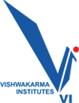
## Department of Electronics Engineering Vishwakarma Institute of Technology, Pune



Course Project Report

Internet of Things

**Title: Robot Arm Manipulation using MQTT**

**BATCH: A3**

**GROUP MEMBERS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Div.** | **Roll No.** | **G.R. No.** | **Name** |
| A | 51 | 11810435 | Utkarsh Jakate |
| A | 52 | 11810570 | Chinmay Jangle |

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Content** | **Page No** |
| 1 | Introduction | 2 |
| 2 | Theory | 2 |
| 3 | Methodology/Workflow | 3 |
| 4 | Block Diagram | 5 |
| 5 | Software Requirements | 5 |
| 6 | Setting up of ROS Project | 6 |
| 7 | Running the Simulation | 8 |
| 8 | Setting up of App Part | 11 |
| 9 | Demo Video | 16 |
| 10 | Conclusion | 16 |

**Introduction:**

The industrial UR5 arm developed by Universal Robotics has got itself in many different industries to automate processes done by humans. These robots can be deployed in harsh environments and can work tirelessly for a long duration with great efficiency. Industries use these arms tremendously. While working in remote areas, these robots are controlled remotely and thus industry uses different IOT protocols to connect and operate these robots.

In this Project we aim to control an Industrial UR5 Robotic Arm via a Mobile App using MQTT Protocol. To achieve this, we will be simulating the Robotic Arm in Gazebo Environment and would be controlled by ROS (Robotic Operating System).

**Theory:**

* **ROS:** The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. Robot Operating System (or ROS) is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. It supports Linux OS.
* **Gazebo:** Gazebo is an open-source 3D robotics simulator. It can access multiple high-performance physics engines including ODE, Bullet, Simbody, and DART. Gazebo also can Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque.
* **HiveMQ:** Itis a MQTT broker - a messaging platform for fast, efficient, and reliable data movement to and from connected IoT devices and enterprise systems. It provides a fast, low-overhead, high-throughput and modern MQTT library for Java by building on modern frameworks. We will be using open source free of cost public broker.

Another very interesting feature of the HiveMQ MQTT Client is its backpressure handling.

* **MQTT Protocol:** The Message Queuing Telemetry Transport is a lightweight, publish-subscribe network protocol that transports messages between devices. The protocol usually runs over TCP/IP; however, any network protocol that provides ordered, lossless, bi-directional connections can support MQTT.

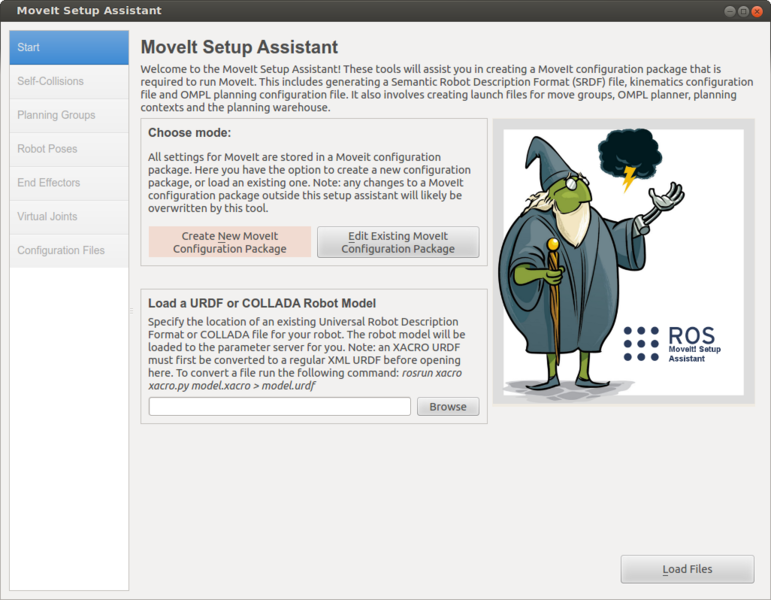
MQTT is a publish/subscribe protocol that allows edge-of-network devices to publish to a broker. Clients connect to this broker, which then mediates communication between the two devices. Each device can subscribe, or register, to particular topics.

* **Android Studio:** Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. Using an Android App, we will try to control the simulation in ROS.
* **Paho Android Service:** The Paho Android Service is an interface to the Paho Java MQTT Client library for the Android Platform. The MQTT connection is encapsulated within an Android-Service that runs in the background of the Android application, keeping it alive when the Android application is switching between different Activities. This layer of abstraction is needed to be able to receive MQTT messages reliably. As the Paho Android Service is based on the Paho Java client library it can be considered stable and used in production. The project is actively maintained by the Eclipse Paho project.

**Methodology / Workflow:** -

1. In our project we have used ROS Melodic and Ubuntu 18.04. ROS supports Python and C++ Programming Language. We will be using Gazebo9 for our project environment.
2. Firstly, we have a URDF model of UR5 Arm to simulate in Gazebo. It is a 6-degree Robotic Arm used for industrial level applications. This package contains a C++ parser for the Unified Robot Description Format (URDF), which is an XML format for representing a robot model.
3. Create a MoveIT package using this URDF model for Robot Manipulation. It is a motion planning framework in ROS for planning paths for our robotic arm. Package comes with different planners for simulation. Using the MoveIT setup assistant, it auto generates different ROS files for simulation. MoveIT setup assistant can be launched by typing:

**roslaunch moveit\_setup\_assistant setup\_assistant.launch**



1. Here are steps to generate a ROS Package in ROS Melodic.

Details of these steps are well documented in their documentation:

<http://docs.ros.org/en/melodic/api/moveit_tutorials/html/index.html>

1. Usage of ROS for simulation, create different python script files for Arm manipulations. These scripts contain code for sending joint angles to planner node, subscribing to MQTT data, receiving logs from Gazebo, etc.
2. We have created a launch file which will run all python scripts (called ROS nodes) together along with Gazebo environment. ROS launch files help in running multiple files required for simulation along with main ROS server. A ROS node will constantly subscribe to a given MQTT Topic.
3. We have an android app which works as a client server which gets connected to MQTT Broker and via the MQTT protocol sends coordinates to the robotic arm using the paho android service.
4. Two different topics are used to send data to robotic arm, one is of joint angles and other is of gripper.
5. Here in the android application, we are planning to take the joint angles as an input from the user which will be sent to the arm. So, the arm will work according to the value assigned for that received command in the python script attached.

