorithm to rigorous unit tests, we can verify its conformity to expected behavior and promptly address any deviations.

By utilizing `tkinter`, the Python framework `unittest` [82] allows for direct integration of unit testing with modularized code. By referencing Figure 41, it can be observed that the project hierarchy has the controlling logic at the parent folder, with algorithms abstracted down a directory into an `algorithms` package. It is possible to modify this structure to create a tests package, where each `.py` file within this package tests the behavior of a corresponding algorithm.

```

AutoCaddie/
|-- main.py
|-- algorithms/
|   |-- __init__.py
|   |-- frame_extraction.py
|   |-- image_standardization.py
|   |-- zero_padding.py
|   |-- kinematic_equation_algorithm.py
|-- tests/
|   |-- __init__.py
|   |-- test_frame_extraction.py
|   |-- test_image_standardization.py
|   |-- test_zero_padding.py
|   |-- test_kinematic_equation_algorithm.py

```

Figure 41: A folder representation of the AutoCaddie software with the unittest module. For every test file within the tests package, there is a corresponding algorithm in the algorithms package.

As shown by Figure 41 the *tests* package contains Python scripts for testing the functionality of each algorithm defined within the *algorithms* package. More specifically, each test module is prefixed with “test_” to adhere to the unit test naming convention. Once again, the *__init__.py* file signifies the directory as a Python package. On a separate note, code invocation can be defined from the *main.py* file. The integration of these algorithms is straightforward because tkinter facilitates the instantiation of GUIs and their interactions with underlying algorithms.

As demonstrated by the code, the *algorithms* package can be imported through a simple import statement. This package contains the scripts necessary to run each algorithm, where *frame_extraction* contains the logic of converting a video file to a list of frames. This logic is defined within the *main.py* file of the software. This example showcases the seamless integration of these data processing algorithms into the tkinter-based application. Similar calls can be used for each algorithm, following this structured pattern, and ensuring a clean separation of code.

Moreover, unittest also provides seamless integration into the software development of AutoCaddie. The following example below showcases code that can be utilized within “test_frame_extraction.py” to unit test the functionality of the frame extraction algorithm. [A16]

This Python class contains a test method for ensuring that the number of frames contained within a video is exactly equal to a specified amount. For the purpose of AutoCaddie, this would be to ensure that a video can be extracted to exactly 90 frames. Similar to before, the *frame_extraction* logic can be imported with a simple import statement. In terms of new code, the *self.assertEqual()* call is used to execute the unit test, and the *unittest.main()* call is used to run the logic from this Python class.

Overall, utilizing these two Python libraries allows for robust software development with AutoCaddie. The algorithmic structure follows a modularized design, where each algorithm is abstracted to separate scripts contained within a single algorithms package. Similarly, testing can be abstracted to a tests package, where there is an equivalent number of testing scripts as there are number of algorithms. By following these design

principles, the development of AutoCaddie can be streamlined, and the resulting software will be optimized.

7.5.2 Frame Extraction

Frame extraction is the first step in image data preprocessing for usage within AutoCaddie's CNN. This process involves breaking down a continuous video feed into distinct images, each representing a specific moment in time. The extracted frames serve as snapshots for precise examination of the golfer's movement.

Each frame can be represented as a 2D matrix of pixels, where each pixel is equivalent to a 3-channel number (24-bits), each channel composing a red, green, and blue value. This matrix format allows for algorithms to manipulate video data effectively. This representation provides a comprehensive understanding of color intensities, which is essential for image processing.

OpenCV allows the software to create the illusion of downloading and extracting a video by live parsing frames from a video feed. The implementation below showcases this relationship, where the function defines the behavior of parsing two separate camera streams. [A17]

In this structure, the frame extraction logic is encapsulated in a separate module, *frame_extraction.py*. The script, when *extract_frames()* is called, reference predefined cameras connected to the computing device, and initializes empty lists of 2D matrices for each camera. It then iterates through the live camera streams and extracts the current frame as it is presented. If there an error from either device, the current moment of time is skipped, and the next frame is checked. After the video feed duration, the camera streams are released to avoid unwanted memory clogging, and the two lists of frames are returned. This frames object can next be utilized by the bounding box annotation algorithm.

7.5.3 Image Standardization

The next step in the procedural preprocessing of video data is ensuring that input data for a CNN maintains consistent format. Frame standardization, in general, applies to formatting an image so that it matches the dimensions of another specified image. However, as described by the inclusion of MediaPipe, this is not necessary so long as the frame contains all the necessary information of the user.

The important information from the user can be extracted as angles using MediaPipe's API. This information is fed into a csv file, where the neural network can analyze it. So, as long as the entirety of the user can be picked up from the camera feeds, then this process is not interrupted.

It is important, for seamless integration, for the user to be the primary focus of the video feed. It is possible to disrupt the focus of the video with a blocked camera or positioning that does not cover the entire body. The video feed standardization is tasked to the user,

to make sure each camera is positioned in such a way that the entire body is within the frame without major blockage. That way, the information can be extracted accurately.

7.5.4 Kinematics

Lastly, quaternion data stores spatial data throughout the swing preparatory stage of the user's initial posture. There are two IMUs positioned on the user's shoulders, one on the left and one on the right. By referencing *Equation 11*, which is reshown below, it is possible to determine the angle between the two quaternion data points for every instance of time during the swing.

$$\theta = \cos^{-1}(2\langle q1, q2 \rangle^2 - 1)$$

Equation 11: Quaternion angle difference using inverse cosine.

It has been established that NumPy will be used to implement matrices operations. Each quaternion data point can also be represented by a four-value vector, and the composition of 10 data points per second for three seconds alludes to a 50x4 dimension matrix for each IMU. Therefore, it is possible to determine θ by the code shown below. [A23]

In this code, Q1 and Q2 contain all the quaternion data points stored as matrices. The function *calculate_angle(Q1, Q2)* returns a vector of the angle between the two IMUs for every instance of time recorded.

7.6 Data Management

AutoCaddie is composed of complex systems, each responsible for managing the transformative flow of data. The process initiates with thorough data collection, which is completed through camera and IMU peripherals. Once the data is gathered, it is augmented in different methods. For the IMU data, data is sent via packet data with a specified header which denotes the role of the specific package sent. The camera data is extracted into frames, where preprocessing algorithms will be applied. Therefore, this data finds a home in storage, where AutoCaddie's efficient data storage mechanisms are employed to organize and manage information systematically.

7.6.1 Data Collection and Storage Requirements

Overall, there are two avenues of data collection through the AutoCaddie system: through the cameras, and through the IMUs. Both of these devices provide valuable information for the AI to analyze, and each data type is managed separately to assess different topics within the user's golf swing technique.

First, there are two cameras tethered to AutoCaddie's central computer. The responsibility of each camera is to provide a video stream of the entire swing motion so that it may be analyzed. However, the beginning of the video sequence is initiated by the "SWING" packet. In short, frames are continuously connected for a short duration, and appended to a list. We know that, for the HP 320 FHD camera, it records at 30 frames per second and 1080p resolution, which is equivalent to 1920x1080 pixels. By multiplying this across 3 seconds, with 30 frames captured per second, with each pixel representing 3 channels (24 bits), it can be determined the expected storage requirement to handle

each video. Through testing, this value was found out to be slightly less than 2 MB. Finally, there are two cameras, so for every swing, there will need to be roughly 4 MB available. This is a temporary requirement, as every video will be deleted after being analyzed.

Similarly, the IMU records quaternion data over time. The research for quaternion data analyzed that, for the processing algorithm, two quaternion matrices of data are necessary, each of size one less than what is transmitted. The MPU6050, the selected IMU unit, can capture quaternion data during the golf swing action. Each quaternion data point consists of 4 axes, each axis containing a 32-bit number. For measuring 10 data points every second for three seconds, it requires $32 \cdot 4 \cdot 10 \cdot 3 = 3840$ bits of data. Moreover, to record acceleration data, it requires 3 axes of measurements, which is equivalent to another 2880 bits. Multiply that by two IMU sensors, and the overall storage required to receive the input data is 1680 bytes of data.

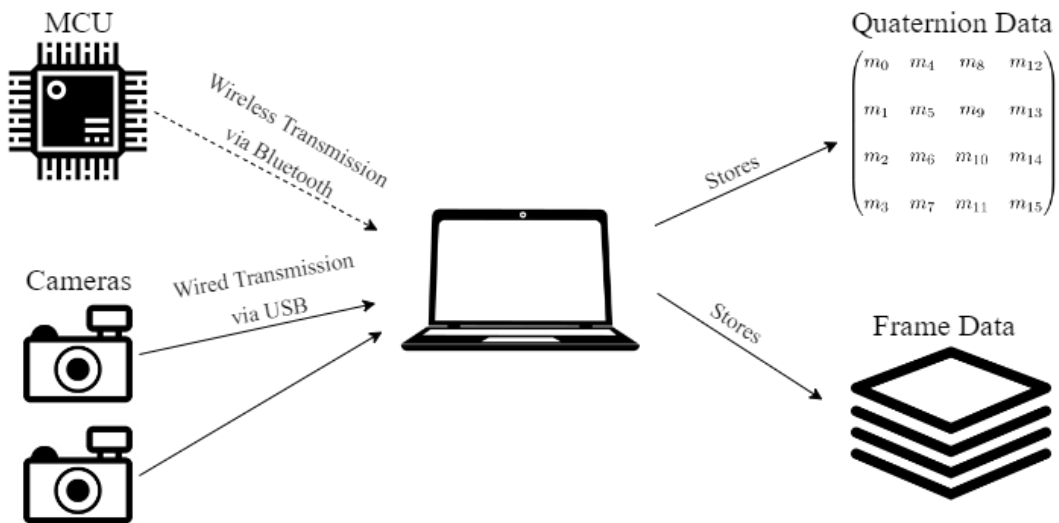


Figure 42: Figure representing the data flow of the peripherals to the computing device. Quaternion data is packaged and transmitted from the MCU via Bluetooth, and video data is transmitted from the cameras via USB.

7.6.2 Video Data Augmentation

Once the data has been received by the computing device, the next step is to process it. Both the camera data and quaternion data experiences manipulation before being directly analyzed, which allows for complete analysis throughout the entire swing process.

First, the camera data undergoes preprocessing algorithms, which reduces the storage requirements necessary in the system. It is still a requirement for AutoCaddie to be able to support the resource demands of acquiring the original data, but understanding the scalability of these operations, as well as the management of augmented data is crucial for understanding the design of the software.

The primary takeaway from the video data is that it does not go through much frame-by-frame manipulation. MediaPipe presents an easy way to implement accurate data, so the important factoid of information is to be able to have it reduced to only save the important aspect of the swing.

It has been previously established that the 3 second video clip, saved through both cameras, is stored with an estimated 4 MB of storage. However, the motion can be roughly found through half of the length of each video, where the rest of the time is spent beginning to move and relaxing after the motion. This redefines the storage intensity of the video data before data processing to around 2 MB of data.

By allowing the user to select the beginning and the end of the swing, the total time it takes to process the swing using the neural network can be cut in half.

7.6.3 Quaternion Data Augmentation

The other data augmentation system within AutoCaddie is through the quaternion data collected by two separate IMU units. It has been established that storing this data initially requires 2800 bytes of data. Therefore, each quaternion matrix can be represented using 1400 bytes of data. Referencing Equation 11, we can determine the additional data requirement necessary for the solution vector θ storing the angle between each quaternion matrix at every instance of time. Specifically, the IMU sensors are each recording 10 data points per second for five seconds, a total of 50 data points each. Therefore, the output vector θ will only contain 50 angles, which can be represented by the float primitive, totaling 100 bytes.

Moreover, at this point, the original data is no longer needed for further analysis, so it can be removed from the system to increase available storage.

7.7 Neural Network and Kinematic Algorithm Interpretation

The previous sections have discussed the implementation of AutoCaddie's neural network, preprocessing algorithms, and kinematic equations. This section covers the next step in the software flow: interpreting the results gathered from both the neural network and kinematic processing.

7.7.1 Neural Network Output Interpretation

The neural network of AutoCaddie serves to take in a user's initial momentum and generate a proper swing to guide the user in the right direction with the fundamentals of their swing.

GolfDB is used to provide information on what a proper golf swing is to look like for the user. It is trained by observing the initial 30 frames of a swing, followed by observing the rest of the motion that can be generated from that movement. Thus, upon deployment, it's purpose is to perform that 2nd stage generation of the rest of the swing motion given the user's first 30 frames of movement. By doing this, the system accounts for users with different paced swings and can be generalizable to different body types.

Not to mention, the network is trained through angles, not directly the video data. Using MediaPipe, the angle data from the golfers is extracted for the inner arm, shoulders, and hips, and fed into a csv file. This is also done for the user. That way, the angles that should be created are plotted on a timeline, and compared to the user's motion.

An example of this is shown below in Figure 43. The orange line represents the predicted angle, and the blue line represents the actual angle. Observing this plot, we can see that the user's arm was extended further than it should have been around the 7-8 frame range. This informs the user that they should relax their arm more towards the end of the swing.

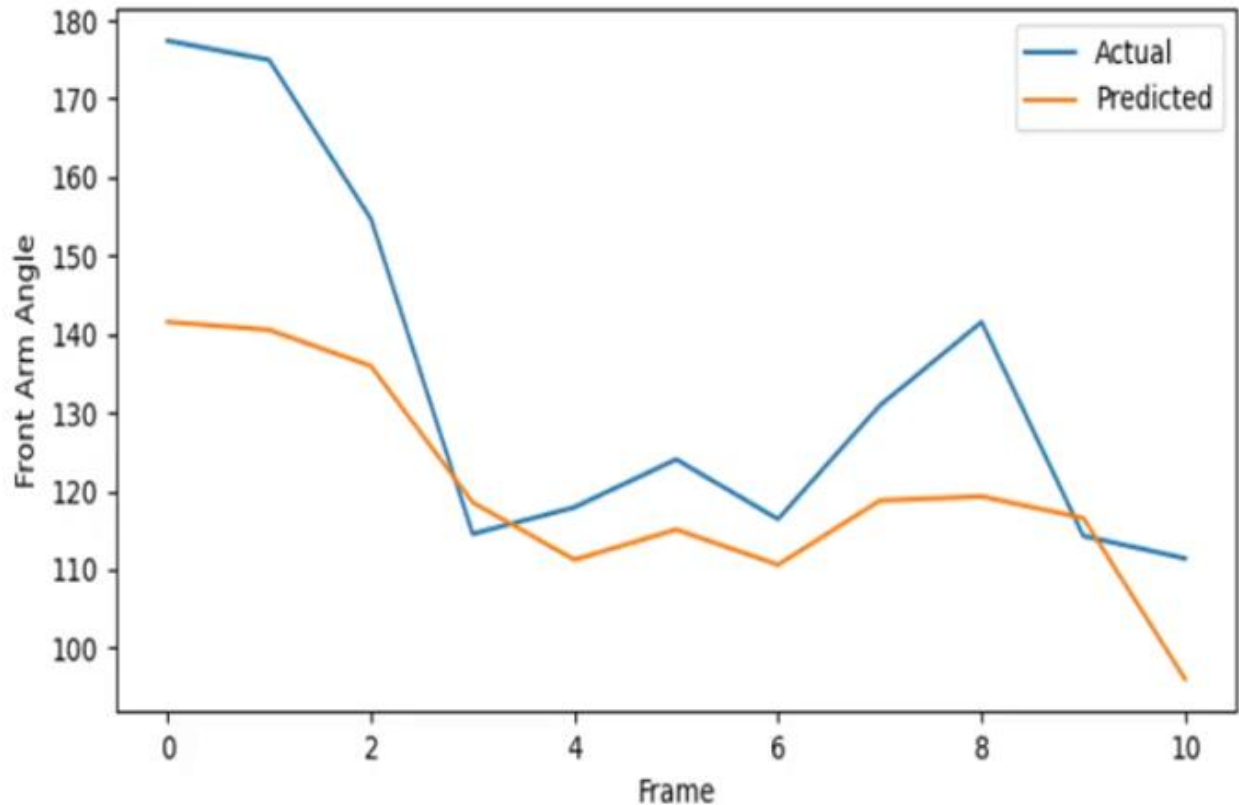


Figure 43: The predicted inner arm angle history versus the user's actual inner arm angle history after the initial 30 frames of a swing.

7.7.2 Quaternion Data Output Interpretation

The quaternion data that is given to the computing device is representative of spatial and rotational coordinates, as well as general acceleration information. Specifically, through the IMUs, there are 10 quaternion data points captured every second, which shows the orientation of the least-dominant arm's upper arm and forearm. By having an IMU on each point of interest on the body, the angle between the two can be calculated, and if the points are linear, there will be results that can justify this.

Furthermore, *Equation 11* defines a vector, θ , that contains every resultant angle from the quaternion processing. This vector can be iterated over, and numerically analyzed. There are two direct ways that will be employed to analyze this behavior: thresholding and visualization.

Thresholding refers to defining a specific value and comparing the angle to determine if it is less than the threshold amount. If there exists an angle or group of angles such that the resultant value is larger than the threshold, then the system marks a flag at this

location. For every point that falls above the threshold, the flag will signify the GUI to explicitly mention where in the swing the arm broke its straightness in respect to the action sequences. If the instance of time, listed as the order of the quaternion data points, falls within the same index range as the frames representing a specific action sequence, then the system can infer that the fault occurred at that specific action.

Formally, let's define the address of the swing to indices $i \in [x, y]$. Both the video streams and quaternion data points were captured in respect to the same starting point in time, which has been ensured through MCU transmitting a "Start" package that the user triggers via button. If θ contains an index, j , such that $\theta_j > \theta_{threshold}$, and $i < j < k$, then the system can define an error within the address action of the swing. However, this threshold value will be discovered through testing, which is outlined in **Chapter 9**.

The visualization of this process refers to creating a timeline and plotting the magnitude of the angle at every instance of time. There will be, at maximum, 50 data points, each separated by $\frac{1}{10}$ th of a second. This way, for the swing duration, any deviations from the expected hyperextended angle behavior can be transparently viewed.

Also, for creating this visualization, it is important to label each action of the swing, like thresholding's methodology. By combining the visualization with the thresholding, it is possible to observe an entire script of the user's swing and make educated conclusions about the body position/ligament linearity using kinematic data.

