Alexa Controlled 3-DoF Articulated Robot using ROS

S B Shakthi Bala[1], Yuvan R S[2], and Anbarasi M P[3]
[1]Department of Robotics and Automation Engineering/PSG College of Technology
[21]Department of Robotics and Automation Engineering/PSG College of Technology
[31]Department of Robotics and Automation Engineering/PSG College of Technology

Abstract- This research paper presents the development and implementation of a simple yet effective Alexa voice-controlled 3-DOF (Degrees of Freedom) robotic arm. The power of Amazon's Alexa voice assistant and ROS (Robot Operating System) is used to enable seamless and intuitive control of the robotic arm through voice commands. The system demonstrates the integration of voice recognition, natural language processing, and robotics, making it accessible and user-friendly.

I. INTRODUCTION

In recent years, the field of robotics and automation has witnessed remarkable advancements, greatly influenced by a multitude of research endeavors aimed at improving the performance, control, and integration of robotic manipulators in various industrial and healthcare applications. Researchers and engineers have been exploring diverse avenues to enhance the capabilities of robotic systems. Rani et al. (2018) have contributed to this effort by developing intelligent tracking control mechanisms for redundant robot manipulators, a critical aspect in achieving precise and efficient manipulation in various applications. Robust control strategies for robotic manipulators have also been a focus of research. Massaoudi, Elleuch, and Damak (2019) have delved into robust control mechanisms for two degrees of freedom (DOF) robot manipulators, which are essential for ensuring stability and precision in complex tasks, from manufacturing to surgery. Hinrichs et al. (2023) have introduced an innovative robotic system designed to anchor patients in a lateral position, significantly reducing the physical strain on healthcare professionals. This development represents a pivotal step in enhancing healthcare processes and addressing the challenges faced by medical staff, particularly in the context of patient care and comfort. Moreover, the interaction between robotic manipulators and technological equipment has been a critical area of exploration. Kim, Zhidkov, and Polyansky (2021) have made notable contributions in this regard, focusing on organizing interactions between a robotic manipulator and a technological equipment control panel. This work not only emphasizes the importance of seamless communication between robots and equipment but also underscores the broader scope of research in industrial automation and smart manufacturing. Furthermore, Zhao, Pashkevich, Klimchik, and Chablat (2021) have presented a novel elastostatic modeling approach for multi-link flexible manipulators, based on a 2D dual-triangle tensegrity mechanism. Their research addresses the growing demand for precision and control in applications where flexibility and adaptability are paramount. These advancements underscore the dynamic and ever-evolving landscape of robotics research, reflecting the multidisciplinary nature of the field and the collaborative efforts to drive innovation across various domains. As we delve deeper into the specific contributions and innovations presented in these papers, it becomes evident that robotics and automation continue to be at the forefront of technological progress, with substantial potential to revolutionize industries and healthcare.

Following the references, this paper discusses about Ubuntu 20.04, ROS Noetic, VS Code, establishing a Catkin workspace, and configuring the robot model using a digital twin, various file types, including YAML, launch, Xacro, URDF, were explained. The explanation covered the setup of the robot's configuration, and the process of creating a custom PID controller. Additionally, both forward and inverse kinematics for the robot were addressed. The communication protocol was also a focal point, and integration with Alexa for voice-controlled robot operation was included. The setup of parameters for configuring the voice assistant was also described.

II. OPERATING SYSTEM USED

Ubuntu 20.04, codenamed "Focal Fossa," is a Long-Term Support (LTS) release of the popular Ubuntu Linux distribution. Released in April 2020, Ubuntu 20.04 Noetic serves as a reliable and stable platform for both desktop and server environments. This release combines the robustness of a mature operating system with modern features and extensive community support.

One of the key highlights of Ubuntu 20.04 is its use of the ROS (Robot Operating System) Noetic Ninjemys framework. ROS is widely used in robotics and automation research, making Ubuntu 20.04 a favored choice for roboticists and

researchers in these fields. ROS Noetic offers improved compatibility with Python 3, which is aligned with the broader industry shift toward Python 3 as the standard. This enhancement simplifies the development of robotic applications and accelerates the adoption of modern programming practices.

In addition to ROS Noetic, Ubuntu 20.04 includes a variety of software packages that cater to different user needs. The GNOME desktop environment provides a user-friendly interface for those on the desktop version, while the server edition offers a stable foundation for hosting web services, databases, and other server applications.

