Use Case

Our aim is to create a boxing game played using computer vision to detect human movements as a control scheme. Handheld controllers and VR headsets are often bulky and unwieldy, so we aim to incorporate computer vision to create a less restrictive and more immersive gaming experience, while providing exercise for users. Our target audience will be gamers and people who enjoy boxing/exercise.

3 components:
Computer Vision Pipeline (Mediapipe/Python/Unity scripts), Taiming
Video Game (Unity), Shithe
Haptic Feedback Device (Arduino/PCB wristband), Eric

Quantitative Design Requirements

Hardware

- Overall design must not be too large/heavy to not impede movement (<100g)
- Haptic feedback produced must be noticeable and have different, clearly distinguishable levels
- Battery must sustain arduino/motor setup for reasonable amount of time (>20 min)
- Wireless communication within reasonable distance (3 meters)

Computer Vision Pipeline

- Low latency(<=50 ms response time between user movement and movement being displayed on screen)
- High accuracy(80% accuracy in matching the real-world gestures with Unity avatars over 5 frames.)
- Good stability(smooth real-time gesture mapping with minimal fluctuation)

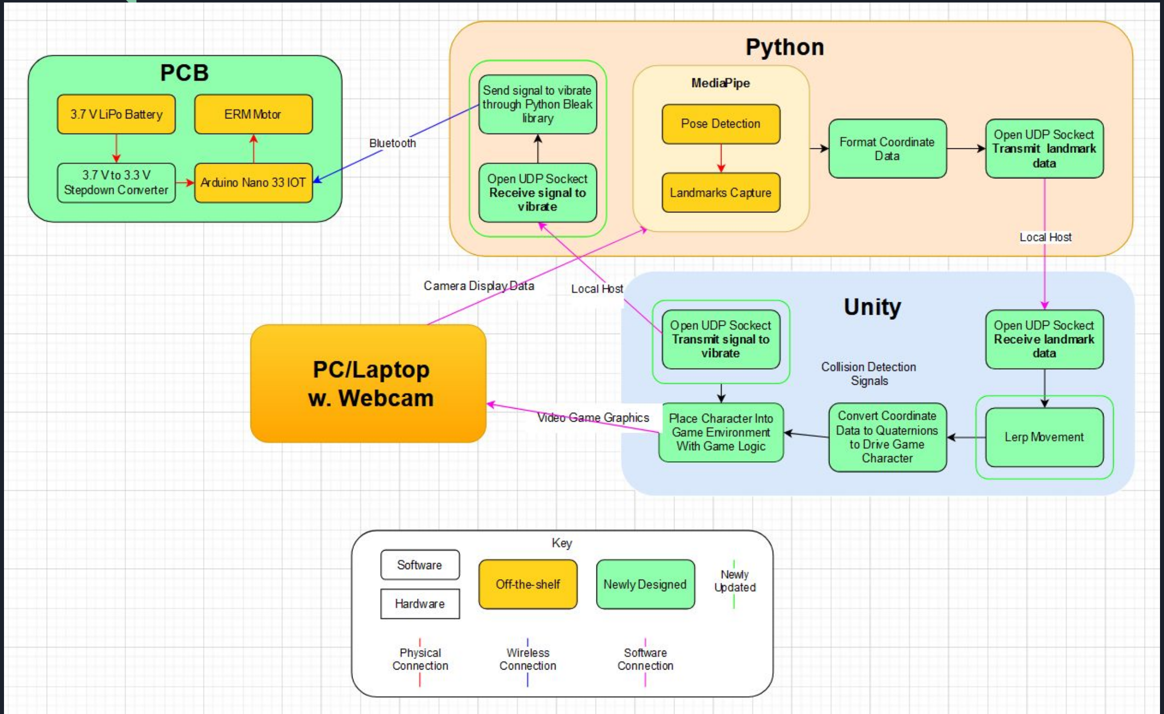
Video Game

- High-frame rate, >= 40 FPS on average, no sudden drops in frame rate
- Low Latency, <= 50 ms between human movement and movement displayed on screen
- Loading time of at most 45 seconds for any required load screens

Overall

- Low latency for entire system (<= 100 ms between human movement and haptic feedback)