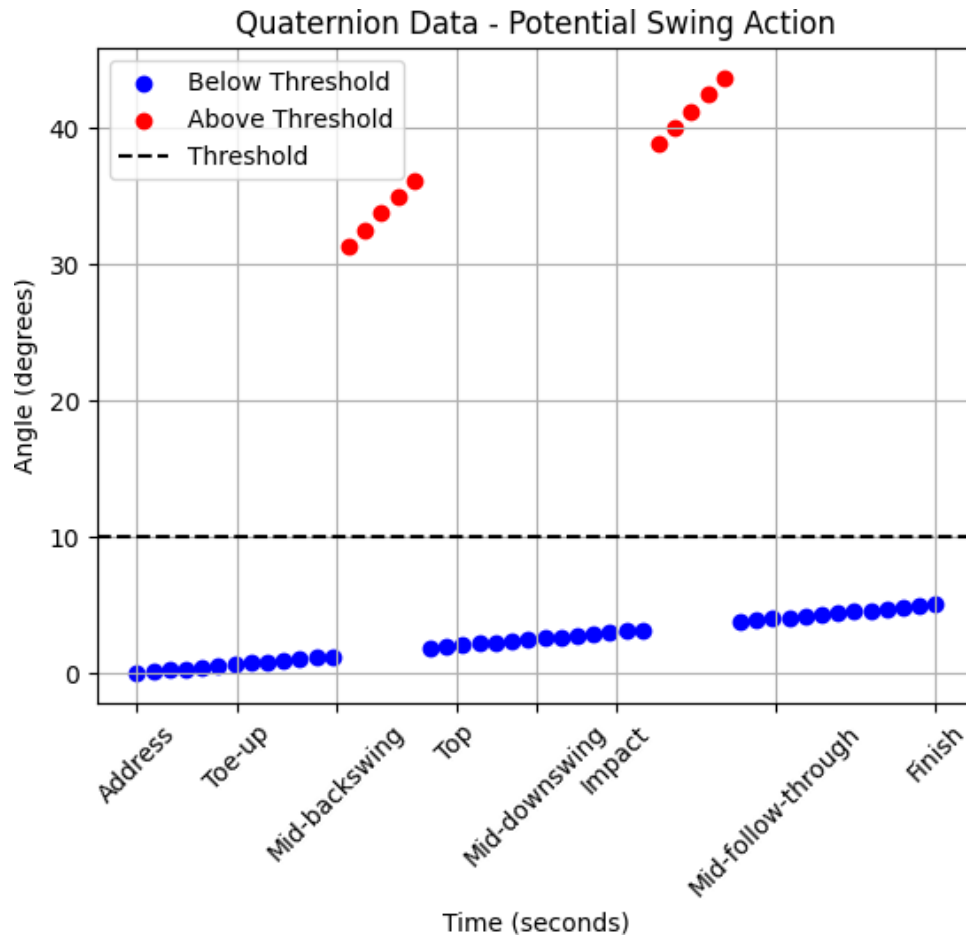


Figure 44: An example of the visualization of quaternion data processing. The blue data points represent instances of time where the angle between the two IMUs is less than the defined threshold, and the red data points represent instances of time where the angle is larger than the threshold. Each action is labeled along the x-axis, so the user can attribute specific errors to the exact segment of the swing. Referenced via [69].

7.8 Graphical User Interface

7.8.1 Framework Selection: *tkinter*

The graphical user interface, or GUI, of AutoCaddie will be representative of the output mechanism of providing feedback to the user. The GUI will be responsible for displaying formatted statistics pertaining to the user's previous golf swing technique and analyzing it using the pre-established data analysis methods. Through the mediums of charts, graphs, and percentages with respect to time and swing actions, the user can gain insightful feedback from AutoCaddie's AI.

As researched in **Section 3.5.6**, the selection of a particular GUI library is a critical decision in AutoCaddie's development. Numerous libraries have been considered such that the selected one needs to contain integration with NumPy, OpenCV, sklearn, Keras, and Tensorflow. Among those reviewed, *tkinter* emerged as the strongest choice. It serves as a cross-platform GUI toolkit for Python which adheres to the project's goals.

One significant advantage that tkinter presents is its compatibility with different operating systems, allowing it to be run within Windows, MacOS, and Linux. Plus, with the design choices of data processing, it can be deployed on a diverse background of computing devices.

Overall, tkinter is the most suitable choice for this project. Its portability and naïve appearance allow for this tool to act powerfully without demanding a large load of resources. Additionally, on most platforms, its ease-of-use aligns with this project's goals. Overall, tkinter will be utilized because it provides the right balance of features, ease-of-development, and cross-platform compatibility.

7.8.2 Parameters and Data Output

The data processing algorithms are responsible for converting the user data into an interpretable form. These algorithms are systematically organized into a modular structure within the algorithms package. The integration of these algorithms into AutoCaddie's software design is intended for a smooth user experience with the GUI. Therefore, the results of these algorithms collectively contribute to a holistic representation of the golfer's swing. The algorithms package feeds data through both a CNN and kinematic equations to create the resulting output data to be formatted by this GUI program. These two subsystems provide a multitude of data that can assist the user with their golf swing with direct analyses of swing technique.

First, using MediaPipe, the video data is extracted into a collection of angle histories. This is representative of the inner arm angle, shoulder straightness, and hip rotation during the swing. This data is fed into a LSTM that can use the initial information of the user's swing to generate the angles of a proper motion. This can be compared to the actual user's swing as was observed.

The integration of the CNN into the AutoCaddie GUI allows users to receive real-time feedback on their technique. The GUI dynamically displays similarity weights for each parameter at every frame, enabling golfers to assess and refine their performance. The implementation involves the following steps:

1. **Frame Sequence Input:** The GUI creates a video file using live recombination of frames from the real-time swing.
2. **Angle Extraction:** The CNN, sourced through MediaPipe, takes the swing and extracts useful angle information to a csv file.
3. **Swing Prediction:** Using the initial 30 frames from the user's swing, the rest of the motion is generated and stored in a csv. This can be referenced by the GUI to create charts.
4. **Real-Time Display:** The GUI dynamically showcases the predicted swing in comparison to the actual swing. This real-time feedback allows golfers to make instant adjustments and better understand their strengths and areas for improvement.

Second, the quaternion data, representing arm orientation during the golf swing, undergoes two primary interpretation processes for GUI presentation: thresholding and visualization. These processes enhance the golfer's understanding of arm straightness and aid in correlating deviations with specific swing actions.

1. Thresholding: The GUI applies a defined angle threshold to identify instances of arm straightness deviation. Flags are raised at frames where the arm angle surpasses the threshold, allowing users to associate errors with specific swing actions.
2. Visualization: A graphical timeline of angle magnitudes at each instance of time is generated. This visual representation, combined with thresholding, enables users to transparently observe and correlate arm deviations with precise segments of the swing.

This methodology, including visualization of data points, has been demonstrated through Figure 44 in **Section 7.7.2**. But this allows for the GUI to dynamically display the threshold results and visual timeline.

Both of these systems incorporate real-time feedback mechanisms, providing golfers with a comprehensive view of both the CNN analysis of their swing technique and the spatial dynamics captured by quaternion data. Moreover, the modular design ensures that the GUI remains intuitive and user-friendly, fostering a valuable tool for golf improvement. [A35]

As shown by the code, the updating of this GUI frame is simply called by the *gui_update()* function call, utilizing the new data results from the previous algorithmic calls. Through this integrated approach, AutoCaddie provides golfers with a sophisticated, user-friendly platform for refining their golf swing technique based on both visual and neural network feedback.

7.8.3 Project Directory Structure

Compiling all this information, a prototype project directory is shown below.

```
AutoCaddie/
|-- main.py
|-- gui_module/
|   |-- __init__.py
|   |-- autocaddie_gui.py
|   |-- other_gui_components.py
|   |-- charts/
|       |-- __init__.py
|       |-- chart_module.py
|       |-- timeline_module.py
|       |-- other_chart_components.py
|   |-- plots/
|       |-- __init__.py
|       |-- plot_module.py
|       |-- other_plot_components.py
|-- algorithms/
|   |-- __init__.py
|   |-- frame_extraction.py
```

```

|-- image_standardization.py
|-- zero_padding.py
|-- kinematic_equation_algorithm.py
-- neural_network/
|-- __init__.py
|-- cnn_module.py
|-- linear_svc_module.py
|-- process_videos.py
|-- other_neural_network_components.py
-- data_processing/
|-- __init__.py
|-- data_processing_module.py
|-- other_data_processing_components.py
-- tests/
|-- __init__.py
|-- test_frame_extraction.py
|-- test_image_standardization.py
|-- test_zero_padding.py
|-- test_kinematic_equation_algorithm.py
|-- test_cnn_module.py
|-- test_linear_svc_module.py
|-- test_data_processing_module.py

```

Figure 45: A prototype project directory structure for AutoCaddie.

As shown by Figure 45, the project can be divided into many different modules, each containing an abstracted responsibility of logic.

The ‘main.py’ file is responsible for controlling the main flow of the software. Depending on the state of the program, whether it be through calibration, data collection, or output presentation, this file will dictate which GUI window is displayed to the user. Thus, each subsequent module within the project can directly be imported into main.py, and used through single line calls. This enhances the readability of the program and allows for transparent readability. A key signifier of a package is the existence of the ‘__init__.py’ file.

The ‘gui_module’ module contains necessary scripts and files for displaying a GUI to the user for AutoCaddie. It has a class declared for plots, which will be the output of the kinematic data, as well as a class declared charts, which can contain statistical data relevant to the neural network. Each of these output mediums contain their own package with individual logic, which can ide extraneous code from the developers.

The ‘algorithms’ module contains the necessary files for completing the data preprocessing algorithms. The procedural transition of video data to standardized frames and csv data can be referenced minimally through the main code by using logic within this module. Not to mention, the kinematic data processing can also be referenced through here.

Similarly, the ‘data_processing’ module is used for calling necessary code for converting preprocessed data to readable output. This involves feeding frame data to AutoCaddie’s CNN and LSTM, and IMU data to the kinematic equation.

The ‘neural_network’ module is the bulkiest section in this project, as it contains a trained LSTM. This network is trained on a collection of 1400 compressed golf swing videos and

is constructed using extensive layer connections. That is why it is important to ensure this module is abstracted, as importing the model provides a significant improvement to code readability.

Lastly, the 'testing' module implements the unit tests for each algorithm implemented. By creating specific tests utilizing the 'unittest' library, the output of each defined function can be compared to expected values to ensure functionality. To follow proper naming conventions, each testing file follows the "test_[file name].py" syntax style.

7.8.4 GUI Design

7.8.4.a Main Menu

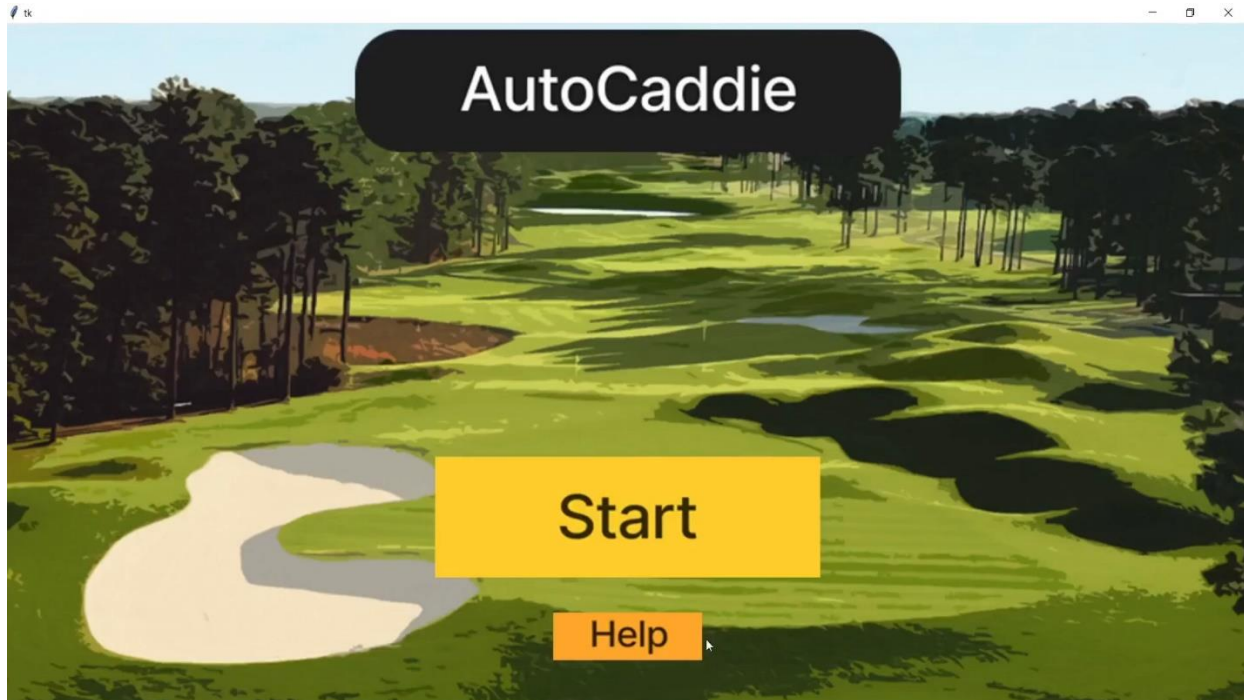


Figure 46: Main menu screen of AutoCaddie.

The figure above represents a prototype for the main menu screen of AutoCaddie. There are two options.

- “Start” is used to begin the synchronization process for the user with the IMU data and camera feeds. This is the button to initiate the software flow of AutoCaddie.
- “Help” is used to view the application manual. This is provided to assist the user through the program.

7.8.4.b Calibration

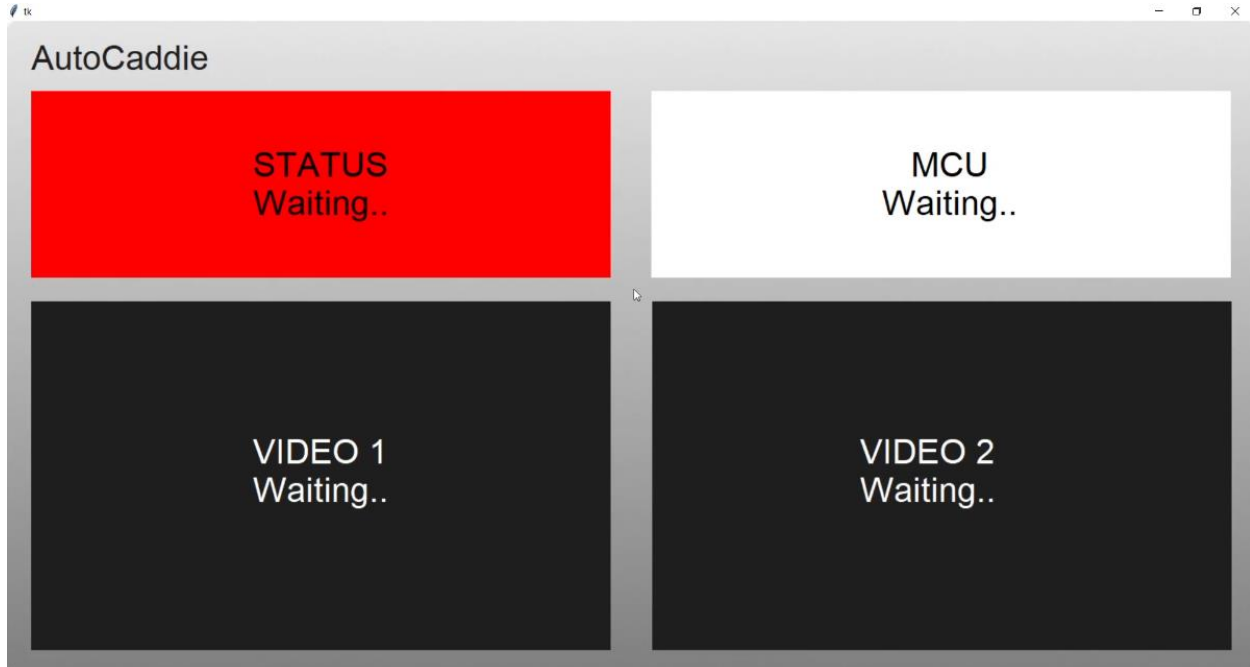


Figure 47: The calibration screen for AutoCaddie.

The figure above represents a prototype for the calibration screen of AutoCaddie. There are two areas warranting attention: the camera feed status and the IMU signal status. At this point in the program, the user must turn on the IMUs and ensure the cameras are properly tethered to the computing device. For each box, the background will change to a dark green, and the text will change to “Ready!” once the setup is completed.

- For the cameras, the setup includes connecting two cameras and opening them through OpenCV. There can be more than two cameras connected to a computer at a time, and the user can switch between the cameras to use by tapping the box on the feed they wish to replace.
- For the MCU, this is dependent on the Bluetooth connection of the HC-06s to the computer. Once this connection is made, and a successful serial connection is established, it will be ready.

At any point in time, these streams can be turned off by simply closing the program using the “x” button on the window.

Once both systems are initialized, the user can begin the recording sequence by simply pushing the “Go” button on the MCU.

7.8.4.c Recording

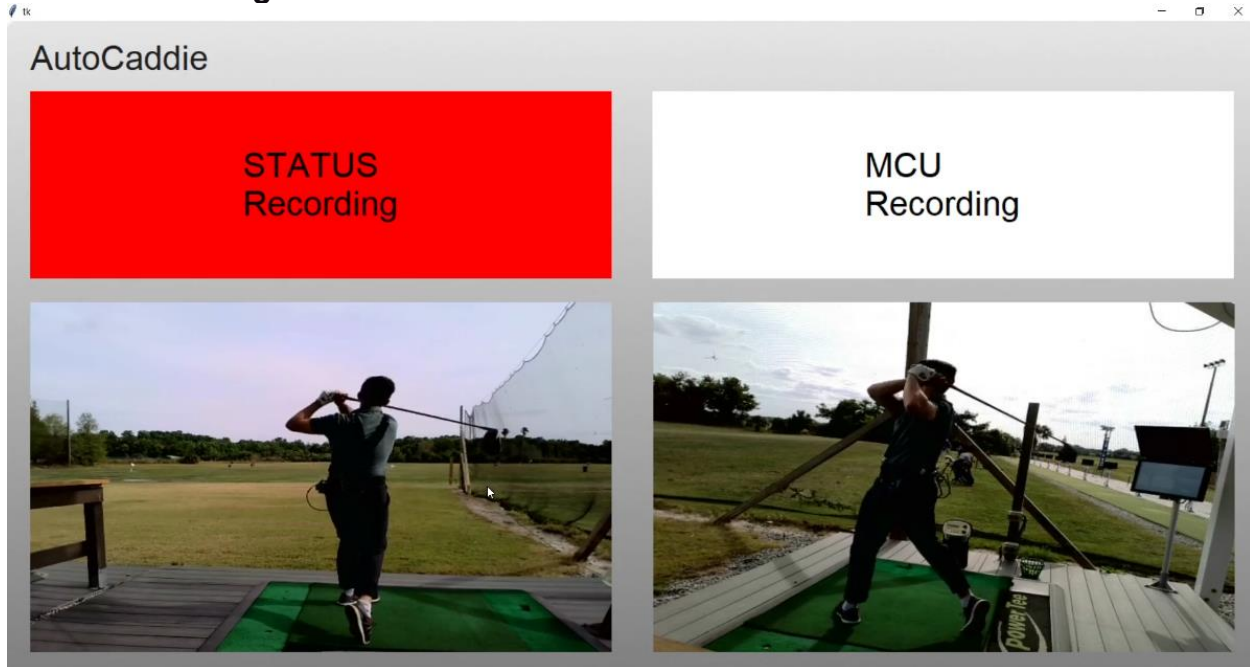


Figure 48: The recording screen for AutoCaddie.

