Not Wii Bowling

Anthony Guerrera, Kimon Anagastopoulous, Benjamin Delgado, Brandon Sutton

I. Abstract

Young people are always looking for easy new ways to unwind and play their favorite video games. In today's world, many people have less time to play on their favorite large gaming consoles due to time and money. As such, a gaming solution that is portable and takes little to no time to set up and play is highly in demand. Playing a short game to give the mind a rest from other work is something that many young adults strive for in today's world. We aim to introduce a new product that solves this demand directly. ''Not Wii Bowling'' is an easily downloadable product that takes minutes to set up and play. With simple in-game graphics and cheap accessories, our version of an online bowling game combines elegance with simplicity. In naming our product ''Not Wii Bowling'', we intended to put a fun twist on Nintendo's Wii Sports Bowling and emphasize that our product is more easily accessible due to it's open source nature. We made use of the ESP32-C6 microcontroller, an IMU, and the Panda3D Python gaming graphics engine to assemble a simple and beautiful gaming experience. The battery that comes as part of the controller lasts as long as you need to play our game. Additionally, we used ingenious techniques to avoid the use of any peripheral buttons to make the game-play even more seamless. We prioritized engineering performance, testing, & engineering best practices in creating this excellent product.

II. Introduction

THIS report is designed to provide an in-depth technical overview of the ''Not Wii Bowling'' product, its use case in today's competitive market, and overall motivation for the idea & product.

Over the past ten years, gaming has been shifting more to the mobile side due to accessibility & convenience [11]. Mobile & desktop gaming currently dominates the gaming market, which accounts for 55% of all gaming software revenues in 2024 & 2025 [12]. Our solution fits nicely right into this market and is placed where there is a real opportunity for product-market fit to generate potential revenue. However, our goal for this product is to make it free & open source to users, as our mission is to help improve the lives of others, & do not benefit from taking any potential profits from this venture. High-performance gaming software takes time and focus to create – we believe high degrees of collaboration and established standards are critical for writing and shipping high-quality software to users.

M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

J. Doe and J. Doe are with Anonymous University.

Manuscript received April 19, 2005; revised August 26, 2015.

The technical contributions of this work extend beyond the product itself, exploring several areas of interactive computing and sensor integration. First, we present a novel approach to motion tracking that combines low-cost IMU sensors with efficient Kalman filtering techniques, addressing the persistent challenge of sensor drift that can limit similar systems. Second, our implementation demonstrates an innovative combination of computer vision that maintains accuracy while minimizing computational overhead. Third, we contribute a validated methodology for real-time wireless transmission of motion data via BLE that achieves low latency requirements for responsive game-play while maintaining power efficiency. Finally, our calibration-based approach for physics simulation parameters provides a replicable framework for translating physical motion to virtual environments with high fidelity. We present empirical evaluation results that quantify the effectiveness of these integrated systems across varying usage patterns and environmental conditions, establishing benchmarks for future work in accessible motion-based gaming. Throughout the development process, we prioritized engineering principles including modular system design, extensive error handling, and computational efficiency to create a solution that advances the technical state of the art while remaining practical for everyday use.

III. State of the Art

Motion-based gaming has significantly evolved over the past two decades, with early systems relying on infrared and accelerometer-based tracking. Modern approaches incorporate sensor fusion, machine learning, and vision-based tracking that enhance interactivity and accuracy. Our project integrates IMU-based motion, tracking, Kalman filtering, Bluetooth Low Energy (BLE) communication, computer vision-based player tracking, and Panda3D graphics engine.

A. Overview

Our project consists of two main nodes: a computer that runs the game, and a remote that detects gestures to send to the game. The computer must run MacOS, and requires Python 3.8.10. Four processes run on the computer. The main game is constructed with Panda3d graphics engine. This process creates three subprocesses: one for BLE reception, one for OpenCV camera-based position detection, and one for speech recognition. These subprocesses send data via sockets to the main process. The remote consists of an ESP32-C6 microcontroller connected using a Qwiic connector to a Sparkfun 9dof IMU. The ESP32-C6 runs one program that collects data from the IMU via I2C, packages the data, and sends it to the computer via Bluetooth Low Energy.

