PICK AND PLACE ROBOT

(MOTION PLANNING)

REPORT

Submitted By

CB.EN. U4AIE20040 - MENTA SAI AKSHAY

CB.EN. U4AIE20062 - SAI ARAVIND V

……………………

For the Completion of

21AIE213 ROBOTIC OPERATING SYSTEMS&ROBOT SIMULATION

CSE - AI

13th July 2022

ACKNOWLEDGEMENT

We would like to thank all those who have helped us in completing this project of "**PICK AND PLACE ROBOT**" under the subject "**21AIE213 ROBOTIC OPERATING SYSTEMS&ROBOT SIMULATION**".

We would like to show our sincere gratitude to our professor SAJITH VARIYAR without whom the project would not have initiated, who taught us the basics to start and visualise the project and enlightened us with the ideas regarding the project, and helped us by clarifying all the doubts whenever being asked.

We would like to thank ourselves. We helped each other and taught each other about various concepts regarding the project which helped in increasing our inner knowledge.

INDEX

ABSTRACT:

Industrial robots are the most widely made and utilized robots in the manufacturing industry. In automotive material handling and general industrial applications, the pick and place robots is developed for maximum flexibility, efficiency, and durability. Now a day a variety, if technologies are employed to aid in the advancement and progress of robotic activities such as simulation modelling and analysis of robots, the first objective of our project, is to implement a pick and place robot that helps us to pick the objects and place it on the required position here we have implemented this robot in gazebo using the Rviz and python and c-Make.

OBJECTIVE:

- Plan and execute a path from robot to a coordinate on the filter radio surface where object is located.
- Plan and execute a path back of our robot
- Be able to detect and avoid all the obstacles and re-plan the path if there is a risk for collision.

TOOLS:

- Gazebo simulator
- RVIZ

INTRODUCTION:

Robots are usually expensive and quite susceptible to damage and undesired interaction with themselves or their surroundings. Three-dimensional simulation is a sustainable method to test and rapidly iterate during robot application development. Robot Operating System (ROS) is a middleware for multiple robot software tools and also provides simulation platforms like Rviz and Gazebo.

Pick and place robots have become common place in today’s manufacturing environment. Typically relegated to simple repetitive and monotonous tasks that robots naturally excel at pick and place robots bring a number of benefits for manufactures. They usually mounted on a stable stand, strategically positioned to reach their entire work envelope advanced vision systems enable them to grasp and move objects on a conveyor belt, which can be used in a variety of different ways. Pick and place robots are used for Assembly, Packaging, Bin Picking and Inspection. The main benefits of pick and place robots are speed and consistency. For these robots the output can even often reach about 200 products per minute.

Different Parts of Pick and Place Robot:

A pick and place robot has several dedicated parts, such as:

Robot Arm tool:

A robotic arm, also known as a manipulator, is the extension of the robot by using cylindrical or spherical parts. links, and joints.

End Effector:

The end effector is the accessory at the end of the robotic arm, that does the required job such as gripping objects. The end effectors can be designed to perform different functionalities based on requirements.

Actuators:

Actuators create the motion in the robotic arm and end effectors. The linear actuators are basically any type of motor, such as servo motor, stepper motor, or hydraulic cylinder.

Controllers:

Controllers synchronize and control the movement of different actuators of a robot, thereby being the brain behind the smooth robotic operation.

Robotic Arm:

Robotic arms are the most common robots to exist in the last century. They are built out of multiple actuators and passive links forming an actuated chain that can be used to manipulate objects using different end-effectors. They are extremely useful in industry as well as personal settings for assembly, drilling, machining, pick-and-place, and other applications. A robot arm could have revolute or prismatic joints for rotation or linear motion respectively.

Working:

- The basic function of a pick and place robot is done by its joints. Joints are analogous to human joints and are used to join the two consecutive rigid bodies in the robot. They can be rotary joint or linear joint.
- To add a joint to any link of a robot, we need to know about the degrees of freedom and degrees of movement for that body part.
- Degrees of freedom implement the linear and rotational movement of the body and Degrees of movement imply the number of axis the body can move

A simple pick and place robot consists of two rigid bodies on a moving base, connected with rotary joint. A rotary joint is a one which provides rotation in 360 degrees around any one of the axes.

The bottom or the base is attached to the ground.

The 1st rigid body is fixed and supports the second rigid body to which the end effector is provided.

The 2nd rigid body is provided with movement in all 3 axes and has 3 degrees of freedom. It is connected to the 1st body with a rotational joint.

The end effector should accommodate all 6 degrees of freedom, to reach all sides of the component, to take up position to any height.

Hierarchy of motion Planning:

Task planning –

Designing a set of high-level goals, such as "go pick up the object in front of you".

Path planning –

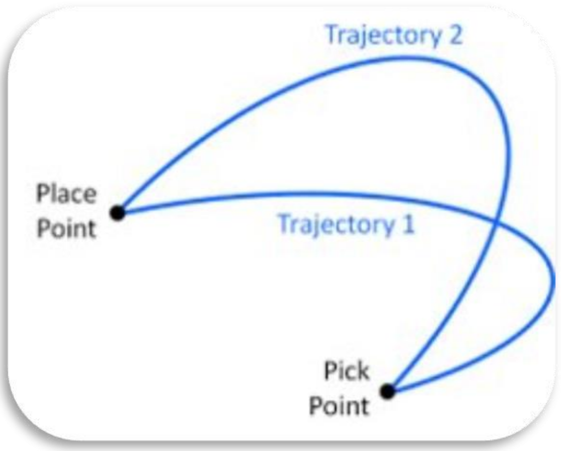
Generating a feasible path from a start point to a goal point. A path usually consists of a set of connected waypoints.

Trajectory planning –

Generating a time schedule for how to follow a path given constraints such as position, velocity, and acceleration.

Low level control –

Once the entire trajectory is planned, there needs to be a control system that can execute the trajectory in a sufficiently accurate manner.



