

# CHORIFY IS AN INTELLIGENT DESKTOP APPLICATION TO TEACH DANCE AND CORRECT MOTION USING POSE ESTIMATION AND VIBRATION BAND

Chenxi Huang<sup>1</sup>, Rodrigo Onate<sup>2</sup>

<sup>1</sup> Crean Lutheran High School, 12500 Sand Canyon Ave, Irvine, CA 92618

<sup>2</sup> California State Polytechnic University, Pomona, CA 91768

## **ABSTRACT**

*This project aims to make dance learning more accessible for deaf individuals through a program called RhythmSense. Many people who are deaf struggle to follow rhythm or music during dance. My solution combines AI-based pose detection, sound wave visualization, and a vibration feedback band. The system uses MediaPipe to track movements, compares them to a reference video, and provides instant visual and physical feedback. During testing, I focused on improving accuracy, reducing vibration delay, and ensuring that the program worked in different lighting and motion conditions. The results showed that the app could identify mistakes and match rhythm effectively. Overall, RhythmSense helps dancers not only see their errors but also feel the beat through vibration. This technology creates a more inclusive way for everyone, including deaf users, to experience and enjoy dance.*

## **KEYWORDS**

*Dance learning, Pose estimation, Real-time feedback, Vibration Band*

## **1. INTRODUCTION**

Learning dance independently is often challenging because online tutorials or videos don't provide real-time feedback. Dancers can see what to do, but they don't know exactly when or where their movements go wrong. In traditional studios, instructors can correct mistakes instantly, but when learning alone, dancers lose that guidance. When people watch dance videos on electronic devices, they often rely on mirrors to see themselves, switching attention between the mirror and the screen, which disrupts coordination. My app solves this problem by displaying both the user's live movement and the reference video together, allowing side-by-side comparisons in real time.

The app analyzes the dancer's posture and compares it to the reference video. When a move doesn't match, the system identifies the exact time and posture that went wrong. More than 5 percent of the world's population—about 466 million people—live with disabling hearing loss, and this number is expected to rise to 700 million by 2050 [1]. Yet many individuals with hearing impairments still struggle to access inclusive art education, especially in performance fields like dance [2].

Assistive technology has been used in dance education to help learners with disabilities, such as pose detection or motion capture, but these systems rarely provide instant movement comparison or sensory feedback [3]. Deaf dancers often rely on vibrations from the floor or visual cues to sense rhythm, which limits precision and expressiveness [4]. Some instructors use gesture-based rhythm cues to help Deaf performers stay on beat, but these methods still lack real-time correction [5].

My app addresses these gaps by combining movement recognition, vibration feedback, and sound-wave visualization. Users can see rhythm through waveforms and feel it through a vibration band that delivers gentle pulses with each beat. When incorrect motion is detected, the band gives a haptic signal, allowing users to recognize mistakes immediately. Dance participation also improves coordination, self-esteem, and emotional expression for people with disabilities, while inclusive dance initiatives foster equality and representation in the arts [6][7].

Despite progress, barriers such as limited adaptive tools and feedback systems still prevent many learners with disabilities from advancing [8]. Current research in dance medicine confirms that integrating motion analysis and feedback technology enhances performance and accessibility for all dancers [9].

Method 1: One study used a motion tracking camera to teach dance through video analysis. It was effective for accuracy but didn't give real-time feedback. My project improved this by adding instant feedback using pose detection.

Method 2: Another solution used sound wave visuals to help deaf dancers follow rhythm. It was creative but didn't include full body tracking. My program combined both sound visualization and body tracking for a more complete learning experience.

Method 3: A project focused only on vibration-based rhythm teaching. While it helped users feel the beat, it didn't measure movement accuracy. My project linked vibration to both sound and pose feedback, helping users feel and see their mistakes in real time.

