Design and Development of an Online Robot Programming and Simulation Framework using the Robot Operating System (ROS)

Ru Sern YEOH
Department of Mechanical Engineering
Faculty of Engineering, University of Malaya
Kuala Lumpur, Malaysia
r.s.yeoh@hotmail.com

Hwa Jen Yap
Department of Mechanical Engineering
Faculty of Engineering, University of Malaya
Kuala Lumpur, Malaysia
hjyap737@um.edu.my

Chee Hau Tan
Department of Mechanical Engineering
Faculty of Engineering, University of Malaya
Kuala Lumpur, Malaysia
cheehau86pl@gmail.com

Sivadas Chandra Sekaran
Aerospace Malaysia Innovation Centre (AMIC), Asia Aerospace City
Selangor, Malaysia
sivadas@amic.my

Siow-Wee Chang
Bioinformatics Programme
Institute of Biological Sciences
University of Malaya
Kuala Lumpur, Malaysia
siowwee@um.edu.my

Kan Ern Liew
Aerospace Malaysia Innovation Centre (AMIC), Asia Aerospace City
Selangor, Malaysia
liew@amic.my

***Abstract*—In the Fourth Industrial Revolution, robotics technology plays an increasingly important role in order to increase productivity through the use of cyber-physical systems. However, industrial robotic arms require expertise in fields such as mechanical and software engineering in order to be used. Furthermore, the modularity of robotic work cells could be improved. In this project, an online robot programming and Simulation framework is developed in the Robot Operating System (ROS). The framework includes a master and slave node that allows for teleoperation of the intended robotic arm. A graphical user interface (GUI) is provided on the master personal computer (PC) in order to receive a target coordinate point for the robotic arm end-effector from the user. The EezyBotArm Mk2 3-dimensional (3D) printed arm is used for control and testing. The kinematics study of the robotic arm is performed and based on the equations derived is used to convert the coordinate point into the corresponding joint variables. The joint variables are then transmitted from the master PC to the slave Raspberry Pi. The Raspberry Pi interfaces with an Arduino UNO board in order to control the servo motors on the robotic arm via pulse width modulation (PWM) signal.**

Keywords—cyber-physical, robot operating system (ROS), modularity, robotic arm, kinematics

I. INTRODUCTION

The idea of the fourth industrial revolution, or better known as Industry 4.0 was initially proposed in Germany [1]. The key technological advancement in Industry 4.0 is the introduction of cyber-physical systems (CPS) in the production floor, which would help to improve the effectiveness and efficiency of the industry. The use of autonomous robots will help to reduce manual labor costs, and increase the productivity of any given industry. Robots are often employed in areas which involve highly repetitive tasks, or involve great risk to human lives. This is because unlike human labor, autonomous robots are able to operate for long hours without fatigue, and are easily replaceable when broken. Combined with the other key pillars of Industry 4.0 such as digital twins and IoT, the flexibility of autonomous robots is further increased. With this, autonomous robots can be easily monitored remotely through the visualization of the data collected remotely from the robots.

However, the field of robotics is complex and is situated at the intersection of core engineering fields such as mechanical, electronics, control and software engineering. Traditionally, this meant that robotics projects are inherently complex and require expertise in multiple fields. Robot Operating System (ROS) is a popular open-sourced platform that is enabling rapid research in the field of robotics. The availability of various modules and development tools has allowed fast prototyping of ideas and a decrease in research costs [2]. Also, ROS is available in many programming languages such as C++, Python, Octave and LISP which further cements its position as a flexible tool.

II. ROS ARCHITECTURE

ROS is a software framework that is used for writing software for robotics application. It provides a collection of tools that can aid in the process of the creation of complex and robust robotics systems across multiple platforms [3]. The ROS file-system is organized into packages, where each package contains important executables, libraries and configuration files. In ROS architecture, the main data structure that is being used through the ROS framework is through the use of messages. Each functionality in the robotic system can be represented as a node in ROS, which allows the ROS framework to be distributed and modular. A ROS Master then manages the communications between each ROS node, without which the ROS nodes would not be able to send or receive messages. Communication between each ROS node then occurs through the concept of publishing and subscribing, where a ROS node may make information computed from the node public by publishing messages to a ROS topic, and other ROS nodes may have access to the information by subscribing to the same ROS topic.