Security is a primary concern for any operating system, and Ubuntu 20.04 addresses this by integrating the latest security updates and features. It benefits from the Linux kernel's security enhancements, AppArmor profiles, and the ability to manage and apply updates easily through the Software Updater.

Ubuntu 20.04 LTS provides users with a consistent and reliable computing experience that is well-supported for five years. This makes it an excellent choice for organizations and individuals looking for a versatile, secure, and community-driven operating system. Whether developing robotics applications, running servers, or simply using it as daily driver, Ubuntu 20.04 Noetic offers a dependable platform to meet the needs. Its combination of stability, performance, and extensive software availability makes it a solid choice in the Linux ecosystem.

III. ROS NOETIC

ROS (Robot Operating System) Noetic Ninjemys is a significant milestone in the world of robotics and automation. Released in May 2020, ROS Noetic is the latest Long-Term Support (LTS) version of this open-source middleware framework. It brings a host of improvements, features, and compatibility enhancements that have solidified its position as a leading platform for developing and controlling robots.

One of the standout features of ROS Noetic is its embrace of Python 3. With the transition from Python 2 to Python 3, ROS Noetic aligns itself with modern software development practices, providing robotics engineers and researchers with access to the latest Python language features and libraries. This transition is crucial for ensuring that ROS remains relevant and well-supported within the broader software ecosystem.

ROS Noetic is designed to be versatile, accommodating a wide range of robots and applications. Whether working on autonomous vehicles, industrial manipulators, drones, or even space robots, ROS Noetic provides a robust and flexible framework for building, simulating, and controlling robotic systems. Its modularity allows developers to assemble the precise set of libraries and tools they need for their specific project, reducing unnecessary overhead.

The release also features improved support for cross-compilation, enabling developers to target a broader range of platforms and architectures. This is particularly valuable for robotics applications where resource constraints and varied hardware are common challenges. Furthermore, ROS Noetic maintains compatibility with previous versions, ensuring that existing codebases can transition smoothly to the latest version. This stability is crucial for industries and research institutions that have invested heavily in ROS-based projects.

III. ROS NOETIC INSTALLATION AND SETUP:

ROS Noetic can be downloaded from the official website of ROS.org. This website provides the required codes to install Noetic and down load other packages and required dependencies. Then a workspace has to be created. For this the following code snippet can be used:

mkdir ~p <workspace_name>_ws/src

Then the packages can be created with required libraries using the code:

catkin_create_pkg <package_name> roscpp rospy std_msgs

Then build the package using:

catkin_make

Then go into the workspace and source it using,

cd <workspace_name>_ws

source devel/setup.bash

This can be used to create new packages in the workspace.

IV. VS CODE

Visual Studio Code (VS Code) is an invaluable integrated development environment (IDE) for ROS (Robot Operating System) programming, offering a range of essential features that streamline and enhance the development process. Its robust ROS integration, through extensions like "ROS" and "ROS Preview," simplifies package management, node launching, and monitoring. Code autocompletion and IntelliSense improve coding efficiency and accuracy, while the integrated terminal allows seamless execution of ROS commands. VS Code's built-in Git support

facilitates version control and collaboration on ROS projects. Debugging tools, task automation, and code navigation make troubleshooting and code exploration effortless. Moreover, its extensibility and a vibrant developer community ensure ROS developers have access to tailored extensions and plugins. Finally, its cross-platform compatibility ensures ROS programmers on different operating systems can enjoy these benefits. In essence, VS Code is a powerful tool that empowers ROS developers to work more efficiently and effectively.

A. DIGITAL TWIN

In ROS (Robot Operating System), a digital twin is a virtual replica of a physical robot or robotic system. It accurately models the robot's behavior and components, facilitating simulation, testing, and algorithm development without the need for physical hardware access. Digital twins enhance safety, support educational endeavors, enable remote operation, and provide valuable insights for monitoring and optimizing robot performance. These virtual models are invaluable tools in ROS, empowering engineers and developers to create and enhance real-world robots efficiently and reliably. A digital twin model can be created using URDF file.

B. URDF AND XACRO FILE

The URDF (Unified Robot Description Format) file is a fundamental component in the world of ROS (Robot Operating System) for describing the physical properties and kinematic structure of robotic systems. It serves as a standardized XML-based format that defines a robot's links, joints, sensors, visual and collision properties, and other critical characteristics. URDF files are pivotal for robot modeling, visualization, and simulation in ROS.

