# ***Style Tracking Expressive Pad System***

*Arcade Dance Rhythm Game Leveraging CV Pose Detection, Depth-Optimized Optics, and Time-multiplexed Illumination*



## Group 8 Authors

| | | | | |
|---|---|---|---|---|
| Christopher Solanilla | Jani Jon Lumibao | Kaila Peeples | Andres Abrams | Blake Whitaker |
| *Computer Engineering* | *Computer Engineering* | *Photonics and Science Engineering* | *Computer Engineering* | *Electrical Engineering* |

## Reviewer Committee:

| | | |
|---|---|---|
| *Dr. Stephen Eikenberry* | *CREOL* | *Professor* |
| *Dr. Saleem Sahawneh* | *ECE* | *Lecturer* |
| *Dr. Justin Phelps* | *ECE* | *Professor* |

## Mentor:

*Dr. Chung Yong Chan*
*Dr. Aravinda Kar*

# 2. Project Description

## 2.1 Motivation and Background

In recent years, rhythm games have surged in popularity among both casual and competitive gamers. Titles like *Dance Dance Revolution (DDR)*, *Pump It Up (PIU)*, and *StepManiaX* offer not only fast-paced gameplay but also unique forms of physical interaction that make them stand out from traditional video games. Among these, *StepManiaX* has played a particularly influential role in the inception of our project, as it is actively available at a UCF campus restaurant and regularly enjoyed by students.


***Figure 2.1*** *StepManiax on UCF at Knightros*

For readers unfamiliar with rhythm games: players typically choose a song, and as the music plays, visual cues (usually arrows) scroll on the screen toward a target zone. Players must step on corresponding arrows on the dance pad in time with the music. Successful timing earns points, while misses break combos and reduce scores. In games aforementioned, players are often using their feet to step on panels that are labeled with directional arrows. In DDR it is with the up down left right arrows, StepManiaX has the same but with an additional center arrow, and Pump it Up has diagonal arrows instead of the 4 up down left right arrows with the center note as well.

As a lifelong fan of rhythm games, I've always aspired to create a game that blends the fast-paced footwork of DDR and PIU with innovative mechanics that reward not only timing accuracy but also expressive performance. This passion became the foundation for our senior design project: a custom-built rhythm arcade machine that reimagines the traditional dance pad format. Our system features a unique 9-panel layout, combining the four cardinal directions, four diagonals, and a center panel. This effectively merges the core mechanics of DDR and PIU into a new hybrid experience.

To take it a *step* further, we are integrating a computer vision system capable of analyzing player movement during gameplay. This system detects whether the player completes the charts with minimal effort or performs dynamic, stylish movements such as spins or arm gestures. Players who demonstrate expressive flair are rewarded through a secondary metric we call the Style Score, adding a new dimension to gameplay that celebrates both precision and creativity.

Our team brings a diverse range of skills and experiences to the project. I contribute both a strong passion for rhythm games and hands-on experience from developing a basic dance pad prototype in the past. One of our teammates has a background in dance, offering valuable insight into expressive movement and physical design. Another teammate serves as the president of the UCF Esports Club, providing a competitive gaming perspective that helps shape our gameplay mechanics and balance. All three of us are Computer Engineering majors with solid experience in hardware integration and software development. Additionally, the teammate with a dance background is well-versed in PCB design, making them an asset for developing the input hardware. Our fourth member, a Photonics Science and Engineering student, brings specialized expertise in optics. This makes her an asset for enhancing the performance of our computer vision system through lens design or optimization.

We believe this blend of technical, creative, and performance-oriented backgrounds makes our team uniquely positioned to create a rhythm game that is both entertaining and technically ambitious. By combining game development, embedded hardware, and real-time computer vision, we aim to push the boundaries of traditional rhythm games. Ultimately, we envision this system not just as a school project, but as a potential commercial product suitable for both arcade and home use. Overall, our project is a tribute to the genre we love and an innovative leap forward in how rhythm games are played.

### 2.2 Existing Product/Past Project/Prior Related Work

### 2.2.1 Dance Dance Revolution

Dance Dance Revolution (DDR), developed by Konami in 1998, is one of the most iconic rhythm games in the world. Players step on a 4-panel dance pad — with up, down, left, and right directional arrows — in time with scrolling on-screen cues synchronized to music. DDR is widely recognized for its role in popularizing rhythm games globally and has been featured in both arcade and home console formats. It uses pressure-sensitive panels and a scoring system based on timing accuracy, rewarding "Perfect," "Great," or "Miss" for each input. However, the game focuses purely on foot-based precision and lacks a scoring component for stylistic or expressive movement. Players who still choose to complete charts with style are called “free stylers” and are highly respected in the community for completing easy to mid level charts with very hard expressive and complicated movement. This movement can range anywhere from spinning, to swaying the arms, to handstands and break dancing.

### 2.2.2 Pump It Up

Pump It Up (PIU), developed by Andamiro in 1999, is a 5-panel dance rhythm game that includes four diagonally placed panels and a center panel. PIU emphasizes freestyle movement more than DDR and is especially popular in South Korea. While PIU retains the same timing-based scoring mechanics as DDR, it introduces more physically varied and complex choreography due to its diagonal input layout. Nonetheless, PIU still lacks any integrated camera system or style-based scoring . Just like DDR, PIU has the same respected players who complete charts with freestyling despite still not being rewarded for doing so in game.

### 2.2.3 Dance Around

Dance Around is a rhythm game similar to Dance Dance Revolution but instead of using pressure-sensitive dance pads, it relies solely on a camera-based motion tracking system. This is done by using VisionPose to generate a 3D model of the player’s body and assessing their dance performance based on their ability to match the poses given and their own expressive movements[2]. During the game, players are prompted to mimic target poses displayed on the monitor, with visual cues that indicate the proper hand or foot placement. However, customer feedback has highlighted several limitations of this system. Customers have stated that the game has consitently misevaluated full body movements and is only capable of capturing the hand and foot positions of the players reliably[3]. Additionally, customers have noted that the calibration process could be overly lengthy, which detracts from the ease of use.

### 2.2.4 Dancerush Stardom

Developed by Konami, DANCERUSH STARDOM is a freestyle rhythm dance game that eliminates the traditional dance pad structure in favor of a large pressure-sensitive surface. The game uses a camera system to provide feedback and record gameplay, but the scoring is still based on foot movement across a large flat pad with visual indicators for steps and slides. The game promotes freestyle dancing, including spins and slides, and is considered more modern and expressive than DDR. However, like Dance Around, it does not feature a true pose recognition or performance grading system. Flair and dance expression are encouraged but not quantitatively rewarded within the game mechanics.

### 2.2.5 StepManiaX

StepManiaX is a rhythm game inspired by DDR and PIU and developed by the creators of StepMania. It uses a 5-panel pad layout with center, up, down, left, and right panels, and is designed for high durability and fitness applications. StepManiaX includes a touchscreen interface and modernized music selection UI, with content designed to be accessible for both casual and serious players. While it modernizes the user experience and supports a wide range of difficulty levels, it retains the traditional scoring focus on timing accuracy. There is no implementation of camera-based tracking or any scoring system that rewards visual expression or dance style beyond note timing.

***Figure 2.2*** *Illustration prototype of StepManiaX, a 5 panel Dance Rhythm Game Arcade Cabinet[6]*

## 2.3 Project Goals

The main goal of this project is to design and build a self-contained arcade-style rhythm game system that includes both the physical dance pad hardware and the video game software. The system will feature a 9-panel layout to allow for expanded gameplay mechanics. In addition to foot-based input, the game will include a computer vision system that evaluates the player's expressive movement using a live camera feed. Our goal is to create an experience that rewards both precision and performance, allowing players to interact with the game through both steps and body movements. The final product will include a working rhythm game engine, a functioning pad-to-PC interface, custom chart creation tools, and vision-based style scoring.

### 2.3.1 Hardware

**Basic Goals:**

- ensure responsive and accurate force detection using force-sensing resistors
- design a singlet aspherical lens to maximize depth of field and image sharpness
- enhance lightning efficiency by implementing synchronized time-multiplexed LED zones
- develop 9-direction interactive arrow pads with RGB LED feedback support responsive gameplay

**Advanced Goals:**

- improve PCB computational time and reduce system cost
- refine pad design for compactness and portability
- improve image clarity by optimizing aspherical lens design.
- optimize LED beam angles and panel placement so there is minimal shadowing across the full motion range

**Stretch Goals:**

- expand system capabilities through wireless connectivity
- improve portability through a foldable dance pad design
- investigate dual-focal-path optics for enhanced depth sensing.
- develop a lighting system that dynamically adjusts LED intensity based on player position

### 2.3.2 Software

**Basic Goals**

- develop a custom rhythm game engine tailored to the 9-panel system
- design an intuitive interface that allows players to select songs and view their performance results
- design the game system to be able to process computer vision locally
- reward the player for executing certain poses during gameplay

**Advanced Goals**

- include a built-in editor that allows players to create and customize their own charts
- improve the accuracy for pose detection utilizing a special lens
- offer a default library of preloaded songs for immediate play

**Stretch Goals**

- allow players to log in, save scores, and compete on a global leaderboard.
- create a mobile app for users to easily login and track data during gameplay
- add configurable startup modes, including an arcade mode with limited songs per play or personal use with no restrictions

### 2.4 Objectives

We aim to complete the following specific tasks over the course of SD1 and early SD2. These objectives are listed in the rough order we plan to execute them, starting from component acquisition and hardware assembly to basic software integration and vision feature prototyping.

1. **Acquire all essential components:** order FSR sensors, RGB LEDs, microcontroller (MCU), Raspberry Pi, USB connectors, power regulators, and camera module based on BOM list.

2. **Design and fabricate the dance pad platform:** cut and assemble the frame using plywood and aluminum, and mount transparent top layers and non-slip base.

3. **Design a custom PCB schematic:** using KiCad or similar, create and route a PCB that connects FSR sensors, LEDs, and MCU. Ensure enough GPIOs and power regulation are included.

4. **Order and assemble PCB:** send design for fabrication through JLPCB or cheaper manufacturers and assemble the physical board with soldered components and headers.

5. **Write and read microcontroller firmware:** design, test, and read analog signals from FSRs and convert them into digital keypress events via USB HID protocol.

6. **Test PCB + FSR response time:** use a debug script to confirm low-latency response (<10ms) when pressing panels.

7. **Integrate pad input with PC:** confirm that pressing physical panels triggers keyboard inputs correctly on a connected computer (e.g., using a diagnostic test page).

8. **Initialize the GitHub repository:** create Github repository for all members and set up version control for both hardware schematics and game software.

9. **Begin a Godot project for the rhythm game engine:** set up a basic Godot 4.3 scene with a UI, note spawning, and music synchronization framework.

10. **Map keyboard inputs to note triggers:** use Godot and test gameplay using manual inputs (before full integration).

11. **Design and implement a basic UI:** create song selection, results screen, and audio-visual feedback for note accuracy (e.g., Perfect/Good/Miss text).

12. **Write a chart parser and loader:** read external step charts from a JSON or custom format and spawn notes accordingly.

13. **Connect the pad to the game:** test real gameplay — stepping on the pad triggers notes and feedback in the game engine.

14. **Integrate camera with Raspberry Pi:** verify that real-time feed is available to the game via local network or USB interface.

15. **Use a pre-trained pose detection library:** use OpenCV, Media Pipe, or OpenPose to extract body key points from live video.

16. **Detect key Poses and Gestures:** aim to design the STEPS system to have the ability to detect specific poses from our list of poses including the Muscle Man pose, the What? pose, the Mantis pose, and the Samurai pose.

17. **Develop a basic Style Score system:** create a Style Score system that assigns bonus points based on detected expressive movements.

18. **Build a simple chart editor UI:** develop a chart editor that can allow the player or moderator to manually align step notes to music and export them for testing.

19. **Design a custom aspherical lens system:** design a custom lens system to resolve ~1mm features across a 40cm field of view at 1.8m distance.

20. **Generate accurate ray layouts in Zemax:** create layouts to evaluate chromatic aberration and field sharpness.

21. **Select a low-cost camera sensor:** buy a low-cost camera sensor then configure the optical system to minimize data throughput by reducing the field of view and processed resolution to ≤ 1.5MP.

22. **Prototype and test multiple lens configurations:** ensure lens and camera sensor are coaxially aligned and geometrically compatible.

23. **Design a time-multiplexed LED illumination system with zoned control:** ensure consistent full-body visibility of the player throughout dynamic motion.

24. **Quantitatively evaluate optical performance:** compare expected vs. actual image sharpness and distortion using test charts or calibration rigs.

25. **Document each milestone:** post photos, commit history, and GitHub issues, and ensure each part is demo-ready for SD1 and SD2 deliverables.

**2.4.2 Prototype illustration/Blueprint**

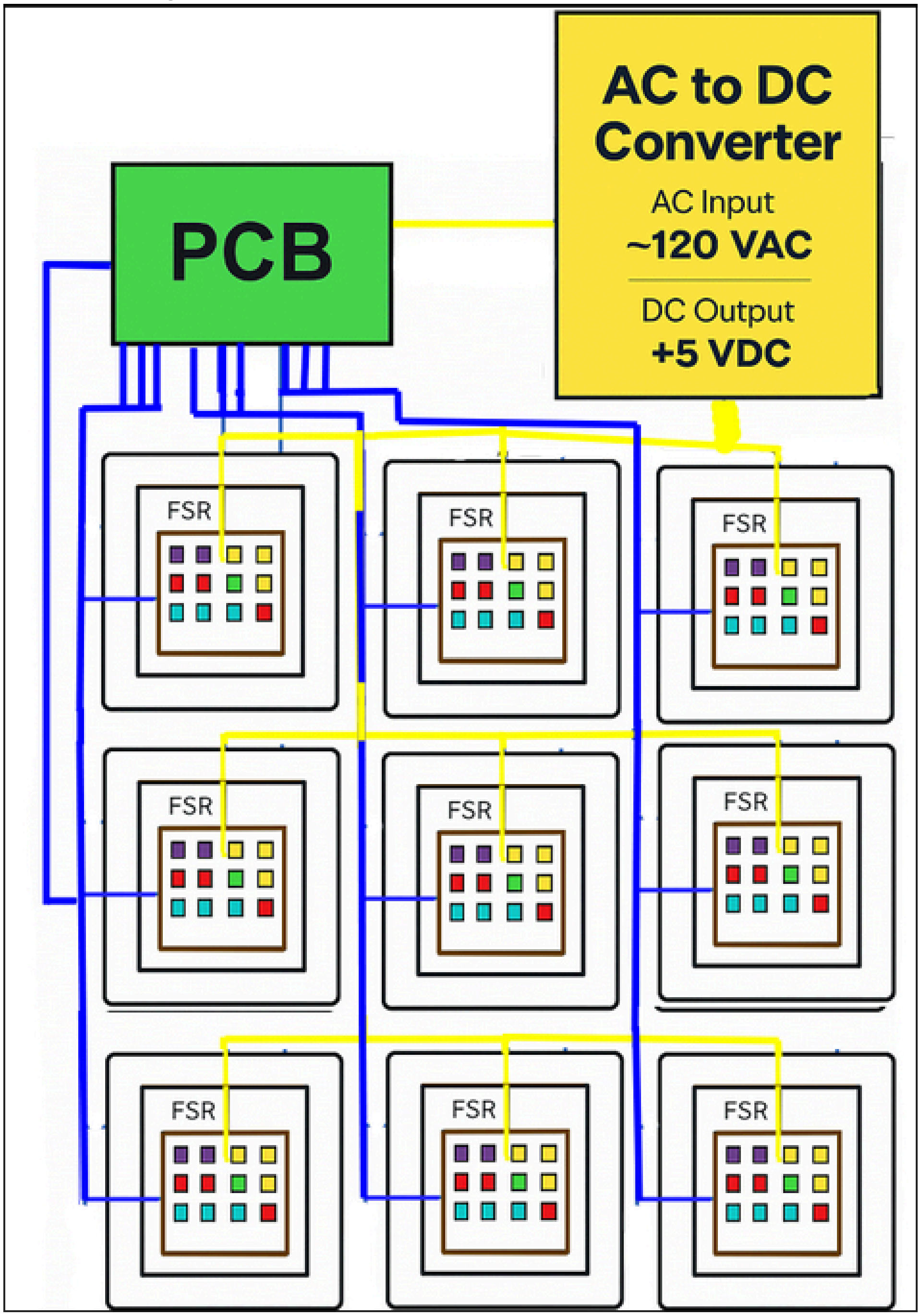



***Figure 2.3*** *Electronic Blueprint Prototype Illustration*

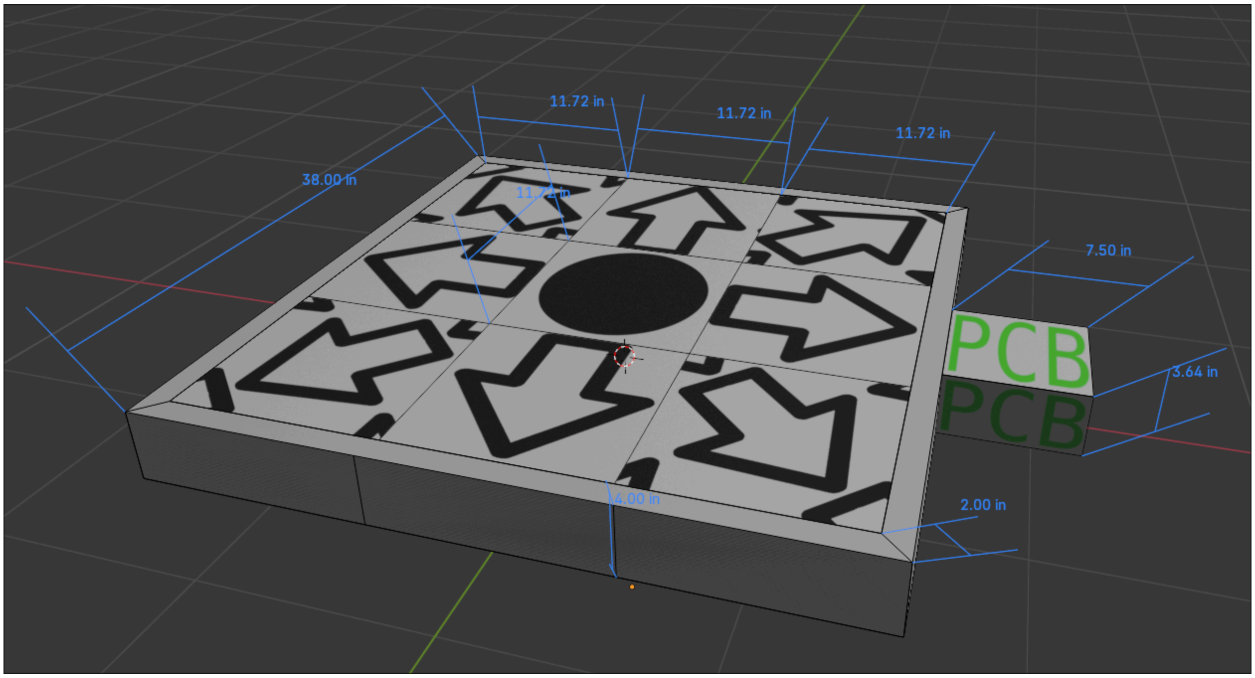



***Figure 2.4*** *Dance Pad 3D Model*

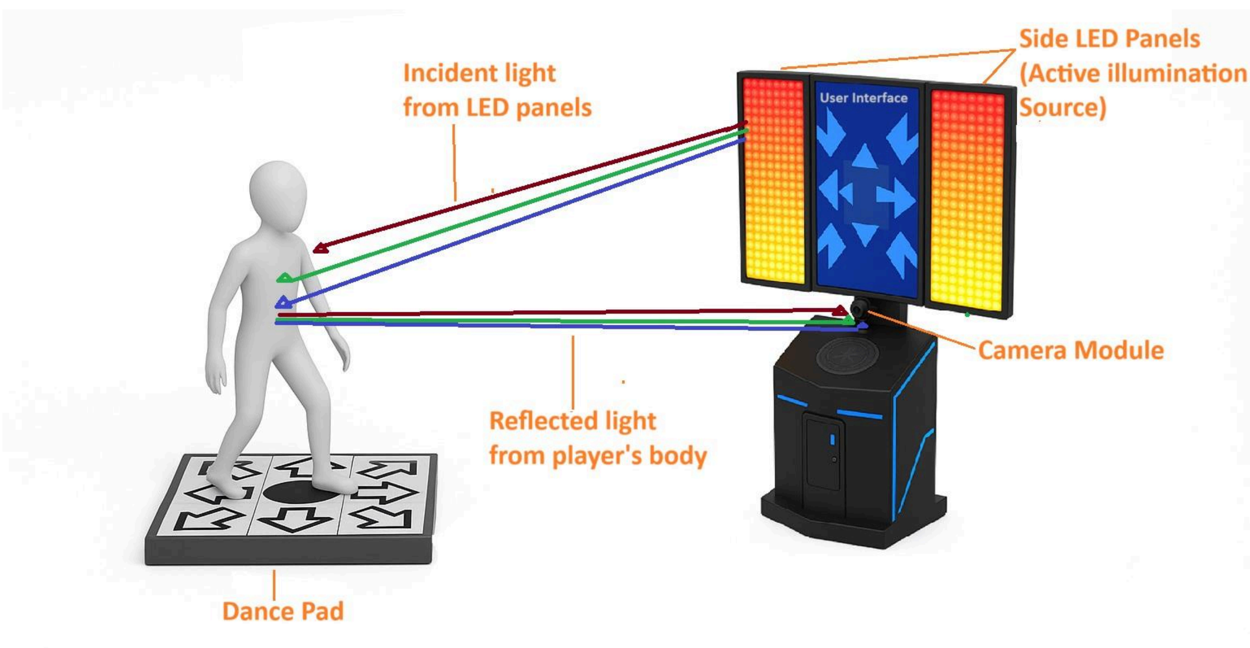



***Figure 2.5*** *Schematic of the optical and illumination system for full-body motion capture (3D base model generated using ChatGPT image tools; annotations and system overlays by author) .*

### 2.5 Project Features and Functionalities

The primary goal of this project is to develop a working proof-of-concept for a new kind of dance rhythm game that integrates physical input with computer vision-based expression tracking. The focus is on delivering a functional and demonstrable system that includes three core components: a responsive 9-panel dance pad, a custom rhythm video game engine, and a vision system that scores player movement based on style and expressiveness.

At the hardware level, the dance pad will use Force-Sensing Resistors (FSRs) beneath each panel to detect foot pressure and translate those inputs into digital signals via a microcontroller. The signals are interpreted as button presses in-game, allowing for responsive gameplay. Each panel will also include LED lighting to provide immediate visual feedback based on the game's state and player interaction. While commercial-grade materials like metal panels or acrylic overlays would be ideal, the immediate priority is to build a stable, functional pad using accessible prototyping materials to prove the input system works reliably.

The software side features a rhythm game engine tailored to the 9-direction input system. Players will step on directional pads in sync with music, guided by scrolling notes on screen. The game will evaluate the player's timing accuracy and display scores at the end of each round. Alongside traditional gameplay scoring, a connected camera system will assess the player's full-body movement using a pose detection library. Based on the amplitude, variation, and expressiveness of the player's dance, the game will generate a secondary "style score." This adds a creative and engaging layer of performance evaluation beyond pure timing.

A basic user interface will allow for song selection, score displays, and navigation through the system. A chart editor tool will also be included, enabling users to import music and design their own charts either manually or with the help of automatic generation tools.

Although not essential to the proof-of-concept, we also envision the long-term possibility of turning the system into a self-contained arcade-style unit. Features like a cabinet enclosure, co-op integration, player logins, and online leaderboards are considered stretch goals that could be implemented later with more resources. The project is designed with scalability in mind: the core technology should work independently, while leaving room for future upgrades to polish and deploy the system as a full commercial or open-source product.

#### 2.5.1 Style Score and Pose-Based Evaluation System

In addition to the traditional timing-based score system, our game introduces a novel secondary metric known as the Style Score, which rewards players for striking expressive and clearly defined poses at designated times during gameplay. Unlike freeform dance scoring, our system focuses on detecting static full-body poses that can be consistently recognized by a pose estimation algorithm.

We intentionally limit the scope of detection to distinct, predefined poses that can be reliably tracked in real-time using a single camera. This approach improves detection accuracy and reduces computational load while still encouraging expressive movement.

During gameplay, specific pose prompts will appear, similar to freestyle moments or bonus sections. These pose prompts will be displayed in a small icon area on screen during designated freestyle sections, giving players 2–3 seconds to match the target pose. If the player strikes the correct pose at the right time, they are awarded bonus points to their Style Score. This incentivizes physical creativity and rewards players who engage more fully with the visual performance aspect of rhythm games.

Each pose is chosen based on ease of detection, body separation, and iconic visual silhouette (all poses are mirrored when displayed to the player):

- **Muscle Man Pose**: Both arms raised as if you’re flexing biceps’
- **What? Pose:** Both arms out as if you don’t know about something
- **Point Up Pose:** Left arm behind the head, pointing out to a plane
- **Tough Guy Pose**: Crossed arms
- **Samurai Pose** : Legs wide apart, one hand near waist as if gripping a katana, other arm pointing forward or out.
- **Mantis Pose**: Right arm raised in front of the chest, left bent above the head, and right leg up with knee up
- **T Pose**: Both arms stretched straight out to the side (useful base for debugging).

Each pose will have keypoint thresholds that define acceptable angles and positions such as a certain limb being within a range of 20 degrees from the desired state. This will be validated using a pre-trained MediaPipe/OpenPose[5] model.

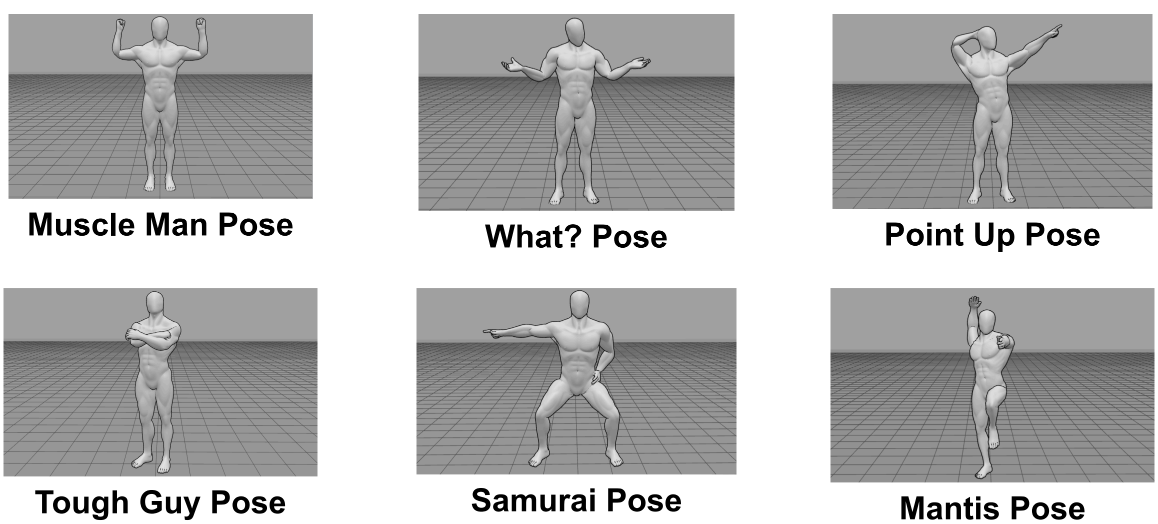



***Figure 2.6*** *Images of the tentative poses (modeled using PoseMy.Art)*

### 2.6 Requirements and Specifications

#### 2.6.1 Parts Specification Requirements Table

***Table 2.1*** *List of all of the project's engineering requirements and specifications. Highlighted in yellow are specifications that will be demonstrated*

| Parameter | Value |
|---|---|
| Overall System | |
| Active power consumption | ≤ 200 W |
| Dance Pad | |
| Size | ~34.5 x 34.5 x 2.5 in |
| Weight | ≤ 30 lbs |
| Cost | ≤ $300 |
| Printed Circuit Board (PCB) | |
| Size | ≤ 10 $cm^2$ |
| Display | |
| Frame rate | ≥ 60 fps |
| Resolution | ≥ 1280 x 720 pixels |
| Refresh rate | ≥ 120 Hz |
| Dance Pad Panels | |
| Size | ~10 $in^2$ |
| Input response time | < 10 ms (almost instantaneous) |
| Force range input | 0-100 N |
| Camera Module | |
| Pose identification accuracy | ≥ 95% detection accuracy for body/limb motion |
| Full-body coverage area | ≥400mm field width at 1m distance, no cutoff or blind spots |

| | |
|---|---|
| Power Supply Unit | |
| Input voltage from wall power via AC-DC converter | ≥ 12 V |
| Output power | ≥ 1.65W (≥ 0.5A at 3.3V) |
| LED Panel | |
| illumination uniformity of the player | ≥ 90% |
| Lighting response time | ≤ 10ms |
| Player visibility (shadow coverage) | Full Body at 1m |

### 2.6.2 Notes on Specifications

Overall System
For active power consumption, most typical dance arcade machines are around 100-200 watts, so for our system we aim to have set our limit to 200 watts.

Dance Pad
For increased portability and potentially affordability and maintenance, a size of about 34.5 x 34.5 x 2.5 inches and weight of at most 30 lbs would be optimal for the player and owner of the pad.

Printed Circuit Board (PCB)
As mentioned before in goals, we want to minimize the size of the PCB as much as we can while also making it easy enough to check for errors on the design.

Display
Whether it be a PC or game system, the system requirements are the same. In order to run our custom made game, the system must be able to run at least 60 frames per second, have a resolution of at least 1280 * 720 pixels, and have a refresh rate of at least 120 Hz.

Dance Pad Panel
Response time from the FSRs must be almost instantaneous and this is very important because rhythm games are dependent on timing of when commands are hit. There should be little to no latency when pressing the pads and triggering the FSRs. Moreover, the amount of force applied needed to trigger the FSRs must be 0-100 N so that even the smallest amount of pressure should be enough to send a signal back to the MCU.

Camera Module

To meet ≥95% tracking accuracy and full-body coverage, the camera must maintain sufficient resolution, wide FOV ( ≥ 400 mm at 1m), and minimal distortion. This ensures consistent player visibility without cutoff or blind spots.

Power Supply Unit

The pad is powered through a wall power cable which should be more than sufficient enough to power the pad. The input voltage of at least 12 V is required to efficiently power the whole pad, and we aim to output a power of at least 1.65 watts.

