DESIGN AND FABRICATION OF THREE FINGERED GRIPPER : SIMULATION AND CONTROL USING ROS

Mini Project Report

Submitted in partial fulfillment of the award of the degree of M.Tech in

Robotics and Automation

of the APJ Abdul Kalam Technological University

GOKUL SOMAN

Register No: TVE19ECRA08

Second Semester

Master of Technology

CET CENTRE FOR INTERDISCIPLINARY RESEARCH

COLLEGE OF ENGINEERING TRIVANDRUM

KERALA

2020

CET CENTRE FOR INTERDISCIPLINARY RESEARCH

COLLEGE OF ENGINEERING TRIVANDRUM

THIRUVANANTHAPURAM - 17

Certificate

*This is to certify that this report entitled **"DESIGN AND FABRICATION OF THREE FINGERED GRIPPER : SIMULATION AND CONTROL USING ROS"** is a bonafide record of the miniproject by **Mr. Gokul Soman, Reg. No: TVE19ECRA08** and of Second Semester, M.Tech under our guidance towards the partial fulfillment of the requirements for the award of **M.Tech in Robotics and Automation** of the **APJ Abdul Kalam Technological University** during the year 2020.*

Guided By:	Head Of Department:
Prof. Ajith R R	**Dr. Sindhu G**
Asst. Professor	Dean (Research)
Dept. of Mechanical Engg.	College of Engineering,
College of Engineering, Tvm	Trivandrum

Mini Project Co-ordinators

Dr. Ranjith S Kumar	**Prof. Lalu V**
Asst. Professor	Asst. Professor
Dept. of Mechanical Engg.	Dept. of Electronics Engg.
College of Engineering, Tvm	College of Engineering,Tvm

ABSTRACT

The objective of this project is to design and develop a three fingered gripper with 8 DOF that can reconfigure itself to grasp different targets. It is an underactuated system with 3 articulated fingers driven by four motors. Each finger can be grasped with two degrees of freedom by a single motor, one of which is fixed to the frame, and the other two fingers can also rotate in the opposite direction with two degrees of freedom along the palm surface. Grasping action of each finger is independently controlled by a motor. The symmetrically opposable fingers centered on parallel joint axes can be rotated from 0 to 180° by the fourth motor.

The prototype could be used as an end effector of ABB IRB 120 in Robotics lab. Underactuated hands could be used in many situations for its lightweight and flexible features for applications in medical, aerospace and unmanned systems.

The procedure includes different stages like specification definition, mechanical design, torque computation for motor selection, simulation, developing the control system and prototype testing. These stages cover various aspects like stress analysis, forward and inverse kinematics, and dynamic modelling. Finally, the developed prototype will be tested to see if it meets the specification definition.

Acknowledgment

I have great pleasure in expressing my gratitude and obligations to **Prof. Jaimon Cletus**, Guide, Department of Mechanical Engineering, College of Engineering, Trivandrum for his valuable guidance and suggestions to make this work a great success.

I express my thanks to **Dr. Ranjith S Kumar**, Asst. Professor Department of Mechanical Engineering, College of Engineering, Trivandrum for all the guidance and encouragement in the fulfillment of this work.

I express my gratitude to **Prof. Lalu V**, Asst. Professor Department of Electronics Engineering, College of Engineering, Trivandrum, for all necessary help extended to me in the fulfillment of this work.

I express my gratitude to **Dr. Jisha V R**, Asst. Professor , Department of Electrical Engineering, College of Engineering, Trivandrum for all the support extended to me in the fulfillment of this work.

I also acknowledge my gratitude to other members of faculty in the Department of Electrical Engineering ,Mechanical Engineering, Electronics Engineering, staffs of FAB lab, staffs of machine shop, staffs of CET school of management, my family and friends for their whole hearted cooperation and encouragement. Above all, I thank GOD Almighty, without whose help, I wouldn't have reached this far.

