Robot-Guided Exercise:

The Role of The Human Model

by

Selena Richards

Advisor: Prof. M. Begum

Spring 2022

Contents

Abstract

Physical therapy after an injury can be difficult for patients to access, whether it be due to location, finances, or other factors. To make physical therapy more accessible, robot-guided exercise training can be used. Commercial anthropomorphic robots have been created and have human-like movement, but may still lack qualities that make it easy for a human to understand its intentions intuitively. A human model that mimics the movement of a robot performing human exercises can be used to complement the robot and increase human understanding. The ideal model would be viewable from different angles and could visibly show the difference between a patient's attempt at the exercise and the robot's demonstration of the same movement. This work will discuss the process of implementing this model, the advantages of pairing it with a robot, and how it can help with future robot-guided exercise training.

Introduction

Although physical therapy would benefit many people who have experienced injury, some people do not use this service. This may be due to a variety of factors, most notably accessibility. Proper physical therapy may take multiple weeks, the financial cost of which adds up quickly. Some injuries require therapy multiple times a week, which may not fit in the schedules of work and personal lives. Others may forgo therapy due to their physical location or due to a lack of transportation. With the help of robot-guided exercise training, physical therapy can become more accessible.

In robotics, the legibility of a robot's motion is determined by how expressive a motion is in its intent.[1] When a robot moves a certain way, a human observer should be able to intuitively know where the robot or its moving limb is going. A previous study performed by Zhao et. al used the arm of a Baxter robot and had human participants try to determine what object its arm intended to pick up based on how it moved.[2] In robot-guided exercise, people must be able to understand the intentions of a robot's movement.

A robot that is capable of making human-like movements is critical to legibility. For this particular purpose, YuMi is an ideal candidate. It is one of the most anatomically accurate robots in regard to the upper body of a human on the commercial market. With seven joints on each arm, it is able to create a large range of human motions, including most arm exercises that could be performed during physical therapy.[3]

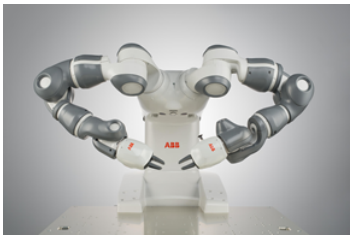

Figure 1. An image of the YuMi robot.[3]

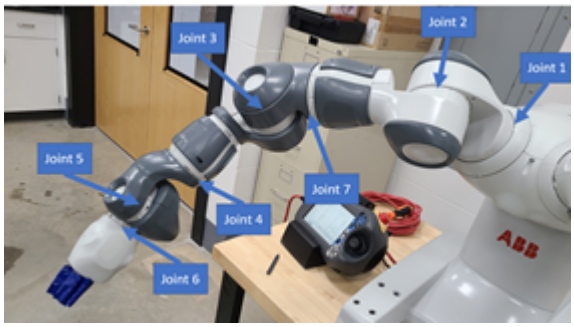

Figure 2. The seven joints that make up the YuMi arm.

Although YuMi's arm has a very similar anatomy to a human arm, it still has slight differences that hinder its legibility. The motion of its upper arm, for example is controlled by three separate joints (Joints 1, 2, 7 in Figure 2), as opposed to the single ball joint in a human shoulder. While they can perform similar motions, this difference may confuse patients who are looking to copy its movement. A human model that has a stronger resemblance and structure to an actual human can help alleviate this problem. Since humans can intuitively understand each other's movements and intentions[2], the model has the potential to have stronger legibility than YuMi.

The human model also has the advantage of being viewable from different angles with the press of a button. When looking at the model from a specific perspective, it may be harder to see how a limb is angled. Offering different camera angles assists patients in understanding the intention of the model and its movement. In addition, the model has colored arrows that display the patient's movement, which allows the user to compare their motion to YuMi's. This provides a visual for patients if they need to adjust how much they rotate a joint in order to complete an exercise correctly.