LED Panels

To ensure the player is always fully illuminated regardless of body position or movement, the LED panels must maintain illumination uniformity of ≥ 90% and deliver sufficient brightness to achieve full body visibility at a 1-meter distance.

### 2.7 Hardware Block Diagram

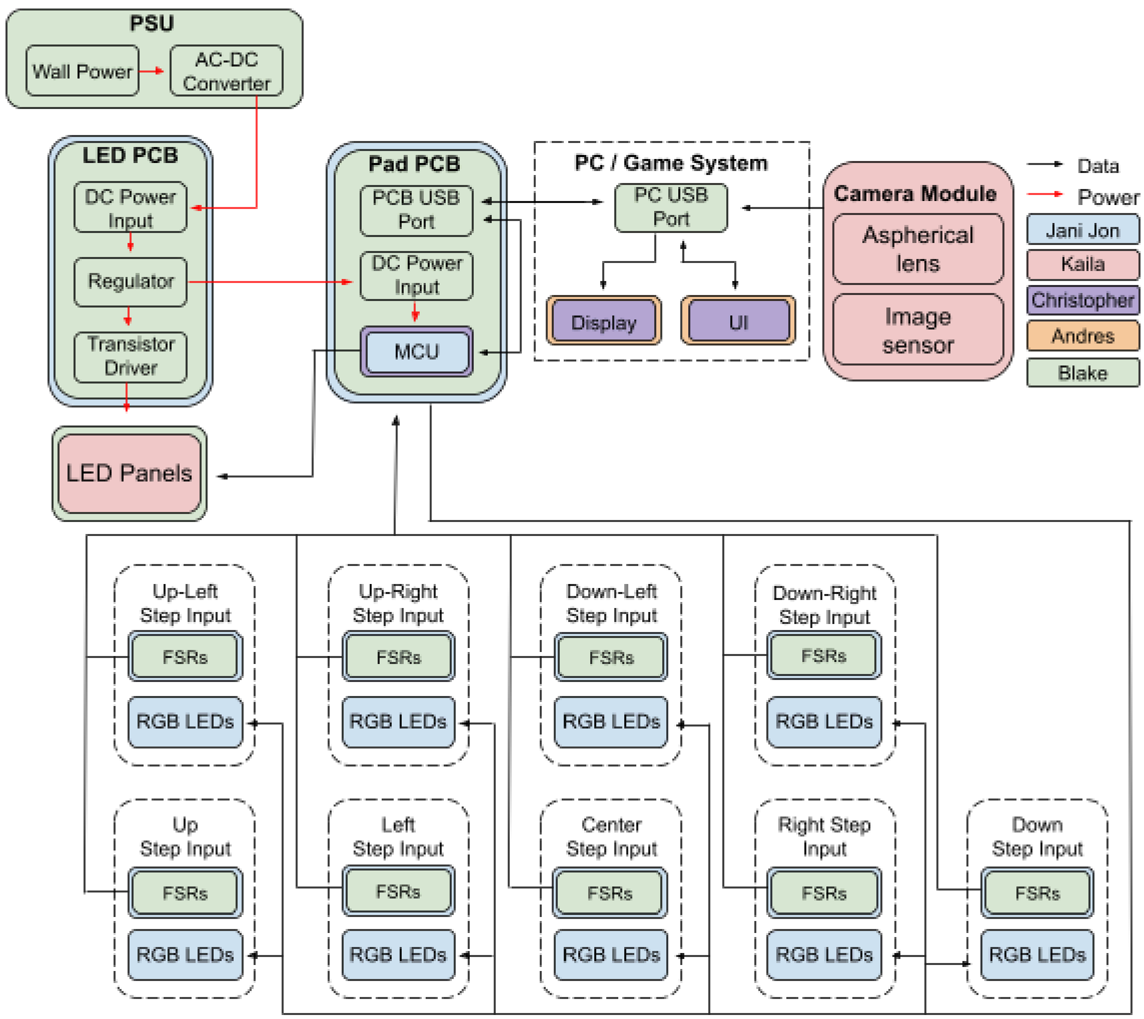



***Figure 2.7*** *Hardware Block Diagram showcasing work distribution and major components of the design*

## 2.8 Software Block Diagram

🟧Andres
🟪Christopher

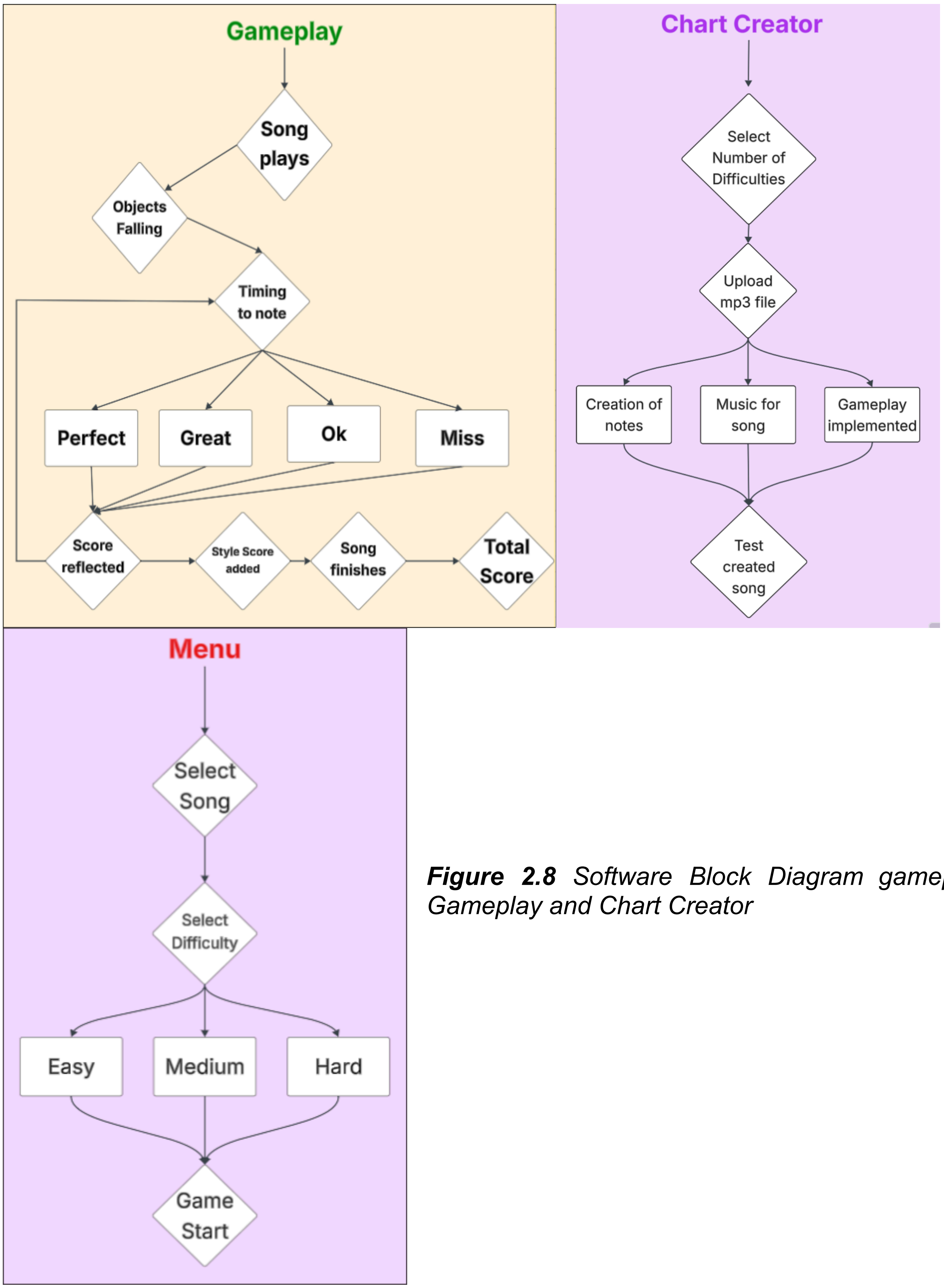
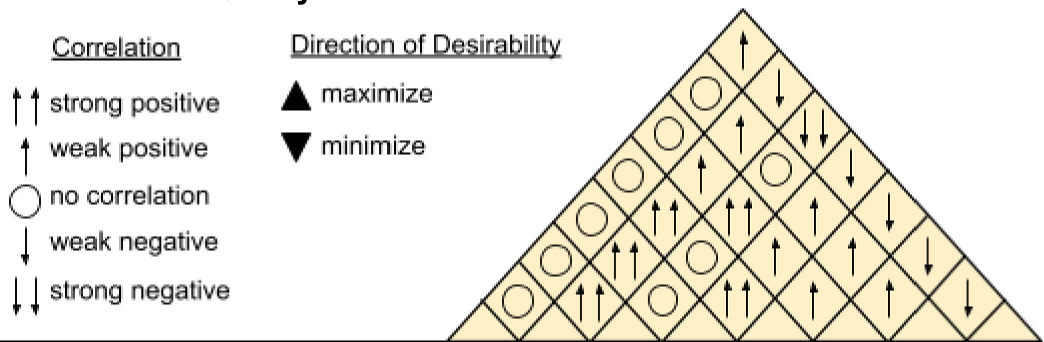



***Figure 2.8*** *Software Block Diagram gameplay Gameplay and Chart Creator*

## 2.10 House of Quality




| Marketing Requirements \ Engineering Requirements | Direction of Desirability | Pad Dimensions | Pose Identification Accuracy | LED Panel Illumination Uniformity | Camera Lens Resolution | Camera Lens Horizontal FOV | Active Power Consumption | Pad Input Latency | Cost of Materials |
|---|---|---|---|---|---|---|---|---|---|
| Direction of Desirability | | ▼ | ▲ | ▲ | ▼ | ▲ | ▼ | ▼ | ▼ |
| Affordability | ▲ | ↑ | ↓ | ↓ | ↓ | ↓ | ↑ | ○ | ↑↑ |
| Consistency | ▲ | ○ | ↑↑ | ↑ | ↑ | ↑↑ | ↑ | ↑↑ | ↓ |
| Durability | ▲ | ↑ | ○ | ○ | ○ | ○ | ↑ | ○ | ↓↓ |
| Easy Installation | ▲ | ↑↑ | ○ | ○ | ○ | ○ | ○ | ○ | ↓ |
| Reliability | ▲ | ○ | ↑↑ | ↑ | ↑ | ↑↑ | ↑ | ↑↑ | ↓ |
| Maintenance | ▲ | ↑ | ○ | ○ | ○ | ○ | ↑ | ○ | ↓ |
| Targets for Engineering Requirements | | ≤ 38 x 38 x 2 in | ≥ 95% | ≥ 90% | ≤ 1 mm | ≥ 1830 mm | ≤ 200 W | ≤ 10 ms | ≤ $400 |

***Figure 2.9*** *House of Quality*

# 3. Research

## 3.1 Technologies

### 3.1.1 Embedded Systems

To integrate embedded systems into our STEPS platform, we require a dedicated computing component. The two most suitable options are a microcontroller unit (MCU) or a single-board computer (SBC). Each has its strengths and tradeoffs, but only one is better suited to execute the functionality required for our project.

SBCs offer significantly higher processing power than MCUs, making them capable of running full operating systems, graphical interfaces, and multitasking. However, the tradeoff is that they also consume more power, require more complex infrastructure, and lack real-time response by default. Meanwhile, MCUs are optimized for fast, real-time control, which is critical for rhythm game input responsiveness. They are also cheaper, easier to wire, and consume less power, making them ideal for precise button and LED control. Therefore, we take a deeper dive into both options to evaluate which is most optimal for our dance pad design.

#### 3.1.1.1 MCU vs SBC

MCUs are designed for task-specific control, particularly those that require low latency and real-time response, such as reading sensor data or controlling actuators. In our system, for input detection, an MCU would allow for minimal system latency, which is a critical factor for timing accuracy in rhythm games.

MCUs also offer faster boot-up times and simpler circuit integration, making them easier to debug and more reliable during development. Most MCUs include a large number of GPIO pins, many with PWM support, which makes them especially suitable for handling LEDs and button inputs with minimal delay. Their hardware simplicity aids in easier troubleshooting and faster hardware prototyping. However, MCUs typically require low-level programming (e.g., in C or C++) and are not capable of natively handling advanced UI, graphics, or audio features. As a result, we would require external modules if we were to implement these into our design.

In terms of cost, most MCU chips cost under $10, making them ideal for scalable, low-cost embedded designs. If the board is damaged or needs revision, the low cost of replacement makes development more forgiving.

In contrast, SBCs offer significantly higher processing power and are well-suited for applications involving graphical interfaces, audio, or wireless communication. They run full operating systems, typically Linux-based, and can be programmed using high-level languages like Python or Java, which simplifies software development.

However, their non-deterministic response time, due to OS-level multitasking, makes them less reliable for real-time input detection. This could introduce latency or jitter in detecting panel hits, affecting the gameplay experience. SBCs also have longer boot times, higher power consumption, and are generally more complex to integrate into custom PCB designs. On top of that, SBCs typically cost between $35–$60, which increases development risk if the board is damaged or requires replacement.

#### 3.1.1.2 MCU is the Better Option… But Why?

While neither MCUs nor SBCs are perfect in every category, MCUs are the more optimal choice for our dance pad design due to their fast and predictable response times, lower power consumption, smaller footprint, and significantly lower cost. As shown in *Table 3.1* below, the MCU excels in the areas most critical to our engineering and marketing goals, such as system responsiveness, efficiency, and affordability, as previously outlined in *Figure 2.9 House of Quality*.

To meet cost targets, we must use components that are low-cost yet reliable, and MCUs typically cost under $10, making them ideal for both prototyping and scalable production. Power consumption is another key concern, especially since our system will run multiple FSRs and RGB LEDs simultaneously, and MCUs are optimized for energy efficiency, unlike SBCs which consume more power even when idle. Real-time performance is especially critical for a rhythm-based game, where millisecond-level delays can affect gameplay. MCUs provide immediate input handling and low-latency output control, which are difficult to achieve on SBCs running multitasking operating systems without extensive tuning.

While SBCs may offer advantages like better multimedia processing or easier development with high-level languages, they introduce longer boot times, higher costs, and non-deterministic behavior, making them less suitable for a system where precision and timing matter most. MCUs, on the other hand, are straightforward to integrate, faster to initialize, and allow for fine-grained control over every component in the system.

Overall, MCUs give us the performance and reliability needed for fast input detection, smooth LED animations, and efficient control, all while keeping costs, power, and complexity well within our project's constraints.

***Table 3.1** Comparison table between MCUs and SBCs using their features*

| Feature | Microcontroller (MCU) | Single Board Computer (SBC) |
|---|---|---|
| Processing Power | | ✓ |
| Real-Time Response | ✓ | |
| Design Complexity | ✓ | |
| Coding Complexity | | ✓ |

| | | |
|---|---|---|
| Power Consumption | ✓ | |
| I/O Handling | ✓ | |
| UI Support | | ✓ |
| Boot Time | ✓ | |
| Cost | ✓ | |

**3.1.1.3 Arduino**

Arduino is a series of open-source microcontroller boards designed for ease of use in both hardware and software. They are especially popular in the prototyping and hobbyist community due to their beginner-friendly development environment (Arduino IDE), and extensive library support for I/O, sensors, LEDs, and communication protocols. Arduino boards typically use AVR-based microcontrollers, although some variants use ARM cores. In this section, we look at three of the most fitting boards for our dance pad design: Arduino Uno, Arduino Mega 2560, and Arduino Leonardo.

One of Arduino’s most iconic boards is the Arduino Uno, which uses the 8-bit ATmega328P microcontroller. It runs at a clock speed of 16 MHz, with 32 KB of flash memory, 2 KB of SRAM, and 1 KB of EEPROM. The Uno includes 14 digital I/O pins (6 with PWM support) and 6 analog inputs (ADC). It’s widely supported and extremely simple to program and wire, making it a go-to for prototyping. However, it lacks native USB HID support, which means it cannot act as a USB joystick or keyboard without external USB-serial bridging. For our dance pad, the Uno would fall short on both ADC and PWM channels unless additional multiplexing components were added.

One of Arduino’s higher-end boards is the Arduino Mega 2560, which features the ATmega2560 microcontroller. Like the Uno, it runs at 16 MHz, but includes significantly more memory: 256 KB of flash, 8 KB of SRAM, and 4 KB of EEPROM. It provides 54 GPIO pins, 15 PWM channels, 16 ADC channels, and 4 UART serial ports, making it highly capable of handling our dance pad’s inputs. The Mega does not support native USB HID, but its I/O abundance makes it ideal for wiring-heavy applications.

Another one of Arduino’s boards is the Arduino Leonardo, which uses the ATmega32U4 microcontroller. It also runs at 16 MHz, with 32 KB of flash memory, 2.5 KB of SRAM, and 1 KB of EEPROM. It provides 20 GPIO pins, 7 PWM channels, and 12 ADC channels. Its standout feature is native USB HID support, allowing it to behave like a USB keyboard or joystick, a feature perfect for our design. While it offers fewer I/O pins than the Mega, it still covers our minimum requirements, and any shortfall can be handled using external components.

***Table 3.2*** *Arduino boards and their features comparison table*

| Feature | Arduino Uno | Arduino Mega 2560 | Arduino Leonardo |
|---|---|---|---|
| Main MCU | **ATmega328P** | **ATmega2560** | **ATmega32U4** |
| Clock Speed | 16 MHz | 16 MHz | 16 MHz |
| Flash Memory | 32 KB | 256 KB | 32 KB |
| SRAM | 2 KB | 8 KB | 2.5 KB |
| EEPROM | 1 KB | 4 KB | 1 KB |
| GPIO Pins | 14 | 54 | 20 |
| PWM Channels | 6 | 15 | 7 |
| ADC Inputs | 6 | 16 | 12 |
| USB Communication | serial-to-USB | serial-to-USB | native USB |
| USB HID Support | none | none | included |
| Serial Ports (UART) | 1 | 4 | 1 |
| Active Power Consumption | Moderate | Moderate-High | Moderate |
| MCU Chip Cost | ~$3 | ~$8 | ~$4 |

#### 3.1.1.4 Teensy

Teensy is a powerful and compact microcontroller board series developed by PJRC. While slightly less common than Arduino in beginner projects, Teensy is widely used in performance-critical embedded systems due to its speed, USB capabilities, and real-time performance. Most Teensy boards use ARM Cortex-M cores, which were specifically designed for embedded applications to offer high energy efficiency and low cost. All Teensy boards also support native USB HID, making them excellent for applications that require low-latency input handling, such as rhythm games. In this section, we explore four Teensy models: Teensy 2.0, Teensy 3.2, Teensy LC (Low Cost), and Teensy 4.1.

The oldest version of them all is the Teensy 2.0 (released in 2009). Out of our other Teensy options, Teensy 2.0 is the only one that doesn't use an ARM core. Instead, much like the Arduino Leonardo, it uses an ATmega32U4 8-bit automatic voltage regulator (AVR). This means that the Teensy 2.0 has almost the same exact specifications as an Arduino Leonardo. For instance, both run at a clock of 16 MHz and include 32 KB of flash memory, 2.5 KB of SRAM, 12 ADC channels, and USB HID support. It provides 25 GPIO pins, which is slightly more than the Leonardo's 20. Despite its age, Teensy 2.0 remains a lightweight and reliable choice for simple USB input devices. However, it lacks the processing power and memory needed for advanced LED animations or real-time multitasking.

Released in 2014, the Teensy 3.2 offers a major leap in performance over the 2.0. It features an MK20DX256VLH7 MCU with a 72 MHz ARM Cortex-M4 processor, supporting both DSP instructions and hardware floating point. It includes 256 KB of flash memory, 64 KB of SRAM, and 34 GPIO pins. The Teensy 3.2 supports 21 analog inputs (ADC) and 12 PWM outputs, and unlike Teensy 2.0, the ADC is 13-bit (not 16-bit as sometimes misreported). It supports USB HID, has robust NeoPixel support (including DMA-based LED control), and is powerful enough to handle real-time step detection, LED animations, and game communication simultaneously. It offers an excellent balance between performance, GPIO availability, and price, making it a strong candidate for mid-level embedded designs like our design.

Introduced in 2015, the Teensy LC is a budget-friendly alternative that still offers modern features. It uses the MKL26Z64VFT4 MCU, based on a 48 MHz ARM Cortex-M0+ processor. It comes with 62 KB of flash memory, 8 KB of SRAM, and 27 GPIO pins. It includes 13 analog inputs and 10 PWM outputs, and it supports USB HID. While not as fast as the 3.2, the Teensy LC still outperforms Teensy 2.0 in nearly all areas. It does not support DMA for NeoPixel LED control, meaning LED animations may add CPU overhead. Still, its low cost and decent GPIO make it a practical option if budget is a major constraint.

The most powerful board in the series, Teensy 4.1, was released in 2020 and uses the MIMXRT1062 MCU with an ARM Cortex-M7 processor running at a blazing 600 MHz. It includes 1 MB of SRAM, 8 MB of QSPI flash memory, and supports external PSRAM or flash via expansion pads. It features 55 GPIO pins, 18 analog inputs, and up to 35 PWM outputs, with USB High-Speed (480 Mbps) support and full native USB HID. Unlike Teensy 3.2 or LC, Teensy 4.1 also supports cache, branch prediction, and complex multitasking, making it ideal for fast input response, dynamic LED control, and interfacing with multiple peripherals simultaneously. While the ADC is only 12-bit and the analog pin count is slightly lower than Teensy 3.2, the vast processing power and memory make Teensy 4.1 a top-tier choice.

***Table 3.3*** *Teensy boards and their features comparison table*

| Feature | Teensy 2.0 | Teensy 3.2 | Teensy LC | Teensy 4.1 |
|---|---|---|---|---|
| MCU Chip | **ATmega32U4** | **MK20DX256VLH7** | **MKL26Z64VFT4** | **MIMXRT1062** |
| Structure | 8-bit AVR | ARM Cortex-M4 | ARM Cortex-M0+ | ARM Cortex-M7 |
| Clock Speed | 16 MHz | 72 MHz | 48 MHz | 600 MHz |
| Native USB Speed | 12 Mbps (full speed) | 12 Mbps (full speed) | 12 Mbps (full speed) | 480 Mbps (high speed) |
| Flash Memory | 32 KB | 256 KB | 62 KB | 8 MB |
| SRAM | 2.5 KB | 64 KB | 8 KB | 1024 KB |
| GPIO Pins | 25 | 34 | 27 | 55 |
| PWM Channels | 7 | 12 | 10 | 35 |
| ADC Inputs | 12 (12-bit) | 21 (13-bit) | 13 (12-bit) | 18 (12-bit) |
| Active Power Consumption | Low | Moderate | Low | High-Very High |
| MCU Chip Cost | ~$4 | ~$6-9 | ~$2-4 | ~$7-11 |

#### 3.1.1.5 Raspberry Pi Pico (RP2040)

Although there are multiple other Raspberry Pi series like Raspberry Pi 4 and Raspberry Pi Zero, they are not efficient enough to be considered for our design. Due to the fact that most other Raspberry Pi use Linux which can make them heavier on power and boot time, less reliable when it comes to real-time responses, harder to interface with timing-sensitive components, and more complex to integrate into our dance pad's PCB. The series we want to look at, however, is the Raspberry Pi Pico series.

The Raspberry Pi Pico is a microcontroller, not a general-purpose computer, that uses its own microcontroller chip, the RP2040. The RP2040 is a dual-core Cortex M0+ microcontroller chip that's known for being affordable while having a high performance. Because the RP2040 is made of bare silicon, it comes with a QFN-56 package, which supports compact PCB design and high-speed operation, allowing it to have good thermal dissipation, short lead lengths for better signal integrity, and maintain a compact size for better use of space. However, having the QFN-56 package also makes soldering into the PCB board challenging if done by someone with little to no experience.

The Raspberry Pi Pico runs at a clock speed of 133 MHz, making it well-suited for performance-intensive tasks. Much like the Teensy boards, Pico also supports USB HID, though it requires software implementation via libraries like the Pico SDK or CircuitPython, and no external components or converters are needed for HID communication. However, while the Pico provides 26 GPIO pins and 16 PWM channels, it includes only 3 12-bit ADC channels, which is insufficient to directly read analog signals from all 9 FSR-based pads. To support more analog inputs, external analog multiplexers or ADC chips would be needed. In terms of memory, the Pico offers 2 MB of flash and 264 KB of SRAM, which is significantly more than typical Arduino boards and even some Teensy models. Overall, Raspberry Pi Pico's RP2040 is a highly affordable chip (~$1) capable of handling complex tasks at high speeds, while maintaining low to moderate active power consumption. However, using this chip will require external components, such as analog multiplexers for more ADC inputs, for our design to be fully functional.

#### 3.1.1.3 ESP32

The ESP32 is a series of low-cost, low-power system-on-a-chips (SOCs). MCU chips used by ESP32 modules typically cost around $1-$3. What makes the ESP32 different from traditional microcontrollers like AVR or RP2040 is that most ESP32 modules have Wi-Fi and/or Bluetooth capabilities. In this section, we analyze and compare the ESP32 (WROOM-32), ESP32-S2, ESP32-S3, and ESP32-C3.

The ESP32-WROOM-32 is one of the earliest and most commonly used modules in the ESP32 series. It features a dual-core 32-bit Xtensa LX6 processor and supports both Wi-Fi and Bluetooth Classic + BLE. Despite being one of the older models, it can still run at up to 240 MHz, making it capable of handling performance-heavy tasks. It comes with 520 KB of SRAM and 4 MB of flash memory by default, though variants with 8 MB or 16 MB of flash are also available. The chip provides up to 34 GPIO pins, 16 PWM channels, and 18 ADC channels, which is sufficient for our design. However, one major limitation is that it does not have native USB HID support without additional hardware. To use the WROOM-32 in a USB-connected setup, we would need to add an external USB-to-HID bridge chip or pair it with another microcontroller that supports USB HID natively. Therefore, unless we design the DDR pad to be fully wireless, it may be more practical to use another ESP32 variant that can support native USB HID directly.

The ESP32-S2 improves on the WROOM-32, most notably, by adding native USB support. It features a single-core Xtensa LX7 processor that can run up to 240 MHz, and, by default, includes 320 KB of SRAM and 4 MB of flash memory. Like other ESP32 modules, it supports Wi-Fi, but unlike the original WROOM-32, it does not support Bluetooth. The chip provides up to 43 GPIO pins, 16 PWM channels, and 20 ADC channels, which is slightly more compared to WROOM-32. What sets the ESP32-S2 apart is its native USB OTG support, which allows it to act as a USB HID device, such as a gamepad or keyboard, without requiring any external USB bridge or extra MCU. The main drawback is its single-core design, which limits multitasking performance compared to dual-core models, but this tradeoff may be acceptable if real-time responsiveness is properly managed.

The ESP32-S3 builds upon the capabilities of the S2 and is arguably the most feature-rich variant in the ESP32 family for the design we're aiming to achieve. It retains native USB HID support, much like S2. It uses a dual-core Xtensa LX7 processor running at up to 240 MHz, with 512 KB of SRAM and 128 KB of RTC memory, offering more headroom for handling concurrent tasks such as sensor polling, LED animations, and USB communication. The S3 has 45 GPIO pins, 8 PWM channels, 20 ADC channels, and it expands functionality with BLE 5.0 support (though it lacks classic Bluetooth). The ESP32-S3 is especially suitable for performance-demanding applications that require both USB connectivity and multiple analog inputs, making it one of the best choices for a wired dance pad built around a single microcontroller.

The ESP32-C3 is a compact, low-cost MCU in the ESP32 family that trades raw performance for simplicity and power efficiency. It features a single-core 32-bit RISC-V processor running at up to 160 MHz, with 400 KB of SRAM and 4 MB of flash memory. While some variants of C3 can support external flash chips up to 16 MB, some don't, like the ESP32-C3-WROOM-02U. One of its most attractive features is its inclusion of native USB 2.0 support, enabling it to function as a USB HID device without requiring external USB-to-serial hardware. However, C3 only supports 22 GPIO pins, 6 hardware LED PWM channels, and 6 ADC channels, which makes it unsuitable for reading all of our dance pad's inputs directly without the use of an analog multiplexer or external ADC chip. It does support BLE 5.0, though it lacks classic Bluetooth. Overall, while it's not as powerful or feature-rich as the S2 or S3, the ESP32-C3 is a budget-friendly option for our designs, especially if analog input limitations can be mitigated with external circuitry.

***Table 3.4*** *ESP32 boards and their features comparison table*

| Feature | ESP32-WROOM-32 | ESP32-S2 | ESP32-S3 | ESP32-C3 |
|---|---|---|---|---|
| MCU Chip | **ESP32-D0WD** | **ESP32-S2** | **ESP32-S3** | **ESP32-C3** |
| Structure | Dual-core Xtensa LX6 | Single-core LX7 | Dual-core LX7 | Single-core RISC-V |
| Clock Speed | 240 MHz | 240 MHz | 240 MHz | 160 MHz |
| Flash Memory | 4-16 MB | 4-16 MB | 4-16 MB | 4-16 MB (some) |
| SRAM | 520 KB | 320 KB (128 KB RTC) | 512 KB (128 KB RTC) | 400 KB |
| GPIO Pins | 34 | 43 | 45 | 22 |

| | | | | |
|---|---|---|---|---|
| PWM Channels | 16 | 16 | 8 | 6 |
| ADC Inputs | 18 | 20 | 20 | 6 |
| Native USB | None | USB OTG | USB OTG | USB 2.0 |
| Wi-Fi | Yes | Yes | Yes | Yes |
| Bluetooth | Classic + BLE | None | BLE 5.0 | BLE 5.0 |
| Active Power Consumption | High | Moderate-High | High | Moderate |
| MCU Chip Cost | ~$2-3 | ~$2 | ~$2.50-3 | ~$1.50-2 |

#### 3.1.1.3 STM32

STM32 is a series of 32-bit ARM Cortex-M microcontrollers. They're known for having long-term reliability, rich peripheral sets (i.e. ADC, USB, timers, etc.), native USB support in most of its variants, strong real-time performance, great power efficiency, and full support in IDEs like STM32CubeIDE, Keil, PlatformIO, and Arduino (for some variants). We will be looking at the most common and fitting variant from each of the following STM32 families: STM32F1, STM32F4, STM32F0, and STM32L4.

One of STM32F1's most common MCU chips is the STM32F103C8, also known as the "Blue Pill" chip. It features a 72 MHz Cortex-M3 processor, 64 KB of flash memory, and 20 KB of SRAM. It also has 37 GPIO pins, 15 PWM channels, and 10 ADC channels, which is sufficient for our design requirements. STM32F103C8 supports native USB Full-Speed, which can be configured for HID communication using STM32CubeMX and the HAL libraries. However, being an older generation chip, it has limited memory for larger LED effects or buffered inputs. Still, it remains a cost-effective and capable option for our design.