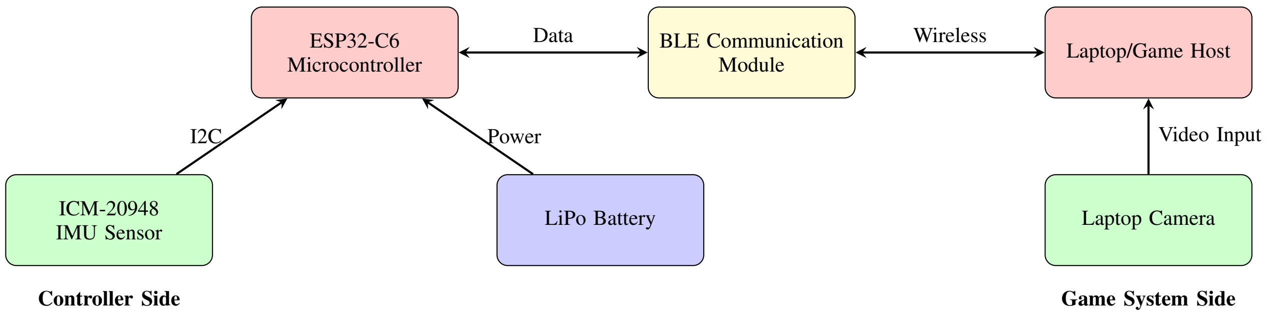


Fig. 1. Hardware system architecture of the Not Wii Bowling controller and game platform showing key components and communication pathways.

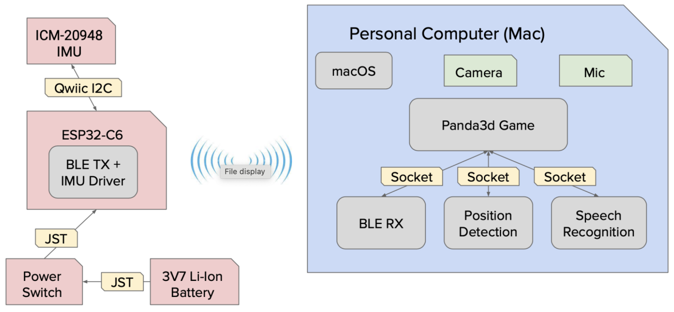


Fig. 2. High Level Design

B. Motion Sensing Technologies in Gaming

The earliest modern gaming system was Nintendo's Wii Remote, which utilized a three-axis accelerometer and infrared tracking to detect movement [1]. Although effective for gross motion detection, the Wii remote suffered from limited precision and motion drift, which necessitated frequent recalibration. Our system aimed to improve upon this by integrating an ICM-20948 Inertial Measurement Unit with Kalman filtering for real-time drift correction.

Another notable motion-tracking system, the Microsoft Kinect, combined depth sensing and computer vision to track user movements, eliminating the need for hand-held controllers [2]. Although Kinect was highly accurate in full-body motion tracking, the requirements of a dedicated processor and significant memory overhead made it very computationally expensive. To address these challenges, our system takes a hybrid approach by combining IMU motor control with OpenCV-based vision tracking for player positioning.

C. Data Filtering and Sensor Fusion

The ESP32-C6 micro-controller is responsible for acquiring real-time motion data from the ICM-20948 IMU sensor, which includes a three-axis accelerometer and gyroscope. The IMU transmits data to the ESP32 via I2C communication, and the ESP32 then sends the processed motion data via Bluetooth Low Energy (BLE) to the laptop on which the game is running.

The accelerometer measures linear acceleration along the X, Y, and Z axes, while the gyroscope records angular velocity. However, raw IMU data is susceptible to sensor drift and noise, making real-time tracking unreliable without additional processing. To address these issues, Kalman filtering is applied to smooth the IMU data and improve the accuracy of motion tracking.

In addition, ESP32 applies orientation correction using gyroscope data, updating the roll, pitch, and yaw of the device. Gravitational effects are compensated by the system to ensure that the expected acceleration values are more reflective of the actual motion.

