AI DANCE TEACHER

By

Yifan Chen
Ziyang Xu
Chao Xu

Final Report for ECE 445, Senior Design, Spring 2020
Group/Project No. 1
May 29th, 2020

Abstract

Nowadays, there are increasing market demands that people wish to do physical exercises like dancing at home which is more convenient compared to going to the fitness/dancing center, especially during the epidemic. We propose AI Dance Teacher, an integrated resolution to support dance study by applying a pose estimation algorithm on a cheap and small edge device: Jetson Nano development board. For the software part, we deploy a lightweight Deep Neural Network (DNN) model to achieve real-time pose estimation. With pose key points, we evaluate dancing scores for each frame and provide vibration feedback through Bluetooth-controlled wearable vibrators. Overall, our hardware-software codesign enhances the experience of a real-time dancing self-learning process at the Edge.

Contents

1. Introduction (Yifan, Chao)

1.1 Problem Statement

Exercises like dance, yoga, and even Tai Chi have become very popular. When taking related courses, the coach often doesn't have enough time to help everyone with each pose. And since everyone learns new things in his or her own pace, classes with other students are not best for every student. If one chooses to learn by following the video, it's also hard to imitate correct gestures, and even worse, wrong gestures may hurt your body. Especially during the epidemic just like the COVID-19 pandemic now, people are in quarantine and unable to visit offline fitness centers. For example, many students take Physical Education classes like aerobics at home, where coaches are not able to correct their poses. It's challenging for teachers to evaluate everyone's movement and give fair grades.

1.2 Project Objective

Our project aims to provide a prospective solution for better self-learning in dancing and to make the learning process more customized. Our AI Dance Teacher is implemented on Jetson Nano [1]. The basic functionalities include real-time 2D pose estimation, evaluation of pose similarity through comparison between the user and the standard instructional video clip, haptic feedback through Bluetooth.

1.3 Solution Overview

Our design is implemented on Jetson Nano with multiple accessories, including a keyboard, a mouse, a monitor, a CSI camera, and a BLE (Bluetooth low energy) module. To support deep learning model inference and visualization we use TensorRT [2], PyTorch [3], and OpenCV [4] modules.

Our design has a real-time pose evaluation based on a standard instructional video. First, we capture the real-time video input of the dancing user and extract the human skeleton from the dancer. We also preprocess skeleton detection from the standard instructional video. For specific preset frames of the video, the real-time pose evaluation algorithm will generate a score indicating the similarity of the learner's pose with the standard pose by comparing the two skeletons. To enable real-time feedback, we also install a vibration unit on the wearable wrist band to indicate how much the user scores. It is remotely controlled using the BLE module. The general block diagram is shown below.

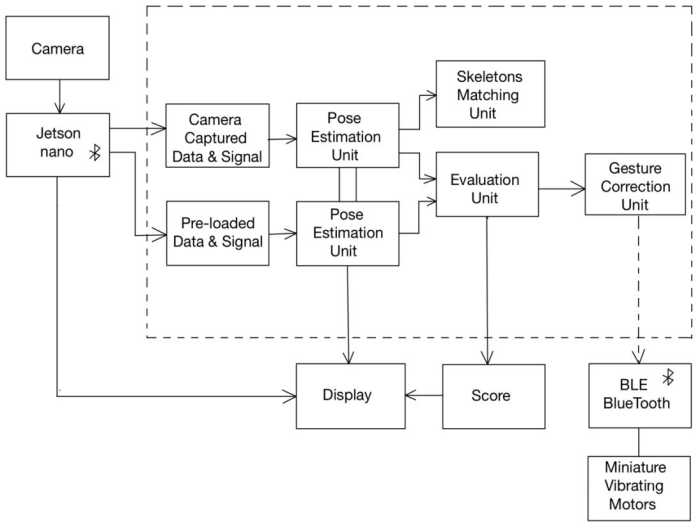


Figure 1. Block Diagram

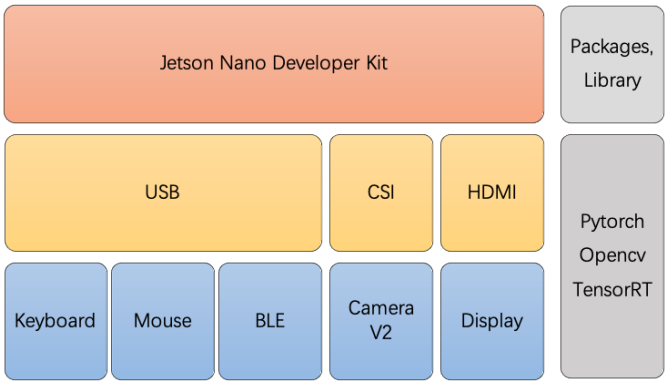


Figure 2. Component Overview

2. Design

Our design is mainly composed of four parts: pose estimation, key-points matching, score evaluation, and vibration feedback. In this section, we will introduce how this project is built in detail. This section is organized as follows: the first part discusses how the software works, which consists of pose estimation, key-points matching, and score evaluation modules. Then, hardware design details are presented.