Trajectory Planning:

Trajectory: Time evolution of position, velocity and acceleration.

Given a change in the end effector position, a smooth transition from initial to the final pose of the end effector is required. This is achieved by change in joint angles, velocity and acceleration.

Inverse Kinematics gives change in joint variable. When the additional dimension of time is incorporated, velocity and acceleration of each joint can be found.

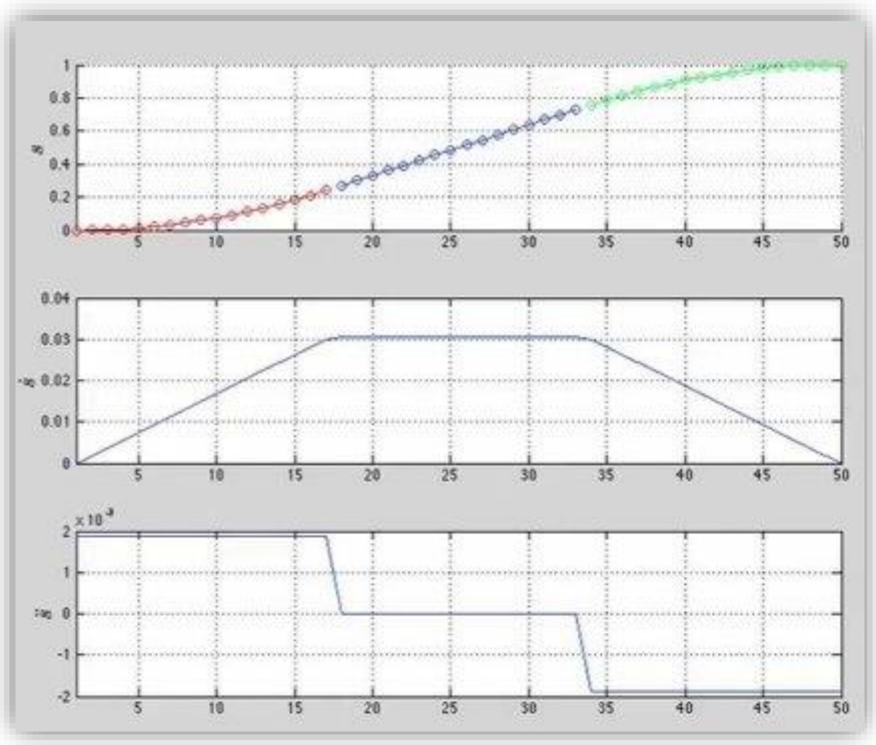
<u>Types of Trajectory Function:</u>

There are various ways to create trajectories that interpolate pose (or joint configurations) over time.

Trapezoidal Velocity trajectory

Trapezoidal velocity trajectories are piecewise trajectories of constant acceleration, zero acceleration, and constant deceleration. This leads to a trapezoidal velocity profile, and a "linear segment with parabolic blend" (LSPB) or s-curve position profile.

This parameterization makes them relatively easy to implement, tune, and validate against requirements such as position, speed, and acceleration limits.



In the following picture, the graph on top represents the displacement graph and the graph in the middle represents the velocity graph and the lower graph represents the acceleration graph.

The name Trapezoidal because, as we can observe the velocity graph it is like as trapezoid with 2/3rd of graph of velocity is a function of time and the remaining 1/3rd means the middle part is constant.

From this observation we can say that when velocity is a function of time then, the integration of it will be displacement and thus it will the quadratic equation and the following graph of that 2/3rd part will be a parabola, but the remaining part will be a straight line, because velocity is constant.

And the drawback of trapezoidal velocity Trajectory is, when we observe the acceleration graph we can see that it contains a discontinuity in middle when velocity is changing from increasing to constant and from constant to decrease.

Polynomial Trajectory

We can interpolate between two waypoints using polynomials of various orders.

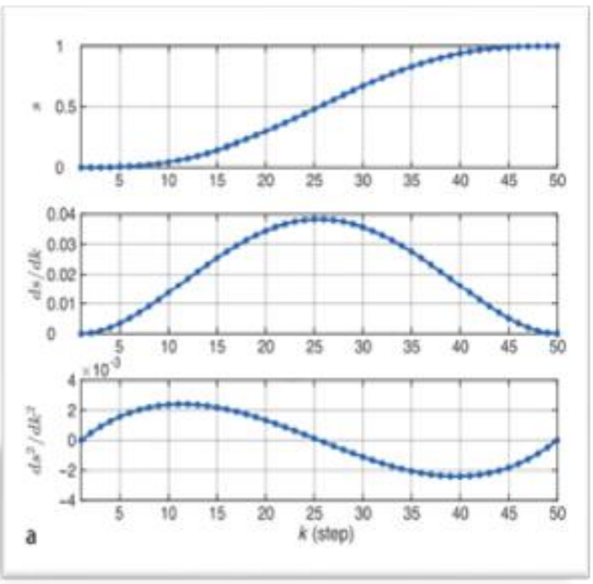
Cubic (3rd order) – Requires 4 boundary conditions: position and velocity at both ends

Quintic (5th order) – Requires 6 boundary conditions: position, velocity, and acceleration at both ends

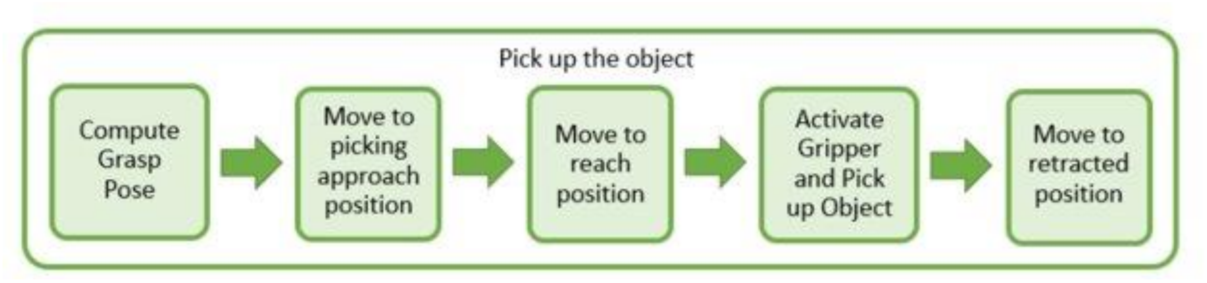
- ➢ The trajectory has defined boundary conditions for position, velocity and acceleration and frequently the velocity and acceleration boundary conditions are all zero
- ➢ We have 6 constraints.
- ➢ So, we use quintic polynomial

$$s(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F$$

Graph of displacement velocity and acceleration of end effector using quintic polynomial,



