NATIONAL INSTITUTE OF TECHNOLOGY

SILCHAR ASSAM-788010, INDIA

Final Report

Robotics Operating System (ROS)

SUBMITTED TO:-

Dr. Partha Pakray

SUBMITTED BY :-

1. Bhaskar Jyoti Bordoloi (1815015)
2. Md. Sarwar Ahmed (1814006)
3. Kimtiraj Handique (1816014)

Information page

Students

Bhaskar Jyoti Bordoloi
1815015

Kimtiraj Handique
1816014

Md Sarwar Ahmed
1814006

Official Instructor Dr. Partha Pakray (Sir)

Starting date 17.5.2021

Completion date 17.7.2021

Table of Contents

ACKNOWLEDGEMENT

Words cannot suffice to even begin to show the gratitude we owe to the people who helped us in this project but we will give it our best try. First of all, we would like to thank our supervisor, Dr. Partha Pakray, Department of Computer Science Engineering, National Institute of Technology, Silchar of whom we are highly indebted for his invaluable technical guidance and moral support during the project work. This work could not have attained its present shape without his generous help, invaluable suggestions, initiative & keen interest in this work.

We would also like to extend our thanks to our friends and colleagues for helping us all the times and being very supportive. Without all of you, this project would not have seen the light of the day.

ABSTRACT

The Robotic arm can be described with the URDF file format, which is a document written in eXtensible Markup Language (XML) that describes all the necessary properties of a robot. This includes the geometry, joints and controllers required to achieve control and translation in a software environment. URDF files work in the ROS environment and the functionality of URDF files can be simulated by using Gazebo. Gazebo is a set of ROS packages used to simulate 3D rigid body motions. It allows the importing of 3D models, generating LiDAR sources and populating the world with obstacles, which makes it a versatile environment for simulation. Developing a virtual model for a robot helps with testing and experimentation, such that the risk to the real robot can be mitigated by catching some possible errors and design flaws before any experimentation in the real world is conducted. It provides an opportunity to prototype without any material cost. So, that the proper sensors and controllers can be chosen based on experiments in the simulator before installing them on the actual robot.

This project represents an upgraded, real world -oriented application version of robot (or more precisely) a robotic arm (using Robot Operating System (ROS) and Gazebo). The main purpose of this work is to provide Covid-19 vaccines (slots/tokens/vaccine dosage) to people by our robotic arm using the Robot Operating System (ROS). Nowadays the vaccination is done by human beings (i.e by health workers-Doctors and Nurses) which makes them highly prone to the live -taking virus daily, so to reduce the workload and save lives of fore -front Covid warriors like Doctors and Nurses, we are trying to build this model. In this project is presented an interfacing of a robotic arm with a motion planner and a 3D viewer (Rviz).In order to do this task, a motion planner, who deals with constraints, i.e. collisions among the arm joints and objects,is used. Also, a 3D viewer, called Rviz, is used to visualize (online and offline) the motion of the arm.

Vision

The vision of the Summer Internship Project is establishing a unique identity in ROS related technologies, by development of high-quality human and knowledge resources in the area of Robot Operating Systems and Gazebo Simulations to meet local, national, and global economic and social needs of a community/ communities, a region or/ and the society at large in a self-sustained manner. The mission of the Summer Internship Project is to train and transform ourselves into responsible thinking engineers/ technologists in the area of emerging and trending interpreter level languages like Python, ROS, Gazebo and also, to motivate ourselves in order to attain professional excellence while implementation and lastly, to proactively engage in the betterment of the society.

The primary goal of this project was basically, to design a Universal Robot Description File (URDF) for the Wheeled Robotic Arm or, Automated Teller Vaccinator (or, e-ATV), which has a Wheeled functionality attached, as a passive link with the Robotic arm's base link.

The URDF file is equipped with all the necessary sensors (depth-rgb-sensor), for generating topics to test control algorithms, navigation routines and SLAM procedures on the real-life counterpart.

Simulations allow us to project Inverse Kinematics on the new virtual world.

The project used the open-source platform Robot Operating System (ROS) along with the Rviz and Gazebo programs to design the platform for testing the URDF functionality.

The URDF file was designed using an existing Simulation Description Format (SDF) file as the foundation. SDF is a more advanced file format for describing robots. However, it does not function in ROS.

Gazebo uses the SDF format to better integrate robots into the world, while ROS is only concerned with the robot itself.

The resulting URDF file was compared step-by-step and verified in the ROS environment.

Major adjustments were necessary to make the file viable, due to backwards incompatibility between the SDF and URDF formats.

AIM

The robot that is being used in this project is the Wheeled Robotic Arm or Automated Teller Vaccinator with a RGB Sensor (e-ATV with a depth sensor), which will be primarily used for scheduling of prospects.

Robots such as the e-ATV can be described with the URDF file format, which is a document written in eXtensible Markup Language (XML) that describes all the necessary properties of a robot. This includes the geometry, joints and controllers required to achieve control and translation in a software environment. URDF files work in the ROS environment and the functionality of URDF files can be simulated by using Gazebo. Gazebo is a set of ROS packages used to simulate 3D rigid body motions. It allows the importing of 3D models, generating Sensor sources and populating the world with obstacles for controllers and end effectuators to react to, which makes it a versatile environment for simulation.

Developing a virtual model for a robot helps with testing and experimentation, such that the risk to the real robot can be mitigated by catching some possible errors and design flaws before any experimentation in the real world is conducted. It provides an opportunity to prototype without any material cost. So, the proper sensors and controllers can be chosen based on experiments in the simulator before installing them on the actual robot.

Objective

At the end of this project, a URDF of the e-ATV should exist in our ROS workspace & work in Gazebo.

It is supposed to contain all the sensors of the actual test WHEELED ARM, which allows the user to conveniently test and debug new software before pushing it onto the real test vehicle. It also allows testing the rover in different conditions by changing its surroundings.

The robot described by the URDF file shall then be simulated in Gazebo.

It simulates the 3D rigid body dynamics of robots along with the static objects in the environment. The 3D environment designed in this project was developed for the purpose of testing the functionality of the URDF file. This includes the motion capabilities of the robot and its behaviour on inclines. Additionally, the sensors' data from the Gazebo environment is visualized in RViz. For example, the 3D point cloud generated by the Depth Sensor (RGB Camera) equipment illustrates how the robot sees the virtual environment in Gazebo. Therefore, the designed environment also served as a useful tool to test the performance of a simulated sensor. The simulation is for research purposes and streamlines the development of the autonomous driving capabilities of the Polaris vehicle. This project is a continuation of the previous research done on developing an autonomous vehicle. Thus, by the conclusion of this project there should be a meaningful contribution to the development of a simulation platform for Polaris. The success of this project can be measured by the usefulness of the simulation platform in future research and development.

INTRODUCTION

Mankind has always strived to give life like qualities to its artifacts in an attempt to find substitutes for himself to carry out his orders and also to work in a hostile environment. The popular concept of a robot is of a machine that looks and works like a human being.

In this project, we're going to explore the robot modeling using the Unified Robot Description Format (URDF). At the end of this project, we will have a model ready and running in Gazebo simulator.

The medical facilities are moving from current state of automation to Robotization, to increase productivity and to deliver precise services. The medical robots of today may not look the least bit like a human being although all the research is directed to provide more and more anthropomorphic and humanlike features and super-human capabilities in these.

One type of robot commonly used in medicals is a robotic manipulator or simply a robotic arm and a two wheeled robot ground below it. Basically, wheeled robot is a robot composed by 3 links and 2 joints. Every robot needs a base link, in this case, the chassis is in charge of connecting all the parts of the robot Arm is an open or closed kinematic chain of rigid links interconnected by movable joints. In some configurations, links can be considered to correspond to human anatomy as waist, upper arm and forearm with joint at shoulder and elbow. At end of arm a wrist joint connects end effectors which may be a tool and its fixture or a gripper or any other device to work. Here how an arm robot can be designed for a workstation where injections can be given to the patients and with par the current ongoing pandemic, the vaccination procedure will be very efficient with the robotic arm. All the various problems and obstructions for the vaccination process has been deeply analyzed and been taken into consideration while designing the robotic arm.

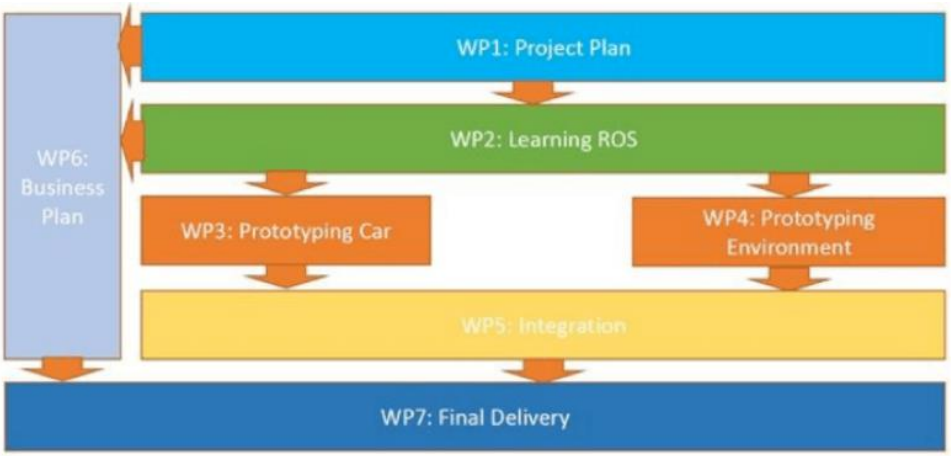
Project Planning

The main framework used: ROS (Robot Operating System) and ROS Development Studio.

Gazebo was used as a physics simulator to simulate the behavior of the vehicle and Rviz was used to actually visualize the sensor data that is produced by the robot.

As a versioning tool, the GitHub/ GitLab was used to keep track of software changes and allow working on different parts of the project at the same time.

The tasks defined in this structure were aimed to be completed as indicated in figure in next page. These milestones were defined such that they would agree with the overall program of the course/ or internship, and allow for reasonable time for each task so that they may finally indicate to be implemented successfully. (The e-ATV has been accredited as CAR, sometimes, in the following images (due to the Robotic Arm's translational movability) for ease of convenience and understanding.)