A Kalman filter is implemented with the purpose of reducing noise and improving motion accuracy for the accelerometer readings. Each axis of acceleration (X, Y, Z) and gyroscope data (roll, pitch, yaw) is processed using an independent Kalman Filter instance, allowing optimal correction for each motion parameter.

Predict Step: The next state of the system is estimated by the filter based on existing data using a known-state transition model, which is typically represented by a set of linear equations. The filter output is the estimated position, velocity, or orientation before the reception of new sensor data

Correction Step: When the new sensor readings arrive, the filter compares the current state with the predicted state. A Kalman gain is computed and the estimate is updated by weighing both the predicted state and the new sensor input, minimizing the impact of noise.

The process noise (Q) and measurement noise (R) values are tuned to balance responsiveness and stability. The system becomes more adaptive to fast motion with higher Q values. Higher R values prioritize stability over rapid state changes. In our system, we experimentally set $Q = 0.025$ and $R = 0.5$, ensuring smooth motion tracking without excessive lag.

D. Communication Protocols

Bluetooth Low Energy (BLE) is a wireless communication protocol designed for low-power, short-range data transfer. Unlike classic Bluetooth, which is optimized for continuous data streaming, BLE is ideal for motion-based tracking applications due to its intermittent, low-latency transmissions.

Our system employs BLE to transmit real-time motion data from the IMU to the laptop, where it is used for in-game physics calculations. The BLE communication is structured as follows:

- Peripheral Role: The ESP32-C6 functions as a BLE peripheral, broadcasting motion data packets.

- Central Node: The laptop acts as the BLE central device, subscribing to two distinct characteristics for acceleration and gyroscope data.
- Packetized Data Transfer: IMU readings are encapsulated in a custom BLE format and transmitted at optimized intervals to balance low latency (10-20 ms) and power efficiency.
- Automatic Reconnection: In case of packet loss, the system implements an auto-reconnect mechanism, ensuring a seamless and stable data link.
- Buffered Data Handling: To handle network instability, recent IMU readings are buffered, preventing sudden motion lag in-game.

Additionally, our implementation supports data transmission via sockets, achieving smooth integration with the game engine. This architecture provides a robust and responsive motion tracking system without significant power overhead.

E. Computer Vision for Motion Tracking

The system employs OpenCV based motion tracking to detect the player's lateral position. Unlike high-end tracking solutions that rely on specialized hardware, our approach uses a standard laptop camera for cost-effectiveness and accessibility purposes

The tracking system processes live camera input to detect objects of interest using a deep learning face detection algorithm. The model, implemented with OpenCV's DNN module, loads a pre-trained Caffe-based SSD for object recognition. After detection, the centroid of the bounding box is computed and tracked across frames using centroid tracking. This implementation outputs positional updates to determine the player's position relative to the frame.

For more robust data handling, our system applies frame resizing and pre-processing techniques to improve computational efficiency without significant data latency. If tracking loss occurs, the system dynamically re-initializes detection to maintain continuous motion tracking. This vision-based approach helps us achieve precise player movement detection without the need for external sensors.

F. Speech Recognition in Gaming

Voice interaction has become a critical part of modern gaming as platforms such as Xbox Kinect and Amazon Alexa incorporate voice control to provide a better user experience. Our integration utilizes real-time speech recognition to streamline player interaction with Python's Speech Recognition library. The built-in library takes advantage of Google's Web Speech API for transcriptional applications of voice commands

In contrast with traditional text-based inputs, the game allows players to register their names and issue voice commands with the purpose of creating a more intuitive experience. During startup, the game continuously listens for user input and displays recognized player names within the game environment. Noise suppressing is utilized to maintain higher accuracy in noisy environments. In addition, the system has fallback mechanisms implemented to prompt uses for reattempts or manual text input in case of voice recognition failure. A standard microphone is sufficient for effective speech recognition eliminating the need for additional hardware or cost.

