Team D4: KUB Design Document

Kristina Banh, Umang Bhatt, Brian Davis
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
Email: {kbanh, umang, briandavis}@cmu.edu

I. Introduction

KUB is a personalized and portable dance trainer that's able to view your movements and offer corrections on a set of moves. Learning to dance is not only difficult, but also expensive and inconvenient as you typically have to commute to a dance studio in order to take class. This makes it harder for an aspiring or beginner dancer to start dancing, and harder for experienced dancers to continue dancing. KUB solves these problems by being accessible and being much less expensive than conventional teaching options.

Currently, there is an application that utilizes a similar technology and method as a trainer, but it's for working out as opposed to dance. There are also dance games that "teach" dance moves, but they offer no correction aspect and dont actually teach the movement very well since its purely for entertainment. This gives KUB an advantage edge as it is an application in a new field. KUB approaches dance correction using joint variance ranking, which automates the correction with a trainer feedback goal of 5 seconds after a movement is complete. This feedback should be relevant, correct, and useful to the user, with a 4 out of 5 average ranking goal from beginner to experienced dancers.

II. Technical Procedure

In this portion of the document, we will review the core design of the pose estimation pipeline. First, we will describe how we get the angular ranking Φ from a frame V. Then, we will describe how pick the angle ϕ_j to correct.

First let us introduce some notation. Let $\mathbf{V}$ be a video feed from a user. Let $V_1, \ldots, V_t \in \mathbf{V}$ be the t frames of the video feed. Without loss of generality, let us fix a frame V, which would be a still image; this allows us to avoid the temporal nature of pose correction to start.

A. Angle Estimation

For our purposes, we identified that mapping the pose into the angular domain will give us a much better representation of the user's pose characteristics than simply using the 2D points. It also allows us to bypass the issues of scale and translation that the Euclidean space suffers from. By identifying the angles between joints, we no longer have to require the user to be centered in the same way as our example data, and we are able to remove the issue of user height entirely. It also simplifies our problem very nicely without the loss of important information.

Given V, we identify the user's pose via Google PoseNet, $f(V) = P \in R^{12\times 2}$. PoseNet gives us one tuple $p_i = (x_i, y_i)$ for each joint i. See Figure 1 for details on all twelve joints, where the top left of the image is considered $(0, 0)$ and the bottom right is considered (w, h). Note that we are dealing with pose estimation measures in a 2D plane and that we are **not** retraining Google PoseNet; rather, we are using Google's pretrained weights for inference.

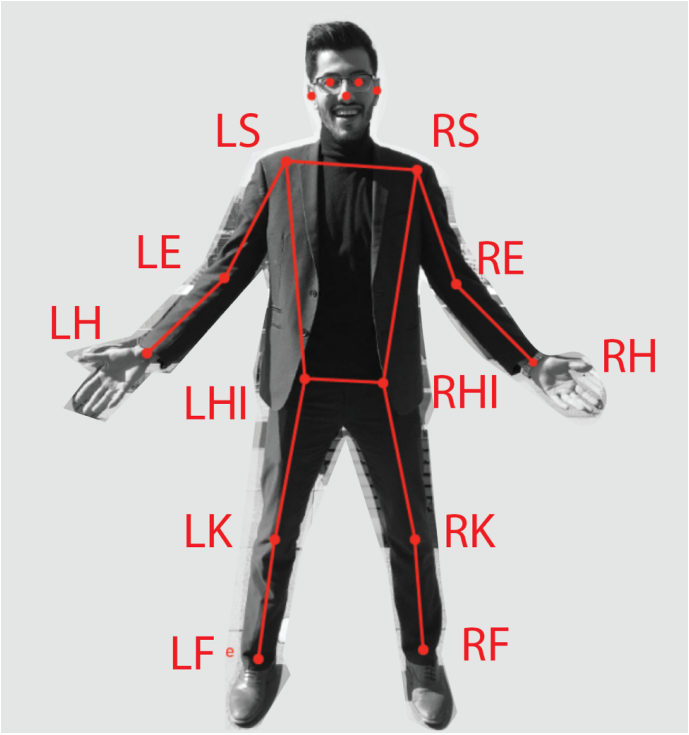

Fig. 1. Labeled Joints

We identify ten angles for our angular ranking $\Theta \in R^{10}$, as shown in Figure 2. In Table I, we show the three joint tuples $p_i, p_j, p_k \in P$ used to find all ten angles, $\theta_i \in \Theta$. The joints are written in the following format: Source (S), Pivot (P), and Target (T). If we assume a triangle between SPT, we want to identify the angle at P to be θ, as shown in Figure 5. Since we have three points in Euclidean space, we want to find the distance between S and P: D_{SP}, between P and T: D_{PT}, and between S and T D_{ST}. We can do this with any real valued distance metric, d. For example, we can let d be the ℓ_2 distance (which would be the norm of the displacement vector between two points in 2D), resulting in the following.