This menu is similar in structure to the calibration menu. The justification for this is to make the GUI as intuitive as possible for the user. As mentioned earlier, this screen encompasses the recording sequence with a “Ready, Set, Go, Stop” system.

- Ready: The MCU beeps once, alerting the user to get into position.
- Set: The MCU beeps once, signifying that the user should be in a proper ready position. Quaternion data is measured from the user’s shoulders during this phase for three seconds.
- Go: The MCU beeps twice quickly, alerting the user to perform their swing sequence. Video data is recorded from both cameras for three seconds during this phase.
- Stop: The MCU beeps twice slowly, informing the user that the sequence has ended. The GUI will switch to the processing results screen, and the user is to return to the computer to observe the results.

7.8.4.d Processing Results



Figure 49: The processing results screen for AutoCaddie.

Once the user has completed their swing, they will be given the opportunity to limit the video size to assist in timely and accurate AI feedback. This section of the GUI serves as an interface for the user to begin the data analysis procedure.

- First, the user will select if they are satisfied with their swing. If not, they have the ability to redo it, which will allow the user to setup and push the button on the MCU to begin recording again.
- Second, the user will select a starting and ending frame. The two frames selected must contain the entire swing motion, but also must remove any extra segments of the video that do not contain appropriate swing information. This allows for the time complexity of the AI to be cut in half.
- Third, the data is sent to the data processing algorithms as described earlier in this chapter. After a short delay, the GUI will switch to the results menu, where the user can observe their feedback.

7.8.4.e Results

There are three main sections of the results: the main page, the hardware feedback, and the computer vision feedback. Each type of data collection provides a separate metric of feedback, so it is abstracted in the analysis.

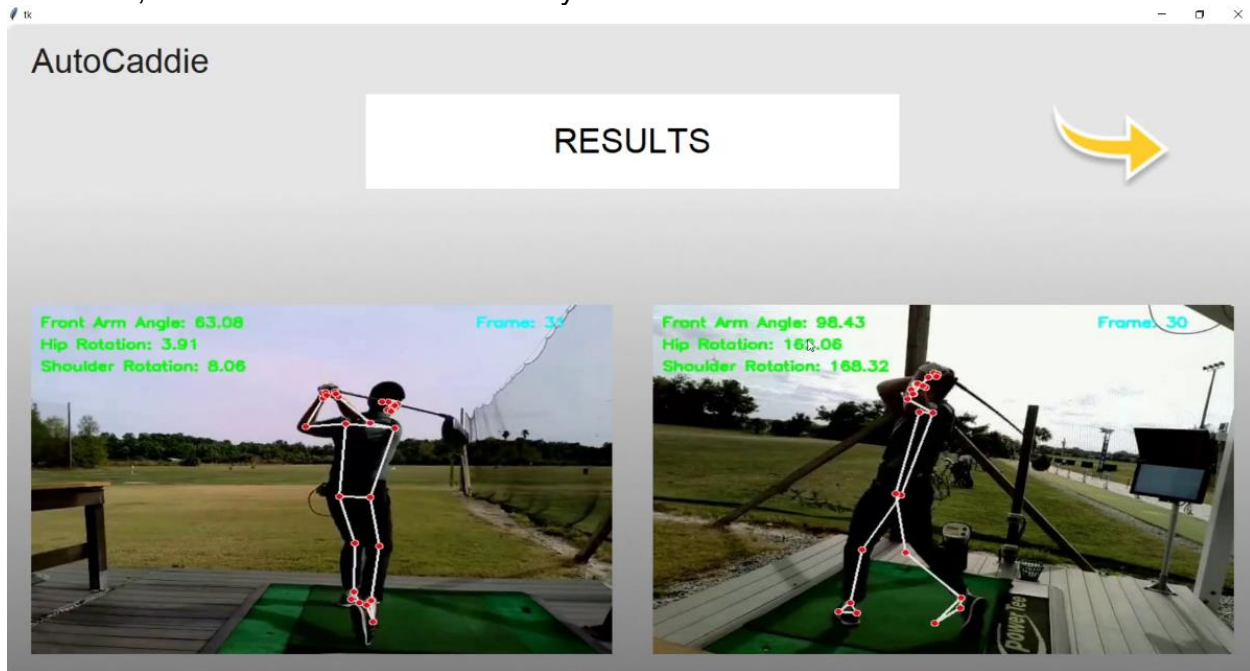


Figure 50: The results landing screen for AutoCaddie.

Figure 50 represents the landing page of the results. The user's swing is displayed through both camera feeds, and with MediaPipe's skeleton mapped onto the user. For the duration of the swing, the angles of the inner arm, hip rotation, and shoulder straightness are displayed in the top left of each feed. This allows the user to reference their swing before going into the statistical analyses.

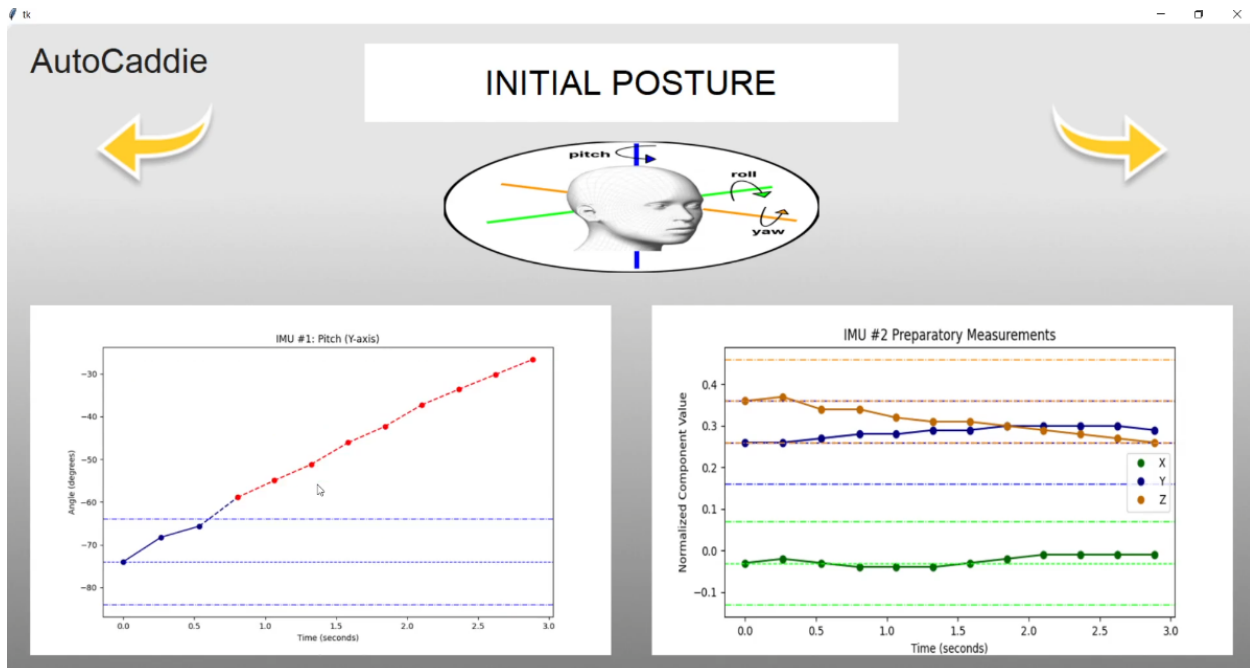


Figure 51: The IMU data feedback screen for AutoCaddie.

Figure 51 represents the IMU data measured during the preparation of the swing. This section contains plotted data of the quaternion data, as well as the roll, pitch, and yaw of each shoulder while setting up for a swing.

Importantly, on the graph on the right in this figure, some of the Euler data points measured are red. This signifies problematic points in time where the user's shoulders moved too far from its initial position. The IMUs measure movement over time, and it has been tested that an acceptable allowance of ± 10 degrees is satisfactory for measuring a proper swing. If the user rotates their shoulders by more than this while set and ready for a swing, then it is noticed by the system and highlighted. Similarly, for the quaternion data, a threshold of ± 0.1 normalized degrees are allowable. Euler angles are provided for more readability for the user.

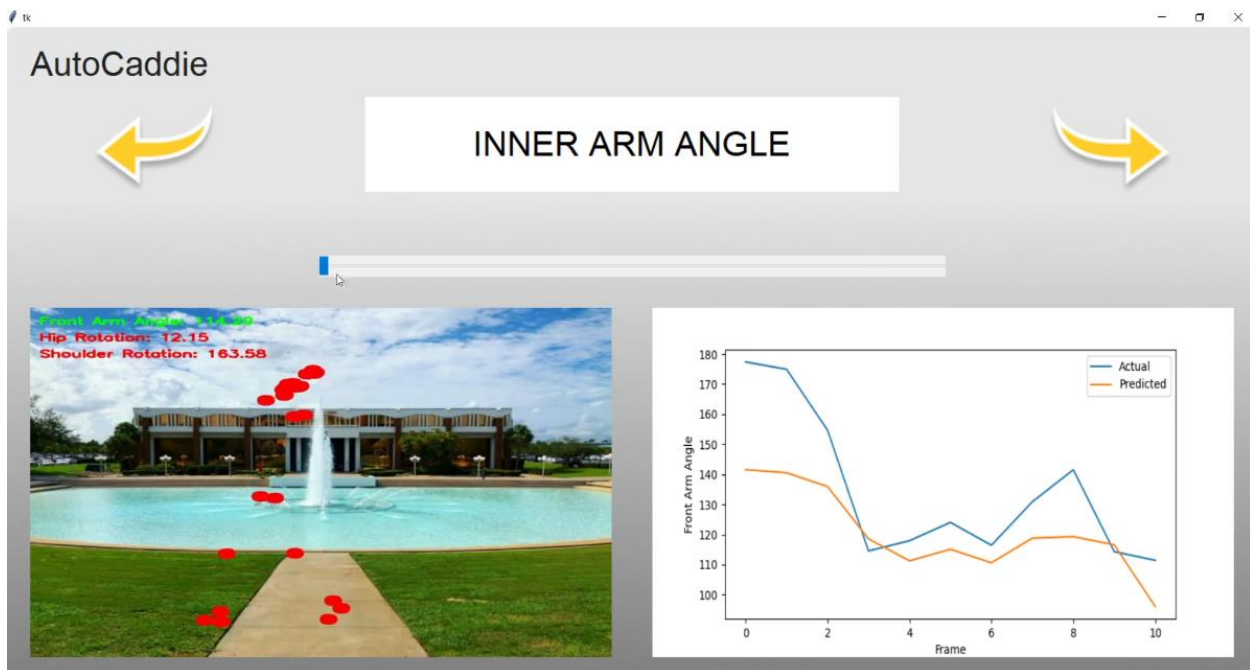


Figure 52: The inner arm angle analysis screen for AutoCaddie. This layout is identical for hip rotation and shoulder straightness analyses.

Figure 52 represents results that are presented in the same modality for inner arm, hip rotation, and shoulder straightness. On the left, a point cloud of the MediaPipe joint tracking is displayed to give a visual of the user's important information. This can also assist the user in determining if any joints or body parts were not captured by the camera, which would signal to switch to the other display.

The graph on the right represents the AI generated swing given the user's initial momentum, plotted against their actual values. For each metric, the graph only shows the generated results for itself. This figure is displaying the inner arm angle history.

Additionally, thresholds have been tested and determined for each parameter for allowable error. For inner arm angle, the threshold value is 20 degrees, while for shoulder straightness and hip rotation, the threshold value is 25 degrees. These values were found to be the most consistent for differentiating technically good swings with novice swings.

The text in the top right of the video feed will display red when the given metric is outside of the allowable threshold when comparing the user's actual swing to the generated swing. These are the instances in time which warrant attention.

By providing tabulated feedback for the user, they can either interpret the results themselves, or send these results to a coach to provide more tailored feedback. The goal of AutoCaddie is to provide the statistics for enhancing the fundamentals of the user, which is done through AI analysis and hardware measurements. This way, AutoCaddie can be deployable anywhere and assist the user in correcting their fundamentals.

7.9 System Integration and Optimization

7.9.1 Overall Integration

Overall, there is much to consider when integrating every subsystem into the overlaying software mechanism of AutoCaddie. It is important that this is performed seamlessly so that the system can perform without inconsistencies. At the scope of AutoCaddie's accuracy, it is imperative that there are minimal factors that can negatively impact the performance of the software, so that the user feedback can be as truthful as possible. Additionally, by providing successful integration, there will be efficient data flow, synchronization, and smoother operations throughout testing.

The components to be integrated are as follows. The two peripheral apparatuses, specifically the cameras and IMU sensors, should provide data in sync to the computing device. This device will contain the software of AutoCaddie, and will manage the data operations, as outlined in **Section 7.6**. Once the data has been received and run through both the computer vision preprocessing algorithms and the quaternion vectorization steps, the CNN and kinematic equations will compute. With the size of the data being manipulated, it is also important to emphasize optimizations within these algorithms to ensure the processes do not delay a final result. Then the results of these operations will be run through different analyses, where the output will be presented to the user. For AutoCaddie to align with its goals, it needs to be capable of real-time analysis. This stresses the importance of optimizing every internal algorithm to be as efficient as possible, while not jeopardizing overall accuracy.

Therefore, it is also important to highlight synchronization. For the user feedback to be comprehensive, certain statistics can reference both physical and video data at specific instances of time. Being able to manage an exact chronological order, which references specific instances of time, allows for AutoCaddie to perform a proper real-time breakdown of actions.

Lastly, the information presented to the user must also be comprehensive without jeopardizing processing speeds. It is important to ensure that the output does not facilitate long operations to generate graphics, extraneous information, or other expenditures. Thus, rigorous testing is a requirement before AutoCaddie can fulfill its goals.

Thus, the previous sections within this chapter discussed the implementation details. The following subsections explain optimizations and benefits for utilizing the design choices implemented.

7.9.2 Algorithmic Optimizations

7.9.2.a Modular Design Benefits

AutoCaddie embraces a modular and object-oriented approach to integrate each processing algorithm. This design philosophy is important for enhancing code maintainability and readability. The utilization of the tkinter framework for GUI

development complements this modular structure, ensuring a robust and efficient software architecture.

In adhering to a modular design, AutoCaddie has segregated algorithms, such as frame extraction, kinematic equations, and others, into discrete logic units. Each of these algorithms resides in its dedicated .py file, contributing to the clarity of code organization and simplifying the debugging and maintenance processes. This separation enables developers to focus on specific functionalities without the complexity of navigating through a monolithic codebase.

Additionally, AutoCaddie's software hierarchy is organized as a folder structure, providing a clear overview of the project's architecture:

- `algorithms/` folder: Contains Python scripts, where each individual script is dedicated to a specific algorithm.
- `init.py` file: Signifies the parent directory as a Python package, enabling clean imports between modules.

This hierarchical arrangement ensures that each algorithm operates as an independent and reusable unit. Moreover, this transitions to the utilization of unit testing to ensure functionality before deployment.

The integration of tkinter and the Python framework unittest directly supports unit testing within this modularized code structure. By referencing Figure 41, the project hierarchy places the controlling logic at the parent folder, with algorithms extracted down to a subdirectory. Similarly, the introduction of a 'tests' package showcases a dedicated space for unit testing each algorithm individually. Therefore, the integration process of unit testing, which ensures program functionality and promotes debugging convenience, is streamlined.

7.9.2.b Frame Extraction Benefits

Frame extraction is the first step in the image data preprocessing pipeline for AutoCaddie's CNN. This process involves breaking down a continuous video feed into distinct images, each representing a specific moment in time. The frames, represented as 2D matrices of pixels, provide the fundamental building blocks for subsequent image processing within the AutoCaddie system. To achieve this, the OpenCV library is used to parse frames from live video feeds. OpenCV's capabilities enable AutoCaddie to create the illusion of downloading and extracting a video by live parsing frames from a video feed.

Within the implementation, there are many benefits for AutoCaddie:

- **Real-time Processing:** By extracting frame data from live video streams, AutoCaddie can operate in real-time without worrying about employing a video parsing algorithm after receiving the camera data. This saves processing speed,

as there is no additional algorithm that needs to be used for this task after acquiring the data.

- **Data Precision:** The representation of frames as 2D matrices, where each matrix contains pixels representative of 3 color channels, ensures precise data to be maintained through subsequent image processing algorithms.
- **Synchronization:** Implemented error checking ensures that if one video does not properly capture an instance of time, neither frame is saved. This ensures continuity for the image processing, as both frame lists are representative of equal moments in time.
- **Memory Optimization:** By properly managing and releasing camera streams, there is less of a resource burden on the system.

7.9.2.c Image Standardization Benefits

Image standardization assumes the next step in the image data processing pipeline. Its objective is to ensure uniformity within input when presented to the CNN. This is an implicit process within AutoCaddie, as MediaPipe is thoroughly capable of detecting the user in different sized inputs so long as they are entirely contained within the frame of recording. Thus, it is imperative that the user ensures their entire body is captured during recording.

By utilizing image standardization, the following benefits are reached.

- **Enhanced Analysis Accuracy:** A consistent input format contributes to the accuracy of the CNN analysis. This standardization minimizes variability and ensures a reliable basis for identifying patterns and features.
- **Optimal CNN Model Compatibility:** The standardized images smoothly integrate with the CNN model's expectations. This compatibility is essential for maximizing the efficiency by reducing the variability of input and boosting the performance overall.