As seen, the overall plan begins with a planning stage, followed by a training stage for group members to get familiar with the technical aspects. In the following weeks, the technical work is carried out parallel to the business aspects of the project, and the overall project is concluded with a presentation of the final implementation and the submission of the final report. In other words, necessary work is carried out to meet more than one milestone at the same time. Figure provided above illustrates this by visualizing work packages with dependencies.

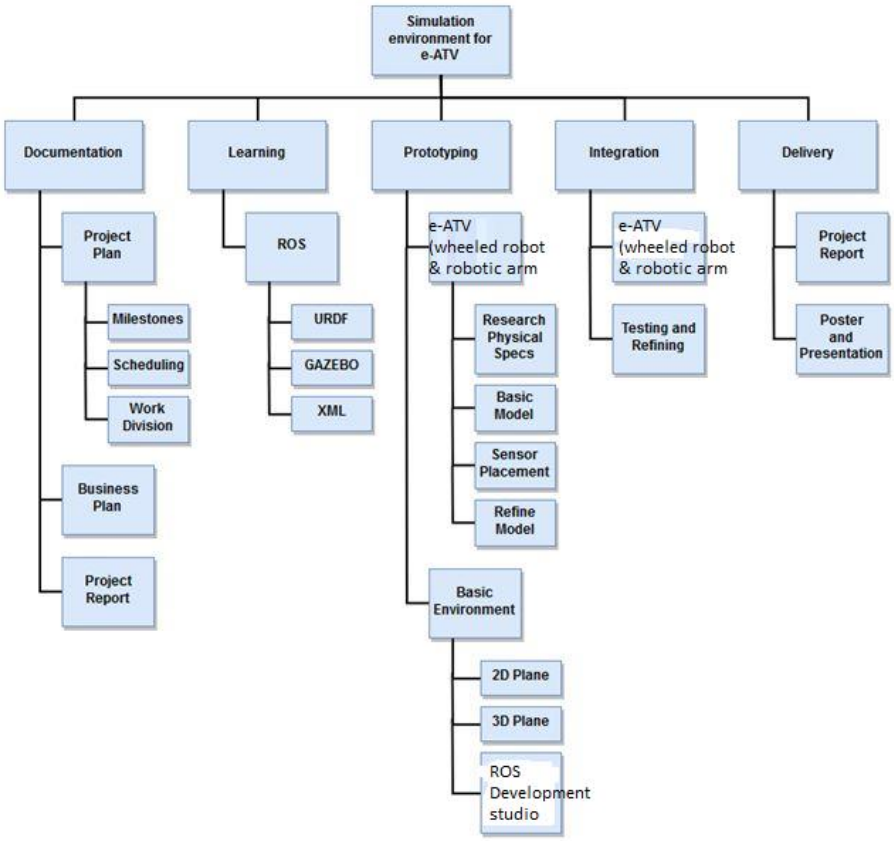


Fig: **Work Breakdown Structure**

Framework:

- ***Robot Operating System (ROS)***

The Robot Operating System (ROS) is an open-source framework for robots. It was developed in 2007 at the Stanford Artificial Intelligence Laboratory. Its main components are hardware abstraction and communication between different modules. There is a multitude of packages for many different robots and applications available that help the user in the testing of robots. Its main components are the so-called nodes, which communicate via the subscription to topics and messages. In the ROS framework, nodes communicate with one another through topics. A node can be a subscriber to multiple topics, and it can advertise numerous topics. The topics are special messages whose definitions are placed inside *.msg files.

- **ROS Nodes**

ROS nodes are basically single separated processes. Each one of these processes is doing a separate task for the robot and a robot control system is usually consisting of multiple of these nodes. This reduces code complexity and also allows for easy extension and expansion of the robot's functionality. Nodes can communicate with each other via messages that get sent to topics. Every node needs to be started individually. They then register to the master, which manages all the nodes currently running. The master always runs in the background.

- **ROS Topics**

ROS topics are a way of decoupling the communication of nodes with each other. A topic functions as a midpoint for the communication. Nodes can send messages to a topic, which can then be requested by other topics. For a node to send a message to a topic it needs to 'publish' to that topic, and for another node to get the information from that topic, it needs to 'subscribe' to that topic. The type of the message needs to match with the type of the topic. This is a convenient way of separating communication by type and allows different nodes to easily access the information of the robot, depending on their needs.

- **Launch files**

Launch files are files in ROS that simplify the startup of a robot or simulation by allowing to group several nodes together and launch them with only one file, instead of launching each node individually. It also allows to easily change some parameters of the simulation by setting parameters for some names and settings. In the launch file defined for our project, the necessary visualisation tools, controllers and parameters are defined and launched together to enhance usability.

- **RViz**

RViz is a 3D Visualization tool for ROS. It is one of the most popular tools for visualization. It takes in a topic as input and visualizes that based on the message type being published. It lets us see the environment from the perspective of the robot.

- **Gazebo**

Once we have all the code ready and running, we need to test our code so that we can make changes if necessary. Doing this on a real robot will be costly and may lead to a wastage of time in setting up robot every time. Hence we use robotic simulations for that. The most popular simulator to work with ROS is Gazebo. It has good community support, it is open source and it is easier to deploy robots on it.

- **Moveit**

The MoveIt Setup Assistant is a graphical user interface for configuring any robot for use with MoveIt. Its primary function is generating a Semantic Robot Description Format (SRDF) file for your robot. Additionally, it generates other necessary configuration files for use with the MoveIt pipeline.

Development of the Simulation

This part describes the development process of the simulation in more detail. This includes the first working prototype.

- **General Structure**

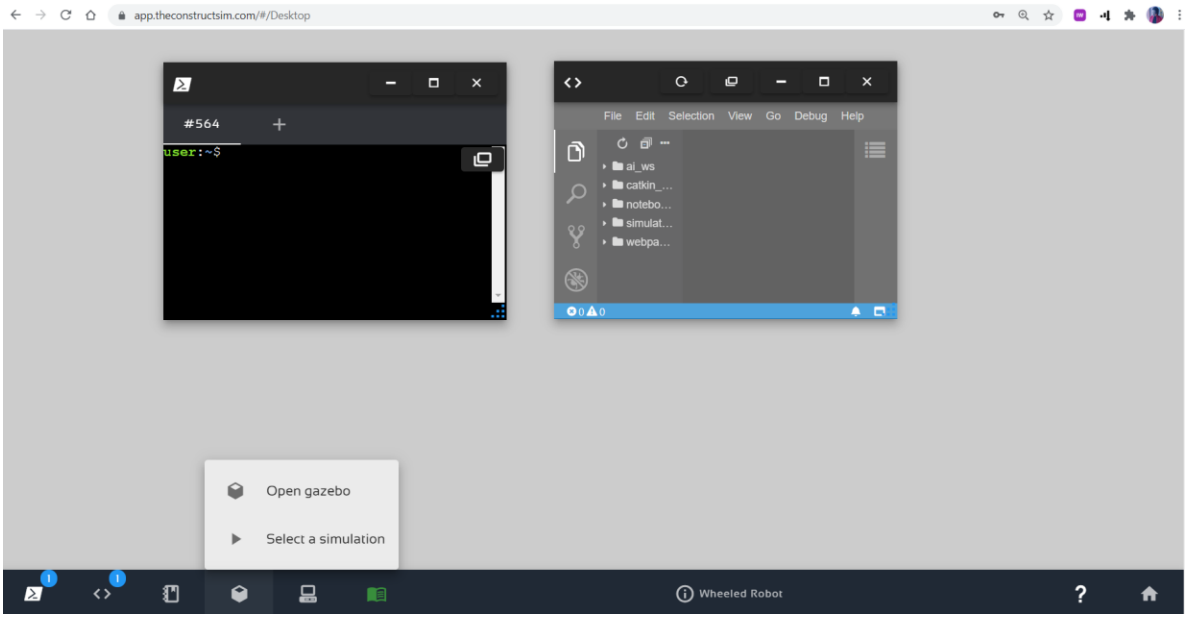
The first step in setting up the project was the general ROS software. We worked on the online ROS development studio https://app.theconstructsim.com/ . The ROS version used was ROS Kinetic. A catkin workspace was created for this project specifically. Catkin is a build system for ROS that is based on CMake. It builds all packages in ROS. Two packages were created in the workspace that includes all the files created in this project, one for the arm and the other for the wheel.

Implementation of the Two wheeled Robot using ROS:

STEP 1

we are going to use **ROSDS (ROS Development Studio)**. In this first part, we are going to create a wheeled robot model, visualize it in RViz and spawn it into a gazebo world.

That's our development environment or **ROSDS** desktop:



STEP 2

We creating our package, inside of **simulation_ws/src**:

```
user:~$ cd ~/simulation_ws/src
 user:~$ catkin_create_pkg m2wr_description urdf
```

The first we are going to do it to create the robot description file. We are using a XACRO file.

STEP 3

We create a new folder **urdf** inside of **~/simulation_ws/src/m2wr_description** and write the code of robot description to a new file:
~/simulation_ws/src/m2wr_description/urdf/m2wr.xacro

STEP 4

We have our robot model defined. Now we are going check it in RViz. In order to do that, we create a **launch** file and that opens **RViz** and fill its robot visualization with our model.

We create a new folder: **~/simulation_ws/src/m2wr_description/launch/rviz.launch**

and then write the code for Rviz launch file.