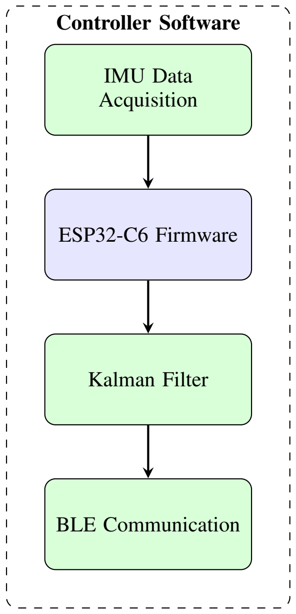


Fig. 3. Controller software architecture showing data flow from motion sensor through processing to BLE transmission.

G. Game Mechanics and Physics Simulation

The game is built based on the Panda3D game engine, which provides scene rendering, physics, and collision detection handling. The mechanics of the game are designed to simulate realistic ball movement, lane interactions, and pin collisions. The core of the physics system is implemented in the mechanics module, which is responsible for managing ball trajectory, spin, speed, and collision handling. The ball movement is determined by player input, which is derived from IMU readings during the swing and transmitted via BLE. The game engine applies linear and angular forces to account for rolling friction and pin interactions. In detail, the ball's acceleration is determined by IMU motion readings adjusted based on the filtered gyroscope readings. Furthermore, each bowling pin is assigned a collision capsule, for which collision events trigger knockdown animations determined by impact force vectors. In order to maintain a realistic environment, the ball's movement is restricted in lane boundaries while lateral motion is controlled by video input.

In addition, the logic module manages player turns, scoring, and game progression. The system is able to track individual frames and assign scores based on standard bowling rules. After a roll's outcome is recorded, the game resets automatically when the frame is completed. The scoreboard updates dynamically to reflect player performance based on the completed frame.

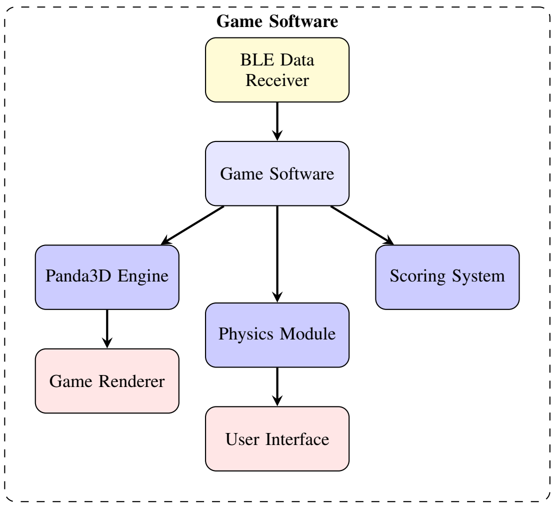


Fig. 4. Game software architecture showing data flow from input sources through game logic to rendering and user interface.

IV. Methods and Results

A. Methods, Problems, and Workarounds

The development process involved multiple revisions in order to address issues such as hardware compatibility, system performance, and developmental constraints. The team used practical testing and scenario validation to refine the system for reliable motion tracking and steady performance.

B. IMU Selection and BLE Communication

Our initial attempt involved using a Raspberry Pi Pico W with its corresponding IMU module for motion data readings. However, pull-up resistor issues on the Pico W resulted in a highly unreliable establishment of our I2C protocol. As a solution, the group decided to pivot to the ESP32-C6, which resolved the issue effectively and provided several advantages. This new approach led to a more compact design, easier integration, and improved BLE disconnection handling. The ESP32-C6's built-in BLE support allows for seamless communication with the node running the game, which reduces connection dropouts and improves data consistency.

C. Code Architecture and Implementation

Our implementation of Not Wii Bowling follows a modular, component-based architecture that prioritizes code maintainability, reusability, and separation of concerns. The system comprises several well-defined modules with clear responsibilities and interfaces, allowing for independent development and testing of components.

1) Architectural Patterns: The software architecture employs a hybrid approach combining elements of the Model-View-Controller (MVC) pattern with event-driven programming. The core game logic serves as the Model, managing game state and scoring rules; the Panda3D rendering system functions as the View, displaying the 3D environment; and the Controller responsibilities are distributed across various input handling components that process motion data and user commands.