One of STM32F4's most common MCU chips is the STM32F407VG, which is a high-performance microcontroller based on the 168 MHz Cortex-M4 core with DSP and FPU support. It includes 1 MB of flash memory and 192 KB of SRAM. It also has 82 GPIO pins, 3 ADCs (12-bit) that can handle up to a total of 16 multiplexed input channels, and supports up to 17 timers, several of which can generate PWM signals with up to 4 channels per timer. It also supports both Full-Speed and High-Speed USB, with High-Speed USB requiring an external PHY (Physical Layer). Overall, the STM32F407VG is ideal for advanced dance pad designs aiming for high responsiveness, complex LED patterns, or additional input features. The tradeoff is higher cost, larger size, and slightly more power consumption.

One of STM32F0's most common MCU chips is the STM32F072RB, which features a 48 MHz Cortex-M0 processor, 128 KB of flash memory, and 16 KB of SRAM. It includes 51 GPIO pins, 18 PWM channels, and 16 ADC channels (12-bit), making it a capable and low-power choice for handling our dance pad inputs and driving RGB LEDs. STM32F072RB supports native USB Full-Speed, and can be configured as a USB HID device using STM32CubeMX and the HAL libraries. While it lacks advanced processing features like DSP or FPU, its simplicity, low-cost, low power consumption, and USB support make it a great option for our design.

One of STM32L4's most versatile MCU chips is the STM32L476RG, which uses an 80 MHz Cortex-M4 core with both DSP and FPU support. It comes with 1 MB of flash memory and 128 KB of SRAM, offering plenty of space for real-time processing and buffering. The STM32L476RG provides up to 76 GPIO pins, 24 PWM channels, and 16 ADC channels (12-bit), allowing for extensive input reading and precise LED control. It also supports native USB Full-Speed and can operate in low-power modes, making it suitable for our design as it helps improve both performance and energy efficiency. The main trade off is its slightly higher cost, but its extensive peripheral set and low-power capabilities make it a strong candidate for our design.

***Table 3.5*** *STM32 common MCU chips from different variants comparison table*

| Feature | STM32F1 | STM32F4 | STM32F0 | STM32L4 |
|---|---|---|---|---|
| MCU Chip | STM32F103C8 | STM32F407VG | STM32F407VG | STM32L476RG |
| Structure | Cortex-M3 | Cortex-M4 | Cortex-M0 | Cortex-M4 |
| Clock Speed | 72 MHz | 168 MHz | 48 MHz | 80 MHz |
| Flash Memory | 64 KB | 1 MB | 128 KB | 1 MB |
| SRAM | 20 KB | 192 KB | 16 KB | 128 KB |
| GPIO Pins | 37 | 82 | 51 | 76 |
| PWM Channels | 15 | ~17 timers each ≤ 4 PWM | 18 | 24 |
| ADC Inputs | 10 | 16 | 16 | 16 |
| Native USB | FS USB | FS USB + HS (w/ PHY) | FS USB | FS USB |
| Active Power Consumption | Moderate | High | Very Low | Very Low |
| MCU Chip Cost | ~$2 | ~$7-10 | ~$1.50-2 | ~$4-5 |

### 3.1.2 Dance Pad Sensor

In order for the pad to send signals to the game, we need sensors that can detect physical input and convert it into electrical signals for the microcontroller to process. There are four common sensor types we can consider for this project: force-sensing resistors (FSRs), load cells, strain gauges, and piezoelectric sensors. We compare them based on cost, accuracy, complexity, size, durability, and their ability to detect both hold steps and taps. First, we'll explore what each sensor is, how it works, and how it meets the project's engineering requirements.

#### 3.1.2.1 Force-Sensing Resistors

Force-sensing resistors (FSRs) are sensors that change resistance in response to applied pressure. They are commonly used in DIY and custom dance pad projects because they are inexpensive, easy to wire, and can be constructed using accessible materials like Velostat or copper tape. FSRs are also thin, flexible, and lightweight, making them ideal for compact pad designs.

Though FSRs offer quick response times and simple analog interfacing (via a voltage divider and an ADC pin), they lack precision and can show non-linear output and signal drift over time. Poor placement or surface inconsistency can lead to dead zones or inconsistent detection. Additionally, they are sensitive to temperature and humidity, affecting their long-term reliability.

Overall, FSRs are a low-cost, simple, and responsive solution for detecting taps and light pressure in dance pads, but may not be ideal for applications requiring high durability or accuracy over time.

#### 3.1.2.2 Load Cells

Load cells are force transducers that measure weight or applied force by detecting mechanical deformation, typically using internal strain gauges. They output either a millivolt signal (analog) or, in some designs, a digital signal after amplification. Load cells are known for their high accuracy, excellent stability, and long-term durability.

Their downside lies in their higher complexity: they require a stable mechanical mounting, precise calibration, and amplification circuitry (such as an HX711 module) to interface with a microcontroller. Load cells also tend to be more expensive and bulky, which may be a constraint for slim pad designs.

In summary, load cells are ideal for high-precision, arcade-grade dance pads where accuracy and reliability are top priorities, but they involve more cost, bulk, and circuit complexity than other options.

#### 3.1.2.3 Strain Gauges (Raw)

Strain gauges are sensing elements that detect strain (deformation) in a material. This is typically a foil or wire pattern that changes electrical resistance when stretched or compressed. Unlike load cells, raw strain gauges don't measure force directly, but rather the strain on a structure, which can then be correlated to force.

When bonded carefully to structural parts of the pad, strain gauges can be very precise and customized for specific points of interest. However, they are also extremely sensitive to environmental noise and temperature, and they require precise installation, bridge circuits (e.g. Wheatstone bridge), and amplification to produce usable signals.

While strain gauges can be affordable, they are technically complex, and not ready-to-use out of the box. They are best suited for custom mechanical frames where advanced integration is possible and where precision and internal structural feedback are desired.

#### 3.1.2.4 Piezoelectric Sensors

Piezoelectric sensors use the piezoelectric effect to convert mechanical stress into electrical charge. When pressure or impact is applied, the crystal inside the sensor generates a voltage spike. Piezo sensors are exceptional for detecting fast impacts, such as dance pad taps, and can distinguish between light and hard hits due to their high sensitivity and fast response.

However, they cannot detect sustained pressure or holds, as they only respond to changes in force, not constant force. They are also prone to false triggers from vibration, footstep echoes, or mechanical noise, and they require careful physical isolation or dampening. On the hardware side, they are easy to wire (typically needing just a series resistor and ADC input), though voltage spikes may require clamping diodes to protect the MCU.

Overall, piezoelectric sensors are affordable, fast, and ideal for games focused solely on tap detection, but they are not suitable for designs requiring hold step input or highly stable measurements.

#### 3.1.2.5 Break Beam Sensors

Break beam sensors are non-contact optical sensors that detect the presence or absence of an object by interrupting a beam of infrared (IR) light between a transmitter and a receiver. When the beam is unbroken, the sensor outputs a steady signal; when the beam is interrupted (e.g. by a footstep), the signal changes, allowing the microcontroller to detect an input event. Break beam sensors are very fast, have high sensitivity, and are not affected by pressure wear, making them highly durable.

However, they cannot detect how hard or long the player is pressing, so they are not capable of reading analog force or hold steps. Alignment between the emitter and receiver must also be precise, and ambient IR sources (like sunlight or reflective surfaces) can interfere with the sensor's reliability. Additionally, their installation typically requires mounting hardware and unobstructed space under or around the panel.

Overall, break beam sensors are a reliable and low-maintenance solution for detecting quick taps or triggered events, but they lack the analog depth needed for pressure-sensitive gameplay and cannot detect holds, making them best suited for tap-only rhythm games or basic input triggering.

### 3.1.3 Monochrome vs RGB Sensor

Selecting the appropriate camera sensor for the S.T.E.P vision system was an important decision in the beginning stages of the design process. Technically a RGB or monochrome sensor could be used. Each sensor type has distinct advantages and weaknesses. Monochrome sensors are known for producing high-contrast and high-resolution images due to the absence of color filter arrays.[11] Which can improve clarity and reduce light loss. These filters are particularly effective in applications where edge detection, structure analysis, or low-light imaging is critical.

They also generate smaller data sizes which reduces computational load. These factors are all very promising when considering a major goal of the system is to minimize the computational load and processing time of the vision system. However, the pose estimation model used in this system is MediaPipe. MediaPipe's system is optimized for RGB input as it relies on color information to identify and track body landmarks accurately, using patterns in hue and saturation to distinguish between limbs and background.[10] Since RGB sensors capture color images in a format that mirrors human visual perception, they are best to use for AI models that interpret motion and orientation. In the S.T.E.P system, the visual feedback is provided to the player via the RGB LED arrays placed in the dance pads. While the motion is captured using the FSR sensors, pose estimation relies purely on visual tracking. In this case, a monochrome sensor would introduce ambiguity in differentiating the background from the player due to no color differentation. With this in mind, the benefits from using a RGB sensor outweigh those of the monochrome sensors in this application. Despite their slightly higher data rates and lower per-pixel sensitivity. The deficiencies in image contrast or sharpness introduced by the RGB filter array are mitigated through diligent and carefully designed lens system and image scale optimization.

### 3.1.4 LED illumination Technology

For the illumination system, both white & RGB LED strips, as well as individual LEDs were considered. Thorough research went into deciding which option would be the most optimal for this system. While RGB LEDs would offer more visual customization, they raise possible challenges due to their color inconsistencies and power complexity making them less ideal for computer vision tasks. White LEDs would provide better illumination with more uniform color output and higher power efficiency[9], but could also cause discomfort to the players' eyes. Ensuring that MediaPipe is able to consistently perform reliable pose detection is more important than aesthetic lighting, but player comfort is an equally important metric.

With that being said, more research went into finding an alternative that would satisfy both the illumination and user comfort requirements. 850 nm LEDs were found to be the most optimal choice due to their ability to illuminate the player without causing discomfort during game play. 850nm is near-infrared and only partially visible to the player, but it is reliably detected by the camera as illumination.[7] During the camera selection process, it was important to find a camera that did not include an IR-cut filter to ensure that the 850nm LEDs would be picked up on screen.

The choice between using individual IR LEDs versus LED strips was also considered but LED strips were quickly determined to be the most optimal solution due to their ease of integration and consistent IR output. Standard 12 V LED strips simplify power distribution compared to the more complex constant-current drivers needed for discrete LEDs, aligning with the system's power efficiency goals. They are also highly practical for testing because they can be easily modified or repositioned as needed. They allow for better uniform and even light coverage across the dance pad area, which supports consistent MediaPipe landmark tracking. While their only downside is having slightly less beam-shaping flexibility, this factor is less important than cost, simplicity and performance, which is why LED strips ultimately outperform discrete individual LEDs in this design.

### 3.1.5 Embedded System Development Languages

Our system will require software development both for the microcontroller unit (MCU) and the rhythm game interface. In this section, we evaluate the most suitable programming languages for each component based on criteria such as speed, ease of development, memory control, and hardware access. The goal is to identify the best language for programming our MCU and, if necessary, a separate language best suited for developing the rhythm game.

#### 3.1.5.1 C

C is a low-level language most commonly used for programming MCUs. It provides direct access to hardware while offering a cleaner abstraction than assembly language. Because C compiles directly into machine code with minimal runtime overhead, it allows extremely fast execution and real-time responsiveness, which is critical for embedded applications like sensor reading or LED control.

Unlike high-level languages, C has no garbage collection, safety checks, or virtual machines. Instead, the programmer is responsible for memory allocation and hardware interfacing. While this increases development complexity, it allows precise control over peripherals like GPIOs, ADCs, and timers. C is widely supported across virtually all microcontroller platforms, including Teensy, Arduino, STM32, and ESP32. Overall, C is not the easiest to learn, but it is the most efficient and reliable language for embedded development.

#### 3.1.5.2 Python

Python is a high-level, interpreted language known for its simplicity, readability, and rapid development. While it is not suitable for low-level embedded programming (due to memory and speed limitations), it is commonly used for PC-side applications, prototyping, and game development, especially when paired with frameworks like Pygame for rhythm games.

In embedded systems, Python is occasionally used on SBCs like the Raspberry Pi, but not on MCUs like Teensy, where C or C++ are more appropriate. However, Python can be used to simulate or visualize data from the dance pad or build a simple game interface that responds to USB HID inputs from the MCU. Its vast ecosystem and ease of integration with USB devices make it a strong candidate for the rhythm game software, but not the microcontroller firmware.

#### 3.1.5.3 Java

Java is another high-level language often used in application development, especially for cross-platform environments. Like Python, Java is not ideal for embedded MCU programming due to its reliance on the Java Virtual Machine (JVM), which adds significant overhead and lacks the direct hardware access needed for real-time control.

However, Java can be used effectively for developing desktop-based rhythm games or user interfaces that communicate with the dance pad via USB. Java’s strong support for event-driven programming, GUI libraries (like JavaFX or Swing), and device input handling makes it a reasonable choice if the rhythm game requires more structure or modularity than Python might provide. Its portability and object-oriented architecture may be overkill for simple games but could benefit larger or more scalable systems.

### 3.1.4 Computer Vision

#### 3.1.4.1 History of Computer Vision

Computer vision (CV) is a multidisciplinary field that enables computers to interpret and process visual information from the world, emulating the capabilities of human vision. The field has evolved significantly over the past six decades, transitioning from basic edge detection algorithms to modern real-time neural network-based systems used in self-driving cars, healthcare, and interactive entertainment.

The foundations of computer vision were laid in the 1960s and 70s, when researchers began exploring how machines could extract information from images. One of the earliest breakthroughs came from Irwin Sobel, who in 1968 developed the Sobel operator, a discrete differentiation filter used for edge detection in images[16]. The

Sobel filter works by approximating the gradient of image intensity, allowing researchers to identify boundaries and shapes within an image. This foundational technique was among the first attempts to compute visual structure from a flat image and is still taught in introductory CV courses today such as CAP4453 at UCF.

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

Gx

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Gy

***Figure 3.1*** *Sobel edge detection filter visualizing intensity gradients in horizontal and vertical directions*

Another pivotal advancement in edge detection came in 1986 with the development of the Canny edge detector by John F. Canny[17]. Unlike earlier filters like Sobel, which detect edges based on intensity gradients, the Canny method applies a multi-stage pipeline that includes Gaussian smoothing, gradient computation, non-maximum suppression, and hysteresis thresholding. The result is a cleaner and more accurate set of edges with reduced noise and better connectivity, making Canny detection widely used in image preprocessing pipelines to this day.

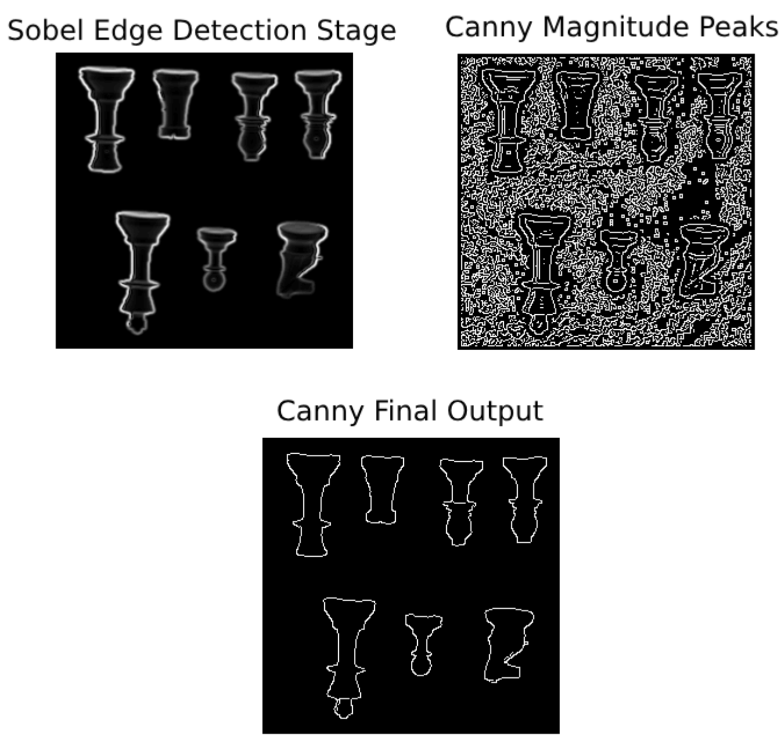



***Figure 3.2*** *Canny Edge detection filter pipeline demonstrated on Chess pieces*

During the same era, researchers experimented with brightness constancy and image segmentation, attempting to recreate how the human visual system identifies objects based on contrast and light intensity. Early efforts were largely confined to static grayscale images, and computing limitations of the time made real-time analysis impossible. In the 1980s, the focus shifted to 3D reconstruction and feature extraction, with researchers developing algorithms that could infer depth and motion from multiple views. This included optical flow, used to estimate motion between consecutive frames of video, and stereo vision, used to simulate binocular perception. These advancements laid the groundwork for robotics and autonomous navigation.

The 1990s saw a surge in pattern recognition and template matching techniques. Researchers developed systems capable of detecting basic objects, faces, and even simple human gestures. However, these systems were often brittle, sensitive to lighting, occlusion, and noise.

The 2000s brought significant improvements with the rise of machine learning, enabling systems to learn visual features from labeled datasets. Techniques like Haar cascades (used in early face detection), SIFT (Scale-Invariant Feature Transform), and HOG (Histogram of Oriented Gradients) became standard. However, these techniques still required manual feature engineering and were limited in complex environments.

The biggest transformation in computer vision came in the 2010s with the rise of deep learning, particularly Convolutional Neural Networks (CNNs). Inspired by biological visual systems, CNNs can learn hierarchical representations of images directly from raw pixel data, eliminating the need for handcrafted features like SIFT or HOG. The breakthrough moment arrived in 2012, when AlexNet[18], a deep CNN developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, achieved a dramatic improvement in classification accuracy during the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

AlexNet introduced innovations such as Rectified Linear Unit(ReLU) activation functions, dropout for regularization, and GPU acceleration for training. Its architecture, consisting of five convolutional layers followed by three fully connected layers, demonstrated how neural networks could automatically learn abstract visual features from millions of labeled images. This marked the beginning of deep learning’s dominance in the field of computer vision, shifting the focus from algorithmic feature engineering to data-driven learning.

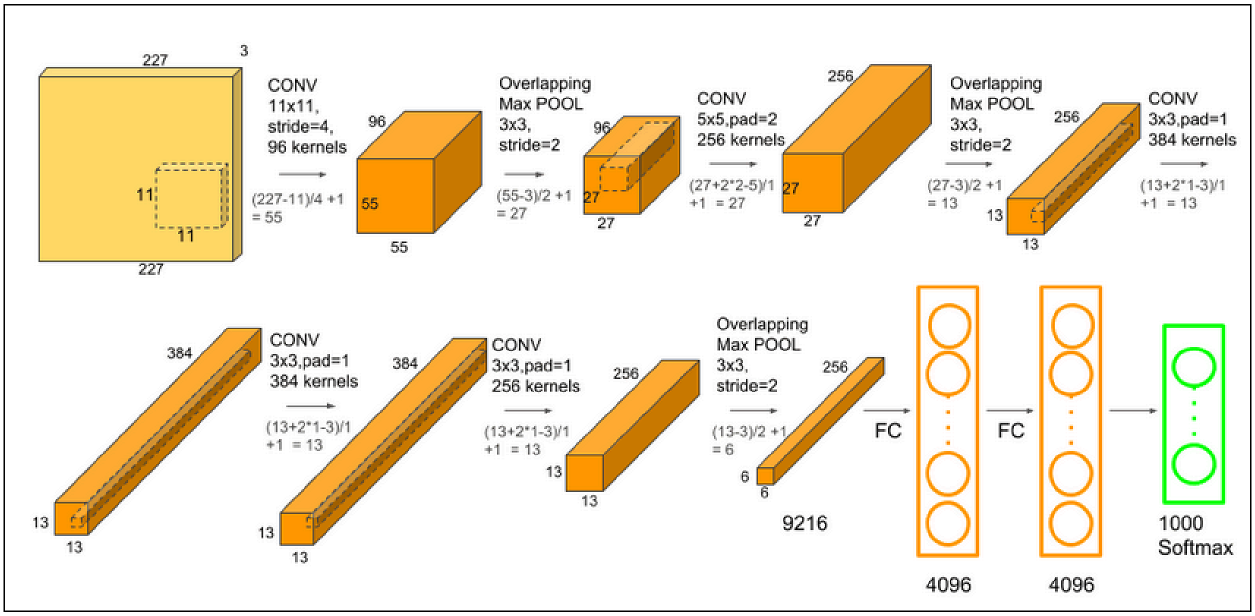



***Figure 3.3*** *AlexNet architecture used in the 2012 ImageNet competition utilizing multiple convolutional layers, ReLU activations, max pooling, and fully connected layers*

Since then, modern CV systems have achieved superhuman performance in tasks like image classification, object detection, and semantic segmentation. Frameworks like TensorFlow, PyTorch, and OpenCV have democratized access to these technologies, making CV accessible to developers and researchers worldwide.

In the 2020s, vision models began adopting transformer architectures, first popularized in natural language processing. Unlike CNNs, which operate on localized spatial kernels, Vision Transformers (ViT) treat image patches as a sequence of tokens and model global context through self-attention mechanisms. This shift enables long-range spatial dependencies and outperforms traditional CNNs on large datasets when given sufficient compute. ViT and similar architectures are at the cutting edge of current research in CV, offering new ways to represent and interpret visual data. While still resource-intensive, they are beginning to influence real-time applications in gesture recognition and pose estimation, potentially useful for future iterations of our rhythm game.

Today, computer vision has become a core enabler of innovation and a cornerstone of industries including healthcare for things like cancer screening, automotive industries for self-driving cars and blindspot detection, security industries for various facial recognition techniques, and even gaming industries for motion capturing. As processing power continues to increase and datasets grow richer, the accuracy, speed, and complexity of CV systems are expected to improve even further, bringing us closer to systems that can truly see and understand our world.

#### 3.1.4.2 History of Computer Vision in Games

Computer vision (CV) technologies have seen increasing integration into modern games, from facial detection and hand tracking to body motion recognition. These advancements have allowed developers to create more immersive and interactive gaming experiences by enabling players to interact with virtual worlds using natural body movements rather than traditional input devices. However, this fusion of CV and gameplay has a long history of experimentation, ranging from novel successes to forgettable gimmicks.

Early attempts to incorporate computer vision in gaming can be traced back to the Sony EyeToy, a webcam-based input device for the PlayStation 2, released in 2003. The EyeToy captured the player's image and motion using basic background subtraction and motion segmentation to enable interaction with on-screen elements. Games like *EyeToy: Play* and *Kung-Fu Live* utilized this setup to allow players to punch objects or perform gestures in rhythm games. However, the lack of robust depth sensing or pose tracking limited the fidelity and complexity of interactions, often making the experience frustrating or imprecise.

The Microsoft Kinect, released in 2010 for the Xbox 360, represented a major leap forward. Utilizing an infrared depth sensor and RGB camera, the Kinect could track full-body skeletons in real-time using a technique called skeleton tracking, which mapped 20 joints of the player's body. This enabled games like *Kinect Adventures* and *Dance Central* to support full-body gameplay without the need for controllers. The Kinect SDK and the associated machine learning models enabled gesture detection and activity recognition, expanding the use of CV beyond entertainment into healthcare, education, and robotics. However, challenges such as latency, lighting interference, and limited tracking accuracy in multi-person scenes persisted.

While the Kinect was not utilizing conventional Computer Vision techniques and opted for IR sensing, it was able to prove very effective for the time allowing less powerful hardware like the xbox to perform simple calculations. By analyzing distortions in an infrared dot pattern, Kinect could calculate depth and track motion without any physical markers.

To make skeletal tracking accessible to developers, Microsoft released the Kinect SDK, enabling applications to extract joint data and recognize poses. A foundational feature of this SDK was its ability to detect up to 20 distinct skeletal joints, as visualized in Figure 3.3.2.A. These include major points such as the head, spine, hips, shoulders, and limbs. This allowed developers to build gesture-based interfaces, games, and virtual assistants capable of interpreting full-body movement.

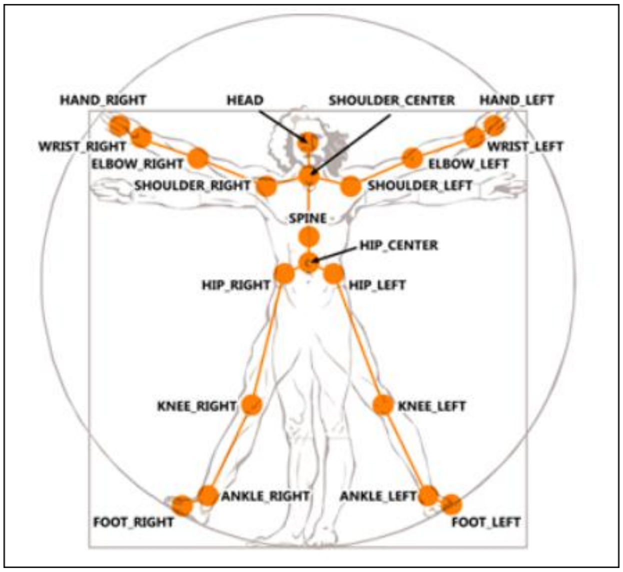



***Figure 3.4*** *Joint layout used by Microsoft Kinect's skeletal tracking system*

Kinect's architecture proved revolutionary for its time. The use of the various hardware shown in Figure 3.3.2.B such as the IR Emitter, Color Sensor, IR Depth Sensor, and Tilt Motor allowed the Kinect to capture accurate per-pixel values along with depth measurements even in low light shown in 3.3.2.C. These technologies were critical in early gesture-based games like *Dance Central* and inspired later research in human-computer interaction.

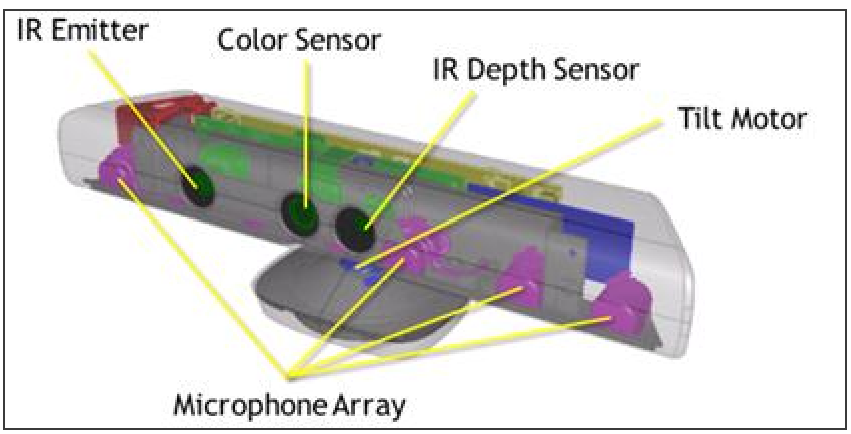



***Figure 3.5*** *Kinect Architecture*


***Figure 3.6*** *3D Depth Image in Grayscale*

Microsoft also provided developer tools like Kinect Studio, which recorded and replayed interaction data to aid debugging and testing. Applications could be built rapidly in WPF using templates like KinectSeletonApplication which visualized tracked joints as ellipses on a canvas. The Kinect SDK also introduced features like seated mode, facial tracking, and support for close-range users via *Near mode*, allowing for more flexibility in design.

Kinect's legacy lives on in modern computer vision applications. Its model of joint-based tracking and use of multimodal input (pose + voice) laid the groundwork for today’s AI-powered body tracking with neural networks and webcams. While contemporary systems like MediaPipe Pose and OpenPose no longer require specialized hardware, the principles they followed, pth estimation, joint modeling, and user-centric design, remain deeply rooted in Kinect's innovations.

Another notable application of CV in gaming is Just Dance, a franchise by Ubisoft launched in 2009. Originally designed for the Nintendo Wii using motion controllers, newer iterations supported Kinect and PlayStation Camera input. Just Dance evaluates player movements by comparing their captured silhouette or skeleton data to pre-recorded dance routines. While this system simplifies feedback into generalized “good” or “perfect” scores, it creates an accessible experience that emphasizes fun over precision. This trade-off reflects a key challenge in CV-based gameplay: achieving a balance between technical accuracy and player enjoyment.

In recent years, computer vision has enabled augmented reality (AR) games such as *Pokémon GO*, which use smartphone cameras and visual SLAM (simultaneous localization and mapping) to blend virtual content with the real world. Although AR relies more heavily on scene understanding than pose detection, these games showcase how CV can extend interactivity beyond fixed hardware setups.

More recently, open-source tools like MediaPipe and OpenPose have democratized access to pose estimation algorithms that were once exclusive to large tech companies. With these libraries, developers can extract real-time body, hand, and facial landmarks using only a standard webcam. This has fueled a resurgence in indie games and research projects that explore motion-based gameplay and accessibility.

Commercial arcade rhythm games have also adopted modern CV technologies. Games such as Dance Around, developed by Konami, utilize 3D cameras and real-time pose tracking to recognize full-body movements and footwork with greater precision. Similarly, Chunithm, a rhythm game by Sega, relies on advanced hand-tracking through infrared or camera-based systems to detect fast, complex gestures like air slashes or upward swipes. While not all the technical details of these proprietary systems are public, they demonstrate how CV, including neural network-based recognition and 3D hand pose estimation, has become viable even in high-speed arcade settings.

In the context of our rhythm game, the evolution of CV in games informs both the possibilities and limitations of the technology. While early systems like EyeToy and Kinect paved the way for body-based interaction, modern pose estimation frameworks enable a more flexible and accessible approach without requiring specialized hardware. Understanding this history allows us to design a system that is both innovative and grounded in proven interaction paradigms.

#### 3.1.4.3 Pose Estimation Techniques

##### 3.1.4.3.1 Classical Techniques

Before the widespread adoption of deep learning, pose estimation was achieved through traditional computer vision methods, many of which relied on handcrafted features and rule-based pipelines. These classical techniques were computationally efficient and suitable for the hardware available at the time, but they came with strict environmental limitations and a lack of generalization across diverse scenes.