There are many projects that have been implemented through the use of ROS architecture. One such project is a limb rehabilitation robot that was proposed researcher [4], which involves a master and slave type of architecture where a master personal computer (PC) is used to host the ROS software framework that then interfaces with various peripherals such as the robot itself and input devices. A simulation software called Gazebo is also run on the master PC in order to simulate the movements of the patient and provide important metrics in the training program.

Besides that, another project made use of the Gazebo simulator in ROS to run simulations on the manipulation task of a robotic arm [5]. The robotic arm was modeled in Gazebo using the Unified Robot Description Format (URDF), which is a type of eXtensible Markup Language (XML) file used to provide information on the joints and links of a manipulator. The URDF format makes use of a stereolithography (STL) file which is generated from computer-aided design (CAD) models in order to provide visualization of the robotic arm in Gazebo. Another simulation tool that is available in the ROS framework is RViz. While Gazebo is able to simulate the physics involved in the operations of a manipulator or robot, Rviz is a lighter weight tool that simply provides visualization of the robot in the simulation.

Researchers proposed a 6 DOF robotic arm that was also designed in the ROS framework [6]. The system used the RViz tool for visualization, and the Moveit package within ROS to solve the FK and IK problems. The developed system makes use of a graphical user interface (GUI) for the user to give an input of the desired point in the 3D workspace for the end-effector to move to. Then, the IK solver in the Moveit package is used to compute the necessary joint variables need to bring the end-effector of the robotic arm to the desired location. The motion planner is able to generate the optimum trajectory and FK analysis is used to display the simulated path of the movement of the robotic arm in RViz. Similar work was also conducted in designing a simulator for a 5 DOF Selective Compliance Articulated Robot Arm (SCARA) used in deburring process [7]. The proposed system utilized a linear trajectory between desired end-effector locations and performed trajectory generation calculations in a customized ROS node.

III. METHODOLOGY

Fig 1. shows the implementation of the project. On the computer, there is a GUI that allows the user to given an input of the desired coordinate point of the robot arm end-effector. Then, a simulation of the configuration of the robot arm is presented to the user. The data on the required joint angles to achieve the configuration is transmitted wirelessly through the wireless local area network (WLAN) router to the Raspberry Pi. The Raspberry then interfaces with the Arduino Uno board in order to control the servo motors at the robot arm joints. The robot arm was fabricated using additive manufacturing and then assembled together manually. Then, Fig. 2 shows the launched window containing the robot visualization and the slider GUI to input the desired coordinate of the robot end-effector and also to control the actuation of the robot gripper. With these tools, the user is able to perform teleoperation on the robotic arm. The flow of the boot-up steps of the system is shown in Fig. 3.