Contents

List of Figures v

List of Tables vi

1 Introduction 1

1.1 Three Fingered Gripper 1

1.2 Underactuation 2

1.3 Objectives 2

1.4 Outline of the Report 2

2 Literature Review 4

3 Mechanical Design and Modelling 5

3.1 Methodology 5

3.2 Robot Specifications 6

3.3 System Description 6

4 Simulation Using ROS 7

4.1 Modelling The Robot URDF 8

4.2 Generating Move-It Package 11

4.2.1 Steps for generating moveit! package 11

4.3 Motion planning of gripper in RViz 14

4.4 Gripper simulation using Gazebo 16

4.4.1 Simulation in Gazebo 16

5 Simulation Results **18**

6 Conclusions **19**

7 References **20**

List of Figures

3.1 Block diagram of RRP configured robot 6

4.1 Creating URDF using URDF plugin in Solidworks 10

4.2 Moveit Setup Assistant startup 12

4.3 Defining Virtual Joints 13

4.4 Planning groups in MoveIt 13

4.5 Defining Planning Groups 14

4.6 Motion planning of robot in RViz 15

4.7 Launching the model in Gazebo 17

4.8 Corresponding execution in Gazebo 17

List of Tables

Chapter 1

Introduction

1.1 Three Fingered Gripper

The robot end effector is the bridge between the robot arm and the environment around it. The actions of the gripper vary with the tasks. A large variety of gripper designs mostly two fingered can be seen in industries performing assembly tasks. An object with circular cross-section cannot be held properly with these set of two fingers atleast not without indeterminacy of position. Since circular and rectangular objects partly or wholly characterize the vast majority of parts encountered in industry in general, a ”universal” robot gripper must be able to handle these effectively.

From an anthropomorphic point of view, Schlesinger has defined the six basic prehensile patterns of the human hand. Their mechanical equivalents require a minimum of three fingers. Therefore, three fingers seem to be essential for the manipulation of objects in general.

include pic of prehensile patterns

The mechanical equivalents of basic prehensile patterns: (a) Two-finger, b) three-finger concentric, and c) three- finger wrap. These three modes allow grasping of most objects commonly found in assembly operations.

1.2 Underactuation

Underactuated robotic hands are the intermediate solution between robotic hands for manipulation , which have the advantages of being versatile, guarantee a stable grasp, but they are expensive, complex to control and with many actuators; and robotic grippers, whose advantages are simplified control, few actuators, but they have the drawbacks of being task specific, and perform an unstable grasp.

An underactuated mechanism allows the grasping of objects in a more natural and more similar to the movement obtained by the human hand. [?]

1.3 Objectives

The project deals with design, simulation and fabrication of a three fingered gripper . The primary objectives of this project are:-

- Design a 3 fingered gripper for a basic pick and place application.
- Perform simulation of the gripper in ROS platform
- Fabricate the gripper and perform a basic pick and place operation (attached to ABB IR 120) using ROS as the trajectory planner.

1.4 Outline of the Report

This project report is organised as follows:

Chapter 1 presents background of the work that describes the way through which the paper is modified into the main objective.

Chapter 2 deals with the literature review. A brief survey of the previous research works of the topic has been discussed in this section.

Chapter 3 deals with the methodology of the work and specifications of the robot under which it is built and finally a brief description about the whole system is mentioned.

Chapter 4 deals with the simulation part which includes the creation of URDF file, simulation robot in Gazebo and finally interfacing the actual hardware with ROS.

Chapter 5 deals with the Fabrication and Hardware Description in detail.

Chapter 6 discuss about the ROS Control of Hardware.

Chapter 7 deals with Simulation Results.

Chapter 8 presents the project conclusion.

Chapter 9 is the list of research papers that were referred for this project.

Chapter 2

Literature Review

Chapter 3

Mechanical Design and Modelling

3.1 Methodology

The methodology adopted for is discussed here. It is a simple and systematic approach. The entire work can be split in to two sections as described.

Simulation Phase

- Create CAD model for simulation satisfying the specifications.
- Create Moveit! package and define controllers.
- Perform Real time simulation in Gazebo using Moveit! Rviz motion planning.