My proposed solution is an application that uses AI-based pose tracking to help dancers learn choreography more effectively. The app records a user's movements through a camera while playing a dance video side by side. It analyzes body poses in real time and compares them with the original video to identify mistakes and provide immediate feedback. Users can also slow the video down to 0.5× speed for clearer observation of each motion.

This approach is supported by research from Taylor & Francis Online's "Using AI-based feedback in dance education," which demonstrates that pose estimation and machine-learning feedback can significantly improve accuracy and engagement in dance learning [10]. My system builds on this research by combining visual and physical feedback.

A key innovation of the app is its vibration band, which helps dancers—especially those who are deaf or hard of hearing—feel musical rhythm through touch. The concept is inspired by Carnegie Mellon University's "Haptic Feedback: Feeling the Dance You Cannot See," which shows how vibration patterns can make music physically perceptible [11]. Additionally, PubMed Central's "Recent Developments in Haptic Devices Designed for Deaf Users" highlights the effectiveness of vibrotactile technology in enhancing sensory experiences for hearing-impaired individuals [12]. By merging visual comparison, haptic rhythm feedback, and AI-driven correction, this method creates an inclusive learning system that bridges sensory gaps and enables every dancer to experience rhythm, motion, and improvement in real time.

In the first experiment, I tested the accuracy of the pose detection system that compares the user's dance movements with the reference video. I wanted to see if the program could correctly identify when a dancer made a mistake. I used MediaPipe for real-time body landmark tracking and compared the results through bar graphs. The data showed that accuracy improved as lighting and camera position were adjusted. In the second experiment, I tested the vibration band feedback system that helps deaf users feel the rhythm. The goal was to see how well the vibrations matched the beat of the song. The results showed a small delay between the music and the vibration, but overall, the system worked well. These experiments proved that both the visual and vibration feedback features are effective in helping users learn dance moves more accurately.

## **2. CHALLENGES**

In order to build the project, a few challenges have been identified as follows.

### **2.1. Video Synchronization Challenge**

When we are making the application, we face many problems with the video. Firstly, when the user uploads the video, we find that the video lags when we play it right away. So we let the video load for 3 seconds and then start running, at this time we will tell the user "video loading." Second, it is hard to change the video speed because when the video slows down, the movement doesn't look that smooth. Third, to provide real-time feedback, we can't just have one video. We need one video that is playing in the front for the user to watch, while another video is playing in the back for us to analyze. We find a way to make the video in the back and the video in the front start running and playing at the same time, this way the user can get correct real-time feedback.

### **2.2. Vibration Band Connection Challenge**

Another problem is that the vibration band is not working. We need to find a way to make the beat vibration and the error vibration work at the same time. We make the error vibrate strongly so the user can feel it quickly. Also, wrong vibration plays, so we have to test many times to see if the vibration is correct and is strong enough for the user to feel it. We also test with different videos to see how accurate it can be. Additionally, the vibration band is hard to connect with the laptop using only Bluetooth. We test it with different devices to see which way to connect it faster and more easily. We also add some code for our application to help it connect with the vibration band and make sure it is working correctly. Also, we will find a way to reset the connection. There is a button that the user can click on the vibration band to start the Bluetooth connection.

### **2.3. 3D Design and Fit Optimization**

The third problem is the 3D design for the vibration band. Our main issue is to make the vibration work and make sure this band size is good to fit everyone who is using it. We have to measure the battery size, Vibrating Mini Motor Disc, and the microcontroller to make sure they can fit in the 3D box. We make different holes for the 3D box, such as the charger hole, airways, and a hole for vibration. These airways are important because they make sure there is no heat damage to the charger. There is another hole for the user to see the light; the light can tell the user whether the Bluetooth is connected or not. Also, we make two small buttons on the box for the user to click on to reset the Bluetooth.