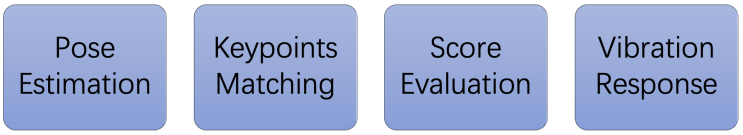


Figure 3. Four Main Parts

2.1 Software (Yifan, Chao)

In order to help users to learn dancing, we want to have their body key-points marked in real-time. In this way, we can compare the user's pose with the instructor's pose in the video both quantitatively and qualitatively/visually. For better visualization of pose difference, we match one pose to the other using an affine transformation matrix, which minimizes the effect of position and rotation differences on scoring. The software flowchart is shown in Figure 4.

2.1.1 Pose Estimation and Pose Matching

Pose estimation is a computer vision technique to detect and locate the joints of human figures in images and videos. We run pre-trained ResNet and DenseNet models that use keypoint task data in MSCOCO format (with 18 keypoints and 21 body parts) [9]. The indices of body points and parts are defined in a JSON file (see Appendix B).

Then, we optimize the pre-trained models with TensorRT using torch2trt, which is a PyTorch to TensorRT converter utilizing the TensorRT Python API. During both the training and testing, we adopt the pre-trained ResNet-18 for detection on 224x224 images.

Body Point	Nose	Left_eye	Right_eye	Left_ear	Right_ear	Left_shoulder	Right_shoulder	Left_elbow	Right_elbow
Index	0	1	2	3	4	5	6	7	8
	Left_wrist	Right_wrist	Left_hip	Right_hip	Left_knee	Right_knee	Left_ankle	Right_ankle	neck
	9	10	11	12	13	14	15	16	17

Table 1. Body Points and Indices in MSCOCO Dataset

For the real-time pose estimation, we capture frames with a resolution of 448 by 448 from the CSI camera using gstreamer_pipeline. There is an API in OpenCV that supports reading and showing video frames named cv2.VideoCapture. The sample codes are attached below. In *process(img)*, we add our customized processing code.

```
def show_camera():
    # To flip the image, modify the flip_method parameter (0 and 2 are the most common)
    print(gstreamer_pipeline(flip_method=0))
    cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
    if cap.isOpened():
        window_handle = cv2.namedWindow("CSI Camera", cv2.WINDOW_AUTOSIZE)
        # Window
        while cv2.getWindowProperty("CSI Camera", 0) >= 0:
            ret_val, img = cap.read()
            ### custom code
            process(img)
            cv2.imshow("CSI Camera", img)
            # This also acts as
            keyCode = cv2.waitKey(1) & 0xFF
            # Stop the program on the ESC key
            if keyCode == 27:
                break
        cap.release()
        cv2.destroyAllWindows()
    else:
        print("Unable to open camera")
```

To boost concurrent processing of both demonstration/teaching videos and real-time camera input, we preprocess the demonstration/teaching videos by doing skeleton detection in advance. The preprocessing procedures are shown as follows:

1. Resize the size of each video frame to 448 by 448 using moviepy package
2. Loop through each video frame: apply pose estimation and draw body skeleton; collect body points and parts for each frame.
3. Save the processed video as a new video clip.
4. Save points and parts to a numpy file.

After preprocessing the video clip, we will start the real-time pose estimation on camera input, which includes pose matching and score evaluation.

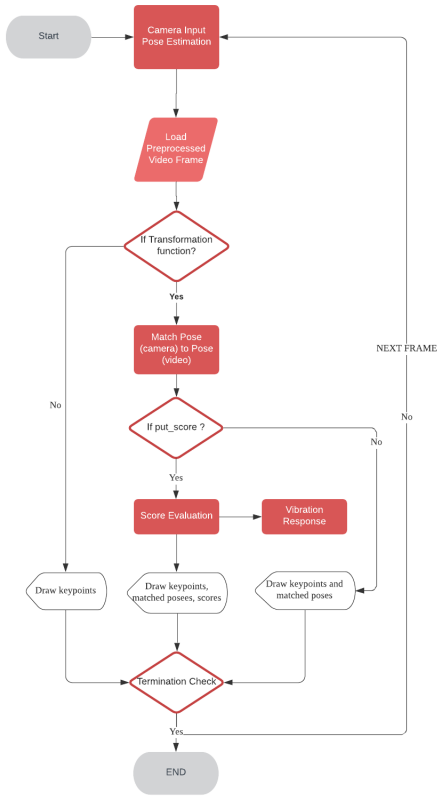


Figure 4. Software Flowchart. The flow describes the processing steps for each pair of frames.

2.1.2 Pose Key-Points Matching

In the last step, we calculate the skeleton structure of our AI Dance user using the pre-trained **trt_pose** model, and let's call it *input set*. The model output is a set of corresponding indices shown in Table 1. At the same time, we will need skeleton points of the standard human pose, and let's call it *model set*.

To compare the shapes of two objects, the objects must be first optimally "superimposed". Since everyone has a different body type and figure, and the orientation of the user and the standard pose can hardly be the same, our design transforms the input set to enable optimal superposition with the model set. We choose Affine Transformation to realize the mapping.

