FitnessVR: Workout Assistant with Motion Detection and Machine Learning

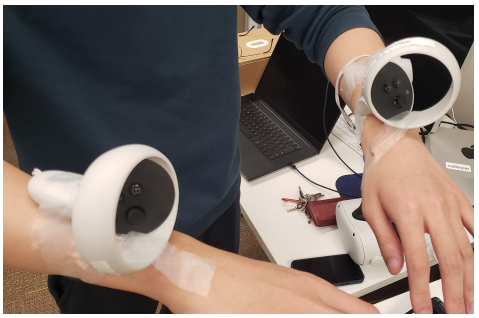
Introduction:

Contemporary mobile workout applications have many limitations, such as it requires inputs from users, users self-reporting the repetitions they do, many statistics are limited to cardio exercises, including time, pace, calories burnt, heart rate, and distance, devices utilize minimum sensors, merely serving as a workout diary and apps do not provide intuitive feedbacks to users.

In this research, we propose FitnessVR, a workout assistant with motion detection and machine learning, addressing the aforementioned problems. First, we demonstrate the automatic detection of set repetition without users having to self-report their progress. Second, we demonstrate how VR headset sensors could be utilized to detect users' movements and workout postures and extract meaningful information, such as which workout the user is currently doing, and to what extent the user is correctly doing the workout using machine learning, deep learning, and physical model. In this research, we thereby demonstrate the proof-of-concept that it is feasible to utilize sensors to provide intuitive feedback to users.

Methods:

Data Collection:



We collected data of ourselves (Kendrick, Kyu) doing jumping jacks and curls for 10 seconds using Oculus Quest 2. Each recorded 10 seconds of exercises, for 5 times each, using the Sensor Capture scene (reused from Lab 1). P1 corresponds to Kendrick, P2 corresponds to Kyu, STD to curls, SIT to jumping jack. All 36 sensor attributes of Oculus Quest 2 were collected. Thus, a total of 20 CSVs were obtained. The Oculus controllers were attached to each arm, at the same position and orientation to remove the confounding variables.

The collected data were cleaned and relabeled, and redistributed into a train and validation set. The data was cleaned to ensure that all data had only the first 10 seconds of the data. data_cleaning.py and create_validation_set.py were used for data cleaning, and to create the testing sets and validation sets. These sets were used to train the model and assess the accuracy of the model.

Set Detection:

To count the repetitions of bicep curls, we used statistical thresholds to determine whether a controller was moving. If the average of the mean of velocities on a controller in the last 20 frames was higher than a threshold, we considered there to be movement on the controller. We kept track of the number of times a controller went from moving to not moving. The bicep curl movement typically involves a pause in motion at the top and bottom of a repetition, so the number of repetitions was equal to the floor of the number of times a controller goes from moving to not moving divided by two. Repetitions were counted and displayed to the user through FitnessVR.cs.

Workout Detection:

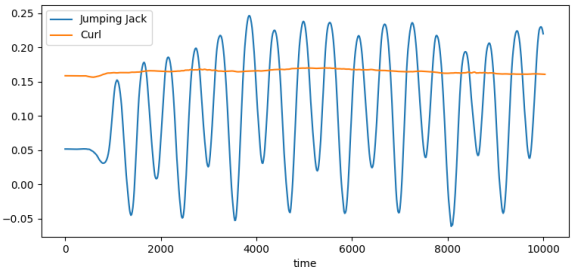
Method I: Decision Tree Machine Learning Model

We model the paper "Sensing Meetings Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application" for our ML model, and use mean, variation and peaks of significant features. We use a decision tree classifier because the paper clarified that mean, variation and peaks would yield promising results if used in decision trees. Because we are only distinguishing between two activities and our validation set is small (2 sets of curls and 3 sets of jumping jacks), we only need to consider the headset's y position as a significant feature. One of the goals of a fitness app would be to support as many workouts as

possible, so we also tested the feasibility of this model while considering all 36 of the VR headset and controllers' attributes. This model is created and evaluated in predict_shallow.py.

