Fitness VR: Workout Assistant with Motion Detection and Machine Learning.

Kyu Park, Dingyi Sun, Kendrick Xie

Problem

- Contemporary mobile workout applications have many limitations:
  - Require 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
  - Apps do not provide intuitive feedbacks to users

Solution: Fitness VR

Problem 1. Require inputs from users, users self-reporting the repetitions they do

Solution 1. Fitness VR detects a repetition and automatically records it, and users do not have to self-report their progress

Problem 2. Many statistics are limited to cardio exercises, including time, pace, calories burnt, heart rate, and distance

Solution 2. Fitness VR targets anaerobic exercises that contemporary apps cannot cover

Problem 3. Devices utilize minimum sensors, merely serving as a workout diary

Solution 3. Fitness VR utilizes accelerometers to actively capture user's movements and workout postures

Problem 4. Apps do not provide intuitive feedbacks to users

Solution 4. Fitness VR aims to detect failed repetitions, providing suggestions to a user's form and weight usage, contributing to statistics tracking, form optimization and injury prevention

Demo - Curl, Jumping Jack

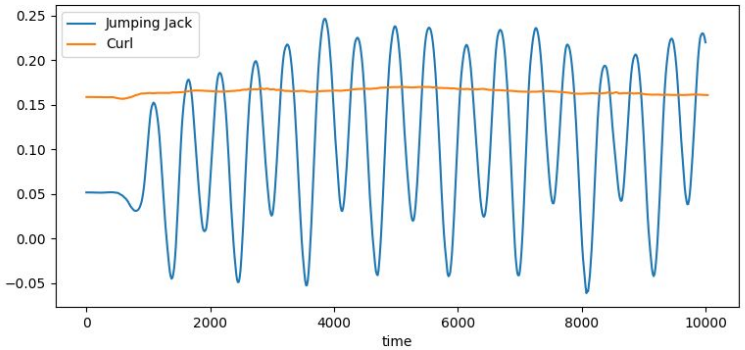
VR Controllers

Workout Detection Method I - Machine Learning (Decision Tree)

- Use mean, variation and peaks of significant features
  - Because we are only distinguishing between two activities and our validation set is small, we only need to consider the headset's y position to achieve 100% accuracy and precision
- The ultimate goal of a workout app would be to include as many workouts as possible
  - Tested decision tree while considering all 36 of the VR headset and controllers' attributes
  - Still 100% accuracy and precision and affect on time for training and classifying the validation set was negligible
  - May need to adjust considered features depending on what activities are supported

Workout Detection Method II - Deep Learning Model

- Often, simple ML on time-independent statistics (mean, variance, peaks, etc) cannot fully capture the time-dependent trends
- Using the entire time-series data could incorporate time-dependent trend in the classification
- Tensorflow's Long-Short Term Memory (LSTM) Deep Learning model was used to classify the activity type (Jumping Jack or Curl) based on the 36 sensor attributes values at a certain time
- Total 14400 time steps (720 time step for each sample, 20 samples), 80% used for training, 20% for testing
- Predicting each timestep



LSTM Deep Learning Model

- The model scored 0.99 accuracy, with its accuracy increasing as epoch (iteration of deep-learning) increasing
- The model successfully classify whether an activity is curl or jumping jack given 36 attributes at a timestep