Prior studies and programs have been created to map human motions to a model, both for research purposes and the entertainment industry (i.e. motion capture for games and movies, virtual reality)[4]. Research mapping human motion to YuMi has also been done and is pivotal to this project. There is limited research on mapping YuMi to a human model, however. This project offers a starting point that can be later used to assist patients in robot-guided exercise training.

Implementation

This section of the work will discuss the methods and implementation of the project, including the software used and overall workflow.

ROS

ROS, or the Robot Operating System, is open-source software that contains a variety of tools that are used to send and receive data from robots, both in real life and simulation. It is made of a framework of processes (nodes), which are vital to controlling the robot and its data.[5] ROS is supported by Python, C++, and Lisp, but there are experimental versions in other languages as well. This particular project mostly utilizes Python, but also uses some ROS in MATLAB and C#. This project takes advantage of a few specific ROS concepts: packages, messages, topics, and bags.[6]

• Packages: Units that organize the ROS software, hold nodes, libraries, datasets, configurations, etc.

• Messages: Define a data structure for what is sent over the ROS network. For example, this project uses Float32MultiArray and Pose messages for each joint.

• Topics: A labeled channel for sending and receiving messages of a specific type. Topics have publishers and subscribers. The publisher posts data to a specific topic and the subscriber listens. The project has topics that listen for Float32MultiArray messages and topics that publish Pose data for each joint.

• Bags: A bag is used to store data from topics over a certain period. This is useful as it allows data to be reused rather than needing to move the robot or simulation each time a feature is tested.

The project uses ROS Noetic, the latest distribution at the time of this writing.

Unity

Unity is a real-time development platform that utilizes 3D models.[7] This is used for gaming, animation, engineering, and several other purposes. Its compatibility with ROS using the rosbridge makes it an ideal candidate for this project, as it allows models to listen directly to ROS topics. Models are able to be transformed in real-time, including rotations, translations, and scales. There are many free 3D models available on the web as well as software that can be used to create human models (i.e. MakeHuman)[8]. Unity also has capabilities for different cameras, lights, and UI, all of which are used for this project.

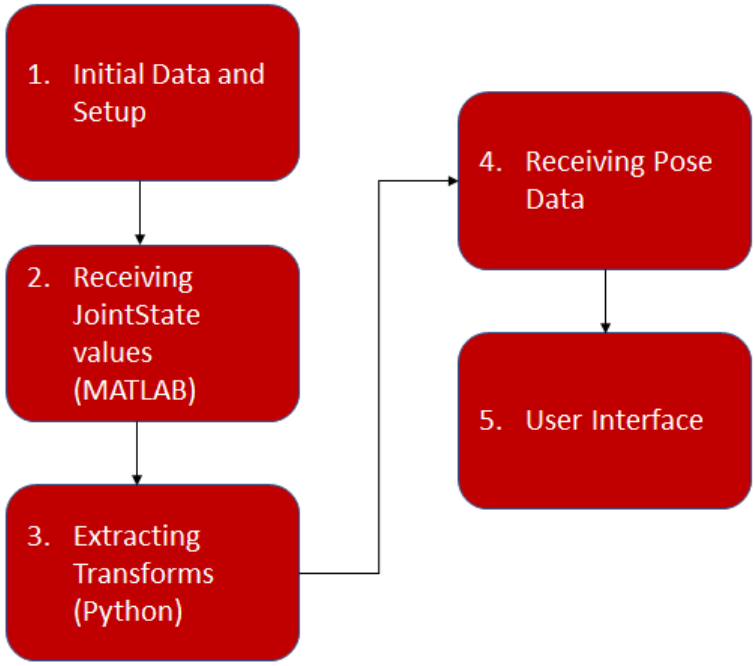

Figure 3. A block diagram showing the general process of the project.

1. Initial Data and Setup

This section will cover how the data was collected and how the system is set up for communication between ROS and Unity to make the human model move.

Before the human model can be animated, data must be collected from YuMi performing human exercises. This study focused on the three main exercises: the front arm rise, sidearm raise, and shoulder press.