Method II: Long-Short Term Memory (LSTM) Deep Learning Model

Another method to accurately classify which workout the user is doing is to utilize the whole attributes and make the prediction model time-dependent. That is, rather than extracting time-independent features such as mean, variances or peaks, we consider that the attributes are time-dependent. For example, consider the attribute "headset_pos.y" for jumping jacks. It is obvious that as one jumps, one's "headset_pos.y" will also fluctuate up and down. However, the trend must be different for curls, where theoretically one's head position should be fixed and likewise "headset_pos.y". In such a case, while using time-independent statistics such as variance of "headset_pos.y" may be useful in distinguishing the two activities, we can improve the accuracy of the classification by incorporating all other 35 features, and their time-dependent trend that may well not be captured by time-independent statistics. Thus, we construct a time-series data using "headset_pos.y", which is then trained into a Tensorflow Keras LSTM classification model. The deep-learning model takes one's VR attributes of 10 seconds of workout activities, and would predict whether the 10 seconds workout activity is curl, or jumping jack.



Here, we used a total of 720 timesteps from the collected data. The timesteps were then distributed randomly into a training set and testing set in 0.8 ratio. The accuracy of the model in classifying the testing set was measured.

Method III: Inverse Kinematics Using Handle Sensor Data

Here, a method that solves the inverse kinematics of the human arm using the pseudo-Jacobian method is presented. In robotics, inverse kinematics is the process of determining the joint angles that are necessary to achieve a desired end-effector position and orientation. The pseudo-Jacobian method is a technique that uses the Jacobian matrix, which relates the velocities of the end-effector to the velocities of the joints, to solve the inverse kinematics problem.

In this program, the raw data from Quest 2 handles is used to determine the position and orientation of the end-effector, which in this case is the human wrist. The program then uses the pseudo-Jacobian method to calculate the joint angles that correspond to the desired end-effector position and orientation.

After obtaining the angles, we now have an accurate depiction of a human arm moving in 3D space, which can be compared with the desired or standard model for a certain exercise to generate ratings and suggestions in real time, making the VR equipment your personal trainer.

The starting point for the IK method is to make a realistic model of the human arm defined by joints, joint angle constraints, and lengths between each pair of joints. To do this, in IK_solver.py we adapted the model from "Closed-Form Inverse Kinematic Solution for Anthropomorphic Motion in Redundant Robot Arms" with L1 and L2 replaced by our group members' arm configuration.

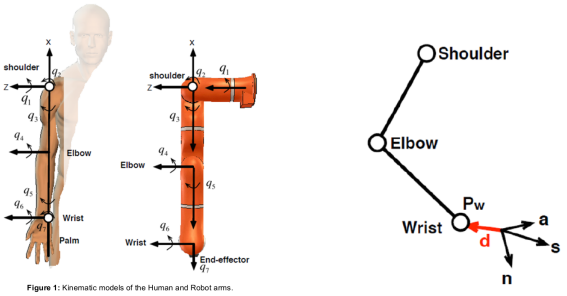


Figure 1: Kinematic models of the Human and Robot arms.

After the modeling process, we find the forward kinematics by the adjacent frame transformation method which is commonly used in robotics:

$$T_{orientation} = \begin{bmatrix} c_1 & c_4(s_1s_2s_3 - c_3s_1c_2) + s_4(s_1c_2s_3 + c_3s_1s_2) & c_4(s_1c_2s_3 + c_3s_1s_2) + s_4(s_1s_2s_3 - c_3s_1c_2) \\ s_1 & c_4(-c_1s_2s_3 - c_1c_2c_3) + s_4(-c_1c_2s_3 - c_3c_1s_2) & c_4(-c_1c_2s_3 - c_3c_1s_2) + s_4(-c_1s_2s_3 - c_1c_2c_3) \\ 0 & c_4(c_2s_3 + c_3s_2) + s_4(-s_2s_3 + c_3c_2) & c_4(-s_2s_3 + c_3c_2) + s_4(c_2s_3 + c_3s_2) \end{bmatrix}$$

$$T_{position} = \begin{bmatrix} L_3(s_1s_2s_3 - c_3s_1c_2) + L_4(c_4(s_1s_2s_3 - c_3s_1c_2) + s_4(s_1c_2s_3 + c_3s_1s_2)) - L_2s_1c_2 \\ L_3(-c_1s_2s_3 + c_3c_1c_2) + L_4(c_4(-c_1s_2s_3 + c_3c_1c_2) + s_4(-c_1c_2s_3 - c_3c_1s_2)) + L_2c_1s_2 \\ L_2s_2 + b + L_1 + L_3(c_2c_3 + c_3s_2) + L_4(c_4c_2c_3 + c_4c_3s_2 - s_4s_2s_3 + s_4c_3c_2) \end{bmatrix}$$