An Affine Transformation ***f*** is the composition of two functions: a translation and a linear map. The linear map is represented as a multiplication by a matrix **A** and the translation as the addition of a vector **b.** An affine map **f()** acting on a vector x is represented as $\vec{y} = f(\vec{x}) = A\vec{x} + \vec{b}$. With an augmented matrix, it is possible to represent both the linear map and the translation using single matrix multiplication as follows, where x is the input set of points and y is the model set. To find the augmented matrix, we apply the least-squares algorithm to solve it and find the affine transformation matrix.

After solving the matrix, we could calculate x': the input pose displayed onto the model pose, which is the best-fit from input pose to the model pose.

We implement the affine transformation using the following function called solve_affine, which takes both the input point set and model point set. The function returns a transformation method that takes a 2D point as input and outputs the 2D point after transformation.

```
def solve_affine(points1, points2):
    ### STEP1: form validate matrices from the two sets of points
    valid_idx = [0,11,12,15,16,17]
    ### y=A2 * x --- A2 = y * x.inverse
    X = []
    Y = []
    ret_x = []
    ret_y = []
    count = 0
    for i in range(18):
        if type(points1[i])!=type(None) and type(points2[i])!=type(None): # else: not valid, we jump
            ret_x.append(points1[i])
            ret_y.append(points2[i])
            if i in valid_idx:
                X.append(points1[i])
                Y.append(points2[i])
                count+=1

    if(count<=3):
        return 0

    X = np.transpose(np.matrix(X))
    Y = np.transpose(np.matrix(Y))
    X = np.vstack((X,[1]*X.shape[1]))
    Y = np.vstack((Y,[1]*Y.shape[1]))
    A2 = Y * X.I
    return lambda x: np.array(A2*np.vstack((np.matrix(x).reshape(2,1),1))).astype(int)[0:2,0]
```

Then, it comes to the workflow for real-time pose matching and display. We use two cv2.VideoCapture API from OpenCV to capture inputs from CSI camera and the preprocessed video, respectively. To process the matching frame by frame, we use a while loop, and do the following in each loop:

1. Read camera frames and video frames use read() method in the cv2.VideoCapture class.
1. Pose estimation and visualization on camera frame.
2. Load the pose (sets of points and parts) of the corresponding video frame from numpy files

3. If the number of selected body points is more than 3 for both camera and video frames (ignore facial points, selected points include nose, hips and ankles), do affine matching from camera pose to video pose and return a transformation method.
4. Draw and Display the points after processing: map all available points and parts in camera pose onto video pose use the transformation method; display the matched points and skeleton on video frame.

2.1.3 Score Evaluation

In the loop of each set of frames, our design evaluates the user's performance in the score evaluation module. The software flow is illustrated in the following diagram (Figure 5). The cosine similarity outputs the weighted sum of all detection points, and its maximum value is 1. In addition to displaying the current score on the screen, we also calculate the cumulative score.

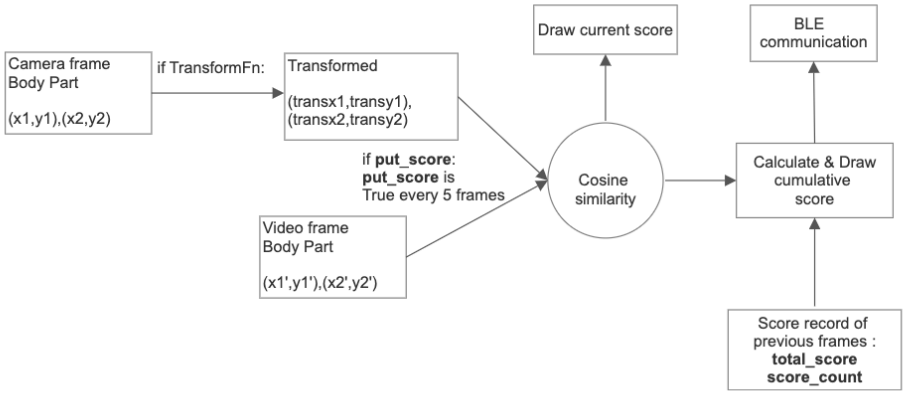


Figure 5. Flowchart for score evaluation

After performing cosine similarity, we set some thresholds for current performance, which is displayed on the screen. And it is worth noting that our Bluetooth communication is based on this performance level, and we will discuss this part in the next section.

Threshold	Display
score>=0.95	Excellent
0.95>score>=0.85	Good
score<0.85	Miss

Table 2. Score corresponding table

2.1.4 Other Functions: Video Clip Control

To improve the user experience, we add a Pause and Continue option for the instruction video use keyboard "P". Our design allows for the adjustment of the video frame rate according to the user input.

