iOS Based Pose Estimation and Gesture Recognition for Robot Manipulation

Nikhil Deshmukh, Celal Savur, Ferat Sahin,
Rochester Institute of Technology,
Department of Electrical and Microelectronic Engineering,
Rochester, NY, USA
{nxd4573, cs1323, feseee}@rit.edu

2022 17th Annual System of Systems Engineering Conference (SOSE) | 978-1-6654-9623-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/SOSE55472.2022.9812685

***Abstract*—Interfacing between robots and humans has come a long way in the past few years, and new methods for smart, robust interaction are needed. Typically, a technician has to program a routine for a robot in order for the robot to be useful. This puts up a significant barrier to entry into the field of automating tasks using robots—not only is a technician and a computer required, but the robot is not adaptive to the immediate needs of the user. The robot is only capable of executing a pre-determined task and for any change to be made the entire system needs to be paused. This project seeks to bridge the gap between user and robot interface, creating an easy-to-use system that allows for adaptive robot control. Using a combination of computer vision and a monocular camera system and integrated LiDAR sensor on an iPhone, gesture recognition and pose estimation was conducted within an independent system to control the Baxter humanoid robot. The gathered data was sent wirelessly to the robot to be interpreted and then replay actions performed by the user.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

As robots and technology become more and more prevalent in society, the need for intuitive and easy to use robot-human interfaces will rise immediately. The applications for an arm robot aren't just within industrial settings where technicians are on hand; for any hobbyist at home or even normal people, having an extra hand around would be invaluable. The issue then comes with how adaptive is the extra hand or hands and how responsive are they to the user's needs. If it takes time to program the task, then the usability of such a robot will be limited. If the "programming" of a routine can be done on-the-fly by anybody without the need for a technician or a computer, then the usability goes up by several times. Ultimately, the systems should portable and intuitive to use.

This robot-human interface can be separated into two problems: pose estimation of the user by the robot and computer vision-based gesture recognition control and feedback from the user to the robot. To accomplish pose estimation, an iPhone app will be using its camera and LIDAR to track skeleton data on a person. This data was then sent wirelessly over to the robot and translated to robotic arm movements on the Baxter robot. Baxter replicated movements captured using the iPhone close to real-time. A record and playback structure was also implemented to replicate the movements. The other problem involved controlling Baxter's gripper in this case; however, the problem can be expanded to include any form of feedback needed to control other robot functions. For this form of control, computer vision techniques were utilized, specifically the Vision API from Apple was used to detect hand gestures. These gestures were used to control external robot functions. No iOS based system has accomplished this before. Other forms of control exist, such as EMG signals for gesture recognition, and 3D depth sensing cameras for pose estimation. However, these still require an external high power computer to do all of the processing. These current systems can be bulky, expensive, and complex to setup. This was an independent app that contains the entire control system, which can be deployed on phones that various people already carry with them.

These two problems were handled independently and combined within a singular application to be deployed on any modern iOS device. Accuracy was determined based on how close the robot's movements are to the user's movements and also how precisely the robot can execute the program. The responsiveness of the system was rated on how quickly the robot can respond to the gesture recognition. Ideally, the system would respond in close too real-time to the user and/or playback recorded movements accurately.

The contribution of this paper is as follows:

- Develops a novel mobile software platform for robot manipulation
- Explores a new modality for hand gesture recognition in the form of iOS-based Vision tracking
- Explores a new implementation of pose estimation in the form of iOS ARKit
- Compares alternative forms of gesture recognition and pose estimation to the iOS-based system.

The rest of the paper is structured as follows, section II provides a brief look into related works, section III introduces the proposed method, and section IV provides results. Section V discusses results, and section VI talks about conclusion. Each section is divided based on the overarching concept (Pose estimation and gesture recognition).

II. Related Works

This section presents a literature review of the alternative methods used for Pose Estimation, Wireless Communication, and Gesture Recognition.