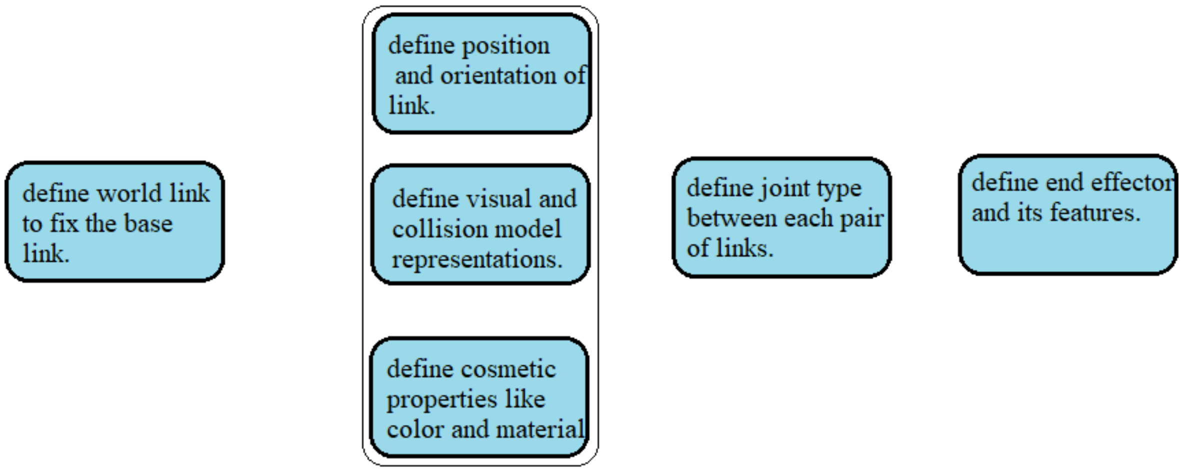
Visualizing the Robot in Rviz:


Pick up the object
Compute Grasp Pose
Move to picking approach position
Move to reach position
Activate Gripper and Pick up Object
Move to retracted position

- Once the waypoints are guaranteed to be collision free, they can be passed on to the trajectory processing module. This module adds timing information to the different waypoints such that we are not only aware of where we should be but also at what times should we be at each waypoint.
- In essence, velocity and acceleration information is appended to the waypoints so that, the desired motions can be executed on a robot.

Designing the Robotic Arm in Xacro:

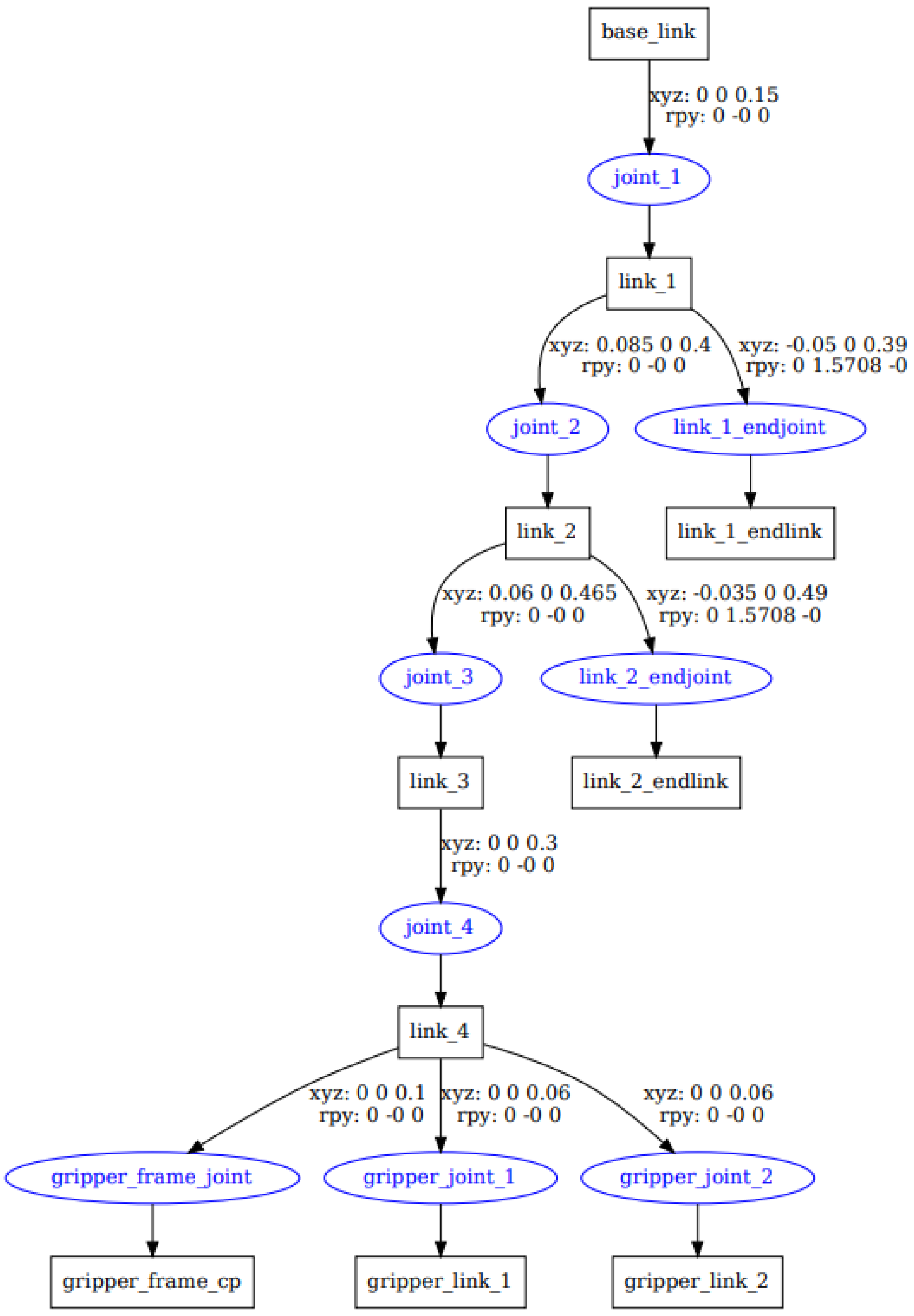
We designed a simple robot with 3 DOF (Revolute-Revolute-Revolute robot arm). A Universal Robot Description Format (URDF) file with the robot's visual and collision model representation is needed for the robot's interaction with ROS.