The Core Game Logic module (`game_logic.py`) encapsulates all rules and state management for the bowling game using a clean, data-oriented design. Enumerations for player turns and data classes for frame data points towards our focus on type safety and code readability. A Hardware Interface Layer is used to provide abstraction over sensor inputs via socket-based communication channels. These channels decouple the hardware implementation from game logic and allow for easier testing and hardware substitution. The Panda3D rendering and physics engine, which is functionally independent from the game logic, manages scene rendering, collision detection, and physics simulation. Finally, User Interface Components handle scoreboard display and user interaction through a component-based approach desired for scalability and testing ease.

2) Communication Patterns: The system leverages multiple communication mechanisms to facilitate inter-component coordination. Event-Based Communication through Panda3D's messenger system enables loosely coupled interactions between components, creating a responsive system where modules can react to events without direct dependencies. Socket-Based Communication enables real-time data transfer between the controller hardware and the game application using a client-server architecture that enhances system robustness through buffering and error handling. Time-based operations are managed via Panda3D's task scheduler, allowing for precise timing control in animations and game state transitions.

3) Error Handling and Resilience: Error handling is integrated throughout the architecture to ensure system stability. Components incorporate graceful degradation mechanisms, exemplified by the position tracker's ability to recover from tracking loss. Resource management procedures ensure proper release of system resources, with explicit socket and process termination routines implemented in the cleanup methods. Exception handling is strategically applied around critical I/O operations to maintain system stability even when external components fail or unexpected conditions arise.

D. Ball Speed Modulation Algorithm

In order to make the game more realistic, we felt that it is important that the player be able to change the speed at which they roll the ball using harder or softer swings just as in real bowling. While we started out using acceleration values, we quickly realized that no matter how much we filtered the data, we could not achieve consistent enough values to tie swing power with ball speed. Instead, we decided to simply use the angular velocity about the y-axis. We then created an algorithm that processes these incoming angular velocity values and applies scaling and thresholding techniques in order to map to a value representing the time the ball is in motion.

To detect the presence or lack of a bowling motion, we needed some way to keep track of historical angular velocity values. The mechanism through which we were able to track past values was a circular buffer. I tested multiple buffer sizes before settling on size five, as it seemed to result in fewer false positives (detecting swing when it should not have) and fewer false negatives (not detecting swing when it should have). This buffer would pop the oldest value every time a new value was read from the socket.

The key to identifying intentional swings versus unintentional swings is detecting consecutive angular velocity values above a certain threshold. Therefore, it is important to carefully choose how many consecutive values we want to consider. Requiring more consecutive values above the threshold decreases the noise of the signal. At the same time, we still want our algorithm to be sensitive enough to detect quick swings.

Once the threshold has been reached for the desired number of consecutive values, the average of said values is taken to be the unscaled power value. Then exponential scaling is applied to amplify the difference between hard and soft swings. Finally, this scaled power is mapped to a roll time that can take on values between 1 and 8. This roll time is then passed to the function that handles moving the ball graphically.

E. Game Deployment and Performance Optimization

The first design involved deploying the game on a Raspberry Pi 4. The team encountered significant performance limitations due to the computational load of the Panda3D engine. An attempt was made to optimize the game by reducing graphical details which yielded unsatisfactory results in terms of game performance and speed.

A proposed solution was the containerization of the game using Docker for easier deployment across systems. Our motivation behind this approach was to simplify dependency management and achieve portability of the game. The goal was to create a consistent environment across different setups, which would, in turn, enable deployment without manual reconfiguration. After some time developing this solution, we realized that graphical applications cannot run within Docker containers without using X window forwarding. This ended up requiring so many external dependencies that it detracted from the purpose of using Docker, which is to improve portability.

Finally we decided to run the game without any containerization on a laptop using the MacOS operating system. We composed a shell script that would download the necessary version of Python and set up the environment. We then composed a second script that acts as an entry point to run the game.