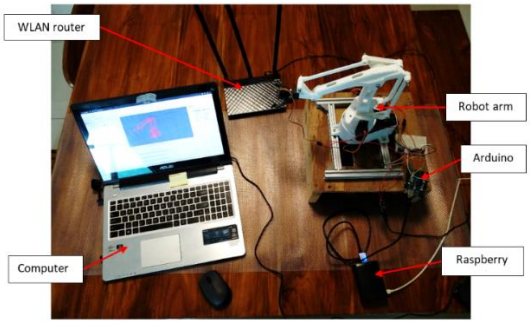

Fig. 1. Setup Layout

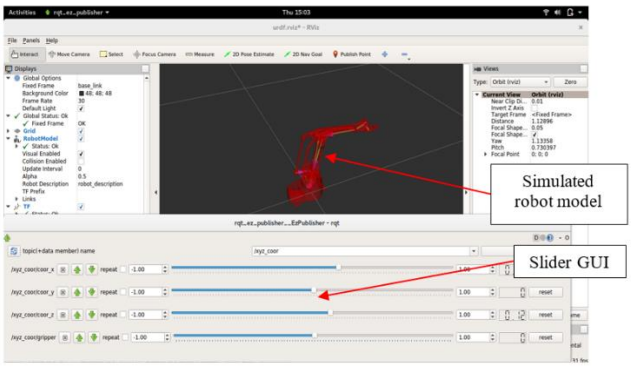

Fig. 2. Visualization tool and GUI

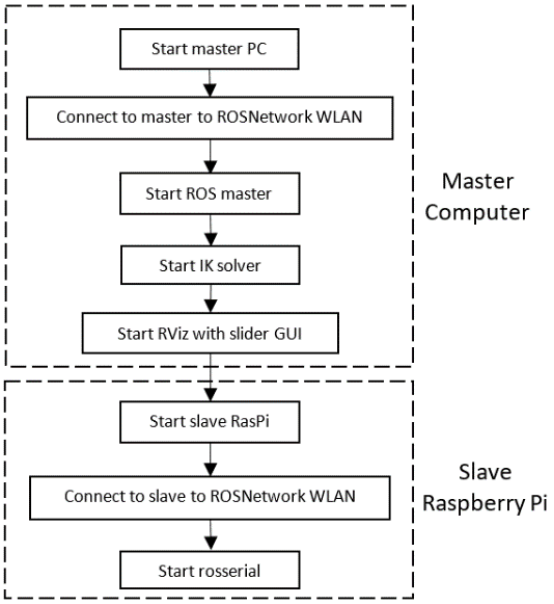

Fig. 3. The flow of the boot-up steps

A. *Software Architecture Block Diagram*

Fig. 4 shows the illustration of the implemented network. Only a single master and a single slave node is used. The entire project will be built upon the ROS framework. It is possible to add additional slave nodes to be controlled, but due to the application of the project, only a single slave node with a single robotic arm is used. The distributed network type of the master-slave model allows teleoperation of the robotic arm, within the boundaries of the WLAN. It is also possible to expand the connection through the use of the internet also.

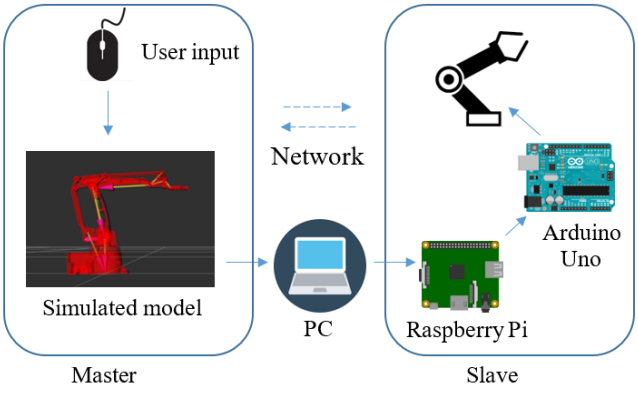

Fig. 4. Block diagram of master-slave network architecture

The master node hosts the ROS master node, provides a GUI for receiving a target coordinate input from the user, performs the calculation for the inverse kinematics, and visualization of the robot model. The master node is hosted by a PC which is running on an Ubuntu operating system (OS). The installed ROS version is the Kinetic Kame. The slave node is hosted by a Raspberry Pi embedded computer. The Raspberry Pi runs on the Ubuntu Mate OS, and the ROS Kinetic Kame is also installed. The Raspberry Pi is connected to an Arduino Uno board through the Universal Serial Bus (USB) connector which is responsible for controlling the servo motors on the robotic arm.