define world link to fix the base link.
define position and orientation of link.
define visual and collision model representations.
define cosmetic properties like color and material
define joint type between each pair of links.
define end effector and its features.

Steps to define a robotic arm URDF File

The URDF file is an XML file with dedicated tags for physical features like the material and color of the links. For the 3D visual representation of the robot links, the <visual></visual> tags are used. They carry information about the geometry (within <geometry></geometry>

tags) which can be a primitive shape usually centered at its geometric center or a mesh with the center defined in the 3D modeling software itself. he collision model is similar to the visual model within <collision></collision> tags However, in order to get collision detection to work or to simulate the robot in something like Gazebo, we need to define a collision element as well. The joints are defined within the <joint></joint> with information about the origin, axis of rotation, parent and child links, and limits for positions.

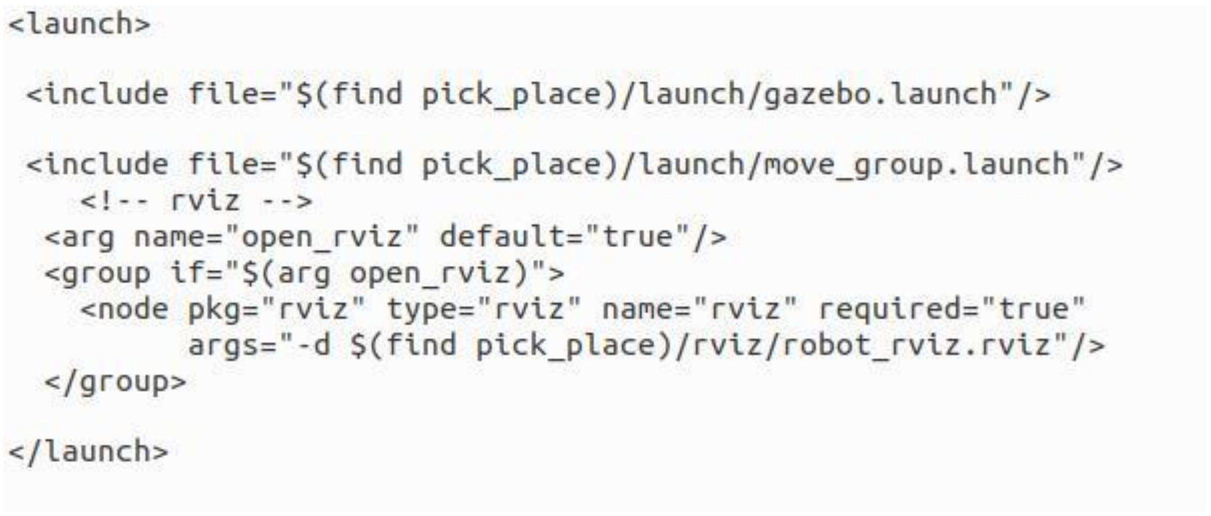


To start the arm in Rviz

Launch files usually bring up a set of nodes for the package that provide some aggregate functionality.

- roslaunch pick_place robot.rviz

main.launch



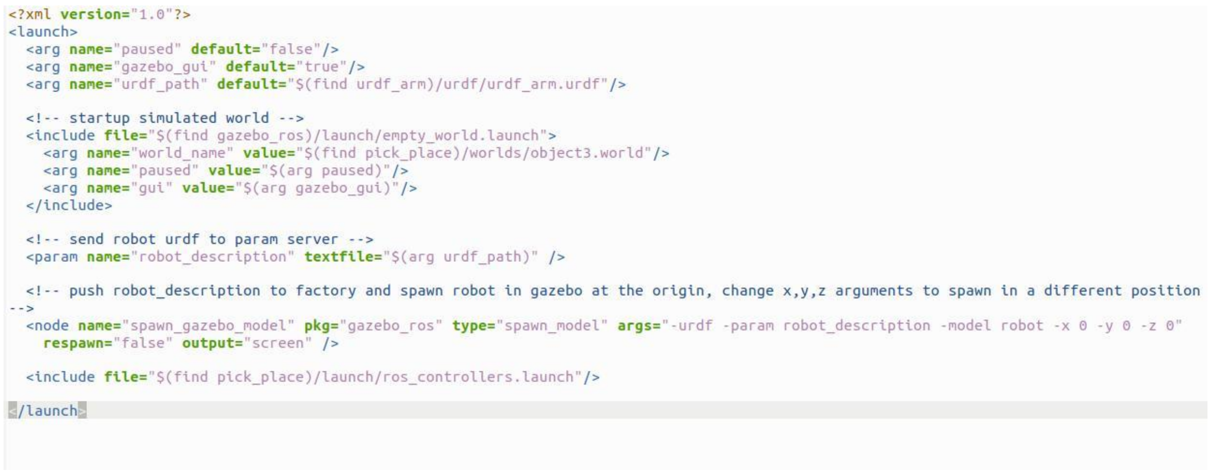
```
<launch>

 <include file="$(find pick_place)/launch/gazebo.launch"/>

 <include file="$(find pick_place)/launch/move_group.launch"/>
    <!-- rviz -->
  <arg name="open_rviz" default="true"/>
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find pick_place)/rviz/robot_rviz.rviz"/>
  </group>

</launch>
```