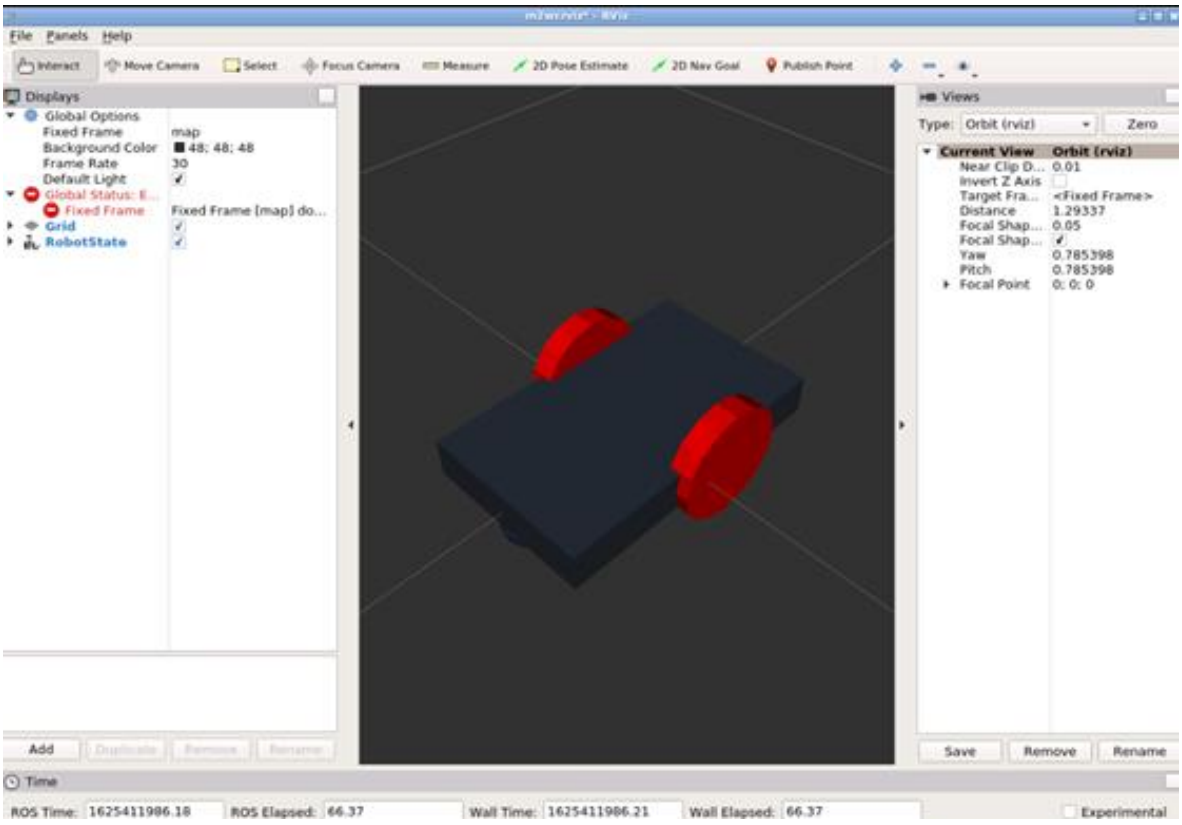
Now,we open a terminal and compile the **simulation_ws**. (we actually don't have anything to be compiled, but we need **catkin** to generate ROS header files, in order to have it into our **$ROS_PACKAGE_PATH**). After compiling it,we launch the **RViz** visualization of the robot.

You'll execute something like:

```
user:~$ cd ~/simulation_ws
 user:~/simulation_ws/$ catkin_make
 user:~/simulation_ws/$ roslaunch m2wr_description rviz.launch
```

Now, we open the **Graphical Tools** application:

Rviz: Robot Visualization



STEP 5

First, we create a new launch file:
~/simulation_ws/src/m2wr_description/launch/spawn.launch

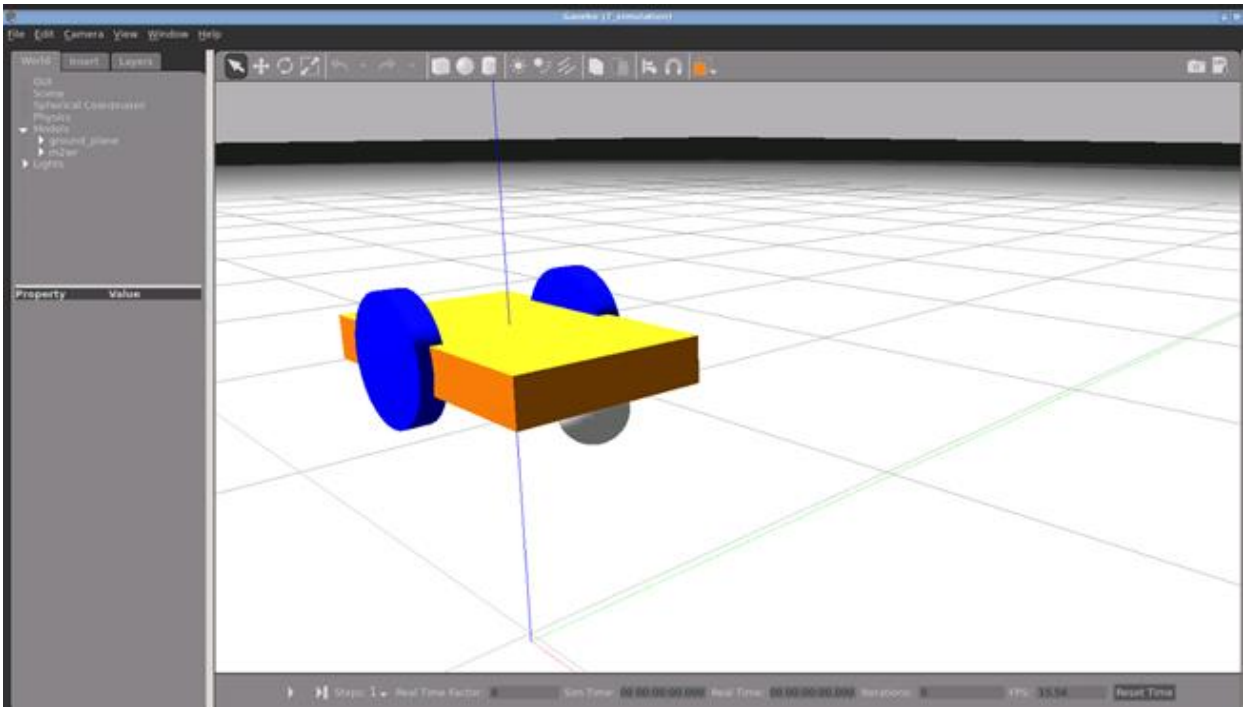
Stop the last launch file we started (for RViz), we are going to use the same terminal once again.

Finally, we spawn the robot to the Gazebo simulation.

In your terminal, execute the following:

```
user:~/simulation_ws$ roslaunch m2wr_description spawn.launch
```

Gazebo Simulator



URDF File for wheeled Robot:

```
<?xml version="1.0" ?>

<robot name="m2wr" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <material name="black">

    <color rgba="0.0 0.0 0.0 1.0"/>

  </material>

  <material name="blue">

    <color rgba="0.203125 0.23828125 0.28515625 1.0"/>

  </material>

  <material name="green">

    <color rgba="0.0 0.8 0.0 1.0"/>

  </material>

  <material name="grey">

    <color rgba="0.2 0.2 0.2 1.0"/>

  </material>

  <material name="orange">

    <color rgba="1.0 0.423529411765 0.0392156862745 1.0"/>

  </material>

  <material name="brown">

    <color rgba="0.870588235294 0.811764705882 0.764705882353 1.0"/>

  </material>

  <material name="red">

    <color rgba="0.80078125 0.12890625 0.1328125 1.0"/>

  </material>

  <material name="white">

    <color rgba="1.0 1.0 1.0 1.0"/>

  </material>


  <gazebo reference="link_chassis">
```

```
  <material>Gazebo/Orange</material>
</gazebo>
<gazebo reference="link_left_wheel">
  <material>Gazebo/Blue</material>
</gazebo>
<gazebo reference="link_right_wheel">
  <material>Gazebo/Blue</material>
</gazebo>

<gazebo>
  <plugin filename="libgazebo_ros_diff_drive.so" name="differential_drive_controller">
    <legacyMode>false</legacyMode>
    <alwaysOn>true</alwaysOn>
    <updateRate>20</updateRate>
    <leftJoint>joint_left_wheel</leftJoint>
    <rightJoint>joint_right_wheel</rightJoint>
    <wheelSeparation>0.2</wheelSeparation>
    <wheelDiameter>0.2</wheelDiameter>
    <torque>0.1</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>link_chassis</robotBaseFrame>
  </plugin>
</gazebo>

<link name="link_chassis">
  <!-- pose and inertial -->
  <pose>0 0 0.1 0 0 0</pose>
```

```
    <inertial>
      <mass value="5"/>
      <origin rpy="0 0 0" xyz="0 0 0.1"/>
      <inertia ixx="0.0395416666667" ixy="0" ixz="0" iyy="0.106208333333" iyz="0"
izz="0.106208333333"/>
    </inertial>
    <!-- body -->
    <collision name="collision_chassis">
      <geometry>
        <box size="0.5 0.3 0.07"/>
      </geometry>
    </collision>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.5 0.3 0.07"/>
      </geometry>
      <material name="blue"/>
    </visual>
    <!-- caster front -->
    <collision name="caster_front_collision">
      <origin rpy=" 0 0 0" xyz="0.35 0 -0.05"/>
      <geometry>
        <sphere radius="0.05"/>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>0</mu>
```

```
            <mu2>0</mu2>
            <slip1>1.0</slip1>
            <slip2>1.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="caster_front_visual">
      <origin rpy=" 0 0 0" xyz="0.2 0 -0.05"/>
      <geometry>
        <sphere radius="0.05"/>
      </geometry>
    </visual>
  </link>

  <link name="link_right_wheel">
    <inertial>
      <mass value="0.2"/>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0"
izz="0.001"/>
    </inertial>
    <collision name="link_right_wheel_collision">
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
    </collision>
    <visual name="link_right_wheel_visual">
```

```
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
    </visual>
  </link>

  <joint name="joint_right_wheel" type="continuous">
    <origin rpy="0 0 0" xyz="-0.05 0.15 0"/>
    <child link="link_right_wheel"/>
    <parent link="link_chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>

  <link name="link_left_wheel">
    <inertial>
      <mass value="0.2"/>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0"
izz="0.001"/>
    </inertial>
    <collision name="link_left_wheel_collision">
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
    </collision>
```

```
    <visual name="link_left_wheel_visual">
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
    </visual>
  </link>

  <joint name="joint_left_wheel" type="continuous">
    <origin rpy="0 0 0" xyz="-0.05 -0.15 0"/>
    <child link="link_left_wheel"/>
    <parent link="link_chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>
</robot>
```

ARM IMPLEMENTATION

a. ROS catkin workspace

The workspace has several folders. Let's look at the function of each folder.

src Folder:

The src folder inside the catkin workspace folder is the place where you can create, or clone, new packages from repositories. ROS packages only build and create an executable when it is in the src folder.

build Folder:

When we run the catkin_make command from the ROS workspace, the catkin tool creates some build files and intermediate cache CMake files inside the build folder. These cache files help prevent rebuilding all the packages when running the catkin_make command.

devel Folder:

When we run the catkin_make command, each package is built, and if the build process is successful, the target executable is created. The executable is stored inside the devel folder, which has shell script files to add the current workspace to the ROS workspace path. We can access the current workspace packages only if we run this script.