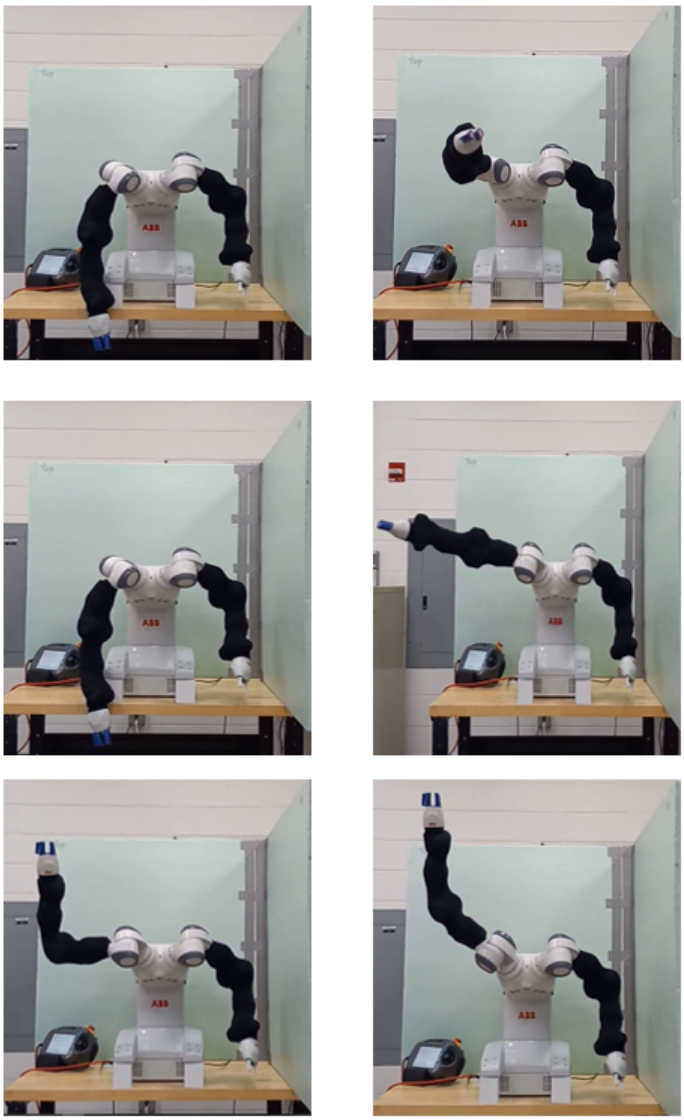

Figure 4. YuMi at the beginning and apex of the front raise (top), side raise (middle), and shoulder press (bottom).

The data for these exercises was recorded during a previous study where a human was connected to sensors that measured the motion of each joint. These measurements were then mapped to YuMi, allowing it to perform the motions of each exercise. The data was stored in a rosbag in the form of JointState data.

2. Receiving JointState Values (MATLAB)

To listen to this data, the rosbag is replayed and JointState data is republished via ros-topics. A MATLAB script subscribes to each of the robot's joints and uses a pre-written function to get the information as a Float32MultiArray, which will later be used as a transformation matrix. This is done for each of YuMi's joints. A publisher is created for each joint, which publishes the collected array to its assigned topic.

3. Extracting Transforms (Python)

After the transforms are collected via MATLAB, the data must be put in a format that can be used by the human model. Since the Float32MultiArray contains data for a transformation matrix, it can easily be reshaped into a matrix, then Pose data. A ROS Pose is a datatype that contains an object's orientation and position, the orientation being stored as a quaternion and the position being stored as a Vector3 (xyz).

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & pos_x \\ R_{21} & R_{22} & R_{23} & pos_y \\ R_{31} & R_{32} & R_{33} & pos_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 1. An example of a transformation matrix, where R components represent a portion of the 3x3 rotation matrix and pos components express a position in an xyz coordinate system.

Differing Coordinate Systems

To get the proper joint rotations, the coordinate systems of YuMi and Unity must be considered. The figures below highlight this difference.

Figure 6. YuMi has a Z-up world, where positive Y points to YuMi's left.