gazebo.launch



```
<?xml version="1.0"?>
<launch>
  <arg name="paused" default="false"/>
  <arg name="gazebo_gui" default="true"/>
  <arg name="urdf_path" default="$(find urdf_arm)/urdf/urdf_arm.urdf"/>

  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find pick_place)/worlds/object3.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gazebo_gui)"/>
  </include>

  <!-- send robot urdf to param server -->
  <param name="robot_description" textfile="$(arg urdf_path)" />

  <!-- push robot_description to factory and spawn robot in gazebo at the origin, change x,y,z arguments to spawn in a different position
-->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />

  <include file="$(find pick_place)/launch/ros_controllers.launch"/>

</launch>
```

<u>Using URDF in gazebo:</u>

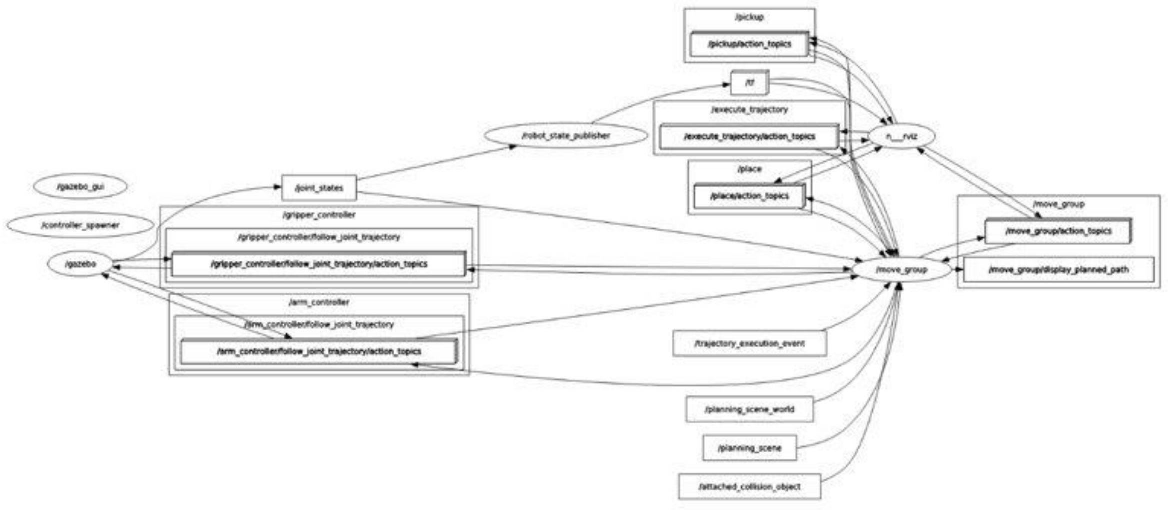
urdf_arm. urdf

```
    </link>
    <gazebo reference="link1">
        <material>Gazebo/Black</material>
       <turnGravityOff>false</turnGravityOff>
    </gazebo>
    <!-- JOINT 1 -->
    <joint name="joint1" type="continuous">
        <origin rpy="0 0 0" xyz="0 0 0.01" />
        <!--parent linknte distance -->
        <parent link="base_link" />
        <child link="link1" />
        <axis xyz="0 0 1" />
    </joint>
    <!-- LINK 2 -->
    <link name="link2">
        <visual>
            <origin rpy="0 0 0" xyz="0 0 0.15" />
            <geometry>
                <cylinder length="0.30" radius="0.03" />
            </geometry>
            <material name="silver">
                <color rgba="0.75 0.75 0.75 1" />
            </material>

        </visual>
        <collision>
            <origin rpy="0 0 0" xyz="0 0 0.15" />
            <geometry>
                <cylinder length="0.30" radius="0.03" />
            </geometry>
        </collision>

<inertial>
            <mass value="0.1" />
            <inertia ixx="0.03" iyy="0.03" izz="0.03" ixy="0.0" ixz="0.0" iyz="0.0" />
        </inertial>
```

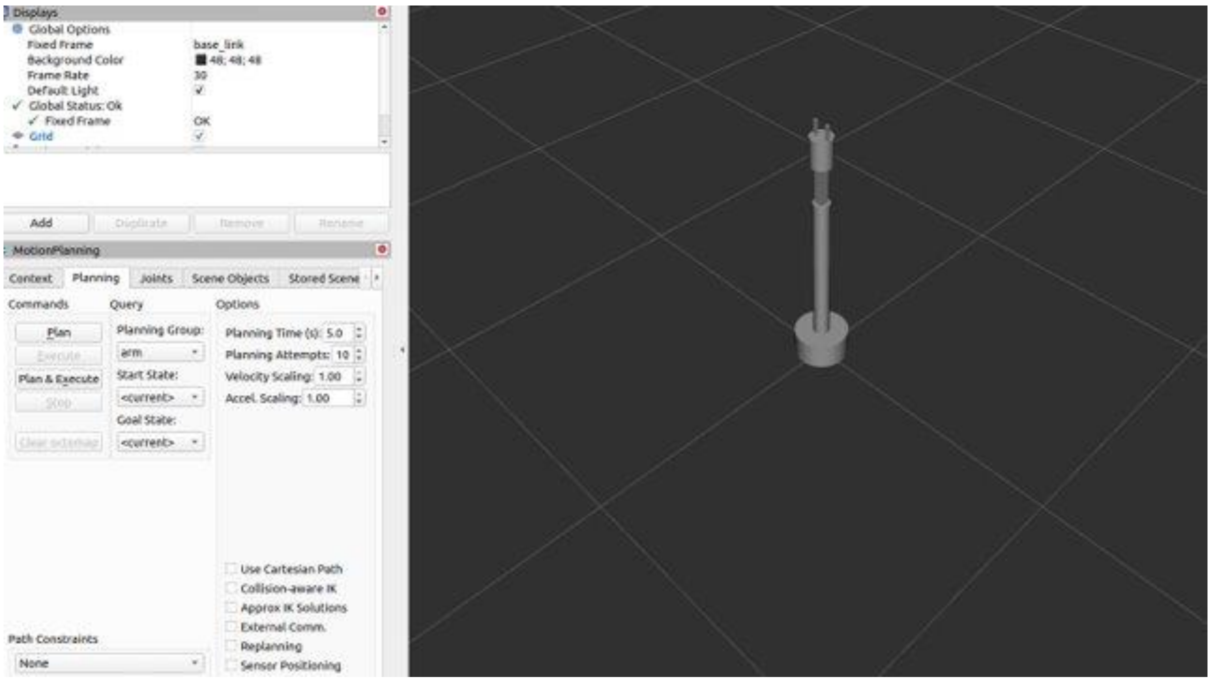
Rqt_Graph:


/pickup
/pickup/action_topics
/tf
/execute_trajectory
/execute_trajectory/action_topics
/robot_state_publisher
n__rviz
/place
/place/action_topics
/gazebo_gui
/joint_states
/controller_spawner
/gripper_controller
/gripper_controller/follow_joint_trajectory
/gazebo
/gripper_controller/follow_joint_trajectory/action_topics
/move_group
/move_group/action_topics
/move_group/display_planned_path
/arm_controller
/arm_controller/follow_joint_trajectory
/arm_controller/follow_joint_trajectory/action_topics
/trajectory_execution_event
/planning_scene_world
/planning_scene
/attached_collision_object

- The above figure is the Ros node graph of our simulation. We have used The Move Group interface is a collection of APIs to access the various capabilities of the move group ROS node of Move It. For our project we have two robotic arm groups named robotic arm and a gripper.

- Trajectories are executed on simulated robots or real hardware, and they are controlled by controllers. Gazebo uses controllers to control the motion and to know the poses and status of the robot.
- Gazebo uses the JointStateController to publish current joint values and JointTrajectoryController to control the execution of trajectories. . Communication between all these components happens over ROS topics and action servers.
- Trajectory controllers contain a name, a type, a list of associated joints, the gains, their constraints, and some other information to do with tolerances. Controllers are provided as ROS action servers, and they need to be powered by nodes. We can launch them using launch files.

6-DOF Robotic Arm in Rviz:



Global Options
Fixed Frame: base_link
Background Color: 48; 48; 48
Frame Rate: 30
Default Light: ✓
✓ Global Status: Ok
✓ Fixed Frame: OK
Grid: ✓

Planning Group: arm
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

RRT(Rapidly exploring random trees) Algorithm:

First, we'll initialize an empty tree. Next, we'll insert the root node that represents the start position into the tree. At this point, we'll just have a tree with a single node that represents the start position. Finally, we'll repeat these steps until we hit the number of iterations or reach the goal, whichever one comes first.

Parameters of RRT

- **Map:** the map of the environment that's partitioned into an obstacle region and an obstacle-free region.

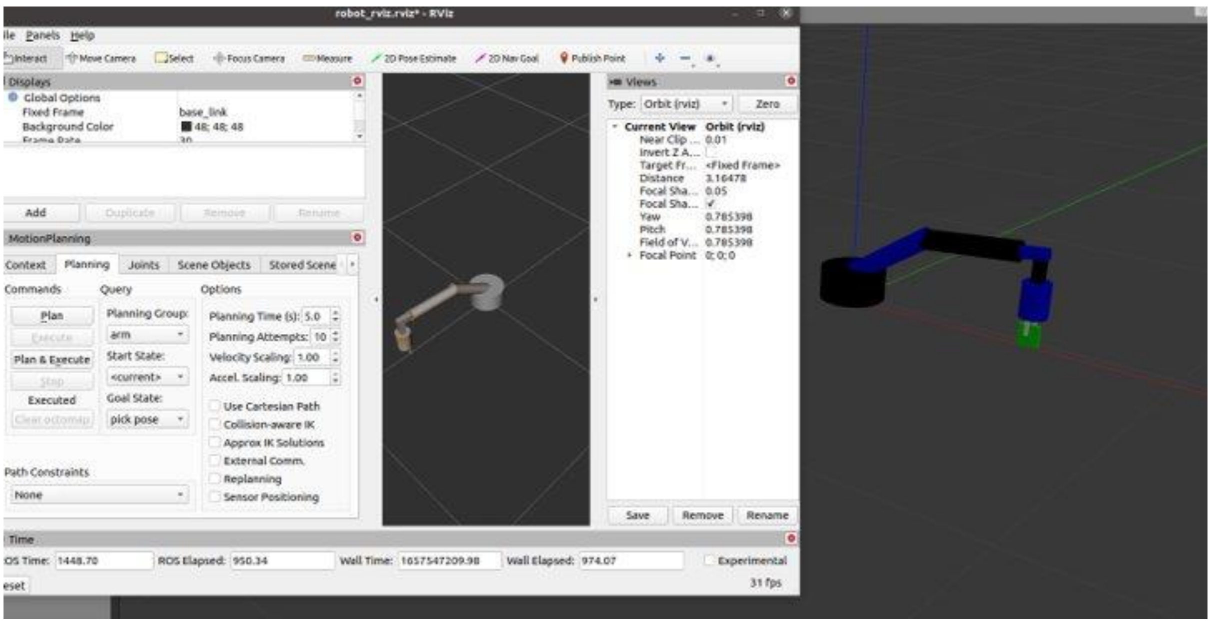
- **Start Position:** the start position of the robot in its environment.
- **Goal Region:** the goal region of the robot in its environment.
- **Number of Iterations:** the number of iterations performed by RRT.