where L represents arm lengths, c represents cosine and s represents sine. Then, we solve for the inverse kinematics using the pseudo-Jacobian method:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = \hat{J} * \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \\ \dot{q_4} \end{bmatrix}$$

where $\hat{J}$ is the Jacobian defined as:

$$\hat{J} = \begin{bmatrix} \partial_{q_1}x & \partial_{q_2}x & \partial_{q_3}x & \partial_{q_4}x \\ \partial_{q_1}y & \partial_{q_2}y & \partial_{q_3}y & \partial_{q_4}y \\ \partial_{q_1}z & \partial_{q_2}z & \partial_{q_3}z & \partial_{q_4}z \\ \hat{\omega}_1^b & \hat{\omega}_2^b & \hat{\omega}_3^b & \hat{\omega}_4^b \end{bmatrix}$$

where $\hat{\omega}_i^b$ can be looked up from T_{0i} as all its information is already contained there.

Plug in the coordinates represented by the forward transformation matrix defined in section I, we can get the exact form of $\hat{J}$. By finding the peusdo-inverse of it, we get the formula for the inverse kinematics of the arm:

$$\overrightarrow{q_{target}} = \overrightarrow{q_{old}} + \hat{J}^+(\overrightarrow{X_{target}} - \overrightarrow{X_{old}})$$

After converting the formula and algorithm into Python, we now can use the inverse_kinematics_solver_Jacobian function to find the arm joint angles of a given pose. Then, we use preprocess.py to extract pose = [x, y, z, alpha, beta, gamma] which represents the 3D coordinates and Euler angles of the handle from raw data files (.csv in the same format as in Lab1). The program feeds a time series of poses into the IK solver, and now we know how exactly an arm moves around with angle = [shoulder 1, shoulder 2, elbow, wrist]. The program in evaluate.py compares each joint angle at a moment to the standard exercise and calculates the inverse of the sum of the difference as the rating.

Real-Time Workout Detection

We then constructed a backend server to support the real-time classification of workouts. The backend server is coded with python, in server.py. The server continuously communicates with Oculus Quest 2 headset (through FitnessVRDetector.cs), receiving the sensor data of 36 attributes. Then, the server runs the pre-constructed LSTM model presented above and predicts the type of workout. Then, it sends back the string to the Oculus Quest 2, which will update the text accordingly.

We have chosen the LSTM model for the classification model in the backend server to enable prediction every 0.5 seconds, and doesn't require the whole data set, but just data at a timestep. Because it is important to predict quickly for real-time prediction, the LSTM model that only requires a timestep data was chosen. The LSTM model supports time-series prediction, so theoretically its accuracy must not be affected even if we send the data of a single frame at different times.

The instructions for real-time workout detection, including Oculus set-up, is included in README.md at the root of the project repository.

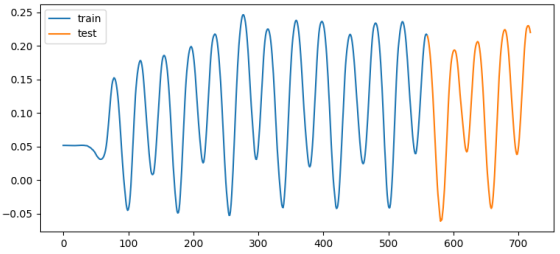
Workout Performance Evaluation System:

Method I: Long-Short Term Memory (LSTM) Deep Learning Model

Here, we leverage the time-series model again to quantify the correctness of the user's posture during the workout. We use the ML model to "forecast" what the "proper" posture would look like as the user works out, and compare it with the actual workout movement of the user. Because we are not professional athletes, we cannot comment on what is a "proper" posture, we here demonstrate the feasibility of the workout performance evaluation system by assuming that the data we trained is the data of "proper" posture.

The evaluation system we developed will output mean-squared error (MSE) as a measure to evaluate how well-formed one's workout is. Theoretically, if one can perfectly replicate the "proper" posture, then the MSE will be 0, increasing as the user deviates farther away from the "proper" posture.