F. Camera Selection for OpenCV Motion Tracking

An external camera was initially considered for OpenCV-based tracking on the RPi4. Once we shifted to using a laptop to run the game, we were able to shift our stack to instead use the built-in laptop camera. The change of hardware maintained sufficient tracking accuracy, minimized hardware requirements, and enhanced usability.

G. Testing, Results, System Performance and Error Handling

The project included a series of tests to evaluate system performance, usability, and robustness. The results confirm that our system achieves low latency motion tracking, stable BLE connection, and an intuitive gameplay experience. This section outlines the testing methodologies and insights gained.

1) Motion Tracking and Kalman Filtering Optimization: A Kalman filter is used with the purpose of refining IMU readings and reducing sensor noise. The filter calibration involved an incremental approach, where complexity was progressively increased into motion testing. This approach allowed our team to consistently evaluate performance and fine-tune the filter according to our needs

In the first stage, raw IMU readings were recorded without filtering using a single speed to establish a performance baseline. This step exposed one of the key issues throughout the project, accelerometer drift. Accelerometer drift is the result of sensor readings deviating over time. During a later stage of development, different speeds were introduced while keeping angles constant to analyze the filter's response. Several adjustments were made in terms of sensitivity and precision to achieve smoother transitions. Finally, gravity components which influenced accelerometer readings required further tuning to maintain consistency across different speeds. While we were able to successfully implement the Kalman filter for accelerometer data, we were not able to achieve a result that would allow for successful swing detection. Therefore, our final product instead uses gyroscope data. However, the Kalman filter is still built in and our code could easily be modified to support acceleration-based detection if further filtering results in a usable product.

2) Bluetooth Low Energy (BLE) Performance Testing: The stability of the BLE system was evaluated using tests focusing on latency, reliability and throughput. Our goal was to assess the server's ability to handle low latency tracking with a stable connection. Latency was measured using microsecond timestamps in which transmission delays were recorded over multiple instances. The reliability of the connection was validated by measuring packet loss rate and RSSI values at increasing distance. The throughput was assessed with varying packet sizes through which the maximum sustainable transmission rate was established for normal operating conditions.

3) Computer Vision Position Tracking Performance Testing: To validate the vision-based position tracking prior to integration in the main application, a specialized test was implemented to isolate the component for performance measurements. Through this approach the team was able to quantify the tracker's capabilities and reliability in preparation for full integration of the system.

The test framework used a mock socket server to simulate the main game's connection interface and collect detailed metrics. We used a controlled environment with a MagicMock interface for evaluation of movement patterns. This method allowed us to detect underlying issues before integration.

This analysis, which was based on a 30-second test session, showed a consistent data rate of 29.90 position updates per second, exceeding our expectation by a large margin. The tracker demonstrated a sufficient detection range, with normalized distance values from -184.00 to 857.00 units, which confirmed it ability to capture lateral movements expected for game play. The system showed appropriate sensitivity to movement, with an average frame-to-frame movement detection of 99.01 units and maximum detected movements of 1038.00 units.