Solution Approach

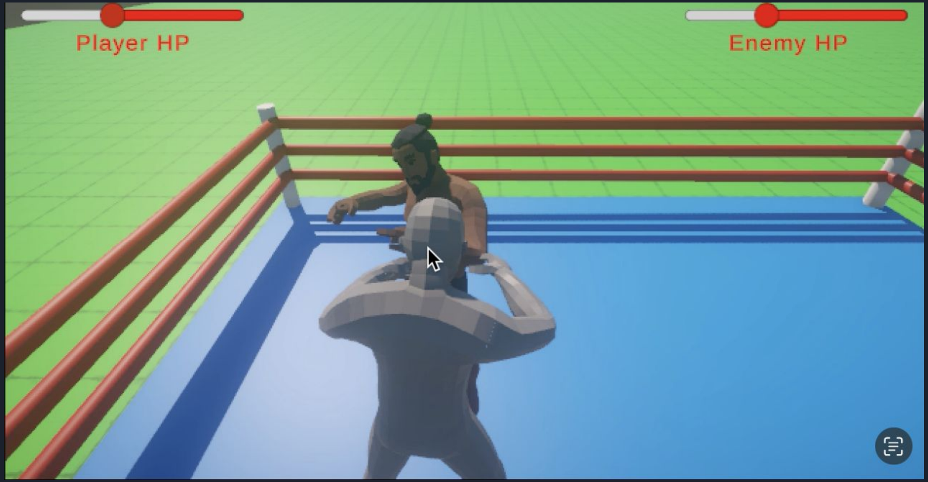


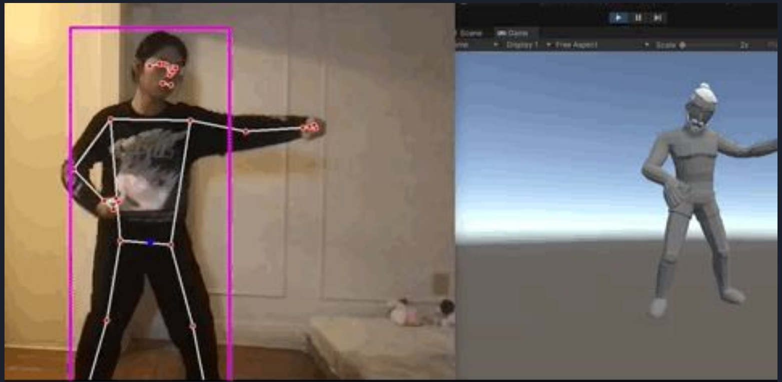
Important Update:

Quaternion Conversion replaced the naive 3D Coordinate Mapping to ensure smooth and accurate mapping free of distortions

Complete Solution

- Video game demo where user is able to move around in environment and throw punches, while receiving haptic feedback
- “Fight” against enemy AI, hitting each other until either player wins or loses, depending on whose health goes to zero first





Test, Verification, Validation - (Watch)

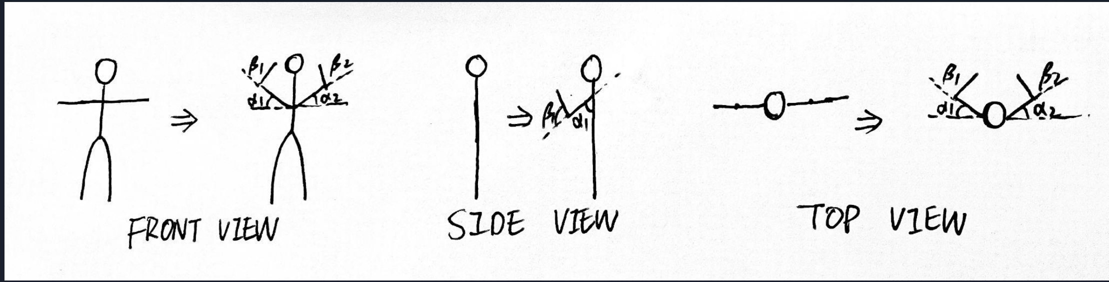
- **Weight Test -** Weigh watch to determine weight, with and without battery
    - Target: 100 g, on par with average watch
- **Vibration Test -** test vibration capabilities, determine if noticeable and if distinct levels can be felt
    - Target: 2 distinct levels of vibration, for player punch connecting and enemy punch connecting
- **Power Test** - Measure power consumption of watch in two ways, while idle and while vibrating to determine minimum and maximum power consumption
    - Target: 1.2 A, this would provide us with 20 minutes of continuous operation for a 400 mAh battery
- **Distance Test** - Measure maximum connectivity distance of bluetooth
    - Target: 3 meters, which is the amount of space we aim to use for our game
- **Ping Test -** Measure delay between bluetooth signal being sent by Python and after received by Arduino, using Python time library
    - Target: 30 ms