Fabrication Phase

- Compute the maximum torque requirement at each joints and select joint motors accordingly.
- Create CAD model for actual fabrication.
- Fabricate the robot and demonstrate real time control on actual robot.

3.2 Robot Specifications

3.3 System Description

The development of the robot also includes two phases simulation phase and fabrication phase. Simulation phase includes creating CAD model for simulation satisfying the specifications, Moveit package and define controllers. Finally performing real time simulation in Gazebo using Moveit-Rviz motion planning. Fabrication phase includes computation of the maximum torque requirement at each joints and selection of joint motors accordingly. Creating CAD model for actual fabrication. After fabrication of the robot, demonstration of real time control on actual robot. The block diagram representation of the desired configuration is shown.

`img/systemdescrip.png`

Figure 3.1: Block diagram of RRP configured robot

Chapter 4

Simulation Using ROS

ROS, the Robot Operating System, is an open source framework for writing robot software and for getting robots to do things. ROS is meant to serve as a common software platform for people who are building and using robots. ROS follows the Unix philosophy of software development in several key aspects. ROS systems are comprised of a large number of independent programs that are constantly communicating with each other. This paradigm was chosen to encourage the reuse of robotics software outside the particular robot and environment that drove its creation.

Roscore is a service that provides connection information to nodes so that they can transmit messages to one another. The most common way to do that is through topics. A topic is a name for a stream of messages with a defined type. Every node connects to roscore at startup to register details of the message streams it publishes and the streams to which it wishes to subscribe. ROS graph node represents a software module that is sending or receiving messages.Every ROS system needs a running roscore ,since without it, nodes cannot find other nodes. Catkin is the ROS build system: the set of tools that ROS uses to generate executable programs, libraries, scripts, and interfaces that other code can use.Before you start writing any ROS code, you need to set up a workspace for this code to live in. A workspace is simply a set of directories in which a related set of ROS code lives. ROS software is organized into packages, each of which contains some combination of code, data, and documentation. The ROS ecosystem includes thousands of publicly available packages in open repositories, and many thousands more packages are certainly lurking behind organizational firewalls.

First step of simulation using ROS is the modelling of the SCARA robot from the URDF file generated by the solid modelling and computer aided design (CAD) software Solidworks.

4.1 Modelling The Robot URDF

In ROS robot models are represented in an XML format called Unified Robot Description Format (URDF). This format is designed to represent a wide variety of robots from a two-wheeled toy to a walking humanoid. URDF is similar to the Simulation Description Format (SDF), which is commonly used to build Gazebo environments around existing robots. URDF is only capable of representing robots whose kinematics can be described by a tree.
On startup, the URDF model of the robot was loaded into the parameter server, under the standard name robot description. The joint state publisher , in response to the slider state in the GUI, is publishing sensor msgs -Joint State messages on the joint states topic. Each message declares the position of each joint in the system. Another node, the robot state publisher, reads the URDF model from the parameter server and is subscribed to joint states . This node combines the 1D position of each joint with the kinematic model to calculate a tree of 6D (position and orientation) coordinate transforms that describe where in space the robot's links are with respect to each other (in other words, it performs forward kinematics).

The robot state publisher is commonly used with robots (both real and simulated) to handle the common task of forward kinematics, allowing the authors of robot drivers to publish just the individual joint state information and not the full coordinate transform tree. And as Rviz is used extensively in ROS development, especially for visualization of data related to coordinate transforms, the URDF display tool is really just a combination of commonly used ROS tools with a simple front end GUI that allows to supply fake joint position information.

URDF format folder is needed for visualizing in Rviz ,simulation in Gazebo and hardware ROS control. To export URDF from Solidworks we need to do the following steps.