Converting a URDF file into an XACRO file is a common practice in ROS to enhance modularity and maintainability. XACRO (XML Macros) is an extension of URDF that introduces powerful features like macros, which allow for the reuse of robot descriptions and simplified parameterization. XACRO files are written in XML and include the URDF definitions along with macros, making them more versatile and human-readable.

This conversion process simplifies robot descriptions, reduces redundancy, and promotes code reuse, making robot modeling and maintenance more efficient and comprehensible in ROS environments. It's a valuable technique for ROS developers seeking to create modular and reusable robot descriptions.

C. PARAMETER SERVER:

In ROS Noetic Ninjemys, the "parameter server" is a vital component facilitating the management and sharing of configuration data among nodes within a robotic system. It acts as a centralized repository for storing parameters such as robot dimensions, sensor calibrations, and control gains. Parameters are organized into namespaces, streamlining their management, particularly in complex robotic setups. The parameter server offers dynamic reconfiguration capabilities, allowing nodes to modify parameters in real-time, enabling on-the-fly adjustments without requiring node restarts. With dedicated APIs and integration with ROS launch files, parameter management is seamless. It serves as a cornerstone for configuring and coordinating ROS nodes, enhancing the efficiency of robotic applications.

D. RVIZ:

RViz, short for "ROS Visualization," is an indispensable tool within the ROS (Robot Operating System) ecosystem, particularly when integrated with ROS Noetic Ninjemys. This 3D visualization tool plays a pivotal role in the development and troubleshooting of robotic systems, providing real-time insights into robot states, sensor data, and environments. It enables users to visualize intricate robotic scenarios, assess robot performance, and debug behaviors, whether in simulation or real-world settings. RViz excels in displaying robot states, sensor data, and occupancy grid maps, making it crucial for tasks like mapping, navigation, and sensor evaluation. Its support for interactive markers facilitates teleoperation and simulation adjustments, while customization options and extensibility allow users to tailor it to specific needs. RViz's integration with ROS Noetic amplifies its utility, aiding in the early identification of issues and the refinement of control strategies, ultimately accelerating the development of reliable robotic systems.

E. GAZEBO:

Gazebo is a powerful and versatile open-source robotics simulator that plays a pivotal role in the world of robotics, particularly when integrated with ROS Noetic Ninjemys. This simulation environment offers a realistic and highly configurable platform for designing, testing, and validating robotic systems, making it an indispensable tool for robotics researchers, developers, and enthusiasts.

One of the key strengths of Gazebo is its seamless integration with ROS Noetic. This integration allows for the development and testing of robotic algorithms and control strategies within a simulated

environment that mirrors the real world. ROS Noetic's extensive libraries and tools, when combined with Gazebo, enable the creation of complex and accurate robot simulations.

Gazebo supports a wide range of robotic platforms, sensors, and environments. Users can model and simulate everything from simple mobile robots to intricate multi-robot systems, aerial drones, and even humanoid robots. The ability to replicate diverse real-world scenarios in a controlled virtual environment is invaluable for refining algorithms, optimizing sensor configurations, and ensuring that robots operate safely and effectively.

Furthermore, Gazebo provides various sensor plugins that mimic real-world sensors, including cameras, lidars, and IMUs, among others. These plugins enable users to develop and validate perception and navigation algorithms before deploying them on physical robots. This reduces the risk of errors and enhances the overall efficiency of the development process.

Gazebo's physics engine accurately simulates the interactions between robots and their environments, accounting for factors such as gravity, friction, and collisions. This realism is essential for testing the stability and robustness of robot control systems and allows developers to identify and address potential issues early in the development cycle.

F. LAUNCH FILES:

In ROS (Robot Operating System), a launch file is a crucial configuration file used to start and manage various ROS nodes, set parameters, and configure system behavior. It provides a streamlined and reproducible way to launch multiple nodes and configure their interactions. Launch files simplify the initialization of complex robotic systems, allowing developers to specify node names, packages, parameters, and arguments. By defining launch dependencies and configurations, users can ensure that ROS nodes are launched in the correct sequence with the desired settings, making launch files an essential tool for orchestrating the behavior of robotic applications in a structured and efficient manner.