Generally, the following command is used to do this.

source ~/<workspace_name>/devel/setup.bash

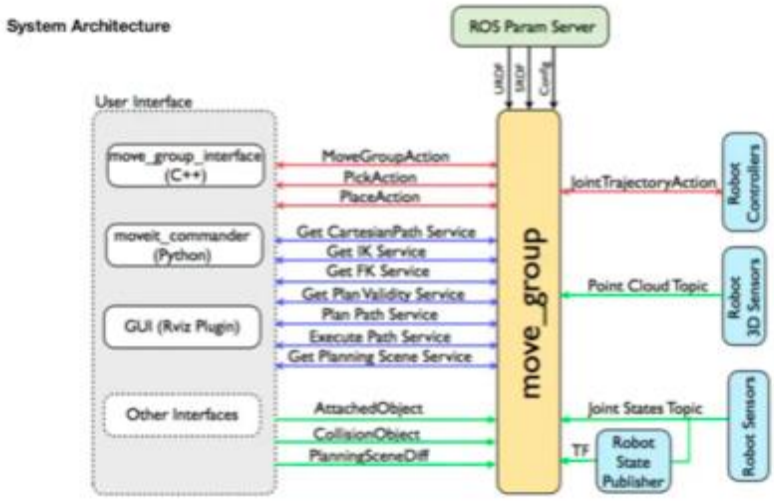
install Folder:

After building the target executable locally, run the following command to install the executable.

$ catkin_make install

b. Moveit!

Moveit! is a state of art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3d perception, kinematics, control and navigation.



Start

- To start the MoveIt Setup Assistant:

roslaunch moveit_setup_assistant setup_assistant.launch

Generation of Self-Collision Matrix :

The **Default Self-Collision Matrix Generator** searches for pairs of links on the robot that can safely be disabled from collision checking, decreasing motion planning processing time.

These pairs of links are disabled when they are always in collision, never in collision, in collision in the robot's default position or when the links are adjacent to each other on the kinematic chain.

Then next, the **sampling density** specifies how many random robot positions to check for self collision. Higher densities require more computation time while lower densities have a higher possibility of disabling pairs that should not be disabled.

The default value is 10,000 collision checks.

Collision checking is done in parallel to decrease processing time.

Addition of Virtual Joints

Virtual joints are used primarily to attach the robot arm to the world.

In our case we select:

- Name: virtual_joint (just to know its a virtual joint)
- Parent: world
- Child: the base_link , that we want to connect to the world.
- Joint Type: Fixed, because we won't move

Addition of Planning Groups

Planning groups are used for semantically describing different parts of our robot, such as defining what an arm is, or what an end effector is.

We will choose the following:

- **Solver**: kdl_kinematics_plugin/KDLKinematicsPlugin as the kinematics solver

This is the plugin in charge of calculating the inverse kinematics. It is **a generic Kinematic solver**, that will be ok for now. It is the default one in Moveit!. It only works with Serial Kinematics Chains.

- Name: openmanipulator_arm seems appropriate.
- Kin Solv. Attempts: 3 seems reasonable.
- Planner: for now we leave this none
- We add the joints now.

Addition of Robot Poses

The Setup Assistant allows us to add certain fixed poses into the configuration.

This helps if, for example, we want to define a certain position of the robot as a **Home position**.

We added four poses:

- Two for the OpenManipulator_ARM group
- Two for the Gripper group

Labelling of End Effectors

We have already added the gripper of the arm.

Now, we will designate this group as a special group: end effectors.

Designating this group as end effectors allows some special operations to happen on them internally.

We set:

- Name: gripper
- Group ENDEffector: gripper
- Parent Link: Link5
- Parent Group: Not necessary here to state it.

ROS controllers:

Here we have two options:

- Add the controllers manually: This allows us to select which type of control we are adding.
- Auto add FollowTrajectoryControl