A. Pose Estimation

Pose estimation is a tried and tested concept. There are various forms of pose estimation using different hardware devices. A very common approach to pose estimation is to use a Kinect sensor for skeleton tracking. A paper published in 2014 elaborates on this exact scenario. [1] Human motions are captured by the Kinect sensor and calculated with Processing Software using SimpleOpenNI wrapper for OpenNIand NITE. UDP protocol is adopted to send reference motion to Baxter robot and joint angles of Baxter robot are calculated based on a vector approach and inverse kinematics approach. The vector-based approach was only able to move 4 out of the 7 joints on Baxter. Extra data is needed of the palm and fingers is required which the Kinect cannot provide. The inverse kinematics approach was able to move all joints; however, the backwards movement is mirrored while sideways movement follow the recorded direction. The Kinect sensor is widely adopted because it is very cost-effective and includes a RGB camera and dual infrared depth sensor. While Kinect can be accurate, the entire system isn't as portable and kinect itself is deprecated.

Another paper also utilizes a kinect to aid with trajectory planning using a Baxter research robot, developed by Rethink Robotics. A function approximation technique (FAT) control system is employed to make the robot follow the trajectory of human motion using the kinect sensor. The UDP communication protocol is employed to send the reference human joint angle data to the robot [2]. However, this project utilizes the kinect V2 while the previous used the older generation kinect. However, while both sensors can accomplish pose estimation, they are now discontinued by Microsoft. As a result, as time passes, they will become less accessible and receive no future support.

B. Wireless Communication

The main goal is to accomplish pose estimation using an iPhone as they are readily available nowadays. While at the time, they aren't as cost-effective as a kinect, in the future lidar-based iPhones will drop in cost as they become more adopted. iPhones are already widely available in the U.S. so the use of their computation power to aid in robotics is both feasible and practical. Baxter is a ROS-based robot, so while the latest iPhone has the technology to conduct pose estimation, it needs to also interface with ROS. A paper from 2014 explains the first interface between iOS and ROS. The authors created a native port of ROS for iOS labeled ROS4iOS. The user can, "start ROS sources hosted on GitHub and apply minimal patches for iOS compilation, concentrate on the C++ portion of ROS' core libraries, create complete ROS nodes running on the mobile device by reusing the same C++ code and design an architecture to interface ROS data structures with iOS user interface elements" [3]. The library itself was a starting point for ROS development on iOS.

A research from 2012 describes earlier implementations of ROSbridge. This is a simplified tutorial introducing ROS and ROSbridge to implement and use state of the art navigation and manipulation algorithms [4]. Another paper from 2015 describes the use of ROSbridge to act as a middleware. The ROSBridge library enables interaction with a ROS environment from external processes. This interaction is facilitated through commands in the form of JSON (JavaScript Object Notation) strings. These commands are mapped to internal ROS functions and enable external processes to act as a ROS node operating within the normal ROS environment [5]. This project used the foundation from ROSbridge and a new modified iOS to ROS wrapper to handle the wireless communication.

C. Gesture Recognition

Gesture recognition has long been a problem domain in computer science, with the goal being interpreting "human gestures via mathematical algorithms" [6]. Gestures are a simple and clear way for humans to communicate with computers. Once these gestures are captured by the computer, the system can then go on to execute the command tied to that specific gesture. This problem has been approached in many ways, such as computer vision, speech commands, and electromyography signals. Each of these approaches have their own advantages and disadvantages.

Computer vision-based gesture recognition necessitates a robust classification algorithm. A 2010 paper explores using two algorithm subsystems, one for static gesture recognition and one for gesture classification. The first system locates the hand region within the camera frame using a cascade classifier. The second system uses vector quantization and a Hidden Markov Model to estimate the motion trajectory of each gesture and then classify. The idea behind this is to allow for the system to recognize different dynamic gestures instead of static ones. However, this method and algorithm are both complex, which is a disadvantage of using computer vision [7].