G. ROS TIMER:

A ROS timer is a crucial component for scheduling and executing tasks at predefined intervals within ROS (Robot Operating System) nodes. It allows developers to automate periodic operations, such as sensor data processing or control loop execution. When creating a timer, users specify the callback function to run at a specified frequency or delay. Timers help maintain real-time behavior and ensure precise execution of tasks, making them invaluable for robotics applications. They can be started, stopped, and set to single-shot mode as needed. ROS timers play a pivotal role in enabling time-sensitive actions and controlling the timing of various processes in robotic systems.

H. ROS SERVICES:

ROS services provide a communication mechanism in the Robot Operating System (ROS) for requesting and receiving specific, stateless functionalities between nodes. Nodes can offer services, defining a set of functions that can be invoked by other nodes. These services typically take requests with specific inputs, process them, and send back responses. ROS services ensure synchronous, point-to-point communication, enabling nodes to request tasks like sensor data retrieval or robot actions. The requesting node blocks until it receives a response or a timeout occurs. This mechanism is valuable for tasks requiring intermittent interactions, such as configuration changes, diagnostic checks, or custom functionalities in a ROS-based robotic system. Thus, a service server called the Angles Converter.srv for angle conversions between ROS and motors has been created.

I. YAML FILE:

A YAML file is a human-readable data serialization format used to structure and represent data hierarchies in a simple and concise manner. YAML files use indentation to define data structures, making them easy for both humans and machines to understand. They are commonly used for configuration files, data exchange between programs, and in various applications like configuration management, data persistence, and more. YAML's readability and versatility make it a popular choice for settings, configurations, and structured data storage, and it has wide support in programming languages and software tools, making it an efficient way to manage and communicate data in a clear and organized format.

These are the two YAML files

- trajectory_contollers.yaml
- joint_state_controller.yaml

which performs the motion control and controlling of states of the joints respectively.

IV. JOINT ANGLE CONVERSION:

The following equations allows to convert the angles of the joints of the robot from the ROS convention in radians to the Arduino one that is in degrees. This conversion is useful to actuate the servomotors of the robot and retrieve its position.

A. *Motor Angle to ROS Angle Conversion:*

- Base Angle in ROS ($baseradians$) = π * (Base Angle in Motor - π) / (2 * 180)
- Shoulder Angle in ROS ($s\square oulderradians$) = 180 - π * (Shoulder Angle in Motor + π) / (2 * 180)
- Elbow Angle in ROS ($elbowdians$) = π * (Elbow Angle in Motor - π) / (2 * 180)
- Gripper Angle in ROS ($gripperdians$) = -π * Gripper Angle in Motor / (2 * 180)

B. *ROS Angle to Motor Angle Conversion:*

- Base Angle in Motor = (π * Base Angle in ROS - π) / (2 * 180)
- Shoulder Angle in Motor = 180 - (Shoulder Angle in ROS * π + π) / (2 * 180)
- Elbow Angle in Motor = (π * Elbow Angle in ROS - π) / (2 * 180)
- Gripper Angle in Motor = - (π * Gripper Angle in ROS) / (2 * 180)

These formulas are used in the anglesconverter.cpp file.

V. CONTROLLER:

The motors responsible for actuating the robotic joints play a crucial role in achieving desired robot movements through commanded actions. This process is known as control and forms the basis of a mathematical field of study involving controllers. The primary objective is to understand techniques for defining a variable termed "command," which serves as input to a system, such as a motor. In practice, this command represents a target, like specifying that a motor's axis should rotate at one radian per second. The control system then orchestrates the motor's motion to meet this target. The input, or target, is compared to the current system state, such as the motor's actual speed. Consequently, the disparity between the output (current state) and the input (desired state) results in a variable known as "error," indicating the extent to which the current state deviates from the desired state. This error guides the control system's actions. A PID controller is used in this for precise controlling of the joints.

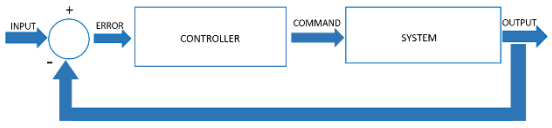

Fig. 1. General block diagram of controller

A launch file control.launch is used to control the movements of joints. This ROS launch file is a pivotal configuration script for a robotic system. It conditions component execution based on the system mode (simulation or real hardware) using the "is_sim" argument. It loads controller configurations from YAML files, sets the "robot_description" parameter using URDF, and handles angle conversions with the "angles_converter.py" node. In real hardware mode, it initiates the "arduinobot_interface_node" for hardware communication. Additionally, it spawns controllers, ensuring proper joint state transformations. This versatile file streamlines ROS-based robotic system setup, catering to various scenarios with efficiency.