Before exporting to URDF file, change to reference pose of the robot. The first joint is revolute from 0 radians to 6.28 radians. So the joint should be keept in 0 radian angle for best response when interfacing with hardware. Negative values can be used in Rviz and Gazebo Simulation. But when interfacing with Arduino for hardware implementation of negative values are not expected. The second joint is revolute joint, the limit angle minimum is 0 radian and maximum is 2.8 radian(160 degree). The third joint is prismatic joint, according to this case keeping the prismatic joint in retracted position is expected to give a radial traverse of 200mm (for the prismatic joint.

Step 1 : Click file. A drop down list will appear. Click on Export as URDF option.

Step 2 : A dialog box will appear probably on left side of the window.

Step 3 : Give name of base link. Give base_link as fixed joint. The origin reference axis will be automatically generated if we give appropriate option. Select the links we are considering as fix. In this case base links are fixed.

Step 4 : In the cell for entering number of child links, give 1(in this case). A child link will be created.

Step 5: Double click on the child link . Replace the name empty link by giving

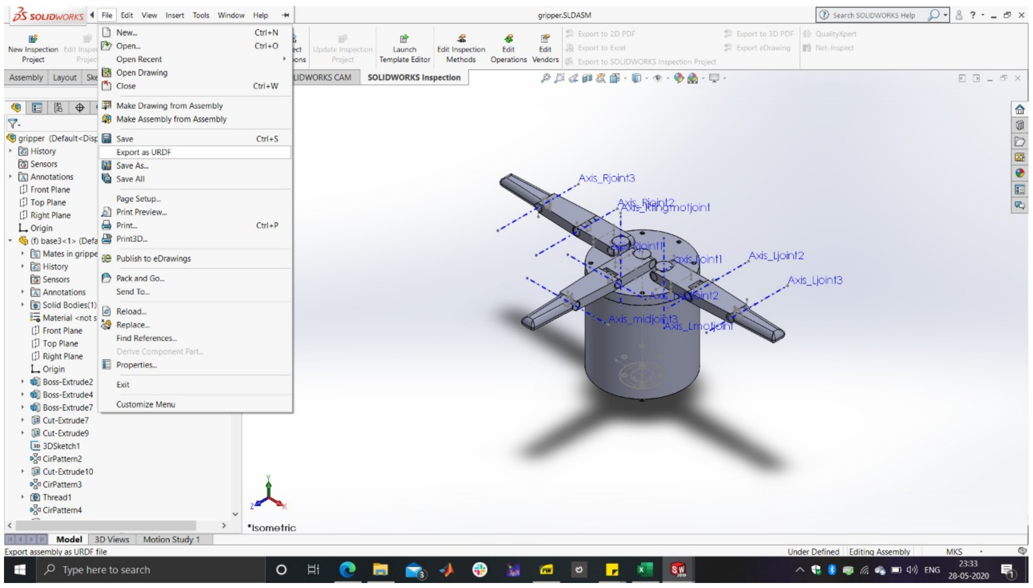


Figure 4.1: Creating URDF using URDF plugin in Solidworks

the name as link1(in this case). Give joint name as R1 (in the case. R1 denotes Revolute Joint 1). Select Joint type as revolute joint.

Step 6 : In the cell for entering number of child links, give 1(in this case). A child link will be created.

Step 7: Double click on the child link . Replace the name empty link by giving the name as link2(in this case). Give joint name as P2(in the case. P2 denotes Prismatic Joint 2). Select joint type as prismatic joint.

Step 8 : In the cell for entering number of child links, give 1(in this case). A child link will be created.

Step 9: Double click on the child link . Replace the name empty link by giving the name as link3(in this case). Give joint name as R3(in the case. R3 denotes Revolute Joint 3). Select joint type as reolute joint.

Step 10: A window will be opened. Click on first joint name R1. Change the joint angles minimum and maximum values to 0 and 6.24 radians respectively. Give effort value as 100 and velocity as 1.

Step 11: Double Click on second joint name P2. Change the joint distance minimum and maximum values to 0 and 0.17 metres respectively. Give effort value as 100 and velocity as 1.