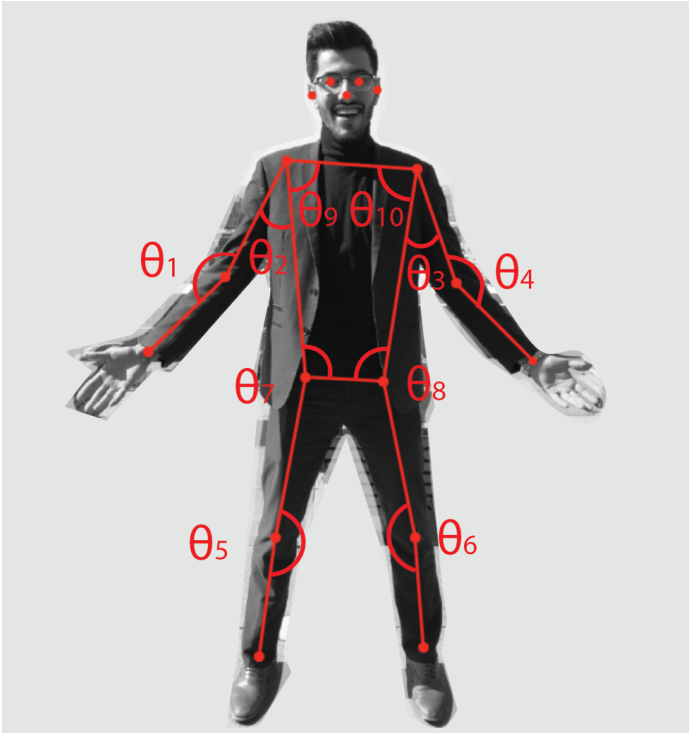

Fig. 2. Labeled Angles

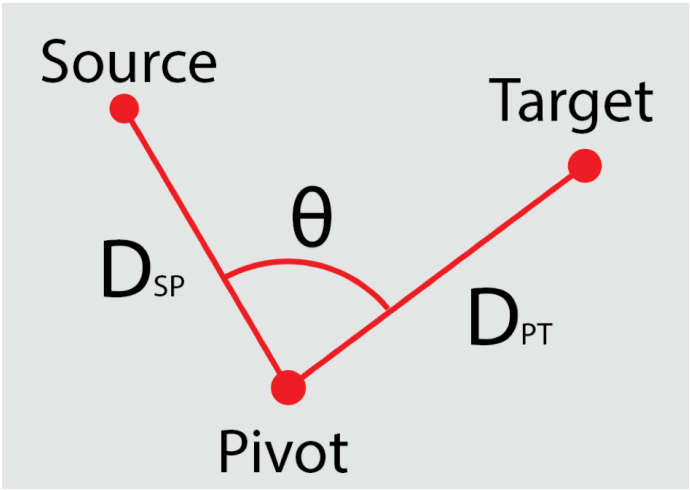

Fig. 3. Diagram of angle components

$$\begin{aligned} D_{\text{SP}} &= d(\text{S}, \text{P}) \\ &= \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} \\ &= \|S - P\|_2 \end{aligned}$$

We will also to try out the following distance metrics to see if the angles between certain joints are more sensitive when we use different distance metrics to get the side lengths of triangle SPT.