The difficulty of using camera vision for gesture recognition, as outlined in the papers that used it, is that there is a lot of data coming in and the algorithm must be robust enough to correctly classify certain gestures as opposed to normal movement. This also illustrates a key drawback to using computer vision for gesture recognition - the user must be facing the camera. There is also a lot of noise/extraneous information that must be sorted out, such as other people within vision or objects moving around. To simplify the processing, the iOS app was developed to instead classify a person's individual fingers and identify their relative location to create gestures.

A conference paper discusses a similar AR-based gesture recognition implemented using an iOS device. However, the Hand Gesture Recognition (HGR) framework was developed using depths provided by an iPhone's 2 cameras to produce a stereo configuration [8]. Some advantages to this implementation include the ability to ignore skin color as the framework is primarily operating using depth information. The depth information is used with a RGB color map to help reduce noise produced by the depth maps. However, fast motion will lead to

desyncs between the depth and color maps which leads to errors. Other limitations include extra noise due to shadows and discoloration in skin color which also affect synchronization between the depth and color maps. While a Core ML model was originally going to be used to conduct the iOS gesture recognition, limitations were discovered preventing successful implementation.

III. Methodology

This section contains all the necessary supporting information pertaining to the type of robot, as well as the pre-processed data, types of features to be extracted from the processed data and a brief explanation of methods and implemented algorithms.

A. Robot Interface

The robot used was Baxter, which is a "a human-sized humanoid robot with dual 7-degree-of-freedom (d-o-f) arms with stationary pedestal, torso, and 2-DOF head, a vision system, a robot control system, a safety system, and an optional gravity-offload controller and collision detection routine" [9].

Baxter is ROS enabled, with a dedicated SDK supporting ROS Indigo and ROS Kinetic, however it only supports up to Python 2.7. Baxter also has two interchangeable grippers. However, the robot that was available only had one working gripper on the right arm.

B. Pose Estimation

An iPhone 12 Pro was used to conduct the actual pose estimation. This specific iPhone is important because the rear housing unit includes a LiDAR sensor. The LiDAR is capable of tracking up to a distance of 15 meters. A custom app was written utilizing ARKit and the RBSManager library to interface with ROSBridge. RBSManager is a Swift-native library for handling the WebSocket connection to a ROS master running ROSBridge [10]. A custom message type was implemented into the library to send the relevant joint angles over the network. The joint angles was extracted from the pose estimation on the iPhone. The iPhone was outputting relative data in the form of a 4x4 transform matrix as shown below in Figure 1.

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ P_x & P_y & P_z & 1 \end{bmatrix}$$

Fig. 1: Transform Matrix

A publisher on the iPhone was written to send this data over a rostopic over the ROSBridge. The ROS master node will subscribe and get then iOS joint angles. These angles was modified using a transformation and offset matrix to transition to the Baxter frame environment. These updated joint angles will then be sent to Baxter to mimic the human movement using forward kinematics. This data was sent as soon as the angles are calculated allowing Baxter to move in real-time with the human operator. Additionally, the x, y, z location of the user's hand was extracted using the iOS pose estimation. This data was sent to Baxter to be used for Inverse Kinematics using Baxter's IK solver.

The iPhone communicated over the ROSBridge over a local wireless network that will have to be created. Latency due to the wireless transfer of data was a limiting in the effectiveness of real-time pose estimation. The iOS app allows for a configurable ROS bridge host connection, and has the ability to display various joint angles of the right side of the body. Additionally, it overlays a 3D skeleton model onto the person to depict a rough visual representation of what the iPhone is interpreting.

C. Gesture Recognition

For gesture recognition, the iOS Vision API was used in conjunction with a Core ML model to detect open and closed hand positions and use these as inputs. Initially, a readily available Core ML model was utilized for initial testing provided from an open-source GitHub repository [11]. Once a working system was implemented, a new model was created and trained using Microsoft Azure and CustomVision.ai. It was ultimately a simple model aimed to detect a closed and open fist on a person's right hand.