Test, Verification, Validation - (Watch) Results

Test	Target	Results
Weight	100 g	53 g
Vibration	2 distinct levels	2 distinct levels
Power	1.2 A	Min: 50 mA Max: 77 mA
Distance	3 m	20+ m
Ping	30 ms	22 ms

Test, Verification, Validation - (Computer Vision)

- **Accuracy Test**
  - Target Accuracy is 80%, which means in five consecutive n*20ms timestamps, 80 % of the ground truth-to-game avatar mappings should have less than 10 degrees difference in each joint angles.
  - Ground truth data are obtained by measuring arm rotation angles from three views (front, side, top).
  - Chosen over 3D inference due to hardware limitations (The latter requires a depth camera.)



The views above cover 4 D.O.Fs of the human arm and 2 perspectives.

Test, Verification, Validation - (Computer Vision)

- **Accuracy Test**
  - Game Avatar data are obtained by projecting 3D transformations to x, z, and x plane (corresponding to each view) and extraction rotational values.
  - The test is repeated five times for User 1 (height: 1.73m (5'8''); arm length: 1.74m) and User 2(height: 1.62m (5'4''); arm length: 1.65m)

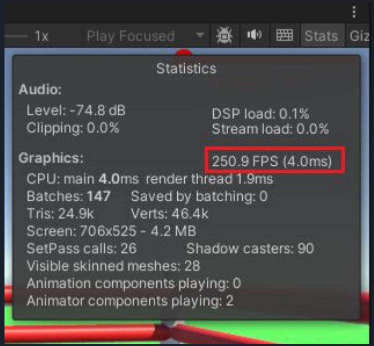
	Front View	Side View	Top View
α1	92%	86%	73%
α2	93%	N/A	76%
β1	89%	83%	67%
β2	91%	N/A	65%
Total	92.5%	84.5%	70.25%

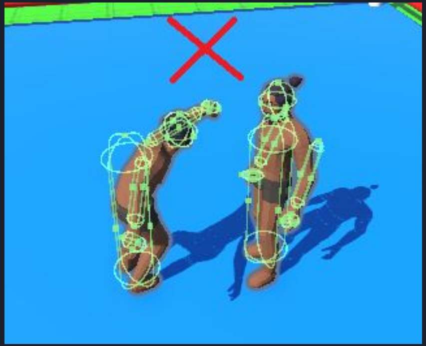
Mapping Accuracy Result (Target: 80%)

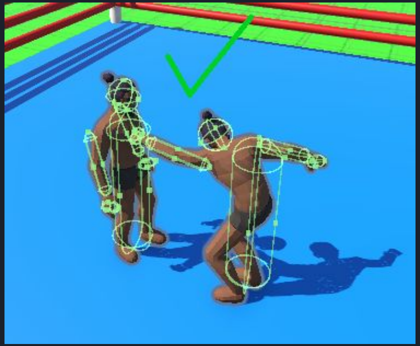
Only **Top View** underperforms due to MediaPipe's limitation on depth estimation

Test, Verification, Validation - (Video Game)

1. Passing = Higher than 40 FPS average with no sudden frame drops far below that target. Tested FPS through the use of Unity's built in statistics.
   Result: Passed with min average FPS of 50 on lower end laptop, Max 250 on desktop
2. Passing = Low Latency with <= 50 ms between human movement and movement in video game.
   Result: Passed with 35 ms ping between computer vision and unity scripts
3. Passing = 100% Accurate hit detection in game with dynamic damage calculation depending on where is hit
   Result: Passed with damage detection working properly







Design Trade-offs

Calman Filtered:
Smoother at the cost
of a 600ms latency!