One of the earliest and most intuitive techniques used in classical pose estimation was background subtraction. This method attempts to isolate the moving subject from a static background by identifying changes in color or brightness between frames. Typically, a “background model” is captured or assumed, and then each new frame is compared pixel-by-pixel to detect motion. Basic versions used simple frame differencing, while more advanced versions employed Gaussian Mixture Models (GMMs) to handle subtle lighting fluctuations or gradual background changes. In a constrained environment this method could effectively identify silhouettes or regions of interest, forming the basis for further analysis like gesture recognition or contour tracking. However, background subtraction quickly broke down in real-world scenarios. Environmental noise such as flickering lights, moving shadows, or dynamic backgrounds like waving trees introduced errors. Furthermore, it required the camera to be stationary, as even slight camera movement could disrupt the entire model. While it was a useful tool for early motion tracking experiments, its fragility under real conditions made it impractical for generalized use, especially in games or public environments.

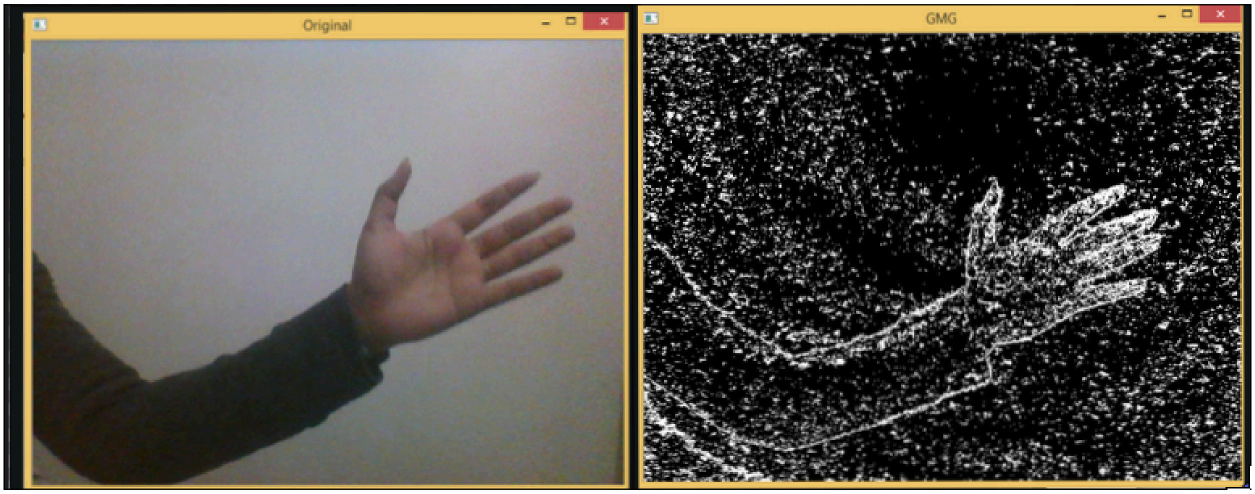



***Figure 3.7*** *Example of Background Subtraction with OpenCV*

Another classical technique that provided more temporal insight into motion was optical flow. Instead of comparing single frames against a background, optical flow analyzes the apparent movement of pixels between consecutive frames to estimate motion vectors. This technique is especially useful for capturing the direction and velocity of motion, making it ideal for analyzing limb movements or detecting walking and waving gestures. The two primary approaches being dense flow and sparse flow offered different trade-offs between resolution and performance. Optical flow has been used in robotics and video analysis to infer actions from continuous motion, and in some early gesture-controlled systems. However, it came with its own set of limitations. Large or fast movements often led to errors due to motion blur or discontinuities, and occlusion, when one part of the body hides another, posed a major challenge. Moreover, flow estimation becomes ambiguous in areas with uniform texture or lighting, where pixel intensity doesn't change enough to provide directional clues. In the context of multi-person scenes or complex poses, optical flow was insufficient on its own, often requiring additional heuristics or model-based constraints to be effective.

Contour detection and shape matching techniques sought to extract body outlines by identifying edges and fitting known templates or convex hulls. These methods enabled rough body pose estimation but failed in the presence of overlapping limbs, varied clothing, or non-standard postures. Similarly, pose-from-silhouette techniques relied on the shape of a segmented figure to infer likely joint locations, using statistical models of human anatomy. These systems performed reasonably well for frontal or side views but were limited by the inherent ambiguity of silhouettes in 2D projections.

Among classical methods, pictorial structures marked a significant conceptual shift by introducing a more structured way to represent human pose. Rather than analyzing motion or edges in isolation, they use a computer science graph-like representation where nodes correspond to body joints and edges represent limbs. Each part was associated with a probability distribution describing its likely appearance, and the spatial relationships between parts were encoded through geometric constraints. The system would then search for the configuration that maximized the overall likelihood across the image, essentially solving an optimization problem to infer the most plausible pose. This approach was robust to some extent against partial occlusion and varying camera angles, making it more suitable than silhouette or contour-based methods for challenging environments. However, it came at a cost. The inference step was computationally expensive, especially for full-body models with many degrees of freedom. These systems also relied heavily on strong contrast and clean segmentation to reliably detect features in the first place. As a result, while pictorial structures provided a powerful theoretical foundation for understanding pose, their practical use was limited until more efficient algorithms and stronger feature representations emerged in the deep learning era.

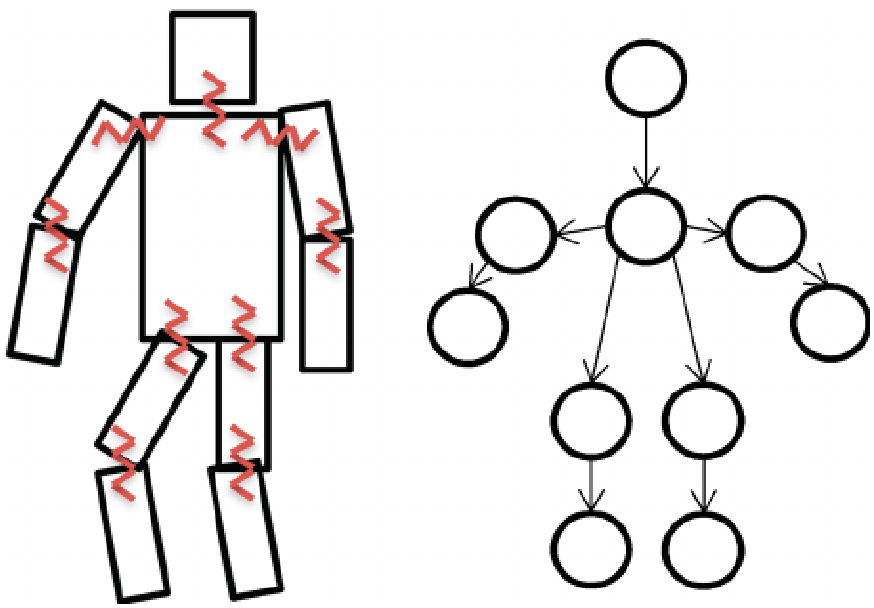


***Figure 3.8*** *Graph representation of a typical pose for a human body*

A major step forward came with the introduction of the Microsoft Kinect in 2010, which helped bridge the gap between classical vision pipelines and modern learning-based systems. The Kinect utilized structured infrared light to capture depth data in addition to standard RGB imaging. This enabled accurate 3D skeletal tracking without requiring wearable sensors or markers. Microsoft's implementation used machine learning decision forests trained to identify joint positions from depth images, allowing for real-time, multi-user tracking on modest consumer hardware such as the Xbox 360. Though highly influential, Kinect's depth-based approach suffered from hardware limitations, including limited range, sunlight interference, and platform dependency.

Despite their constraints, these classical techniques laid the groundwork for the learning-based methods that followed. They demonstrated the feasibility of markerless tracking and inspired early applications in gaming, gesture control, and human-computer interaction. The lessons learned from these systems informed the design of modern neural network architectures and dataset collection strategies used in today's pose estimation models.

#### 3.1.4.2 Modern Techniques

As hardware improved and large annotated datasets became available, convolutional neural networks (CNNs) emerged as the dominant paradigm for pose estimation. These models replaced handcrafted pipelines with data-driven learning, enabling greater robustness, scalability, and accuracy.

The emergence of CNNs has drastically transformed the field of human pose estimation, enabling high-accuracy, real-time detection of key body joints from standard RGB imagery. Unlike classical techniques that relied on handcrafted features or background modeling, modern CNN-based systems can generalize to a wide variety of poses, environments, and occlusion conditions through extensive training on large-scale annotated datasets.

One of the most influential convolutional neural network-based pose estimation frameworks is OpenPose[19], developed by the Carnegie Mellon Perceptual Computing Lab. OpenPose was the first open-source system capable of performing real-time, multi-person 2D pose estimation with relatively high accuracy. It uses a method called Part Affinity Fields, which predicts not only the positions of individual body key points such as wrists, elbows, and knees, but also the directional associations between these keypoints. This allows the system to effectively group detected joints into separate individuals, even when multiple people are present in the same scene or overlapping in view.

While OpenPose set a major milestone in pose estimation research, its accuracy comes with significant computational demands. It typically requires a dedicated GPU in order to achieve real-time performance, and its performance degrades considerably on devices without powerful hardware. As a result, OpenPose is not well-suited for mobile, embedded, or browser-based environments where memory, power, and processing capabilities are limited. The framework is built on Caffe and makes extensive use of OpenCV for tasks such as image preprocessing, rendering keypoints, drawing skeletal lines, and managing post processing routines. This design makes it modular and relatively easy to modify, but also heavy and resource-intensive compared to more modern, lightweight alternatives.

In practice, many of the tasks handled by OpenPose’s post processing pipeline can be re-implemented with simpler code using OpenCV and basic geometric rules. For example, drawing connections between keypoints based on distance thresholds or grouping joints into a rough pose can be achieved using a few lines of logic, especially when used in combination with another library like MediaPipe to handle keypoint detection. This makes OpenPose seem unnecessarily complex for applications that only require basic gesture detection, such as identifying if someone is raising their hands, jumping, or standing still.

***Figure 3.9*** *OpenPose working on a large amount of subjects*

In contrast to OpenPose, MediaPipe[21] Pose was developed by Google with an emphasis on speed, efficiency, and platform flexibility. It was designed to operate on low-powered devices without sacrificing too much accuracy. The system uses a two-stage convolutional neural network pipeline. First, it detects the region of interest around a person using a lightweight detector. Then, it applies a landmark model to predict thirty-three key points across the entire body, including finer details such as fingers, feet, and facial landmarks.

What sets MediaPipe Pose apart is its ability to deliver real-time performance even on CPUs and embedded systems. It runs reliably on mobile phones, Raspberry Pi boards, and NVIDIA Jetson devices without the need for a dedicated GPU. Its modular architecture is built around a graph-based processing model, where each component in the pipeline is represented as a node. This allows developers to customize, replace, or extend parts of the pipeline, such as input normalization, inference logic, or output smoothing.

In addition to its lightweight design, MediaPipe includes smoothing filters and motion tracking features that reduce jitter and increase temporal stability. This makes it especially suitable for applications involving fast movements or inconsistent lighting. Although it may not match OpenPose in multi-person detection or sub-pixel keypoint accuracy, its practical balance between speed, resource usage, and flexibility makes it an ideal choice for mobile games, real-time interaction, and low-latency gesture recognition.

OpenPose remains a powerful benchmark in academic and industrial settings where multi-person detection, fine-grained keypoint accuracy, and research flexibility are prioritized over efficiency. Its ability to consistently detect multiple overlapping subjects and produce detailed skeletons makes it valuable for high-end applications such as motion capture, live broadcasting, and scientific analysis. However, its reliance on GPU acceleration, heavy memory footprint, and large model size make it impractical for use in resource-constrained environments, such as mobile games or embedded systems. MediaPipe Pose offers a more balanced solution by maintaining reasonably high accuracy while optimizing for speed, modularity, and ease of deployment. Its ability to operate smoothly on CPU-based systems allows developers to use it in real-time applications where latency must be minimized and where external hardware acceleration is not feasible. Furthermore, its graph-based architecture and wide platform support (including Android, iOS, and desktop) enable developers to prototype and iterate across a broad range of devices with minimal changes to the underlying codebase.

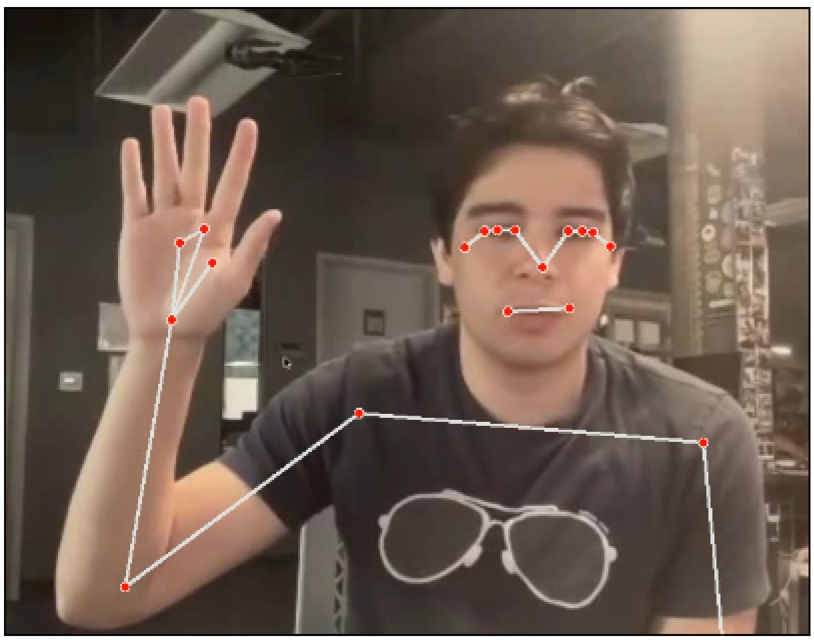


***Figure 3.10*** *Example prototyping of OpenCV and MediaPipe on Christopher Solanilla*

For scenarios demanding ultra-fast performance and even lighter computational demands, BlazePose[20] takes this optimization a step further. Its use of single-shot detection eliminates the need for separate region proposals or sequential processing steps, significantly reducing latency. Combined with model quantization and neural architecture search, BlazePose achieves performance that exceeds 30 frames per second on modern smartphones using only CPU resources. This makes it highly suitable for applications in fitness tracking, augmented reality (AR), and gesture-based interfaces where responsiveness and battery efficiency are critical. Additionally, its prediction of 33 keypoints, including subtle facial and foot positions, enables more expressive pose understanding despite its lightweight design. The most intriguing feature of BlazePose is that it is able to predict within a high accuracy x, y, and z coordinates for problems that may require 3D location.

In the context of our rhythm-based game, where low latency, reliable body tracking, and smooth animations are essential for gameplay responsiveness, both MediaPipe Pose and BlazePose represent ideal choices. Given that the game must run on commodity hardware without access to high-end GPUs, the selection of a lightweight model is not only a technical preference but a hard requirement. MediaPipe's modular pipeline offers a flexible starting point for tuning the pipeline to match the rhythm game's input structure and timing constraints, while BlazePose introduces an opportunity to push performance boundaries even further if frame rate becomes a bottleneck. These models align well with the constraints of real-time play and allow us to maintain a consistent experience across platforms, from desktops to embedded consoles.

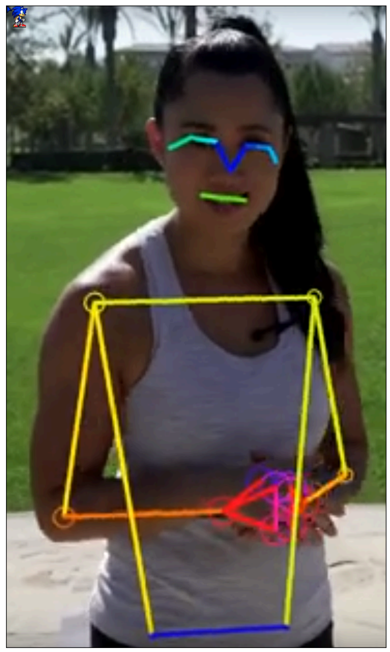


***Figure 3.11*** *BlazePose demonstrating high performant pose tracking capabilities with 3 dimensions*

#### 3.1.4.4 Prototyping with MediaPipe and OpenCV

To better understand the practical limitations and advantages of MediaPipe Pose, early prototyping work was conducted prior to this project using a custom-built C++ application developed for a separate accessibility-focused project. This project, titled *Body Language Detector*, was created as part of a hackathon hosted by my workplace. Its goal was to explore the use of computer vision in assisting visually impaired users by interpreting body language in real time. The application used a webcam feed to identify and analyze nonverbal cues, such as hand gestures and body posture, and translate these visual signals into auditory or textual feedback. The project aimed to give blind or low-vision users more access to subtle social signals that are typically communicated without speech.

To meet the performance requirements of real-time interaction, the application was developed in C++ and used GTK for the graphical interface. This allowed for a native, low-overhead UI that could display pose information and system feedback without introducing latency. OpenCV was used to handle image processing and visualization tasks, while MediaPipe Pose provided the backbone for keypoint detection. Rather than relying on Python, which would have simplified development but introduced performance overhead, the entire pipeline was compiled using Bazel to ensure smooth operation on Linux systems. Particular care was taken to support CPU-only inference, making the program compatible with resource-constrained environments such as embedded systems or older computers. The system was tested on a Gentoo-based Linux setup, demonstrating stable performance and reliable keypoint detection without the need for external GPU acceleration.

As seen in Figure 3.9, the prototype successfully tracked the user's full-body pose using a standard webcam. Landmarks such as the shoulders, elbows, knees, and wrists were recognized in real time, and gesture logic was applied by calculating the relative distances and angles between these points. For example, the application could detect whether a person had raised both arms, crossed their arms, or shifted their weight to one side. These features allowed for basic pose classification and helped shape an understanding of what kinds of gestures could be reliably detected with lightweight pose estimation frameworks. The experience of building gesture classifiers using simple geometric heuristics informed the rhythm game's input design, where fast, unambiguous pose recognition is essential for gameplay accuracy and responsiveness.

The project is available on a public GitHub repository and includes build instructions, MediaPipe submodules, and example code for evaluating gestures using OpenCV and landmark data. Although the original purpose of the tool was to improve accessibility for users with vision impairments, the prototyping process provided valuable insights into the deployment of real-time pose estimation systems on constrained hardware. Specifically, it helped highlight the trade-offs between model complexity and responsiveness, and demonstrated the effectiveness of MediaPipe's filters and tracking logic in noisy or unpredictable environments. These lessons directly informed the technical direction of the project. Both offer reliable, low-latency pose estimation that is well-suited for interactive applications running on systems without high-end graphics hardware.

#### 3.1.4.5 Considerations for Embedded Systems

For this project, integrating computer vision into a rhythm game requires careful consideration of the limitations imposed by potential embedded platforms such as the Nvidia Jetson While the Jetson Nano offers a relatively capable computing platform with integrated GPU acceleration via its 128-core Maxwell GPU, it does not match the capabilities of desktop-class hardware. Therefore, key decisions surrounding model architecture, inference optimization, runtime configuration, and thermal design must be made in order to maintain a responsive and stable user experience.

#### 3.1.4.5.1 Hardware Capabilities of the Jetson Nano

The NVIDIA Jetson Nano serves as a compact and cost-effective embedded computing platform aimed at enabling edge AI applications. It is built around a quad-core ARM Cortex-A57 CPU operating at up to 1.43 GHz, paired with a 128-core Maxwell GPU capable of executing parallel computations required by deep learning inference. The board is equipped with 4 GB of LPDDR4 memory, which is shared between the CPU and GPU, making memory contention a critical factor in system performance. This unified memory architecture reduces the overhead of copying data between processors but also introduces constraints when both the vision pipeline and the game engine are competing for memory bandwidth and capacity.

Despite its modest specifications compared to desktop-class GPUs or higher-tier Jetson modules such as the TX2 or Xavier NX, the Nano is capable of running real-time inference workloads when models are properly optimized. However, it cannot support large-scale convolutional neural networks without suffering from memory exhaustion, cache contention, or reduced throughput. Pose estimation systems that rely on deep neural networks, particularly those involving multiple stacked layers or high-resolution heatmaps, can quickly push the Jetson Nano to its limits. OpenPose, for example, typically requires over 2 GB of RAM just for model inference at full resolution, and relies on multi-branch processing stages to estimate keypoints and limb associations. These demands make it infeasible to deploy OpenPose in its standard form without significant pruning, quantization, and architectural simplification. Even in highly constrained configurations, OpenPose achieves only 3 to 5 frames per second on the Jetson Nano, rendering it unsuitable for real-time interactive use in latency-sensitive applications like rhythm games.

In contrast, lightweight alternatives such as MediaPipe's BlazePose or Pose models are specifically designed to operate under embedded constraints. MediaPipe relies on a streamlined architecture that detects a single person per frame using a detector-tracker approach, typically involving an initial region-of-interest (ROI) proposal followed by a landmark regression stage. These stages are implemented as modular nodes within a dataflow graph that can be reconfigured or pruned depending on the use case. By reducing input resolution, disabling optional filters, and leveraging fixed-point arithmetic, the MediaPipe pipeline can be tuned to achieve 15 to 25 frames per second on the Jetson Nano, even when running alongside other processes such as camera capture and basic game logic.

Furthermore, the Jetson Nano’s GPU can accelerate some portions of the pipeline through CUDA, though MediaPipe does not natively integrate with NVIDIA TensorRT, which limits the ability to fully offload inference from the CPU unless custom GPU kernels are developed. Nevertheless, the combination of a streamlined pose estimation model, configurable processing graph, and low-memory footprint makes MediaPipe a viable candidate for real-time single-user gesture recognition on this hardware platform.

### 3.1.4.5.2 Software Stack and Runtime Optimization

Computer vision applications on the Jetson Nano are typically developed using the NVIDIA JetPack SDK, which includes CUDA for GPU computation, cuDNN for deep learning operations, and TensorRT for inference optimization. While these tools are powerful, they are not directly compatible with every pose estimation framework. MediaPipe, for example, is not natively integrated with TensorRT and instead relies primarily on CPU inference or custom GPU pipelines. This can restrict its ability to fully leverage the Nano's GPU unless a custom build is created and compiled specifically for Jetson's architecture. In contrast, OpenPose requires significant manual configuration to function on the Jetson Nano. This includes aligning CUDA versions, recompiling OpenCV with GPU support, and configuring swap space to avoid out-of-memory crashes. These dependencies and the resulting fragility of the system make OpenPose impractical for a rhythm game targeting consistent frame rates and predictable behavior.

MediaPipe provides a more stable and lightweight alternative. It offers precompiled binaries for both Python and C++ that support CPU inference, and its modular graph-based processing structure allows developers to disable optional components to reduce resource consumption. When paired with inference backends such as TensorFlow Lite or the ONNX Runtime, MediaPipe can run efficiently on the Jetson Nano with minimal adjustments. The flexibility of this architecture is particularly advantageous when attempting to tailor the pose detection pipeline to balance latency, precision, and power usage.

### 3.1.4.5.3 Thermal and Power Management Constraints

In embedded systems development, particularly those involving computer vision workloads, power consumption and heat generation become critical engineering concerns that must be addressed early in the design process. The NVIDIA Jetson Nano provides two selectable power modes: a default 5-watt mode and a more performance-oriented 10-watt mode, which requires a barrel-jack power supply for stable operation. While enabling the higher power mode improves the responsiveness of both neural network inference and general system tasks, it also introduces higher thermal output, which can become problematic in the absence of sufficient cooling mechanisms. Continuous execution of convolutional neural network models, such as those used for pose estimation, alongside a concurrently running rhythm game engine, places sustained computational stress on the device, increasing the risk of thermal throttling if the system exceeds its safe temperature thresholds.

To mitigate thermal buildup and avoid performance degradation over time, the Jetson Nano requires active cooling solutions such as compact fans or aluminum heatsinks. Even with such additions, software design must complement the hardware limitations by adopting strategies that minimize unnecessary computational overhead. For instance, reducing the frequency of camera frame polling or selectively enabling vision processing only during gameplay segments that require it can dramatically lower heat output. The rhythm game being developed does not rely on continuous full-body pose monitoring at all times; instead, it only requires pose classification during specific gameplay prompts that challenge the player to perform a particular movement. This creates an opportunity to disable or suspend the pose estimation system during periods when it is not in use, thereby reducing CPU and GPU workload and allowing the device to operate in a lower-power state for the majority of gameplay.

Such an approach aligns well with the real-time demands of rhythm games, where pose detection can be activated just prior to a prompt and then evaluated in a short time window before being deactivated again. This intermittent activation pattern not only supports power savings but also simplifies thermal management, since the vision pipeline is no longer running continuously in the background. Moreover, additional efficiency can be gained by limiting the input resolution of the camera, lowering the target frame rate for pose estimation to around 15 frames per second, or batching non-time-critical processing tasks so they execute during quieter moments of gameplay. These strategies allow for more predictable thermal behavior and reduce the likelihood of the device reaching critical temperatures that trigger throttling or system instability

#### 3.1.4.5.4 Game Engine Integration and Resource Synchronization

The rhythm game is being developed using the Godot Engine, which introduces additional integration considerations. Godot allows for C++ extensions through GDExtension or GDNative, making it possible to incorporate MediaPipe's C++ API directly into the game. However, careful synchronization between the engine and the pose estimation system is critical. Since both the game and the pose detector may require access to OpenGL contexts or video memory, concurrent usage can result in resource contention or graphical glitches if not managed properly. To avoid such conflicts, pose estimation should be executed in a dedicated thread or separate process. Pose data can then be shared with the game engine using inter-process communication or memory-mapped files, ensuring that Godot can operate smoothly without interruptions caused by the vision pipeline.

This architectural decoupling also helps to minimize the impact of intermittent performance drops in the vision system. Since pose detection and gameplay operate semi-independently, short delays in processing frames do not immediately disrupt game logic. Instead, previously detected poses can be held until the next update cycle, maintaining consistency for gameplay elements that rely on accurate gesture input.

#### 3.1.4.5.5 Memory Footprint and Latency Targets

The Jetson Nano’s 4 GB of RAM must support the operating system, the game engine, camera buffering, model inference, and other runtime operations. MediaPipe’s pose detection typically consumes between 100 and 200 megabytes of memory, depending on image resolution and whether smoothing filters are enabled. Additional RAM is used by the video capture pipeline, graphical assets, and internal Godot processes. Without proper profiling, it is easy to overcommit memory and cause performance to degrade unpredictably.

For rhythm-based gameplay, responsiveness is crucial. The full pipeline from camera input to pose interpretation and game response should ideally remain under 150 milliseconds of latency. Delays beyond this threshold can disrupt the timing-sensitive nature of rhythm gameplay and break player immersion. Achieving this target requires efficient memory management, parallelization of processing stages, and reduction of unnecessary overhead in both vision and game logic subsystems.

#### 3.1.4.5.6 Summary and Design Tradeoffs

Based on the platform analysis, MediaPipe remains the most viable pose estimation solution for the NVIDIA Jetson Nano. It balances performance, configurability, and platform compatibility in a way that suits the needs of embedded game development. While OpenPose offers more detailed multi-person tracking, its memory requirements and complex setup disqualify it from practical use in this environment. BlazePose, as an extension of the MediaPipe ecosystem, provides an additional fallback option in cases where even greater inference speed is necessary or if resource usage exceeds acceptable thresholds. By carefully managing resource allocation, thermal conditions, and runtime behavior, it is possible to build a responsive and reliable rhythm game experience that leverages pose estimation on embedded hardware without sacrificing performance.

#### 3.1.4.6 Programming Languages for Game and Computer Vision

The selection of programming languages for this project plays a central role in shaping both the development workflow and the real-time performance of the computer vision and game engine components. Given the nature of embedded systems and the integration of computer vision models for pose detection, careful language choices must be made that reflect both the technical demands of the system and the development constraints faced by the team.

For the computer vision subsystem, both C/C++ and Python are widely adopted in the field and are supported by key libraries such as OpenCV and MediaPipe. Python offers a fast and accessible prototyping experience due to its dynamic typing, extensive library ecosystem, and concise syntax. MediaPipe, in particular, provides precompiled Python wheels that allow for rapid experimentation and integration with OpenCV, making it an ideal language for early-stage development and algorithm testing. However, Python’s interpreted nature and higher runtime overhead make it less suitable for performance-critical or resource-constrained deployments, especially on embedded hardware like the Jetson Nano.

In contrast, C/C++ offers significant advantages in terms of execution speed, memory control, and portability, particularly for embedded and production environments. MediaPipe’s core is written in C++ using the Bazel build system, which provides high performance but also introduces greater complexity during compilation and integration. The C/C++ API grants lower-level access to graph construction, buffer management, and custom kernel development, enabling developers to fine-tune pipelines for optimal performance. However, setting up a MediaPipe C/C++ environment on Linux, especially for cross-compilation or GPU acceleration, can be challenging and may require in-depth knowledge of build systems and dependency management.

Given the tradeoffs between flexibility and performance, many projects begin with Python for rapid prototyping before transitioning to C++ for deployment. However, due to prior experience working with MediaPipe, OpenCV, and embedded C++ development, we are able to bypass the initial Python prototyping phase and move directly into a C++ implementation. This decision allows us to focus our efforts on building a production-ready system from the outset while maintaining the performance and low-level control necessary for real-time computer vision on the Jetson Nano. Although Python remains a powerful tool for testing and experimentation, our familiarity with the C++ development pipeline enables faster integration, better performance tuning, and more efficient use of system resources.

The rhythm game itself is being developed in the Godot Engine, which uses GDScript, a high-level, dynamically typed language with Python-like syntax, designed specifically for Godot's architecture. While GDScript is expressive and well-integrated with Godot's node system, it does not offer the low-level performance or external library compatibility needed for high-throughput computer vision. Therefore, GDScript will be used primarily for game logic and UI, while pose estimation will be handled externally in C/C++ and communicated to the game engine via inter-process communication (IPC) or shared memory. If deeper integration becomes necessary, Godot supports C/C++ bindings through GDExtension, enabling MediaPipe's C/C++ modules to interface directly with the engine while maintaining real-time performance.

### 3.1.4.7 Evaluation Criteria

The computer vision system in this rhythm-based dance game serves as an event-driven input mechanism rather than a continuously running subsystem. Its role is to detect specific player poses at designated moments defined by the game's chart or music track. While it is not active at all times, it remains an essential component during gameplay segments that involve pose-based challenges. To function effectively in this context, the system must meet key requirements in terms of latency, accuracy, robustness, and integration. These criteria ensure the pose detection system can reliably support time-sensitive interactions without introducing disruptions to the player's experience.

#### 3.1.4.7.1 Latency

Timing is one of the most important aspects of rhythm game design. For players to feel connected to the beat and confident in their inputs, the delay between a physical gesture and the system's recognition of that gesture must be imperceptible. For this application, the maximum allowable end-to-end latency is defined as 100 milliseconds.

This measurement begins at the moment the camera captures a frame and ends when the pose data is successfully delivered to the game engine for evaluation.