B. Software Architecture Block Diagram

Fig. 5 shows the block diagram of the software architecture employed in this project. The rectangular boxes represent the topics, while the ovals represent the nodes where the nodes publish and subscribe to messages. An explanation of the structure of ROS nodes and topics can be given below:

- **/GUI** – Represents the GUI node that interacts with the user in order to obtain a target coordinate in the form of an X, Y and Z-axis coordinate points which will be published to the **/xyz_coor** ROS topic. The GUI to be employed here is a simple slider where the user can control the X, Y and Z points individually.
- **/compute_IK** – The ROS node that receives the target coordinate point from the GUI via the **/xyz_coor** topic. The computation for the inverse kinematics is performed on this node using the equations derived in section 3.3 Inverse Kinematic Analysis of Robotic Arm and section 3.4 Open-loop Chain Equivalent of Robotic Arm. The output angles for the robot model control is published to the **/Ard_angle** topic, while the output angles for the robot visualization in RViz is published to the **/joint_states** topic.
- **/serial_node** – The ROS node that is subscribed to the **/Ard_angle** topic and is responsible for communication with the Arduino board.
- **/robot_state_publisher** – The ROS node that is responsible for the calculation of the vectors to represent the robot model in RViz. Calculates the necessary transformations needed based on the information on the URDF file and publishes the information to RViz via the **/tf** and **/tf_static** topics.

Fig. 6 shows the organization of nodes between the Master and the Slave. All the computation is performed on the master node, which is the PC, while the slave node functions only to connect the Arduino board to the ROS environment and execute the servo actuation on the robot model. The Master contains the **GUI**, **robot_state_publisher** and the **compute_IK** nodes, while the Slave contains the **serial_node**. However, all these nodes appear to be in the same environment due to the use of the same ROS Master. When creating the nodes, the nodes are explicitly subscribed to the ROS Master which is hosted on the Master by specifying the ROS_MASTER_URI. This enables the ROS environment to be distributed across multiple machines but functioning as a single environment.

C. Robotic Arm Model

In this project, the EezybotArm Mk2 robotic arm is selected for use. The EezybotArm mk2 is an open-source robotic arm design that was modeled after the ABB IRB460 robotic arm. It has 3 DOF and gripper end-effector. It uses a closed-loop kinematic design and servo motors for actuation. The robotic arm is designed to be fabricated by additive manufacturing or 3-Dimensional (3D) printing. It consists of 19 3D printed parts and 55 non-printed parts. Fig. 7 shows the CAD model of the manipulator and its exploded view detailing the 3D printed parts.