A closed fist acted as the command to close Baxter's gripper while an open hand shall open Baxter's gripper. To avoid flooding Baxter with repetitive commands, an operation was only published if there was a change from an open to closed fist. All the proposed logic was implemented in the same app that was conducting the pose estimation. All computation was done on the iOS device, and final commands were sent to Baxter over ROSBridge. This alleviated computation time on Baxter's end and kept latency minimal. The alternative form of Gesture Recognition that was also be implemented was using tracked finger points to determine a gesture. Using the Vision API, the locations of the thumb, index, and middle fingertip joints were determined. Using this data, the distance was calculated with respect to the wrist joint. Based on a specific threshold, it can be determined if the hand is closed or open based on the combination of distances. It is important to note that this method has no knowledge of depth information, and only the X, and Y data are used for the calculations. These calculations were conducted alongside the pose estimation algorithm and sent to Baxter in real-time.

D. Expected Results

Baxter's movements are slightly clunky and have a small amount of wobble and drift to them. For this reason, the final result may not have the desired accuracy and/or speed. Additionally, as this is a monocular system, certain body positions may be difficult to determine based on the camera angle. It is expected that the pose estimation will translate well into Baxter's joint movement system, however mimicking the movements in perfect real-time may not be achievable. Also, ideally, the gesture recognition is real-time as well but the delays in data transmission through wireless connections,

along with any delays in classification, could result in a larger latency. Gesture Recognition may also become less accurate the farther the user is from the cameras, depending on the dataset used in the Core ML model. The finger joint-based model may also suffer the same issues with regard to accuracy and distance as the user becomes farther away from the camera.

The primary objective is to use forward kinematics to mirror movements to Baxter. This will allow for true mimicry. While inverse kinematics(IK) would reach the same end effector position, the approach may be different depending on the IK solver being utilized. From the research, ARKit provides both local and universal 4x4 transformation matrices. This should be enough information to compute the forward kinematics and joint angles on the device to be sent to Baxter. However, additional processing will also be needed to convert the real-world coordinate system's to Baxter's coordinate system. Some data will also need to be truncated as Baxter does not have the same reach as a person. In addition, Baxter has more degrees of freedom than what the pose estimation can track. As a result, some joint angles may need to be omitted based on the extracted data from the pose estimation.

IV. Results

A. Pose Estimation

The pose estimation was conducted using ARKit 5. Specifically, *ARBodyTrackingConfiguration* is used to create a *BodyTrackedEntity*. A 3D skeleton model is then overlayed onto the user in the AR View giving a visual representation on the iPhone's joint representations. From this entity, the joint transforms can be extracted from a total of 92 unique joints.

The joint information is determined using Apple's custom machine learning algorithm running on the A12 Neural engine in the iPhone 12 Pro. The relevant joints are listed below in Table I

TABLE I: Relevant ARKit Joints

Left ARKit Joint	Right ARKit Joint
left_hand	right_hand
left_arm	right_arm
left_forearm	right_forearm
left_shoulder_1	right_shoulder_1
left_handIndexEnd	right_handIndexEnd

As Baxter is a 7 DOF robot, 3 joints were omitted when calculating and sending joint angles. These are the wrist twist and wrist pitch as the ARKit algorithm doesn't have a large enough degree of accuracy tracking the wrist joint movements. To calculate the joint angles for Shoulder Pitch and Elbow Pitch, the following Equation, Equation 1, was used.

$$\begin{aligned} radians &= atan2(C.y - B.y, C.x - B.x) \\ &\quad - atan2(A.y - B.y, A.x - B.x) \\ angle &= |radians * \frac{180}{\pi}| \end{aligned} \tag{1}$$

A, B, and C represent the joint XYZ locations from the corresponding joint transforms. For example, A = shoulder, B = Arm, C = forearm, would result in the Shoulder Pitch angle. It is important to note that all transform matrices are universal transforms with respect to the universal joint which is the hip joint. Additionally, to convert to Baxter's environment, some angles needed offsets of $\pm 90°$ or $\pm 180°$.

With all the joint data extracted, the relevant data was then mapped to Baxter's joints, shown in Figure 2 below, with the wrist twist and pitch being locked in their "0" position.