7.9.2.d Kinematics Benefits

As a separate but synchronous system within AutoCaddie, spatial data processing is computed through quaternion data points. Quaternion data serves as a valuable resource for storing the spatial information of the user's shoulders throughout the preparatory stage of the swing. Specifically, quaternion data can be represented by a 4-dimensional vector, which calls for the utilization of NumPy. In other words, the performance benefits of the kinematic equations running alongside the image processing data include boosted accuracy of calculations, as well as every mathematical optimization presented by the NumPy library. Formally, these benefits are as follows:

- **Precision in Spatial Representation:** By using quaternion data points over Euler angles, it is possible to track more dimensions of the input data without jeopardizing storage requirements. Quaternion data is highly accurate, and allows for linearity calculations to be simplified into a single vector equation.

- **Efficient Matrix Operations:** Leveraging NumPy for matrix operations enhances computational efficiency, enabling swift and optimized calculations of quaternion angle differences.
- **Dynamic Angle Calculation:** The ability to calculate angles between quaternion data points for every instance recorded during the swing provides a comprehensive perspective of each shoulder's spatial orientation.
- **Numerical Stability:** NumPy's numerical stability ensures reliable and accurate results, crucial for maintaining the integrity of spatial data processing in kinematic analysis.

It is worth mentioning that the implementation libraries for the image data preprocessing algorithms utilize NumPy's matrix operations. It has been established that each frame is a 2D collection of integers, so when operated upon by NumPy's functions, the analysis process is simplified.

Specifically, NumPy's standout feature in regards to image data and neural networks is its seamless integration with Singular Value Decomposition (SVD). SVD is a sophisticated technique utilized extensively in data analysis, signal processing, and, notably, in the image processing realm of AutoCaddie. With SVD, complex matrices are reduced into three concise matrices, capturing nuanced patterns and crucial relationships inherent in the data. A diagram of the SVD transformation of a matrix is shown below.

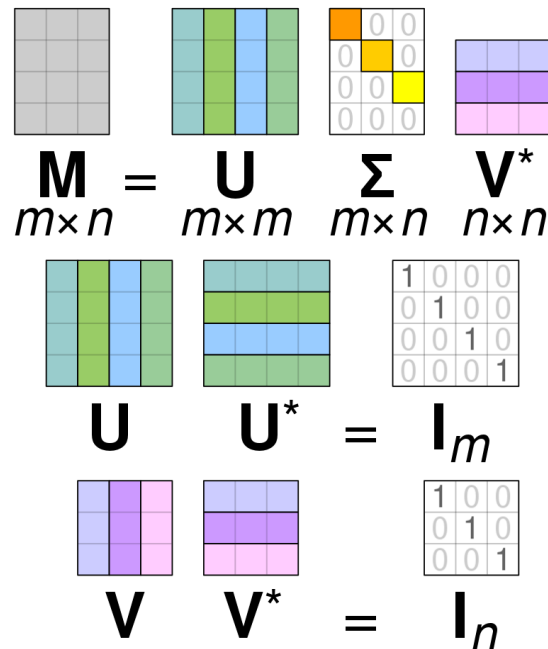


Figure 53: The Singular Value Decomposition of a Matrix. A matrix of dimensions $m \times n$ is reduced into three compact matrices. Matrix U contains orthogonal column vectors, and matrix V^* contains orthogonal row vectors. Matrix Σ is a diagonal matrix which contains the singular values of A . Many matrix operations can be completed using this reduced form of the original matrix A . Referenced via [86].

In the context of AutoCaddie, this translates into the processing of video data with strong precision. The live video feed, through the data preprocessing algorithms, can be dissected into a collection of frames, where certain segments map to exact actions within

the golf swing sequence. Moreover, each frame can be represented as a matrix of pixels. Through SVD, these matrices are simplified, shedding redundant information, and eliminating noise while preserving the vital aspects. In essence, this reduction in dimensionality streamlines the data, making it more manageable and refining its quality.

7.9.2.e Summary

Overall, each of the implementation strategies for the data processing algorithms maintain the line between performance and resource demands. In optimizing these algorithms, a modular design philosophy is embraced, enhancing code maintainability and readability. The software hierarchy, organized into a folder structure, promotes algorithm independence, and streamlined unit testing. Frame extraction employs OpenCV, offering real-time processing, data precision, synchronization, and memory optimization. Image standardization is implicit, and ensures consistent input dimensions for effective CNN analysis, enhancing accuracy and model compatibility. Kinematics, employing quaternion data and NumPy, ensures precise spatial representation with efficient matrix operations, dynamic angle calculation, and numerical stability, contributing to a comprehensive understanding of the user's posture consistency while preparing for a golf swing. Overall, these optimizations lead to a robust and efficient system for golf swing analysis.

7.10 Summary of Software Design

7.10.1 Introduction

AutoCaddie is a state-of-the-art system tailored for analyzing golf swings, integrating advanced machine learning models and software algorithms within a user-friendly interface. The system adeptly processes video and sensor data, providing real-time feedback to golfers. This feedback is crucial for players looking to enhance their skills, offering insights that would be challenging to obtain through traditional coaching methods. By leveraging the power of AI and machine learning, AutoCaddie stands at the forefront of technological innovation in sports training, offering a blend of accuracy, immediacy, and accessibility.

7.10.2 Machine Learning and AI Subsystems

AutoCaddie's machine learning model is a cornerstone of its functionality, utilizing the expansive GolfDB database for robust training. This neural network-driven approach enables the system to map out intricate details of golf swings, linking specific actions to relevant parameters. The model iteratively refines its predictions, reducing errors and fine-tuning its accuracy. Complementing this, the AI subsystem is divided into two integral components: a video system and a physical sensor system. The video system meticulously preprocesses and frames golf swings, extracting pivotal features for analysis. In parallel, the sensor system, equipped with IMUs, captures detailed quaternion data, enriching the swing analysis with a layer of depth and precision.

7.10.3 Software Implementation and Design

At the heart of AutoCaddie is its software, hosted on a computing device like a laptop, serving as the central hub for data processing and feedback dissemination. This software

is engineered to handle data inputs from various sources, applying a series of preprocessing algorithms before funneling the data into the machine learning model. The design of the software is centered around creating an AI-driven, intuitive system that can analyze golf swings in real-time. It integrates data from both IMUs and video cameras, processing it through advanced machine learning algorithms and presenting the analyzed feedback through an interactive GUI.

7.10.4 Technical Aspects of Software

Python is the backbone of AutoCaddie's software development, chosen for its simplicity, versatility, and extensive support for relevant libraries like NumPy, crucial for handling complex mathematical operations. The GUI, crafted using tkinter, is designed to be user-friendly and efficient, presenting data in a manner that is both informative and easy to interpret. Development is carried out in Visual Studio Code, a versatile IDE supporting Git integration for streamlined version control. AutoCaddie's commitment to robustness and reliability is underscored by its comprehensive testing regimen, which includes unit testing and integration testing, ensuring the software's performance and reliability.

7.10.5 Communication and Data Processing

The communication framework within AutoCaddie ensures efficient and accurate data transmission between the microcontroller unit (MCU), cameras, neural network, and other internal components. Data processing algorithms like frame extraction, image standardization, and kinematic equations are integrated into the system with a focus on modularity and maintainability. This modular approach not only facilitates ease of testing but also enhances the scalability and robustness of the software architecture.

7.10.6 Data Management and Analysis

AutoCaddie exhibits an adept strategy in managing and processing data from cameras and IMUs, emphasizing efficiency to enable real-time feedback. The system skillfully interprets neural network outputs and kinematic data, providing a comprehensive analysis of each golf swing. This integration of spatial-temporal data analysis ensures a detailed and holistic understanding of the swing mechanics, offering valuable insights to the user.

7.10.7 GUI and System Integration

The GUI, based on tkinter, is meticulously structured to offer a seamless and intuitive user experience. It effectively displays processed data, enhancing the user's understanding of their swing technique. The integration of the system is a testament to AutoCaddie's commitment to accuracy and efficiency. Algorithmic optimizations across different components ensure a balance between performance and resource demands, augmenting the overall effectiveness of the system.

7.10.8 Optimization and Validation

Optimizing the neural network is a critical task, balancing model complexity with operational efficiency to enable real-time analysis. The system undergoes regular updates and refinements, based on a comprehensive validation process that includes cross-validation techniques and real-world testing. These processes ensure the model's effectiveness and reliability in practical scenarios, making it a dependable tool for golfers.

7.10.9 Conclusion

In conclusion, AutoCaddie represents a groundbreaking solution in the realm of golf swing analysis. Its fusion of advanced machine learning, sophisticated software engineering, and user-centered design results in a system that not only excels technologically but is also practical and user-friendly. AutoCaddie stands out as an exemplary innovation in sports technology, offering a valuable tool for golfers of various skill levels to enhance their performance.

Chapter 8: CAD Design and Fabrication

8.2.1 PCB Board Layout

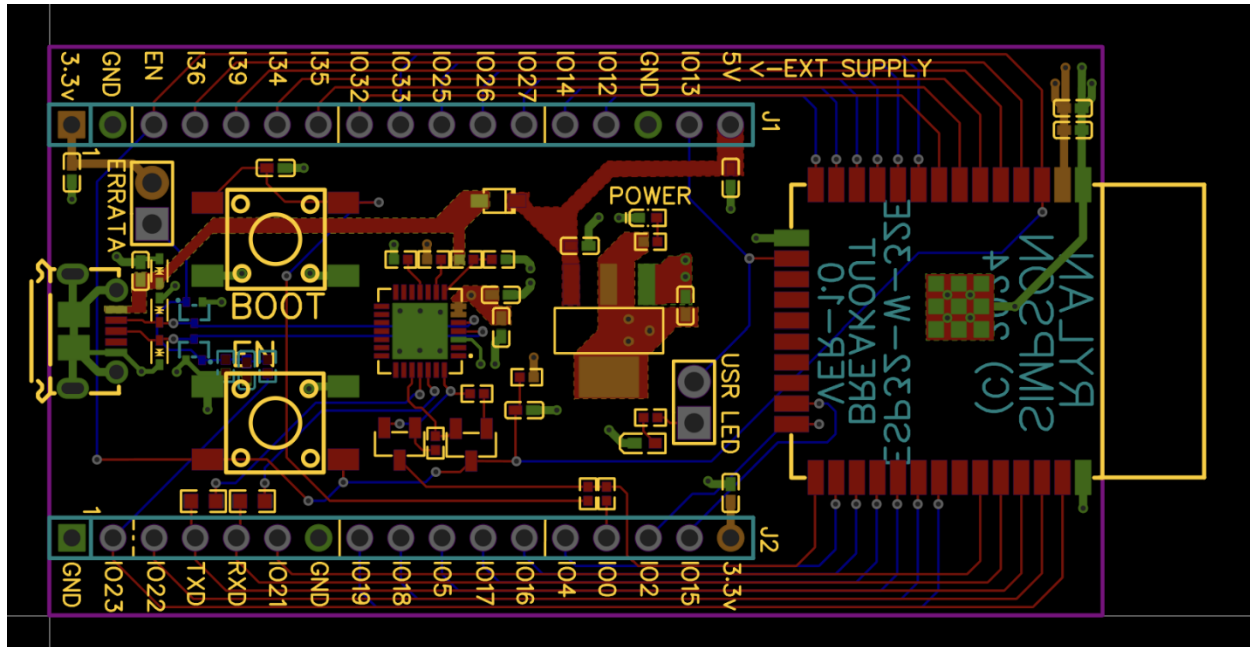


Figure 54: Programming Board Layout

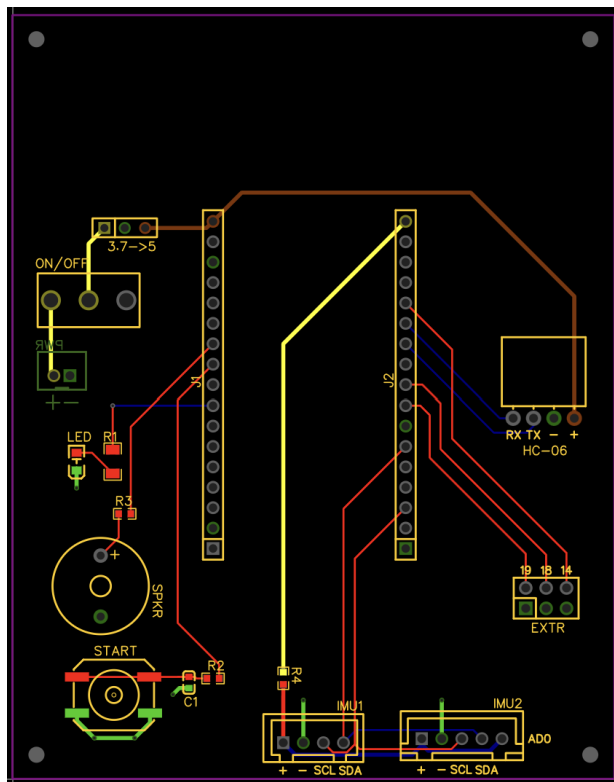


Figure 55: Backplane board layout

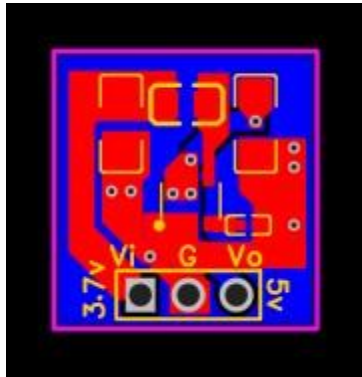


Figure 56: 3.7v to 5v regulator board layout

Each board has been designed in the web-based CAD software EasyEDA. This software allows for simple integration of parts from supported libraries and catalogs, integrating both the schematic symbols and circuit footprints. Given that supported parts provide the symbols and footprints, easy bill of material and pick-and-place file generation can be performed, allowing for the request of PCB assembly services.

The programming board was designed to follow recommended design patterns and layouts, including that of power planning and JLC04161H-3313A impedance control. These design formats ensure reliable data transfer and stable power railing.

The backplane board was designed as a simple interface between the programming board and peripheral hardware. The board was designed to be larger than the dimensions of the 6.6Ah battery that will be used to power the board (~70mm x ~55mm). Further, enough space must be provided to the HC-06 module as to not interfere with its profile or wireless signals.

Finally, the power regulator board was designed to match the recommended board layout for the AP regulator. This design seeks to reduce any noise or parasitic effects of the power load or power supply, and seeks to generate a stable power output signal.

8.2.2 PCB Fabrication

The central PCBs will be ordered and manufactured through JLPCB, following the specifications and design requirements. Post-acquisition, the assembly process entails soldering remaining components such as lower connectors onto the PCB. Surface mount components may undergo soldering using a reflow oven, ensuring precise and uniform attachment. The other components that use through holes can be hand-soldered onto the PCB. This approach in component attachment ensures the integrity and reliability of the central PCB, aligning with industry standards and best practices in electronic manufacturing.

8.3 Kinematics system and hardware

8.3.1 PCB and IMU enclosures

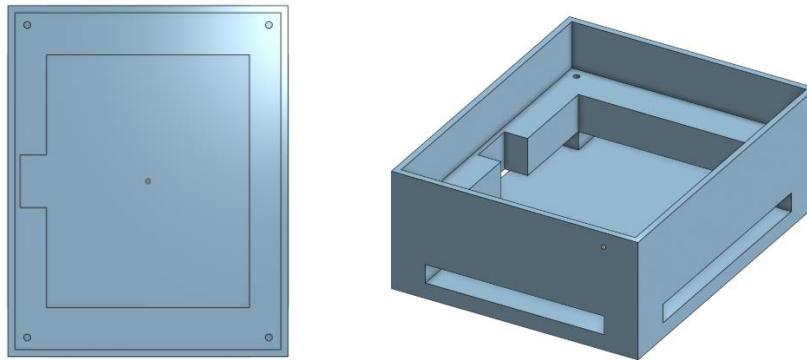


Figure 57: (LEFT) PCB mounting box; Top view. (RIGHT) PCB Mounting Box; Isometric view.

Designing enclosures for both the central PCB and the IMUs is important for ensuring the protection and stability of these hardware components, particularly as they are wearable devices. The enclosures serve a dual purpose: shielding the components from potential physical impact, and providing a secure means of attachment to the wearer. The enclosures are designed as CAD models, where they can be 3D printed using PLA plastics, offering a decent balance between cost effectiveness, strength, and flexibility.

To facilitate the attachment of these enclosures onto a wearer, a practical approach involves incorporating Velcro straps into the design. These straps can be fed through designated slots or loops on the enclosures, allowing for adjustable and secure fastening around the user's body. To secure the IMUs and PCB within the enclosures, the components are securely fastened using screws. The design includes through holes placed on the PCB and IMU, which align with corresponding points on the enclosure. The enclosure is designed to allow the screws to cut their own threads as they are tightened. The points at which the screws engage with the enclosure have holes which diameters are slightly smaller than the diameters of the screws. As such, as the screws are tightened into the holes, they cut into the PLA, creating their own threads. This self-threading mechanism ensures a robust connection, preventing the components from shifting during movement while maintaining the structural integrity of the wearable device.

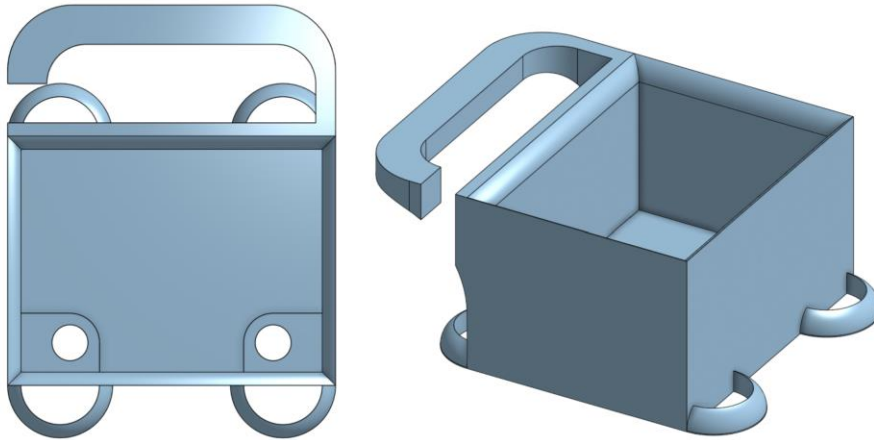


Figure 58: (LEFT) IMU mounting box; Top view. (RIGHT) IMU Mounting Box; Isometric view.

The IMUs will have their own enclosure mounts, designed such that the IMUs are securely placed within the interior of the open box. Screw holes can be used to immobilize the IMUs with respect to the enclosure. Furthermore, an integrated hook on the enclosure can be used to lock in the wires connected to the IMU devices, such that they do not pull away directly from the connectors, and are thus more secure.