VI. KINEMATICS OF ARTICULATED ARM:

There are two purposes to find kinematics of an articulated arm, which are:

- To determine the position of the gripper by knowing angle of each joint which is known as forward kinematics.
- To calculate angle knowing position of the gripper in the space which is known as Inverse kinematics.

A. *FORWARD KINEMATICS*

In order to move robot by forward kinematics, A python package known as tf is used to calculate link transformation matrix. A Transformation matrix of base frame with respect to inertial frame (end effector frame) is required in order to solve forward kinematics problem.

Since it is not possible at one shot, let us break down into pieces where Transformation matrix of each link with respect to previous link is found and then multiplied in order to obtain the total transformation matrix of base frame with respect to inertial frame.

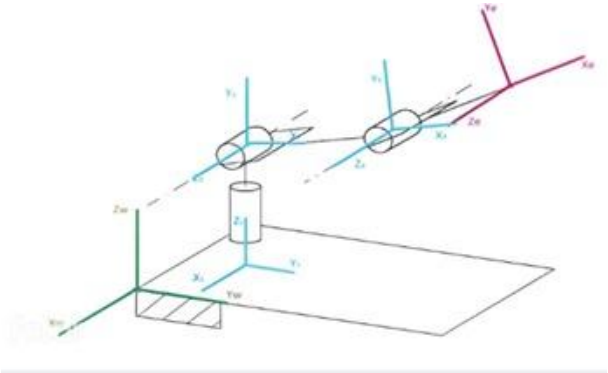

Fig. 2. Kinematic diagram of the robot with local frame representation

The total transformation matrix can be found by,

$$T = T_{w1}(q1)\, T_{12}(q2) + T_{23}(q3)\, T_{3e}$$

where,

T_{w1}- Transformation matrix of frame 1 with respect to world frame

q1 - Angle made by joint 1

T_{12}- Transformation matrix of frame 2 with respect to frame 1

q2 - Angle made by joint 2

T_{23} - Transformation matrix of frame 3 with respect to frame2

q3 - Angle made by joint 3

T_{3e} -Transformation matrix of end effector frame with respect to frame3

The skeletal framework of all the co-ordinate frames associated with each of joints is in RVIZ as,

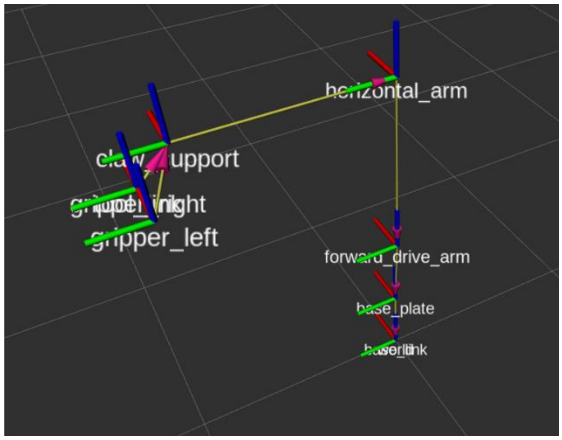

Fig. 3. Kinematic diagram visualization in Rviz

This transformation matrix is found automatically and joint angles are updated through feedback by means of Tf package, for visualization, RVIZ software is used for publishing the results of user input through robot state publisher which is subscribed to Tf package. By using the command "ros topic echo /tf", One can visualize the changes occurring on Rotational and Transformation matrix of the arm as it is moved.

Fig. 4. Graph of nodes in use

B. INVERSE KINEMATICS

In order to move robot through inverse kinematics, The end effector position is given on the 3-dimensional space and the joint angles are calculated. A library known as "MOVE GROUP" is used for exchange of functionalities and make them available to the user. There can actually be several configurations of the joints to bring gripper on the same position. This converges to a point where direct kinematics is ambiguous to have a unique solution, while inverse kinematics have multiple solutions, Infinite solution as well as no solution.