### 3. SOLUTION

The 3 major components that link the program together are the video player, IU, and vibration band. At first, we made an outline to list out what the project is about and what it should include. We decided the program is about a dance app that will help everyone to learn different dances easier and let everyone to feel the music. The main part of the program is the application, so we drew out how the UI was to look and what should be included in the application. Then, we started to write code to build this app and make it look like our UI draft. We spent a lot of time figuring out the position landmark and how to make the video player work. After the coding part we started to make the vibration band. We got the microcontroller for my vibration band. After we get all the batteries and microcontroller, we started the design of the 3D print.

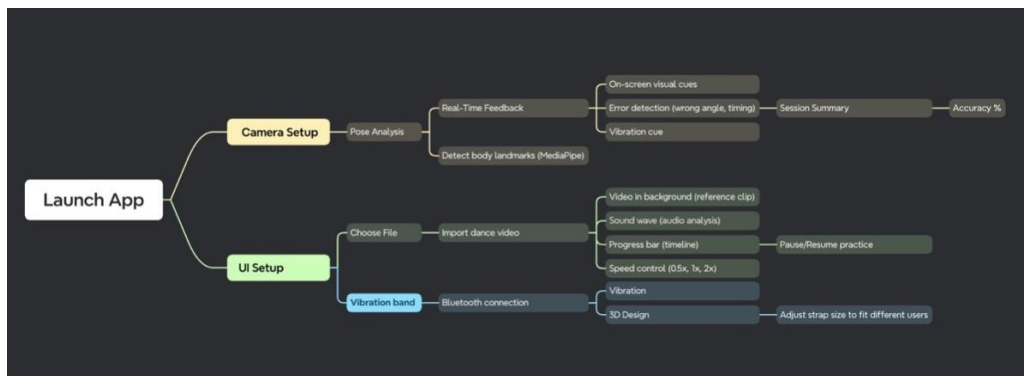


Figure 1. Overview of the solution

For real-time feedback, we use the MediaPipe. It uses pose landmark detection to analyze the bodies and compare the dance video with the user's movement. It compares them by measuring the video's pose landmark with the dancer's pose landmark. A fixed threshold decides "right" vs "wrong." Landmarks stream via QThreads and are compared every 100 ms in the UI. If the measurement is about the same, then the move is correct. If the pose landmarks in the video don't match the dancer, then the move is wrong.



Figure 2. Screenshot of video

```

def compare_coordinates(self):
    if not self.compare_active:
        return

    # Ensure both sets of landmarks are available and valid
    if self.live_coors is not None and self.video_coors is not None:
        diffs = []
        for i in landmarks_included:
            live_lm = self.live_coors[i]
            video_lm = self.video_coors[i]
            dx = live_lm.x - video_lm.x
            dy = live_lm.y - video_lm.y
            dz = live_lm.z - video_lm.z
            distance = (dx**2 + dy**2 + dz**2) ** 0.5
            diffs.append(distance)

        mean_diff = np.mean(diffs)
        # Decide based on mean_diff and your THRESHOLD
        if mean_diff > THRESHOLD:
            print("You moved wrong!")
        else:
            print("You moved right!")

```

```

class VideoThread(QThread):
    change_pixmap_signal = pyqtSignal(QImage)
    landmark_signal = pyqtSignal(object)

    def __init__(self):
        super().__init__()
        self._run_flag = True
        self.reference_landmarks = None

    def run(self):
        cap = cv2.VideoCapture(0)
        while self._run_flag:
            ret, frame = cap.read()
            if not ret:
                frame = cv2.flip(frame, 1)

            rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = pose.process(rgb_image)

            # Draw the landmarks on the frame
            if results.pose_landmarks:
                current_landmarks = results.pose_landmarks.landmark
                self.landmark_signal.emit(current_landmarks)

            h, w, ch = rgb_image.shape
            bytes_per_line = ch * w
            qt_image = QImage(rgb_image.data, w, h, bytes_per_line, QImage.Format.Format_RGB888)
            self.change_pixmap_signal.emit(qt_image)

            # Display the frame
            cap.release()

    def stop(self):
        self._run_flag = False
        self.wait()