**Block Diagram:**



Publish Data for Arm to Broker



ROS and MoveIt package will give necessary instructions to Arm to move.



ROS Nodes will subscribe to MQTT Topic.

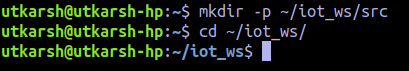
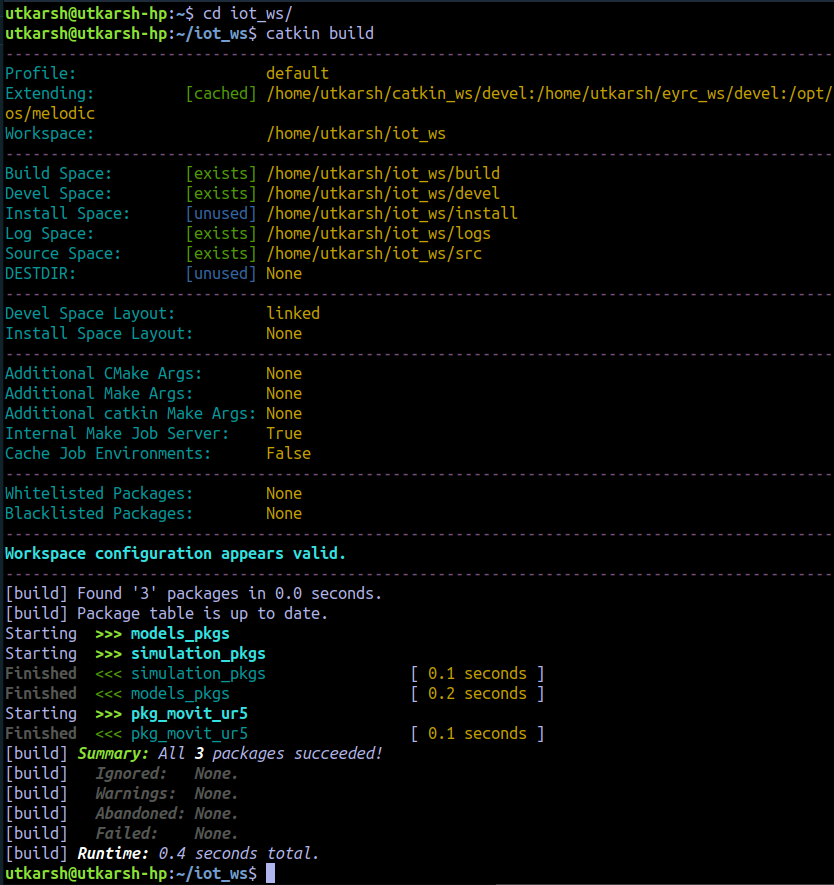


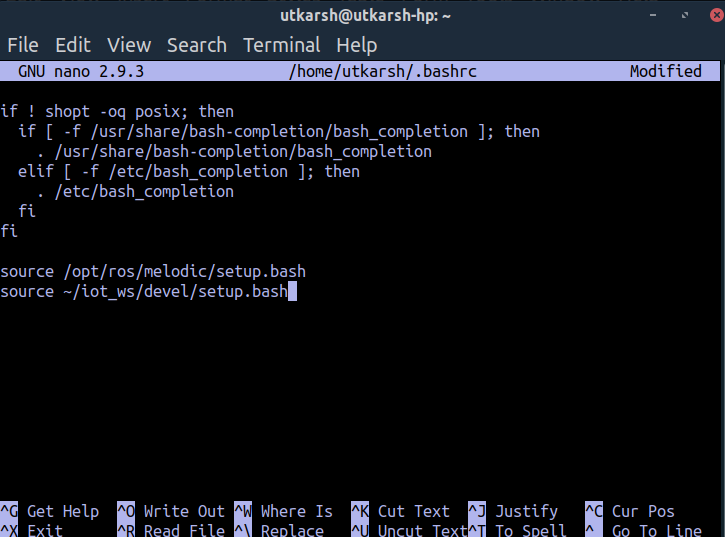
**Required Software:**

**Software Requirements:**

1. <https://releases.ubuntu.com/18.04/>, for installing Ubuntu 18.04.
2. <http://wiki.ros.org/ROS/Installation>, for installing ROS and Gazebo with documentation and tutorials.
3. <http://wiki.ros.org/rviz/UserGuide>, for Rviz.
4. <https://moveit.ros.org/moveit!/ros/2018/05/23/firstmelodicrelease.html>, MoveIT packages with documentation.
5. <https://developer.android.com/studio?gclsrc=aw.ds&&gclid=Cj0KCQjw4cOEBhDMARIsAA3XDRh_URCGVgczBH-190lmh8pApvgEfdo3IU3HOSSoWf5ZSA_-L5r4djoaAqqsEALw_wcB> for installing Android Studio.
6. **Git link for Project:** <https://github.com/UtkarshJakate/IOT_CP_ROS.git>
7. **Git link for App:** <https://github.com/LLawliet19/IOTCPmqttapp>