2.2 Hardware (Ziyang)

2.2.1 Wired and Wireless Connection Sketch

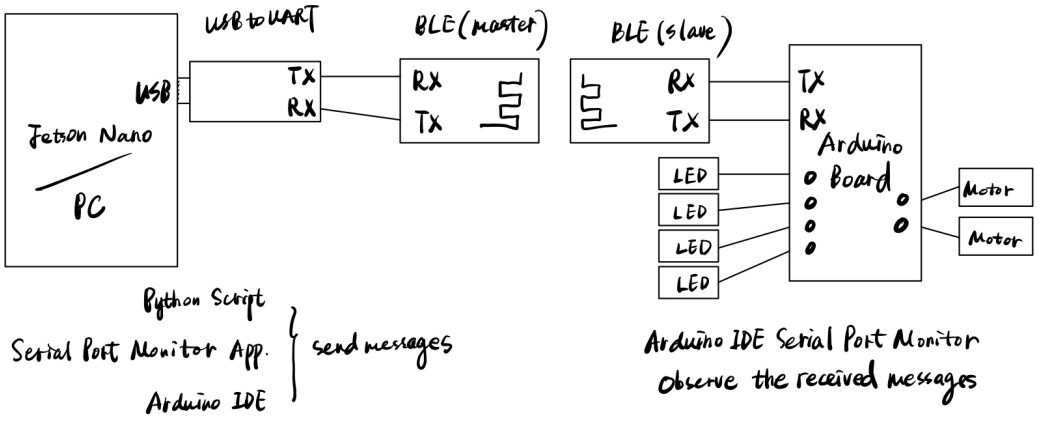


Figure 6. Sketch of Connection

2.2.2 PCB Overview

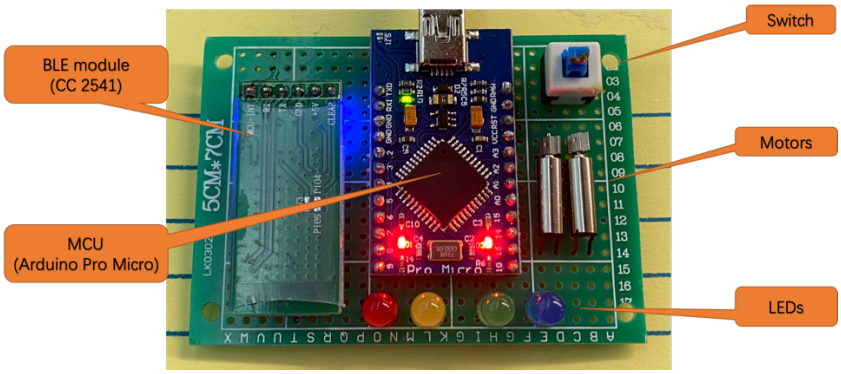


Figure 7. PCB

2.2.3 Communication Device

When the users are dancing, they need to receive feedback to adjust the dancing poses. Thus, we need some communication devices. After comparing three kinds of communication devices: BLE, nRF24L01, and Wi-Fi module, we finally decide to use BLE modules as our wireless communication devices for the following reasons:

I. Wi-Fi module can communicate from one to many but consume a large amount of power.
II. nRF24L01 communicates using SPI protocol, which can cooperate with ATtiny 85, but it seems not easy for us to figure out the usage of the SPI protocol and its verification. Additionally, nRF24L01 can transmit messages from one to many but we still failed after following a lot of tutorials from Google, therefore eventually we give up using it.
III. BLE consumes the least power among these three types of devices. It is easy for us to test its functions using Serial Port Monitor and control it through AT commands. We finally decided to achieve one-to-one communication and regard one-to-many as our future work.

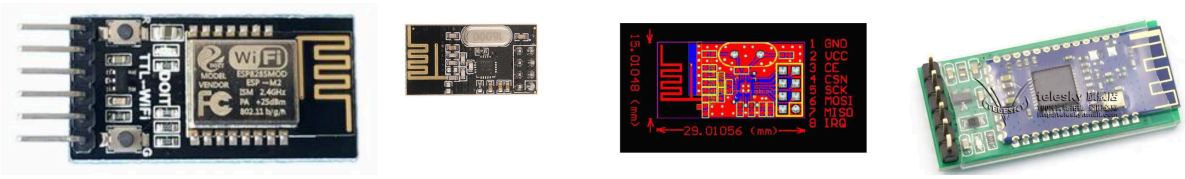


Figure 8. Left: Wi-Fi; Middle: nRF24L01; Right: BLE

2.2.4 MCU (Microcontroller Unit)

Since the wearable devices are required to be lightweight, it is expected that the MCU can be as small as possible. After comparing two small MCUs: ATtiny 85 and Pro Micro, we finally decide to use Micro Pro board for the following reasons:

I. The virtual board of ATtiny 85 is needed to be installed to Arduino IDE.
II. ATtiny 85 does not have TX and RX serial port.
III. ATtiny 85 does not have adequate libraries we need, like "SoftwareSerial", which can analog TX and RX port by using normal Pin I/O port.

Figure 9. Left: ATtiny 85; Right: Micro Pro

The board needs to be installed into Arduino IDE if there is no corresponding virtual board preinstalled in it. Although Micro Pro is a little larger than ATtiny 85, we plan to use it to process the message received from the slave communication device, since it is more convenient for us to study and make a good use of it.