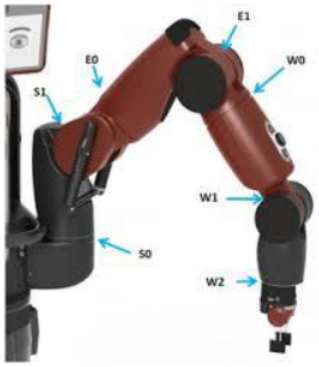

Fig. 2: Baxter Joints

The mapped joints and motions are illustrated in Table II below. It is important to note that the gripper twist is not functional on the available Baxter Robot.

TABLE II: Baxter Mapped Joints

Baxter Joint	ARKit Joint Angle	Actively Tracked
S0	Shoulder Twist	Yes
S1	Shoulder Pitch	Yes
E0	Elbow Twist	Yes
E1	Elbow Pitch	Yes
W0	Wrist Twist	No
W1	Wrist Pitch	No
W2	Gripper Twist	No

B. Gesture Recognition

Initially, a Core ML model was tested in conjunction with the iOS Vision API. However, this implementation couldn't be further developed as the accuracy was too unreliable at the distance needed to also conduct the pose estimation. As a result, only the computer vision-based finger tracking was fully implemented in the final revision. Every AR skeleton update would send a new camera frame to the Vision API. This captured image was then converted to a *CvPixelBuffer* to be able to read the pixel information using the Vision API. This was sent to a *VNImageRequestHandler* which conducts multiple vision requests on the image using the custom Hand Gesture Processor (HGP) handler. The handler processes observations on three fingers and determines their location within the frame in the form of XY coordinates. This information was then used in conjunction with the location of the wrist joint to

calculate the 2D distance between each finger and the wrist. A threshold of 18 pixels was found to be the optimal distance to achieve a strong confidence level. It is important to note that since the threshold is specified in pixels, the accuracy is location-dependent. As a result, this threshold was found when the iPhone was a far enough distance away to still conduct the pose estimation. Based on the distances calculated, the HGP will cycle between fives states: Pinched, Possible Pinched, Apart, Possible Apart, and Unknown.

The XY data was then transformed to the iOS AR view coordinate system to be displayed on screen. While initially this computation was conducted on every frame update from the pose estimation algorithm, it was found to cause frame drops which would then cause the application to lag. As a result, the gesture recognition algorithm was cut to run on every 6 updates. This yielded a fluid application while not lagging too far on the real-time aspect of the gesture recognition. With approximately 60 updates per second, the gesture recognition algorithm is updating every tenth of a second.

C. iOS App

Each algorithm was first tested in an individual app before being combined into one complete package. The joint angles were sent to Baxter using RBSManager and ROSBridge over a local wireless network. A simple host input screen was added to specify the ROSBridge IP.

RBSManager was modified and given a Float32MultiArray message type. This is natively supported by ROS; however, it was missing from the iOS library. This message type allowed the iOS publisher to send two arrays to unique topics (*/robot/ios/commands/left* and */robot/ios/commands/right*) for the left and right arms of Baxter. Each array contained the necessary joint angles for that arm in the order specified: $[S0, S1, E0, E1, 0]$. The app supports a live view of the joint angles updating in real-time for the left side of the body. It also gives dynamic feedback by subscribing to a ROS topic which specifies if the gripper is closed or open. This is then displayed on the app to easily be seen. All of this information can be seen in Figure 3 to the right along with the overlayed 3D skeleton visualization.

Figure 3 also depicts the overlayed HGP points. The green color indicates a pinched state, while red indicates the apart state. Any almost pinched or almost apart states are colored orange. While the colors and relative location are accurate, the absolute position is slightly off. This is because the Vision API is generally utilized on a live camera feed working independently. However, to get both the pose estimation and Vision API working simultaneously, the captured frames need to be pulled from the AR updates instead. Transformation between the live feed coordinate system and overlay system is natively supported. The switch views button within the app gives the option to only use the gesture recognition algorithm on a live feed. The resulting overlay has correct absolute positioning of the hand.