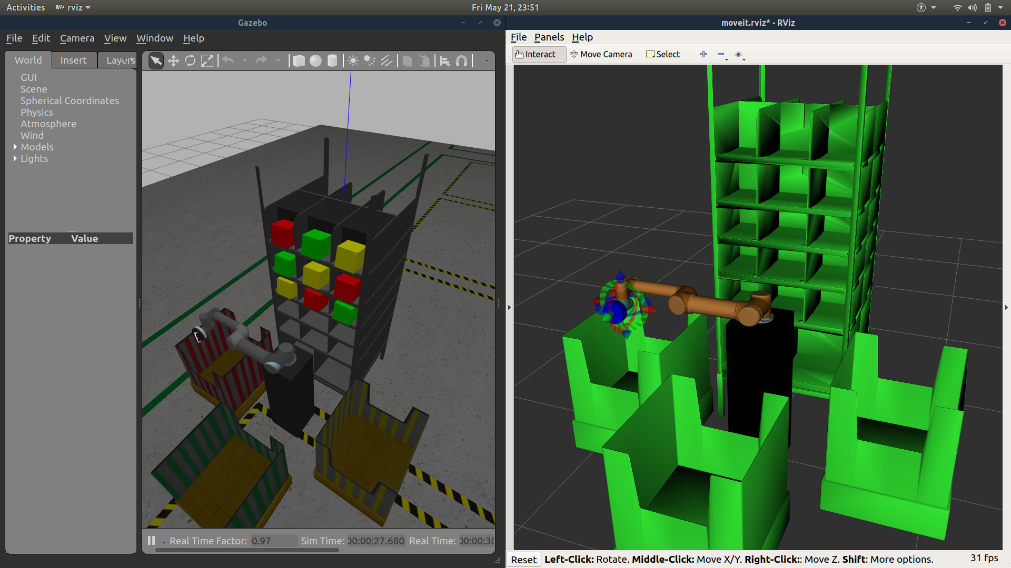
**Setting up of ROS Project:**

1. ROS setup and installation steps are given [http://wiki.ros.org](http://wiki.ros.org/). Create a workspace **mkdir -p ~/iot\_ws/src** (I have used iot\_ws as my workspace name, you can have your own name). Then **cd ~/iot\_ws/**
2. To build workspace you have to options **catkin\_make** (older, mostly shown in documentation) and **catkin\_build** (better, used by me).
3. Go in src folder,
4. Here clone the files which are there in the git folder. Using **git clone** command.
5. On doing running **ls** command you will see three folders
6. These are the folders required for simulation.
7. There may be error while simulation showing joint\_state\_publisher missing, in that case use **sudo apt-get install -y joint-state-publisher** to install the dependency.
8. Now build the project. Use **catkin build**. (use should be in **~/iot\_ws.** The output should be like the following with no errors in building.
9. Come to your home directory (use just **cd**) and add **nano ~/.bashrc.**
10.  Here, add **source ~/iot\_ws/devel/setup.bash** in the file.
11. Ctrl+X to exit, say yes to modified data and the file is saved. Setting of Ros Project is done.