This latency budget includes image acquisition, any pre-processing applied to the frame, neural network inference for pose detection, post-processing steps like keypoint filtering or pose smoothing, and the transfer of results into the game logic. Delays beyond 100 milliseconds can create a noticeable disconnect between the player's actions and the game's response. At 150 milliseconds, players with strong rhythm sensitivity may perceive the system as being off-tempo or unresponsive.

To minimize latency, the pose detection pipeline must use lightweight models such as BlazePose or a stripped-down version of MediaPipe Pose. Frame resolutions should be kept moderate, and smoothing filters should be configured for minimal delay. Where possible, inference should be performed asynchronously on a separate thread. Hardware acceleration through CUDA or TensorRT, if compatible, can further reduce inference time and help maintain a consistent frame rate.

**3.1.4.7.2 Accuracy**

Accurate pose recognition is required to ensure the system scores player input fairly and consistently. Errors in classification, including false positives where poses are detected incorrectly or false negatives where valid poses are missed, directly impact gameplay quality. Incorrect detections can result in dropped combos, missed prompts, or undeserved penalties, reducing player satisfaction and overall trust in the game.

To address this, the pose estimation system should be configured to detect only a small set of predefined poses used in gameplay. These can be manually defined or trained using a dataset that includes common variations. Confidence thresholds should be enforced to prevent low-quality predictions from being processed. Temporal validation, where poses must be held across multiple frames to register, can help eliminate flickering caused by brief or unstable detections.

In scenarios where transfer learning is feasible, the pose detector can be fine-tuned using a small number of labeled examples that reflect the specific movement patterns and player behaviors expected in the game. This improves classification accuracy without the need to retrain an entire model from scratch.

**3.1.4.7.3 Robustness**

The game is expected to operate in a range of real-world conditions, meaning the vision system must be resilient to variability in lighting, background clutter, clothing, and camera positioning. Environments may include bedrooms, living rooms, classrooms, or arcades, each introducing different visual challenges.

Lighting changes can introduce shadows or glare. Players may wear loose, patterned, or dark clothing that affects keypoint visibility. Camera placement may vary slightly in height, distance, or tilt. To maintain stability across these conditions, the pose estimation system must be robust. Models that are invariant to scale and orientation, particularly those using heat maps or landmark regression, perform better in uncontrolled environments.

Testing should include augmenting the input video stream with artificial noise, brightness variation, and occlusion to evaluate the system's tolerance. Ideally, the pose detector should not require frequent recalibration and should support dynamic input normalization during runtime.

#### 3.1.4.7.4 Integration with the game engine

In a rhythm-based game where player gestures are a core part of gameplay, the computer vision system must integrate cleanly with the game engine to avoid introducing delays, instability, or data inconsistency. The pose detection pipeline should operate in parallel with the main game loop, allowing the game engine to remain responsive and frame-accurate even under high processing load.

A practical approach is to run the vision system in a dedicated thread or as a separate process. This design reduces the likelihood of memory contention and helps isolate crashes or slowdowns in the vision pipeline from the rest of the game. Data such as keypoint coordinates, joint angles, or classified pose states can be communicated through shared memory buffers, local sockets, or lightweight message-passing protocols.

The interface between the vision module and the game logic should be structured and minimal. Pose data should arrive in a predictable format with consistent timing to allow the game engine to make real-time decisions, such as triggering animations, updating score counters, or initiating visual effects. Synchronization mechanisms such as frame counters, time stamps, or event queues can help ensure that pose detections are correctly matched to the corresponding frames of gameplay. When evaluating different CV pipelines, attention should be paid to their ability to support such integration models. Systems that allow for modular runtime graphs, asynchronous processing, and efficient memory management are typically better suited for integration in real-time game environments.

### 3.1.5 Game Engines

#### 3.1.5.1 Godot

Godot is a very useful game engine that is beginner friendly, is open source, and has a lot of documentation as to how to use each function available in Godot. Furthermore due to Godot being open source, there are a lot of third party tutorials and walkthroughs regarding Godot teaching and guiding users how to accomplish various tasks for different types of games. These tutorials include rhythm games which is what we will be working on.

Godot's game engine seems very easy to not only install, but also use once it has already been installed. Godot's ability to create various scenes, that of which hold a variety of assets such as sprites, nodes, sound, animations and other useful assets seamlessly make it very simple to use. Godot’s user interface also is very simple to understand as one does not become overwhelmed.

Godot also has its own scripting language called GDScript, which is very similar to Python which is used to be able to change various aspects of each object being made. Due to Godot’s scripting language being based off of python, GDScript is a comfortable language to understand especially for game scripting beginners who don't have a lot of programming experience, specifically users without prior game design programming knowledge. Through using GDScript, there are various functionalities that can be done using it such as changing the positioning of an object or sprite across the y-axis or x-axis which we will be using for the notes that will follow the beat. Another attribute that can be changed using GDScript, are the colors of the objects after a certain button(s) is/are pressed which we will be also using for the color changes of the arrows when they are pressed.

Another fantastic feature that Godot has, is the compatibility mode that is used when creating our project. Due to us having a microcontroller that will not have as much computing power as a modern computer would have, having this as a feature will lower the load that the microcontroller would need to output.

We also have a member who has used Godot in the past and due to their experience with Godot we had this option as a priority, however we checked other sources as well to test which would be best for the objective we are trying to complete.

In the end of checking which would be our best option to completing our objective, we decided that Godot would be our game engine of choice for this specific project.

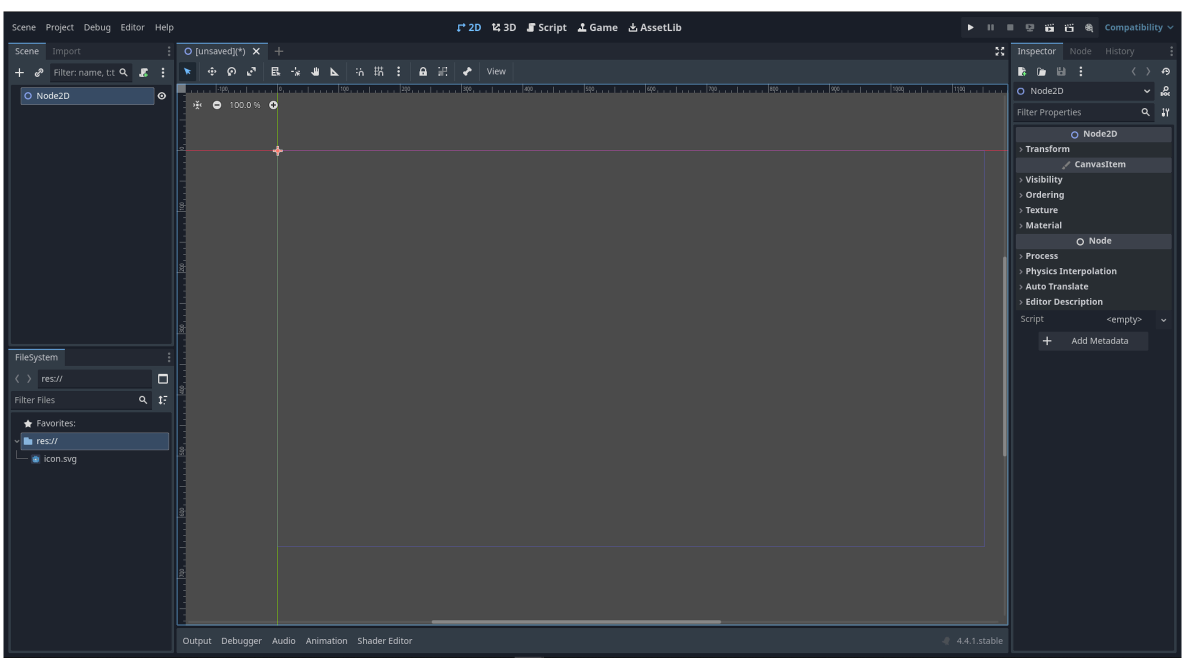



***Figure 3.12*** *Godot default interface*

Godot's interface is very simple to understand. The nodes that will be on a specific scene is located on the top left side where the scene tab is, the assets that are implemented throughout the entire project will be located right below that in the file system tab, and when clicking on a specific node, their properties will be located on the right side in the inspector tab.

**Scene Tab**

The scene tab is located by default on the top left  of the Godot's user interface. The scene tab hosts all of the different nodes that are located on the specific scene that is selected. These nodes that can be added to the scene could range from different sprites, to different types of polygons some of which can add collision detection, and even audio files that can be played while the scene is being played. Afterwards, once all the different nodes are placed on the scene selected, one could add a script onto each of those individual nodes that allows modifications to the objects.

These modifications could range from the color of the polygon, the position of the sprite, or even a custom property that one chooses to add to the object. Another way that script can be attached, is by attaching them to one of the scenes as a whole. Doing so will affect any and all objects that are inside the scene. This feature makes it easier to modify a scene as a whole rather than individually modifying each object that exists in the scene and makes the process seamless.The scene tab is very useful to keep one organized as to what is going on in the scene and be able to see where everything is positioned.

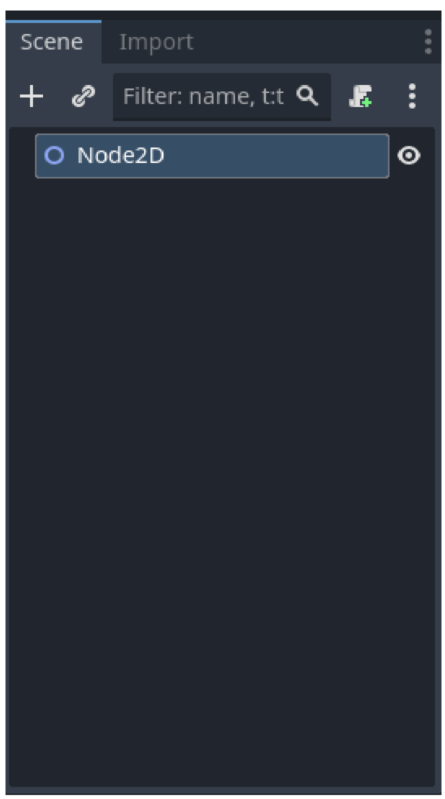



***Figure 3.13*** *Scene Tab*

**File System Tab**

The file system holds all of the assets that are needed for the project. This includes any scripts that are being run, any scenes that are made, any sprites that are imported, and much more.

The file system tab also allows for the creation of various folders for better organization for the project. Having multiple assets be bunched up makes working on it confusing, however utilizing the file system the folders make it easier to search for a specific group of assets such as one for how an arrow functions or searching for how the menu interface is set up, makes it seamless to update certain assets when needed.

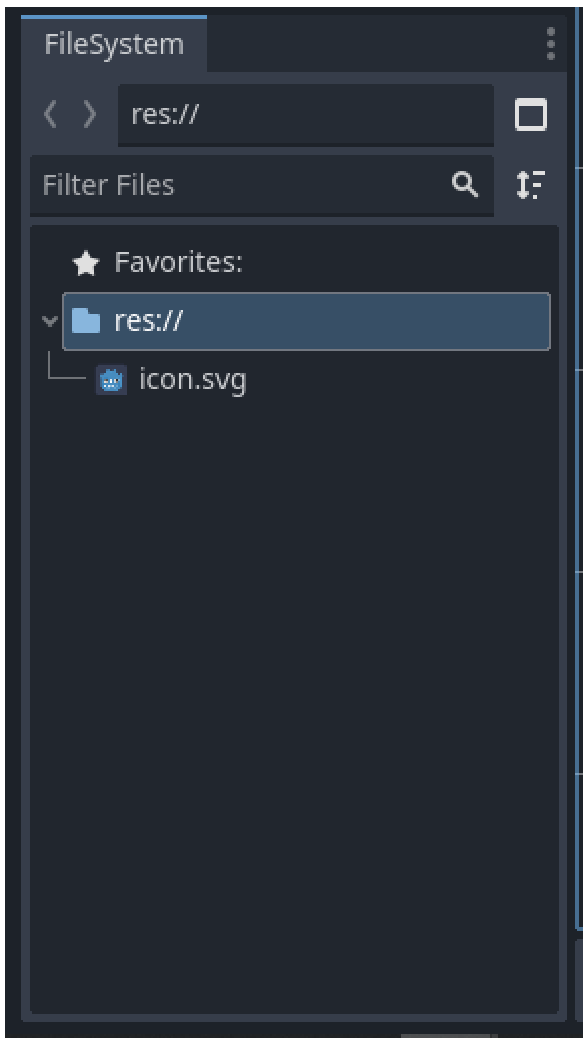


***Figure 3.14*** *FileSystem Tab*

**Inspector Tab**
The inspector tab holds all the information in regards to a certain object that is being highlighted. Whether that be an imported sprite, a custom made polygon that has the collision feature attached to it, or even an audio source that needs to play music, the inspector tab will have that information stored for users to be able to make modifications manually to certain attributes.

For example, users can decide that they would like to change the positioning of a certain sprite or polygon, under the transform dropdown one could access the various transformations that one could make such as moving across the x-axis or move across the y-axis. Users can also decide that they would like to rotate an object a certain way, this can also be done in the transform section.

There are many other attributes that users can add to the object selected through the inspector tab such as scripts that were mentioned before.

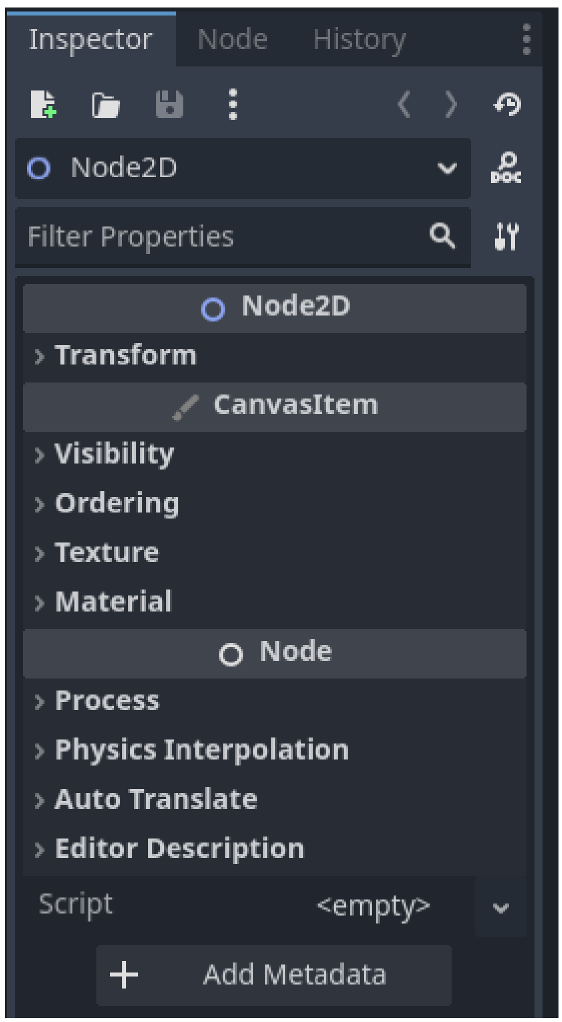



***Figure 3.15*** *Inspector Tab*

### 3.1.5.2 Unity

Unity is another game engine that exists that could be used to make 2D games. Unity however is more focused on 3D games rather than 2D games. Some features such as the Tilemap system, the SpriteShape, and pixel-perfect rendering exist that could help us create our 2D rhythm game that we are trying to accomplish.

Unity also offers audio synchronization tools and a timeline that could make it easier to have both the player actions along with the musical beats to be in tune, which is a key factor in terms of how the game should function while it is running.

The downside of using Unity, is that for scripting, it uses a language called C# which is a quite advanced language to use and isn't a very beginner friendly scripting language for game building. Even though it is a very powerful language to learn, its learning curve is very high without prior experience using the language.

Furthermore, due to how large unity is in terms of their features, it becomes very overwhelming understanding what unity has in store vs what it does not have in store. After opening it for the first time, Unity has a variety of libraries that come installed with it causing the engine to take a long time to open for the first time.

One good attribute that Unity has, is the built in version control called Unity Version Control. While it is a good feature to have, due to us being a group of 5 people, it would cost us an additional $14 monthly to use their built in system.

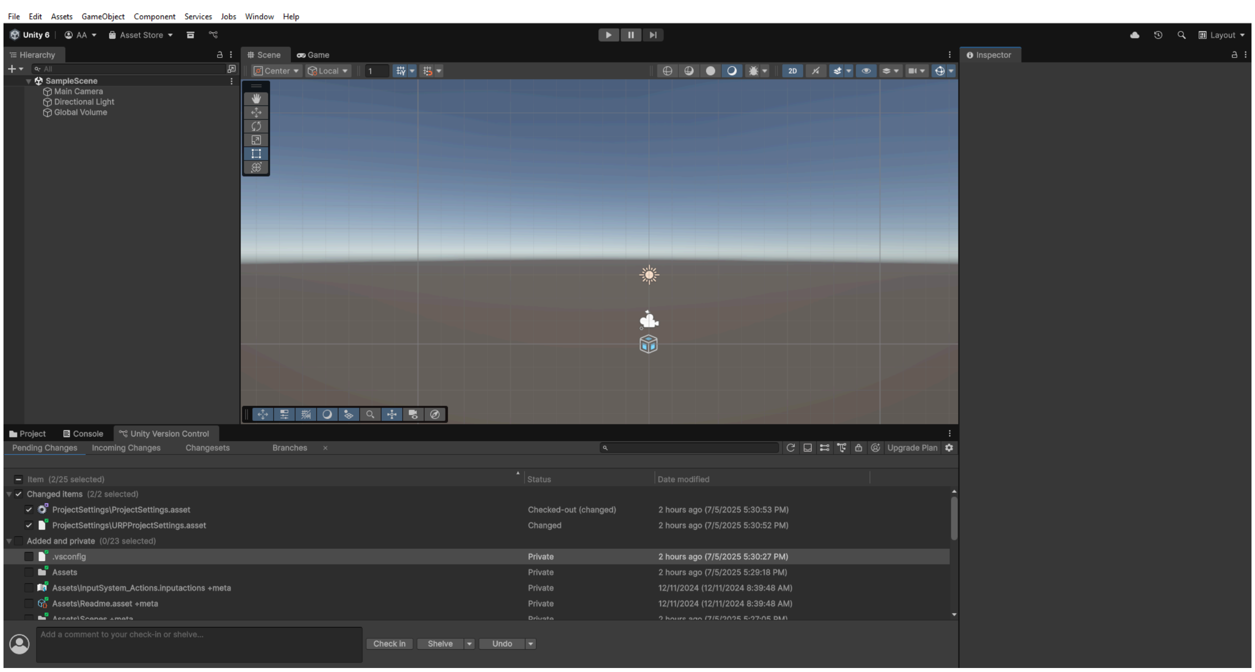



***Figure 3.16*** *Unity default interface*

#### 3.1.5.3 Unreal Engine

Unreal Engine is a game engine that one can install using the Epic Games launcher created by Epic Games themselves that is mainly used for 3D games. They are mostly known for being used for AAA game development which are games that are very large with many assets attached to the game mostly made in the 3D space. However, Unreal Engine has added the capabilities to create 2D games through their “Paper 2D” functionality allowing to only see the space in the 2D perspective rather than adding the Z-direction. This functionality allows 2D sprites to be made and other 2D assets that could be used for a 2D game being developed on their game engine.

One of the downsides to using Unreal Engine however, is that although it has tutorials on how to create games, all of the ones that they offer on their website are mostly geared towards those who are aiming to create a 3D game. Since we are making a 2D game, those tutorials do not work for us entirely. There are however some aspects of those tutorials that we could use such as their scripting of sprites and event handling. Unreal Engine also has the same problem that Unity does which is the load that it has on the system.

### 3.1.6 Version Control

#### 3.1.6.1 Github

Github is a very popular version control manager that allows multiple people to work on a project at a given time. By using their Git Repositories, users can access all of their uploaded documents through their cloud service. This allows for users to work on one computer, upload any and all documents or files they need for their project, and be able to work on another computer and retrieve all the documents they need to continue working. Furthermore, since Github is indeed a version control manager, it is able to retrieve past versions of projects in the case that a mistake was made that caused the entire project as a whole to crash. This allows restoring the project to a previous state that it was in. However one must be careful when doing so as if not done so properly could be detrimental causing hours if not days of progress to be lost.

Another useful feature that github has is its ability to work on separate branches than the main one being used. What these branches allow one to do is work on a specific portion of the project separately using the same main project source code and make modifications without modifying the main source code. This is extremely useful for us due to the separate portions that we will be working on and not wanting to make mistakes that others would have to fix later on. To add on to the branch, users can also merge projects together, allowing for branches that one worked on to be combined with the main project code and be updated. This allows for example one person to make changes to the main project code based on one feature, and once done, the user is able to merge with the main code that others have access to as well. One must be very careful with merging as there is a possibility of overrides that can cause projects to crash, so one must be sure that the merging procedure is done correctly so that mistakes do not appear at a later time.

For our project we will be using Github mainly for the game design portion as we will have 2 people working on it simultaneously.

### 3.2 Part Selection

#### 3.2.1 MCU

After comparing a wide range of microcontroller options, the Teensy 4.1 is the most optimal choice for our dance pad system. It outperforms other MCUs like Arduino boards (Uno, Mega, Leonardo), Raspberry Pi Pico, ESP32 variants, and even most STM32 chips in the areas that matter most to our design: real-time responsiveness, native USB HID support, memory size, and I/O capacity.

Unlike Arduinos, which are easy to use but limited in speed (16 MHz) and memory, Teensy 4.1 runs at 600 MHz with a powerful ARM Cortex-M7, giving us extremely low latency, which is critical for rhythm game input timing. Compared to ESP32 boards, which offer Wi-Fi/Bluetooth and good general performance, Teensy 4.1 has faster USB communication (480 Mbps) and far better real-time determinism. While the Raspberry Pi Pico is extremely affordable, it lacks native USB HID and has fewer ADC inputs, requiring extra circuitry. Even STM32 boards, while flexible and powerful, generally require more setup and may not match Teensy 4.1’s combination of ease of use, speed, and built-in USB HID support.

With 55 GPIO pins, 35 PWM outputs, 18 ADC inputs, 8 MB of flash, and 1 MB of SRAM, Teensy 4.1 comfortably supports our 9 dance pad panels, RGB LEDs, LED panel, and signal processing without needing extra expanders. Although its processing power may be more than necessary, this overhead ensures room for future scalability (e.g. two-player pads, more lighting effects).

Overall, Teensy 4.1 provides the best performance-to-complexity ratio, meeting our real-time, I/O, and USB communication needs while remaining compact, efficient, and development-friendly.

#### 3.2.2 Dance Pad Sensor

Upon analyzing all contending sensors for our project, the most optimal choice is clearly the force-sensing resistor (FSR). FSRs are the most commonly used sensors in both DIY dance pad builds and even in many modern arcade dance machines. They are affordable and reliable components that offer fast response times, are simple to wire (usually requiring just a voltage divider circuit), and can detect both types of inputs required by our system: quick impacts and sustained press-and-hold steps.

Their flexibility and thin profile also make them ideal for fitting under lightweight or low-profile panel designs. As long as our physical construction ensures proper placement and force distribution, we can effectively minimize their main drawbacks: potential dead zones, non-linearity, and slightly lower accuracy compared to load cells or strain gauges. Another major advantage of FSRs is cost. Being relatively inexpensive, they allow for cost-effective replacements if damaged during testing or extended use, keeping long-term maintenance costs low. This aligns with our project's goals of affordability and maintainability.

FSRs can be easily wired and pair well with Teensy 4.1. Much like Teensy 4.1, the FSR's simplicity contributes to lower system complexity, enabling us to focus more on refining input responsiveness, game logic, and RGB LED feedback. Overall, FSRs offer the best balance of performance, simplicity, and cost for our dance pad sensor needs.

***Table 3.5** shows features of the four mentioned types of sensors and how they compare to one another*
*(Legend: ↑↑ = very high; ↑ = high;* M = moderate; ↓ = *moderate/low; ↓↓ = very low)*

| Feature | FSRs | Load Cells | Strain Gauges (Raw) | Piezoelectric Sensors | Break Beam Sensors |
|---|---|---|---|---|---|
| Size | ↓↓ | ↑ | ↓↓ | ↓ | M |
| Cost | ↓↓ | ↑ | ↓ | ↓ | ↓ |
| Response Time | ↑ | ↑ | ↑ | ↑↑ (impact) | ↑↑ |
| Accuracy | ↓ | ↑↑ | ↑↑ | ↑ (impact) | ↓ |
| Complexity | ↓↓ | ↑ | ↑↑ | M | ↓ |
| Durability | ↓ | ↑↑ | M | ↓ | ↑ |
| Impact Input | ✓ | ✓ | ✓ | ✓ | ✓ |
| Sustain Input | ✓ | ✓ | ✓ | | |

### 3.2.3 Camera Comparison

Several camera modules were considered to balance resolution, frame rate, and price. The chosen camera needed a pixel pitch suitable for resolving ~ 1mm features, a frame rate of at least 30 FPS for smooth gameplay, a cost preferably under $100 to fit system goals, and compatibility with the 850nm illumination. The  Arducam AR0234 USB 2MP global shutter camera was identified as a strong candidate. Its 3.0 μm pixel pitch, RGB output, and use of no IR-cut filter makes it ideal  to properly sense the NIR illumination. The global shutter also significantly reduces motion blur, which is important during fast player movement.[29]

Through more research, a lower-cost alternative using the same AR0234 sensor was found through SVPRO. The SVPRO 2MP Global SHutter USB camera module offers similar specifications, including global shutter operation, M12 lens compatibility, and RGB output, but for a better price. Its lower cost makes it more ideal for the prototype phase. However, it may have reduced support and documentation compared to the Arducam-branded product. The Raspberry Pi HQ camera was also considered because of its modular CS-mount compatibility, but its rolling shutter and low frame rate made it less suitable for pose estimation. Overall, the AR0234 offers a balanced trade-off between pixel size, cost, and motion capture. The specifications for these candidate cameras are summarized in Table 3.2.3

***Table 3.6*** *Camera comparison table*

| Camera | Pixel Pitch(μm) | Frame Rate(FPS) | Price (USD) | Shutter Type |
|---|---|---|---|---|
| Raspberry Pi HQ (IMX477) | 1.55 | 20 | $53.78 | Rolling |
| Arducam AR0234 Global Shutter | 3.0 | 60 | $109.99 | Global |
| SVPRO AR0234 Global Shutter | 3.0 | 60 | $76.99 | Global |

### 3.2.4 Lens Comparison

A wide field of view and short focal length were prioritized to capture the entire 114’’ x 114’ dance pad area from approximately 1.8m away. Research into wide-angle M12 lenses focussed on models with a horizontal field of view near 93°, focal lengths around 3 mm, minimal distortion, and a price under or as close to $100 as possible.[12] These lenses are well-suited for compact vision systems, and the M12 mount allows easy replacement and adjustment during prototype testing.

Although the SVPRO AR0234 camera module includes pre-installed wide-angle M12 lens, alternative 3 mm M12 aspherical lenses will be evaluated to verify critical requirements such as distortion, IR transmission, and image sharpness for the 850 nm active illumination. This ensures the final lens selection supports consistent MediaPipe landmark detection and meets the project's performance targets. Commercially available aspherical M12 lenses with focal lengths around 3 mm were identified as the best balance of wide-angle coverage and low distortion. Aspherical designs help reduce spherical aberration and maintain sharpness across the entire field of view, which is important for MediaPipe pose tracking and consistent landmark detection.[12] Although a fully custom-designed aspherical lens could in theory optimize every optical parameter, the cost and fabrication complexity made that impractical for thai prototype. Therefore, an off the shelf aspherical wide-angle M12 lens will be the baseline option.

After evaluating multiple M12-mount spherical lenses compatible with the sensor, the Commonlands CIL329 was the most appropriate. This lens has a 2.8 mm focal length with a132° diagonal field of view, approximately 100 - 110° HFOV on the AR0234. Although the CIL 329's specified image circle is slightly undersized for the AR0234 sensor's diagonal, a quick compatibility check indicates it should still provide sufficient coverage for this prototype. This field of view allows the camera to capture the full 114'' x 114'' dance pad area from a working distance of 1.8m without excessive mounting height or tilt. Its aspherical design supports reduced geometric distortion, which is beneficial for MediaPipe tracking, while remaining compatible with typical project constraints on cost and mechanical mounting. Two additional lenses were also evaluated for comparison and their specifications are summarized in table 3.7. Both lenses provided compatible focal lengths and apertures but did not have the horizontal field of view needed.

***Table 3.7*** *Lens options and their specifications*

| Lens | Focal Length(mm) | Horizontal FOV (°) | Aperture (f/#) | Price (USD) |
|---|---|---|---|---|
| Commonlands CIL034 | 3.2 | ~75 | 2.4 | $39 |
| Commonlands CIL329 | 2.8 | ~120 | 2.4 | $39 |
| Commonlands 036 | 3.3 | ~75 | 2.2 | $19 |

### 3.2.5 LED Part Comparison

The main selection criteria for these strips was their ability to provide sufficient radiant flux to support reliable MediaPipe detection, a wide beam angle near 120°for consistent coverage, and moderate power consumption to fit within the available 12 V supply constraints. Three candidates were compared and their specifications were summarized in Table 3.8. The Waveform IRFlex 850nm, DC12/24V 5050 SMD strip, and the 360 Digital signature 3528 SMD strip. The 5050 SMD option provides both high output and a moderate price point for 60 LEDs per meter and a power draw of approximately 14.4

W/m. 3528 SMD offers lower power consumption but has a higher price per reel making it less ideal. The 2835 SMD strip provides the most LED density at 120 LED/m and has the lowest power draw of about 9.6W/m, making it the most premium candidate of the three at $55 per 5m.

All three options maintain a wide beam angle near 120°, supporting robust coverage of the player’s body during movement. Based on these tradeoffs such as cost and power, the 3528 SMD strip was selected as the baseline for initial prototyping. This is due to its cost balance, radiant power, and easy 12 V integration with time-multiplexed control zones. Final LED strip selection will be confirmed after prototype testing of illumination uniformity and camera sensitivity.