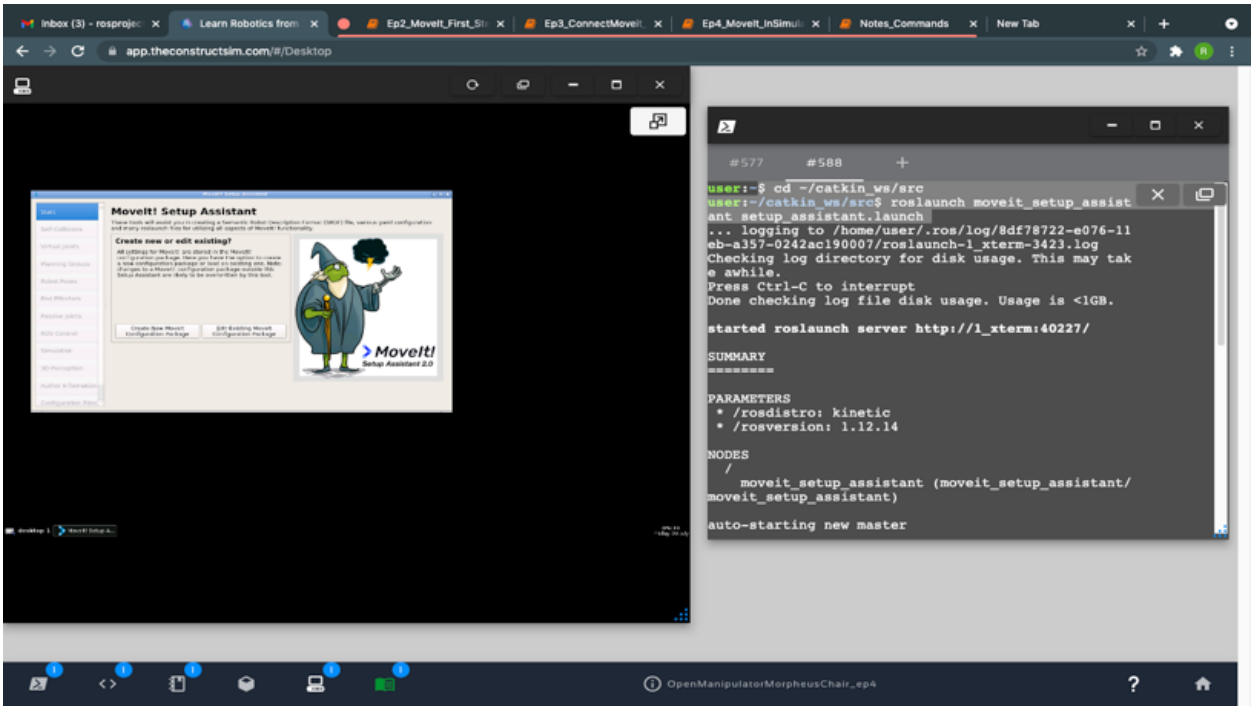


Figure. moveit wizard

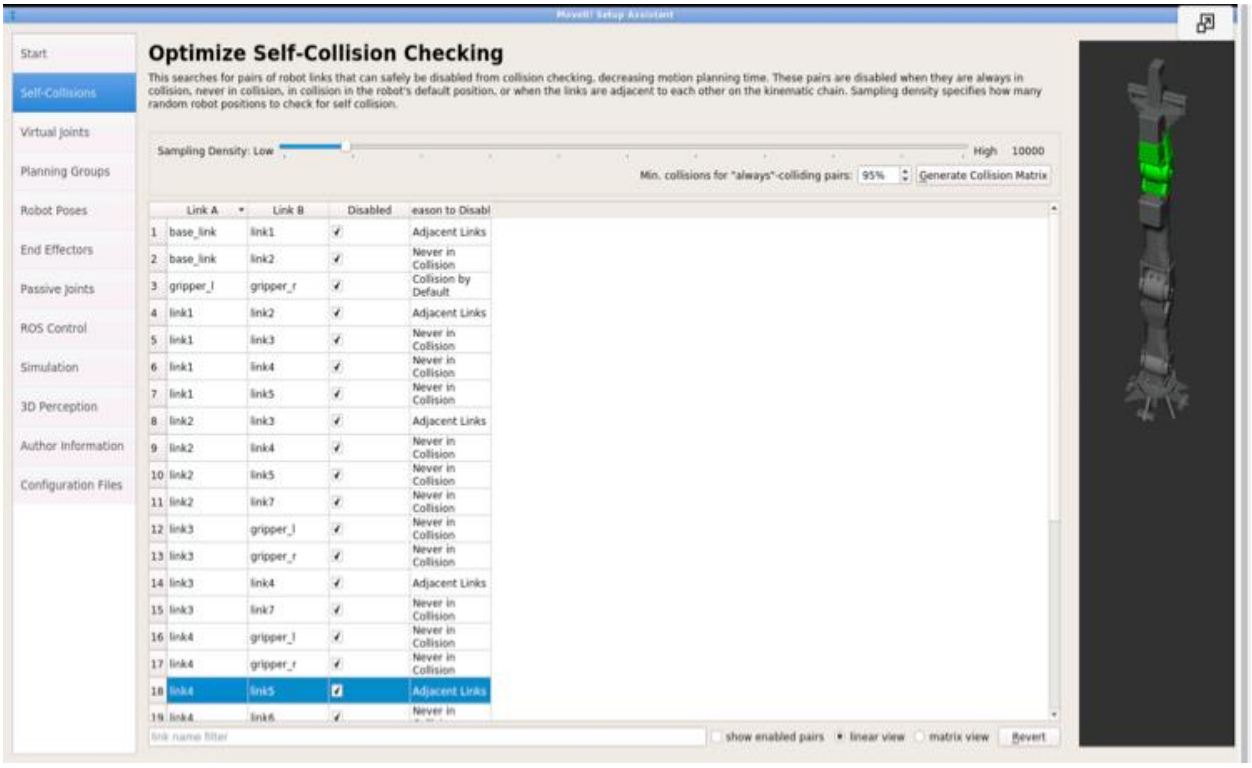


Figure. collision matrix

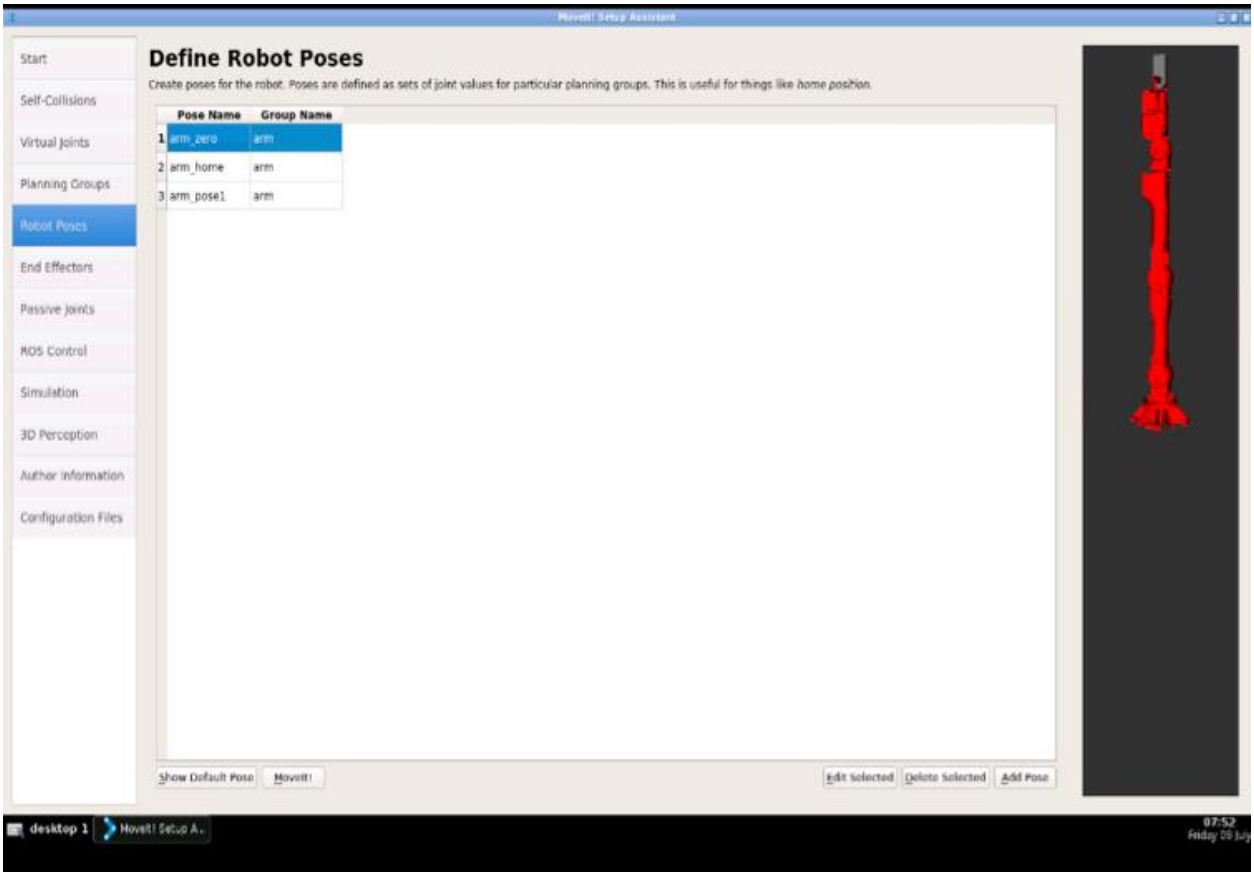


Figure. robot poses

Finally, **we generate the package** for our Gazebo Arm Simulation.

---

c. **Launching the Simulation for our Robotic arm**

The most standard way to use moveit is through TrajectoryJointControllers.

It is essentially the controllers that can be setup automatically in the **Moveit! Wizard**.

So Movit! will send goals to an action server of message type control_msgs/FollowJointTrajectoryAction
And it is this trajectory_server that is going to receive the goal and give the feedback to moveit.

roscd open_manipulator_core/launch

ls controller_launch.launch

We are launching TWO launch files:

- **trajectory_server**, from package open_manipulator_core:

This is the one that is responsible for creating the ActionServer of type: control_msgs/FollowJointTrajectory

This will convert theFollowJointTrajectory messages into position of the joints of the robotic arm.

- **gripper_commander**, from package open_manipulator_core: This is a topic subscriber of type std_msgs/String that will be in charge of converting Strings of open_gripper and close_gripper into its corresponding gripper position values

- **We configured MoveIt!** to be able to use the FollowJointTrajectory action server :

We shall start from the movit package we created earlier as,

Roslaunch openmanipulator_ep2_movit_configopen_manipulator_planning_execution.launch

We should now be able to see in the Graphical Interface or in our local computer if we have just executed everything there, the RVIZ with the OpenManipulator.

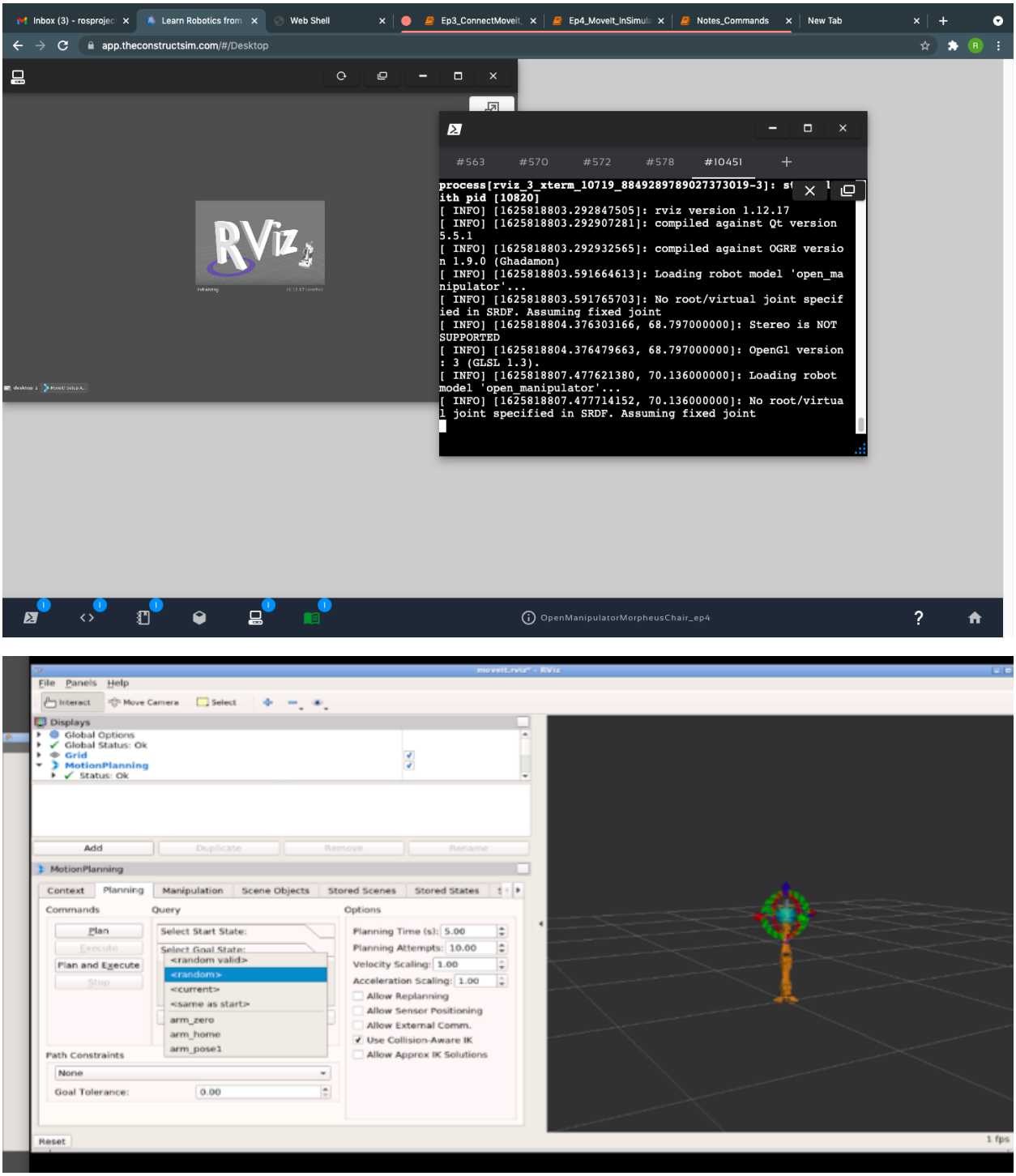


Figure. simulation in Rviz (Home-pose of the Robotic Arm)