This would be formulation for the Chebyshev distance, which would give us the greatest difference along any dimensions (x or y).

$$\begin{aligned} D_{\text{SP}} &= d(\text{S}, \text{P}) \\ &= \|S - P\|_\infty \\ &= \max(|x_s - x_p|, |y_s - y_p|) \end{aligned}$$

This would be formulation for the Manhattan distance, which would give us the sum of the difference along all dimensions.

$$\begin{aligned} D_{\text{SP}} &= d(\text{S}, \text{P}) \\ &= \|S - P\|_1 \\ &= |x_s - x_p| + |y_s - y_p| \end{aligned}$$

We will use the same distance metric d to get all side lengths. Once we have D_{SP}, D_{PT}, and D_{ST}, we will solve the following to get θ, which follows from the Law of Cosines. We first present a general vectorized formulation, followed by a formulation specific to the Euclidean space.

$$\begin{aligned} \theta &= \arccos\left(\frac{\vec{D_{\text{SP}}} \cdot \vec{D_{\text{PT}}}}{\|\vec{D_{\text{SP}}}\|\|\vec{D_{\text{PT}}}\|}\right) \\ &= \arccos\left(\frac{D_{\text{SP}}^2 + D_{\text{PT}}^2 - D_{\text{ST}}^2}{2D_{\text{SP}}D_{\text{PT}}}\right) \end{aligned} \tag{1}$$

Angle (θ)	Source (S)	Pivot (P)	Target (T)
θ_1	LH	LE	LS
θ_2	LE	LS	LHI
θ_3	RE	RS	RHI
θ_4	RH	RE	RS
θ_5	LHI	LK	LF
θ_6	RHI	RK	RF
θ_7	LS	LHI	RHI
θ_8	LHI	RHI	RS
θ_9	RS	LS	LHI
θ_{10}	RHI	RS	LS

TABLE I
COMPREHENSIVE SOURCE, PIVOT, AND TARGET SPECIFICATIONS FOR EACH TRACKED ANGLE

After consulting with dancers, we learned that it would be difficult to ask a dancer to correct their left hip (θ_7, the angle between one's torso and hip line) separately from their right hip, or their right shoulder (θ_10, the interior angle between one's torso and shoulder) separately from their left. As such, we need to edit our angle vector to be the combination of both, Θ_{OLD}. We define the following two angles.

$$\theta_{7\prime} = \theta_7 - \theta_8$$

$$\theta_{8\prime} = \theta_9 - \theta_{10}$$

As such, we define an angle vector $\Phi \in R^8$, where we the first six elements of Φ and Θ are the same and then we set ϕ_7 to $\theta_{7\prime}$ and set ϕ_8 to $\theta_{8\prime}$.

B. Angle Correction

Once we identify Φ for the current frame V, we need to rank all eight angles by how much they vary from the expert ground truth, $\mathbf{E}$. Let $U_{a,1}, \ldots, U_{a,t} \in \mathbf{U_a}$ be video frames from expert a. Let $U_{a,t}$ be the same frame as frame V_i from the user. Using Equation 1, we can find the expert angle vector $\Phi_a \in R^8$.

Since any angle vector Φ is invariant to scale or translation along the 2D plane of the frame, we take an expectation of angle over all experts performing a given pose, irrespective of whether the angle vector estimates come from the same or multiple instructors. Let $\mathbf{U}$ be the set of k expert video feeds. We can get an aggregate expert angle vector, $\mathbf{E} = \Phi_{\text{agg}} \in R^8$, by taking the empirical mean of all angle vectors from the k experts.

$$\mathbf{E} = \Phi_{\text{agg}} = \frac{1}{k} \sum_{i=1}^{k} \Phi_k$$

Our goal is to find how much Φ differs from $\mathbf{E}$ and in what elements they differ: we will correct the dimension along which they differ most. To better identify which dimension of Φ to correct, we need to make stronger assumptions.

First, since we have k expert angle vectors, we actually have a distribution in the space of angle vectors. If we assume a normal distribution over the angle vectors, then we can assume the following:

$$\Phi_k \sim \mathcal{N}(\mu, \mathbf{\Sigma})$$

Thus, the mean vector $\mu \in R^8$ contains the mean value for all eight angles of interest in the expert distribution and the covariance matrix $\mathbf{\Sigma} \in R^{8\times 8}$ contains the variance for the eight angles of interest along the diagonal. Therefore, let $\rho = \text{diag}(\mathbf{\Sigma}) \in R^8$.