***Table 3.8*** *LED Strip comparison table*

| Feature | Waveform IRFlex 850nm | DC12/24V | 360 Digital Signage |
|---|---|---|---|
| LED Type | 2835 SMD | 5050 SMD | 3528 SMD |
| Beam Angle | 120 | 120 | 120 |
| LED Density (/m) | 120 | 60 | 60 |
| Power(W/m) | 9.6 | 14.4 | 7.2 |
| Price ($) | $55 | $28.98 | $45.89 |

### 3.2.6 Embedded System Development Languages

We are using C language to program our Teensy 4.1 microcontroller because it offers the most efficient, direct, and low-overhead way to interact with the hardware components of our system, specifically the FSRs, RGB LEDs, and USB HID interface. C provides precise control over memory, peripherals, and timing, which is crucial for our real-time step detection and LED response requirements. Unlike higher-level languages, C allows us to write low-latency code that can directly manipulate GPIO pins, read analog signals from FSRs via ADC channels, and update LED states with minimal delay.

Teensy 4.1 supports C (and C++) natively through the Arduino IDE and Teensyduino extension, allowing us to easily configure the board for USB HID communication, which is essential for translating foot inputs into instant keyboard-like signals recognized by the game. Since our system is latency-sensitive and must respond with near-instant feedback for both impact and press-and-hold inputs, using C gives us the deterministic performance we need. Overall, C is the most appropriate language for programming the embedded side of our dance pad system, offering speed, reliability, and full access to the hardware's capabilities.

### 3.2.7 Computer Vision Library Selection

After evaluating several options for implementing pose detection and real-time vision-based input, we have selected MediaPipe and OpenCV as the primary libraries for our computer vision subsystem. This decision reflects both technical capabilities and our prior development experience.

MediaPipe, developed by Google, offers a lightweight and efficient pose estimation pipeline that can run on commodity hardware, including embedded systems such as the Jetson Nano. It supports real-time inference with acceptable latency, making it suitable for our rhythm game's timing-sensitive requirements. MediaPipe provides prebuilt models such as BlazePose, which can detect and track 33 body landmarks with a balance of accuracy and speed. Its modular graph-based architecture allows customization of the processing pipeline, enabling us to skip unused stages and fine-tune performance parameters as needed.

OpenCV complements MediaPipe by offering a wide range of image processing utilities, camera input handling, and matrix operations that are useful for gesture classification and data smoothing. It also provides tools for visualization, geometric transformations, and real-time filtering, all of which are valuable during debugging and refinement of the pose recognition system.

Our previous work with these libraries in related projects, such as a pose-driven accessibility tool, has shown that they are reliable, portable, and efficient when compiled natively in C++ using Bazel. By combining MediaPipe's pose estimation models with OpenCV's low-level processing capabilities, we avoid the need to train models from scratch and instead benefit from a well-supported and mature ecosystem.

# 4. Standards and Design Constraints

### Standard: IPC-2221A – Generic Standard on Printed Board Design

The **IPC-2221A, "Generic Standard on Printed Board Design,"** serves as the foundational standard for the design of printed circuit boards (PCBs). Published by IPC, the global electronics industry association, this document establishes the generic requirements for the design of organic printed boards, from single-sided boards to complex multilayer structures. It provides a robust framework of rules and principles that govern nearly every aspect of PCB layout, ensuring that the final product is not only functional and reliable but also manufacturable. For any electronic design project, applying the principles of IPC-2221A is a critical step in translating a schematic diagram into a physically sound and dependable product.

***Core Philosophy: Design for Manufacturability (DFM)***

At its core, IPC-2221A is a guide for Design for Manufacturability (DFM). It creates a common language and a set of baseline requirements between the designer and the fabrication house. By establishing rules for spacing, feature sizes, and tolerances, the standard ensures that a design can be reliably produced using standard manufacturing processes, which helps to control costs, improve yields, and reduce production time. The standard also defines three performance classes based on the intended end-use environment, which dictates the stringency of the design rules:

- **Class 1 (General Electronic Products):** For applications where the primary requirement is the function of the completed assembly.
- **Class 2 (Dedicated Service Electronic Products):** For products requiring high reliability and an extended service life, where uninterrupted service is desired but not critical. This is the target class for the dance pad project.
- **Class 3 (High Performance/Harsh Environment Electronic Products):** For mission-critical products where continued high performance or performance-on-demand is essential.

***Electrical Design Considerations***

IPC-2221A provides extensive guidance on the electrical aspects of PCB design to ensure both signal integrity and user safety.

- **Conductor Spacing (Clearance):** One of the most critical safety considerations in PCB design is the spacing between conductive elements. Insufficient spacing can lead to dielectric breakdown (arcing) between traces, especially at higher voltages. IPC-2221A provides detailed tables (such as Table 6-1) that specify the minimum required clearance based on the peak DC or AC voltage between conductors. These requirements vary based on whether the conductors are on internal or external layers and whether they are coated. For the dance pad, which operates at a low voltage (5V), the minimum spacing requirements are easily met, but acknowledging this standard is crucial for demonstrating sound design practice.
- **Conductor Sizing for Current Capacity:** The width and thickness of a PCB trace determine its current-carrying capacity. A trace that is too small for the current it must carry will overheat due to its own resistance, which can damage the PCB laminate or cause the trace to fail entirely. IPC-2221A provides charts and formulas that relate a conductor's cross-sectional area to its temperature rise for a given amount of current. This allows designers to select an appropriate trace width to ensure the board operates within safe

thermal limits. For the power and ground traces on the dance pad's PCB, these guidelines were used to ensure they could handle the total current draw of the system without significant heating.

***Mechanical and Physical Design Rules***

Beyond electrical rules, IPC-2221A specifies the physical and mechanical characteristics of the board to ensure its structural integrity and compatibility with assembly processes.

- **Holes and Interconnections (Vias):** Vias are plated-through holes that form electrical connections between different layers of a PCB. Their reliability is paramount to the function of a multilayer board. IPC-2221A provides specific guidelines for via design, including:
    - **Annular Ring:** The annular ring is the ring of copper that surrounds a drilled hole. The standard specifies a minimum acceptable annular ring width to ensure a solid connection between the via barrel and the trace after drilling and plating, accounting for manufacturing tolerances. For Class 2 designs, a robust annular ring is required to prevent breakout, where the drill hole is not completely surrounded by the copper pad.
    - **Aspect Ratio:** This is the ratio of the board's thickness to the diameter of the drilled hole. A high aspect ratio can make it difficult to achieve reliable copper plating down the entire barrel of the via. The standard provides limits on aspect ratios to ensure manufacturability.
- **Thermal Management:** The standard addresses the need to manage heat generated by components. A common technique guided by IPC-2221A is the use of **thermal relief pads**. When a component pin needs to connect to a large copper plane (like a ground plane), a direct connection would act as a large heat sink, making it very difficult to solder the component. A thermal relief pad creates small copper spokes to make the connection, which reduces the heat transfer during soldering while still providing an adequate electrical connection.

***Documentation Requirements***

A critical function of the IPC-2221A standard is to define the requirements for a complete and unambiguous documentation package. A design is only as good as the documentation that communicates it to the manufacturer. The standard calls for a set of documents that typically includes:

- **Fabrication Drawing:** A detailed drawing that specifies the board's dimensions, layer stack-up, materials, drill hole information, and any special manufacturing notes.
- **Assembly Drawing:** Shows the location and orientation of all components on the board.
- **Gerber Files:** The industry-standard file format that describes each layer of the PCB (copper layers, solder mask, silkscreen, etc.).
- **Bill of Materials (BOM):** A complete list of all components to be mounted on the board.

By following the documentation guidelines of IPC-2221A, the design intent for the dance pad's electronics is clearly and professionally conveyed, minimizing the risk of manufacturing errors.

***Sources***


- IPC International, Inc. "IPC-2221A: Generic Standard on Printed Board Design." May 2003.
- Coombs, Clyde F. *Printed Circuits Handbook*. 6th ed., McGraw-Hill, 2008.
- "IPC-2221 Standards in PCB Design." Sierra Circuits, Inc.


**Standard: Universal Serial Bus (USB)**

The **Universal Serial Bus (USB)** is a ubiquitous industry standard that establishes specifications for cables, connectors, and communication protocols for connection, communication, and power supply between computers and peripheral devices. Its development was intended to standardize the connection of computer peripherals, thereby replacing a multitude of legacy interfaces. For this project, adherence to the USB standard is critical for ensuring interoperability and a seamless user experience.

***USB 2.0 Specification***

While several revisions of the USB standard exist, the **USB 2.0 specification** was selected for this project due to its widespread compatibility and sufficient data throughput for the required application. USB 2.0 provides a maximum data signaling rate of **480 Mbit/s (High Speed)**, which is more than adequate for transmitting the state changes of the dance pad's nine input tiles. Furthermore, the standard specifies a bus power supply of 5V, with a high-power device permitted to draw a maximum of **500mA**, a critical constraint for the system's power budget.

***USB Human Interface Device (HID) Class***

A key component of the USB specification relevant to this project is the **Human Interface Device (HID) class**. The HID class is a device class specification that defines a protocol for low-latency, low-power peripherals, such as keyboards, mice, and game controllers. The primary advantage of utilizing the HID class is that it obviates the need for custom host-side drivers, as support is natively integrated into modern operating systems, including Windows, macOS, and Linux.

Device functionality is communicated to the host computer through a series of **descriptors** during the enumeration process:

- **Device Descriptors:** Identify the peripheral with a unique Vendor ID (VID) and Product ID (PID).
- **HID Descriptors:** Specify that the device conforms to the HID class.
- **Report Descriptors:** This is the most critical descriptor for this project. It defines the structure and format of the data packets, known as **reports**, that the device will send to the host. For the dance pad, the report descriptor will be configured to define a data structure containing the boolean state of each of the nine input tiles, effectively representing them as buttons.

Once enumerated, the device transmits data to the host via **Input Reports** over a dedicated interrupt-based endpoint, ensuring that state changes (i.e., tile presses and releases) are communicated with minimal latency.

***Sources***


- USB Implementers Forum (USB-IF). "Device Class Definition for Human Interface Devices (HID)." Version 1.11, 27 June 2001. https://www.usb.org/hid
- USB Implementers Forum (USB-IF). "Universal Serial Bus Specification." Revision 2.0, 27 April 2000.


## Standard: FCC Part 15 – Unintentional Radiators

The Federal Communications Commission (FCC) establishes regulations to manage electromagnetic interference under **Title 47, Part 15 of the Code of Federal Regulations**. This standard governs the operation of radio frequency (RF) devices without requiring an individual license. Any electronic device incorporating digital logic and operating with clock frequencies greater than 9 kHz is capable of generating electromagnetic energy. While not its primary purpose, this energy can radiate from the device and cause interference with radio communications. Such devices are classified by the FCC as **unintentional radiators**.

### *Device Classification: Class B*

FCC Part 15 defines two primary classifications for unintentional radiators based on the intended market and environment:

- **Class A:** For devices used in commercial, industrial, or business environments. The limits on radiated and conducted emissions are less restrictive.
- **Class B:** For devices intended for use in residential environments. The emissions limits are significantly more stringent to prevent interference with consumer electronics such as televisions, radios, and Wi-Fi networks.

Given that the 9-tile dance pad is a consumer-oriented gaming peripheral, it falls under the **Class B classification**. Therefore, the design must consider stricter emission limits applicable to residential devices.

### *Design for Compliance and EMI Mitigation*

While formal FCC certification is beyond the scope of this academic project, incorporating design principles aimed at minimizing electromagnetic interference (EMI) is a fundamental aspect of professional engineering practice. The following strategies have been considered in the design of the device's printed circuit board (PCB) and overall system architecture:

- **PCB Ground Plane:** The PCB is designed with a large, contiguous ground plane. This provides a low impedance return path for digital signals, which minimizes the area of current loops that can act as efficient radiating antennas.
- **Decoupling Capacitors:** Small ceramic capacitors are placed physically close to the power and ground pins of the microcontroller and other integrated circuits. These capacitors serve as a local charge reservoir and shunt high-frequency noise from the power distribution network to the ground plane, preventing its propagation.
- **Signal Integrity:** The slew rates of high-speed digital signals are controlled where possible. Sharper signal transitions contain higher-frequency harmonic content, which can contribute to radiated emissions.
- **Cable Shielding:** A shielded USB cable will be specified for connecting the device to the host computer. The shield helps to contain common-mode noise generated by the device's circuitry and prevents the cable from acting as an antenna.

By implementing these EMI mitigation techniques, the design proactively addresses the requirements of FCC Part 15, ensuring the device operates as a responsible and non-interfering electronic product.

***Sources***

- United States, Code of Federal Regulations, Title 47, Part 15. "Radio Frequency Devices."
- Federal Communications Commission. "OET Bulletin 62: Understanding the FCC Regulations for Low-Power, Non-Licensed Transmitters." October 1993.

## Standard: UL 62368-1 – Hazard-Based Safety for ICT & AV Equipment

The **UL 62368-1** standard represents a modern, hazard-based approach to product safety for Information and Communication Technology (ICT) and Audio/Video (AV) equipment. Published by Underwriters Laboratories (UL), this standard shifts from a traditional incident-based model to a proactive, performance-based methodology focused on identifying potential hazards and implementing safeguards to mitigate them. As the dance pad is an electronic peripheral that directly interfaces with both a host computer and a human user, ensuring its safety in accordance with established principles like those in UL 62368-1 is a primary design objective.

***Hazard-Based Safety Engineering (HBSE) Principles***

UL 62368-1 is built on the principles of Hazard-Based Safety Engineering (HBSE), which involves a three-step process:

1. **Identify Energy Sources:** Pinpoint all potential energy sources within the product.
2. **Classify Energy Levels:** Categorize the energy sources based on their potential to cause pain or injury to a user or damage to the equipment.
3. **Implement Safeguards:** Apply appropriate safeguards to protect users from any energy sources classified as hazardous.

For the dance pad project, the primary energy sources are electrical and mechanical.

***Application and Design Safeguards***

While this academic project will not undergo formal UL certification, the design philosophy of UL 62368-1 has been integrated into the development process to ensure user safety.

- **Electrical Hazard Mitigation:** The device is powered exclusively by a 5V DC source via a standard USB 2.0 port, which is classified as an ES1 (Electrical Energy Source Class 1) environment under UL 62368-1. This low voltage is not considered hazardous and does not present a risk of electric shock. To protect against potential short circuits or overcurrent conditions, the custom printed circuit board (PCB) design incorporates a resettable Polymeric Positive Temperature Coefficient (PPTC) fuse. This device automatically interrupts the circuit in an overcurrent event and resets when the fault is cleared, providing robust protection against fire hazards. All internal wiring is appropriately insulated and secured to prevent abrasion or pinching.

- **Mechanical Hazard Mitigation:** The dance pad is subjected to significant and repeated mechanical stress during gameplay. The enclosure is constructed from high-impact polycarbonate and plywood; materials selected for durability and resistance to fracture. All external corners and edges of the enclosure are rounded to a radius of no less than 3mm to prevent cuts or injuries from sharp edges. The internal structure is designed to distribute impact forces, ensuring the assembly remains mechanically sound and that no internal components can become dislodged and create a secondary hazard.

By adopting the hazard-based principles of UL 62368-1, the design of the dance pad prioritizes user safety, addressing potential electrical and mechanical risks through deliberate material selection and the implementation of appropriate safeguards.

### *Sources*


- UL Standards. "UL 62368-1: Audio/video, information and communication technology equipment - Part 1: Safety requirements." UL.com.
- UL Standards. "Hazard-Based Safety Engineering (HBSE) & UL 62368-1." UL.com.


## Standard: RoHS – Restriction of Hazardous Substances

The **Restriction of Hazardous Substances (RoHS) Directive**, originating in the European Union, is a critical environmental and health-focused standard for the electronics industry. The directive (specifically RoHS 3, Directive 2015/863/EU) restricts the use of ten specific hazardous materials in the manufacture of various types of electrical and electronic equipment. Although this project is not intended for commercial sale in the EU, voluntary adherence to RoHS principles represents responsible engineering practice, minimizing environmental impact and ensuring the final product is safe for users and for disposal.

### *Restricted Substances*

The RoHS directive restricts the following substances to a maximum concentration of 0.1% by weight (1000 ppm), apart from Cadmium, which is limited to 0.01% (100 ppm):

- Lead (Pb)
- Mercury (Hg)
- Cadmium (Cd)
- Hexavalent Chromium ($Cr^{6+}$)

- Polybrominated Biphenyls (PBB)
- Polybrominated Diphenyl Ethers (PBDE)
- Four specific Phthalates (DEHP, BBP, DBP, DIBP)

***Implementation in Project Design and Assembly***

A conscious effort was made throughout the procurement and assembly phases to ensure the components and materials used in the dance pad are RoHS compliant.

- **Component Procurement:** All active and passive electronic components, including the microcontroller, resistors, capacitors, connectors, and diodes, were sourced from reputable distributors. During selection, components were explicitly filtered and verified to be "RoHS Compliant" based on manufacturer's datasheets. This ensures that the fundamental building blocks of electronics are free from restricted hazardous substances.
- **Printed Circuit Board (PCB) Fabrication:** The custom PCBs for the project were manufactured by a fabrication house that offers a RoHS-compliant manufacturing process. This guarantees that the PCB substrate, solder mask, and surface finish (e.g., ENIG - Electroless Nickel Immersion Gold, or lead-free HASL - Hot Air Solder Leveling) do not contain restricted materials.
- **Soldering and Assembly:** The most significant step taken to ensure RoHS compliance during in-house assembly was the exclusive use of **lead-free solder**. Instead of traditional tin-lead (SnPb) solder, a tin-silver-copper (Sn-Ag-Cu, or SAC) alloy was used for all soldering tasks. While lead-free solder requires higher working temperatures and presents different wetting characteristics, its use eliminates the most common hazardous substance found in electronics assembly, protecting both the assembler and the environment.

By adhering to the RoHS directive, this project demonstrates a commitment to modern, environmentally conscious design standards that extend beyond immediate functional requirements.

***Sources***


- European Commission. "Restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS)."
- "RoHS Guide." Mouser Electronics.


## Standard: IPC-A-610 – Acceptability of Electronic Assemblies

The **IPC-A-610, "Acceptability of Electronic Assemblies,"** is the most widely recognized global standard for the workmanship and quality of printed circuit board assemblies (PCAs). Developed by the Association Connecting Electronics Industries (IPC), this standard provides comprehensive, visually supported criteria for evaluating the quality of electronic assemblies. Adhering to the principles outlined in IPC-A-610 is essential for ensuring the long-term reliability and functionality of the dance pad custom electronics. A failure in a solder joint or a misplaced component could lead to intermittent or total failure of the device during use.

### *Workmanship Classification*

IPC-A-610 defines three classes of product quality, reflecting the intended life cycle and operational environment of the assembly:

- **Class 1 (General Electronic Products):** For products where the primary requirement is the function of the completed assembly.
- **Class 2 (Dedicated Service Electronic Products):** For products requiring extended reliability and continued performance. Uninterrupted service is desired but not critical.
- **Class 3 (High Performance/Harsh Environment Electronic Products):** For products where continued high performance or performance-on-demand is critical.

For this project, **Class 2** was established as the target quality standard. As a game controller subject to physical impact and repeated use, a high degree of reliability is necessary to ensure a consistent user experience.

### *Application of IPC-A-610 Criteria*

The criteria of IPC-A-610 were applied during the hand-assembly and inspection phases of the custom electronics.

- **Soldering Quality:** All solder joints were created with the goal of meeting Class 2 acceptability criteria. This includes ensuring proper wetting on both the component lead and the PCB pad (with a contact angle of less than 90°), forming a concave solder fillet, and avoiding common defects such as cold joints, disturbed joints, excess solder, or solder bridging between adjacent pads.
- **Component Placement:** Meticulous care was taken during component placement to ensure correct orientation for polarized components, such as diodes and electrolytic capacitors. All components were placed centrally on their respective land patterns on the PCB before soldering, as specified by the standard.
- **Post-Assembly Inspection and Cleaning:** Following assembly, each board underwent a thorough visual inspection using magnification and proper lighting. This inspection verified that solder joints were acceptable and that no foreign object debris (FOD), such as stray wire clippings or solder balls, was present on the board. Furthermore, the boards were cleaned with isopropyl

alcohol to remove flux residue, which can be corrosive over time and can lead to electrical leakage paths, in alignment with IPC standards for post-assembly cleanliness.

By targeting IPC-A-610 Class 2 workmanship, the project ensures that the assembled electronics are not only functional but also robust and reliable, capable of withstanding the demands of their intended application.

***Sources***

- IPC International, Inc. "IPC-A-610H: Acceptability of Electronic Assemblies."
- "Understanding IPC-A-610." EMSG Inc.

### 4.1.1 IEC 62471 - Photobiological Safety

When designing the LED panels, it was important to verify whether the system adhered to relevent photobiological safety standards. IEC 62471 is an internationally recognized framework standard that defines safety evaluation criteria for optical radiation, including visible, ultraviolet, and infrared sources. The purpose of this standard is to ensure systems protect human eyes and skin from photobiological hazards associated with artificial light sources, such as LEDs. IEC 62471 classifies systems into risk groups based on their radiometric output and potential health hazards, ranging from Exempt (RG0) to Risk Group 3 (high risk). [14].

For S.T.E.P, 850 nm infrared LED strips are being used to provide active illumination for pose detection. Since 850nm is near-infrared and partially outside of the visible range, it is still covered by IEC 62471, which encompasses wavelengths between 200 nm and 3000 nm.[] The LEDs being used are placed approximately 1.8 meters from players. According to industry summaries of IEC 62471, systems using 850nm LEDs typically fall within Risk Group 1 provided their radiant exposure stays below the established safety thresholds. [14]

Although minimizing the radiant flux is recommended to comply with IEC 62471, in this project the actual risk is very minimal. The 850nm LEDs are partially visible, mostly safe, and extremely common in consumer motion-tracking systems. The player distance of 1.8meters, combined with time-multiplexing and wide distribution of LEDs, means that the design itself is very low risk and well within Risk Group 1. No hazardous exposure conditions are expected.

### 4.1.2 ISO 9241-210 - Ergonomics of human-system interaction

Another important standard was ISO 9241-210. ISO 9241-210 defines principles for designing interactive systems with a focus on user comfort, safety, and effectiveness.[13] This standard is relevant to the illumination system because it emphasizes minimizing discomfort. Glare, and distractions. important that the system does not affect the player's comfort. Avoiding glare and minimizing distracting illumination effects is important for user comfort and effectiveness. The system will follow ISO 9241-210 by ensuring the LEDs do not produce visually uncomfortable brightness (by selecting 850nm over visible white), by time-multiplexing zones to prevent perceptible filcker, and by mounting the arrays to avoid direct illumination into the players' eyes. Together, these measures support a user-centered design that prioritizes both an enjoyable and non-distracting gameplay experience.

#### 4.1.3 IEC 60598 Luminaires

The final standard considered is IEC 60598. IEC 60598 specifies safety requirements for luminaires with respect to electrical, thermal, and mechanical hazards[15]. Although originally written for traditional lighting fixtures, its principles remain relevant due to the LED arrays function as a luminaire in the player environment. The system will follow these guidelines by ensuring properly rated wiring, insulation, and current-limiting fuses to prevent electrical shock, overheating, or mechanical failure. Given that the proposed 12V LED illumination operates at low voltage with moderate current, no additional IEC 60598 certification is anticipated beyond following standard best practices.

Finally basic electromagnetic compatibility considerations (EMC) practices were considered to ensure the LED driver circuitry does not interfere with the camera system. The PCB will include standard layout practices such as decoupling capacitors and solid grounding to reduce any risk of electrical noise affecting camera performance.

### 4.2 Optical Design Constraints

The main design constraint for the optical system is achieving a sufficient horizontal field of view (HFOV) to cover the entire dance pad array. The system must monitor a 2.9 m * 2.9 m play area from a camera mounted approximately 1.8m away. This geometric requirement translates to a horizontal field of view of approximately 93°, based on trigonometric calculations. Maintaining this coverage ensures that all nine dance pads remain consistently visible to the pose estimation algorithm at all times. Minimizing the possibility of tracking errors or missing key player landmarks. To achieve this constraint, a 3mm focal length aspherical lens was designed, balancing the need for wide-angle capture with minimal distortion and acceptable pixel resolution. This design choice guarantees that the system can consistently and reliably observe the entire interactive area without gaps, which is essential for gameplay accuracy and user satisfaction.

With a 93° FOV covering the dance pad array, each pad will occupy a relatively small number of pixels in a 1080p frame. This means there will be a trade-off between covering a large area and maintaining enough pixel density for precise pose detection. The 3 mm aspherical lens balances this trade-off by providing wide coverage while maintaining acceptable pixel density to allow MediaPipe to resolve key landmarks with adequate clarity. This constraint ensures that gameplay remains responsive and reliable, without introducing latency or missed detections.

The physical placement of the camera module is another important constraint. The camera must be positioned to achieve the target field of view while avoiding occlusions caused by player movements, and minimizing distortion. The camera is planned to be mounted at the average torso height of approximately 1-1.5m above the floor, and angled downward to encompass the entire 2.9 m * 2.9 m dance pad array, This position balances both the horizontal and vertical FOV requirements while maintaining a natural perspective of the player's posture. It also helps minimize motion blur and ensures that MediaPipe receives high-quality data without key body joints being blocked by sudden movements. Consistent, predictable mounting geometry is essential to calibrate the tracking system and maintain performance between game sessions.

Uniformity of illumination is a critical design constraint to ensure that MediaPipe's AI pose estimation functions accurately across the entire play area. A non-uniform lighting pattern with hotspots or dark corners could cause the system to lose track of the player, reducing gameplay accuracy. An issue that could arise in dimly lit rooms such as some arcade enivronments. The target uniformity is specified as at least 85%, ensuring that the lowest illuminated areas maintain at least 85% of the brightness of the most illuminated regions. To achieve this, the LED layout was designed to distribute light evenly, with time-multiplexed zone activation reducing local overheating and maintaining consistent brightness throughout the tracking space. During testing, uniformity will be verified by analyzing pixel intensities of a calibration target using the camera sensor, as a practical substitute for more complex photodiode measurements. This constraint supports a robust, repeatable, and comfortable user experience.

The power consumption of the LED-based illumination system was considered to ensure the system operated within reasonable current limits while maintaining efficient performance. Time-multiplexing was adopted to keep the instantaneous current per zone modest, simplifying the PCB layout and reducing any potential thermal buildup. By activating only one lighting zone per camera frame (at 120 fps with an 8.3 ms frame duration), and cycling through all four zones every 33 ms, the system limits instantaneous current draw 1.5A per zone instead of a peak 6A if operated simultaneously. This strategy supports simpler, more reliable driver electronics while ensuring consistent illumination without perceptible flicker for the pose detection algorithm.

## 5. Comparison of AI Engines

The goal of using various Artificial Intelligence engines is to see various aspects of information given by Artificial Intelligence and compare them with one another. One main question that was asked to the 4 engines used was: **What would be the best way to create a rhythm game for a beginner? Would using a game engine be best or are there other alternatives to complete this objective?** The reason for the prompt was to find out what would be the best way to create a rhythm game for a project of this magnitude. We have heard of game engines that existed just as Godot, Unity, Unreal

Engine, and Scratch, however we wanted to see which of those listed are beginner friendly. Furthermore we wanted to see if Artificial Intelligence would have recommended any other sources that aren't game engines that exist that are also beginner friendly that could be used to accomplish our objective. What we found were the following responses from each Artificial Intelligence engine when given the prompt above.

### 5.1 ChatGPT

```
"Using a game engine is strongly recommended for beginners because it handles many hard parts (rendering, audio, input, timing) so you can focus on the gameplay.

Top Beginner-Friendly Game Engines:
1. Unity (with C#)
✅ Strong community, lots of tutorials
✅ Built-in support for music syncing, 2D and 3D
✅ Asset Store has ready-made rhythm tools
📘 Tutorial: "How to make a rhythm game in Unity" (e.g., Brackeys' or Code Monkey's guides)

2. Godot (with GDScript or C#)
✅ Lightweight, open-source, no royalties
✅ Very beginner-friendly scripting
✅ Good 2D support
📘 Search: "Godot rhythm game tutorial"

3. Scratch (for absolute beginners or younger users)
✅ Drag-and-drop programming
✅ Great for learning the logic of timing and input

❌ Limited in terms of polish and performance"
```

(ChatGPT GPT-4, 2025)

**After looking at ChatGPT, some helpful information was given as to what we can use for the game design. Game engines are a recommended resources to use for those who are coding games as a beginner. Game engines are able to help users with difficult portions of game design such as inputs and audio. The ones that ChatGPT recommended were Unity and Godot with scratch being added as well.**

### 5.2 Gemini

```
For a beginner looking to create a rhythm game, using a game engine
is generally the best and most recommended approach. While
alternatives exist, game engines offer significant advantages in
terms of ease of use, built-in features, and community support, which
are crucial for someone starting out.
Here's why game engines are ideal and what alternatives exist:
Why Game Engines are Best for Beginners
Game engines provide a framework and tools that simplify many complex
aspects of game development, especially for rhythm games:
Audio Synchronization: Rhythm games are all about precise timing with
music. Game engines have robust audio systems that make it much
easier to load, play, and synchronize events with audio tracks. Many
even offer tools for beat mapping or have plugins that help analyze
audio to extract beat information.
Visuals and Animations: Creating the visual feedback for hits, note
scrolling, and character animations is streamlined with an engine's
visual editors and animation tools.
Input Handling: Game engines simplify capturing player input (key
presses, taps, etc.) and checking them against timed events.
Physics (if needed): While less critical for core rhythm game
mechanics, if you want any physical interactions (e.g., objects
reacting to hits), the engine's physics engine will handle it.
UI Development: Most engines have intuitive UI systems for creating
scores, combo meters, health bars, and menus.
Cross-Platform Deployment: If you eventually want your game on
different platforms (PC, mobile, web), engines often have built-in
tools for exporting to multiple targets.
Community and Resources: Popular game engines have vast communities,
tutorials, forums, and asset stores, which are invaluable for
learning and troubleshooting.
Recommended Game Engines for Rhythm Games (Beginner-Friendly)
Godot Engine:
Pros: Free and open-source, lightweight, excellent for 2D games
(which is common for rhythm games), has its own beginner-friendly
scripting language (GDScript, similar to Python), and a growing
community.
Considerations: While its audio features are good, you might need to
dive a bit into scripting for advanced synchronization.
```

```
Unity:
Pros: Extremely versatile, widely used, massive community and asset store, robust audio tools, and a good choice for both 2D and 3D.
Considerations: Can be a bit more overwhelming at first due to its vastness, uses C# for scripting (which is a good language to learn, but might be a slightly steeper curve than GDScript for absolute beginners).
GameMaker Studio 2:
Pros: Very beginner-friendly, especially if you prefer a visual scripting approach (though it also has its own scripting language, GML), excellent for 2D games, and a good option for quickly prototyping ideas.
Considerations: Less powerful for complex 3D games compared to Unity or Unreal.
```

(Google Gemini 2.5 Flash, 2025)

Google Gemini gave us a more indepth look as to why game engines would be best resource to use when creating a rhythm game. Google Gemini explains that game engines handle much of the work in regards to audio synchronization, inputs, and have communities that contain many forums, tutorials, and other useful tools for creating games. The game engines that Google Gemini recommended were similar to those of ChatGPT, including Godot and Unity, however did not mention scratch but rather mentioned GameMaker Studio.