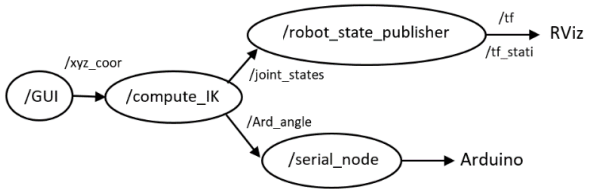

Fig. 5. Block Diagram of Software Architecture

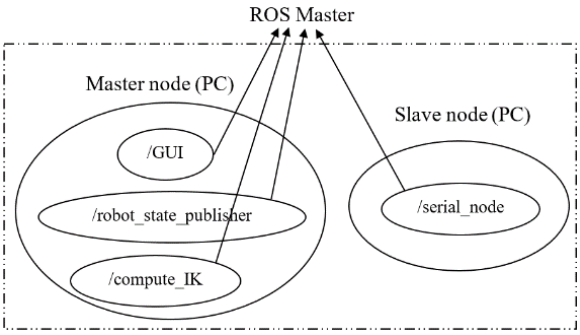

Fig. 6. Node Organization of Master and Slave

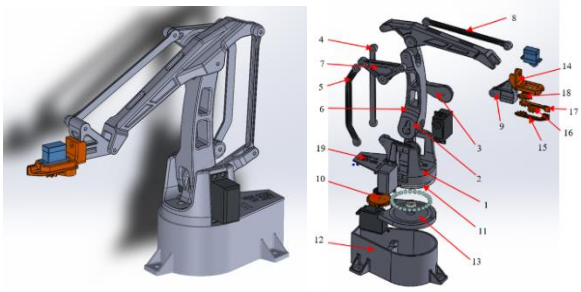

Fig. 7. CAD model EezybotArm Mk 2

D. Inverse Kinematic (IK) for Robotic Arm

In this project, the robotic arm should be able to bring the end-effector to the desired location given a set of coordinate points as the input. In order to achieve this, the inverse kinematics (IK) of the EezybotArm Mk2 must be developed. The home or default position where all the joint angles are set at 0°. The coordinate frame is then assumed as follows: X-axis refers to robot arm move forward/backward, Y-axis goes to left/right, and Z-axis moves up/down. This follows the 'right-hand rule' convention for the axis naming.

Fig. 8 shows the joint variables to be controlled in the robotic arm model. The joint variable $\theta 1$ is a rotation of the robot base piece about the Z-axis. The joint variable $\theta 4$ is the rotation of the robot main arm piece about the y-axis. Lastly, the joint variable $\theta 5$ is the rotation of the robot V-arm piece about the y-axis. Open-loop Chain Equivalent

E. *Open-loop Chain Equivalent*

The ROS visualization tool RViz does not support the mechanics of closed-loop chain type manipulators. Therefore, in order to successfully visualize the EezybotArm Mk2 within RViz, the robotic arm model must be converted to an equivalent group of open-loop chains. The joint angles from the open-loop equivalent chains will then be used back to control the visualized model in RViz in order for it to properly display the correct configurations given the joint angles *θ1*, *θ4* and *θ5*. Fig. 9 shows the open-loop chain equivalent of the robotic arm model. Then, the Denavit–Hartenberg (DH) parameters can be derived for the open-loop chain equivalents in order to solve the FK analysis. Implicitly in RViz, this analysis is done based on the configurations provided in the URDF file. Then, based on the results of the FK calculations, RViz is able to provide the correct visualization of the configuration of the robotic arm based on the joint variables provided. The DH Convention analysis for each chain is shown in Fig. 10.