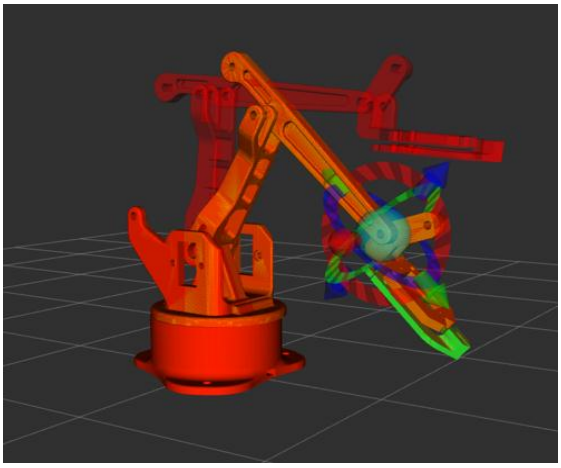

Fig. 5. Robot visualization in Rviz

Fig. 6. Robot visualization in Gazebo

Instead of solving this problem with set of equations, optimizers are used that recursively look for the best solution that solve inverse kinematics problem. For this purpose, a software "MOVE IT" which offers solution to solve inverse kinematic, Trajectory planning and obstacle avoidance problem is used, URDF model of the robot is provided to use "MOVE IT" and it will be able to solve inverse kinematic problem and the gripper can be able to move in three-dimensional space.

In this case a goal is assigned to the gripper and "MOVE IT" will calculate trajectory that each joint has to execute in order to move the gripper the current position to the desired position, by avoiding obstacles.

The core component of “MOVE IT” is “MOVE GROUP” that manages all functionalities and the interfaces of ROS. The purpose of this node is to enable communication between all the “MOVE IT” modules and allow the exchange of messages between all the functionalities and make them available to the user via a graphical interface and also via a programming interface or API.

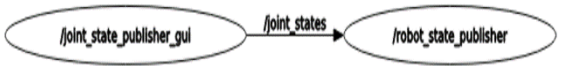

Fig. 7. Graph of nodes in use

VII. COMMUNICATION PROTOCOL USED ON ROS:

A. PUBLISHER SUBSCRIBER PROTOCOL

The Publisher-Subscriber pattern is a fundamental communication paradigm used in the Robot Operating System (ROS), which is a popular middleware framework for building robotic systems. This pattern allows different parts of a ROS system to communicate with each other in a distributed and decoupled manner. Here's an overview of how the Publisher-Subscriber protocol works in ROS:

Publishers: Publishers are components in a ROS system that produce data (e.g., sensor data, robot state information, camera images) and make it available to other parts of the system. Publishers encapsulate the data and send it to a specific topic. Each topic represents a particular type of data, and multiple publishers can publish data to the same topic.

Subscribers: Subscribers are components that want to receive and process data from specific topics. Subscribers subscribe to one or more topics and receive data whenever a publisher publishes new information to those topics. Subscribers can process data in real-time or store it for later analysis.

Topics: Topics are named communication channels in ROS. They act as message buses where data is published and subscribed to. Topics are organized by their data type, and multiple publishers and subscribers can interact with the same topic as long as they use compatible data types.

Messages: Data exchanged between publishers and subscribers are packaged into messages. Messages are structured data formats defined using the ROS message description language. Each message type corresponds to a specific topic, and publishers and subscribers must agree on the message type to exchange data correctly.

B. SERVICE-SERVER PROTOCOL

In the Robot Operating System (ROS), the Service-Server pattern is another communication paradigm that complements the Publisher-Subscriber pattern. While the Publisher-Subscriber pattern is used for asynchronous communication, the Service-Server pattern is used for synchronous, request-response-style communication between ROS nodes. Here's an overview of how the Service-Server protocol works in ROS:

Service Server: The Service Server is a ROS node that provides a specific service. A service is essentially a named function with a defined input (request) and output (response). The service server registers itself with a specific service name.

Service Client: The Service Client is another ROS node or component that requests a service by sending a service request to the service server. The service client specifies the service name and provides the necessary request data.

Service Request: The service request is a message (data structure) that contains the input data for the service. It is sent from the service client to the service server when the client requests the service.

Service Response: The service response is a message (data structure) that contains the output data generated by the service server in response to the client's request. The service server sends the response back to the service client.