We use scikit-learn's Random Forest Regressor to forecast what the "proper" workout movement would look like. While we could demonstrate the feasibility of the method for all 36 attributes for both curl and jumping jack, we only demonstrate evaluating the user's head position, "headset_pos.y" while they jump jack. We selected a sample file, assuming that the first 8 seconds consisted of the "proper" posture, and the last 2 seconds as the evaluation period, where the model will "evaluate" how correct the posture is during the last 2 seconds. This is demonstrated in time_series.py



Method II: Inverse Kinematics Using Handle Sensor Data

Using the same Jacobian method, we attempt to evaluate the correctness of the workout. In evaluate.py, we propose a rating mechanism based on how far apart each joint angle is at a given time to the "proper" posture, and we use the inverse of the accumulated errors as the rating. We also provide a visualization tool in visualize_arm.py.

Results:

Workout Detection:

Method I: Decision Tree Machine Learning Model

When considering only the headset's y position as a significant feature, our decision tree model achieved 100% accuracy and precision.

	precision	recall	f1-score	support
CUR	1.00	1.00	1.00	2
JUM	1.00	1.00	1.00	3
accuracy			1.00	5
macro avg	1.00	1.00	1.00	5
weighted avg	1.00	1.00	1.00	5

When considering all 36 of the VR headset and controllers' attributes as significant, we still achieved 100% accuracy and precision and the effect on time for training and classifying the validation set was negligible. This demonstrates that this model could be adapted to consider other significant features if more workouts are supported.

Method II: Long-Short Term Memory (LSTM) Deep Learning Model

The accuracy of the model's classification was measured, as well as the prediction's test loss, precision, recall, f1 score and confusion matrix. All these measures can describe the accuracy of the model.

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1417
1	0.99	0.98	0.99	1459
accuracy			0.99	2876
macro avg	0.99	0.99	0.99	2876
weighted avg	0.99	0.99	0.99	2876

First, we report the accuracy score of 0.99, and test loss of 0.05. We notice the increasing accuracy as the neural network is trained every epoch. The accuracy is high enough to conclude that the LSTM deep learning model could accurately classify whether the 10 seconds data is of curl or jumping jack.

We then report the precision, recall and f1 score in predicting the workout activity. In predicting curl, the model reported 0.98, 0.99, 0.99 in precision, recall and f1 score. In predicting jumping jacks, the model reported 0.99, 0.98, 0.99 f1 score. Overall, the model was effective in classifying both curl and jumping jacks.

Finally, we report the confusion matrix.

	True Class		
Predicted Class		Curl	Jump
	Curl	1403	14
	Jump	27	1432

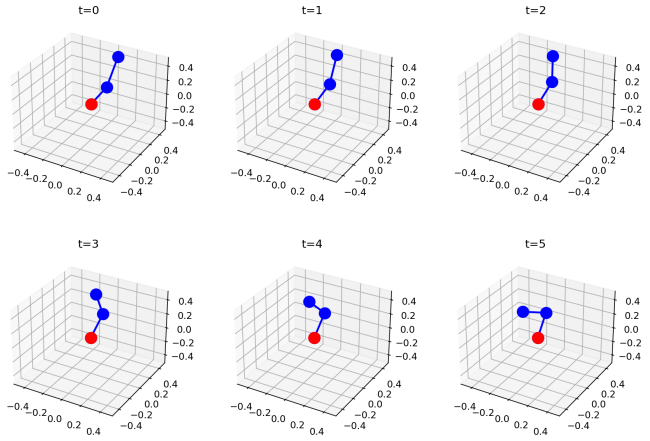
Overall, the LSTM deep-learning model is accurate in classifying the workout activity.

Method III: Inverse Kinematics Using Handle Sensor Data

Solved joint angles from raw data:

```
PS D:\code\mobile\mc-labs-digital-dominators\Python\FitnessVR\IK> python evaluate.py
    Angle: [0.025235546666666667, -0.01182420333333333, 0.008963777999999995, 3.5686415602970176]
    Angle: [0.02518234, -0.011723809999999998, 0.009117181999999996, 3.5887180826827088]
    Angle: [0.024965373333333332, -0.01248784333333333, 0.010025616999999995, 3.63949494661098]
    Angle: [0.66256513217192O3, -0.3555542337743628, -0.21792767650121, 4.620001721759621]
    Angle: [-0.010175120000000001, -0.01527756333333333, 0.013128174333333329, 5.950953157416206]
    Angle: [-0.00946236, -0.011584859999999999, 0.005645359999999996, 0.00936426215961498]
    Angle: [0.5713036871611066, -0.7240894773338803, 0.003443498735752836, 4.544586044780947]
```