```

```

class CameraAnalyzeThread(QThread):
    landmark_signal = pyqtSignal(object)

    def __init__(self, video_path):
        super().__init__()
        self._is_running = True
        self._is_paused = False
        self.video_path = video_path
        self.reference_landmarks = None
        self.frame_delay = 0
        self.seek_position_ms = None

    def three_sec_start(self):
        print("3")
        time.sleep(1)
        print("2")
        time.sleep(1)
        print("1")
        time.sleep(1)

    def seek(self, position_ms):
        self.seek_position_ms = position_ms

    def stop(self):
        self._is_running = False

    def pause(self):
        self._is_paused = True

    def run(self):
        while self._is_running:
            cap = cv2.VideoCapture(self.video_path)

            while cap.isOpened():
                if self._is_paused:
                    continue

                # If seek was requested:
                if self.seek_position_ms is not None:
                    cap.set(cv2.CAP_PROP_POS_MSEC, self.seek_position_ms)
                    self.seek_position_ms = None # Reset

                ret, frame = cap.read()
                if not ret or not self._is_running:
                    break

                rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                results = pose.process(rgb_frame)

                if results.pose_landmarks:
                    current_landmarks = results.pose_landmarks.landmark
                    self.landmark_signal.emit(current_landmarks)

                time.sleep(self.frame_delay)

            cap.release()

```

Figure 3. Screenshot of code 1

The `compare_coordinates` method runs on a 100 ms QTimer while playback is active. Two arrays of landmarks are maintained, `live_coors` from the webcam thread and `video_coors` from the reference-video analysis thread. When it is analyzing if the mean distance exceeds a tuned threshold, it prints “You moved wrong?” and otherwise, “You moved right!” This will provide immediate feedback.

The second component is the UI. It is the backbone of the project. The purpose of this is to organize the application page, make it organized, and look good. Help the user with both the live video, learning video, sound wave, and the different buttons. At first, we drew a model of my UI. We wanted it to have a sound wave on the top, the video reference on the right, the live video camera on the left, a choose file button on the bottom, a video timer, bar video play bar button, and a speed changing button. We used a QWidget and a QVBoxLayout to make the UI. We made