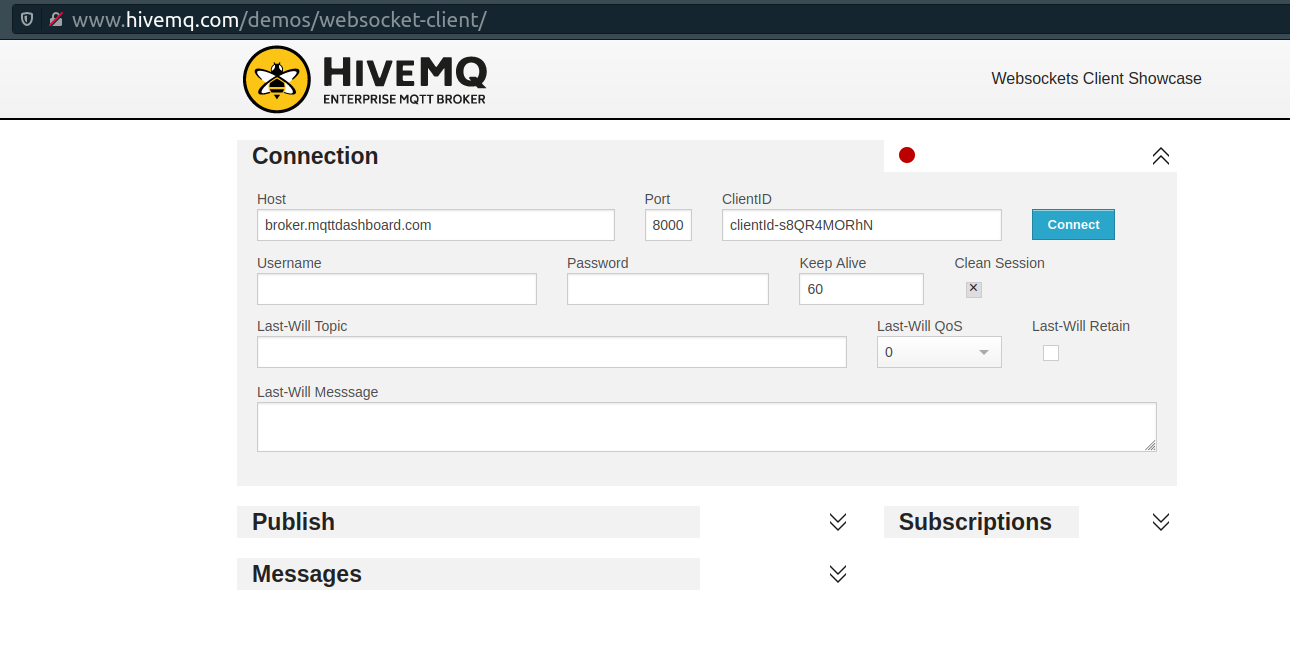


**Running the Simulation**

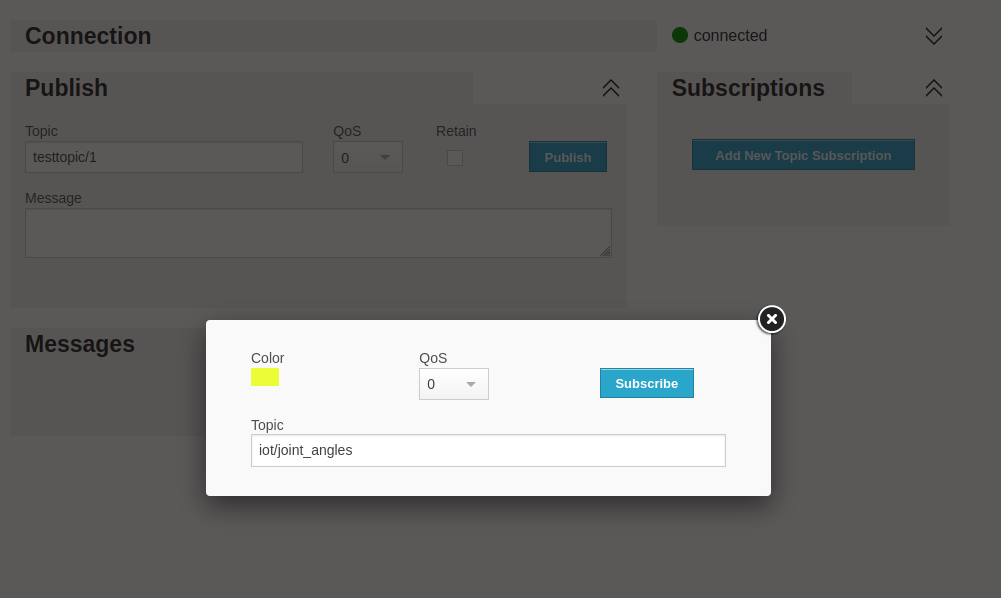
1. First, we need to run the Gazrebo and Rviz file. For the same we need to launch the .launch file which can run the roscore (start ros server) and other important files like gazebo file, rviz file, rosnodes (here the boxes are spawned using a rosnode), etc. Command for this is **roslaunch simulation\_pkgs ur5\_simulate.launch .**
2. Gazebo and Rviz screen will open, it may take some time to load all the files.



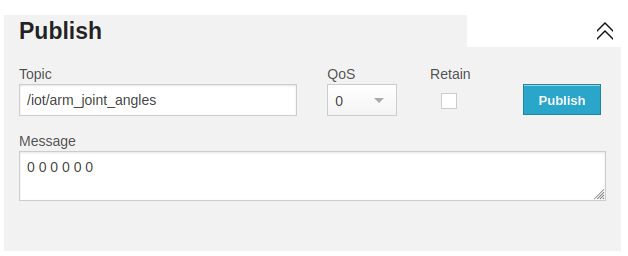
1. In a browser, search <http://www.hivemq.com/demos/websocket-client/> for publishing or subscribing to a MQTT Topic. Click on Connect.



4. Now, in the project we are using 2 rostopic, one for vaccum gripper and on for joint\_angles. Their topic names are **/iot/vacuum\_gripper** and **/iot/arm\_joint\_angles.** You can add new subscription here in broker.

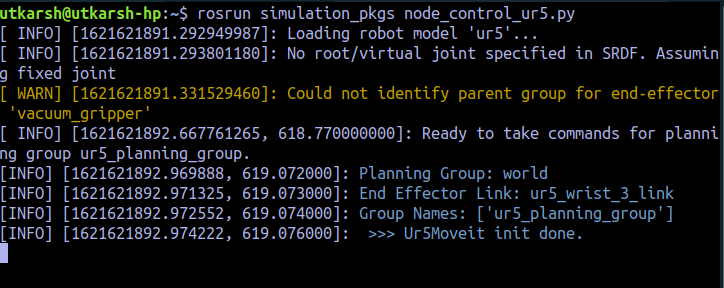


5. For Publishing data to a topic, add topic name in Publish section and add the required message in Message section.



6. Get new tab on terminal and run different rosnode files.

* **rosrun simulation\_pkgs node\_control\_ur5.py:** For MoveIT ROS controller of Arm. This is the main code which control the arm in Gazebo.



* **rosrun simulation\_pkgs subscriber\_for\_vaccum\_gripper.py:** Subscribe iot/vaccum\_gripper.