### 5.3 Microsoft Copilot

```
1. Using a Game Engine (Highly Recommended!)
Game engines handle graphics, sound, input, and more -- all the essentials for a rhythm game.
Unity (with C#)
One of the most popular engines for rhythm games.
Tons of community tutorials and asset store packages.
Great audio timing tools via AudioSource and Timeline.
Godot (with GDScript)
Lightweight and beginner-friendly.
Has a built-in animation player and audio support.
Open-source with lots of documentation.
2. No-Code or Low-Code Tools
If coding feels overwhelming, these tools let you focus more on
```

```
design and logic.
Construct 3 or GDevelop
Drag-and-drop interfaces.
You can create rhythm-based actions with timers and animations.
Perfect for prototyping without writing code.
3. Web-Based Tools
Want to build something playable in a browser?
p5.js (JavaScript-based creative coding library)
Lets you sync visuals to audio beats.
Requires some coding, but great for creative freedom.
Scratch
Ultra beginner-friendly.
You can make simple rhythm games with sprite animations and sound
blocks.
```

(Microsoft Copilot, 2025)

Microsoft pilot wasn't a very useful tool as it gave similar information to ChatGPT and Google Gemini in regards to which game engines to use and had no deviation in regards to that information. Using a no-code design would not be useful for us and we would not be accessing the web for our project thus both section 2 and section 3 do not apply to our project.

**5.4 DeepSeek**

```
Creating a rhythm game as a beginner is an exciting challenge! Using
a game engine is overwhelmingly the best choice for a beginner.
Here's why and how to approach it:
Why a Game Engine is Best:
Abstraction of Complexity: Engines handle graphics rendering, physics
(if needed), input management, audio playback, and cross-platform
deployment. You focus on your game logic.
Built-in Tools: Visual editors, animation systems, asset managers,
and debuggers significantly speed up development.
Asset Pipelines: Easily import graphics, sounds, and music.
Community & Resources: Massive tutorials, documentation, forums, and
asset stores specifically for game engines. Crucial for learning.
Rapid Prototyping: Test core mechanics (like hitting notes to the
beat) very quickly.
```

```
Precision Timing: Engines provide high-resolution timers and
frame-independent update loops, essential for rhythm game accuracy.
Recommended Game Engines for Beginners (Rhythm Focus):
Unity (C#):
Pros: Largest community & tutorials, vast asset store, powerful &
flexible, excellent 2D & 3D support, good performance. Many rhythm
game tutorials exist.
Cons: Can feel bloated for very small projects, newer input system
has a learning curve.
Why for Rhythm: Mature audio APIs (AudioSource, AudioClip),
coroutines for timing, strong animation tools for note
highways/effects. Great for both 2D and 3D rhythm games.
Godot (GDScript - Python-like, C#):
Pros: Lightweight, fast, completely free & open-source, incredibly
intuitive scene/node system, excellent 2D support, growing community.
GDScript is easy to learn.
Cons: Smaller community/resources than Unity (but growing fast), 3D
is capable but less mature than Unity/Unreal, asset store smaller.
Why for Rhythm: Built-in AudioStreamPlayer nodes, strong signal
system for event handling (great for hit/miss events), Tween nodes
for smooth animations. Very well-suited for 2D rhythm games.
Unreal Engine (Blueprints Visual Scripting, C++):
Pros: Stunning visuals "out of the box," powerful Blueprints visual
scripting (great if you struggle with text code), best-in-class 3D.
Cons: Steeper learning curve overall, larger download/overhead, C++
is complex for true beginners, potentially overkill for a simple 2D
rhythm game.
Why for Rhythm: Robust audio system, Blueprints can be used for core
timing logic, great for ambitious 3D rhythm games (like VR).
Avoid (for a Beginner) / Use With Caution:
Pure Programming Languages (Python/Pygame, JavaScript/HTML5 Canvas,
C++/SDL, Java/LibGDX):
Pros: Maximum control, deep learning, lightweight.
Cons: You have to build everything from scratch: rendering
shapes/sprites, playing sounds precisely, handling input, managing
game states, timing. High risk of getting bogged down in low-level
details before making game progress. Audio timing precision is
notoriously tricky without engine help.
When Consider? Only if you have significant programming experience
```

```
already and specifically want the challenge of building the
underlying systems. Not recommended for a first project.
Best Approach for a Beginner Using an Engine:
Choose Your Engine: Pick Unity (broadest resources) or Godot
(simplicity, great 2D) based on your preference. Download it.
Start TINY: Your first goal is NOT "Guitar Hero Clone." Aim for: "A
single button that I have to press when a moving dot hits a line,
synchronized to a simple drum beat."
Learn Engine Fundamentals:
Follow the official "Getting Started" tutorials for your engine.
Understand core concepts: Scenes/Objects, Components/Nodes,
Transforms, Basic Scripting (C# for Unity, GDScript for Godot).
Core Rhythm Prototype:
Audio: Import a short, simple loop (e.g., a 4-beat drum pattern).
Learn to play it.
Note Spawning: Write code to spawn a visual object (a cube, circle)
at a specific time before it needs to be hit. Hardcode the timing at
first.
Note Movement: Move the spawned object towards a "hit line" at a
constant speed. Ensure its arrival time matches the beat time.
Input Detection: Detect when the player presses the designated key
(e.g., Spacebar).
Judgment: When the key is pressed, check the distance of the note
object to the hit line.
Very close (e.g., within 50ms window): "Perfect!" (Destroy note, add
score).
Close (e.g., 50-100ms): "Good!"
Too early/late: "Miss!" (Destroy note, maybe show penalty).
Feedback: Show text ("Perfect!"), play sounds, add particle effects
on hit/miss.
Refine & Expand: Once the core loop works:
Level Design: Create a system (like a text file or simple array) to
define note timings instead of hardcoding.
Multiple Lanes/Buttons: Add more hit lines and map them to different
keys (e.g., Left/Right Arrow, A/S/D keys).
Better Graphics: Replace cubes with sprites, add a background.
UI: Add a start screen, score display, combo counter.
Music: Use a full song instead of a loop. Crucially, learn about
audio offset calibration to sync your note timing perfectly with the
```

```
track in the engine.
Difficulty: Adjust note speed, timing window size, or note density.
Crucial Rhythm Game Specifics:
Precise Timing: Use the engine's high-resolution time (Time.time in
Unity, OS.get_ticks_usec() in Godot) for judgment, not frame-based
logic. Account for display latency if needed (less critical early
on).
Audio Analysis (Advanced): Initially, manually chart your note
timings by listening to the song. Later, explore tools or libraries
(like BASS, FMOD integration) for beat detection, but manual charting
is standard for precise rhythm games.
Input Lag: Be aware that monitors/TVs, audio devices, and peripherals
add lag. Calibration settings in your game are essential later on.
Alternatives ONLY if Engines Feel Too Heavy (Still Not Ideal):
Specialized Frameworks:
LÖVE (Lua): Simpler framework than raw OpenGL, but still requires
building most systems. Lua is easy to learn.
Phaser (JavaScript): Web-based framework. Good if you know JS and
want a web game. Still requires more setup than an engine.
Rhythm Game Creation Tools (Limited): Tools like "K-Shoot Mania
Editor" or "StepMania/Etterna Editor" exist, but they are for
creating charts for existing, specific rhythm game engines, not
building your own unique game from scratch.
Conclusion:
Start with a game engine (Unity or Godot are top recommendations for
beginners). Focus relentlessly on building the tiniest possible
playable rhythm prototype (one button, one note type, one beat).
Learn the engine fundamentals through that process. Resist the urge
to build your dream game immediately. Completing a tiny, functional
prototype is infinitely more valuable and motivating than getting
lost in the complexity of a "from-scratch" approach or an overly
ambitious first design. Good luck!
```

(Deepseek R-1, 2025)

DeepSeek gave us the most amount of information regarding the game engine and the rhythm game. It recommended Unity and Godot as the top 2 choices for deciding which game engine to use, however it also included Unreal Engine which is a common game engine that is used. Furthermore DeepSeek also gave us insightful tips as to how to proceed with the game design of the rhythm game itself. These tips include starting small and afterwards adding more features, following online tutorials for the game engine of choice, and also attempting prototyping with different aspects that will be implemented. DeepSeek was the most helpful in this case and had a positive outcome on how we continued with the project.

### 5.5 Conclusion

Using these Artificial Systems, it gave us a lot of information as to where we can start on the software side of creating the rhythm system. All 4 searches recommended that the game engine would be the most useful resource to use due to its friendliness to beginners, not only that but for a project of this magnitude it would be the more effective one to use.

Furthermore, Artificial Intelligence gave us some useful game engines to use. These game engines are ones we have heard of, with those being Godot, Unity, and Unreal Engine. ChatGPT specifically mentioned Godot and Unity with also adding benefits of what they have as features. For example, according to ChatGPT, Godot is lightweight, open-source, does not have any royalties, has very beginner-friendly scripting, and has good 2D support.

Similarly, both Microsoft Copilot and DeepSeek have a very similar response regarding using Godot as a game engine saying that Godot is lightweight and beginner-friendly and is open-source. Microsoft Copilot specifically mentions having a built in animation player and audio support. Deepseek gives us specific cons, some of which we don't need to acknowledge due to us not needing it, which is “3D capability is less mature than Unity”. This is irrelevant to us since we will be making a 2D game. However, having a small asset store causes us to need to hand make objects or find assets that we are able to use as long as we abide by their terms and services.

In the end, with the help of artificial intelligence, we had a start as to which game engine we could research to help us the best.

# 6. Hardware Design

## 6.1 Dance Pad Controller Board

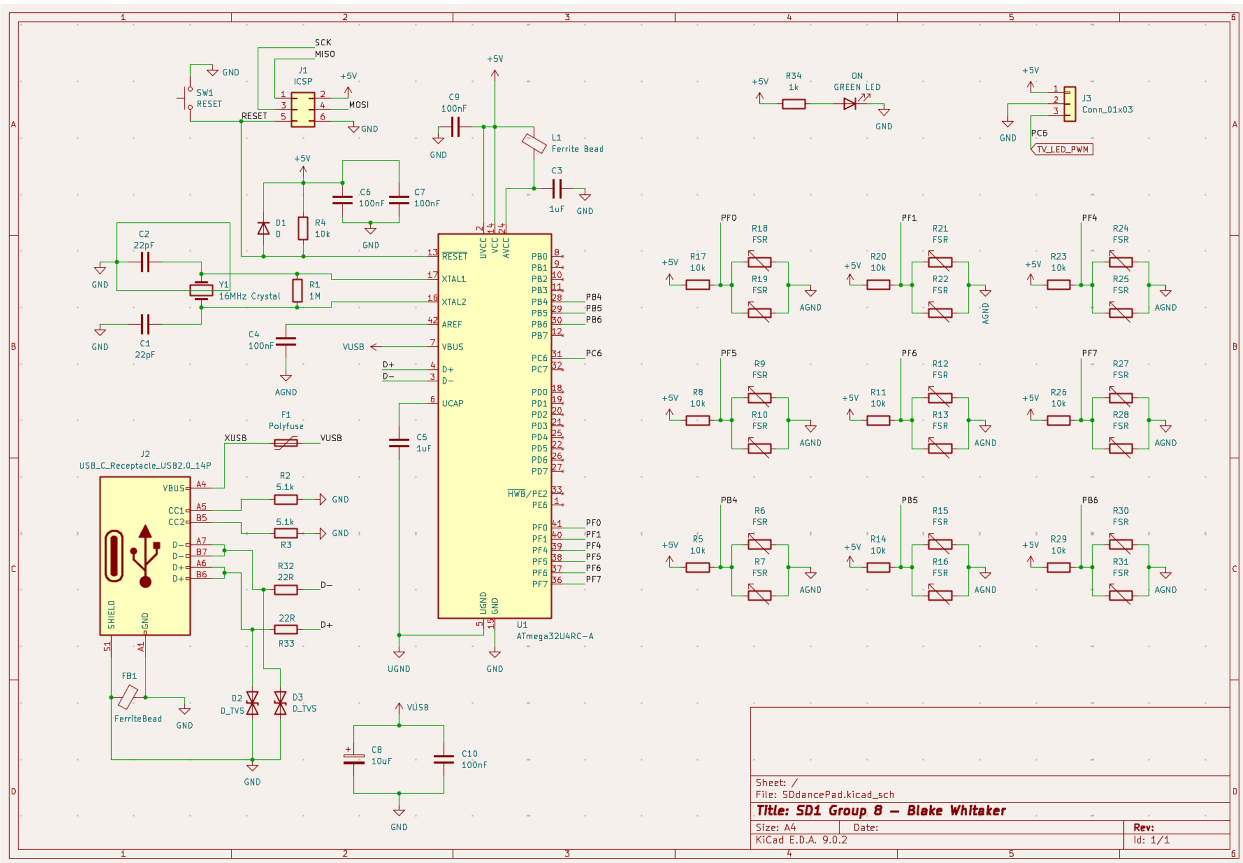



***Figure 6.1*** *Dance Pad Controller Board Schematic*

This board is the "brain" of the entire system. Its primary jobs are to read the inputs from the nine dance pad sensors, act as a USB game controller to communicate with the PC, and send control signals for the lighting.

<u>**Main Components and Functions:**</u>

**Microcontroller (U1 - ATmega32U4):** This is the central processor. We chose this specific chip because it has built-in USB capabilities, which allows it to be easily recognized by a computer as a game controller without needing extra components.

- **Clock (Y1):** The 16MHz crystal oscillator provides a stable and accurate clock signal, which is essential for reliable USB communication.
- **Reset (SW1):** The pushbutton allows for a manual reset of the microcontroller during development and testing.
- **Programming (J1 - ICSP):** This 6-pin header is the In-Circuit Serial Programming port. It is used one time to load the Arduino bootloader onto the chip, which allows us to program it easily via USB.

**USB Communication (J2):**

- A **USB-C Receptacle** provides the data connection to the PC.
- The connection is protected by a **500mA Polyfuse (F1)** to prevent the board from drawing too much current and damaging the computer's USB port.
- **ESD Diodes (D2, D3)** protect the sensitive D+ and D- data lines from static discharge.
- **5.1kΩ Resistors (R2, R3)** on the CC1 and CC2 pins are required by the USB-C standard to identify the board as a device to the host computer.

**Sensor Interface (FSR Circuits):**

- The board has nine identical sensor circuits, one for each dance tile.
- Each circuit is a **voltage divider**. For example, the first tile uses two off-board Force-Sensitive Resistors (FSRs), represented by R18 and R19, which are wired in parallel. They form a voltage divider with the on-board **10kΩ resistor (R17)**.
- When a player steps on a tile, the FSRs' resistance decreases, causing the voltage at the output to change.
- The output of each of the nine circuits is connected to a unique **Analog-to-Digital Converter (ADC)** pin on the microcontroller (e.g., PF0, PF1, etc.), which reads this voltage change.

**Power Filtering:**

- The board receives its 5V power through the 3-pin connector, **J3**.
- To ensure the microcontroller has clean, stable power, several filters are used. A **Ferrite Bead (L1)** and a **1μF capacitor (C3)** create a low-pass filter for the analog power supply (**AVCC**). This is critical for getting accurate readings from the FSR sensors.
- Several **100nF decoupling capacitors (C6, C7, etc.)** are placed near the chip's power pins to filter out high-frequency noise.

**Indicators and Control Output:**

- A green **"ON" LED** (connected via resistor R34) provides a simple visual confirmation that the board is powered on.
- The **3-pin connector (J3)** also sends a PWM (Pulse-Width Modulation) control signal from pin **PC6** to the Power Hub Board to control the brightness of the TV's IR illumination LEDs.

## 6.2 Power Hub Board

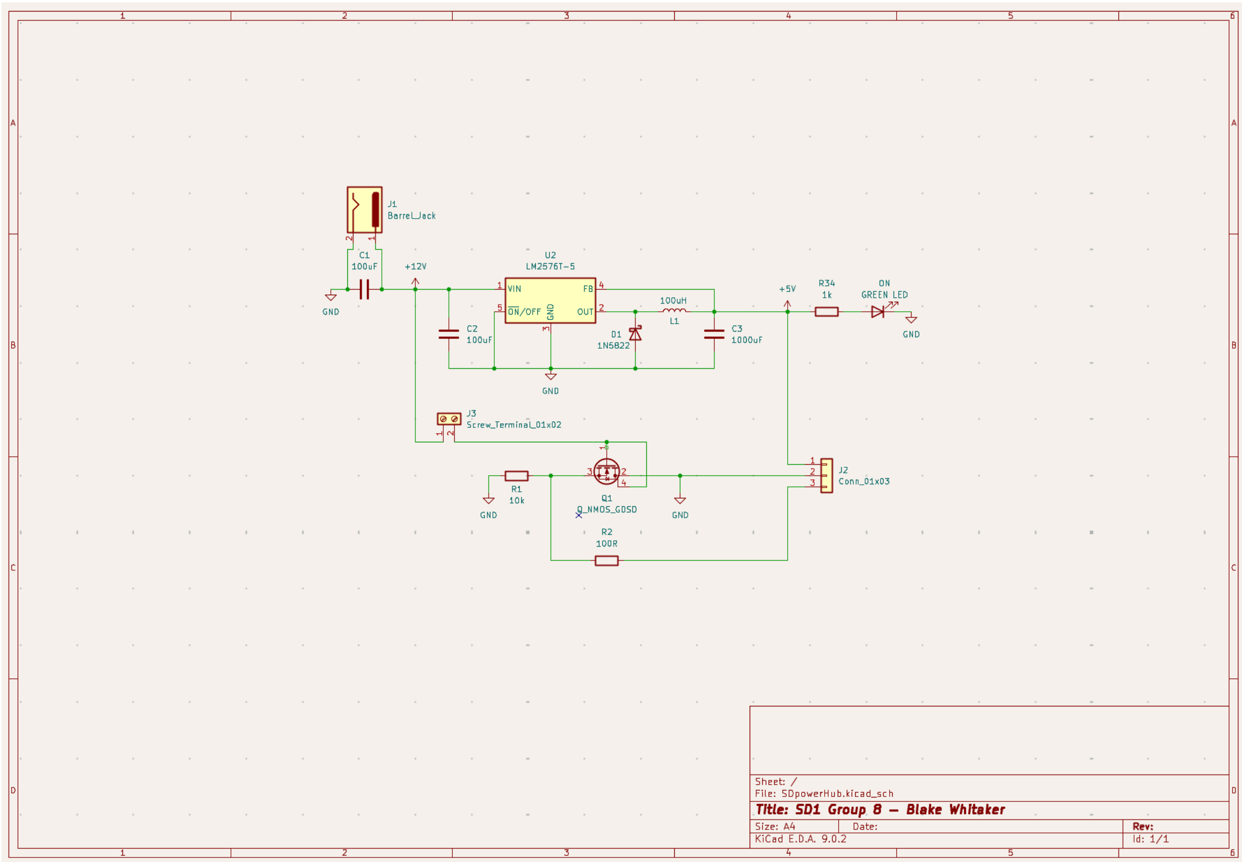



***Figure 6.2*** *Power Hub Board Schematic*

This board is a dedicated power supply and driver unit. Its purpose is to take the main 12V input and safely distribute power to the rest of the system, keeping the noisy, high-current components separate from the sensitive controller board.

**Main Power Input (J1):** A standard **DC Barrel Jack** receives 12V from an external AC-to-DC power brick. A **100µF capacitor (C1)** helps to smooth this incoming voltage.

**5V Power Supply (U2 - LM2576T-5):** This is a highly efficient **5V switching (buck) regulator**.

- It takes the 12V input and steps it down to a stable 5V output.
- The circuit includes a **100µH inductor (L1)**, a **Schottky diode (D1)**, and input/output capacitors (**C2, C3**) as required by the regulator's datasheet for stable operation.
- This 5V rail is used exclusively to power the Dance Pad Controller Board via the J2 connector.

**IR LED Driver (Q1 - N-Channel MOSFET):**

- This MOSFET acts as a high-speed electronic switch for the 12V IR LED strip, which connects via the **2-pin screw terminal (J3)**.
- The PWM signal from the controller board comes in through **Pin 3 of J2**.
- A **100Ω resistor (R2)** on the MOSFET's Gate protects the microcontroller pin, and a **10kΩ resistor (R1)** pulls the gate to ground to ensure the LEDs stay off when there is no signal.
- When the PWM signal is high, the MOSFET turns on, allowing current to flow through the LEDs to ground, turning them on. By pulsing this signal, we can control the brightness.

**Board-to-Board Link (J2):** This **3-pin JST connector** is the link to the main controller.

- It sends the **+5V** and **GND** from this board to power the controller.
- It receives the **TV_LED_PWM** signal to control the MOSFET.
- A green **"ON" LED** (connected via resistor R34) provides a simple visual confirmation that the board is powered on.

This two-board system creates a modular, robust, and electrically clean design that meets all the project requirements.

### 6.3 Optical Imaging System Design

The optical subsystem is designed to reliably capture and analyze a player's full-body movements in real time, enabling the system's pose-based "Style Score" feature. This subsystem integrates a global shutter camera, wide-angle lens, and an optimized near-infrared (NIR) LED illumination system to provide robust visual tracking across the entire play area.

The vision system must cover the 2.9m x 2.9m dance pad at a typical player distance of 1.83m, maintaining a horizontal field of view of approximately 93°.This ensures consistent landmark detection and minimizes distortion even at the edges of the field.

A time-multiplexed 850nm near-infrared LED system was selected to provide uniform, comfortable illumination that is invisible or only partially visible to the player but reliably detected by the camera. This design minimizes distractions while maintaining sufficient brightness for accurate MediaPipe landmark tracking.

In alignment with the project's broader objectives stated in Chapter 2, the optical subsystem emphasizes consistent player coverage, high image clarity, safe illumination, and efficient integration with the vision-based scoring system. The imaging system is designed to deliver high spatial resolution, wide field of view, and low-latency image capture that supports real-time pose estimation. After evaluating multiple options, the Arducam AR0234 camera module was selected. This camera was selected based on its technical specifications, including a 2.3MP resolution (1920 * 1200), 3μm pixel pitch, and frame rates up to 60 FPS over USB 3.0. The global shutter feature helps reduce motion artifacts during rapid gameplay.

Given the dance pad dimensions are 2.9m x 2.9m and the working distance is 1.83m, the optical system must maintain a horizontal field of view of approximately 93° to reliably capture the entire play area with minimal occlusions. Commercial lenses with longer focal lengths or narrow fields of view are impractical, since they would have required either excessive installation height or more aggressive mounting angles to fully capture the 2.9 m x 2.9m play area. To avoid these challenges, the system instead uses a commercially available short focal length M12 lens that is approximately 3mm. It has a diagonal FOV between 100-114 degrees. The key engineering requirements for the optical subsystem are summarized in Table 6.1.

***Table 6.1*** *Engineering requirements for the optical imaging subsystem.*

| Component | Parameter | Specification | Unit |
|---|---|---|---|
| Camera | a) Pixel Pitch<br>b) Resolution<br>c) Frame Rate | a) 3<br>b) 1920 x 1200<br>c) ≥ 60 | a) μm<br>b) pixels<br>c) FPS |
| Lens (M12 3mm) | a) Focal Length<br>b) Horizontal Field of View | a) 3<br>b) 93 | a)mm<br>b)degrees |
| Optical Subsystem | a) Brightness Uniformity<br>b) Resolution at 6ft | a) ≥85<br><br>b) ≥ 3 (resolving ~ 1mm features) | a)%<br><br>b) pixels/mm |

As summarized in Table 6.1, the selected camera module and lens combination achieves a pixel pitch of 3 μm with a resolution of 1920 x 1200 pixels, is sufficient to meet the target of ≥ 3 pixels/mm needed for ~1mm feature detection at a 1.83m distance. The approximate 93° horizontal field of view ensures the full dance pad remains visible, while the geometric distortion and brightness uniformity specifications help preserve consistent landmark tracking across the entire field of play.

### 6.4 Illumination System Design

Consistent and uniform illumination is essential. In a dim arcade environment or under changing lighting conditions, the system still needs to have consistent illumination of the player. Therefore, relying on ambient light alone is insufficient for consistent pose estimation. To account for this, an active illumination system was designed to provide uniform coverage across the entire 2.9m x 2.9m dance pad.

850nm near-infrared (NIR) LED strips were selected as the primary illumination source. Near-infrared wavelengths are partially invisible to the human eye, minimizing player distraction and discomfort while remaining fully detectable by the camera’s image sensor, which does not have an IR-cut filter. This approach allows for reliable landmark visibility for MediaPipe while not compromising on user comfort.

To further optimize power consumption and thermal performance, a time-multiplexing strategy is being used. In this scheme, the dance pad area will be divided into multiple LED illumination zones, with only one zone activated per camera frame. Cycling rapidly at the camera's frame rate of 120 FPS, resulting in a complete zone cycle every 33ms. Because this frequency exceeds the human flicker fusion threshold, players and the camera module will still perceive the illumination as continuous.

The selected LED strips are standard 12 V SMD-based products, offering good cost efficiency and a wide choice of densities (typically 60-120 LEDs/m). Each strip is rated at approximately 14.4 W/m, with power budgets calculated to keep the current draw under 2A per zone. Table 6.2 summarizes the engineering requirements for the illumination system.

***Table 6.2*** *Engineering requirements for the illumination subsystem*

| Component | Parameter | Specification | Unit |
|---|---|---|---|
| LED Illumination | a) Wavelength<br>b) Brightness Uniformity<br>c) Beam Angle<br>d) Zone Switch Time<br>e) Power per Zone | a) 850<br>b) ≥85<br>c) ≥ 120<br>d) ≤ 33<br>e) ≤ 18 | a) nm<br>b) %<br>c) degrees<br>d) ms<br>e) W |
| Illumination System | a) Coverage Area<br>b) Player Distance | a) 2.9 * 2.9<br>b) 1.83 | a) m<br>b) m |

As summarized in Table 6.2, the 850 nm LED strips are designed to deliver at least 85% uniform brightness across the entire field of view, with a wide beam angle of ≥ 120° to minimize shadows. Time-multiplexing with a zone switch time of ≤ 33 ms reduces the total system load while maintaining a seamless visual.

Safety and photobiological compliance were also considered. IEC 62471 guidelines were reviewed, and the selected 850 nm LED strips fall within Risk Group 1 at specified radiant exposure and player distance. Although typically arcade lighting is rarely completely dark, its variability, color saturation, and moving shadows make it unreliable for consistent pose detection. Therefore, the OR illumination system is necessary to establish a stable, uniform baseline at the player's distance of 1.83m. Providing radiant intensity equivalent to at least 300 lux of visible illumination for consistent landmark detection. Since the dance pad is intended to be portable for use in home settings, the lighting conditions there could be even more unpredictable, ranging from bright windows to dim living rooms. Including a controlled illumination system ensures consistent MediaPipe performance regardless of whether the system is deployed in a commercial arcade, dorm room, or any other household living space.

The camera module will be mounted at approximately 1.5m height with a downward angle of about 15-20° to capture the full play area while minimizing occlusions from player movement. The LED strips are currently planned to be mounted around the monitor frame to provide even IR coverage while avoiding direct glare into the player's eyes. However, other LED mounting configurations will also be tested in SD2 to determine the most effective combination of coverage uniformity, ease of integration, and player comfort. The final arrangement will be chosen based on uniformity measurements and practical field trails to ensure consistent landmark detection performance.

### 6.5 Optical System Testing and Verification

The optical and illumination subsystems will be tested to confirm they meet the design requirements described above. Testing will focus on verifying the horizontal field of view covers the entire dance pad area, confirming the spatial resolution supports ~ 1 mm feature detection, measuring illumination uniformity across the player area, ensuring time- multiplexing achieves seamless coverage without perceptible flicker, and confirming the system achieves at least 300 lux equivalent at the player's distance.

Initial testing will involve capturing images of a printed calibration checkerboard to measure geometric accuracy, including field of view, distortion, and alignment[8]. Separately, a uniform reflectivity target such as a matte white board will be used to verify even illumination across the play area. Pixel intensity measurements from these images will be analyzed to assess both brightness uniformity and overall image quality. If any issues such as uneven lighting, distortion, or poor focus are identified, adjustments will be made to the lens focus, LED placement, or camera alignment prior to full integration.

## 7. Software Design

### 7.1 Input/Output

The primary method of input for the rhythm game is the custom-built dance pad, which functions similarly to a keyboard from the perspective of the operating system. During gameplay, the game continuously polls for input every frame, much like a traditional PC game responds to keyboard input. Our custom PCB includes a microcontroller (MCU) responsible for interpreting signals from force sensing resistors (FSRs) embedded in each panel of the dance pad. These signals are converted into digital keypresses and transmitted to the host computer over a USB connection. As far as the PC is concerned, the dance pad appears as a standard USB keyboard, with each directional panel mapped to a specific keycode.

This input abstraction allows for flexibility during development and testing. Although the dance pad is the intended method of interaction, players could also use a conventional keyboard to play the game. However, this alternative would lack the physicality and immersive experience that the dance pad is designed to provide.

The game maintains an internal representation of the current input state, tracking which directional inputs are actively being held and recording the precise timing of each press event. When a player steps on a panel corresponding to a direction (such as up, down, left, right, center, or any of the diagonals), the game updates its internal state to reflect that the direction is currently active. It also records the timestamp or frame number at which the input was first detected. This information is used for evaluating the accuracy of the player's timing during gameplay.

When the player lifts their foot and the input is released, the game updates the state to mark that direction as no longer active. This real-time management of directional states and associated timing data ensures that the game can accurately judge the player's inputs against the expected cues defined in the song chart. A diagram of this input and output flow is shown in Figure 7.1.