Gazebo

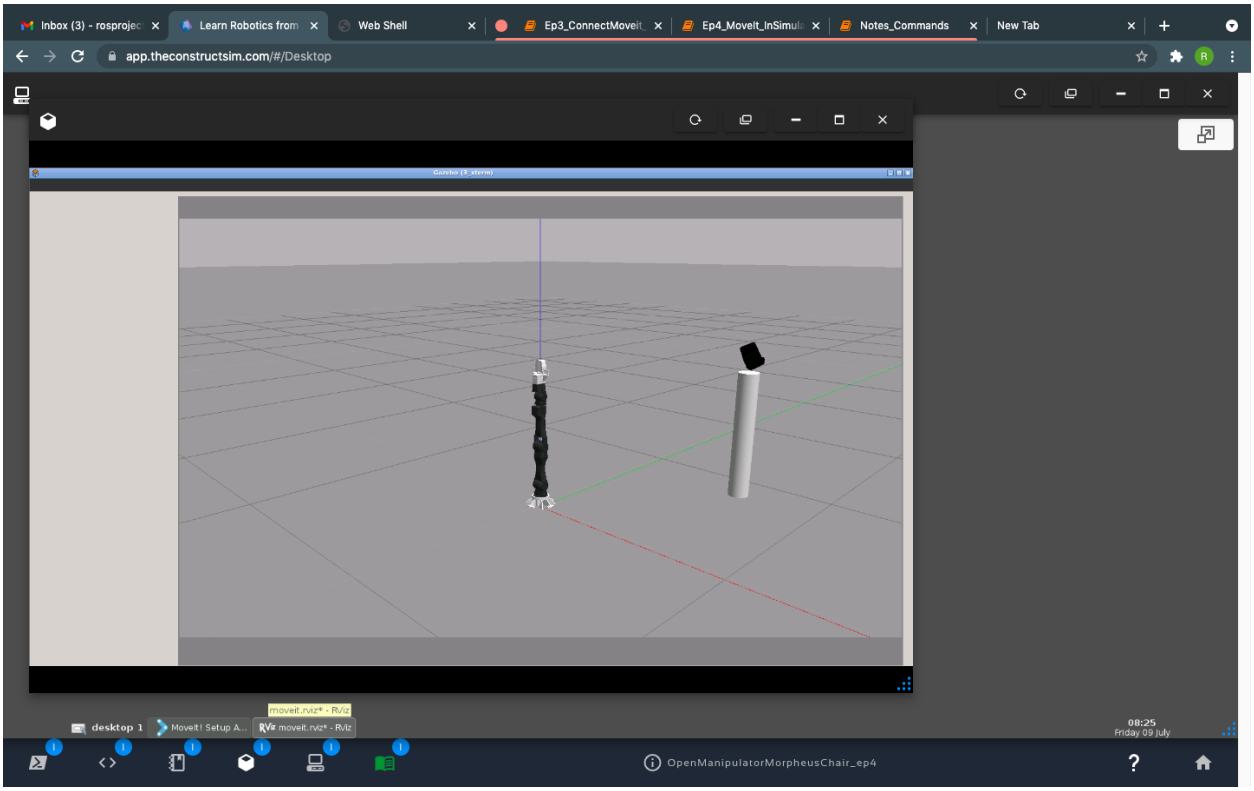


Figure. simulation in Gazebo(Home-pose of the Robotic Arm as perceived by Humans)

Publishing a new pose for the robotic arm:

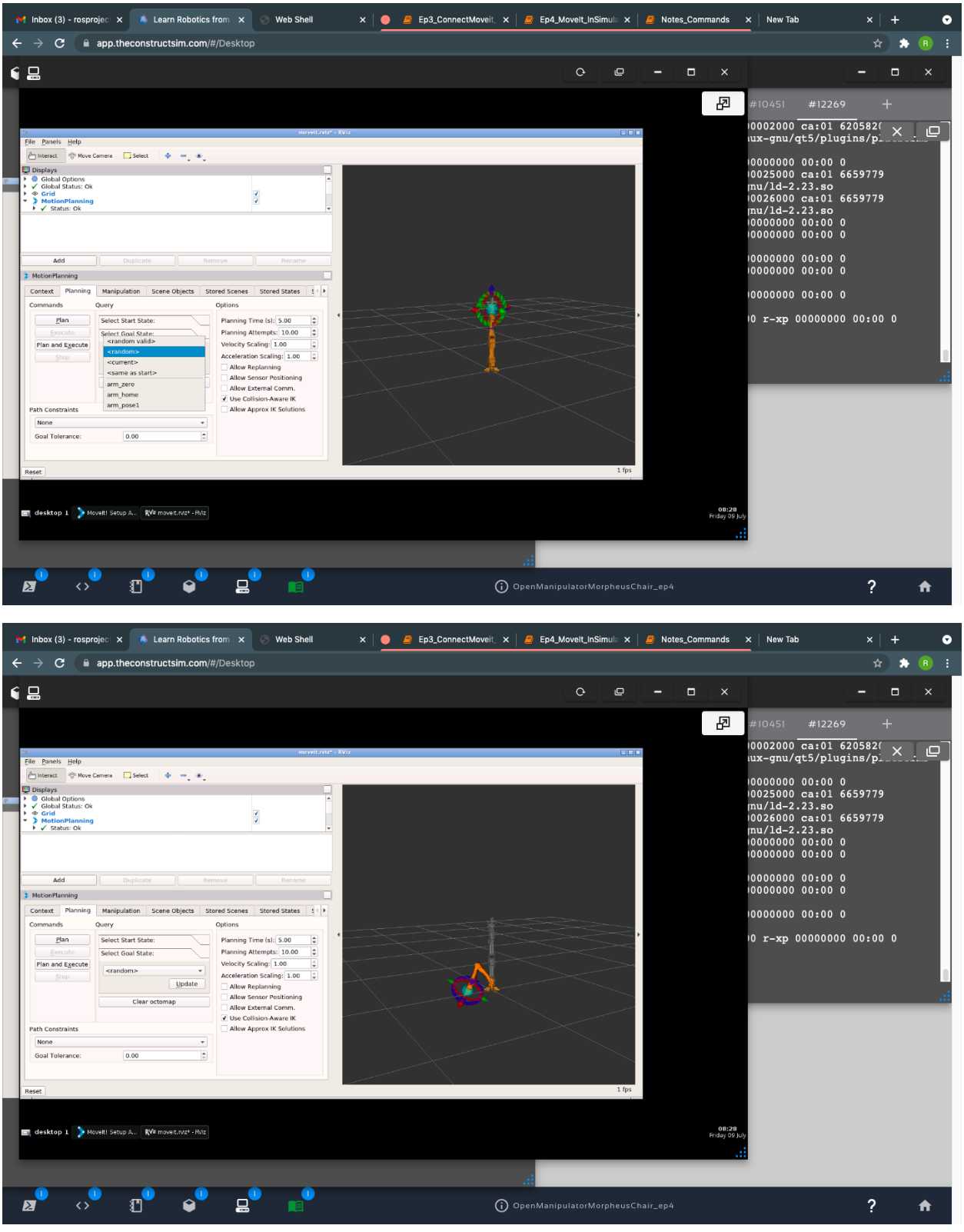


Figure. Updating in Rviz wizard

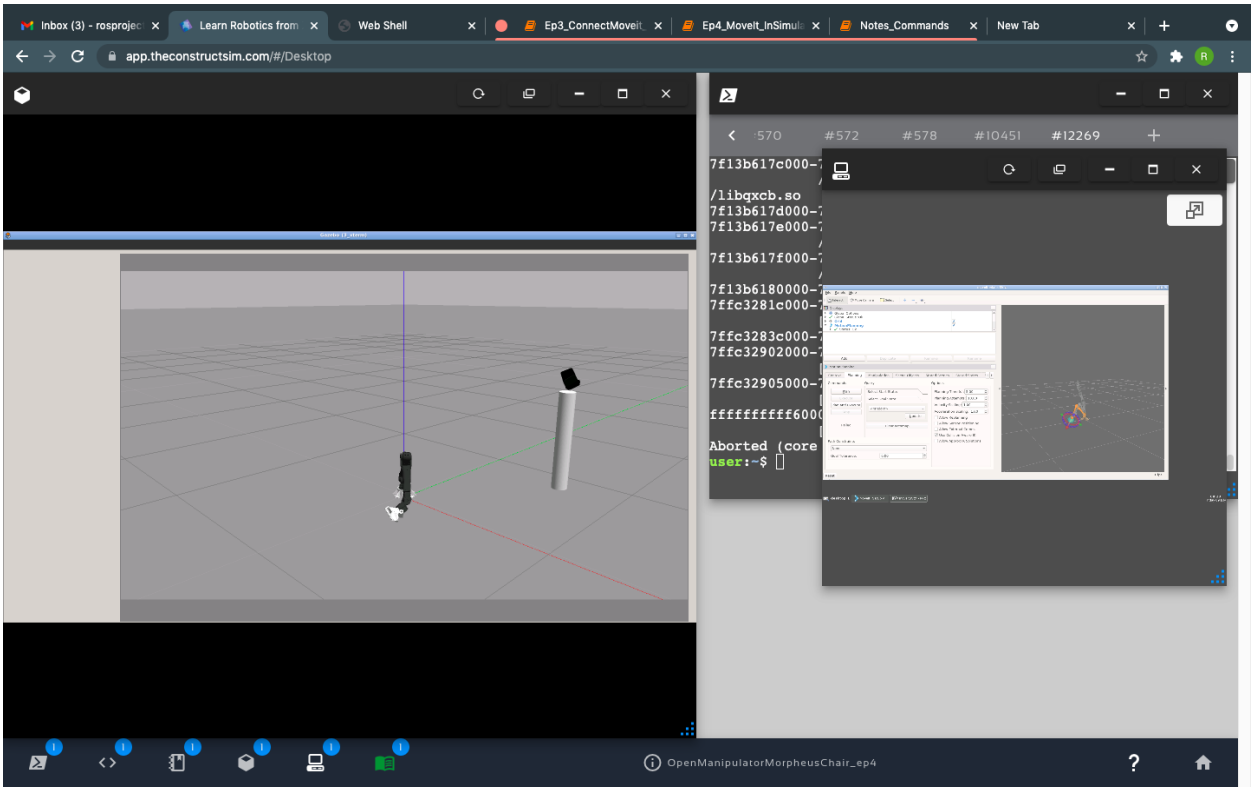


Figure. Changes as seen in Gazebo simulation.