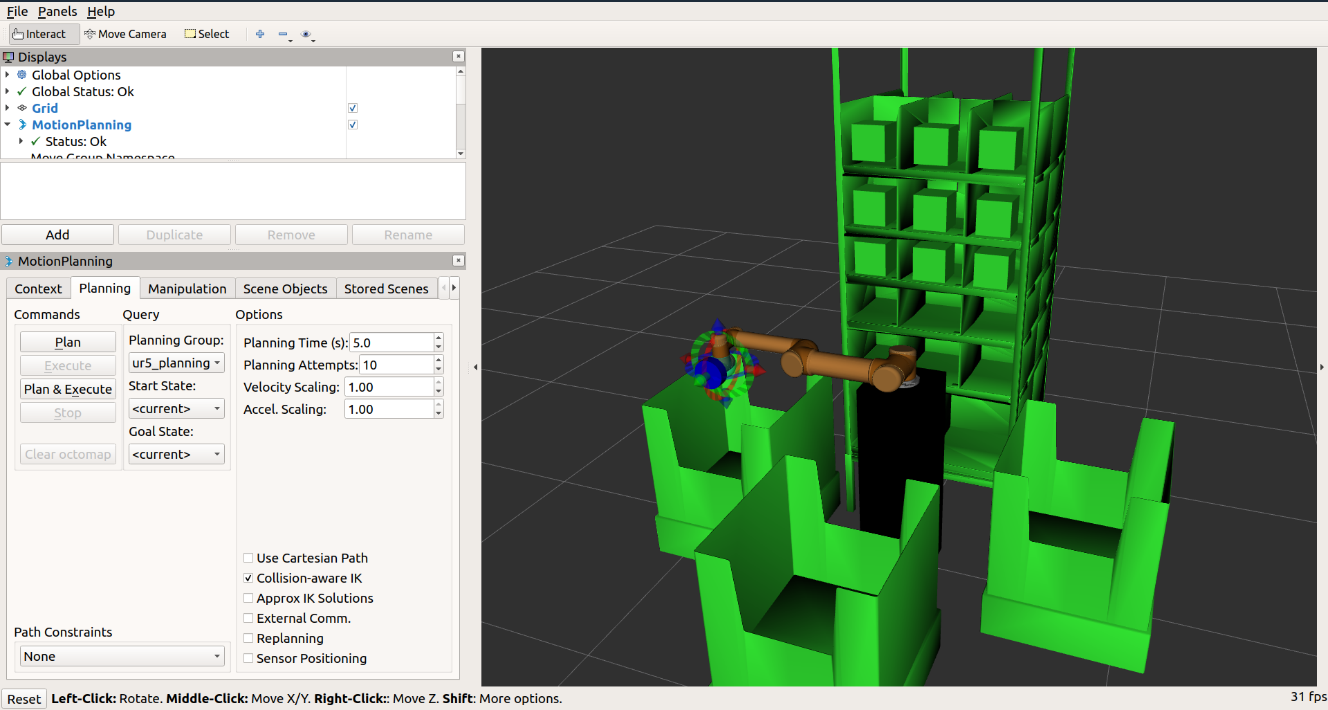


* **rosrun simulation\_pkgs subscriber\_for\_joint\_angles.py:** Subscribe /iot/arm\_joint\_angles.



7. All Setup is done for the project.

8. Rviz can be used to see motion planning of Arm. The planning section can help you control arm without rosnode. The arrows and discs show at end effector of arm are used to fix end location of arm. It can be used to see arms movement.



**Setting up of App Part**

* 1. Clone the Git link provided for Android App.
  2. Dealing with MQTT protocol on Android is quite easy. MQTT works on TCP/IP stack, this means that the only requirement of the mobile device is the capability to connect to the internet.
  3. There are only few Android libraries which implements MQTT protocol, but fortunately they are well documented and work fine.

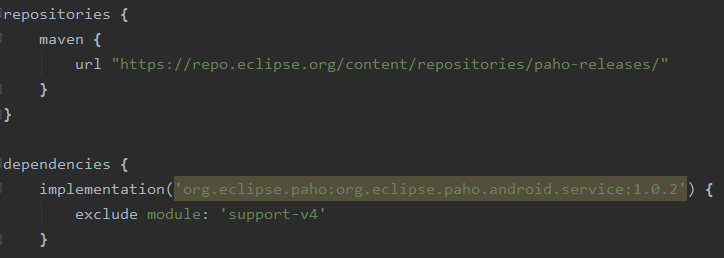
The most famous libraries are:

* Eclipse Paho Android Service
* HiveMQ MQTT Client library

I have chosen the Eclipse Paho library because it’s the most used one and it’s implemented in different programming languages and platforms.

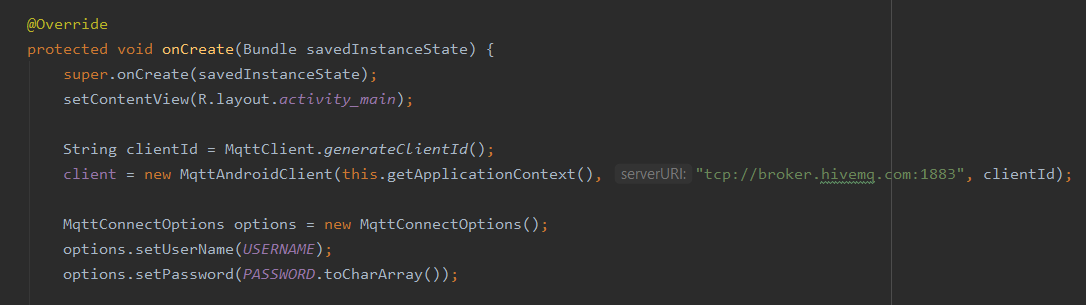
**Steps to import Paho library in Android Studio.**

To add the Paho Android Service as a dependency to your app add the following parts to your gradle file.

****

* 1. Here in the code the first part adds the Paho release repository to the gradle config, so that Gradle can find the packaged Paho Android Service JAR. The second part adds the Paho Android Service as a dependency to the application.

1. To create and establish an MQTT-connection use the following code snipped. This code given connects the application to the MQTT broker using the server URL and as we already implemented the PAHO service in code which works here to connect with client.



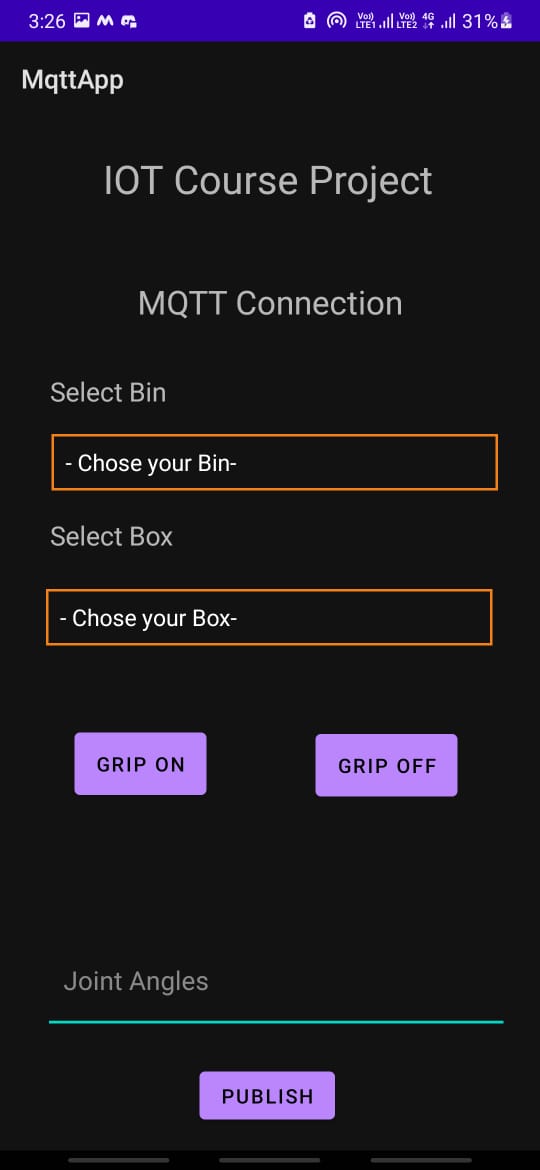
1. Now to check the connection status check the below code snippet which helps to get a toast as a “connected” for the successful connection and “Connection Failed” for failure in the connection.