2.2.5 Signal Indicators

Score Levels	>0.95	>0.85	>0.70	<=0.70
LEDs	Blue	Green	Yellow	Red
Motors	Quite Brief Vibration	Brief Vibration	Strong Vibration	Quite Strong Vibration
Screen Display	Excellent	Good	Miss	Miss

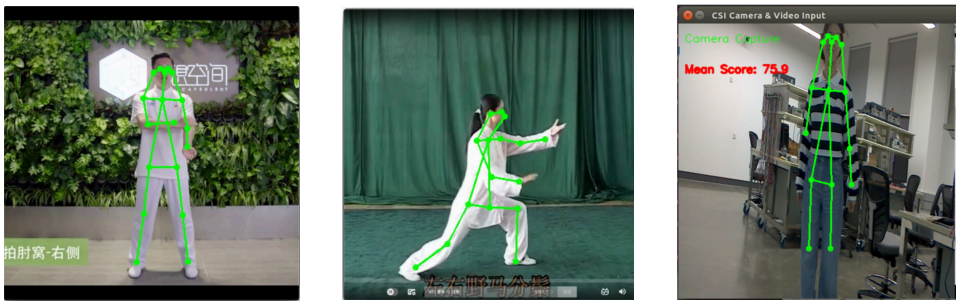
Table 3. Score levels, Performance and Vibration patterns

3. Design Verification

This section demonstrates how we test and verify for desired functionalities and shows results in more detail. In the software part, our design focuses on the visual results, the frame rate and the rationality of the score. Thus, we will verify our design in these aspects.

3.1 Pose Estimation (Yifan)

To verify pose estimation, we check if the model can generate correct detection of human poses on single image inputs first and then real-time video. The detection points and skeleton should be in line with human body parts. The verification images include detection on video clip and camera input (Figure 10).



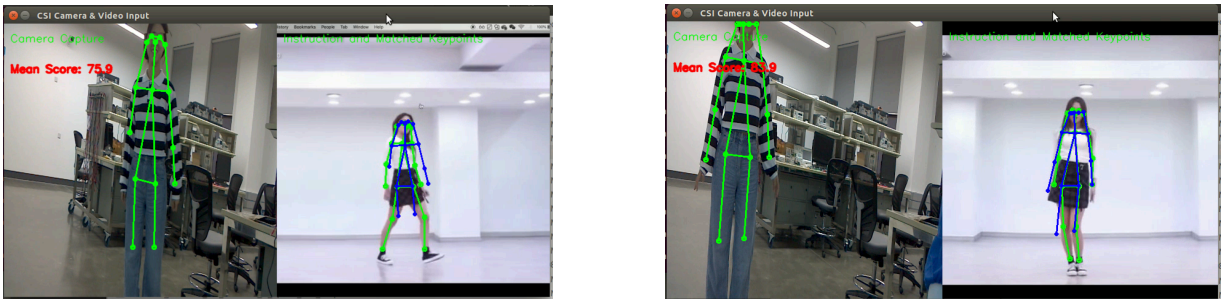
(a) (b) (c)

Figure 10. Verification of Pose Estimation Results. (a) and (b) are pose estimation results from a video frame, and (c) is the result from a camera frame

3.2 Pose Key-points Matching (Yifan)

We stack the camera frame and video frame together in an OpenCV window. In the right part of the window, it displays the instructor's image and skeleton detection results. It also displays the match skeleton from camera frame to video frame, which is drawn in blue.

In Figure 11, we verify the matching result. (a) shows the matching of two different poses, and (b) shows the matching of two similar poses. As we can see, our match eliminates differences in scale, location and direction of each pair of frames.

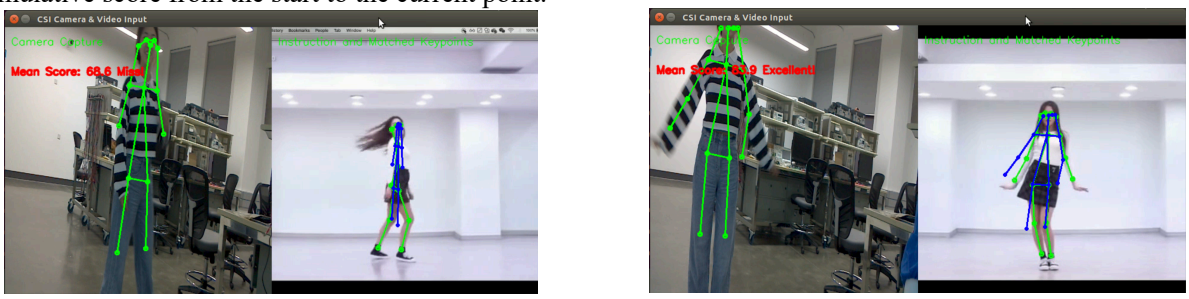


(a) (b)

Figure 11. Verification of pose key-points matching

3.3 Score Evaluation (Yifan)

We verified the score evaluation module by comparing the cumulative score and performance for correct pose and wrong pose. In Figure 12 (a), the user does a different pose from the instruction pose, the performance evaluation indicates it is a "Miss", which is displayed on the top left corner in red text. While in (b), the user does a similar pose with the instruction pose, and the performance level is Excellent. The mean score shows the cumulative score from the start to the current point.