8.3.2 Enclosure box mounting

As the IMU and MCU PCB units are worn by the user during the use of AutoCaddie, the enclosure boxes must be secured to the user. As mentioned in the prior section, Velcro straps can be used to secure the enclosures to the user. With the central PCB for the MCU, two Velcro straps can be fed through the enclosure. These Velcro straps can be used as a form of “belt”, where the enclosure box should be secured on the user’s waist.

The IMU enclosures will also utilize Velcro straps to secure the IMUs in a similar manner to that of the central PCB. The IMU mounts can be affixed to the user via means such as Velcro pads. A pair of suspenders with Velcro regions on the shoulders can be supplied to the user, such that the enclosures with Velcro on their bottoms can be easily placed and removed at the correct locations, depending on the height of the user. On the user, keeping them secure, with some slack to account for user movement. Further utilizing the suspenders, sheathed wires extending from the IMUs can be tied down to prevent wires from moving around erratically.



Figure 59: Example of elastic suspenders that can be supplied to the user for IMU mounting

Chapter 9: Testing Plan

9.1 Hardware Component Test and Verification

Hardware testing for the AutoCaddie architecture is characterized by constant referencing and comparing of component documentation as well as deliberation on the topic of best practices of IMU calibration, wireless transmission behaviors, and MCU component control. Below is a detailed description of how each component will be tested and verified:

9.1.1 MCU Wiring Verification

As the design of the MCU's programming board is independent from the backboard containing the peripherals, the board and architectures similar to the board can be used in conjunction with development materials such as breadboards to validate the designs of the hardware systems and programming of the MCU. Should all validations of the hardware connections and the MCU programming succeed, further development of the hardware systems can proceed

9.1.2 IMU Component Testing

The Inertial Measurement Units (IMUs) are the focal point which differentiates AutoCaddie from other golf-related machine learning projects. Using these hardware components, real-time hard data can be given to the user which details the straightness of their arm. Given the correlation between the straightness of one's arm and the quality of one's golf swing, this data will be used to vastly improve the form of one's golf swing.

The IMUs will be tested using a breadboards and development boards using the Arduino platform. After the necessary library integrations, the MCU program will be used to interface with the IMUs via I2C communications, and will use the MPU6050 libraries to calibrate the IMUs and initialize the DMP algorithmic processes, as well as to collect and display the relevant data. Should the I2C communication succeed and the data be displayed, the correctness of the data can be confirmed using a tool such as a goniometer, which is used to measure angles of rotation between two vectors. Should the correctness of the IMUs be confirmed, the IMU integration design can be validated.

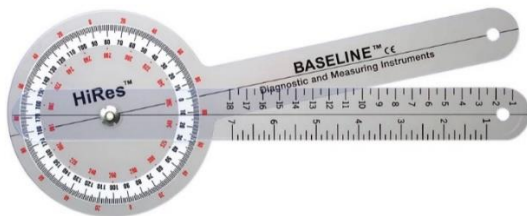


Figure 60: An example of a goniometer to be used in IMU angle testing.

9.1.3 Wireless Transceiver Testing

The HC-06 Bluetooth Wireless Transceiver is the juncture which connects hardware to software on the AutoCaddie architecture. The data collected from the IMUs will be sent along wires to the MCU, where the data will be formatted for transmission. Finally, the

MCU will send the formatted data to the HC-06 transceiver, where the paired device will receive a stream of data from the hardware system.

The HC-06 module will be tested using breadboard materials and a development boards. Once the device is connected and powered, it will be set into a pairing mode, where it can be paired with a separate Bluetooth compatible device. This device can open a Bluetooth Serial terminal, where it can receive any data sent by the HC-06. Sample will be sent to the HC-06 for transmission, and during the reception of the data, the validity and robustness of the Bluetooth connection schema via the HC-06 Bluetooth module can be determined.

9.2 Neural Network Testing

9.2.1 Neural Network Optimization

Optimizing the neural network for the AutoCaddie project involves a comprehensive approach, encompassing various aspects from architecture design to real-time performance enhancement. Here's a detailed look at each key strategy:

The architecture of our neural network is fundamental to its success. For the complex task of analyzing golf swings, a hybrid architecture that marries the spatial analytical prowess of Convolutional Neural Networks (CNNs) with the temporal sequencing capabilities of Recurrent Neural Networks (RNNs) is recommended. CNNs can adeptly process and interpret the visual aspects of the golf swing, such as the position and movement of the golfer and club. On the other hand, RNNs, especially those with Long Short-Term Memory (LSTM) units, are well-suited for understanding the sequential nature of the swing, capturing nuances over time. This hybrid model aims to thoroughly analyze each phase of the swing, ensuring no critical component is overlooked.

Achieving a balance between the model's complexity and its operational efficiency is a delicate task. A highly complex model, while potentially more accurate, may require substantial computational resources, rendering it impractical for real-time analysis, which is crucial for immediate feedback during training sessions. Techniques like model pruning, which involves trimming non-critical neurons; quantization, reducing the precision of the numerical parameters; and leveraging depth-wise separable convolutions, a more efficient form of convolution operation, can significantly enhance computational efficiency. These approaches aim to retain the model's accuracy while ensuring it remains lightweight and agile for real-time processing.

The input data's quality directly influences the model's performance. Effective preprocessing techniques are critical. These include frame normalization to maintain consistency across different lighting conditions and videos, alignment to ensure a standardized input, and background subtraction to reduce noise and focus the model on the golfer's movements. Additionally, augmenting the data by introducing variations like

random cropping, rotations, and horizontal flipping can robustly train the model to recognize swings under various conditions and viewpoints, enhancing its ability to generalize.

Tuning hyperparameters is a crucial step in optimizing the neural network. Hyperparameters such as the learning rate determine how quickly or slowly the model learns; the batch size affects the memory usage and speed of training; and the configuration of layers and neurons impacts the model's ability to capture complex patterns. Employing methods like grid search, random search, or Bayesian optimization can systematically explore the hyperparameter space to find the most effective combination for our model, balancing learning efficiency with predictive accuracy.

Regularization techniques are essential in preventing the model from overfitting, particularly vital due to the possibly limited size of specialized golf swing datasets. Techniques like dropout, which randomly disables neurons during training, and L2 regularization, which penalizes large weights, help the model generalize better to unseen data. Early stopping, where training is halted once the model's performance on a validation set ceases to improve, also prevents overfitting. These methods ensure that the model learns the general patterns in the data, rather than memorizing the training set.

The loss function guides the model's learning process. In our context, a cross-entropy loss function is fitting as it effectively handles classification tasks, like identifying different phases in a golf swing. Complementing the loss function, evaluation metrics like accuracy, precision, recall, and the F1 score offer a multifaceted view of the model's performance. These metrics help assess not just the model's overall accuracy, but also its ability to balance false positives and false negatives, providing a nuanced understanding of its predictive capabilities.

Transfer learning can be an invaluable strategy given the challenges of acquiring extensive labeled golf swing data. By pre-training the model on a larger, related dataset and then fine-tuning it on our specific golf swing data, we can jump-start the learning process, benefiting from learned patterns that can be applied to our task. This approach can lead to significant improvements in model performance, especially in terms of learning speed and final accuracy.

For applications requiring real-time analysis, like swing feedback during practice, optimizing the model for quick inference is crucial. Techniques such as model quantization, which reduces the model's precision, and knowledge distillation, where a smaller, faster model is trained to replicate the behavior of a larger, more accurate one, can be explored. These techniques aim to reduce the computational load, enabling the model to analyze swings swiftly without substantial loss in accuracy, a critical factor for real-time applications.

The following Python code snippet exemplifies the approach discussed. It showcases a hybrid architecture where time-distributed CNN layers are employed to extract and analyze spatial features from each frame of the golf swing video. These features are then sequentially processed by LSTM layers, capturing the temporal dynamics and transitions of the golf swing. This combination aims to harness the strengths of both CNNs and LSTMs, making the model adept at understanding the intricate spatial-temporal patterns of golf swings. The code is a representative example of how such a hybrid model could be structured, providing a foundation that can be further tailored and refined to meet the specific demands of our golf swing analysis task. [A36]

In conclusion, optimizing our neural network encompasses a comprehensive strategy involving architectural decisions, data processing, computational efficiency, and ongoing learning and adaptation. These strategies are designed to develop a model that is not only robust and accurate but also efficient and adaptable, aligning with the specific challenges and demands of golf swing analysis in the AutoCaddie project.

9.2.2 Neural Network Validation

The process of validating our neural network in the AutoCaddie project is as crucial as its development and optimization. Validation involves a comprehensive set of methodologies and practices designed to rigorously assess the performance and reliability of the model under varied conditions, ensuring its effectiveness in real-world golf swing analysis scenarios. Here is an example of how you would prepare a validation set. [A37]

The foundation of a robust validation process is the preparation of an extensive and representative validation dataset. This dataset, distinct from the training data, should encompass a broad spectrum of golf swings, capturing variations in player techniques, environmental conditions, and camera angles. A diverse dataset is instrumental in evaluating the model's generalization capabilities, ensuring that it performs consistently across different scenarios and player styles.

Incorporating cross-validation techniques, such as k-fold cross-validation, is pivotal in thoroughly testing the model's performance. This approach, where the dataset is split into several subsets with the model trained and validated on different combinations of these subsets, offers a detailed and holistic view of the model's performance. It helps identify any specific data segments where the model might underperform, allowing for targeted improvements. An example of what k-fold cross-validation would look like is as follows. [A38]

Analyzing key performance metrics is an integral part of the validation process. Metrics such as accuracy, precision, recall, F1-score, and confusion matrices provide a multi-dimensional view of the model's performance. While accuracy gives an overall effectiveness measure, precision and recall focus on the model's capability to correctly classify specific events in the golf swing. The F1-score harmonizes these two metrics,

offering a balanced view, and confusion matrices present a detailed breakdown of the model's performance across all categories. Below is an example of what the code could look like to calculate key performance metrics. [A39]

Conducting a thorough error analysis is essential to understand where and why the model may falter. By scrutinizing misclassifications and errors, we can gain insights into the model's limitations and areas needing refinement. This analysis is crucial for iterative improvements and for fine-tuning the model to better handle challenging scenarios. Below is an example of calculating error. [A40]

Testing the model in real-world conditions is critical to ensure its practical applicability. This step involves evaluating the model with live or recorded golf swings in various uncontrolled environments. Such real-world testing is vital to confirm the model's reliability and effectiveness outside laboratory conditions.

Sensitivity and specificity analysis further enriches our validation process. These metrics are particularly important in tasks like golf swing analysis where false positives and false negatives carry significant implications. They help us understand the model's effectiveness in correctly identifying true positives (actual events) and true negatives (non-events), ensuring reliability in its practical applications.

Finally, user testing with players offers invaluable insights. Feedback from end-users about the model's utility, user experience, and practical applicability in real-world settings is critical. This feedback not only validates the model's technical performance but also its usability and effectiveness in meeting the users' actual needs.

In summary, validating our neural network model is a comprehensive and continuous process that encompasses a wide range of methodologies, from technical performance assessments to practical usability checks. Through rigorous and multifaceted validation, we aim to ensure that our model is not just a technological achievement but a reliable, effective, and user-friendly tool for golf swing analysis.

9.3 Software Integration Testing

9.3.1 Introduction

The algorithmic structure of AutoCaddie showcases a modular design philosophy. It fosters an organized and scalable approach to software development, as showcased by **Chapter 7**. Each processing algorithm within AutoCaddie is encapsulated within its dedicated logical unit, which promotes maintainability and reusability. This is particularly advantageous for debugging and maintenance by allowing developers to concentrate on specific functionalities without having to navigate through a complex, monolithic codebase. The software hierarchy is represented as a folder structure, where different logical groupings are separated into modules.

tkinter is utilized within AutoCaddie for the development of the project's GUI. The cross-platform compatibility and simplistic design abilities of tkinter align with the software's modular design. The organizational overview of AutoCaddie's software hierarchy, represented as a folder structure, ensures that each algorithm functions as an independent and reusable unit, allowing for streamlined testing.

9.3.2 Unit Testing with tkinter and Python unittest

Overall, unit testing is an important facet of software development. The process of unit testing ensures the reliability and functionality of individual components within a system. This testing method involves subjecting discrete units of code to different tests to ascertain their conformity to expected behavior. In the context of AutoCaddie, where data processing algorithms form the backbone of the system, robust unit testing facilitates maintainability.

The implementation strategy for unit testing in AutoCaddie revolves around the modularized code structure. The software allows for separate pieces of functionality to operate as their own files. This allows for the creation of a dedicated 'tests' package within the project, containing individual testing modules corresponding to each algorithm. An example of this is shown in the figure below. By utilizing unittest's library, a testing Python file can be created to analyze the functionality of a specific algorithm.

```
AutoCaddie/  
|-- main.py  
|-- algorithms/  
|   |-- __init__.py  
|   |-- frame_extraction.py  
|-- tests/  
|   |-- __init__.py  
|   |-- test_frame_extraction.py
```

Figure 61: An example of implementing a testing Python file for the frame extraction algorithm. By using the naming convention "test_[FILE NAME].py", wxPython recognizes this as a testing file.

The Python unittest module integrates smoothly within AutoCaddie, allowing for an efficient means of conducting unit tests. The integration is straightforward, with testing modules referencing their corresponding algorithms through import statements. In Figure 61, the frame extraction algorithm, named as "frame_extraction.py," is tested by "test_frame_extraction.py." Similarly, the naming convention of testing files follows this convention: "test_[FILE NAME].py." This direct integration allows developers to execute highly specific tests and analyze overall precision and functionality.

The unittest module is setup directly within the project resources. It allows for direct testing during development of the project and ensure that the data presented by the GUI is accurate before deployment.

For every testing file, there are criteria that need to be ensured to be accurate. The following list comprehensively covers the structure of each testing file within the "testing" package in AutoCaddie.

1. Is the input data collected as expected?
 - 1.1. This serves as a prerequisite check for the parameters of interest for the algorithm. Each data processing method requires a specific combination of input data, an example being that the zero-padding algorithm expects an image of lesser or equal to dimensions of the GolfDB training data.
 - 1.2. This test checks the input data, and assesses the format in respect to the expected format.
2. Does the function handle empty data?
 - 2.1. In every live system, there is the expectation that empty data is collected during idle time or errors. Therefore, it is the responsibility of each program to ensure that the data being used is not empty to not forward an invalid data point.
3. Is the initialization process functional?
 - 3.1. For objects or variables needed for the algorithm, does the Python file set them up correctly?
4. Does the algorithm provide the correct functionality?
 - 4.1. This stage consists of several parts. These tests are as follows:
 - 4.1.1. For a normal data point, is the outcome the same as expected?
 - 4.1.2. For an extreme data point (outlier), does the algorithm still work as expected?
 - 4.1.3. For data points that are expected to produce errors, does the algorithm assess invalid data?
5. Does the algorithm invoke correctly?
 - 5.1. When calling the algorithm, ensure the correct function is referenced when given specific parameters.
6. How performant is the algorithm?
 - 6.1. During execution, it is important to analyze the execution time of each algorithm. This is imperative to ensure real-time data analysis within AutoCaddie.
 - 6.2. First, the execution time of the algorithm needs to be observed. Using manual analysis or Python benchmarking, determine if the execution time is acceptable.
 - 6.3. Second, ensure that the memory and resource consumption is not overbearing on the system.
7. Does the algorithm perform cleanup correctly?
 - 7.1. If the algorithm involves resource allocation, ensure that it is properly released after execution.
 - 7.2. Ensure that the algorithm is cyclic, or that all conditions return to their original state (unless static calling is used).

For each of the data processing algorithms involved in AutoCaddie, each of these tests will be executed. Overall, the testing process ensures correct data input, handling of empty data scenarios, and accurate initialization of objects. Functionality tests cover normal and extreme data points, error-handling capabilities, and proper algorithmic invocation. Performance assessments focus on execution time and resource usage for

real-time data analysis. The evaluation also ensures correct cleanup processes, emphasizing resource release and the return to original conditions, crucial for algorithmic cyclic operations.

9.4 Summary of Testing Plan

The above was an in-depth analysis of how each of the core components of the AutoCaddie project will function and be implemented. The software components, those being the machine learning neural network and the GUI to be designed, and the hardware components, those being the IMUs, MCU, and transceiver as well as their designed enclosures, were all thoroughly explored. The testing plans for each of these components are laid out in their entirety, defining how the minutiae of the project will be individually verified. As the testing plans are meant to ensure that all components work with their wiring orientation, enclosures, and code, this necessitates that the forms that the hardware currently takes are highly subject to change in the future, as well as the software components. Going forward, the operation of the project from a user's view will be explored, detailing exactly how AutoCaddie will feel and be used in the hands of an aspiring golf master.

Chapter 10: Justification for Usage of Laptop as Central Processing Device

The choice of hardware for this project can significantly impact the effectiveness and accessibility of a solution. In this section, the usage of a personal computer will be compared to that of an external processor with a tethered display. Through an examination of factors such as storage, cost considerations, and processing capabilities, we aim to select the optimal choice for this specific application.

10.1 Selection of Processing Device

To ensure that the best external processor is selected, it must consider cost, memory, and processing capabilities. Below is a table comparing the best selections for this project.

	Jetson Nano	Google Coral	Khadas Vim3	Orange Pi 5
GPU	128-core Maxwell	Integrated GC7000 Lite Graphics	5.0 TOPS NPU	Integrated ARM Mali-G610
CPU	Quad-core ARM A57 @ 1.43 GHz	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)	Amlogic A311D - x4 2.2Ghz Cortex A73, x2 1.8Ghz Cortex A53	4-core Cortex-A76 and 4-core Cortex-A55
Memory	4 GB 64-bit LPDDR4 25.6 GB/s	4 GB LPDDR4	4 GB LPDDR4	16 GB
Storage	microSD	microSD	32 GB	16 GB
Connectors	Ethernet, HDMI, display port, USB	Bluetooth 4.2, Wi-Fi 2x2 MIMO, USB, HDMI, Ethernet	Wi-Fi, Bluetooth 5.0, USB, Ethernet, HDMI	USB, HDMI, DP1.4, Ethernet,
Power Consumption	5V DC, 4A	5V DC	5V DC	5V DC, 4A
Cost	\$150	\$170	\$150	\$138

Table 11: Comparison of external computer chips to operate AutoCaddie's software.

This table accounts for a multitude of comparisons. The Jetson Nano, Google Cora, Khadas Vim3, and Orange Pi 5 were selected due to their capabilities and scope for this project. Each of these boards were researched in terms of their GPU, CPU, memory limit, storage requirement, valid connections, power consumption, and cost.