Visualization of one cycle of lifting the weight (red dot representing the shoulder):



Rating:

```
PS D:\code\mobile\mc-labs-digital-dominators\Python\FitnessVR\IK> python evaluate.py
Rating for the exercise: 1.0491165731363872
```

Real-Time Set and Workout Detection:

We evaluate whether the real time set and workout detection are accurate. In this demo video https://www.youtube.com/watch?v=CEkmbaKuXTE, we demonstrated jumping jack and curls, and how FitnessVR detects it. Out of 14 jumping jacks, the proposed set detection method reported 0 repetition. Out of 7 curls, it reported 3 repetitions. Thus, the overall accuracy was 14%. Notably, our method for counting repetitions relies on counting the number of times a controller goes from moving to still. The jumping jack motion does not typically have pauses in motion, so our method was not effective for that exercise.

However, the real-time workout detection came out fairly reasonably. In the demo video, during the 10 seconds of jumping jack, the LSTM model accurately classified for 9.5 seconds. During the 14 seconds of curl, the LMTM model accurately classified for 13 seconds. Overall, the accuracy exceeded 90%. However, we do also admit that the accuracy highly differed for other people; another participant saw low accuracy, around 20% for workout detection.

We also note that the real-time detection result significantly differs from the offline detection method where we predict the pre-collected samples; while the LSTM model yielded in 99% accuracy in timestep classification, when the very same model is adopted to the real-time detection, the accuracy significantly drops.

Conclusion/Discussion:

We demonstrated the feasibility and proof of concept of a workout assistant VR application that uses VR sensor data to predict and correct the user's workout posture. We report high accuracy when ML and LSTM models predicted offline pre-collected data sets, but reports comparably lower accuracy when the LSTM model was adapted to the real-time measurement. However, we failed to present an accurate method to detect repeated sets of jumping jack and curls. We also demonstrated the IK method's feasibility in determining human arm poses and angles. It does not rely on statistical features over a period of time, such that it can respond instantly to the movement of handles, providing immediate suggestions to the user..

We also acknowledge some limitations of this research. First, the extremely small sample size would have constructed a very poor ML and LSTM model that could only be applicable to a small population and yield inaccurate results to the large population. Also, because we only collected 10 seconds of the workout, the time-series model would be extremely inaccurate for workouts that are done for longer than 10 seconds. We alleviate this problem by taking a modulo, but we admit this is not the best practice. Also, the set detection using statistical thresholds is too simple, as it ignores all the complex movement of different workout movements. While the IK method is efficient, we also acknowledge that it should be combined with other ML algorithms to provide complete functionality that includes auto-start, auto-synchronization, and more.

There are 3 ways in which we can make the Inverse Kinematics method more realistic, robust, and convenient: We can use the machine learning method developed in methods I and II to identify the start of each movement, and then synchronize it with the standard movement. In this way, we do not have to manually align the movements and do not have to tell the algorithm when to start or pause running. It makes the program easier to use and less energy-consuming; We can compare the movements in real-time and provide suggestions for the user to improve immediately. To do this, we can integrate the algorithm with the ML models into C sharp and install it onto the headset. This is not done due to time constraints; The algorithm relies on pre-defined arm configurations to run correctly. Instead, we can ask the user to do a few certain poses and use an algorithm to estimate the arm configs like lengths, angle constraints, and so on.

In conclusion, when these limitations and potential improvements are addressed, we believe that using ML and VR could be used to create a virtual fitness trainer. We expect the commercial version of it, using smartwatches instead of Quest 2 handles, to take over gyms around the world in the near future.

References:

1. Sensing Meetings Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application
2. Closed-Form Inverse Kinematic Solution for Anthropomorphic Motion in Redundant Robot Arms
3. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods

Contributions:

Report Writing - Kyu, Kendrick, Dingyi
Motion Capture - Kyu, Kendrick, Dingyi
Unity Scene - Kyu, Kendrick
Machine Learning Model - Kendrick
Deep Learning Model - Kyu
Inverse-Kinematics Algorithm - Dingyi
Real-Time Backend - Kyu, Kendrick