C. ACTION SERVER & CLIENT PROTOCOL

In the Robot Operating System (ROS), the Action Server protocol is a communication pattern designed for handling long-running, asynchronous tasks and actions. It provides a more advanced and flexible way to manage actions that can take a significant amount of time to complete and need to provide feedback on their progress. The Action Server pattern extends the capabilities of ROS beyond the simple Publisher-Subscriber and Service-Server patterns. Here's an overview of how the Action Server protocol works in ROS:

Action Server: The Action Server is a ROS node that is responsible for managing and executing a specific action. Each action type corresponds to a unique action server. An action server registers itself with a specific action name.

Action Client: The Action Client is another ROS node or component that initiates an action by sending a request to the action server. Unlike services, action clients can preempt and cancel ongoing actions, and they can receive periodic feedback on the progress of the action.

Action Goal: The Action Goal is a message (data structure) that the action client sends to the action server to initiate an action. It contains the necessary input data and specifications for the action.

Action Result: The Action Result is a message (data structure) that the action server sends back to the action client when the action is successfully completed. It contains the final result of the action.

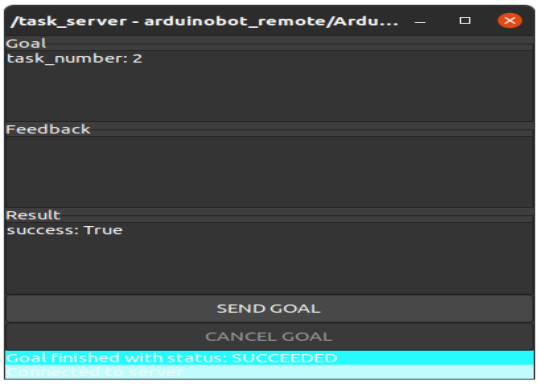

Fig. 8. Task Server

Action Feedback: The Action Feedback is a message (data structure) that the action server periodically sends to the action client to provide updates on the progress of the action. This allows the client to monitor the ongoing process.

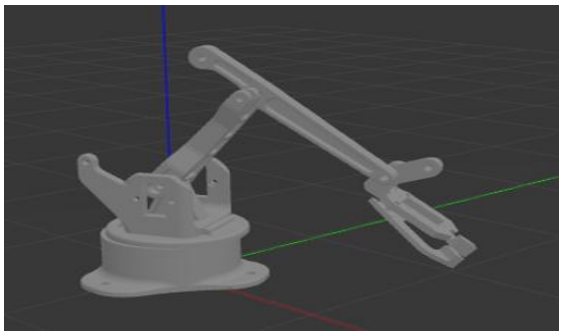

Fig. 9. Robot moves when task server sends a command

IX. ALEXA INTEGRATION:

The robot used on industries are very often controlled and moved by a very large and difficult joysticks, especially during the learning of the movements, this way of moving robot is suitable for an industrial environment where dust, water and strict security measures requires the robot to be moved with these kinds of devices. However, the world witnessing a transition from the industrial to domestic robotics and in this scenario the robots are shifting from having a safe and secure space reserved only for their angling to share their space with human. In this new scenario where robots interact with humans, Voice might be the better tool that allowed us to communicate with one another.

Through voice one can communicate their intension or can request the execution of some tasks and also get some feedback from the robot. And this increase the awareness about the movements that the robot is going to make among all the available voice assistant. “Amazon Alexa” is used because it offers the possibility to create or add new functionalities and behaviours also called skills to voice assistant. Also, for developing such functionalities Amazon provides some very intuitive API (Application Program Interface) and also a GUI (Graphical user interface) for testing these functionalities.

Amazon Alexa developer web is used to create new skills, new behaviour of the assistant and using this one can trigger the execution of the task of the robot with the voice, before that one need to connect the robot to the internet so that it is reachable from the assistant, and to do so, a very simple and useful tool called “NGROK” is used and this tool allows us to open a HTTP port on the PC to the internet. It will also provide a public address where the robot, the PC is reachable in this way in a computer.