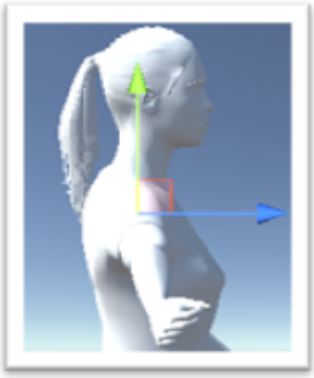

Figure 7. Unity has a Y-up world, where positive X points to the model's left and positive Z points forward.

To get YuMi's data in the correct coordinate system, the following transformation may be performed:

$$M_{YuMi->Unity} = R_{YuMi} * R_x(90) * R_y(90)$$

Where,

- $M_{YuMi->Unity}$ is YuMi's coordinates in Unity's coordinate system
- R_{YuMi} is the rotation matrix received from YuMi
- $R_{x(90)}$ is a 90-degree rotation around the x-axis
- $R_{y(90)}$ is a 90-degree rotation around the y-axis

After the overall coordinate systems have been aligned, the starting positions of each joint must also be accounted for. In YuMi's starting position, some joints already have preset rotations, such as a Joint 3. The human model does not have these. To adjust for this, the

pre-rotation must be accounted for by taking the inverse and multiplying it by the current rotation. For example,

$$M = P^{-1} * R_{YuMi}$$

Where,
- P^{-1} is the inverse of YuMi's rotation in the home position,
- R_{YuMi} is the rotation matrix received from YuMi,
- M is the resulting matrix.

Mapping From YuMi to Human Model

For each YuMi joint, a subscriber is created that listens for Float32MultiArray data. How this data is mapped to the human model is dependent upon the joint.

Mapping the Shoulder Joint

The shoulder joint on the human model works similarly to a ball joint on a real person, which can move in various planes. On YuMi, however, the shoulder joint is controlled by Joint 1 and Joint 2 (see Figure 2), which can only rotate around one axis. To account for this, the following rotation is performed:

$$M = J_2 * J_1$$

Where,
- J_2 is the rotation matrix received from Joint 2
- J_1 is the rotation matrix received from Joint 1
- M is the resulting matrix.

After M is calculated, it is transformed into a quaternion using the as_quat() function from the scipy library. This quaternion as well as the position are made into a Pose object, which is then published to the shoulder joint Pose topic.

Mapping the Elbow Joint

The elbow joint on the human model is a hinge joint similar to YuMi's, so it does not need to be combined with another joint in the same way that the shoulder does. The rotation matrix received from YuMi is multiplied by the inverse of YuMi's elbow's starting position before being transformed into a quaternion. This quaternion is paired with the position

data received from the transformation matrix, which is then published to a Pose topic corresponding to the elbow.

4. Rosbridge and Receiving Pose Data

To establish communication between the human model and ROS, the rosbridge library is used.[9] This is an open-source project that allows ROS topics to be subscribed to by individual GameObjects in Unity. To use this, the library must be installed and Unity's WebSocket IP address must match that of the system. Once the rosbridge server is launched, ROS messages can then be communicated between ROS and Unity.[10]

ROS Connector Script

The main script that allows communication between ROS and Unity is the ROS Connector, which is utilized by the rosbridge. This project follows the Microsoft Serializer and Websocket Sharp Protocol.

On the ROS Connector, a controller script is added for each joint on the arm of the human model. To mimic the three exercises performed by YuMi, the project focuses on the right arm, specifically the shoulder and elbow, referred to as "upper arm" and "lower arm" by Unity.

The shoulder controller script contains behavior that subscribes to ROS Pose data. It subscribes to the "/upper_arm_r" topic, which is published by the Python script in Section 3. Upon starting Unity, the model's position and rotation are set according to their local coordinates, which are based on the rotation and location of the clavicle GameObject. The joint is updated every frame by receiving ROS messages containing Pose data, which is used to rotate the upper arm GameObject.

The elbow controller script is similar to the shoulder controller script and controls the lower arm GameObject. For both scripts, the x, y, and z positions of the received quaternions needed to be adjusted due to Unity's processing of the quaternion order.