APPENDIX

1. URDF FILE FOR ARM

```
2.  <?xml&nbspversion="1.0"?>
3.  <!--&nbspOpen_Manipulator&nbspChain -->
4.  <robot name="open_manipulator" xmlns:xacro="http://www.ros.org/wiki/xacro">
5.
6.  <!--&nbspImport&nbspall&nbspGazebo-customization&nbspelements,&nbspincluding&nbspGazebo&nbspcolors -->
7.    <xacro:include
filename="$(find&nbspopen_manipulator_support_description)/urdf/open_manipulator.gazebo.xacro"/>
8.  <!--&nbspImport&nbspRviz&nbspcolors -->
9.    <xacro:include filename="$(find&nbspopen_manipulator_support_description)/urdf/materials.xacro" />
10. <!--&nbspImport&nbspinertial&nbspProperties-->
11.   <xacro:include filename="$(find&nbspopen_manipulator_support_description)/urdf/inertial_properties.xacro" />
12. <!--&nbspImport&nbspinertial&nbspProperties-->
13.   <xacro:include filename="$(find&nbspopen_manipulator_support_description)/urdf/mass_properties.xacro" />
14.
15.  <!--&nbspWorld -->
16.    <link name="world">
17.    </link>
18.
19. <!--&nbspWorld&nbspfixed&nbspjoint -->
20.    <joint name="world_fixed" type="fixed">
21.      <parent link="world"/>
22.      <child link="base_link"/>
```

```
23.          </joint>
24.
25.        <!--&nbspBase&nbspLink -->
26.          <link name="base_link">
27.            <inertial>
28.              <mass value="${bl_mass}" />
29.              <origin xyz="${bl_cmx} ${bl_cmy} ${bl_cmz}"/>
30.              <inertia
31.                ixx="${bl_ixx}"
32.                ixy="${bl_ixy}"
33.                ixz="${bl_ixz}"
34.                iyy="${bl_iyy}"
35.                iyz="${bl_iyz}"
36.                izz="${bl_izz}" />
37.            </inertial>
38.            <visual>
39.              <geometry>
40.                  <mesh filename="package://open_manipulator_support_description/meshes/base_link.STL"/>
41.              </geometry>
42.              <material name="grey"/>
43.            </visual>
44.            <collision>
45.              <geometry>
46.                <mesh filename="package://open_manipulator_support_description/meshes/base_link.STL" />
47.              </geometry>
48.            </collision>
49.          </link>
50.
51.        <!--&nbspLink&nbsp1 -->
52.          <link name="link1">
53.            <visual>
54.              <geometry>
55.                <mesh filename="package://open_manipulator_support_description/meshes/link1.STL"/>
56.              </geometry>
```

```
57.             <material name="grey"/>
58.           </visual>
59.
60.           <collision>
61.             <geometry>
62.               <mesh filename="package://open_manipulator_support_description/meshes/link1.STL"/>
63.             </geometry>
64.           </collision>
65.
66.           <inertial>
67.             <mass value="${l1_mass}" />
68.             <origin xyz="${l1_cmx} ${l1_cmy} ${l1_cmz}"/>
69.             <inertia
70.               ixx="${l1_ixx}"
71.               ixy="${l1_ixy}"
72.               ixz="${l1_ixz}"
73.               iyy="${l1_iyy}"
74.               iyz="${l1_iyz}"
75.               izz="${l1_izz}" />
76.           </inertial>
77.         </link>
78.
79.       <!--&nbspJoint&nbsp1 -->
80.         <joint name="id_1" type="revolute">
81.           <parent link="base_link"/>
82.           <child link="link1"/>
83.           <origin xyz="0&nbsp0&nbsp0.0405"/>
84.           <axis xyz="0&nbsp0&nbsp1"/>
85.           <limit velocity="2.8" effort="3.1" lower="${-1.1*pi}" upper="${0.9*pi}" />
86.         </joint>
87.
88.       <!--&nbspTransmission&nbsp1 -->
89.         <transmission name="tran1">
90.           <type>transmission_interface/SimpleTransmission</type>
```

```
91.         <joint name="id_1">
92.           <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
93.         </joint>
94.         <actuator name="motor1">
95.           <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
96.           <mechanicalReduction>1</mechanicalReduction>
97.         </actuator>
98.       </transmission>
99.
100.    <!-- &nbspLink&nbsp2-->
101.     <link name="link2">
102.
103.       <visual>
104.         <geometry>
105.           <mesh filename="package://open_manipulator_support_description/meshes/link2.STL"/>
106.         </geometry>
107.         <material name="grey"/>
108.       </visual>
109.
110.       <collision>
111.         <geometry>
112.           <mesh filename="package://open_manipulator_support_description/meshes/link2.STL"/>
113.         </geometry>
114.       </collision>
115.
116.       <inertial>
117.         <mass value="${l2_mass}" />
118.         <origin xyz="${l2_cmx} ${l2_cmy} ${l2_cmz}"/>
119.         <inertia
120.           ixx="${l2_ixx}"
121.           ixy="${l2_ixy}"
122.           ixz="${l2_ixz}"
123.           iyy="${l2_iyy}"
124.           iyz="${l2_iyz}"
```

```
125.              izz="${l2_izz}" />
126.            </inertial>
127.          </link>
128.
129.        <!-- &nbspJoint&nbsp2 -->
130.          <joint name="id_2" type="revolute">
131.            <parent link="link1"/>
132.            <child link="link2"/>
133.            <origin xyz="0&nbsp0&nbsp0.056"/>
134.            <axis xyz="1&nbsp0&nbsp0"/>
135.            <limit velocity="2.9" effort="9.9" lower="-2.0" upper="2.0" />
136.          </joint>
137.
138.        <!--&nbspTransmission&nbsp2 -->
139.          <transmission name="tran2">
140.            <type>transmission_interface/SimpleTransmission</type>
141.            <joint name="id_2">
142.              <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
143.            </joint>
144.            <actuator name="motor2">
145.              <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
146.              <mechanicalReduction>1</mechanicalReduction>
147.            </actuator>
148.          </transmission>
149.
150.        <!-- &nbspLink&nbsp3 -->
151.          <link name="link3">
152.            <visual>
153.              <geometry>
154.                <mesh filename="package://open_manipulator_support_description/meshes/link3.STL"/>
155.              </geometry>
156.              <material name="grey"/>
157.            </visual>
158.            <collision>
```

```
159.          <geometry>
160.            <mesh filename="package://open_manipulator_support_description/meshes/link3.STL"/>
161.          </geometry>
162.        </collision>
163.
164.        <inertial>
165.          <mass value="${l3_mass}" />
166.          <origin xyz="${l3_cmx} ${l3_cmy} ${l3_cmz}"/>
167.          <inertia
168.            ixx="${l3_ixx}"
169.            ixy="${l3_ixy}"
170.            ixz="${l3_ixz}"
171.            iyy="${l3_iyy}"
172.            iyz="${l3_iyz}"
173.            izz="${l3_izz}" />
174.        </inertial>
175.      </link>
176.
177.      <!-- &nbspJoint&nbsp3 -->
178.        <joint name="id_3" type="revolute">
179.          <parent link="link2"/>
180.          <child link="link3"/>
181.          <origin xyz="0&nbsp0 .1935" />
182.          <axis xyz="1&nbsp0&nbsp0"/>
183.          <limit velocity="2.9" effort="9.9" lower="-2.5" upper="2.5" />
184.        </joint>
185.
186.      <!--&nbspTransmission&nbsp3 -->
187.        <transmission name="tran3">
188.          <type>transmission_interface/SimpleTransmission</type>
189.          <joint name="id_3">
190.            <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
191.          </joint>
192.          <actuator name="motor3">
```

```
193.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
194.         <mechanicalReduction>1</mechanicalReduction>
195.       </actuator>
196.     </transmission>
197.
198.   <!-- &nbspLink&nbsp4 -->
199.     <link name="link4">
200.       <visual>
201.         <geometry>
202.           <mesh filename="package://open_manipulator_support_description/meshes/link4.STL"/>
203.         </geometry>
204.         <material name="grey"/>
205.       </visual>
206.
207.       <collision>
208.         <geometry>
209.           <mesh filename="package://open_manipulator_support_description/meshes/link4.STL"/>
210.         </geometry>
211.       </collision>
212.
213.       <inertial>
214.         <mass value="${l4_mass}" />
215.         <origin xyz="${l4_cmx} ${l4_cmy} ${l4_cmz}"/>
216.         <inertia
217.           ixx="${l4_ixx}"
218.           ixy="${l4_ixy}"
219.           ixz="${l4_ixz}"
220.           iyy="${l4_iyy}"
221.           iyz="${l4_iyz}"
222.           izz="${l4_izz}" />
223.       </inertial>
224.
225.     </link>
226.
```

```
227.    <!-- &nbspJoint&nbsp4-->
228.      <joint name="id_4" type="revolute">
229.        <parent link="link3"/>
230.        <child link="link4"/>
231.        <origin xyz="0&nbsp0&nbsp0.201"/>
232.        <axis xyz="0&nbsp0&nbsp1"/>
233.        <limit velocity="2.8" effort="3.1" lower="${-pi}" upper="${pi}" />
234.      </joint>
235.
236.    <!--&nbspTransmission&nbsp4 -->
237.      <transmission name="tran4">
238.        <type>transmission_interface/SimpleTransmission</type>
239.        <joint name="id_4">
240.          <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
241.        </joint>
242.        <actuator name="motor4">
243.          <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
244.          <mechanicalReduction>1</mechanicalReduction>
245.        </actuator>
246.      </transmission>
247.
248.    <!-- &nbspLink&nbsp5 -->
249.      <link name="link5">
250.        <visual>
251.          <geometry>
252.            <mesh filename="package://open_manipulator_support_description/meshes/link5.STL"/>
253.          </geometry>
254.          <material name="grey"/>
255.        </visual>
256.
257.        <collision>
258.          <geometry>
259.            <mesh filename="package://open_manipulator_support_description/meshes/link5.STL"/>
260.          </geometry>
```

```
261.       </collision>
262.
263.       <inertial>
264.         <mass value="${l5_mass}" />
265.         <origin xyz="${l5_cmx} ${l5_cmy} ${l5_cmz}"/>
266.         <inertia
267.           ixx="${l5_ixx}"
268.           ixy="${l5_ixy}"
269.           ixz="${l5_ixz}"
270.           iyy="${l5_iyy}"
271.           iyz="${l5_iyz}"
272.           izz="${l5_izz}" />
273.       </inertial>
274.     </link>
275.
276.    <!-- &nbspJoint&nbsp5-->
277.     <joint name="id_5" type="revolute">
278.       <parent link="link4"/>
279.       <child link="link5"/>
280.       <origin xyz="0&nbsp0&nbsp0.0405"/>
281.       <axis xyz="1&nbsp0&nbsp0"/>
282.       <limit velocity="2.8" effort="3.1" lower="${-pi/2}" upper="2.2" />
283.     </joint>
284.
285.    <!--&nbspTransmission&nbsp5 -->
286.     <transmission name="tran5">
287.       <type>transmission_interface/SimpleTransmission</type>
288.       <joint name="id_5">
289.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
290.       </joint>
291.       <actuator name="motor5">
292.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
293.         <mechanicalReduction>1</mechanicalReduction>
294.       </actuator>
```

```
295.       </transmission>
296.
297.     <!-- &nbspLink&nbsp6 -->
298.       <link name="link6">
299.         <inertial>
300.           <mass value="${l6_mass}" />
301.           <origin xyz="${l6_cmx} ${l6_cmy} ${l6_cmz}"/>
302.           <inertia
303.             ixx="${l6_ixx}"
304.             ixy="${l6_ixy}"
305.             ixz="${l6_ixz}"
306.             iyy="${l6_iyy}"
307.             iyz="${l6_iyz}"
308.             izz="${l6_izz}" />
309.         </inertial>
310.         <visual>
311.           <geometry>
312.             <mesh filename="package://open_manipulator_support_description/meshes/link6.STL" />
313.           </geometry>
314.           <material name="grey" />
315.         </visual>
316.         <collision>
317.           <geometry>
318.             <mesh filename="package://open_manipulator_support_description/meshes/link6.STL" />
319.           </geometry>
320.         </collision>
321.       </link>
322.
323.     <!-- &nbspJoint&nbsp6 -->
324.       <joint name="id_6" type="revolute">
325.         <origin xyz="0&nbsp0&nbsp0.064"/>
326.         <parent link="link5" />
327.         <child link="link6" />
328.         <axis xyz="0&nbsp0&nbsp1" />
```

```
329.       <limit velocity="2.8" effort="3.1" lower="${-pi}" upper="${pi}" />
330.     </joint>
331.
332.   <!--&nbspTransmission&nbsp6 -->
333.     <transmission name="tran6">
334.       <type>transmission_interface/SimpleTransmission</type>
335.       <joint name="id_6">
336.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
337.       </joint>
338.       <actuator name="motor6">
339.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
340.         <mechanicalReduction>1</mechanicalReduction>
341.       </actuator>
342.     </transmission>
343.
344.   <!--&nbspLink&nbsp7-->
345.     <link name="link7">
346.       <inertial>
347.         <mass value="${l7_mass}" />
348.         <origin xyz="${l7_cmx} ${l7_cmy} ${l7_cmz}"/>
349.         <inertia
350.           ixx="${l7_ixx}"
351.           ixy="${l7_ixy}"
352.           ixz="${l7_ixz}"
353.           iyy="${l7_iyy}"
354.           iyz="${l7_iyz}"
355.           izz="${l7_izz}" />
356.       </inertial>
357.       <visual>
358.         <geometry>
359.           <mesh
360.             filename="package://open_manipulator_support_description/meshes/link7.STL" />
361.         </geometry>
362.         <material name="grey" />
```

```
363.       </visual>
364.       <collision>
365.         <geometry>
366.           <mesh
367.             filename="package://open_manipulator_support_description/meshes/link7.STL" />
368.         </geometry>
369.       </collision>
370.     </link>
371.
372.   <!--&nbspJoint&nbsp7 -->
373.       <joint
374.       name="id_7"
375.       type="revolute">
376.       <origin xyz="0&nbsp0&nbsp0.039" />
377.       <parent link="link6" />
378.       <child link="link7" />
379.       <axis xyz="0&nbsp0&nbsp1" />
380.       <!--
381.       <limit&nbspvelocity="2.8"&nbspeffort="3.1"&nbsplower="${-pi}"&nbspupper="${pi}" />
382.       -->
383.       <limit velocity="2.8" effort="3.1" lower="${-pi}" upper="-1.5" />
384.     </joint>
385.
386.     <!--&nbspTransmission&nbsp7 -->
387.     <transmission name="tran7">
388.       <type>transmission_interface/SimpleTransmission</type>
389.       <joint name="id_7">
390.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
391.       </joint>
392.       <actuator name="motor7">
393.         <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
394.         <mechanicalReduction>1</mechanicalReduction>
395.       </actuator>
396.     </transmission>
```

```
397.
398.     <!-- &nbspGripper&nbspLeft-->
399.
400.   <link name="gripper_l">
401.     <inertial>
402.       <mass value="${g_mass}" />
403.       <origin xyz="${g_cmx} ${g_cmy} ${g_cmz}"/>
404.       <inertia
405.         ixx="${g_ixx}"
406.         ixy="${g_ixy}"
407.         ixz="${g_ixz}"
408.         iyy="${g_iyy}"
409.         iyz="${g_iyz}"
410.         izz="${g_izz}" />
411.     </inertial>
412.
413.     <visual>
414.       <geometry>
415.         <mesh
416.           filename="package://open_manipulator_support_description/meshes/gripper.STL" />
417.       </geometry>
418.       <material name="grey" />
419.     </visual>
420.
421.    <collision>
422.       <geometry>
423.         <mesh
424.           filename="package://open_manipulator_support_description/meshes/gripper.STL" />
425.       </geometry>
426.     </collision>
427.   </link>
428.
429.   <!--Gripper&nbspLeft&nbspJoint -->
430.     <joint name="joint8" type="prismatic">
```

```
431.          <origin xyz="-0.015&nbsp0&nbsp0.0595" rpy="0&nbsp0 ${pi}"/>
432.          <parent link="link6" />
433.          <child link="gripper_l" />
434.          <axis xyz="1&nbsp0&nbsp0" />
435.          <limit velocity="0.1" effort=".1" lower="-.033" upper="0"/>
436.          <mimic joint="id_7" multiplier="-0.01" offset="-0.0110133334" />
437.        </joint>
438.
439.       <!-- &nbspGripper&nbspRight -->
440.         <link name="gripper_r">
441.          <inertial>
442.            <mass value="${g_mass}" />
443.            <origin xyz="${-g_cmx} ${g_cmy} ${g_cmz}"/>
444.            <inertia
445.              ixx="${g_ixx}"
446.              ixy="${g_ixy}"
447.              ixz="${g_ixz}"
448.              iyy="${g_iyy}"
449.              iyz="${g_iyz}"
450.              izz="${g_izz}" />
451.          </inertial>
452.            <visual>
453.            <geometry>
454.              <mesh filename="package://open_manipulator_support_description/meshes/gripper.STL" />
455.            </geometry>
456.            <material name="grey" />
457.          </visual>
458.
459.          <collision>
460.            <geometry>
461.              <mesh filename="package://open_manipulator_support_description/meshes/gripper.STL" />
462.            </geometry>
463.          </collision>
464.        </link>
```

```
465.
466.      <!--Gripper&nbspRight&nbspJoint -->
467.        <joint name="joint9" type="prismatic">
468.         <origin xyz="0.015&nbsp0&nbsp0.0595" />
469.         <parent link="link6" />
470.         <child link="gripper_r" />
471.         <axis xyz="1&nbsp0&nbsp0" />
472.            <limit velocity="0.1" effort=".1" lower="0" upper=".033"/>
473.         <mimic joint="joint8" multiplier="1" offset="0" />
474.        </joint>
475.
476.      </robot>
```