(a) (b)

Figure 12. Verification of score evaluation module.

3.4 BLE (Ziyang)

3.4.1 Setup

BLE could be initialized through the AT command, which can change the parameters of a BLE, like, address, PIN, role, etc. To pair two BLEs, one of them should be a master BLE, which is distinguished by “role=1”, while the slave one should be “role=0” (AT command: AT+ROLE[para]). [10][11]

```
[02:24:27.123] AT

[02:24:27.135] OK
[02:24:38.963] AT+PIN

[02:24:38.983] +PIN=123456
[02:24:44.994] AT+NAME

[02:24:45.014] +NAME=master_new
[02:24:52.375] AT+ROLE

[02:24:52.396] +ROLE=1
```

Figure 13. Snippet: BLEs are manually pre-set up through serial port monitor app.

3.4.2 Pairing (in python script with our main program)

A master BLE could inquire slave devices through the AT command: AT+INQ and connect the found devices through AT+CONN [para]. For a master BLE in connecting mode, it will blink quickly, while a slave BLE will blink slowly. If the two BLEs successfully pair with each other, then both of them will stop blinking and are always on afterward.

The BLE pairing process is controlled by a Python function, which is written in the main program. We aim to get BLEs paired automatically when we run the main program.

```
    serial_port.write("AT\r\n".encode('ascii'))
    serial_port.write("AT+INQ\r\n".encode('ascii'))
    time.sleep(5)
    serial_port.write("AT+CONN1\r\n".encode('ascii'))
```

Figure 14. Snippet: BLEs paired automatically when we run the program.

3.4.3 Verification

We send a message once per second from PC to master BLE and observe the serial port output of the slave BLE through the serial port monitor. We finally find that the slave BLE can frequently wirelessly receive the same message, which can be shown on the monitor. Then we can verify that this pair of BLEs has been successfully set up and can wirelessly transmit and receive messages.

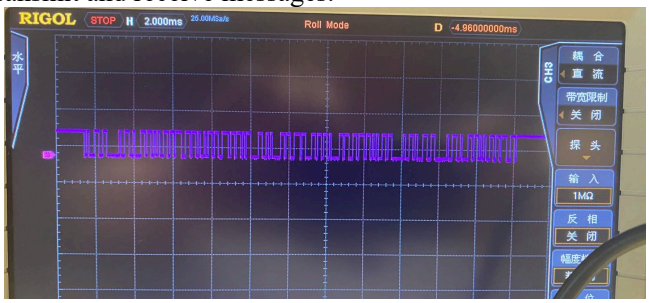


Figure 15. We use Oscillator to capture a message package. The signal is acquired from the TX port of the slave BLE, which will be sent to MCU through the RX port on the Arduino board. This signal is the same as that from the TX port of USBtoUART module on PC.

3.5 Micro Pro MCU (Ziyang)

3.5.1 Pin Signal Control

The programs are edited in Arduino IDE and loaded to Micro pro board.

3.5.2 Serial Port

Since we use Bluetooth as our wireless communication device, we need to make sure that the serial port on our MCU board can correctly work. According to the previous part, we verify that the pair of BLEs successfully connect with each other and are able to correctly transmit and receive messages. It is necessary for us to verify whether the RX port on Micro Pro board can get the messages from TX port of the slave BLE.

3.5.3 Verification

At first, the serial port monitor of Arduino IDE cannot show any message, which means the message from slave BLE cannot access to MCU from RX port of the Arduino board. After research, we find the solution in the corner of the website [8]. We now can receive the correct message. Additionally, different messages from slave BLE can control the Pin I/O port of the board (HIGH or LOW).

```
void setup() {
// Referecence:
//http://arduino.stackexchange.com/questions/1471/arduino-pro-micro-get-data-out-of-tx-pin
  Serial1.begin(9600);
  Serial.begin(9600);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(2,OUTPUT);
  digitalWrite(2,HIGH);
}
```

Figure 16. Snippet: Serial1 for TX, RX port; Serial for Serial Port Monitor

3.6 Signal Indicators and Motor Vibration Modes (Ziyang)

3.6.1 LEDs Activation and Motor Vibration Activation

For the convenience of demonstration and intuitive observations for users, we use LEDs to indicate the status of vibrational motor because they will work synchronously and simultaneously.

3.6.2 Verification

As is mentioned previously, Micro Pro board has been successfully setup and can function well. To verify whether the circuit (connection between LEDs and motors) on the PCB is correct or not, we then use this board to test it.