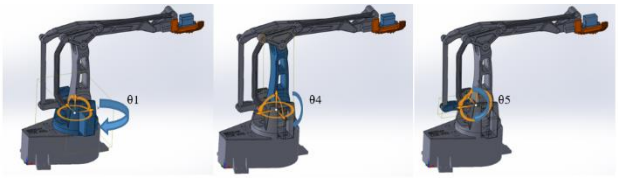

Fig. 8. Joint variables for the robotic arm

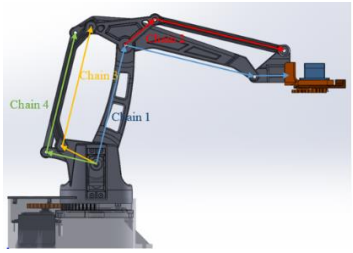

Fig. 9. Open-loop chain equivalent

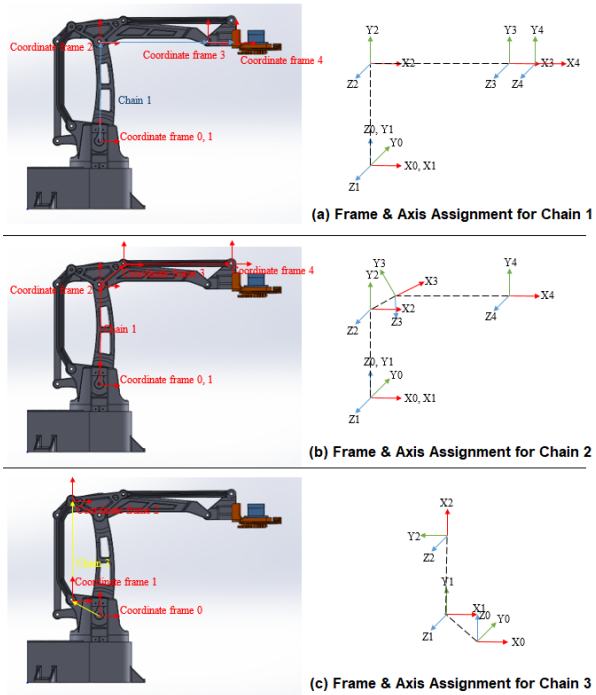

Fig. 10. Frame and Axis Assignment for each chain

IV. Experiment and Validation

A. *Computational Time for Inverse Kinematics (IK)*

The computational time of inverse kinematics is the time taken by node **/compute_IK** to process the information. In this experiment, the time from a coordinate point being published to the **/xyz_coor** to the time the joint variables calculated from the inverse kinematics is published to the **/joint_states** and **/Ard_angle** topics are taken. This is the computational time required to calculate the necessary joint variables through inverse kinematics.

In setting up the experiment, the *printxyz.py* file which creates the **/compute_IK** node is modified. The system time is recorded when a message from the **/xyz_coor** topic is received, and the system time is recorded again when calculated output is published to the **/joint_states** and **/Ard_angle** topics. The two system times are compared and recorded into a comma-separated values (csv) file. The data is collected for 1000 random coordinate point calculations, and the results are plotted.

Fig. 11 shows the graph of frequency against the computational time is a plot from the data collected. The computational time in the experiment ranged from 474,930 nanoseconds to 1,880,775 nanoseconds. The mean is 963,706 nanoseconds, the mod is 741,005 nanoseconds, and median as 91,5051 nanoseconds. This shows that the average computational time is less than 1 millisecond. Thus, in the system, all the nodes are set to publish to the respective topics at a maximum frequency of 10Hz to ensure that there is ample time for the computation of IK to be completed between receiving coordinate inputs from the GUI.

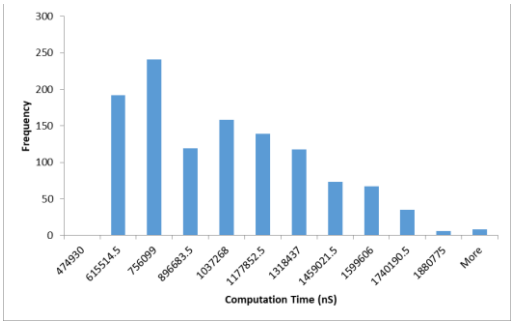

Fig. 11. Graph of frequency against computational time

B. *Data Transmission across WLAN*

Fig. 12 shows the transmission of data from the **/compute_IK** node to the **/serial_node** that happens across the master and slave. The first part of this experiment is conducted in order to ensure that there is no packet loss in the transmission of data across the master and slave. In the second part of the experiment, the time delay between the message being transmitted from the **/compute_IK** node to the **/serial_node** is recorded.

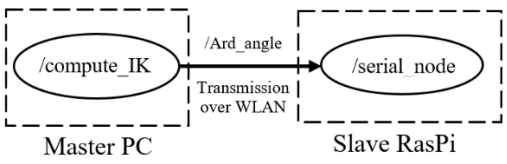

Fig. 12. Data transmission across multiple machines

In setting up the first part of the experiment, the message being sent out of the **/compute_IK** node and the message being received by the **/serial_node** are compared. First, the system is set up. Then, both the outgoing and the incoming message is logged into a csv file for comparison. The data log from both the outgoing and incoming messages are compared for similarity, and the data are tabulated in Table 1.