- **Smoothing: Lerp (Linear Interpolation) VS. Calman Filter**
  - **Lerp** transitions between two values (in our case, the target (next UDP received set of coordinates) and the current set of coordinates) over time by blending a fraction of the target value into the current value, ensuring smooth transitions between poses rather than snapping to the target positions immediately.

$$Lerp(a, b, t) = (1 - t)a + tb$$

  - The **Kalman filter** is a mathematical algorithm that estimates the true state of a system by combining noisy measurements and predictions based on system dynamics.
  - Lerp Pros: Simple to implement, Computationally Efficient;
  - Cons: No noise handling, Constant smoothing rate (Introducing lag for fast-changing data
  - Calman Filter Pros: Noise Robustness, Predictive Power;
  - Cons: Computationally intensive, Higher Latency

Lerped:
Fast,
but less smoother.

Design Trade-offs

- **Lerp (Linear Interpolation) Internal Trade-off**

```
for (int i = 0; i < pred3d.Count; i++)
{
    pred3dCurrent[i] = Vector3.Lerp(pred3dCurrent[i], pred3d[i], Time.deltaTime * MULTIPLIER);
}
```

  - pred3dCurrent[i] starts at the current position of the joint.
  - pred3d[i] is the new position detected by the pose detection system for that joint.
  - Vector3.Lerp moves pred3dCurrent[i] closer to pred3d[i] based on a smoothing factor (Time.deltaTime * MULTIPLIER).

  High `Time.deltaTime` reduces latency by speeding up the lerping process, but it compromises smoothness by making transitions abrupt.

  Large MULTIPLIER -> Smaller latency **BUT** worse smoothing performance

  Small MULTIPLIER -> Better smoothing performance **BUT** larger latency

Project Management / Next Steps

	PROJECT TITLE	Computer Vision Boxing				
WBS NUMBER	**TASK TITLE**	**TASK OWNER**	**START DATE**	**DUE DATE**	**DURATION**	**PCT OF TASK COMPLETE**
1	Project Conception and Initiation					
1.6	Interim Demo	All	10/23/24	11/18/24	25	100.00%
1.7	Final Presentation Slides	All	11/19/24	12/1/24	12	100.00%
1.8	Poster PDF	All		12/x/24		0.00%
1.9	Final Youtube Video w Wordpress Link	All		12/x/24		0.00%
1.10	Public Demo	All		12/x/24		0.00%
1.11	Final Report	All		12/x/24		0.00%
2	Project Implementation					
2.1	Computer Vision Research	Eric	9/9/24	9/15/24	6	100.00%
2.2	Game Design Research/Brainstorming	Shithe, Taiming	9/16/24	10/20/24	34	100.00%
2.2a	Game Creation - Game Logic	Shithe, Taiming	10/11/24	10/18/24	7	100.00%
2.2b	Game Creation - CV Integration	Taiming	10/1/24	10/23/24	22	100.00%
	Game Creation - CV Tuning	Taiming	12/2	12/9/24	7	0.00%
2.2c	Game Creation - Graphics	Shithe	9/30/24	10/18/24	18	100.00%
2.2d	Game Creation - Debugging/Polish	Shithe, Taiming	10/7/24	10/28/24	21	100.00%
2.2e	Game Creation - Switching Character Models	Shithe,Taiming	12/2/24	12/9/24	7	0.00%
2.3a	Hardware - Research	Eric	9/16/24	9/22/24	6	100.00%
2.3b	Hardware - PCB Design	Eric	9/23/24	9/29/24	6	100.00%
2.3c	Hardware - Parts Turnaround	Eric	9/30/24	10/14/24	14	100.00%
2.3d	Hardware - Assembly	Eric	10/14/24	10/18/24	4	100.00%
2.3e	Hardware - Arduino Code	Eric	10/11/24	10/25/24	14	100.00%
2.3f	Hardware - Debugging	Eric	10/28/24	11/15/24	17	100.00%
2.3f	Hardware - Testing	Eric	11/18/24	11/29/24	11	100.00%
2.4a	Integration - Beta Testing	All	12/2/24	12/9/24	7	0.00%
2.4b	Integration - Alpha Testing	All	11/23/24	12/15/24	22	80.00%