1. Now as mentioned above you can connect and subscribe to the topic needed for the communication.
2. Now in this I added the names of the topic in the MainActivity.java itself in the form of variables, so that one can access these topics further in the code just by accessing the variable.

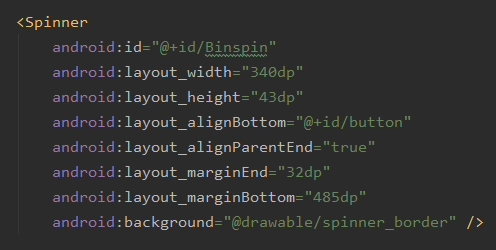


**Android App UI**



Now the diagram shows the main UI of the app which can see just after opening the app. In which there are two spinners for the Bin and the Box. You can take the reference of the same from the gazebo environment.

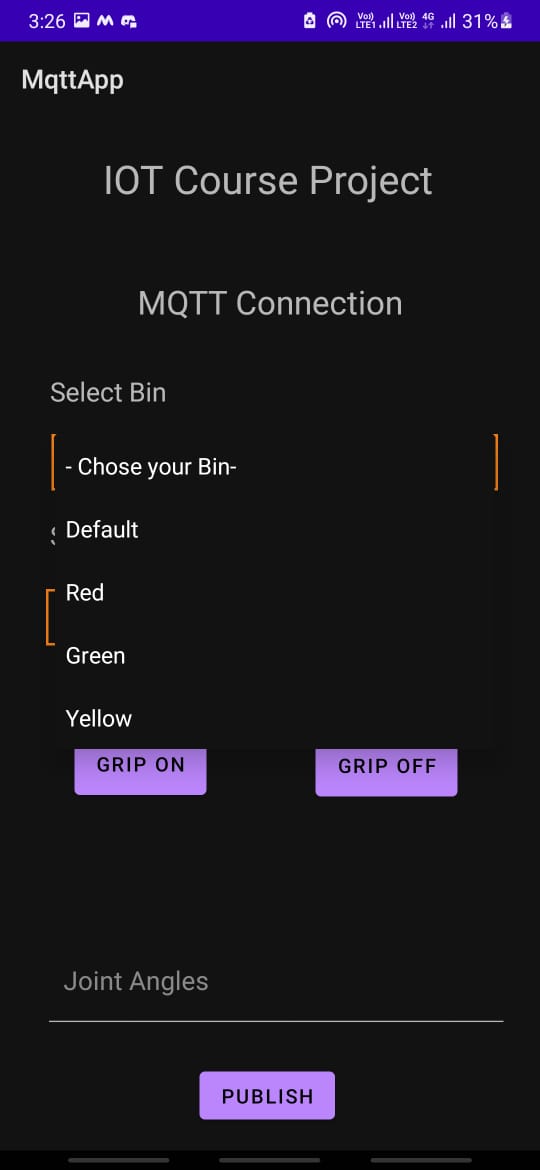
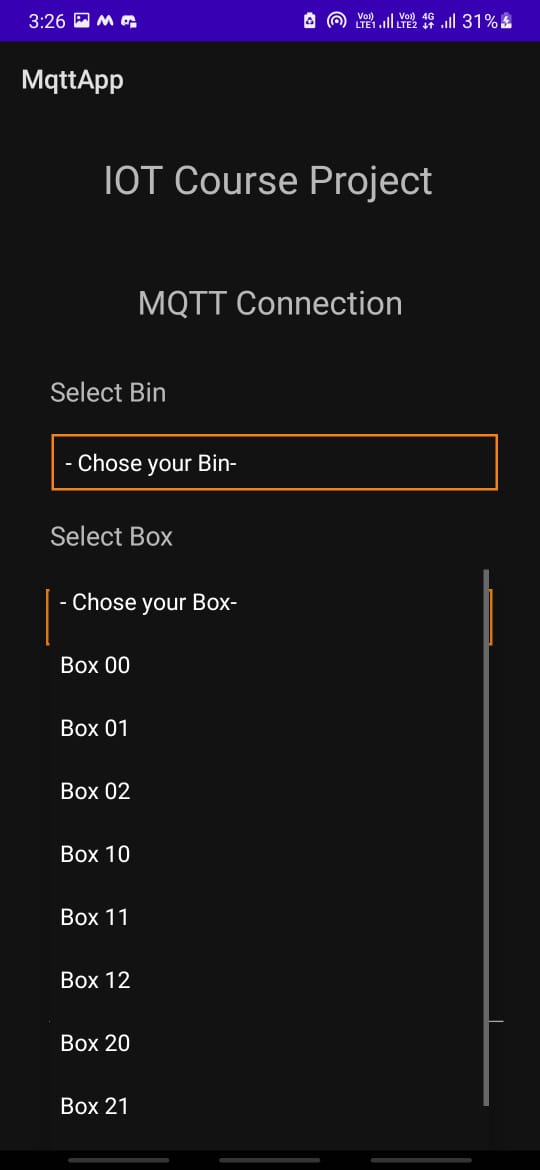
So here after opening the app it gets connected to the HIVE broker and as explained the two topics are assigned to the Bin and Box accordingly.



Now to make the spinner you must just need to add the spinner option in the xml file and set it according to your need. The code snippet for the same is as below.