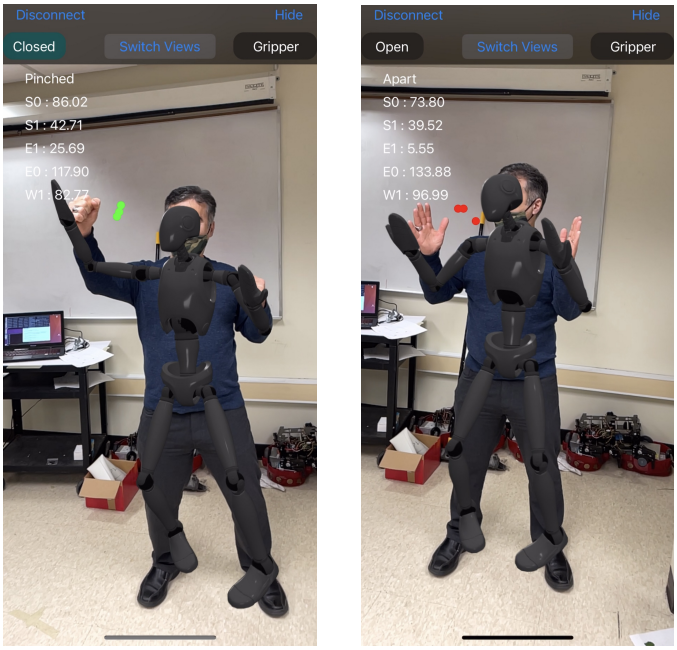

Fig. 3: iOS App Pose Estimation and Gesture Recognition

V. Discussion

The iOS app was able to achieve near real-time communication and mimicry between the Baxter robot using forward kinematics. However, using inverse kinematics yielded alternative results. The XYZ data from the left and right ARKit hand joints were used as end effector locations for the Baxter IK solver. This allowed Baxter to use inverse kinematics to move the gripper to the same hand position in the real world. However, there was no orientation quaternion given. As a result, some locations had multiple solutions to reach the goal, so Baxter does not always follow the same motion. The orientation was locked to the overhead position as shown in Figure 4 shown below.

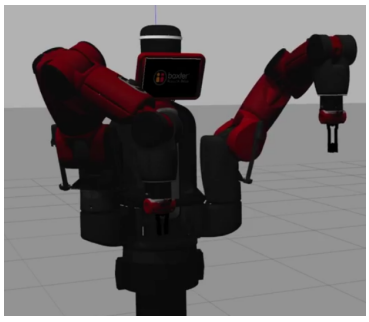

Fig. 4: Baxter Overhead Orientation

As a result of this orientation, the location data needed to be truncated in the X and Z direction as the fixed orientation limits Baxter's range of movements. Inverse kinematics was also slower in comparison to forward kinematics as now Baxter had to do the IK computations before being able to move. Due to the slow nature, the configuration was set to

record and playback system where the XYZ locations were stored in an array and then sent to Baxter to replicate once a recording was complete. The gesture recognition algorithm was able to successfully identify the open and closed fist with only two limitations/constraints. Since the Vision API only relies on the camera and has no depth information, it is locked to a distance of approximately 6 feet from the camera. Any larger distance will result in the halt of the HGP and the state will be frozen in the last known state until the user is within 6 feet again. Additionally, because this is a monocular system, if the user's hand is horizontal with the camera, the accuracy falls as it is very difficult to identify the fingertip locations. The entire deployed system on Baxter can be seen in Figure 5 below.