Algorithm 1 RRT Algorithm

```
Generate_RRT($x_{init}, K, \Delta_t$)
$G\_init(t)(x_{init}()$;
for $i \leftarrow 1$ to $K$ do
    $x_{rand} \leftarrow$ Random_State();
    $x_{near} \leftarrow$ NEAREST_Neighbor($x_{rand}$,G);
    u $\leftarrow$ Select_Input($x_{rand}, x_{near}$);
    $xq_{new} \leftarrow$ New_State($x_{near}$,u,$\delta_t$);
    G.add_vertex($x_{new}$);
    G.add_edge($x_{near}, x_{new}, u$);
    retuen G
end for
```

<u>GAZEBO SIMULATION:</u>

Pick Pose:


robot_rviz.rviz* - RViz
ile Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
MotionPlanning
Context Planning Joints Scene Objects Stored Scene
Commands Query Options
Plan Execute Plan & Execute Stop Executed Clear octomap
Views
Zero
Save Remove Rename
Time
eset

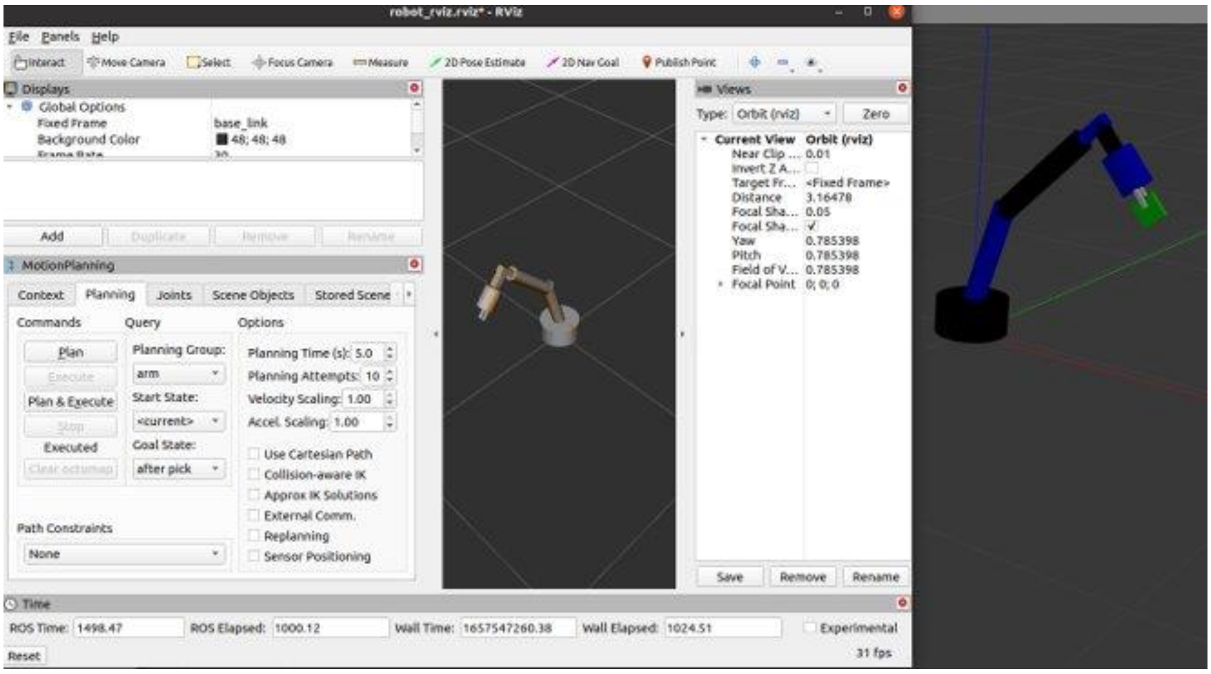
Global Options
Fixed Frame: base_link
Background Color: 48; 48; 48
Frame Rate: 30

Planning Group: arm
Start State: <current>
Goal State: pick pose
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning

Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0.01
Invert Z A...:
Target Fr...: <Fixed Frame>
Distance: 3.16478
Focal Sha...: 0.05
Focal Sha...: ✓
Yaw: 0.785398
Pitch: 0.785398
Field of V...: 0.785398
Focal Point: 0; 0; 0

OS Time: 1448.70 ROS Elapsed: 950.34 Wall Time: 1657547209.98 Wall Elapsed: 974.07 Experimental
31 fps

After Pick Pose:



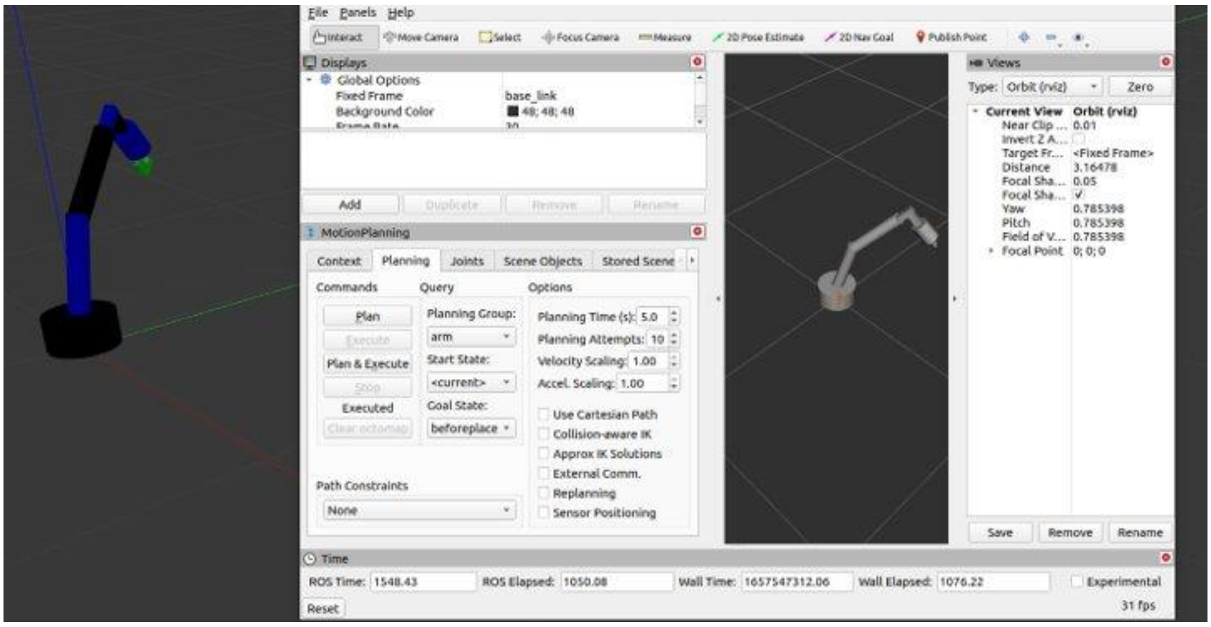
Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30