1. To access this spinner, you need to go to MainActivity.java and access it through its unique id assigned.



 (1) (2)

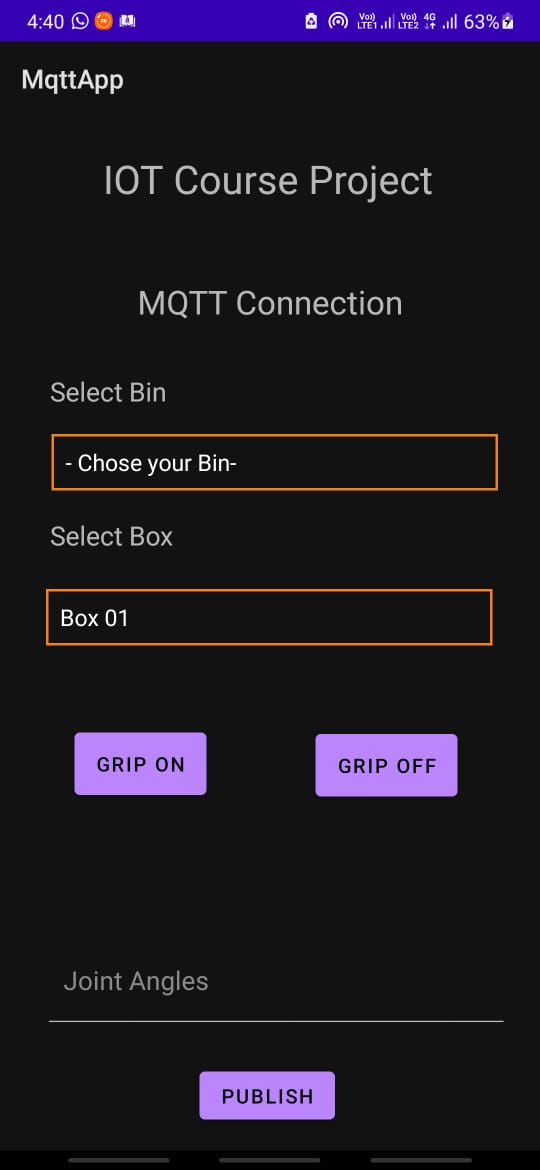
1. From the fig 1st and 2nd, you can see the options given in the section. Once the user selects these options from the list the coordinates assigned to that sent to the MQTT broker which can be then received by the robotic arm so that the arm can then work accordingly.

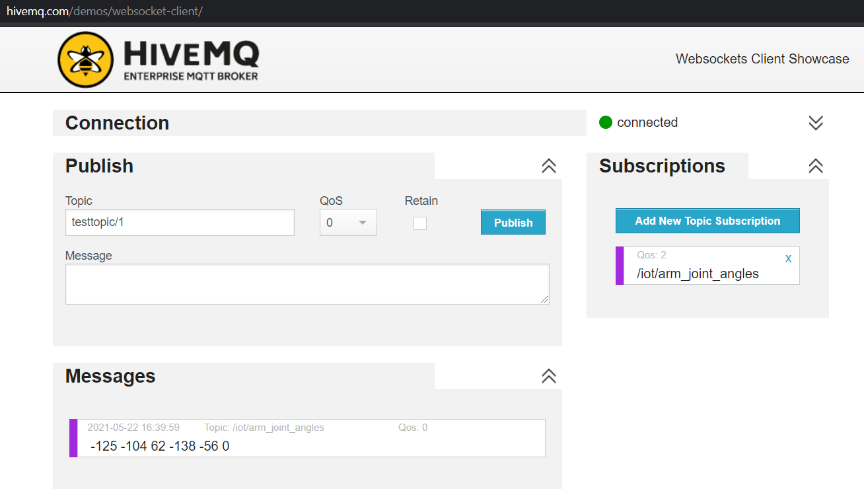
Also, you can see the input text box below the screen. This can be use for sending the different coordinates or any other string to the broker other than the given in the spinners.

Now taking one activity as an example:

1) First just subscribe the required two topics. The method is same as mentioned in the gazebo part.

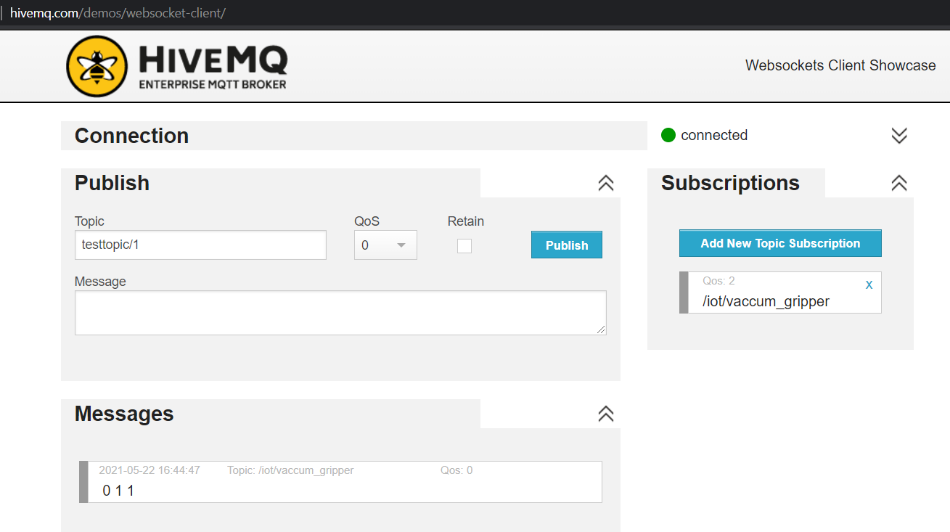
2) Now chose the box from the app





Note: Here I chose the 2nd box of the 1st row (Box 01) so accordingly the HIVEMQ receives the coordinates of that box.