5. User Interface and Patient Visuals

To improve legibility, the human model is visible from five different angles: front view, left view, right view, back view, and top view. This is implemented using Unity Cameras,

which are translated and rotated to specific positions that capture the human model's view. A set of UI buttons in the corner of the screen allow users to toggle between the different cameras. The cameras and buttons are operated through a controller script, which sets the correct camera active when a button is clicked.

Figure 8. The human model as seen from various angles.

Representing Patient Movement

When a patient performs an exercise, they may perform it incorrectly. Therefore it would be helpful for him or her to have a visual representation of their motion compared to YuMi's. To show patient movement, there are two possible implementations. The first implementation involves showing two human models: one that demonstrates YuMi's movement, and another that demonstrates the patient's movement. For the purposes of demonstration, patient data is artificially generated by slightly altering the value of z within the quaternion. The UI shows YuMi's model on the top screen and the patient's model on the bottom screen, with the difference in Euler angles in the top right corner. Clicking the camera button changes the camera angle for both models.

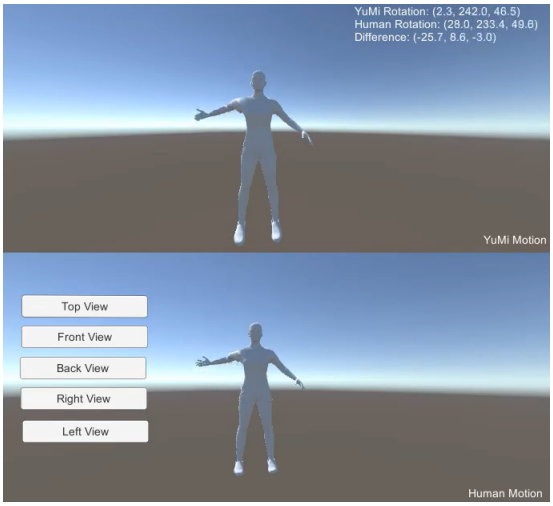

Figure 9. The implementation with two models: one to represent YuMi's movement (top) and one to represent the patient's movement (bottom).

The second implementation includes a GameObject containing three arrows that represent the patient's movement. This can be implemented in a similar way to how YuMi's joints are sent to ROS, with each joint corresponding to a GameObject on the human model. An Axis object containing arrows that represent the x, y, and z axes is currently used as a visual for the patient's movement of each joint. This object is used to highlight the difference between the patient's movement and YuMi's movement.

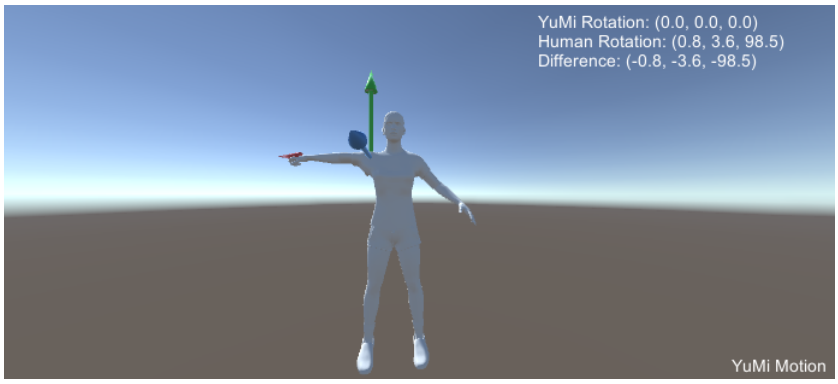

Figure 10. The implementation with an Axis object that represents the patient's movement. The red arrow should stay aligned with the model's arm if the exercise is being performed correctly.

Results

Current Functionality

Network communication between all scripts, Unity, and ROS are fully functional. The rosbag containing the data for YuMi's JointStates is successfully captured by MATLAB, then turned into a Float32MultiArray with the values for the transform, which is sent over the ROS network. The Python script uses the scipy library to turn these values into a transformation matrix and then into quaternion data, which can be used by Unity.