Planning Group: arm
Start State: <current>
Goal State: after pick
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Path Constraints: None

Type: Orbit (rviz) Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z A...
Target Fr... <Fixed Frame>
Distance 3.16478
Focal Sha... 0.05
Focal Sha... ✓
Yaw 0.785398
Pitch 0.785398
Field of V... 0.785398
Focal Point 0; 0; 0

ROS Time: 1498.47 ROS Elapsed: 1000.12 Wall Time: 1657547260.38 Wall Elapsed: 1024.51
31 fps

Before Place Pose:


File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
MotionPlanning
Context Planning Joints Scene Objects Stored Scene
Commands Query Options
Plan Execute Plan & Execute Stop Executed Clear octomap
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
Views
Save Remove Rename
Time
Experimental
Reset

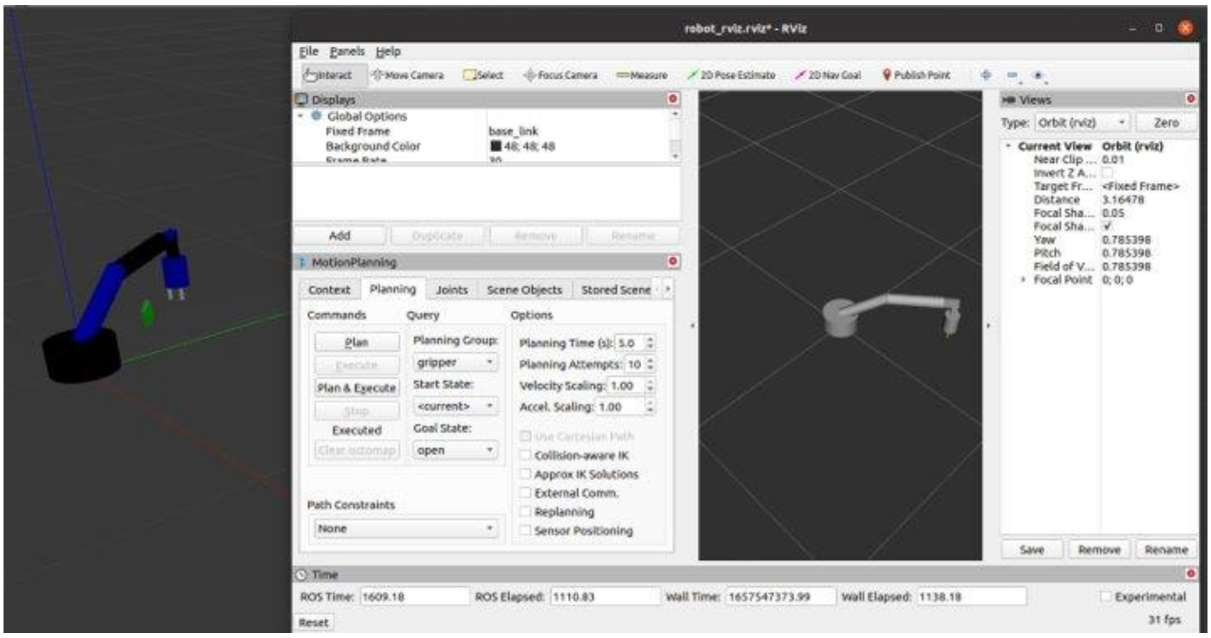
Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30

Planning Group: arm
Start State: <current>
Goal State: beforeplace
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Path Constraints: None

Type: Orbit (rviz) Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z A...
Target Fr... <Fixed Frame>
Distance 3.16478
Focal Sha... 0.05
Focal Sha... ✓
Yaw 0.785398
Pitch 0.785398
Field of V... 0.785398
Focal Point 0; 0; 0

ROS Time: 1548.43 ROS Elapsed: 1050.08 Wall Time: 1657547312.06 Wall Elapsed: 1076.22
31 fps

Place Pose:



Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30

Planning Group: gripper
Start State: <current>
Goal State: open
Executed
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Type: Orbit (rviz)
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z A... ☐
Target Fr... <Fixed Frame>
Distance 3.16478
Focal Sha... 0.05
Focal Sha... ✓
Yaw 0.785398
Pitch 0.785398
Field of V... 0.785398
Focal Point 0; 0; 0

ROS Time: 1609.18 ROS Elapsed: 1110.83 Wall Time: 1657547373.99 Wall Elapsed: 1138.18 ☐ Experimental
31 fps

References:

https://youtu.be/3Kmlpe8kgbk

https://github.com/Salman-H/pick-place-robot

https://www.researchgate.net/publication/332565132_Pick_and_Place_Robotic_Arm_Using_Arduino

https://www.irjet.net/archives/V8/i2/IRJET-V8I2311.pdf

Thank You