We can then rewrite each element of our user angle vector, Φ, in terms of the expert mean and expert standard deviation, effectively finding z-scores in the statistical sense.

$$\phi_i = \mu_i + \alpha_i(\sqrt{\rho_i})$$

$$\alpha_i = \frac{\phi_i - \mu_i}{\sqrt{\rho_i}}$$

We are left with eight scalar α values each representing how many standard deviations away from the expert a user lies. We then identify which user angle differs most from the expert distribution: this is the angle we want to correct.

$$\phi_{\text{fix}} = \phi_i \in \arg\max_i |\alpha_i|$$

In order to prompt the user to correct her behavior, we need to tell her which joint to correct and how to correct it. To start, we fill in the statement below, replacing [DIRECTION] with either "Increase" or "Decrease" and replacing [JOINT] with the joint name from Table II. To get the direction of the correction, we leverage the sign of the α value in question: positive alpha corresponds to "Decrease" and negative alpha corresponds to "Increase". This output is then sent to a Text-To-Speech engine to then play back to the user.

ANGLE (ϕ)	JOINT OF INTEREST	SUPPORTING JOINT
ϕ_1	RIGHT HAND	RIGHT ARM
ϕ_2	RIGHT ARM	TORSO
ϕ_3	LEFT ARM	TORSO
ϕ_4	LEFT HAND	LEFT ARM
ϕ_5	RIGHT LEG	RIGHT FOOT
ϕ_6	LEFT LEG	RIGHT FOOT
ϕ_7	HIPS	N/A
ϕ_8	SHOULDERS	N/A

TABLE II
ANGLE-JOINT NAME MAPPING

"[DIRECTION] the angle between your [JOINT] and your [SUPPORTING JOINT]"

If ϕ_7 or ϕ_8 need to be corrected, we will use the following verbiage.

"Level your [JOINT]"

In essence, we take advantage of the variance in the expert angle distribution to chose which joint to correct. We then leverage that variance ranking to correct the user's pose. Every time the user provides a frame V we correct at most one joint. To facilitate termination, we impose a lower bound on $|\alpha_i|$.

If $|\alpha_i| \leq \epsilon$ holds, then we do not consider correcting the i-th angle. We intend to set epsilon to 1: this implies that we will only correct movements to be within one standard deviation of the expert mean. Once all eight angles satisfy the aforementioned condition, we will show the following message.

"Great job, you've mastered this pose!"

C. Temporal Pose Estimation

Our stretch goal involves scaling from one frame V to a sequence of frames $V_1, \ldots, V_t$. In order to meet this goal, we have to make a handful of edits to the still image estimation described above. We need identify a frame matching procedure and then selection for a frame to correct. For now, let us assume that frame matching is done via the user interface (we will have a metronome and instructor video to ensure that the user aligns their movement with that of the instructor). For every user frame V_j, we calculate a set of eight α values using the procedure described above. Thereafter, we select the largest α_i value from all possible frames t.

$$\phi_{\text{fix}} = \phi_i \in \arg\max_{i\in[8], j\in[t]} |\alpha_{i,j}|$$

We repeat this process until all frames are within one standard deviation of the expert mean at that frame.

III. SYSTEM DESIGN

On a high level, our system has to accomplish a few things:

1) Capture a video feed of the user
2) Extract pose estimates from the user feed

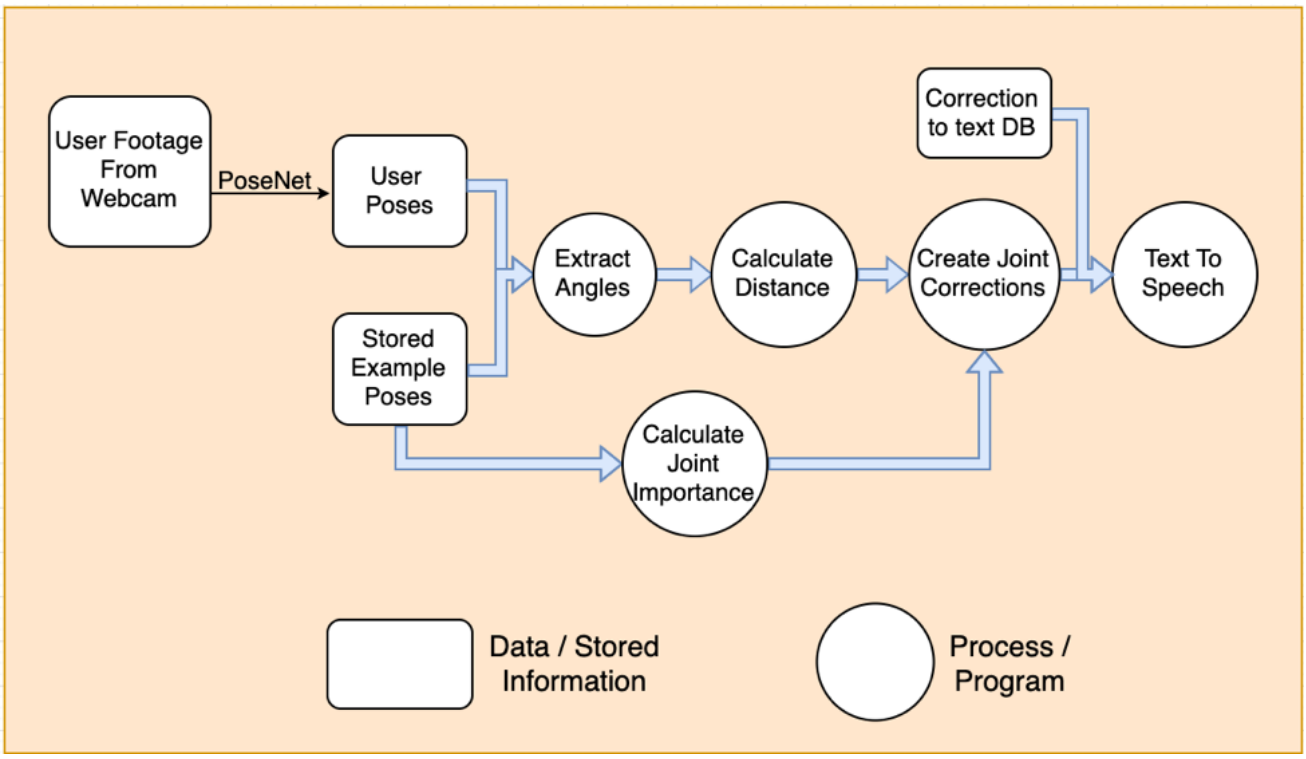

Fig. 4. Block diagram of system components

3) Calculate the distance between the user pose estimates and the stored examples for the given movement
4) Decide what movements need correction the most
5) Communicate the feedback to the user

The first requirement will be satisfied by the web camera in whatever device the user is using our application. For our purposes, we will mount the 1080p streaming camera listed in the Bill of Materials for better quality. However, the program will work with any stock web-cam.

The second step will be accomplished using PoseNet. It can be run locally on a CPU, though we will ultimately try to utilize our access to Titan X GPUs to speed up as well as increase the performance of the pose estimates.

The poses are extracted as a JSON object, which we will then pass to our distance calculator written in python. This function will extract the angles from the pose, and determine the difference between them and the corresponding frames stored in memory for the given movement. These differences will be ranked using the method discussed, and will return the highest ranked difference between the user movement and the instructor's.

This difference will contain information about the joints that need to be corrected, and will be passed to a python script that contains a dictionary that maps this difference to a text script that the user will be able to understand. Finally, this script will be passed to the text to the Mozilla text to speech engine and spoken aloud to the user. The script will also be displayed on screen for the user to review.

IV. User Interface Design

Let's imagine user Bob as he interacts with KUB Trainer. When he first opens the web application, he is welcomed and is asked to select which dance pose or movement he wants to learn. He decides to choose the first arabesque tendu pose and is then taken to a screen with a demonstration of the pose by KUB. Bob changes his mind, and decides to choose a different pose. He goes back to the initial selection screen, but decides to choose the same pose again. After watching KUB perform the pose he wants to learn, he's ready to do the pose. He chooses to ask the application to wait 5 seconds so he can back up from his web camera and get into the correct frame. Once the countdown ends, he tries to copy KUB's demonstration, which is still up on the screen next to his own camera feed that acts as both a mirror and a way to record his joint data. After 10 seconds, KUB gives him a correction. Bob wants to see KUB demonstrate once again to better understand his correction, so he goes back to the earlier screen where KUB performs his pose. He gets a better visualization of what his first arabesque tendu should look like, so he tries the pose again. This time, he gets a different correction and chooses to try another time, applying this new correction. After being satisfied with his performance on the current pose, he decides to try something new and goes back to the initial selection screen. Bob wants to try a dance movement instead of a pose this time, so he chooses a different movement than before. He is guided to a similar KUB demonstration, except this time KUB also has counts