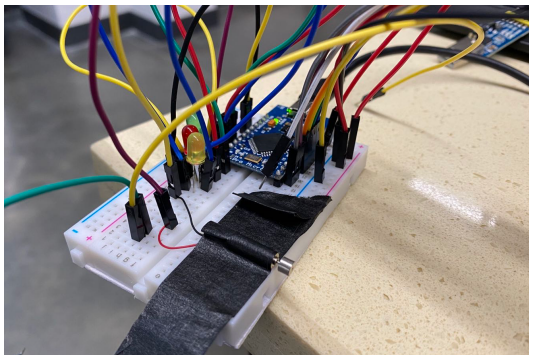


Figure 17. The signal indicators and motors function correctly

3.7 Wearable Device (Ziyang)

3.7.1 Pack Them Together

Finally, we can find all the PCB parts work well, and therefore, we transfer all of them from the bread board to an empty PCB board and weld them together.

3.7.2 Verification

We verify the wearable device can perform normally, which indicates that there is no mistake when we weld the PCB and pack it with an elbow pad.

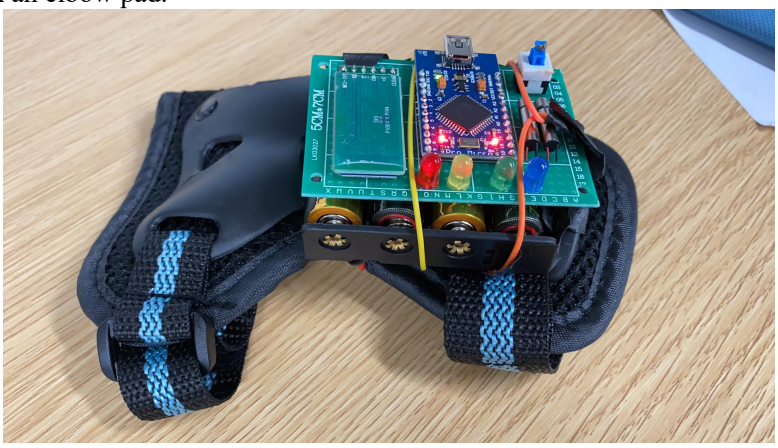


Figure 18. Wearable Device

4. Costs (Yifan, Ziyang)

We purchase some devices and modules that are not used in our final senior design since we need to compare the feasibility and performance.

4.1 Parts

Part	Manufacturer	aRetail Cost (rmb)	Counts	Bulk Purchase Cost (rmb)	Actual Cost (rmb)
Jetson Nano Board	**NVIDIA**	**888**	**1**	**888**	**888**
IMX219 Raspberry Pi Camera v2	**weixue**	**143**	**1**	**143**	**143**
Jetson Nano Wi-Fi Module	**NVIDIA**	**75**	**1**	**75**	**87**
ATtiny 85	**shixun**	**9.17**	**4**	**36.7**	**36.7**
Micro Pro	**shixun**	**30.72**	**4**	**122.90**	**122.90**
Bluetooth Module	**telesky**	**26.89**	**5**	**134.46**	**131.46**
nRF24L01 module	**risym**	**6.78**	**5**	**31.90**	**31.90**
Wi-Fi module	**hc**	**14**	**5**	**70**	**67.90**
USB to TTL Module	**telesky**	**22.17**	**5**	**44.34**	**44.34**
Elbow and Knee Pad	**luoshen**	**35**	**1**	**35**	**35**
Motors	**rongyuhuafu**	**5.9**	**8**	**47.2**	**47.2**
Batteries	**Unknown**	**14.90**	**2**	**29.80**	**29.80**
Battery Box	**arthyly**	**3.74**	**4**	**14.97**	**14.97**
Total					**1680.17**

Table 4. Parts Cost

4.2 Labor

It takes us a semester to complete this project. The actual implementation time is from May 6th to May 26th. The testing time is from May 26th to May 29th. The daily time we spend on this project is about 6 hours per person. Thus, the total labor cost is 20 * 6 * 2 = 240 hours (two person has worked on the implementation part). Also, we spend some time waiting for the delivery of the hardware.

5. Conclusion

5.1 Accomplishments (Yifan)

Through this project, we've realized an AI dance teacher, which is able to assist users to practice and learn dancing. The visualization of dancing poses helps users to recognize their mistakes more easily and accurately. We've also learnt a lot in this project. We put computer vision model into practical use, and have experimented with different communication technologies including RFID, BLE and WIFI. More importantly, we've learnt to use our engineering skills to turn an idea into reality.

5.2 Uncertainties (Yifan)

The 2D pose estimation model cannot detect poses accurately when different body parts have too much overlap, for example, body parts cannot be detected in a very clean manner from a side pose and a bow pose. This uncertainty can be tackled by using 3D pose estimation model that detects 3D key-points from an RGB-D (depth) camera.

5.3 Ethical considerations (Yifan)

Our design takes human body ratios and figures into consideration. We treat fairly all persons and do not engage in acts of discrimination.
Our design is safe for users. It doesn't exert pressure on players to do the exact pose as the instructor does but gives reference on their performance. The vibration is also light and harmless to human body.

5.4 Future work (Chao)

Both the hardware section and the software section expect some future work to make the AI Dance Teacher a user-friendly.