Conclusion

In addition to the points discussed in our project paper/ report, completing this project successfully has been very beneficial to the group members in many additional ways.

The **importance of having a clear project plan** (of constructing a virtual environment for the **Wheeled and Arm Robot**, respectively), and following it as closely as possible has been obvious in all stages of the work.

However, we have also tried to incorporate all the learn adaptability when needed was one of the key points to achieve success in a project.

We have also learned that doing a project for the sake of itself, no matter how interesting, is never enough and business aspects should always be considered. In addition, it was helpful to be part of a project from the very beginning until its completion, and observe and solve many kinds of problems that arise during its implementation.

In a brief conclusion, the project has been completed successfully as a result of hard work, careful planning, good teamwork and supportive instruction.

It began as an idea to assist in further research in the domains of ROS and simulations and has but, developed to an end-product that has successfully fulfilled its desired objective

References

BOOKS:

1. Learning Robotics Using Python, by Lentin Joseph,
Copyright © 2015 Packt Publishing, ISBN 978-1-78328-753-6.

2.Learning ROS for Robotics Programming, by Enrique Fernandez, Luis Sanchez Crespo, Anil Mahtani, Aaron Martinez ; Second Edition, Copyright © 2015 Packt Publishing, ISBN 978-1-78398-750-0.

Websites :

[0]Workspace and simulation interface :[Online] Available at: https://apptheconstructsim.com

[1] About ROS. [Online] Available at: http://www.ros.org/about-ros [Retrieved 27.5.2021]

[2] ROS Nodes. [Online] Available at: http://wiki.ros.org/Nodes [Retrieved 28.5.2021]

[3] ROS Topics. [Online] Available at: http://wiki.ros.org/Topics [Retrieved 29.5.2021]

[4] Launch Files. [Online] Available at: http://wiki.ros.org/roslaunch [Retrieved 30.5.2021]

[5] Gazebo. [Online] Available at: http://wiki.ros.org/gazebo [Retrieved 01.6.2021]

[6] How to Setup a Developer PC for ROS.

[Online] Available at:

https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/ ROS2019/how_to_setup_developer_pc.pdf [Retrieved 02.6.2021]

[7] SDF to URDF converter. [Online] Available at:

http://answers.gazebosim.org/question/1310/sdf-to-urdf-converter/

[Retrieved 07.6.2021].