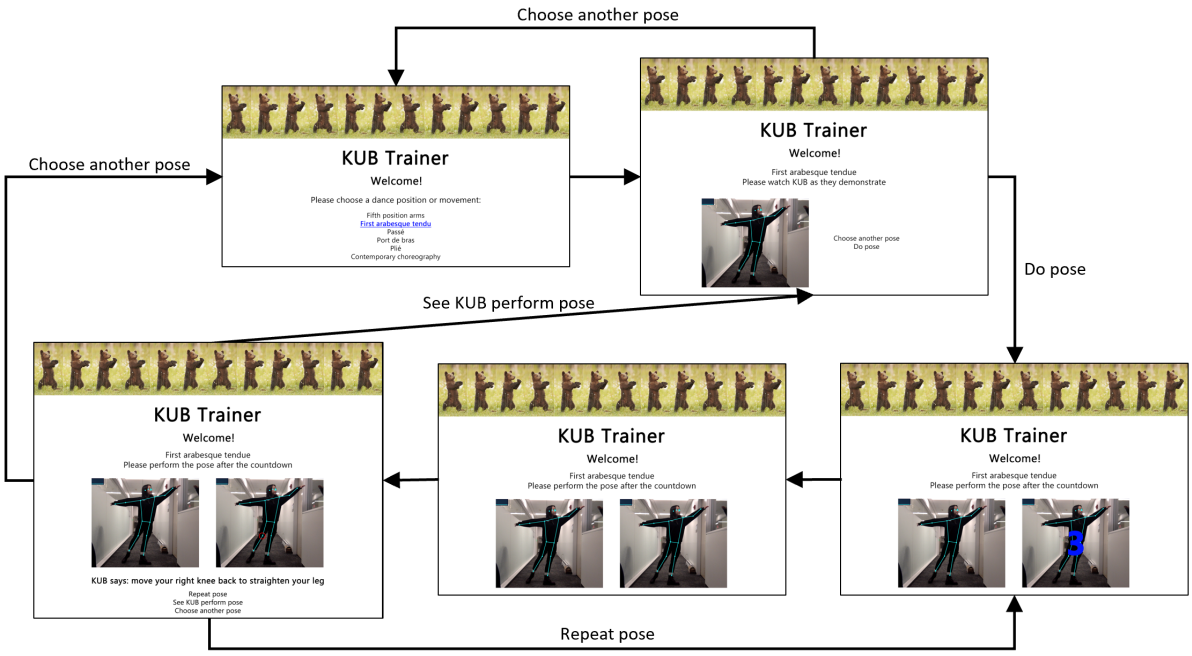

Fig. 5. User flow

associated with where Bob's joints should be at a certain time. After the same countdown as before to perform the movement, he performs the new movement with the counts he hears. KUB gives him a correction 10 seconds after he completes the movement. He continues using KUB Trainer to learn how to dance.

Our UI is designed to have as few moving parts as possible so that the user can delve into learning their movements without having to learn how to use the app first. The app consists of 3 main pages.

The landing page will be very minimalistic, and will simply show the list of moves that we currently have available. One we have different categories of moves, it will show the categories first, and will reveal the moves when the dropdown is clicked. Next to each move we will have an example image of what the instructor looks like while performing it.

Once a user selects a particular move, it will take them to the instruction page for that movement. Here, they will be able to repeatedly watch the instructor perform the movement. They will also be able to set the amount of time the program will wait for them to get ready to perform the movement themselves. The user may decide to return to the movement selection page at any point, allowing them to change their move.

Once the user is satisfied that they know what to do. They will select the "DO POSE" option that will take them to the performance page. Here, the program will count down the amount of time that the user specified it to wait. Then, it will announce the start of the move and record the user's performance. There will be a side by side split between the instructor performing the pose, and the user's video feed. We expect them to perform the pose at the same speed as the instructor, and the instructor video will help them align it perfectly.

Once recorded, the program will calculate what instructions need to be made, and show/announce the corrections that need to be made. Afterwards, the user can either choose to repeat the movement, watch the instruction video again, or return to the landing page to choose another movement.

V. Validation and Testing

Dancing in general is a very qualitative activity, and is hard to put hard quantitative measurements on correctness. Therefore, we will mostly rely on human feedback to test the correctness of our application. This is especially important since the app is entirely user-centric and will need to be tuned to ensure that it provides a suitable user experience.