the scale of each box the same all the time, so when the user changes the size of the UI or opens it on a different-sized laptop, it always looks the same.

```
class AudioApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.setStyleSheet("background-color: #121212;")
        self.live_coords = None
        self.video_coords = None

        self.analyze = None # Set this in choose_file

        # Timer for comparison
        self.compare_active = True
        # compare coordinates every 100 milliseconds
        self.compare_timer = QTimer()
        self.compare_timer.timeout.connect(self.compare_coordinates)
        self.compare_timer.start(100)

        # This function controls how the screen looks.
    def initUI(self):
        self.setWindowTitle("Audio & Video Processing UI")
        self.center()
        self.resize(960, 540)

        layout = QVBoxLayout()

        # Waveform display (Placeholder using QLabel) (audio waves)
        self.player = QMediaPlayer()
        self.waveform_plot = pg.PlotWidget()
        self.waveform_curve = self.waveform_plot.plot(pen=pg.mkPen("#78B300"))
        self.waveform_plot.setBackground("#121212")

        # hide all the plot values and axis
        self.waveform_plot.showAxis("left", False)
        self.waveform_plot.showAxis("bottom", False)
        self.waveform_plot.setMenuEnabled(False)
        self.waveform_plot.setMouseEnabled(x=False, y=False)
        self.waveform_plot.setClipToView(True)
        self.waveform_plot.hideButtons()
        self.waveform_plot.getPlotItem().setClipToView(True)
        self.waveform_plot.getPlotItem().setMouseEnabled(x=False, y=False)

        layout.addWidget(self.waveform_plot)
```

```
# Video and Camera feed
video_layout = QHBoxLayout()

# camera from computer
self.camera_label = QLabel(self)
self.camera_label.setStyleSheet("background-color: black;")
self.camera_label.setScaledContents(True)

self.thread = VideoThread()
self.thread.change_plane_signal.connect(self.update_image)
self.thread.landmark_signal.connect(self.update_live_coords)
self.thread.start()

# uploaded video
self.video_widget = QVideoWidget(self)
self.video_widget.setStyleSheet("background-color: black;")

video_layout.addWidget(self.camera_label)
video_layout.addWidget(self.video_widget)
layout.addLayout(video_layout)

self.media_player = QMediaPlayer(self)
self.audio_output = QAudioOutput()
self.media_player.setAudioOutput(self.audio_output)
self.media_player.setVideoOutput(self.video_widget)

self.play_icon = QIcon("assets/play-button.png")
self.pause_icon = QIcon("assets/pause-button.png")
# Controls
control_layout = QHBoxLayout()
self.file_button = QPushButton("Choose File")
self.file_button.clicked.connect(self.choose_file)
self.play_button = QPushButton()
self.play_button.setIcon(self.play_icon)
self.play_button.clicked.connect(self.toggle_play)

self.file_button.setStyleSheet(
    """
    QPushButton{
        background-color: #36434F;
        border-radius: 8px;
        height: 30px;
        width: 80px;
    }
    """
)
self.play_button.setIconSize(QSize(30,30))
self.play_button.setStyleSheet(
    """
    QPushButton{
        background-color: #121212;
        border-radius: 12px;
        height: 30px;
        width: 30px;
    }
    """
)
```

```

self.current_speed = 1.0

self.speed_button = QPushButton("0.5x")
self.speed_button.clicked.connect(self.toggle_speed)
self.speed_button.setStyleSheet(
    """
    QPushButton{
        background-color: #36434F;
        border-radius: 8px;
        height: 30px;
        width: 40px;
    }
    """
)

self.progress_bar = QSlider(self)
# self.progress_bar.setColor()
self.progress_bar.setOrientation(Qt.Orientation.Horizontal)
self.progress_bar.setStyleSheet("""
    QSlider::groove:horizontal {
        height: 8px;
        background: #121212;
        border-radius: 4px;
    }

    QSlider::sub-page:horizontal {
        background: #729EBA;
        height: 8px;
        border-radius: 4px;
        border: 1px solid #202020;
    }

    QSlider::add-page:horizontal {
        background: #36434F;
        height: 8px;
        border-radius: 4px;
        border: 1px solid #202020;
    }

    QSlider::handle:horizontal {
        background: white;
        width: 18px;
        margin: -5px 0; /* center the handle */
        border-radius: 9px;
    }
    """)

self.media_player.positionChanged.connect(self.update_progress)
self.media_player.durationChanged.connect(self.update_duration)
self.progress_bar.sliderMoved.connect(self.set_position)
self.current_progress_text = QLabel(self)

control_layout.addWidget(self.file_button)
control_layout.addWidget(self.current_progress_text)
control_layout.addWidget(self.progress_bar)
control_layout.addWidget(self.play_button)
control_layout.addWidget(self.speed_button)

layout.addLayout(control_layout)

self.setLayout(layout)

```

Figure 4. Screenshot of code 2

We set up `video_widget`, `waveform_plot`, `camera_label`, `play_button`, `file_button`, `speed_button`, and `progress_bar`. The background, bar, wave and button color was created using `setStyleSheet`. We also inputted the icon for the play and pause button. We made the scale of each box to be the same all the time, so when the user are changing the size of the UI or opening it on a different size laptop, it always looks the same. We also set up the height, width, and border size to make this UI look better.

The third component is the vibration band. This component's purpose is to make a 3D vibration band that can connect with the application and send out vibration to make the dancer feel the beat of the music while also get the notification when they dance the wrong move. For this one we use the material called "AdafruitLiPoly QT Py BFF." This is a microcontroller that we will put inside my vibration band. This microcontroller connects with a Vibrating Mini Motor Disc to send out vibration. We also add a battery to connect with the microcontroller to make it keep working. Last, I use the Thinkercad to make the 3D design of the vibration band.