First, the GPUs and CPUs from each dev board are satisfactory for this project. The only downside is the presence of single processors on each board, limiting the stream of

processing to one thread. This gives the Orange Pi 5 the overall advantage in terms of memory and storage requirements.

Second, the valid connection types each board can support are listed. For this project, the important ports are USB for camera recording, Bluetooth for packet data receiving, and power. The power consumption comparison did not provide extra information, as each board utilizes the same power consumption amount. However, the Google Coral and Khadas Vim3 come with Bluetooth already built into the chip. For the Jetson Nano and the Orange Pi 5, an external serial Bluetooth chip is required.

Ultimately, since each board is similar in price, as well as processing capabilities, the debate stems down to which board practically fits the scope of the project the best. Due to previous experiments with MediaPipe, as well as large community support and a reputable company, the Jetson Nano fits the criteria of this project the best. With 4 GB of memory, as well as very powerful CPUs and GPUs, this board stands above the others.

10.2 Storage Considerations

The storage capacity of the Jetson Nano is a critical aspect, particularly in scenarios involving video data processing. Calculating the storage requirements, as referenced in Chapter 7, for video data captured at HP 320 FHD cameras at 30fps with 1080p resolution for three seconds yields 4 MB required.

Additionally, MediaPipe will apply image manipulation to map joints onto the user, which doubles this required storage space to 14 MBs. Not to mention, the IMU packet transmission sees timestamps stored as floats, as well as two data streams sending 4 floats each five times per second, which adds 900 bytes to this total. To account for the data receiving storage requirements, a projected total of 15 MB is necessary.

Moreover, the GUI program, developed through Tkinter, contains assets and widgets that can quickly raise the total storage requirements. Storing the asset data in a preliminary GUI design requires 30 MB. This number will rise as tables and graphs are created based on the user's swing. This does not include the Python scripts to be created for data processing, analysis, program functionality, and unit testing. Not to mention, the GUI will store additional videos from the GolfDB database for user reference. For the sake of simplicity, the current testing plan has four videos to be stored, which will require 1.2 MBs of storage.

Additionally, MediaPipe requires 50.7 MBs to install on a device with a Raspberry OS 64-bit [87] It also requires Python v3.8+, which needs 97.2 MBs [88] . Putting all of this together so far, there is a required 194.1 MBs of storage required. With the completion of the AutoCaddie GUI, as well as generated tables and graphics for the user, it is safe to allocate a maximum of 1 GB of storage to contain the system. This includes the AI system, which is run using a deep neural network. MobileNetV2, which was used for SwingNet's analysis of the golf swing with high accuracy [1] uses 14 MB of storage, which is a lower threshold [48] The developed model will not call for anything that feeds the software system to surpass 1 GB of storage.

Therefore, it is reasonable, purely based on storage requirements, to use the Jetson Nano for this project. To reserve space for 1 GB of data, it needs to be contained within a microSD card. This introduces additional problems, which will be explained in later sections.

10.3 Cost Analysis

It has been recently confirmed that it is reasonable to use a Jetson Nano in terms of storage requirements for this project. However, this brings on another topic for debate: how much additional cost to the user does using this hardware system introduce? The required parts for this hardware system are as follows.

The Jetson Nano uses a HDMI connector to provide display information. Therefore, a HDMI cable is required. To finish this connection, a decent display is required. This can be fulfilled using a 7 inch LCD display. Separately, the Jetson Nano needs to connect with the MCU attached to the user, meaning a Bluetooth connection is required. The board itself does not have a Bluetooth card, so an external serial Bluetooth adapter is required. Additionally, a microSD card is needed to store the project. The Jetson Nano needs a power supply to operate – specifically one with 5V/4A power. Lastly, a case for the Jetson Nano needs to be created, and additional housing for the power supply is not required but can be created using 3D modeling. A table breaking down the best determined part costs is shown below.

Part	Price	Link
Jetson Nano	\$150	[Click Here]
HDMI Cable	\$7	[Click Here]
7-Inch LCD Display	\$50	[Click Here]
Serial Bluetooth Adapter	\$25	[Click Here]
MicroSD (12 GB)	\$10	[Click Here]
Power Supply (5V/4A)	\$30 + \$5 for AA batteries	[Click Here]
Case	\$18	[Click Here]
Total	\$295	

Table 12: A cost-analysis table for the required parts to implement the Jetson Nano into AutoCaddie.

As broken down by the administrative information, the total for this project outside of this hardware suggestion is \$200.76. This does not include any coverage for wire sheathing, on-user applications (Velcro, pins), or taxes. Our estimation for the total price of the extra material is \$40 at maximum. Applying a 6% (minimum) sales tax onto everything, the total price is inflated to \$255.21.

In the project goals, it was decided that the absolute maximum price limit for this project is \$500. This is in account to providing the user with an affordable system. AutoCaddie aims to be a supplement to golf instruction, but is critical for new players seeking to improve a basic golf swing. Thus, for novices, coaching instruction can range anywhere

from \$50-\$150 per hour. At the best case, practicing three hours a week at the minimum \$50 price puts a monthly total of \$600. Therefore, AutoCaddie can make up the cost of coaching to the user within one month.

The required pieces of equipment costs \$395.41, as tabled in Chapter 12. Adding the \$295 price of the Jetson Nano system to the project far exceeds the upper limit of \$500. Therefore, it does not follow our project's goals to use this hardware system in terms of additional cost.

That being said, AutoCaddie suggests a personal computer, as the modern consumer already has possession of a laptop, which only puts the requirement on acquiring the IMU system and the software.

10.4 Processing Capabilities

The Jetson Nano presents 4 GB of memory to be used. It also utilizes one processor, meaning that all operations must be completed in a single thread by the same unit. Thus, it is imperative to understand the maximum strain of memory that AutoCaddie utilizes at once to determine if this idea is possible.

MediaPipe operates at a device minimum of 3.8 GBs [89] This already is almost exceeding the maximum limit given by the device. However, by testing locally, the conclusion is also made that running computer vision data recording, with the Bluetooth idle receiving script running and the GUI home menu – which is the least memory intensive screen in the GUI – requires approximately 500 MBs for a five second duration. For testing purposes and to ensure reliability, 1 GB of memory should be allotted at all times for the software. This puts an active total of 4.8 GB of memory required during data recording. This does not account for the processing delay that introducing the Jetson Nano into the project would cause.

Additionally, projects have been completed using MobileNetV2 through a Jetson Nano. Users have reported that using this CNN model, which was utilized by SwingNet, operates with a custom model at 1.719 GB of memory [90] This fact has been recognized by Jetson developers through their forum. This puts the total active memory required to around 6 GB.

Moreover, we cannot raise the memory limit of the Nano to a different model, as the price maximum has already been exceeded. Thus, it is not practical to use this hardware selection for processing purposes during AutoCaddie's data recording and analysis.

10.5 Final Justifications

Due to the cost and processing analysis, it is already unfit to use the Jetson Nano for the AutoCaddie project. Plus, since this processor was selected as the best-fit board for the project, it does not follow suite to utilize this hardware setup. Some additional justifications are as follows.

Firstly, if the Jetson Nano were to be used for this project, it would be necessary to run it at full capacity at all times. It would also call for a different software flow, as MediaPipe's analysis cannot run at the same time as any other process, and balancing the CNN would require careful timing as well. But, running the device at full capacity during most of its lifecycle in AutoCaddie is not good practice, as it can cause heat damage to the device itself and impact its longevity. If the Jetson Nano ever reaches its end-of-life phase, then having this design impact overall longevity is not good practice.

Second, the Jetson Nano setup would require the usage of a microSD card to store the software. This is not good practice, as running the entire software loop through external storage mainly will create higher rates of memory failure, which jeopardizes the data analysis accuracy. AutoCaddie puts a large emphasis on accuracy of data, so this goes against its goals.

Thirdly, it is common knowledge that a significantly larger audience has access to a laptop than a Jetson Nano. Therefore, in terms of the additional cost required for the user, they can save money by utilizing a personal computer rather than this hardware setup. Moreover, it is more convenient of a setup, as setting up an additional display with more tethered wires can present more areas of mistake. A large focus of AutoCaddie is to be convenient, so having the additional hardware system might present itself as too complex whereas a laptop system with webcams and the personal hardware setup is simpler.

Fourthly, by using the Jetson Nano, we'd be running AutoCaddie's software and data analysis processes as minimum processing power. This is necessary to make it even possible to run the software in the first place. Overall, this will make the system more inconvenient, as laptops are inherently stronger than the Jetson Nano.

Lastly, users have reported running MediaPipe with the Jetson Nano, and only reaching framerates of 24 fps on 30 fps devices [91] This falls below the minimum required data recording rate outlined in the project's goals.

Overall, it does not follow good practice to utilize the Jetson Nano, or any external processor for this project. By balancing convenience with cost, having the user run AutoCaddie through a laptop or personal computer aligns significantly better with the project's goals.

Chapter 11: Conclusion

AutoCaddie stands as a significant advancement in the field of golf swing analysis, combining innovative machine learning techniques with practical software engineering and a focus on user experience. Developed as a sophisticated tool for golfers, it breaks new ground in sports technology.

The heart of AutoCaddie is its machine learning model, which uses the extensive GolfDB database to analyze golf swings with impressive accuracy. This model, coupled with a sophisticated AI system, processes video and sensor data to provide detailed feedback on golf swings. The AI consists of two main parts: a video analysis system and a sensor network. Together, they capture and interpret complex data, offering golfers insights previously unattainable with conventional coaching methods.

Python, known for its simplicity and versatility, forms the backbone of AutoCaddie's software development. The user-friendly GUI, developed using tkinter, displays data in a clear, understandable manner. The choice of tools like Visual Studio Code for development and a focus on comprehensive testing ensure the system's reliability and performance.

Efficient data communication within AutoCaddie is key to its functionality. The system's modular design in data processing algorithms enhances its adaptability and robustness, ensuring that it can handle the demands of real-time golf swing analysis.

AutoCaddie excels in managing and interpreting data from various sources, including cameras and IMUs. This capability enables it to provide immediate, comprehensive feedback on golf swings, combining detailed analysis with practical advice.

The integration of the GUI and the rest of the system is a testament to AutoCaddie's commitment to a seamless user experience. Algorithmic optimizations ensure a balance between high performance and efficient resource use, enhancing the system's overall effectiveness.

Ongoing optimization and validation are key aspects of the project. Regular updates and refinements, guided by thorough validation processes, ensure the system remains effective and reliable in real-world scenarios.

In conclusion, AutoCaddie is more than just a technological achievement; it's a practical, user-friendly solution for golfers at various skill levels. It represents a harmonious blend of academic research and real-world application, marking a significant step forward in sports technology and training methods. This project exemplifies the potential of engineering innovation to make a tangible impact in the world of sports.

Chapter 12: Administrative Content

12.1 Estimated Budget

Estimated Budget

Item	Amount	Cost (Indv.)	Cost (Total)
MPU6050 IMU	2	\$9.99	\$19.98
ESP32-WROOM-32E MCU	1	\$3.00	\$3.00
PCB Orders	1	\$165	\$165.00
Transceiver (HC-06)	1	\$9.99	\$9.99
Li-Ion Battery (6.6Ah)	1	\$24.50	\$24.50
Camera (HP 320 FHD)	2	\$28.51	\$57.02
Electronic Components	1	\$31	\$31.00
Enclosure Printing	3	\$0	\$0.00
Suspender Mount	1	\$9.99	\$9.99
Tripods	2	\$13.99	\$27.98
Velcro	1	26.97	\$26.97
USB Extenders	2	\$9.99	\$19.98
Grand Total			
\$395.41			

Table 13: Estimated costs of AutoCaddie project.

12.2 Semester Milestones

First Semester

September 4th

- Database resources found

October 1st

- Artificial Intelligence methodology researched

October 16th

- Artificial Intelligence methodology defined

October 1st

- Hardware resourced and validated

October 30th

- Hardware systems designed

Second Semester

January 14th

- Hardware feedback model completed

January 28th

- Hardware systems designs completed

February 4th

- Artificial Intelligence vision system completed

March 3rd

- Complete Artificial Intelligence trained

March 22nd

- Feedback systems integrated

April 5th

- Testing results gathered

April 10th

- Final product completed

12.3 Distribution of Responsibilities

	Louis Deal	Rylan Simpson	Joshua Glaspey	Jarett Artman
Hardware Design	X	X		
Hardware Component Research	X	X		
MCU Research	X	X		
PCB Design Research	X	X		
PCB Schematic and Board Design	X	X		
Enclosure Mount Design		X		
Sensor Integration	X	X	X	
Firmware Code	X			
IMU Research		X		
Camera Research		X		
Battery Research		X		
Wireless Transceiver Research	X			
Quaternion Model Research		X		
IMU Application Research		X		
Quaternion Application Research		X	X	
Feedback Application Research	X	X	X	
Engineering Requirements	X	X	X	X
Engineering Requirements Organization	X	X		X
Engineering Constraints	X	X	X	X
Software Design			X	X
Software Research			X	X
GUI Design			X	
Video Data Acquisition			X	
Video Frame Extraction			X	
Video Frame Standardization			X	
Video Frame Bounding Box Annotation			X	
Video Frame Cropping			X	
Video Frame Padding			X	
Software Application Construction			X	
Kinematic Software Integration		X	X	
Machine Learning and Network			X	X
Machine Learning Research			X	X
Prototype Network Code				X
Network Optimizations				X
Network Validation				X
Network Output and Hyperparameter Selection			X	X
Action Detection via Neural Network Research			X	X
Communication				
Bluetooth Communication		X	X	
I2C Communication	X			
Testing and Validation				
Hardware Testing	X	X		
ESP32-WROOM-32E Implementation Testing	X	X		
MPU6050 IMU Implementation Testing	X	X		
Software Unit Testing	X	X	X	
Neural Network Testing				X

Table 14: Distributions of Responsibilities Table.

Appendix

Appendix A: Code Examples

[A1] Logic for importing live video data and extracting frames.

```
import cv2

def video_to_frame_list(video_path):
    vid_capture = cv2.VideoCapture(video_path)
    frames = []
    while True:
        success, frame = vid_capture.read()
        if not success:
            break
        frames.append(frame)
    vid_capture.release()
    return frames
```

[A2] Referencing bounding boxes to crop images to their region of interest.

```
import cv2

def crop_frames(frames, x, y, width, height):
    cropped_frames = []
    for frame in frames:
        cropped_frame = frame[y:y+height, x:x+width]
        cropped_frames.append(cropped_frame)
    return cropped_frames
```

[A3] ESP32 Initialization Code

```
void setup()
{
  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having
    compilation difficulties
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  // Initialize serial communication
  Serial.begin(9600);
  //while (!Serial)
  //    ; // WAIT FOR PROGRAMMER TO OPEN UP PROGRAMMING TERMINAL

  // initialize device
  Serial.println(F("Initializing I2C devices...\n"));
  Serial.println(F("Initializing IMU 1...\n"));
  mpu1.initialize();
  Serial.println(F("Initializing IMU 2...\n"));
  mpu2.initialize();

  // pinMode(INTERRUPT_PIN, INPUT); //For MPU6050 INT

  // Verify connection
  Serial.println(F("Testing device connections..."));
  Serial.println(mpu1.testConnection() ? F("MPU6050-1 connection successful") :
F("MPU6050-1 connection failed"));
  Serial.println(mpu2.testConnection() ? F("MPU6050-2 connection successful") :
F("MPU6050-2 connection failed"));

  // Load and configure the DMP
  Serial.println(F("Initializing DMP..."));
  devStatus1 = mpu1.dmpInitialize();
  devStatus2 = mpu2.dmpInitialize();

  // Supply calibration offsets
  mpu1.setXGyroOffset(75.00000);
  mpu1.setYGyroOffset(-106.00000);
  mpu1.setZGyroOffset(-68.00000);
  mpu1.setZAccelOffset(5090.00000); // 1688 factory default for my test chip
```



```

mpu2.setXGyroOffset(9.00000);
mpu2.setYGyroOffset(-40.00000);
mpu2.setZGyroOffset(-24.00000);
mpu2.setZAccelOffset(4936.00000); // 1688 factory default for my test chip

// CALIB DATA: Xaccel, Yaccel, Zaccel, Xgyro, Ygyro, Zgyro
// -1870.00000, -2499.00000, 5090.00000, 75.00000, -106.00000, -68.00000
// -270.00000, -105.00000, 4936.00000, 9.00000, 40.00000, 24.00000
//

// Ensure connections are valid
if (devStatus1 == 0)
{
    // Calibration Time: generate offsets and calibrate our MPU6050
    Serial.println(F("Printing offsets of IMU1..."));
    mpu1.CalibrateAccel(6);
    mpu1.CalibrateGyro(6);
    mpu1.PrintActiveOffsets();
    // Turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP of IMU1..."));
    mpu1.setDMPEnabled(true);

    // Set DMP Ready flag so the main loop() function knows it's okay to use
    it

    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady1 = true;

    // Get expected DMP packet size for later comparison
    packetSize1 = mpu1.dmpGetFIFOPacketSize();
}
else
{
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus1);
    Serial.println(F(")"));
}

if (devStatus2 == 0)
{
    // Calibration Time: generate offsets and calibrate our MPU6050
    Serial.println(F("Printing offsets of IMU1..."));

```

```

    mpu2.CalibrateAccel(6);
    mpu2.CalibrateGyro(6);
    mpu2.PrintActiveOffsets();
    // Turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP of IMU2..."));
    mpu2.setDMPEnabled(true);

    // Set DMP Ready flag so the main loop() function knows it's okay to use
it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady2 = true;

    // Get expected DMP packet size for later comparison
    packetSize2 = mpu2.dmpGetFIFOPacketSize();
}
else
{
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus2);
    Serial.println(F(")"));
}

BTSerial.begin(9600, SERIAL_8N1, RXD2, TXD2); // Initialize UART serial port
for Bluetooth communication
// Configure LED for output & power indication
pinMode(LED_PIN, OUTPUT);
digitalWrite(LED_PIN, HIGH);
pinMode(BUTTON_PIN, INPUT_PULLUP); // Set button pin as input with internal
pull-up resistor
pinMode(BUZZER_PIN, OUTPUT); // Set buzzerPin as output

// Configure GPIO wake-up
attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), wakeUp, ONLOW);
esp_sleep_enable_ext0_wakeup((gpio_num_t)BUTTON_PIN, LOW); // Enable wake-up
on

Serial.println(F("Press the button to record data.\n"));

delay(1000); // Add a short delay to allow serial data to be sent
}