There are three main metrics that need to be tested for this app to be successful. The first is for the feedback produced by the application to be useful, and faithful to what a trained dancer would also give. The second is for the app to provide a friendly user experience that is easy to navigate, and simple to follow at a glance. Lastly, we need to ensure that the app is reliable, and is able to perform consistently without error. By focusing on these three metrics, we hope to provide an application that is able to consistently provide feedback that is useful for a user to improve, in a way that is easy to interpret and navigate.

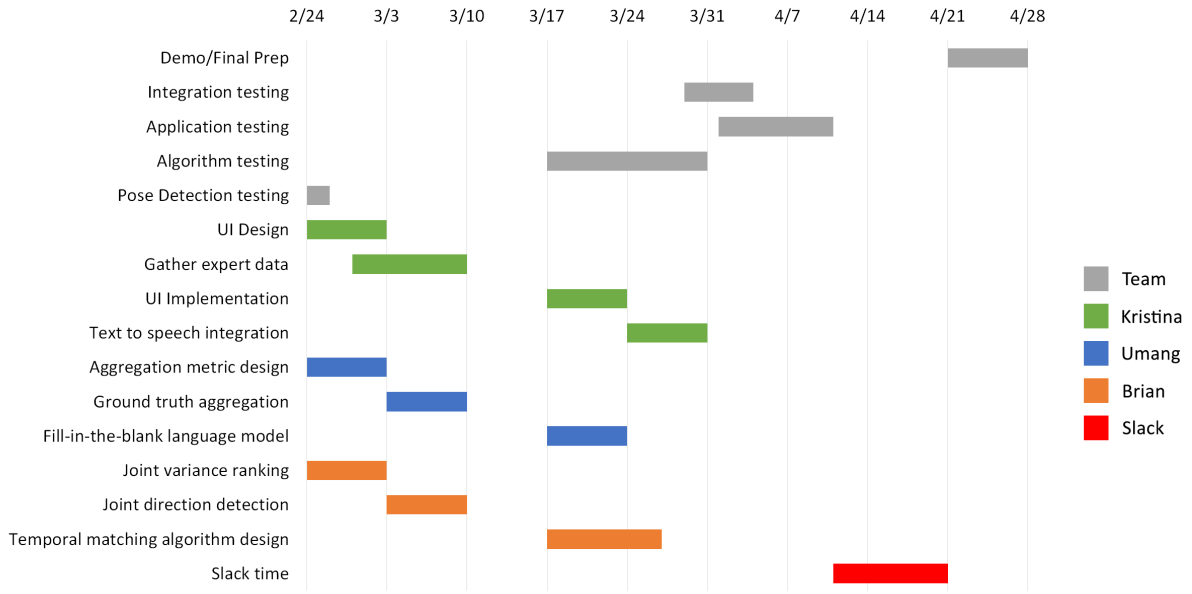

Fig. 6. Schedule

Item	Source	Cost
MacBook Pro 2018, 2.2 Ghz, i7, 16GB RAM, 256GB SSD	Personal Computer	N/A
Logitech HD Pro Webcam C920	Amazon	$67.37
AWS (for inference)	AWS	$150 from free credits
NVIDIA Titan X GPUs (for stretch goal)	AWS	N/A
Google PoseNet	https://github.com/tensorflow/tfjs-models/tree/master/posenet	
Python		
JavaScript		
HTML		
CSS		
Node.js		
Mozilla TTS		

Fig. 7. Bill of Materials

A. *Instruction Feedback*

Since the ultimate goal of this application is to fully replace the need for an instructor, it must be able to provide instructions that are as useful as those that a paid instructor would provide. For example, we would not want the app to be critiquing a subtle misalignment of the user's foot when there is a larger postural correction that needs to be made first. In other words, we need to be able to evaluate that the way in which the application ranks the importance of correcting each joint is consistent with the rankings of a trained dancer.

The only way that we see to accomplish this is through surveying real trained dancers. Luckily, since Kristina is a trained dancer, we will have her guidance during the execution phase to keep us grounded. For further feedback, we will poll 10 other trained dancers and have them rate the feedback in a few categories. The categories will be:

1) Feedback correctness
2) Feedback importance
3) Feedback clarity

Feedback correctness refers to the binary judgment of whether the feedback we provide is correct or not. An example of incorrect feedback would be if we tell the dancer to raise their left arm more when in fact they needed to lower it.