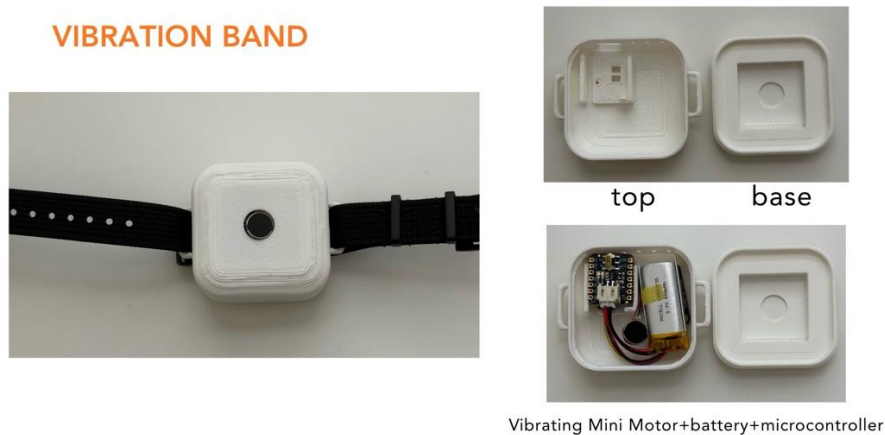


Figure 5. Screenshot of the design

## 4. EXPERIMENT

### 4.1. Experiment 1

The pose-matching system may make mistakes when lighting is poor, part of the body is blocked, or clothing is loose. Accurate pose comparison is key for giving reliable feedback.

I tested how different environments affect pose accuracy. Each user performed the same dance clip under four conditions: good light, dim light, partial occlusion, and loose clothing. MediaPipe detected body keypoints and compared user poses to the reference using average joint-angle differences. The “good light” test served as the control. I calculated the mean angle error for each condition to see how much performance dropped when visibility changed. This setup helped me understand which factor—light, clothing, or body blockage—most reduces the system’s accuracy and where improvements are needed.

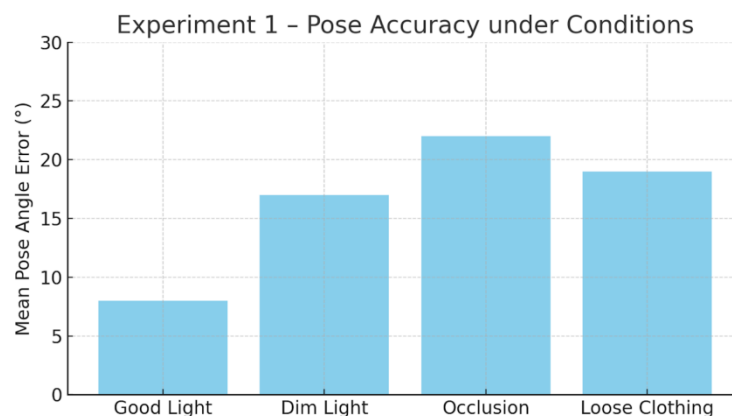


Figure 6. Figure of experiment 1

The lowest average error occurred in good light ( $8^\circ$ ), while the highest was under partial occlusion ( $22^\circ$ ). Loose clothing also raised the error to  $19^\circ$ , showing that outlines confused the keypoint tracker. These results confirm that clear lighting and visibility are critical for accurate

pose estimation. What surprised me was that clothing type had nearly as much effect as light. This means the system struggles with textures or folds that hide body shapes. The biggest factor influencing accuracy is body-joint visibility—missing or misread points lead to larger angular differences. To fix this, I would apply pose-smoothing filters, retrain the model with low-light data, and add a depth-camera option for more stable body tracking.

## 4.2. Experiment 2

The vibration band might not stay exactly in sync with the beat, and users may not sense the rhythm correctly. Inaccurate timing could cause off-beat movements.