The communication between Unity and ROS is also fully functional. GameObjects such as the lower and upper arms of the human model can receive Pose data over ROS and move accordingly. This is done with the ROS Connector package. The current Unity setup has two human models: one that performs YuMi's received data, and another that represents "human" data, as if the patient were performing the exercise at the same time as YuMi. The user interface also includes functioning buttons that switch between the five cameras.

Future Improvements

There are currently issues regarding the transformations performed in order to match YuMi's coordinates with the human model's coordinates, causing the model to sometimes move in ways that do not match YuMi's movements. The following discrepancies include:
• The differing coordinate systems. YuMi is in a Z-up world, while the human model is in a Y-Up world. This is accounted for by performing a rotation around the x-axis, followed by a rotation around the y-axis.
• YuMi's home coordinates. When YuMi is first turned on, most joints are already pre-rotated, even in the home position. The upper arm and lower arm GameObjects of the human model do not have pre-rotations. This is accounted for by multiplying the given

transformation by the inverse of the rotation done to a joint in YuMi's home position.
• Global coordinates vs. local coordinates. The values collected by MATLAB from YuMi are in global coordinates, while the coordinates for the model are given as local rotations. Multiple debugging attempts were made to feed the human model global coordinate data and move in the global coordinate system, but this still resulted in strange movements. It is believed the source of the problem lies here.
• Differences in human joints and YuMi joints. As discussed in previous sections, the human shoulder is a ball joint, while YuMi uses two rotational joints to simulate the movement of the ball joint. This is accounted for by multiplying the rotation of the first joint by that of the second joint.
• Unity's interpretation of the quaternion. Unity's quaternions come in XYZW format, but when represented as Euler angles in the Unity interface, they do not appear in the expected XYZ order. The correct order was perceived by sending 90 degree rotations around each axis, one at a time, over the ROS network and observing which axis actually rotated in Unity.

Conclusion

Though the model does not move exactly as desired, there is great promise in using a human model alongside the YuMi robot. The added legibility would assist physical therapy patients in understanding the intentions of YuMi as it performs exercises. The technologies used such as ROS, Unity, and programming languages all showcase the capability to achieve the goal of the project.

The source code for this project can be found at https://github.com/selena-radaza/senior-thesis.

Acknowledgements

Thank you to Dr. Begum for overseeing this project and all of your helpful feedback. Thank you to PhD student Paul Gesel for answering so many of my questions.

Bibliography

1. Dragan, Anca D., et al. "Legibility and Predictability of Robot Motion." 2013 8th ACM/IEEE *International Conference on Human-Robot Interaction (HRI)*, IEEE, 2013.

2. Zhao, Min. "An Experimental Study for Identifying Features of Legible Manipulator Paths." *Experimental Robotics*, edited by Rahul Shome et al., Switzerland, Springer International Publishing, 2016, pp. 639–653.

3. "ABB's Collaborative Robot -YuMi." *Robotics*, https://new.abb.com/products/robotics/collaborative-robots/irb-14000-yumi. Accessed 14 May 2022.

4. Powell, Wendy, et al. "Virtual Reality in Entertainment the State of the Industry BAFTA LONDON." *Bafta.Org*, https://www.bafta.org/sites/default/files/uploads/vrpaperbaftasept2017.pdf. Accessed 14 May 2022.

5. "ROS/Introduction - ROS Wiki." *Ros.Org*, 2019, wiki.ros.org/ROS/Introduction.

6. "ROS/Concepts - ROS Wiki." *Ros.Org*, http://wiki.ros.org/ROS/Concepts. Accessed 14 May 2022.

7. Unity Technologies. "Unity." *Unity*, https://unity.com/. Accessed 14 May 2022.

8. "Www.Makehumancommunity.Org." *Makehumancommunity.Org*, http://www.makehumancommunity.org/. Accessed 14 May 2022.

9. “Rosbridge_suite - ROS Wiki.” *Ros.Org*, http://wiki.ros.org/rosbridge_suite. Accessed 14 May 2022.

10. Siemens, "Ros-Sharp Project." *github.com*, https://github.com/siemens/ros-sharp