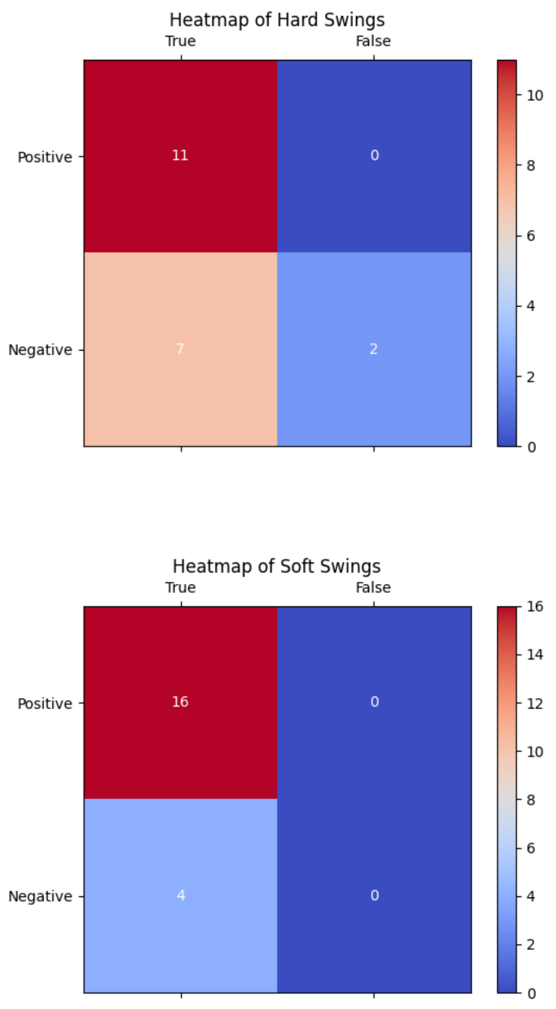
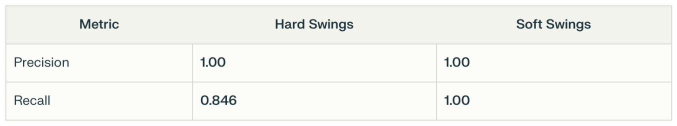


Fig. 5. Heat Maps for Hard and Soft Swings

4) Ball Speed Modulation Testing: To gauge the performance of the physical-swing power-detection algorithm, we set out to test how well hard swings were detected in comparison to soft swings. This test was fairly qualitative, as we used our eyes to determine how well the speed of the bowling ball correlated with the power of the swinging motion we made with our arms. The test consisted of twenty hard swings and twenty soft swings. We laid out four different potential result categories that each test would fall into. This test hinges on combining two different metrics: detection and classification. The detection metric is defined by whether or not a ”bowling” motion has occurred. The classification metric is defined by whether or not the resulting speed of the ball matches the power applied during the physical motion. The heat maps displayed have four quadrants, each corresponding to an outcome.

- A ”True Positive” test case means that the motion was correctly detected and classified.
- A ”True Negative” test case means that the motion was correctly detected but incorrectly classified
- A ”False Positive” test case means that the motion was detected when it should not have been.
- A ”False Negative” test case means that the motion was not detected when it should have been.



Metric	Hard Swings	Soft Swings
Precision	1.00	1.00
Recall	0.846	1.00

Fig. 6. Precision and Recall Figures for Test Data

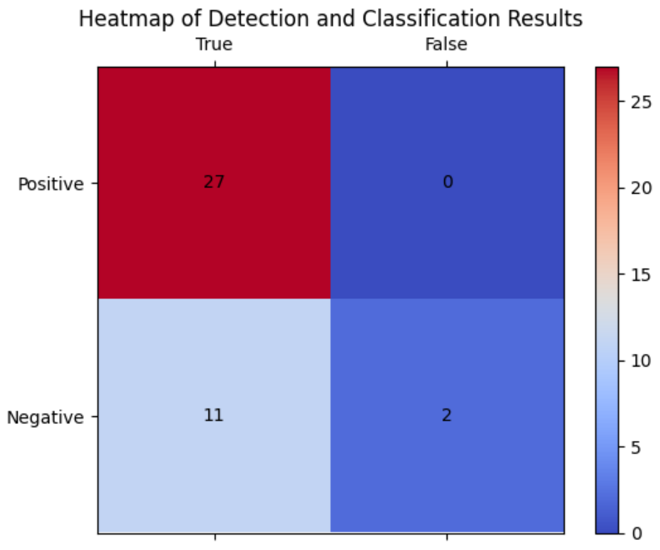


Fig. 7. Heatmap of Combined Results for Hard and Soft Swings

From these results, we can calculate some powerful metrics. Precision is defined as the ratio of True Positives to the sum of all Positive outcomes, both True and False. Recall is defined as the ratio of True Positives to the sum of True Positives and False Negatives.

Furthermore, these results allow us to glean some interesting insights about our algorithm. First of all, the algorithm performs better in both detecting and classifying soft swings in comparison to hard swings. No false negatives were recorded for soft swings, which means our algorithm is perfect at detecting the swinging motion when the user swings the remote softly. Additionally, soft swings resulted in a higher True Positive count (16) relative to hard swings (11).