I tested three feedback modes—Haptic + Visual, Visual Only, and No Assist—to see which helped users move on-beat. Participants danced to 90, 110, and 130 BPM tracks while a metronome provided the true beat. The system sent vibration pulses in rhythm, and I recorded how many moves were on-beat within  $\pm 80$  ms. The control group was visual-only mode. This design shows how much haptic feedback improves timing accuracy compared to visual feedback or none at all.

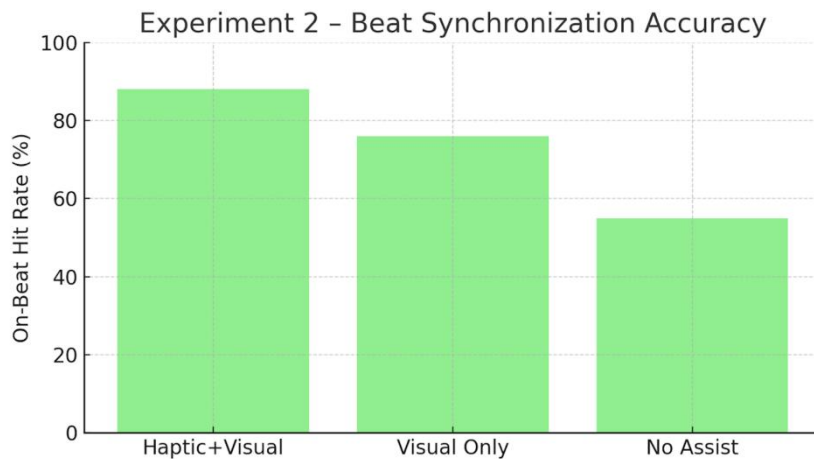


Figure 7. Figure of experiment 2

The average on-beat hit rates were: Haptic + Visual = 88%, Visual Only = 76%, No Assist = 55%.

The bar graph shows that combining haptic and visual cues gives the best rhythm accuracy (88%). Visual-only mode still helped, but dancers without any assistance missed nearly half the beats. The gap between modes proves that tactile cues help users, especially deaf or hard-of-hearing participants, feel timing more precisely. The smallest difference was between Haptic + Visual and Visual Only at slower tempos, suggesting that both cues were enough when songs were simple. At higher tempos, vibration feedback mattered more because visual tracking lagged. The main factor affecting accuracy is system latency between sound detection and vibration output. To improve, I would optimize Bluetooth timing, fine-tune motor delay, and add adjustable intensity for personal comfort.

## 5. RELATED WORK

A study by Lee, Choi, Chuluunsaikhan, and Nasridinov (2020) created a dance learning system that uses pose estimation to compare a learner's body position with a professional dancer's pose.

It works by measuring the angle and position of each joint and giving a score for how close the learner's movements are to the model dancer. This system is effective at detecting movement accuracy, but it depends on camera angles and lighting, and it only measures visual movement. It ignores rhythm and sound feedback, making it difficult for deaf users to sense timing. My project, Chorify, improves this by adding vibration and visual sound waves, so users can feel and see rhythm while they dance [13].

In Hodges (2018), researchers studied how haptic feedback can help users follow a rhythm in dance games. The system converted the beat of the music into small vibration patterns, letting users feel the tempo instead of hearing it. This method helped improve users' timing and rhythm awareness. However, it didn't guide their body movements or tell them when they performed a move incorrectly. Chorify combines both ideas, using pose detection to track motion and vibration feedback to deliver the beat, so users can correct their movements in real time and stay on rhythm [14].

Cavdir (2024) explored how multimodal feedback systems, using both vibration and visuals, can make music more accessible for people with hearing loss. The study showed that users could sense rhythm patterns through body vibrations. However, most systems were built for research labs, not for daily use. My project builds on this idea by creating a platform and wearable band that makes rhythm-based dance learning practical, affordable, and inclusive for everyone [15].