TABLE 1 COMPARISON BETWEEN OUTGOING AND INCOMING MESSAGES (RAW DATA)

Incoming/Outgoing	Number of Data Points
Similar	1,530
Dissimilar	3,698

From the 5,228 data points collected, it can be seen that only 1,530 of the messages being sent and received are similar. The percentage of times where the sent and received messages are dissimilar, which the percentage of dissimilarity (D) is:

$$D = \frac{3968}{5228} \times 100\% = 75.89\% \quad (1)$$

Next, the message data is rounded down to 4 decimal points and checked again. The percentage of times dissimilar is calculated again, and it is found that the similarity between sent and received messages is shown in Table 2, which the percentage of dissimilarity is zero (0). This shows that there is a loss of transmission data due to the use of the floating-point numbers in the joint variable data. However, after rounding down to 4 decimal points, the data being sent and received has become the same. This shows that the information in the transmitted data is being preserved up to 4 decimal points.

TABLE 2 COMPARISON BETWEEN OUTGOING AND INCOMING MESSAGES (ROUNDING)

Incoming/Outgoing	Number of Data Points
Similar	5,228
Dissimilar	0

In the second part of the experiment, the latency of the data transfer across the network is measured. The message structure in ROS makes use of headers in order to embed some metadata. One type of metadata that can be embedded is the time at which the message is published to the topic. Using this, the header time can be compared to the system time when the message is received by the **/serial_node** to determine the latency in the transmission across the master and slave.

エラー! 参照元が見つかりません。 3 shows the tabulated data from the experiment. From the analysis being conducted, the mean delay time is 47,664,370 ns, the median delay time is 44,825,077 ns and the mode delay time is 44,711,113 ns. This shows that the average delay time between the transmission of the message from the **/compute_IK** node to the **/serial_node** node is about 0.045s (45,000,000 ns) . With a computational time of 0.001s (1,000,000 ns) and a publishing frequency of 10Hz, this ensures that no message that is published will be missed.

TABLE 3 TIME DELAY

Delay time (ns)			Frequency
0	–	40,489,912	1
40,489,912	–	61,939,621	1528
61,939,621	–	83,389,330	20
83,389,330	–	104,839,039	7
104,839,039	–	126,288,748	5
126,288,748	–	147,738,457	7
147,738,457	–	169,188,166	8
169,188,166	–	190,637,875	3
190,637,875	–	212,087,584	0
212,087,584	–	233,537,293	1
233,537,293	–	254,987,002	3

C. Accuracy of Robot Actuation (IK)

In this experiment, the accuracy of the robotic arm is inspected. In setting up the experiment, a grid of 1cm boxes is used to track the movement of the robotic arm. Then, the movement is compared to the visualized model on the Master PC for verification. The robotic arm is made to trace a box. The image of the robotic arm at the four vertices of the box is captured. The experiment is repeated for both the X-Z plane and the X-Y plane. Fig. 13 shows the result of the movements tested on the robotic arm in the X-Y plane. The image of the physical robot arm is compared to the simulated movement of the robotic arm. From the images, it can be seen that the physical robot arm has poor accuracy, although the received data are correct. The robotic arm is unable to track a straight movement.

Next, the movement of the physical robotic arm is compared to that of the simulated robotic arm in the X-Z plane, as shown in Fig. 14. Once again, the physical robotic arm shows a poor ability to track the movement in a straight line. However, this is not due to the incorrect joint variables being calculated by the IK. This is because the simulated robot is able to track straight movements using similar joint variables as shown. Therefore, the inaccuracy is due to a physical limitation of the servo motors and the build of the robotic arm itself. During the assembly of the robotic arm parts, some of the holes which held the pivot joints of the robotic arm parts were too small due to the additive manufacturing process. Therefore, they had to be manually drilled through with a hand drill, thus introducing tolerance errors.