Step 12: Double Click on third joint name R3. Change the joint angles minimum and maximum values to 0 and 3.14 radians respectively. Give effort value as 100 and velocity as 1.

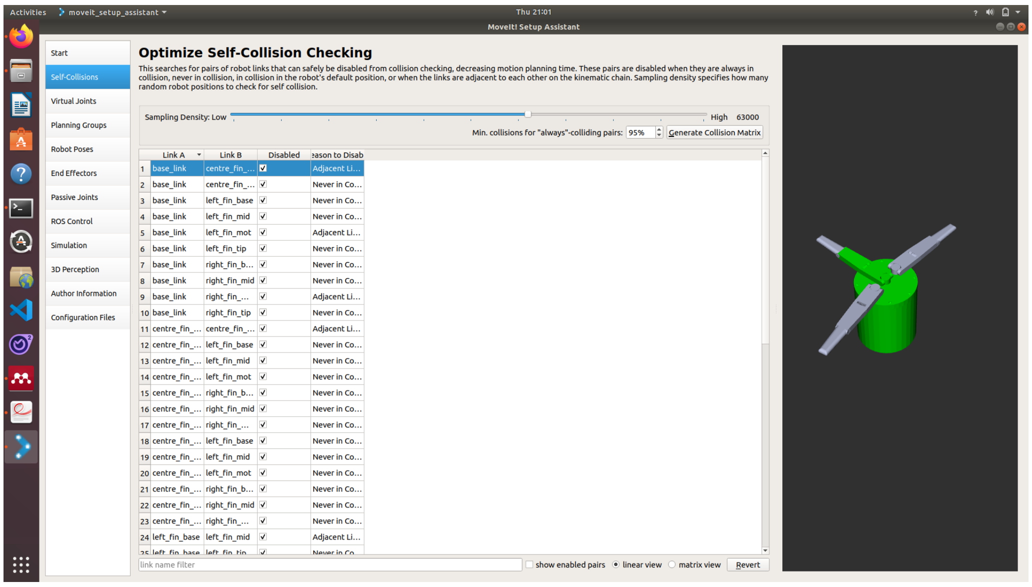
Step 13: Click on Finish.

The URDF folder will be created in the destination given. Copy the folder to some location in drives so that we can access from Windows. URDF is an XML format for representing a robot model.

4.2 Generating Move-It Package

4.2.1 Steps for generating moveit! package

- Intialising the move it package by **roslaunch moveit_setup_assistant setup_assistant.launch**.
- A new moveit package window is opened, click on **create New Moveit! Configuration Package** → Browse then from the desired location the package robot_1 is selected from which URDF is loaded on the moveit package.
- If the robot model is successfully loaded, robot model appears on window next.
- Next **Self-Collision matrix** needs to be generated. It checks for each link pair and categorizes the links as always in collision, never in collision, default in collision, adjacent links disabled, and sometimes in collision, and it disables the pair of links which makes any kind of collision.
- **Virtual Joint** is for mobile robots and hence is not needed here.



Optimize Self-Collision Checking

This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs 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. Sampling density specifies how many random robot positions to check for self collision.