Feedback importance refers to the degree to which our

feedback will actually help the user. This is linked to whether or not we are able to properly rank how important each of the joints are to a move. Ideally, we would like to correctly diagnose the most important issue that the dancer needs to work on, while not mentioning the finer corrections until those have been worked on.

Feedback clarity refers to the ease at which the user is able to understand the given feedback. This score will reflect how clear the prose our app generates is, as well as the placement of our feedback within the interface.

We will poll the dancers and have them rate us in each of these 3 categories on a scale of 1 through 5. A successful mark will be an average of 4 in each of these categories.

B. UI Design and Ease of Use

We would like to minimize the complexity of the interface as much as we can so that a user is able to intuitively know how to operate our program without any guidance. In order to test this aspect of our app, we will be conducting user surveys by having people of all backgrounds test the application with no interjections from us. We will have the user go through the entire cycle of starting the application, selecting a dance, performing the movement, and receiving the feedback. We will then have the user rate us in the following categories:

1) General ease of use
2) Application Flow
3) UI design

The first and most important category is how easy the app is to use. This score will reflect the general attitude of the user towards the application. It will reflect whether the user had trouble navigating to the necessary objectives, and if they were confused during the process. A score in this category should be indicative of whether or not a user would use the app again.

The second category will assess the flow of the application. It will ask the user whether the links between different pages is logical and well thought out. If a user thinks that it takes too much effort to access a certain feature, or that there need to be fewer/more connections between important pages, it will be reflected in this category.

The last category will asses the design itself. It will include what the user thinks of the layout of each individual page, the placement of the camera/text, and whether the items are segmented intuitively.

Similar to the feedback score, we will poll the users and have them rate us in each of these 3 categories on a scale of 1 through 5. A successful mark will be an average of 4 in each of these categories.

C. Consistency

This aspect of validation is more technical, and will not rely entirely on user polls. Here, we will take a look at the rates at which our programs successfully perform in various metrics. While testing out applications of a similar scope, we realized that a major problem some of them had was that they would freeze in the middle of a movement and automatically restart, or improperly capture movements and get confused. We would like to minimize this confusion, and have developed 2 metrics to counteract it.

The first metric measures the program's ability to extract all of the joints from a given image. In running PoseNet solely on our CPUs, we noticed that in some frames joints were not being detected properly, resulting in incomplete pose estimates. A full pose it not necessary at every frame, as we can fill in the information from previous frames. However, having the joints makes the program more accurate. Therefore, we will aim to have a 95 percent joint capture rate by tuning the network parameters and utilizing GPUs.

The other metric relates to the overall speed of the performance. We would like to be consistent in getting feedback back to the user within 10 seconds. We agreed that 10 seconds was the maximum wait time we would want to wait as users. Therefore, for all moves, we would like to achieve a 90 percent feedback rate within 10 seconds of the completion of the move. If we can achieve these metrics, we can be sure that the app will run consistently enough to be convenient.

VI. Project Management

To divide up the work, Kristina is mainly in charge of the web application design and implementation, which includes getting user feedback to iterate on design, making sure all the UI elements work together, and integrating the algorithms and data with the front end. She is also responsible for gathering the expert dance trainer data, since she has the dance expertise.

Brian and Umang are working on the algorithm design and implementation, as well as the data pipelining. While they will mostly be working together, Umang will be focusing more on designing the aggregation metric and aggregation of the ground truth data that Kristina is gathering, as well as the language model that translates the raw joint correction into an understandable statement. Umang will be generating the aforementioned z-scores. Brian will focus on the joint-specific tasks of getting the joint variance ranking from those z-scores and detecting the direction of joint correction, the sign of the α value. He will also lead our stretch goal by designing the temporal variant of our procedure.

All team members will work together to test integration, the actual correction of joints, and the overall application. See Figure 6 for a breakdown of the schedule.

VII. Bill of Materials

KUB will be a web application using JavaScript, Node.js, HTML, and CSS. PoseNet, a TensorFlow model, will be used to detect and track movement of joints in real time and to save the information in JSON format. Algorithms and processing will be written in Python. A MacBook Pro 2018, 2.2 Ghz, i7, 16GB RAM, 256GB SSD will be used to run the web application and a Logitech HD Pro Webcam C920 will be used as the web camera. For a stretch goal of better accuracy and faster processing, two NVIDIA Titan X GPUs will also be used to run the application. Mozilla TTS will be used as the text-to-speech platform. See Figure 7 for specifics.