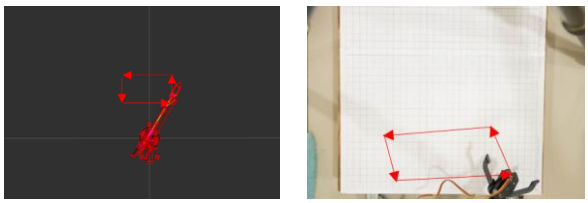

Fig. 13. Simulation (left) vs Actual Robot Movement (right) in X-Y Plane

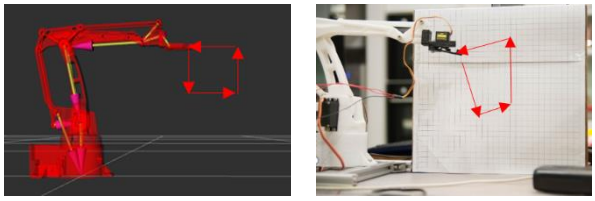

Fig. 14. Simulation (left) vs Actual Robot Movement (right) in X-Z Plane

V. Conclusion

An online robot programming framework was developed in the ROS environment. The computer has been used as a master node, which provided a GUI for the user to provide a target coordinate to move the robotic arm. The IK analysis for the robotic arm model was developed. The IK calculation was performed on the master PC in order to obtain the required joint variables to bring the robotic arm end-effector to the desired location. The visualization of the configuration of the robotic arm was provided in RViz to aid the use of teleoperation. The Raspberry Pi is used as the slave node, which is connected to the master node through a WLAN connection. The Raspberry Pi interfaced with and Arduino Uno board to control the servo motors on the robotic arm by PWM signals. In experimentation, the IK calculations were able to be performed in under 1 millisecond. The transmission of data across showed some losses but was able to preserve the joint variables data with up to 4 decimal point precision. The transmission speed per message took about 45 milliseconds each. However, the physical robot could not be controlled accurately due to physical limitations.

Acknowledgment

This work was supported by the University of Malaya Research University Grant (ST002-2019). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

[1] V. Roblek, M. Meško and A. Krapež, "A complex view of Industry 4.0", SAGE Open, vol 6(2), 2016, 2158244016653987.

[2] J.P. Barbosa, F.P.C. Lima, L.S. Coutinho, J.P.R.R. Leite, J.B. Machado, C.H. Valerio, G.S. Bastos, "ROS, Android and cloud robotics: How to make a powerful low-cost robot", int. Conf. on Adv. Robotics (ICAR) 2015, 27-31 July 2015, Istanbul, Turkey.

[3] J. Lentin, "Learning robotics using Python: Design, simulate, program, and prototype an interactive autonomous mobile robot from scratch with the help of Python, ROS, and Open-CV!", Community experience distilled (pp. 1 online resource (1 volume)). Retrieved from Knovel. Restricted to UCB, UCI, UCLA, UCM, UCR, UCSB, and UCSD http://uclibs.org/PID/277475, 2015.

[4] Z. Du, Y. Sun, Y. Su and W. Dong, "A ROS/Gazebo based method in developing virtual training scene for upper limb rehabilitation", IEEE Int. Conf. on Progress in Info. Computing, 16-18 May 2014, Shanghai, China, pp. 307-311.

[5] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, Weng, S., H. Deng, Y. Hu J. Zhang, "Manipulation task simulation using ROS and GazeboIEEE Int. Conf. on Robotics and Biomimetics (ROBIO 2014), 5-10 Dec 2014, Bali, Indonesia.

[6] R.K. Megalingam, V. Sivanantham, S.G. Kumar, P.S. Teja, S. K. Gangireddy and V.V. Gedela, "Design and development of inverse kinematics based 6 DoF robotic arm using ROS", Int. J. of Pure and App. Math., vol 118(18), 2018, pp. 2597-2603.

[7] S. Ergur and M. Ozkan, "Trajectory planning of industrial robots for 3-D visualization a ROS-based simulation framework", IEEE Int. Symposium on Robotics and Mfg. Automation (ROMA), 15-16 Dec 2014, Kuala Lumpur, Malaysia.

[8] A. Araújo, D. Portugal, M.S. Couceiro and R.P. Rocha, "Integrating Arduino-based educational mobile robots in ROS" 13th Int. Conf. on Autonomous Robot Systems, 24-24 April 2013, Lisbon, Portugal.