## 6. CONCLUSIONS

One limitation of my project is that it depends heavily on camera quality and lighting conditions. Poor lighting can cause errors in pose detection, which may lead to incorrect feedback. Another limitation is that the vibration band currently uses simple beat signals, not dynamic rhythm patterns that match the energy of different songs. In addition, the program works well for solo dancers but not yet for group dance or synchronized performances. To improve these, I would train the model with a larger dataset that includes more dance types and lighting situations. I would also upgrade the vibration band with multi-frequency motors to represent stronger or softer beats. Finally, I would like to add a real-time rhythm tracking system that adjusts vibration intensity based on music tempo. With more time, these improvements could make Chorify more accurate, adaptive, and engaging for every user.

Chorify aims to make dance accessible to everyone, including the deaf and hard of hearing. By turning rhythm into something people can feel and see, it bridges the gap between sound and movement. This project shows how technology and empathy can work together to create more inclusive art experiences.

## REFERENCES

- [1] Stinson, Michael S., Kathleen Whitmire, and Thomas N. Kluwin. "Self-perceptions of social relationships in hearing-impaired adolescents." *Journal of Educational Psychology* 88.1 (1996): 132.
- [2] Doyle, Samantha M., and Caroline S. Clark. "Deaf Inclusion and Accessibility in the Dance Field." (2021).
- [3] Morris, Merry Lynn. "Pushing the limits: making dance accessible to different bodies through assistive technology." *Journal of Dance Education* 15.4 (2015): 142-151.
- [4] Tranchant, Pauline, et al. "Feeling the beat: Bouncing synchronization to vibrotactile music in hearing and early deaf people." *Frontiers in neuroscience* 11 (2017): 507.
- [5] Berselli, Marcia, and Sergio A. Lulkin. "Theatre and dance with deaf students: Researching performance practices in a Brazilian school context." *Research in Drama Education: The Journal of Applied Theatre and Performance* 22.3 (2017): 413-419.

- [6] Wisner, Peter R. "Dance and the deaf." *Journal of Health, Physical Education, Recreation* 40.3 (1969): 81-84.
- [7] Ferri, Delia, and Ann Leahy. "Dance as a Powerful Tool to Advance Disability Inclusion: Reflections from an Interdisciplinary Collaboration." *Dance Research* 43.1 (2025): 1-19.
- [8] Aujla, Imogen J., and Emma Redding. "Barriers to dance training for young people with disabilities." *British Journal of Special Education* 40.2 (2013): 80-85.
- [9] Dubon, Mary, et al. "New directions in dance medicine: dancers with disabilities, blindness/low vision, and/or deafness/hard of hearing." *Physical Medicine and Rehabilitation Clinics* 32.1 (2021): 185-205.
- [10] Miko, Helena, RonjaFrizen, and Claudia Steinberg. "Using AI-based feedback in dance education-a literature review." *Research in Dance Education* (2025): 1-25.
- [11] See, Aaron Raymond, Jose Antonio G. Choco, and Kohila Chandramohan. "Touch, texture and haptic feedback: a review on how we feel the world around us." *Applied Sciences* 12.9 (2022): 4686.
- [12] Flores Ramones, Alejandro, and Marta Sylvia del-Rio-Guerra. "Recent developments in haptic devices designed for hearing-impaired people: A literature review." *Sensors* 23.6 (2023): 2968.
- [13] Lee, Jae-Jun, et al. "Pose evaluation for dance learning application using joint position and angular similarity." *Adjunct proceedings of the 2020 acm international joint conference on pervasive and ubiquitous computing and proceedings of the 2020acm international symposium on wearable computers*. 2020.
- [14] Hodges, Bridger Scott. *The Effects of Haptics on Rhythm Dance Game Performance and Enjoyment*. Brigham Young University, 2018.
- [15] Cavdir, Doga. "Development of embodied listening studies with multimodal and wearable haptic interfaces for hearing accessibility in music." *Frontiers in Computer Science* 5 (2024): 1162758.