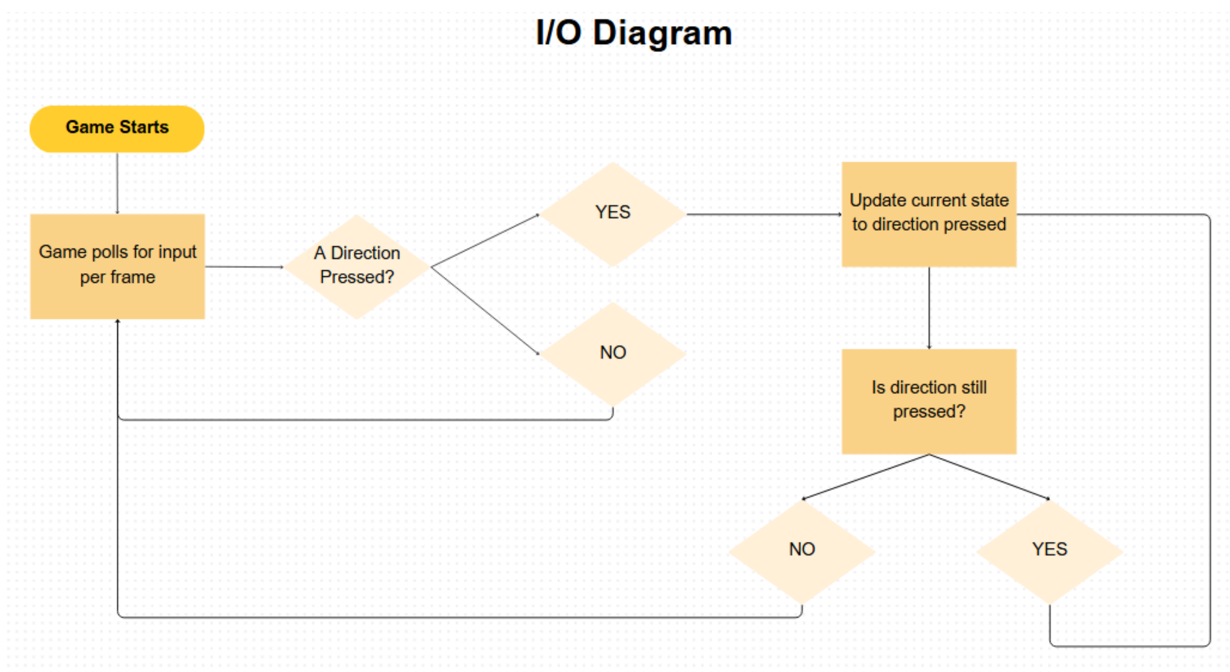



***Figure 7.1** Input/Output Flow Diagram*

### 7.2 State

The concept of state plays a central role in the software's logic. The input state is composed of several components, with each of the nine directional inputs represented as individual objects. These objects include a boolean indicating whether the input is currently held down, and if so, they also store the time or frame at which the press began. This enables precise tracking of how long a panel has been held and when the input occurred relative to the rhythm chart.

In addition to directional input, the system also tracks the player's current pose using the output from the computer vision subsystem. The pose state is updated every frame based on the latest keypoint and gesture analysis results. Each pose is identified by a label (such as “hands up” or “lean left”) and is associated with a confidence score and the time at which it was first detected. The pose state behaves similarly to directional input, allowing the game to evaluate whether the player is performing the correct pose at a given moment in the song.

The system also maintains a variable that stores the most recently changed input or pose, useful for debugging, scoring, and triggering visual effects. At the start of each song, the input state is initialized to a neutral condition, with no active inputs or poses. As gameplay progresses, the state evolves to reflect the player’s ongoing actions. The flow of this state update process is illustrated in Figure 7.2, which outlines how the game responds to input and pose events.

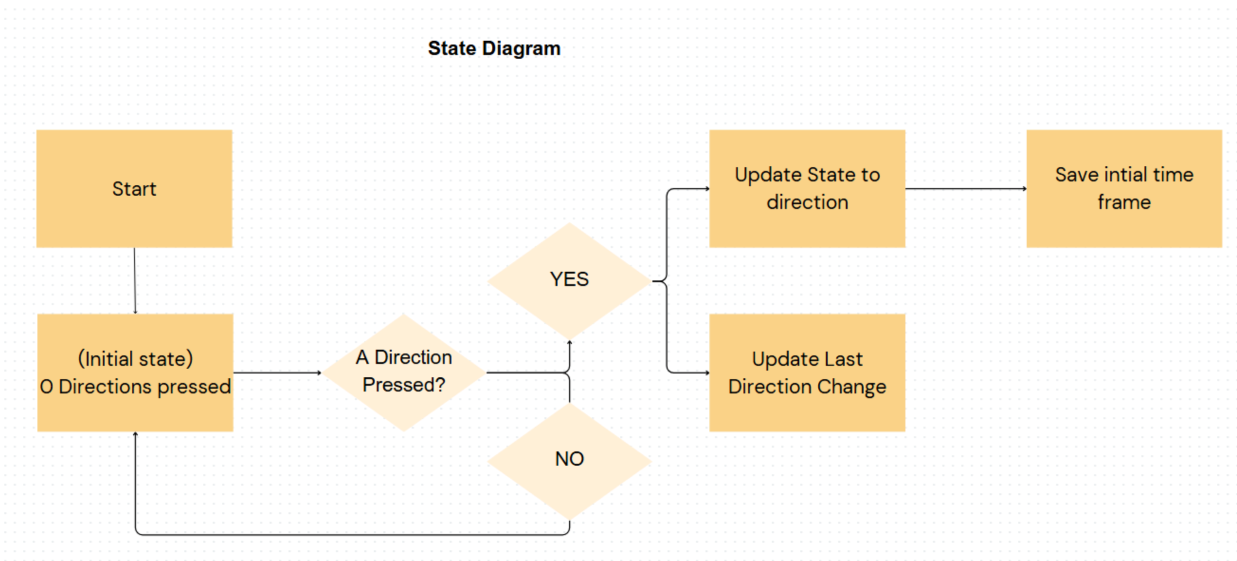



***Figure 7.2*** *Note State Flow Diagram*

### 7.3 Event handling

Scoring in the rhythm game is driven by discrete gameplay events that are defined in the song chart. These events fall into two main categories: directional note events and pose-based gesture events. When the game encounters one of these events during playback, it checks the current input state to determine whether the player has correctly performed the expected action.

Each event is associated with a specific timestamp or frame number. At the moment an event is triggered, the game compares the expected input or pose to the actual state of the player, as recorded in the current frame and recent frame history. A timing window is applied to judge how closely the player's action aligns with the event.

The judgment criteria follow a frame-based window around the expected frame of the event:

- If the input or pose is detected within ±3 frames, the judgment is marked as Perfect.
- If detected within ±5 frames, it is marked as Great.
- If within ±8 frames, it is considered Okay.
- If the correct input or pose is not detected within 8 frames of the event, it is scored as a Miss.

This scoring logic is applied consistently to both directional and pose events. However, additional tolerance may be applied to pose events to account for the inherent variability of real-time pose detection. Factors such as model inference delay, confidence thresholds, and temporal smoothing can introduce minor inconsistencies, so the system may offer a slightly wider scoring window for gestures without compromising the integrity of the gameplay.

By evaluating input and pose accuracy within tightly defined timing windows, the game can provide responsive feedback and fair scoring that rewards player precision while accommodating the occasional noise present in computer vision systems.

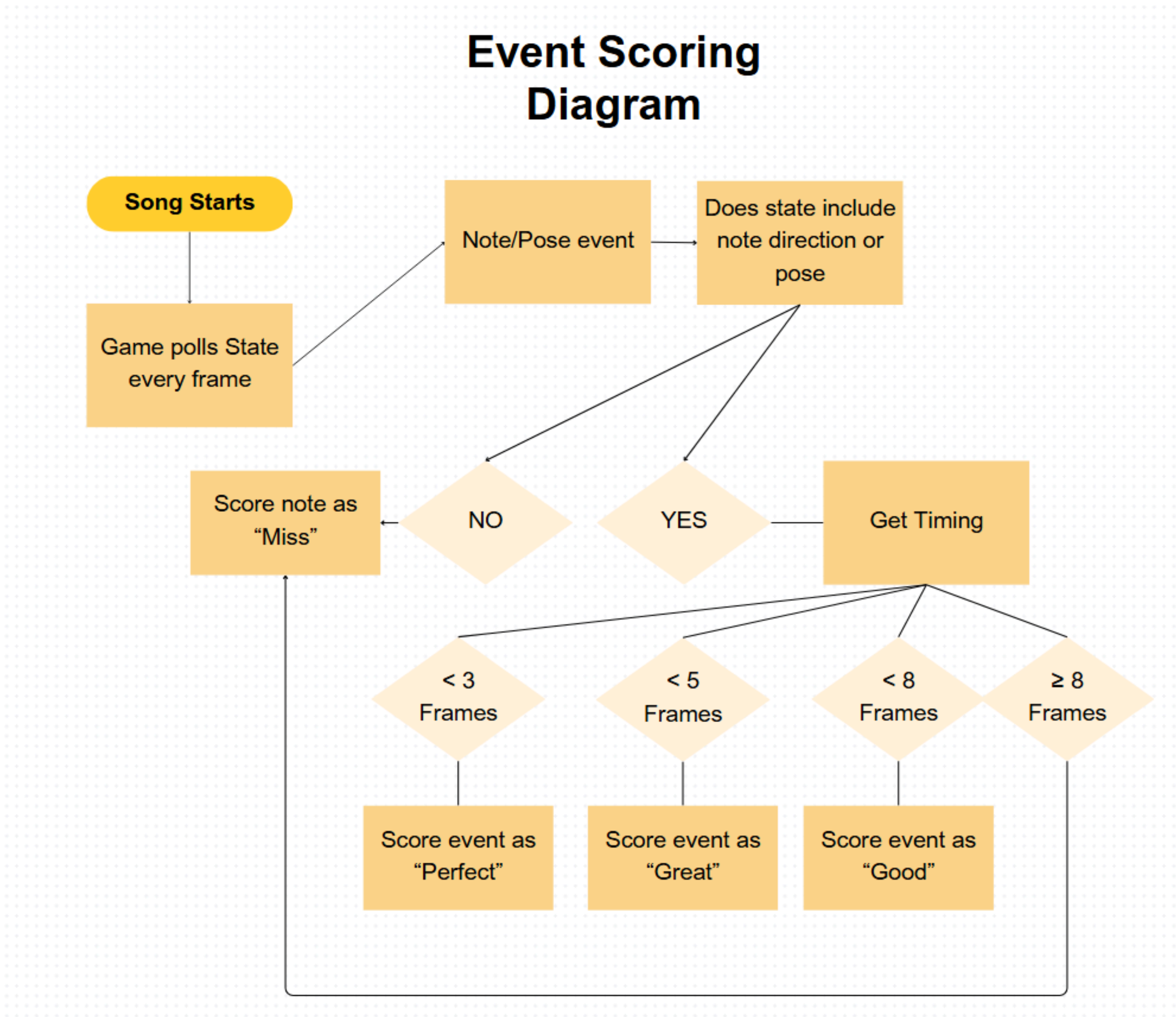



***Figure 7.3*** *Event Scoring Flowchart*

### 7.4 Pose Handling

Using the OpenCV and Mediapipe software, we first obtain the image sent by the camera module that will show the person. Afterwards our machine learning model activates that is what allows the person to be mapped using a series of dots as seen in ***Figure 3.10***. We will then retrieve the series of dots location and with this information we can determine if a certain pose is being performed. If a certain pose is being performed and recognized, then the state will be updated to reflect that. However, if a pose is not being performed the state will be updated to show that there is no pose that is being performed.

Using this system there is a possibility due to artificial intelligence, that the state could be incorrectly updated, however with our system there is a 90% accuracy in correctly determining the correct state when performing these series of poses at certain portions of the game's design.

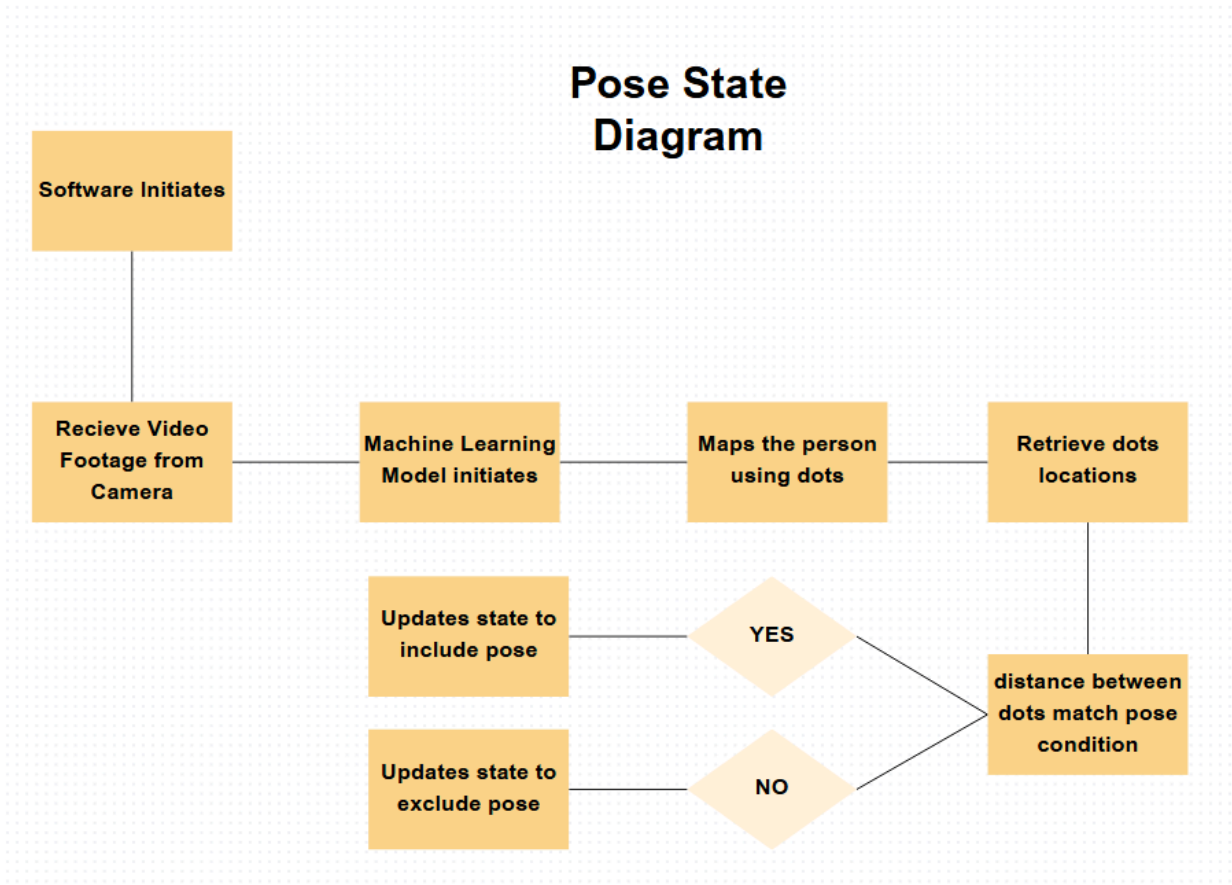



***Figure 7.4*** *Pose State Flow Diagram*

## 8. System Fabrication/Prototype Construction


***Figure 8.1*** *Top view: 10x10 inches central acrylic tile, 12x12 w/ border, secured on wood base w/ standoff blocks*

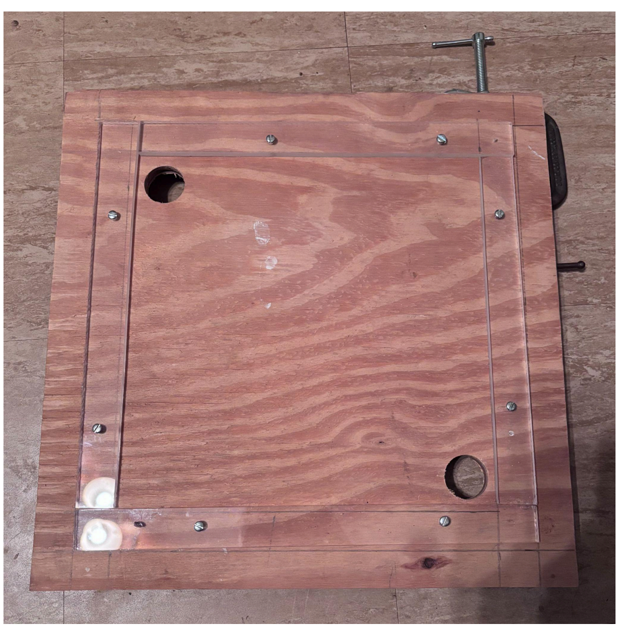


***Figure 8.2****Top view w/ central tile removed showing electrical routing holes*

***Figure 8.3*** *Bottom view: standoff blocks to allow for easy cable routing*

***Figure 8.4*** *Top View: w/ Force Sensitive Resistors (FSR) placed under acrylic tile.*

# 9. Optoelectronics Feasibility Study and Testing

## 9.1 Feasibility Study

The primary aim  of the S.T.E.P system is to function efficiently as a rhythm game playable in both home and arcade environments, with a cost and power-efficient architecture. A critical aspect of this is the optical and illumination subsystems. When planning the lens, it was determined that sourcing an off-the-shelf aspherical M12 lens was the most practical and cost-efficient strategy, rather than developing a custom-manufactured optic. A custom aspherical lens would have been expensive and time-consuming to prototype, while commercial M12 aspherical options still met the project's required field of view, resolution, and IR transmission for MediaPipe performance.

Cost efficiency also drove the camera selection. Initially, higher priced Arducam modules were considered, but research found the SVPRO AR0234-based global shutter camera to provide comparable performance for a lower price. This module supports M12 lenses, has no IR-cut filter, and maintains the necessary 3μm pixel pitch and 60 FPS frame rate for reliable gameplay tracking.

For the illumination system, a key design consideration was ensuring consistent player coverage while avoiding excessive instantaneous current demands. Arcade environments can have variable or dim ambient lighting, making active illumination important for tracking. Initially, larger LED panels inspired by other rhythm games (like Dance Around) were explored. However, powering all zones simultaneously would have increased the  peak current demands  to around 69 W,which would place higher demands on the power supply and wiring. By instead lighting only one zone at a time, the peak current was reduced to around 18 W, supporting simpler and more efficient system integration.

Additional layout options were explored, including ring lights, edge-mounted LED bars, and corner-mounted strips. These were evaluated for feasibility, wiring complexity, and potential glare or reflections. Ultimately, monitor-mounted LED arrays offered the best compromise of player coverage, mechanical simplicity, and consistent geometry relative to the camera, while supporting portability.

### 9.1.2 Design Requirements and Specifications

The optical and illumination subsystems must together ensure accurate, consistent pose tracking under diverse conditions. For the optical design, the target was to achieve at least 3 pixels per millimeter resolution, a frame rate of 60 FPS minimum, and a horizontal field of view near 77-93 to reliably capture the entire 2.9 m x 2.9 m dance pad area from a 1.83 m distance.  The lens was expected to maintain minimal geometric distortion, with sufficient sharpness to resolve ~1mm limb features. For the illumination system, the design aimed for at least 85% brightness uniformity, 850 nm near-IR wavelength for user comfort, and an equivalent brightness of 300 lux at the player distance, while managing power with a time-multiplexed scheme to limit peak current draw. These requirements guided component selection, mechanical layout, and optical design.

### 9.2 Optical System Design

The purpose of the vision subsystem is to capture spatially accurate, high-contrast images of the player's movements during gameplay, supplying reliable data to the MediaPipe pose estimation framework for style-based scoring. The optical design is critical for minimizing ambiguity in landmark detection, reducing the computational burden on the pose estimation model, and improving real-time responsiveness.

The optical system targets a minimum of 3 pixels per millimeter to resolve features approximately 1 mm in size on the dance pad. With a pixel pitch of 3.0μm, the required image-space resolution is:

3 x 3μm = 9.0 μm
The system magnification M is then:

$$M = \frac{Image\ size}{object\ size} = \frac{9.0\mu m}{1mm} = \frac{9.0}{1000} = 0.009$$

Given the working distance of approximately 1830 mm to the player:

$s_1 = 1830\ mm$

$s_2 = M * s_1 = 0.009 * 1830\ mm = 16.47\ mm$

Using the thin lens equation:

$$f = \left(\frac{1}{s1} + \frac{1}{s2}\right)^{-1} = \left(\frac{1}{1830\ mm} + \frac{1}{16.47\ mm}\right)^{-1} = 16.32mm$$

While this effective focal length is theoretically moderate, the system must cover a very large field of view from a 1.83m distance. That corresponds to a required horizontal field of view:

$$\theta_H = 2 * arctan\left(\frac{2896/2}{1830}\right) = 77.3°$$

This confirms a wide-angle lens with a focal length closer to 3 mm is needed, consistent with the prototype lens choice (CIL329). That lean, with a diagonal FOV near 120° (translating to roughly 90-100° horizontal on the AR0234 sensor), ensures the system can fully view the dance pad area without demanding excessive installation height or extreme tilt angles.

### 9.3 Optical Illumination System Testing and Verification

The optical and illumination system will be carefully tested to ensure they meet these engineering requirements before integration into the full S.T.E.P cabinet. Testing will focus on verifying that the field of view fully covers the 2.9 m x 2.9 m dance pad, confirming spatial resolution supports ~ 1mm feature detection, measuring illumination uniformity across the player’s body, ensuring time-multiplexed zones maintain seamless perceived lighting without flicker, and verifying that a minimum of 300 lux equivalent illumination is achieved at a player distance of 1.83 m.

Geometric and image quality verification will be carried out using a printed calibration checkerboard pattern to check the camera’s coverage, measure geometric distortion, and confirm the intended horizontal field of view. Separately, a uniform reflectivity target will be used to verify even illumination across the dance pad area. Pixel intensity measurements will then be analyzed frame-by-frame to confirm both brightness uniformity and signal-to-noise performance, ensuring no major shadows or hotspots occur during typical player movements.

# 10. Administrative Content

## 10.1 Budget

We are aiming to limit the budget of this project to $500. *Table 10.2* as shown below lists the bill of materials. Although we want to minimize costs, we also want to ensure we have materials that have a good enough quality so that they are reliable and efficient. This includes the framework (hard materials), PCB, RGB LEDs, LED ring, camera module, and force-sensing resistors. If any of these were bought very cheap without taking into account better affordable options, we would face consequences.

**10.2 Bill of Materials**

***Table 10.1*** *Itemized Bill of Materials*

| Item | Dimensions | Estimated Unit Cost | Quantity | Estimated Total Cost |
|---|---|---|---|---|
| SVP AR0234 Sensor | 38mm x 38mm | $65.99 | 1 | $65.99 |
| Commonlands CIL329 lens | 20mm x 14mm | $39.00 | 1 | $39.00 |
| DC12/24V 5050 SMD | 5m x 0.01m | $28.99 | 1 | $28.99 |
| PCB | undetermined | $50 | 1 | $50 |
| Force-Sensing Resistors | 12.7mm x 57mm | $5 | 18 | $90 |
| Plywood | 36 `` x 38 x ¼ `` | $50 | 1 | $50 |
| Aluminum Square tubing | 38`` x 38`` x 1/8`` | $10 | 6 | $60 |
| Polycarbonate Sheets | 11.75 ``x 11.75`` x 0.25 `` | $0 | 9 | $0(already have) |
| Non-Slip Rubber Mat | 2ft x 4ft x 3mm | $20 | 1 | $20 |

### 10.3 Distribution of Worktable

***Table 10.2*** *Distribution of responsibilities of each member for this project*

| Name | Major | Responsibilities |
|---|---|---|
| Andres Abrams | Computer Engineer | Software Assistant |
| | | Software Design and Implementation |
| | | Game Design |

| Name | Major | Responsibilities |
|---|---|---|
| Blake Whitaker | Electrical Engineer | Hardware Lead |
| | | PSU Design and Implementation |
| | | PCB Design |

| Name | Major | Responsibilities |
|---|---|---|
| Christopher Solanila | Computer Engineer | Project/Software Lead |
| | | Website Design and Management |
| | | Software Design and Implementation |
| | | Game Design |
| | | Computer Vision Implementation |

| Name | Major | Responsibilities |
|---|---|---|
| Jani Jon Lumibao | Computer Engineer | Hardware Assistant |
| | | MCU Selection and Implementation |
| | | Embedded Programming |

| Name | Major | Responsibilities |
|---|---|---|
| Kaila Peeples | Photonics Engineer | Lens Design and Simulation |
| | | Camera Module Integration |
| | | Optical and Illumination System Optimization |
| | | Image Quality Calibration |
| | | Illumination planning and testing |

### 10.4 Project Milestones for SD1 and SD2

#### 10.4.1 Project Milestones for SD1

***Table 10.3*** *Project Milestones SD1*

| Due Week | Advancement |
|---|---|
| 1 | Group Creation and have base idea |
| 2 | Researching and Innovating based off idea |
| 3 | Divide and Conquer Document completed, Have at least 1 committee member |
| 4 | Meet for revisions (if any) of Divide and Conquer Document with committee members, upload revised document into group website |
| 4-5 | Individual Research |
| 6 | 40 pages finished |
| 7 | Meet for revisions (if any) with committee members |
| 7-8 | Testing of components |
| 9 | 100 pages finished turn in Midterm Report |
| 10 | Meet for revisions (if any) with committee members |
| 10-11 | Start on video, finishing touches to 120 page document |
| 12 | 150 page document finished, mini video finalized |

### 10.4.2 Project Milestones for SD2

***Table 10.4*** *Project Milestones for SD2*

| Anticipated Start Week | Advancement |
|---|---|
| 1 | Camera Completion |
| 1 | PAD completion |
| 1 | Game Completion |
| 4 | Pad and Game integration |
| 4 | Camera detection |
| 10 | Camera integration |
| 16 | Final Day and Live Demo |

**Appendices**

[1]Guerra-Filho, G. B. (2005). Optical Motion Capture: Theory and Implementation. *Revista de Informática Teórica e Aplicada (RITA)*, 12(2), 1–18.

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=07147486b65d12c4326ccb3ad54ca612b52e1ac3

[2] *Dance around information*. RemyWiki. (n.d.). https://remywiki.com/DANCE_aROUND_Information

[3] *Dance around (AC) - bemani games - music game forums*. ZIv. (n.d.). https://zenius-i-vanisher.com/v5.2/thread?threadid=11041&page=2#:~:text=The%20games%20body%20tracking%20is,games%20that%20cater%20to%20everyone.

[4] *Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields IEEE TPAMI, 2019. https://doi.org/10.1109/TPAMI.2019.292925*

[5] *Google MediaPipe. (2024). Pose Estimation. Retrieved from = https://developers.google.com/mediapipe/solutions/vision/pose*

[6] *Step Revolution. (2024). StepManiaX. Kyle Ward Retrieved from https://stepmaniax.com*

[7] Sciotex. (n.d.). *Light colors used in machine vision*. Retrieved from https://sciotex.com/light-colors-used-in-machine-vision/

[8] Ultralytics. (2025). *A guide to camera calibration for computer vision in 2025*. Retrieved from https://www.ultralytics.com/blog/a-guide-to-camera-calibration-for-computer-vision-in-2025#:~:text=Oftentimes%2C%20they%20end%20up%20capturing,for%20real%2Dworld%20AI%20applications

[9] Flexfire LEDs. (n.d.). *Color rendering index (CRI) and LED lighting: What is CRI?* Retrieved from https://www.flexfireleds.com/color-rendering-index-cri-and-led-lighting-what-is-cri/#:~:text=CRIs%20under%2080%20are%20generally,one%20measurement%20for%20light%20quality

[10] MassedCompute. (n.d.). *How does the size of an image affect the training time of a CNN?* Retrieved from https://massedcompute.com/faq-answers/?question=How%20does%20the%20size%20of%20an%20image%20affect%20the%20training%20time%20of%20a%20CNN

[11] Supertek Module. (n.d.). *Monochrome camera vs. color: Which is best?* Retrieved from

https://www.supertekmodule.com/monochrome-camera-vs-color/#:~:text=Monochrome%20cameras%20are%20often%20used,color%20cameras%20are%20better%20suited

[12] Commonlands. (n.d.). *Wide angle 3mm M12 lens CIL329*. Commonlands. Retrieved July 10, 2025, from https://commonlands.com/products/wide-angle-3mm-m12-lens-cil329

[13] International Organization for Standardization. (2010). *ISO 9241-210: Ergonomics of human-system interaction*. Retrieved from https://cdn.standards.iteh.ai/samples/52075/c30c5ea5097843ecb89a9d417f9cdab1/ISO-9241-210-2010.pdf

[14] Smart Vision Lights. (n.d.). *IEC/EN 62471 summary: Photobiological safety of lamps and lamp systems*. Retrieved from https://smartvisionlights.com/wp-content/uploads/IEC_62471_summary.pdf

[15] IEC. (2006). *IEC 60598-1: Luminaires - Part 1: General requirements and tests*. Retrieved from https://www.lisungroup.com/wp-content/uploads/2019/07/IEC60598-1-2003-A1-2006-Standard-Free-Download.pdf

[16] ] Sobel, I., & Feldman, G. (1968). *An Isotropic 3×3 Image Gradient Operator*. Introduced the Sobel operator for edge detection. Retrieved from https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator

[17] Canny, J. F. (1986). *A Computational Approach to Edge Detection*. Proposed the multi-stage Canny detector with Gaussian, non-max suppression, and hysteresis. Retrieved from https://www.researchgate.net/publication/224377985_A_Computational_Approach_To_Edge_Detection

[18] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks* (AlexNet) Retrieved from https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

[19] Hidalgo, G., et al. (2019). *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields* Retrieved from https://www.ri.cmu.edu/publications/openpose-whole-body-pose-estimation/

[20] Google Research. (2020). *BlazePose: On-device, Real-time Body Pose Tracking Retrieved from https://research.google/blog/on-device-real-time-body-pose-tracking-with-mediapipe-blazepose/*

[21] Google AI. *MediaPipe Pose Landmarker* lightweight, real‑time 33‑landmark body pose detection. Retrieved from https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker

*[22] Canva. (2025). Canva [Web application]. Retrieved from https://www.canva.com/*

[23] OpenAI. (2023). ChatGPT (Feb 13 version) [Large language model]. https://chat.openai.com

*[24] Microsoft. (2025). Copilot [AI assistant]. Microsoft Corporation. https://www.microsoft.com/copilot*

*[25] Gemini. (2025, July 6). [Response to a query about rhythm game development] [Large language model]. Google. https://gemini.google.com*

*[26] DeepSeek (2024), [large language model developed by DeepSeek AI]. https://chat.deepseek.com/*

*[27] Unity Technologies. (2025). Unity (6000.0.52f1 LTS) [Computer software]. Retrieved from https://unity.com/download*

*[28] Godot Engine. (2025). Godot Engine (4.4.1) [Computer software]. Retrieved from https://godotengine.org/*

[29] Image Engineering. (2015). *Camera tests: White Paper 1.0*. Kerpen, Germany: Image Engineering GmbH & Co. Retrieved from https://www.image-engineering.de/content/library/white_paper/cameratests_whitepaper_1.0.pdf