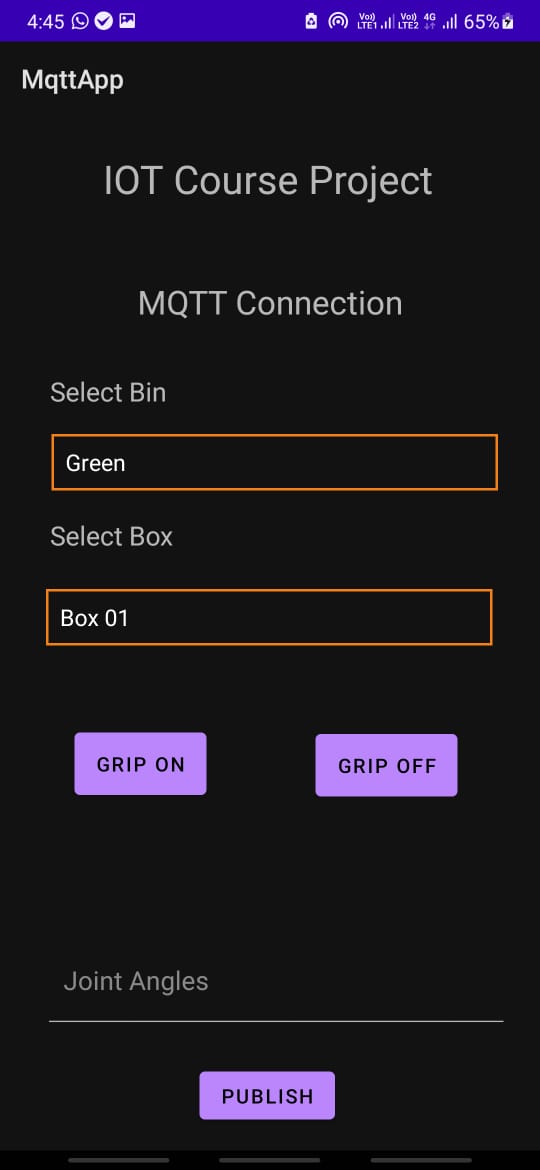
3) For picking the box just click the “GRIP ON” button

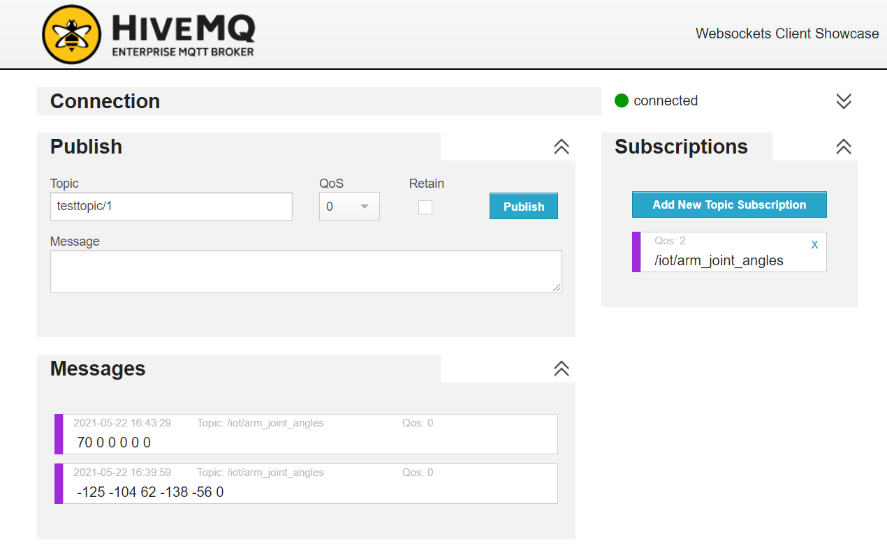


Here after clicking GRIP ON button the respective box gets picked.

Here the 01 denotes the box and 1 in the last denotes it picked up the box.

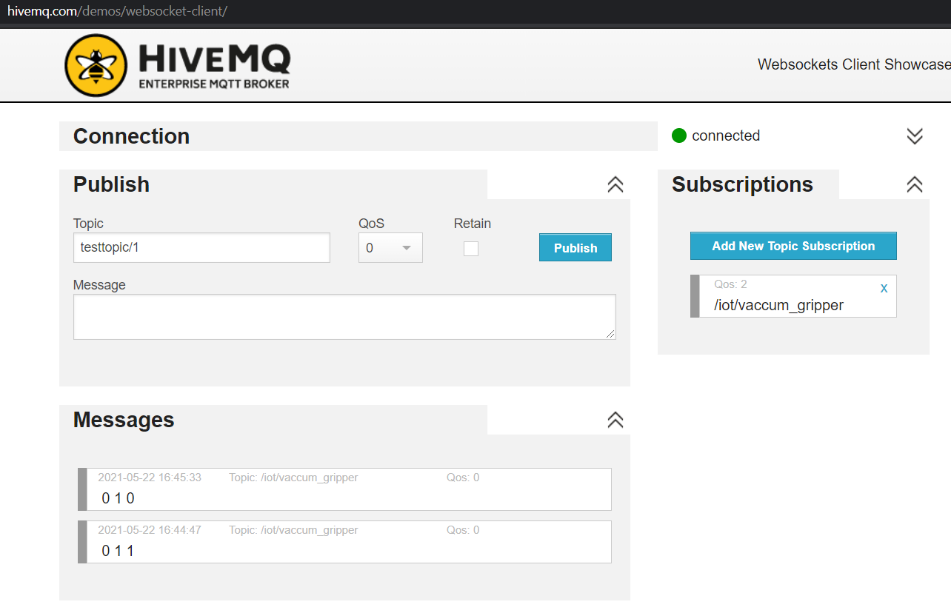
4) Afterward chose the bin form the respective spinner where you want to put the box so that the arm goes on that position of the box





Here I chose the Green Bin from the option, so accordingly the HIVEMQ receives the coordinates of that box.

5) In the last step just click the “GRIP OFF” button so that the arm leaves the box in the bin.



Here the 01 denotes the Box and last 0 denotes to leave that object i.e., gripper off.

**Demo Video Link:** <https://www.youtube.com/watch?v=MnFrrV90W3o>

**Conclusion:** The Project address one practical application in which MQTT Protocol can be used. Instead of using Android App. If we automate the process the sending of data, Arm will be able to work on its own. For example, we can use such arm in packaging industry to pick a package in a sequence. This Project helped us in learning the working of a Robotic Arm, simulation of Robotic arm using MoveIT Ros package and then controlling the simulation using MQTT.