Fig. 5: Complete iOS System

VI. Conclusion

Based on the findings of this research, it can be seen that the monocular iOS system is successful in controlling a humanoid robot. Given the constraints of a one camera system, the iPhone was able to accurately track arm movements in the X, Y, and Z directions and relay them to Baxter in near real-time. Although the gesture recognition was purposely throttled due to computational power, the speed at which it detected hand gestures was not significantly affected. The app was still able to transmit gripper controls in near real-time as well. The inclusion of the LiDAR sensor on the iPhone is what truly allowed the pose estimation algorithm to have the accuracy that it did. While it may not necessarily have the same degree of precision as an Intel real-sense or kinect camera system, technology will only improve over time. The accuracy for the iOS system is still strong enough at this given time and the software and hardware are still being actively maintained and updated. The main goal of the system was to be portable and intuitive, while still maintaining a strong degree of accuracy. The gesture recognition algorithm, while simplistic was still effective. It is not as accurate as EMG sensors as computer vision is constrained by the number of cameras and resolution. However, to reiterate, this system is still more portable and requires no external hardware other than a small iPhone which is now widely available.

VII. Future Works

The main future work would be to possibly further optimize the gesture recognition algorithm and add a few more gestures to the HGP. Additionally, a large portion of development would go towards calculating an orientation end effector position based on the retrieved joint transform data from ARKit. This would allow the IK method to follow true mimicry similar to the forward kinematics method. Inverse kinematics is inherently more accurate when trying to move an end effector to a known position, so achieving proper accuracy with this method would be very beneficial.

Acknowledgment

The authors are grateful to the staff of Multi-Agent Bio-Robotics Laboratory (MABL), the CM Collaborative Robotics (CMCR) Lab, and the Electrical Engineering Department at RIT for their valuable inputs. This material is based upon work partially supported by the National Science Foundation under Award No. DGE-2125362. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] H. Reddivari, C. Yang, Z. Ju, P. Liang, Z. Li, and B. Xu, "Teleoperation control of baxter robot using body motion tracking," in *2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, pp. 1–6.

[2] G. Peng, C. Yang, Y. Jiang, L. Cheng, and P. Liang, "Teleoperation control of baxter robot based on human motion capture," in *2016 IEEE International Conference on Information and Automation (ICIA)*, 2016, pp. 1026–1031.

[3] R. Chauvin, F. Ferland, D. Letourneau, and F. Michaud, "Ros4ios: Native ros development on ios devices," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2085–2085.

[4] C. Crick, G. Jay, S. Osentoski, and O. C. Jenkins, "Ros and rosbridge: Roboticists out of the loop," in *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2012, pp. 493–494.

[5] R. Codd-Downey and M. Jenkin, "Rcon: Dynamic mobile interfaces for command and control of ros-enabled robots," in *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 02, 2015, pp. 66–73.

[6] J. Kobylarz, J. J. Bird, D. R. Faria, E. P. Ribeiro, and A. Ekárt, "Thumbs up, thumbs down:non-verbal human-robot interaction through real-time emg classification via inductive and supervised transductive transfer learning," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 12, pp. 6021–6031, December 2020. [Online]. Available: https://publications.aston.ac.uk/id/eprint/41366/

[7] W. Ke, W. Li, L. Ruifeng, and Z. Lijun, "Real-time hand gesture recognition for service robot," in *2010 International Conference on Intelligent Computation Technology and Automation*, vol. 2, pp. 976–979.

[8] E. C. E. Vidal and M. M. T. Rodrigo, "Hand gesture recognition for smartphone-based augmented reality applications," in *Virtual, Augmented and Mixed Reality. Design and Interaction*, J. Y. C. Chen and G. Fragomeni, Eds. Cham: Springer International Publishing, 2020, pp. 346–366.

[9] R. L. Williams, "Baxter humanoid robot kinematics - ohio university," Apr 2017. [Online]. Available: https://www.ohio.edu/mechanical-faculty/williams/html/PDF/BaxterKinematics.pdf

[10] Wesgood, "Wesgood/rbsmanager: A swift library for connecting to ros using rosbridge and websockets," Jan 2018. [Online]. Available: https://github.com/wesgood/RBSManager

[11] Hanleyweng, "Hand gesture recognition." Oct 2017. [Online]. Available: https://github.com/hanleyweng/Gesture-Recognition-101-CoreML-ARKit