Both hard swings and soft swings had a perfect precision score. This can be attributed to the absence of any False Positives across all trials. On the other hand, hard swings had a lower recall score due to the presence of False Negatives in the results. This leads us to hypothesize that the performance of our algorithm depends on the amount of time that the remote control is in motion. Softer swings are in motion for a longer period, whereas hard swings are in motion for a shorter period. Our algorithm struggles in the latter case.

Overall, the algorithm performs relatively well. Anyone who has played Wii Sports Bowling will remember how inconsistently the motion-controlled remotes behaved. Additionally, Wii Bowling used the release of a button to detect the presence of an intentional bowling motion. We set out to achieve detection without using any buttons, instead relying on characteristics of the motion. In the end, our testing shows successful detection and classification, with 67.5% or approximately 2/3 of trials resulting in a True Positive.

5) Usability Testing: Peer conducted testing session were conducted to evaluate user experience, ease of setup, and game responsiveness. During the final stage of development, participants were asked to set up the game, interact with the motion tracking features, and provide feedback on the overall game experience. A primary observation was that the initial setup process on a laptop was complicated. The setup required the manual installment of multiple dependencies. For this reason, the setup process was simplified to streamline installation. In terms of game play, motion tracking was generally found to be responsive. However, some users experienced minor uses in terms of accuracy which were likely attributed to different play styles.

Additional feedback suggested that the GUI had room for improvement with clearer in-game instructions and visual clues for player positioning. The received feedback also suggested that it would be beneficial to add customization options such as multiplayer features and different bowling scenes. Overall, the usability testis provided insight into the strengths of our system as well as aspects with room for improvement.

V. Conclusion

This project successfully implemented a camera-based motion-tracking and remote gesture-based bowling simulation using Bluetooth Low Energy (BLE), OpenCV, and a physics-based game engine in Panda3D. The system demonstrated low-latency motion detection, effective player interaction, and a realistic game experience. The custom BLE server provided a stable communication with minimal packet loss and low overhead. Despite our implementation's ability to process motion input accurately and adapt to player speeds with filtering, certain limitations remain.

Future work on the Not Wii Bowling system could explore several enhancements to improve user experience and improve overall performance. The implementation of multiplayer functionality over network connections could allow players to compete remotely instead of limiting interactions to in-person game play. This would require additional client-server architecture which could not be achieved within a reasonable timeline for our project purposes. In addition, the game could be expanded to include other game modes such as time challenges or alternative scoring systems, which would appeal towards different player preferences. Finally, the implementation of AI components would enable both single-player challenges against virtual opponents of various difficulties. Such a system could utilize existing motion and tracking data for player profiling and personal training recommendations

In conclusion, the project demonstrated a solid foundation of motion-based gaming applications and highlighted key areas for future enhancement and broader usability. The modular architecture we developed provides a flexible framework for implementing these future improvements while maintaining system stability and performance.

References

[1] S. Park and J. Lee, ”Nintendo Wii Remote: Sensor fusion and applications,” *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1145-1152, 2009.

[2] Z. Zhang, ”Microsoft Kinect sensor and its effect,” *IEEE MultiMedia*, vol. 19, no. 2, pp. 4-10, 2012.

[3] G. Welch and G. Bishop, ”An Introduction to the Kalman Filter,” *University of North Carolina at Chapel Hill Technical Report*, 2001.

[4] S. Julier and J. Uhlmann, ”A new extension of the Kalman filter to nonlinear systems,” *Proceedings of SPIE 3068, Signal Processing, Sensor Fusion, and Target Recognition VI*, 1997.

[5] J. Han, S. Lee, and H. Park, ”Comparative analysis of Bluetooth Classic and Bluetooth Low Energy: A power and latency perspective,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1685-1702, 2018.

[6] R. Prasad and M. Walker, ”Voice-based user interfaces for gaming applications,” *International Journal of Speech Technology*, vol. 23, no. 2, pp. 117-128, 2021.