```

[A4] ESP32 Main Loop

```
void loop()
{
    // if programming failed, don't try to do anything
    if (!(dmpReady1 && dmpReady2))
    {
        BTSerial.println("ERROR (1)");
        // Flash to signal error
        digitalWrite(LED_PIN, LOW);
        delay(500);
        digitalWrite(LED_PIN, HIGH);
        delay(500);
        digitalWrite(LED_PIN, LOW);
        delay(500);
        digitalWrite(LED_PIN, HIGH);
        delay(500);
        digitalWrite(LED_PIN, LOW);
    }

    delay(100);

    Serial.println("Sleeping...");

    esp_light_sleep_start();

    if (isRecording)
    {
        Serial.println("Sending data.");
        recordData();
        isRecording = false;
    }
    else
        Serial.println("ISR error");
}
```

[A5] ESP32 Sleep wakeup and recording routine

```
// Interrupt service routine
void IRAM_ATTR wakeUp()
{
    delay(20); // Debounce
    isRecording = true;
    digitalWrite(BUZZER_PIN, HIGH);
    delay(100);
}
```

```

    digitalWrite(BUZZER_PIN, LOW);
}

// Define logical data reading
void recordData()
{
    if (!(dmpReady1 && dmpReady2))
    {
        BTSerial.println("ERROR (1)");
        return;
    }
    else
    {
        BTSerial.println("READY");
        // Buzz once
        digitalWrite(BUZZER_PIN, HIGH);
        delay(200);
        digitalWrite(BUZZER_PIN, LOW);
        delay(3000);
        // Buzz once again after wait for user to get ready
        BTSerial.println("PREP");
        digitalWrite(BUZZER_PIN, HIGH);
        delay(200);
        digitalWrite(BUZZER_PIN, LOW);
        recordTime = millis();
        currentTime = millis(); // Get current time

        while (currentTime - recordTime <= 3000)
        { // If less than 3 seconds elapsed

            // read a packet from FIFO
            if (mpu1.dmpGetCurrentFIFOPacket(fifoBuffer1))
            { // Get the Latest packet of IMU1
#ifdef OUTPUT_READABLE_QUATERNION
                // display quaternion values in easy matrix form: w x y z
                mpu1.dmpGetQuaternion(&q1, fifoBuffer1);
                String output = String(currentTime) + "," +
                    "quat_1," +
                    String(q1.w) + "," +
                    String(q1.x) + "," +
                    String(q1.y) + "," +
                    String(q1.z);
                BTSerial.println(output);
#endif
            }
        }
    }
}

```

```

        if (mpu2.dmpGetCurrentFIFOPacket(fifoBuffer2))
        { // Get the Latest packet of IMU2
#ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu2.dmpGetQuaternion(&q2, fifoBuffer2);
            String output = String(currentTime) + "," +
                            "quat_2," +
                            String(q2.w) + "," +
                            String(q2.x) + "," +
                            String(q2.y) + "," +
                            String(q2.z);
            BTSerial.println(output);
#endif
        }

        delay(SAMPLING_RATE_PREP); // Sampling delay
        currentTime = millis(); // Get current time
    }

    digitalWrite(BUZZER_PIN, HIGH);
    delay(150);
    digitalWrite(BUZZER_PIN, LOW);
    delay(150);
    digitalWrite(BUZZER_PIN, HIGH);
    delay(150);
    digitalWrite(BUZZER_PIN, LOW);
    recordTime = millis();
    BTSerial.println("SWING");
    currentTime = millis(); // Get current time

    while (currentTime - recordTime <= 3000)
    { // If less than 3 seconds elapsed

        // read a packet from FIFO
        if (mpu1.dmpGetCurrentFIFOPacket(fifoBuffer1))
        { // Get the Latest packet of IMU1
#ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu1.dmpGetQuaternion(&q1, fifoBuffer1);
            String output = String(currentTime) + "," +
                            "quat_1," +
                            String(q1.w) + "," +
                            String(q1.x) + "," +
                            String(q1.y) + "," +

```

```

        String(q1.z);
        BTSerial.println(output);
#endif
    }

    if (mpu2.dmpGetCurrentFIFOPacket(fifoBuffer2))
    { // Get the Latest packet of IMU2
#ifdef OUTPUT_READABLE_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu2.dmpGetQuaternion(&q2, fifoBuffer2);
        String output = String(currentTime) + "," +
            "quat_2," +
            String(q2.w) + "," +
            String(q2.x) + "," +
            String(q2.y) + "," +
            String(q2.z);
        BTSerial.println(output);
#endif
    }

    delay(SAMPLING_RATE_SWING); // Sampling delay
    currentTime = millis(); // Get current time
}
BTSerial.println("END");
digitalWrite(BUZZER_PIN, HIGH);
delay(500);
digitalWrite(BUZZER_PIN, LOW);
delay(500);
digitalWrite(BUZZER_PIN, HIGH);
delay(500);
digitalWrite(BUZZER_PIN, LOW);
recordTime = millis();
}
}

```

[A6] Hybrid LSTM and CNN code prototype code using tensorflow

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, LSTM, Dense, Flatten, TimeDistributed, Dropout

def create_hybrid_cnn_lstm_model(input_shape, num_classes):
    model = Sequential()

    # CNN layers - TimeDistributed wrapper to process each frame
    model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input_shape=input_shape))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Conv2D(64, (3, 3), activation='relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Conv2D(128, (3, 3), activation='relu')))
    model.add(TimeDistributed(Flatten()))

    # LSTM layers
    model.add(LSTM(50, return_sequences=True))
    model.add(Dropout(0.5))
    model.add(LSTM(50))

    # Dense layer for classification
    model.add(Dense(num_classes, activation='softmax'))

    return model

# Example: assuming the input is a sequence of 5 frames, each of size 64x64 with 3 channels (RGB)
input_shape = (5, 64, 64, 3)
num_classes = 8 # Example: 8 different events in a golf swing

model = create_hybrid_cnn_lstm_model(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

[A7] Basic implementation of using Java to write to the console.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

[A8] Basic implementation of using Python to write to the console.

```
print("Hello, World!")
```


[A9] Implementation of the quaternion angle calculation.

```
import numpy as np

# Calculate the Quaternion Angle Difference
# Precondition:
# q1 and q2 are defined as vectors of size [4,1]
def calculate_quaternion_angle_difference(q1, q2):
    return np.arccos(2 * np.dot(q1,q2) ** 2 - 1)

# Define Quaternion Data
q1 = np.array([w1, x1, y1, z1])
q2 = np.array([w2, x2, y2, z2])

# Find the Quaternion Angle Difference
theta = calculate_quaternion_angle_difference(q1, q2)
```

[A10] Logic for connecting via a Bluetooth socket to a transmitting device.

```
# Bluetooth server configuration
server_socket = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
port = 1 # RFCOMM port number
server_socket.bind(("", port))
server_socket.listen(1)
```

[A11] Logic for receiving data from the MCU through AutoCaddie.

```
# Receive data from the MCU
try:
    while True:
        data = client_socket.recv(1024) # Receive data
        if not data:
            break
        print("Received data:", data.decode("utf-8")) # Decode the received data
except Exception as e:
    print("Error:", str(e))
```

[A12] Implementation of using synchronization to extract images from two live video feeds.

```
# Initialize camera objects using distinct indices
camera_1 = cv2.VideoCapture(0)
camera_2 = cv2.VideoCapture(1)

# Arrays to store frames
frames_camera_1 = []
frames_camera_2 = []

while True:
    # Capture frames from both cameras
    ret_1, frame_1 = camera_1.read()
    ret_2, frame_2 = camera_2.read()

    # Check if frames are successfully captured
    if not ret_1 or not ret_2:
        break

    # Store frames in arrays
    frames_camera_1.append(frame_1)
    frames_camera_2.append(frame_2)
```

[A13] Basic setup of a parent wrapping function for the AI system of AutoCaddie.

```
# Function to analyze swing using neural network
def analyze_swing(frames_camera_1, frames_camera_2):
    # Define the neural network logic here
    # Call neural network model to process frame data and compare to the database
    # Return a list of floats representing the score of swing features

# Example function call
swing_analysis_object = analyze_swing(frames_camera_1, frames_camera_2)
```

[A14] Code to strip header data from a MCU data package.

```
# Gather the csv data from the MCU
header, csv_data = data.split(",", 1)
if header.strip() == "data":
    # Call the analyze_quaternion_data function with quaternion data
    analyze_quaternion_data(csv_data.strip())
```

[A15] Code to initialize the GUI application of AutoCaddie.

```
import wx
from algorithms import frame_extraction

class AutoCaddieApp(wx.App):
    def OnInit(self):
        # Call the frame extraction algorithm
        frames = frame_extraction.extract_frames("video_file.mp4")
```

[A16] Code for unit testing the video frame extraction through the software application.

```
import unittest
from algorithms.frame_extraction import extract_frames

class TestFrameExtraction(unittest.TestCase):
    def test_extract_frames_single_video(self):
        # Test extracting frames from a single video
        video_path = "path/to/single_video.mp4"
        frames = extract_frames(video_path)
        self.assertEqual(len(frames), expected_number_of_frames)

if __name__ == '__main__':
    unittest.main()
```

[A17] Software application implementation for extracting frames from a live video feed.

```
import cv2

def extract_frames(camera_indices=[0, 1]):
    cameras = [cv2.VideoCapture(idx) for idx in camera_indices]
    frames = [[] for _ in camera_indices]
    while True:
        ret_frames = [camera.read() for camera in cameras]
        if not all(ret_frames):
            break
        for i, (ret, frame) in enumerate(ret_frames):
            if ret:
                frames[i].append(frame)
    for camera in cameras:
        camera.release()
    return frames
```

[A18] Implementation of connecting MobileNetV2 to AutoCaddie's software application.

```
import cv2
import numpy as np
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

def annotate_bounding_boxes(frames):
    # Load pre-trained MobileNetV2 model
    model = MobileNetV2(weights='imagenet')

    # Bounding box annotation process
    annotated_frames = []
    for frame in frames:
        # Preprocess frame for MobileNetV2
        img = preprocess_input(frame)
        img = cv2.resize(img, (224, 224))
        img = np.expand_dims(img, axis=0)

        # Run MobileNetV2 model
        predictions = model.predict(img)

        # Extract bounding box coordinates from prediction
        bounding_box = extract_bounding_box(predictions)

        # Draw bounding box on the frame
        annotated_frame = draw_bounding_box(frame, bounding_box)
        annotated_frames.append(annotated_frame)

    return annotated_frames
```

[A19] Snippet of the algorithm for cropping images to the region of interest.

```
for frame in frames:
    x, y, width, height = bounding_box
    cropped_frame = frame[y:y+height, x:x+width]
```

[A20] Implementation of zero padding through the software application.

```
import cv2

def zero_pad_frame(frame, target_size):
    height, width, _ = frame.shape
    pad_height = (target_size - height) // 2
    pad_width = (target_size - width) // 2

    # Applying zero-padding
    padded_frame = cv2.copyMakeBorder(frame, pad_height, pad_height, pad_width,
    pad_width, cv2.BORDER_CONSTANT, value=[0, 0, 0])
    return padded_frame
```

[A21] Implementation of model tuning of hyperparameters during the development process.

```
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

# Create a LinearSVC model
model = LinearSVC()

# Define the hyperparameters to tune
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Train the model with the best hyperparameters
best_model = LinearSVC(C=grid_search.best_params_['C'])
best_model.fit(X_train, y_train)
```

[A22] Connection of the action recognition neural network to AutoCaddie's software application.

```
from sklearn.externals import joblib
import numpy as np

def load_model():
    try:
        # Load the saved model if it exists
        model = joblib.load('linear_svc_model.pkl')
        return model
    except FileNotFoundError:
        return None

def determine_action_frames(input_frames):
    model = load_model()

    if model is not None:
        predicted_actions = model.predict(input_frames)

        # Access the probability scores for each event
        action_probabilities = model.decision_function(input_frames)

        # Select the frame with the highest probability for each predicted action
        selected_frames = [np.argmax(probabilities) for probabilities in
action_probabilities]
        return selected_frames
    else:
        return None
```

[A23] Implementation of the quaternion data processing in AutoCaddie's software application.

```
import numpy as np

def calculate_angles(Q1, Q2):
    dot_products = np.sum(Q1 * Q2, axis=1)
    squared_terms = 2 * dot_products**2 - 1
    theta = np.arccos(squared_terms)
    return theta
```

[A24] Process of isolating a particular action using detected starting frames.

```
action = frames[actions[i] : actions[i + 1] - 1]
```

[A25] Hyperparameter Identification and mapping code prototy

```
import numpy as np

# Assume model_output is an array of probabilities from our neural network model for each frame
# Each row represents a frame, and each column represents a probability for an event

# Define the events for readability
events = ["Address", "Toe-up", "Mid-backswing", "Top", "Mid-downswing", "Impact", "Mid-follow-through", "Finish"]

def extract_event_probabilities(model_output):
    # Find the index of the maximum probability for each frame
    max_event_indices = np.argmax(model_output, axis=1)

    # Translate indices to event names
    identified_events = [events[index] for index in max_event_indices]
    return identified_events

# Example usage
model_output = np.random.rand(100, len(events)) # Sample output
identified_events = extract_event_probabilities(model_output)
print("Identified Events per Frame:", identified_events)

def thresholding_analysis(model_output, threshold=0.7):
    # Thresholding to identify significant event probabilities
    significant_events = model_output > threshold

    # Mapping to event names
    for frame_index, events in enumerate(significant_events):
        print(f"Frame {frame_index}: ", end="")
        for event_index, is_significant in enumerate(events):
            if is_significant:
                print(events[event_index], end=" ")
        print()

# Example usage
thresholding_analysis(model_output)
```

[A26] Using wxpython to initialize AutoCaddie's GUI.

```
import wx
from gui_module import AutoCaddieGUI

if __name__ == '__main__':
    app = wx.App()
    AutoCaddieGUI(None, title='AutoCaddie GUI')
    app.MainLoop()
```

[A27] Example of how to start AutoCaddie's GUI for the user.

```
app = wx.App()
```

[A28] Example of initializing the GUI window for a generic interface.

```
class MyFrame(wx.Frame):
    def __init__(self, parent, title):
        super(MyFrame, self).__init__(parent, title=title, size=(400, 300))
```

[A29] Example of including widgets on a GUI window.

```
class MyFrame(wx.Frame):
    def __init__(self, parent, title):
        super(MyFrame, self).__init__(parent, title=title, size=(400, 300))

        # Creating a button
        btn = wx.Button(self, label='Click me')

        # Binding button click event to the on_button_click function
        btn.Bind(wx.EVT_BUTTON, self.on_button_click)

    def on_button_click(self, event):
        print("Button clicked!")
```

[A30] Example of providing scalability so the GUI window can adapt to different screen sizes.

```
class AutoCaddieGUI(wx.Frame):
    def InitUI(self):
        panel = wx.Panel(self)
        hbox = wx.BoxSizer(wx.HORIZONTAL)
        # Insert code for attaching components to the Horizontal Box here.
        panel.SetSizer(hbox)
```

[A31] Example of creating a menu through AutoCaddie's GUI.

```
menubar = wx.MenuBar()
fileMenu = wx.Menu()
fileMenu.Append(wx.ID_EXIT, 'Exit', 'Exit application')
menubar.Append(fileMenu, '&File')
self.SetMenuBar(menubar)
```

[A32] Example of binding an action to a widget on a GUI window.

```
self.Bind(wx.EVT_MENU, self.OnQuit, id=wx.ID_EXIT)
```

[A33] Implementation of properly leaving a GUI application.

```
def OnQuit(self, event):
    self.Close()
```

[A34] Integration of the neural network for frame processing through the GUI.

```
# Neural Network Integration for GUI
def integrate_neural_network(frames, golfdb_database):
    similarity_weights = []

    for frame in frames:
        weights = cnn_process_frame(frame, golfdb_database)
        similarity_weights.append(weights)

    return similarity_weights
```


[A35] Example of live updating of data through the analysis process of AutoCaddie.

```
# GUI Implementation
def update_gui(frames, golfdb_database, quaternion_data, threshold):
    # Neural Network Integration
    similarity_weights = integrate_neural_network(frames, golfdb_database)

    # Quaternion Data Visualization
    visualize_quaternion_data(quaternion_data, threshold)

    # Update GUI with real-time feedback
    gui_update(similarity_weights)
```

[A36] Hybrid prototype example with tensorflow

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, LSTM, Dense, Flatten, TimeDistributed, Dropout

def create_hybrid_cnn_lstm_model(input_shape, num_classes):
    model = Sequential()

    # CNN layers - TimeDistributed wrapper to process each frame
    model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input_shape=input_shape))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Conv2D(64, (3, 3), activation='relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Conv2D(128, (3, 3), activation='relu')))
    model.add(TimeDistributed(Flatten()))

    # LSTM layers
    model.add(LSTM(50, return_sequences=True))
    model.add(Dropout(0.5))
    model.add(LSTM(50))

    # Dense layer for classification
    model.add(Dense(num_classes, activation='softmax'))

    return model

# Example: assuming the input is a sequence of 5 frames, each of size 64x64 with 3 channels (RGB)
input_shape = (5, 64, 64, 3)
num_classes = 8 # Example: 8 different events in a golf swing

model = create_hybrid_cnn_lstm_model(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

[A37] Creating a training testing split using tensor flow

```
from sklearn.model_selection import train_test_split
import numpy as np

# Assuming 'data' is your dataset and 'labels' are the corresponding labels
data = np.load('golf_swing_data.npy')
labels = np.load('golf_swing_labels.npy')

# Splitting the dataset into training and validation sets
train_data, val_data, train_labels, val_labels = train_test_split(data, labels, test_size=0.2, random_state=42)

# Now, 'val_data' and 'val_labels' can be used as your validation dataset
```

[A38] K fold validation testing using sklearn

```
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a simple model for demonstration purposes
def create_model():
    model = Sequential([Dense(10, input_shape=(input_shape,), activation='relu'),
                        Dense(num_classes, activation='softmax')])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# k-Fold Cross-Validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = []

for train, test in kfold.split(data, labels):
    model = create_model()
    model.fit(data[train], labels[train], epochs=10, batch_size=32, verbose=0)
    score = model.evaluate(data[test], labels[test], verbose=0)
    scores.append(score)

# 'scores' contains the evaluation results for each fold
```

[A39] Implementation of calculating performance metrics through the entire golf swing sequence.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Assume 'predictions' and 'true_labels' are the model's predictions and the true labels, respectively
predictions = model.predict(val_data)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(val_labels, axis=1)