A. SPECIFYING PARAMETERS FOR VOICE ASSISTANT:

Create an Amazon account and then login to “Amazon developer console” and Create a New skill and specify a trigger word, which in this case is “Activate the robot”

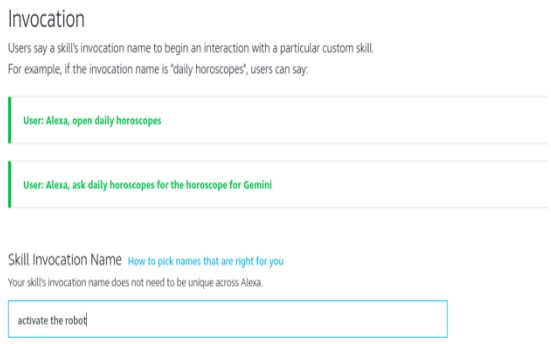

Fig. 10. Invocation set-up

Now specify the intents such as “Wake_intent”, “Pick_intent”, “Sleep_intent”, Which in this case is specified as follows:

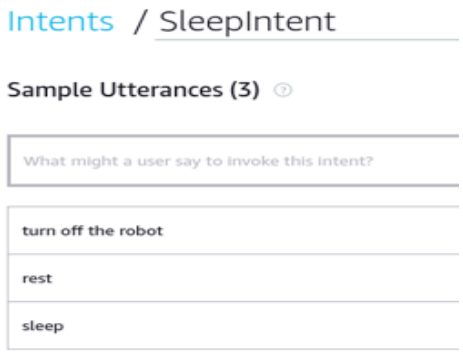

Fig. 11. Setting up SleepIntent

Fig. 12. Setting up WakeIntent

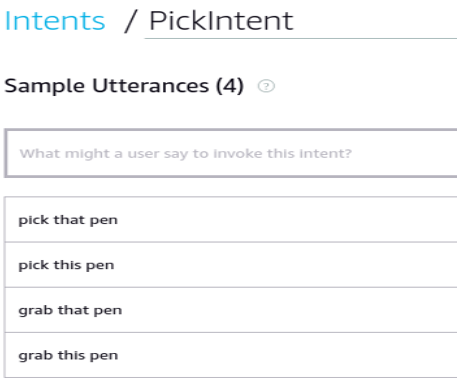

Fig. 13. Setting up PickIntent

The API is specified through an endpoint to access the robot's controls.

By this way the integration of Alexa can be possible with the articulated arm.

Fig. 14. Setting up Endpoints

X. CONCLUSION:

In conclusion, this research showcases the successful creation of an Alexa voice-controlled 3-DOF robotic arm, bridging the gap between voice assistants and robotics. The system's performance, as validated through experiments and user feedback, demonstrates its reliability and practicality. This innovation has the potential to transform the way people interact with robotic systems, offering enhanced accessibility and ease of use. As voice-controlled technology continues to advance, this work contributes to the growing field of human-robot collaboration, opening doors to a wide range of applications that benefit from intuitive voice-based control in robotics.

REFERENCE

[1]. Smith, C., & Christensen, H. I. (2009). IEEE Robotics & Automation Magazine.
[2]. Rani, M., Ruchika, & Kumar, N. (2018). Intelligent Tracking Control of Redundant Robot Manipulators including Actuator Dynamics.
[3]. Kim, N. V., Zhidkov, V. N., & Polyansky, V. V. (2021). Organizing Interactions Between a Robotic Manipulator and a Technological Equipment Control Panel.
[4]. Yang, X., Xu, Z., Zhang, W., Zhang, W., & Liu, P. X. (2021). Model-free control of manipulators in task space containing mismatched uncertainty.
[5]. Hinrichs, P., Seibert, K., Gómez, P. A., Pfingsthorn, M., & Hein, A. (2023). A Robotic system to anchor a patient in a Lateral Position and Reduce Nurse's Physical Strain.
[6]. Massaoudi, F., Elleuch, D., & Damak, T. (2019). Robust Control for a Two DOF Robot Manipulator.
[7]. He, W., Li, H., Wang, Y., & Liu, S. (2020). Suppression of the Disturbance of Robotic Manipulators Based on Nonlinear Disturbance Observer and Fuzzy Logic System.
[8]. Zhao, W., Pashkevich, A., Klimchik, A., & Chablat, D. (2021). Elastostatic modeling of multi-link flexible manipulator based on 2D dual-triangle tensegrity mechanism.
[9]. Alonso, R., Concas, E., & Recupero, D. R. (2021). An Abstraction Layer Exploiting Voice Assistant.
[10]. Yan, Q., Cai, J., Zhang, Y., & Yang, Z. (2021). Adaptive Iterative Learning Control for Robot Manipulators with Time-Varying Parameters and Arbitrary Initial Errors.