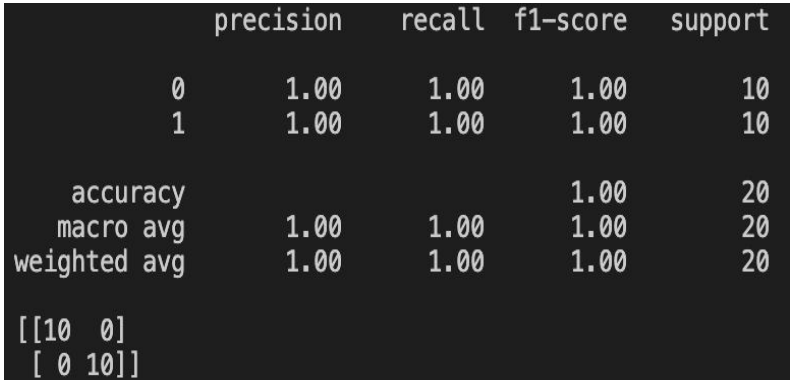
```
Epoch 1/10
360/360 [==============================] - 2s 2ms/step - loss: 0.4158 - accuracy: 0.8703 - val_loss: 0.1862 - val_accuracy: 0.9579
Epoch 2/10
360/360 [==============================] - 0s 809us/step - loss: 0.1229 - accuracy: 0.9686 - val_loss: 0.0939 - val_accuracy: 0.9788
Epoch 3/10
360/360 [==============================] - 0s 818us/step - loss: 0.0851 - accuracy: 0.9773 - val_loss: 0.1035 - val_accuracy: 0.9850
Epoch 4/10
360/360 [==============================] - 0s 804us/step - loss: 0.0760 - accuracy: 0.9817 - val_loss: 0.0607 - val_accuracy: 0.9854
Epoch 5/10
360/360 [==============================] - 0s 823us/step - loss: 0.0756 - accuracy: 0.9816 - val_loss: 0.0644 - val_accuracy: 0.9854
Epoch 6/10
360/360 [==============================] - 0s 794us/step - loss: 0.0665 - accuracy: 0.9823 - val_loss: 0.0577 - val_accuracy: 0.9857
Epoch 7/10
360/360 [==============================] - 0s 797us/step - loss: 0.0640 - accuracy: 0.9834 - val_loss: 0.0565 - val_accuracy: 0.9857
Epoch 8/10
360/360 [==============================] - 0s 810us/step - loss: 0.0608 - accuracy: 0.9833 - val_loss: 0.0516 - val_accuracy: 0.9857
Epoch 9/10
360/360 [==============================] - 0s 805us/step - loss: 0.0632 - accuracy: 0.9835 - val_loss: 0.0577 - val_accuracy: 0.9857
Epoch 10/10
360/360 [==============================] - 0s 807us/step - loss: 0.0596 - accuracy: 0.9834 - val_loss: 0.0525 - val_accuracy: 0.9857
90/90 [==============================] - 0s 383us/step - loss: 0.0525 - accuracy: 0.9857
Test accuracy: 0.9857441186904907
Test loss: 0.05247641727328305
```

	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

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

LSTM 720 Timesteps Combined

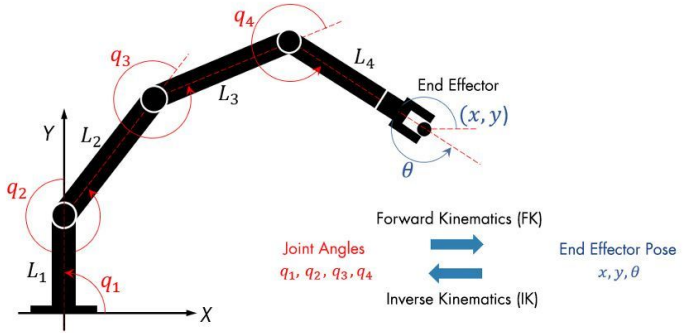
- “Mode” of the predicted classification could be used
- Each CSV contains 720 timesteps, so predict each timestep’s classification, calculate the most common classification (0 or 1) and then calculate the CSV’s workout activity
- Because each timestep was accurate, predicting 720 steps = 1 CSV also accurate, not surprising



	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	10
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

```
[[10  0]
 [ 0 10]]
```

Workout Detection Method III: ARM JOINT IDENTIFICATION: HC+IK



From "Closed-Form Inverse Kinematic Solution for Anthropomorphic Motion in Redundant Robot Arms"

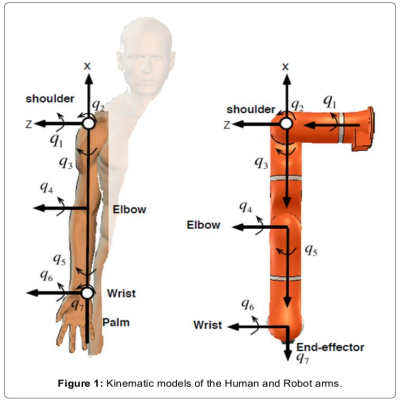


Figure 1: Kinematic models of the Human and Robot arms.

i	α_i	a_i	d_i	θ_i
1	90°	0	0	q_1
2	90°	0	0	q_2
3	90°	0	L_u	q_3
4	90°	0	0	q_4
5	90°	0	L_f	q_5
6	90°	0	0	q_6
7	0	0	L_h	q_7

The Model

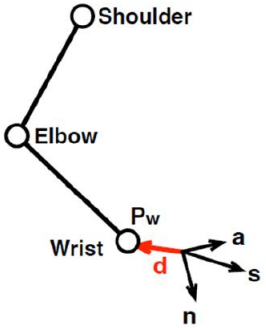


Figure 2: Wrist position with respect to the end-effector frame.