For the hardware, the first issue is that the wearable end device on the wristband is still too large and exposed to the air. To make the wearing experience more seamless and comfortable while better maintaining the device integrity, it is expected to encapsulate the PCB board and minimize its volume occupy, for example replacing that heavy power supply with light coin batteries. If it is lightweight enough, user will be also more sensitive to the vibration motor's haptic feedback. Additionally, we also wish to improve the haptic feedback by introducing more vibrators. Currently, there is a hard-to-solve problem of one-to-multiple Bluetooth control. If we can implement the wireless communication from one master BLE to multiple slave BLEs, we will be able to provide feedback at more body parts rather than one elbow only without adding more master BLEs.

For the software, there is much space for improvement at the application level. It is of great significance to build databases for saving user accounts on the cloud so that users can access the dancing data on various end nodes. In addition, more sample dancing videos in different categories should be provided for dancing practices. Certainly, users can also upload videos that they want to learn.

References

[1] "Jetson Nano Developer Kit," NVIDIA 2020. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano-developer-kit/?ncid=pa-srch-goog-75869&gclid=Cj0KCQjwmpb0BRCBARIsAG7y4zaE_ikMlZLd9-GTlVIQLp3UOAcxoiJctPbuMOkIP56TO3czZI3NolMaAuArEALw_wcB#jetson-nano-dev-kit [Accessed: 02-April-2020]

[2] NVIDIA TensorRT, Programmable Inference Accelerator, NVIDIA 2020 [Online] Available: https://developer.nvidia.com/tensorrt [Accessed: 02-May-2020]

[3] Pytorch, From Research to Production, Pytorch 2020 [Online] Available: https://pytorch.org/ [Accessed: 01-May-2020]

[4] OpenCV 2.4 2020 [Online] Available: https://opencv.org/ [Accessed: 07-May-2020]

[5] OpenPose: Realtime Multi-Person 2D PoseEstimation using Part Affinity Fields. Zhe Cao et al. https://arxiv.org/abs/1812.08008 [Accessed: 02-April-2020]

[6] OpenPose light tensorflow version: https://github.com/ildoonet/tf-pose-estimation

[7] "Single Pose Comparison," Medium 13-Dec-2017. [Online] Available: https://becominghuman.ai/single-pose-comparison-a-fun-application-using-human-pose-estimation-part-2-4fd16a8bf0d3 [Accessed: 02-April-2020]

[8] Arduino Pro Micro, get data out of Tx pin? [Online] Available: http://arduino.stackexchange.com/questions/1471/arduino-pro-micro-get-data-out-of-tx-pin [Accessed: 20-May-2020]

[9] Trt_pose, Github, 03-Mar-2020 [Online] Available: https://github.com/NVIDIA-AI-IOT/trt_pose [Accessed: 01-May-2020]

[10] CC2541 AT Commands [Online] Available: https://www.electrodragon.com/w/CC2541#AT_commands [Accessed: 08-May-2020]

[11] BLK-MD-BC04-B BLUETOOTH MODULE, AT Commands [Online] Available: http://diwo.bq.com/wp-content/uploads/2014/11/BLK-MD-BC04-B_AT-COMMANDS.pdf [Accessed: 09-May-2020]

Appendix A Requirement and Verification Table (Ziyang, Yifan)

Requirement	Verification	Verification status (Y or N)
1. Detect body points and parts from video and camera frame with decent accuracy.	Verified by visualizing detection points on original image. The detection points and skeleton should be in line with human body parts.	Y
2. Display the pose estimation in real-time (frame rate: 10fps)	Verified by timing the total processing time for each frame. And it was also verified in the demo.	Y
3. Match one pose key-point set onto the other to eliminate the difference in locations and orientation.	Verified by visualizing matched skeleton for correct pose and wrong pose.	Y
4. Evaluate the correctness of the movement by comparing poses from video frame and from camera frame.	Verified by comparing the cumulative score and performance for correct pose and wrong pose.	Y
5. Pair a master BLE and a slave one	Verified by sending and receiving messages through Serial Port Monitors	Y
6. Vibration response from four wearable devices.	Verified by LED and vibration impression	Partially Done, only realized vibration response from one device
7. Pause the video and adjust pose	Verified during the demo. (we'll also upload videos on BB to demonstrate it.)	Y
8. Response to one vibrator in real-time via Bluetooth, and display of cumulative score (once every 5 frames)	Verified during the demo	Y
9. Adjust video frame rate as user input in order to learn the slower dance.	Verified during the demo.	Y

Table 5. Requirement and Verification Table

Appendix B JSON file for human key-points definition (Yifan)

{"supercategory": "person", "id": 1, "name": "person", "keypoints": ["nose", "left_eye", "right_eye", "left_ear", "right_ear", "left_shoulder", "right_shoulder", "left_elbow", "right_elbow", "left_wrist", "right_wrist", "left_hip", "right_hip", "left_knee", "right_knee", "left_ankle", "right_ankle", "neck"], "skeleton": [[16, 14], [14, 12], [17, 15], [15, 13], [12, 13], [6, 8], [7, 9], [8, 10], [9, 11], [2, 3], [1, 2], [1, 3], [2, 4], [3, 5], [4, 6], [5, 7], [18, 1], [18, 6], [18, 7], [18, 12], [18, 13]]}