# Calculating various performance metrics
accuracy = accuracy_score(true_classes, predicted_classes)
precision = precision_score(true_classes, predicted_classes, average='weighted')
recall = recall_score(true_classes, predicted_classes, average='weighted')
f1 = f1_score(true_classes, predicted_classes, average='weighted')
conf_matrix = confusion_matrix(true_classes, predicted_classes)

# Print the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

[A40] Example of providing insights to errors computed through AutoCaddie's AI.

```
# Analyzing where the model predictions go wrong
error_indices = np.where(predicted_classes != true_classes)[0]

# Analyzing errors
for index in error_indices[:10]: # Just looking at the first 10 errors
    print(f>Data Index: {index}, Predicted: {predicted_classes[index]}, True: {true_classes[index]}")

# Further analysis can be conducted to understand the nature of these errors
```

Appendix B: Datasheets / Hardware Pin Configurations

- [B1]. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf
- [B2]. <https://www.rajguruelectronics.com/Product/707/HC-06%20core%20bluetooth%20module.pdf>
- [B3]. https://cdn-shop.adafruit.com/product-files/5035/5035_10050mAh_3.7V_A1____20210511.pdf
- [B4]. <https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/RM-MPU-6000A.pdf>
- [B5]. <https://www.diodes.com/assets/Datasheets/AP1603.pdf>

Appendix C: References

- [1] W. McNally, K. Vats, T. Pinto, C. Dulhanty, J. McPhee, and A. Wong, "GolfDB: A Video Database for Golf Swing Sequencing." Accessed: Sep. 27, 2023. [Online]. Available: <https://arxiv.org/pdf/1903.06528.pdf>
- [2] P. Cintia, "Coach2vec: autoencoding the playing style of soccer coaches." Accessed: Sep. 26, 2023. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/2106/2106.15444.pdf>
- [3] G. Searle, "Singularity: Using A Neural Network to Predict the Outcome of Plate Appearances," *Baseball Prospectus*, Jul. 09, 2020. <https://www.baseballprospectus.com/news/article/59993/singularity-using-a-neural-network-to-predict-the-outcome-of-plate-appearances/> (accessed Sep. 26, 2023).
- [4] "Home," www.alfaswing.ai. <https://www.alfaswing.ai/> (accessed Sep. 26, 2023).
- [5] "AI Coach | 18Birdies," *18birdies.com*. <https://18birdies.com/aicoach/> (accessed Sep. 26, 2023).
- [6] "Sportsbox AI," *SB*. <https://www.sportsbox.ai/> (accessed Sep. 26, 2023).
- [7] P. Carroll, "Apple iPhone XR camera review: Top-ranked single-lens phone," *DXOMARK*, Dec. 06, 2018. <https://www.dxomark.com/apple-iphone-xr-camera-review/>
- [8] C.-C. Liao, D.-H. Hwang, and H. Koike, "AI Golf: Golf Swing Analysis Tool for Self-Training," *IEEE Access*, vol. 10, pp. 106286–106295, 2022, doi: <https://doi.org/10.1109/access.2022.3210261>.
- [9] "SMARTGOLF AI Coach | SMARTGOLF," Jun. 28, 2022. <https://smartgolf.biz/en/product/smartgolf-ai-x/>
- [10] X. Zhu, Q. Wang, and W. D. Lu, "Memristor networks for real-time neural activity analysis," *Nature Communications*, vol. 11, no. 1, May 2020, doi: <https://doi.org/10.1038/s41467-020-16261-1>.
- [11] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-Time Guitar Amplifier Emulation with Deep Learning," *Applied Sciences*, vol. 10, no. 3, p. 766, Jan. 2020, doi: <https://doi.org/10.3390/app10030766>.
- [12] "DeepMind - What if solving one problem could unlock solutions to thousands more?," *Deepmind*. <https://www.deepmind.com/>
- [13] H. Dong *et al.*, "A deep convolutional neural network for real-time full profile analysis of big powder diffraction data," *npj Computational Materials*, vol. 7, no. 1, pp. 1–9, May 2021, doi: <https://doi.org/10.1038/s41524-021-00542-4>.
- [14] M. Versteyhe, H. De Vroey, F. Debrouwere, H. Hallez, and K. Claeys, "A novel method to estimate the full knee joint kinematics using low cost IMU sensors for easy to implement low cost diagnostics," *Sensors*, vol. 20, no. 6, p. 1683, 2020. doi:10.3390/s20061683
- [15] J. H. Challis, "Quaternions as a solution to determining the angular kinematics of human movement," *BMC Biomedical Engineering*, vol. 2, no. 1, 2020. doi:10.1186/s42490-020-00039-z
- [16] Md. M. Rahman, K. B. Gan, N. A. Aziz, A. Huong, and H. W. You, "Upper Limb Joint Angle Estimation Using Wearable Imus and personalized calibration algorithm," *Mathematics*, vol. 11, no. 4, p. 970, 2023. doi:10.3390/math11040970

- [17] Leo J. Roberts, Mervyn S Jackson, Ian H. Grundy. "The effects of cognitive interference during the preparation and execution of the golf swing," International Journal of Sport and Exercise Psychology, pp. 413-428, 2021.
- [18] Timothy Fulton, Josiah Ertz, Abraham Rohler, Fabio Fontana, Mickey Mack. "The Effects of a Visual Distraction," International Journal of Golf Science, no. 3, pp. 26-34, 2014.
- [19] Saba M Hoesseini, David Fukada, Joon-Hyuk Park. "Optimal Multi-Modal Sensory Feedback to Maximize Motor Learning Efficiency," Department of Mechanical and Aerospace Engineering, University of Central Florida.
- [20] G. Oppy and D. Dowe, "The Turing Test (Stanford Encyclopedia of Philosophy)," Stanford.edu, Apr. 09, 2003. <https://plato.stanford.edu/entries/turing-test/>
- [21] H. Singh, "How are Images Stored on a Computer | Grayscale & RGB Image Formats," Analytics Vidhya, Mar. 16, 2021. <https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/>
- [22] Krizhevsky, A., Sutskever, I., Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, 2012, 25: 1097-1105.
- [23] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep Learning for Computer Vision: A Brief Review," Computational Intelligence and Neuroscience, vol. 2018, pp. 1–13, 2018, doi: <https://doi.org/10.1155/2018/7068349>.
- [24] "Papers with Code - SDAE Explained," paperswithcode.com. <https://paperswithcode.com/method/sdae> (accessed Nov. 12, 2023).
- [25] S. Park, J. Chang, H. Jeong, J.-H. Lee, and J.-Y. Park, "Accurate and Efficient 3D Human Pose Estimation Algorithm using Single Depth Images for Pose Analysis in Golf." Accessed: Sep. 27, 2023. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017_workshops/w2/papers/Park_Accurate_and_Efficient_CVPR_2017_paper.pdf
- [26] B. Tian and W. Wei, "Research Overview on Edge Detection Algorithms Based on Deep Learning and Image Fusion," Security and Communication Networks, vol. 2022, pp. 1–11, Sep. 2022, doi: <https://doi.org/10.1155/2022/1155814>.
- [27] "Can we trust bounding box annotations for object detection? | IEEE Conference Publication | IEEE Xplore," [ieeexplore.ieee.org](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9857206). <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9857206> (accessed Nov. 12, 2023).
- [28] M. Hashemi, "Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation," Journal of Big Data, vol. 6, no. 1, Nov. 2019, doi: <https://doi.org/10.1186/s40537-019-0263-7>.
- [29] Hashemi, Mahdi. (2019). Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. Journal of Big Data. 6. 10.1186/s40537-019-0263-7.
- [30] Marais, Marc. (2021). Golf Swing Sequencing using Computer Vision. 10.13140/RG.2.2.14864.48641.
- [31] K. Simonyan and A. Zisserman, "Two-Stream Convolutional Networks for Action Recognition in Videos." Available:

- https://papers.nips.cc/paper_files/paper/2014/file/00ec53c4682d36f5c4359f4ae7bd7ba1-Paper.pdf
- [32] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale Video Classification with Convolutional Neural Networks." Available: <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42455.pdf>
 - [33] "MobileNet V2 — Torchvision main documentation," pytorch.org. <https://pytorch.org/vision/main/models/mobilenetv2.html> (accessed Sep. 27, 2023).
 - [34] J. Diebel, "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors," 2006. Available: <https://www.astro.rug.nl/software/kapteyn-beta/downloads/attitude.pdf>
 - [35] "Euler angles - Knowino," www.tau.ac.il. https://www.tau.ac.il/~tsirel/dump/Static/knowino.org/wiki/Euler_angles.html (accessed Nov. 12, 2023).
 - [36] "Trilinear interpolation," Wikipedia, Jan. 08, 2023. https://en.wikipedia.org/wiki/Trilinear_interpolation (accessed Nov. 12, 2023).
 - [37] J. H. Challis, "Quaternions as a solution to determining the angular kinematics of human movement," BMC Biomedical Engineering, vol. 2, no. 1, Mar. 2020, doi: <https://doi.org/10.1186/s42490-020-00039-z>.
 - [38] J. Novotny, "A Programmers' Guide to Python: Advantages & Disadvantages," Linode Guides & Tutorials, Mar. 09, 2023. <https://www.linode.com/docs/guides/pros-and-cons-of-python/>
 - [39] Javatpoint, "Advantages and disadvantages of Java - Javatpoint," www.javatpoint.com. <https://www.javatpoint.com/advantages-and-disadvantages-of-java>
 - [40] "Advantages and Disadvantages of C++ Programming Language - Javatpoint," www.javatpoint.com. <https://www.javatpoint.com/advantages-and-disadvantages-of-cpp-language>
 - [41] "Python vs JavaScript - Javatpoint," www.javatpoint.com. <https://www.javatpoint.com/python-vs-javascript>
 - [42] A. Saxena, "Introduction to NumPy," Medium, Jun. 01, 2020. <https://medium.com/analytics-vidhya/introduction-to-numpy-279bbc88c615>
 - [43] "Introduction — SciPy v1.9.0 Manual," docs.scipy.org. <https://docs.scipy.org/doc/scipy-1.9.0/tutorial/general.html> (accessed Nov. 12, 2023)
 - [44] "Pandas Vs NumPy: What's The Difference? [2023]," InterviewBit, Oct. 22, 2021. <https://www.interviewbit.com/blog/pandas-vs-numpy/>
 - [45] OpenCV, "OpenCV library," Opencv.org, 2019. <https://opencv.org/>
 - [46] K. Kroening, "ffmpeg-python: Python bindings for FFmpeg," GitHub, Nov. 12, 2023. <https://github.com/kkroening/ffmpeg-python> (accessed Nov. 12, 2023).
 - [47] A. C. Salian, "video2images: Video 2 Image converter," PyPI. <https://pypi.org/project/video2images/> (accessed Nov. 12, 2023).
 - [48] Keras, "Home - Keras Documentation," Keras.io, 2019. <https://keras.io/>
 - [49] scikit learn, "1.4. Support Vector Machines — scikit-learn 0.20.3 documentation," Scikit-learn.org, 2018. <https://scikit-learn.org/stable/modules/svm.html>

- [50] "Models & datasets," TensorFlow. <https://www.tensorflow.org/resources/models-datasets>
- [51] "torchvision.models — Torchvision 0.10.0 documentation," pytorch.org. <https://pytorch.org/vision/stable/models.html>
- [52] R. C. Limited, "PyQt5: Python bindings for the Qt cross platform application toolkit," PyPI. <https://pypi.org/project/PyQt5/#:~:text=PyQt5%205.15.10&text=PyQt5%20is%20a%20c> omprehensive%20set (accessed Nov. 12, 2023).
- [53] Python Software Foundation, "tkinter — Python interface to Tcl/Tk — Python 3.7.2 documentation," python.org, 2019. <https://docs.python.org/3/library/tkinter.html>
- [54] T. wxPython Team, "Welcome to wxPython!," wxPython, Aug. 02, 2021. <https://wxpython.org/index.html>
- [55] "Kivy: Cross-platform Python Framework for NUI," kivy.org. <https://kivy.org/>
- [56] "PySimpleGUI," www.pysimplegui.org. <https://www.pysimplegui.org/en/latest/>
- [57] "American National Standards Institute - ANSI Home." *American National Standards Institute - ANSI*, ansi.org/. Accessed 30 Oct. 2023.
- [58] "PyTorch Benchmark — PyTorch Tutorials 2.1.0+Cu121 Documentation." Pytorch.org, pytorch.org/tutorials/recipes/recipes/benchmark.html. Accessed 30 Oct. 2023.
- [59] van Rossum, Guido, et al. "PEP 8 – Style Guide for Python Code | Peps.python.org." Peps.python.org, 5 July 2001, peps.python.org/pep-0008/.
- [60] "Rules of Machine Learning: | ML Universal Guides." Google Developers, developers.google.com/machine-learning/guides/rules-of-ml.
- [61] Network Architecture of SwingNet. 2019. Accessed: Sep. 26, 2023. [Online]. Available: <https://arxiv.org/pdf/1903.06528.pdf> <https://arxiv.org/pdf/1412.6980.pdf>
- [62] "IEEE/EIA - Standard for Information Technology - Software Life Cycle Processes," in IEEE/EIA 12207.0-1996 , vol., no., pp.1-88, 31 March 1998, doi: 10.1109/IEEESTD.1998.88083.
- [63] Darrah, Marjorie. "Verification and Validation of Neural Networks: a Sampling of Research in Progress." SPIE Proceedings, 2003.
- [64] "User Experience Design (UX) Standards and Guidelines | Enterprise Brand and Marketing Guide." Brandguide.asu.edu, brandguide.asu.edu/execution-guidelines/web/ux-design.
- [65] U.S. Access Board. "Section508.Gov." Www.section508.Gov, Sept. 2023, www.section508.gov/test/web-software/. Accessed 29 Oct. 2023.
- [66] "ANDI - Accessibility Testing Tool - Install." Www.ssa.gov, www.ssa.gov/accessibility/andi/help/install.html.
- [67] Lauke, Patrick. "Release CCA 2.5.0 · ThePacielloGroup/CCA-Win." GitHub, 3 June 2017, github.com/ThePacielloGroup/CCA-Win/releases/tag/2.5.0. Accessed 30 Oct. 2023.
- [68] "Section508.Gov." Www.section508.Gov, Aug. 2023, www.section508.gov/develop/universal-design/.
- [69] OpenAI. "ChatGPT: Revolutionizing Research and Writing," 2023. [Online]. Available: <https://www.openai.com/research/chatgpt>. Accessed: Oct. 29 2023.

- [70] hix.ai. "HIX.AI: All-In-One ChatGPT Copilot for Web." Chrome.google.com, 13 Oct. 2023, chrome.google.com/webstore/detail/hixai-all-in-one-chatgpt/njggknpmkjapgklcfhaiigafiiebpchm. Accessed 30 Oct. 2023.
- [71] "ChatSonic - like ChatGPT but with Superpowers." Writesonic.com, writesonic.com/chat. Accessed 29 Oct. 2023.
- [72] "Your AI-Powered Copilot for the Web." Www.microsoft.com, Feb. 2023, www.microsoft.com/en-us/bing?form=MA13FV. Accessed 29 Oct. 2023.
- [73] Demo, GPT-3. "YouChat | Discover AI Use Cases." Gpt3demo.com, 23 Dec. 2022, gpt3demo.com/apps/youchat. Accessed 30 Oct. 2023.
- [74] "Claude." Claude.ai, July 2023, claude.ai/. Accessed 29 Oct. 2023.
- [75] "Bard - Chat Based AI Tool from Google, Powered by PaLM 2." Bard.google.com, 21 Mar. 2023, bard.google.com/chat. Accessed 29 Oct. 2023.
- [76] "AutoGPT Official." AutoGPT Official, 16 Aug. 2023, autogpt.net/.
- [77] "CopyAI: Create Marketing Copy in Seconds." Www.copy.ai, www.copy.ai/. Accessed 29 Oct. 2023.
- [78] "WEBENCH-CIRCUIT-DESIGNER Design tool | TI.com," [www.ti.com](https://www.ti.com/tool/WEBENCH-CIRCUIT-DESIGNER). <https://www.ti.com/tool/WEBENCH-CIRCUIT-DESIGNER>
- [79] Oracle. "Java Documentation." Oracle Help Center, 23 Jan. 1996, docs.oracle.com/en/java/.
- [80] Ahmed, Reshma. "What Is Git ? – Explore a Distributed Version Control Tool." Edureka, Edureka, 16 Nov. 2016, www.edureka.co/blog/what-is-git/. Accessed 29 Oct. 2023.
- [81] "StackEdit." Stackedit.io, stackedit.io/app. Accessed 30 Oct. 2023.
- [82] "unittest — Unit testing framework — Python 3.8.2 documentation," docs.python.org. <https://docs.python.org/3/library/unittest.html>
- [83] "🔥 Make GridSearch Faster for xgboost 🔥 | Kaggle," www.kaggle.com. <https://www.kaggle.com/discussions/general/407135> (accessed Nov. 26, 2023).
- [84] "Joblib: running Python functions as pipeline jobs — joblib 1.3.2 documentation," joblib.readthedocs.io. <https://joblib.readthedocs.io/en/stable/>
- [85] M. Marais and D. Brown, "Golf Swing Sequencing Using Computer Vision," Lecture Notes in Computer Science, pp. 351–365, Jan. 2022, doi: https://doi.org/10.1007/978-3-031-04881-4_28.
- [86] "Singular Value Decomposition." Wikipedia, 30 Dec. 2020, en.wikipedia.org/wiki/Singular_value_decomposition.
- [87] T. M. Authors, "mediapipe: MediaPipe is the simplest way for researchers and developers to build world-class ML solutions and applications for mobile, edge, cloud and the web.," PyPI. <https://pypi.org/project/mediapipe/>
- [88] "Python Release Python 3.8.0," Python.org. <https://www.python.org/downloads/release/python-380/>
- [89] "Pose landmarks detection task guide | MediaPipe," Google Developers. https://developers.google.com/mediapipe/solutions/vision/pose_landmarker
- [90] "Minimum configuration requirements to run Mediapipe · Issue #1941 · google/mediapipe," GitHub. <https://github.com/google/mediapipe/issues/1941> (accessed Apr. 21, 2024).
- [91] "Problem with limited memory after loading model - Jetson Nano," NVIDIA Developer Forums, Jul. 28, 2021. <https://forums.developer.nvidia.com/t/problem->

with-limited-memory-after-loading-model-jetson-nano/184956 (accessed Apr. 21, 2024).