Solution for inverse kinematic problem

Let $P_e=[x_e \;\; y_e \;\; z_e]^T$, $P_w=[x_w \;\; y_w \;\; z_w]^T$ be the position of the elbow and wrist with respect to the base frame. The position of the elbow is computed by (4), with the estimation of the swivel angle, using (11). The position of the wrist is computed using (3). Let the desired position and orientation of the end- effector be given by

$$T=\begin{bmatrix} n_d & s_d & a_d & p_d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Since the position of the wrist and elbow are computed, then the solution for the first 4 joint angles has a

closed form which is shown below:

$$q_1=\arctan 2(y_e,x_e) \tag{13}$$

$$q_2=\arctan 2\left(\sqrt{(x_e)^2+(y_e)^2},z_e\right) \tag{14}$$

$$q_3=\arctan 2(-M_3,M_1) \tag{15}$$

$$q_4=\arctan 2\left(\sqrt{(M_1)^2+(M_3)^2},M_2-L_1\right) \tag{16}$$

Where

$$M_1=x_w\cos(q_1)\cos(q_2)+y_w\sin(q_1)\cos(q_2)z_w\sin(q_2) \tag{17}$$

$$M_2=x_w\cos(q_1)\sin(q_2)y_w\sin(q_1)\sin(q_2)z_w\cos(q_2) \tag{18}$$

$$M_3=x_w\sin(q_1)+y_w\cos(q_2) \tag{19}$$

For $q_5\,\varepsilon(0,\pi)$

$$q_5=\arctan 2(-a_y^{(4)},-a_x^{(4)}) \tag{24}$$

$$q_6=\arctan 2\left(-\sqrt{(a_y^{(4)})^2+(a_x^{(4)})^2},a_z^{(4)}\right) \tag{25}$$

$$q_7=\arctan 2(-n_z^{(4)},s_z^{(4)}) \tag{26}$$

For $q_5\,\varepsilon(-\pi,0)$

Current Progress

```python
def inverse_kinematics_solver_Jacobian(max_trial, curr_angle, curr_pose, target_pose):
    # preprocess data and transform to the correct frame
    start_pose = curr_pose
    target_pose = target_pose
    [x_t, y_t, z_t, alpha_t, beta_t, gamma_t] = target_pose

    # enter the numerical search loop
    count = 0
    prev_pose_abs_diff = 1
    curr_pose = start_pose

    while(1):
        [q1, q2, q3, q4] = curr_angle
        [x, y, z, alpha, beta, gamma] = curr_pose

        # find the difference to target pose
        pose_abs_diff = (x_t-x)**2 + (y_t-y)**2 + (z_t-z)**2
        pose_diff = 0.01 * np.matrix([x_t-x, y_t-y, z_t-z]).transpose()

        # stay in loop if the difference between current and target pose keeps decreasing
        if (pose_abs_diff > prev_pose_abs_diff):
            break
        else:
            prev_pose_abs_diff = pose_abs_diff

        # compute pseudo inverse of J based on current joint angles
        J = find_Jacobian(curr_angle)
        J_inv = np.linalg.pinv(J)

        # calculate intermediate angles and pose
        ang_diff = np.matmul(J_inv, pose_diff)
        q1 -= ang_diff[0,0]
        q2 += ang_diff[1,0]
        q3 += ang_diff[2,0]

        # update intermediate angles and pose
        curr_angle = [q1, q2, q3, gamma_t]
        curr_pose = foward_kinematics_solver_Jacobian(curr_angle) + [0,0,gamma_t]
        count += 1

        # break if can't find solution in max number of trials
        if (count == max_trial):
            return None

    print("Solution found after "+ str(count) + " iterations")
    print("pose_diff is: " + str(pose_abs_diff))
    print("Angles: " + str(curr_angle))
    return curr_angle
```

```
[ ] pose = inverse_kinematics_solver_Jacobian(100000, curr_angle, curr_pose, tp1)

Solution found after 3190 iterations
pose_diff is: 2.7425675742311726e-29
Angles: [0.09999999999999538, 0.15000000000004501, 0.49999999999995026, 0.15]
```

```
[ ] pose = inverse_kinematics_solver_Jacobian(100000, curr_angle, curr_pose, tp2)

Solution found after 2975 iterations
pose_diff is: 1.79303235228101e-29
Angles: [-0.06999999999999747, 0.15000000000001235, -0.2999999999999948, 0.15]
```

```
[ ] pose = inverse_kinematics_solver_Jacobian(100000, curr_angle, curr_pose, tp3)

Solution found after 3246 iterations
pose_diff is: 2.5919935563541175e-29
Angles: [0.16999999999999713, 0.05000000000004092, 0.11999999999995896, 0.15]
```

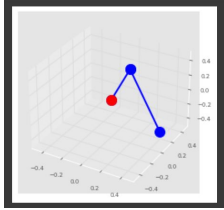
Todo List for the next week

1. Implement rating function for a given set of exercise (based on direct comparison to that recorded from professional bodybuilder .)
2. Visualization of arm pose in 3D space (using matplotlib). Partially done.

Try my best to do but not guaranteed to finish on time:

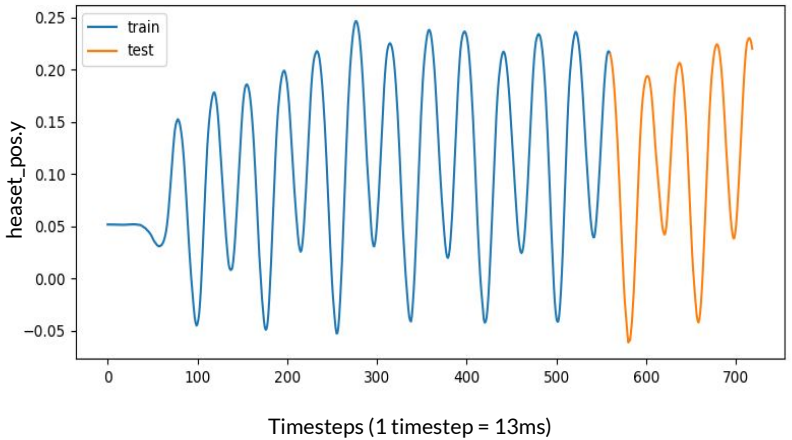
Translate the algorithm into C# to enable real-time assessment on Quest 2

```
# Visualize the arm in 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Define the positions of the joints
joint1_pos = shoulder_pos
joint2_pos = shoulder_pos + np.array([0, L1*np.cos(theta1), L1*np.sin(theta1)])
joint3_pos = joint2_pos
joint4_pos = end_effector_pos - np.array([0, 0, L2])
joints_pos = np.array([joint1_pos, joint2_pos, joint3_pos, joint4_pos])

# Plot the arm
ax.plot(joints_pos[:,0], joints_pos[:,1], joints_pos[:,2], 'bo-', linewidth=2, markersize=12)
ax.plot(joint1_pos[0], joint1_pos[1], joint1_pos[2], 'ro', markersize=12)

ax.set_xlim([-0.5, 0.5])
ax.set_ylim([-0.5, 0.5])
ax.set_zlim([-0.5, 0.5])
plt.show()
```



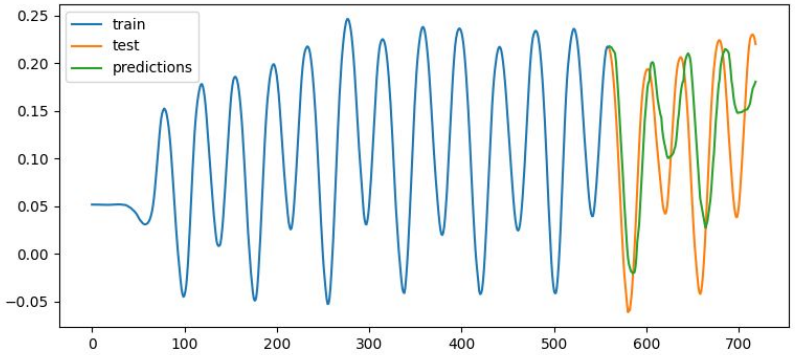
Workout Performance Evaluation System

- The goal is to demonstrate the feasibility of quantifying how “proper” one’s workout posture is
- We do not define what a “proper” posture is, but rather demonstrate how our evaluation system can quantify the difference between the “proper” posture and the user’s posture
- We use time-series ML forecast model to “forecast” what the “proper” posture is
- 100 previous timesteps (1.3 seconds) used for prediction
- Assume that the first 8 seconds (train) is the “proper” posture; we calculate how much the user deviates from the “proper” posture in the last 2 seconds (test)



Workout Performance Evaluation System

- We use mean-standard error (MSE) to calculate to what extent user deviates from the “proper” form
- Perfect workout where user matches with the “proper” form will result in 0 MSE, while the farther they deviate, the higher MSE is



- We leave the question “so what is the proper posture for curl and jumping jack?” to the professional athletes

Test error (mse):
0.004456442493572125

Realtime ML Workaround:

- Establish TCP connection between Oculus application and computer
- On Oculus application, convert the variable we want to send to a string and send it to the computer as a byte array
- On computer, convert the byte array back to a string and convert it to our desired type for ML in Python
- Viability:
    - We have successfully sent a C# variable from a Unity application to a computer running a Python script to list for incoming data
    - TODO: convert our received Python variable into a dataframe usable by our ML models and send back activity found

Conclusion/Discussion

- We present the functionality of workout set detection methods
- We have presented three different methods in detecting the workout
  - Limitation: Small sample number
- We have presented a feasibility of the workout-evaluation method using MSE
  - Feeding “proper” postures into the time-series prediction model can then assess the posture of the user
  - “Proper” posture must be predefined