Sampling Density: Low — High 63000

Min. collisions for "always"-colliding pairs: 95%

	Link A	Link B	Disabled	eason to Disab
1	base_link	centre_fin_...	✓	Adjacent Li...
2	base_link	centre_fin_...	✓	Never in Co...
3	base_link	left_fin_base	✓	Never in Co...
4	base_link	left_fin_mid	✓	Never in Co...
5	base_link	left_fin_mot	✓	Adjacent Li...
6	base_link	left_fin_tip	✓	Never in Co...
7	base_link	right_fin_b...	✓	Never in Co...
8	base_link	right_fin_mid	✓	Never in Co...
9	base_link	right_fin_...	✓	Adjacent Li...
10	base_link	right_fin_tip	✓	Never in Co...
11	centre_fin_...	centre_fin_...	✓	Adjacent Li...
12	centre_fin_...	left_fin_base	✓	Never in Co...
13	centre_fin_...	left_fin_mid	✓	Never in Co...
14	centre_fin_...	left_fin_mot	✓	Never in Co...
15	centre_fin_...	right_fin_b...	✓	Never in Co...
16	centre_fin_...	right_fin_mid	✓	Never in Co...
17	centre_fin_...	right_fin_...	✓	Never in Co...
18	centre_fin_...	left_fin_base	✓	Never in Co...
19	centre_fin_...	left_fin_mid	✓	Never in Co...
20	centre_fin_...	left_fin_mot	✓	Never in Co...
21	centre_fin_...	right_fin_b...	✓	Never in Co...
22	centre_fin_...	right_fin_mid	✓	Never in Co...
23	centre_fin_...	right_fin_...	✓	Never in Co...
24	left_fin_base	left_fin_mid	✓	Adjacent Li...
25	left_fin_base	left_fin_tip	✓	Never in Co

☐ show enabled pairs [✓ linear view] matrix view

Figure 4.2: Moveit Setup Assistant startup

- **Planning group** is a group of joints/links in a robotic arm which plans together in order to achieve a goal position of a link or the end effector.Two planning groups are to be created, one for the arm and one for the gripper,since end-effector is not used, only one planning group is created.

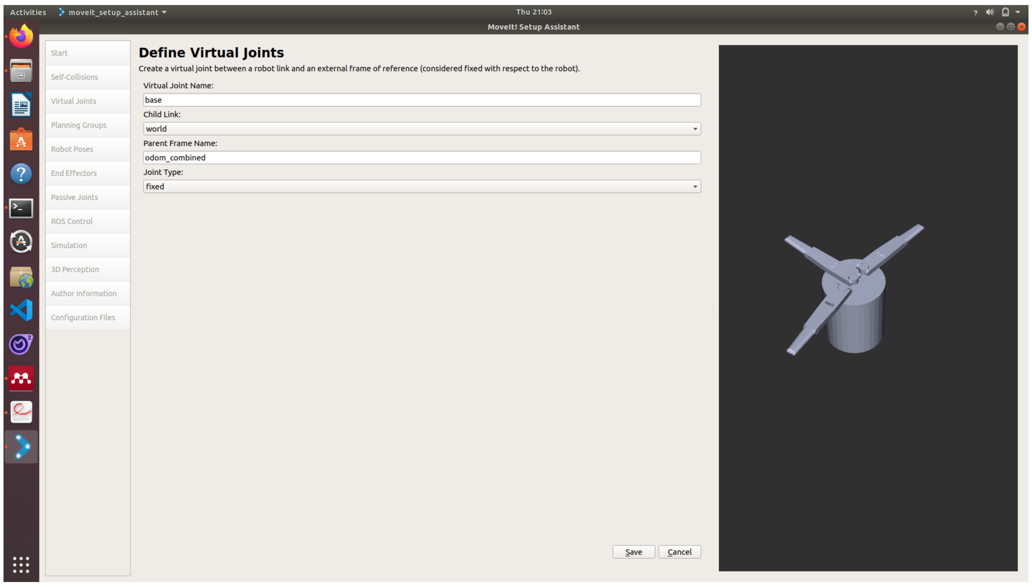
- Select **Planning Group**, planning group window appears, in this window **Group Name** is given. Next is the selection of the **Kinematic Solver**. From the drop down box select **kdl_kinematics_plugin/KDL Kinematics Plugin**.

- Then inside the **gripper group** ,**Kinematic Chain** is added, there **centre_fin_mid_joint** to **right_fin_tip_joint** have to be added.

- For adding different poses for robot, click on the **robot poses**

- Next is adding **robot pose**, where fixed poses like the zero position and home position in the robot configuration are added.

- The last step is to generate configuration files. Click on the **Browse** button to locate a folder where configuration file that is generated by the Setup Assistant tool has to be saved .



Define Virtual Joints

Create a virtual joint between a robot link and an external frame of reference (considered fixed with respect to the robot).

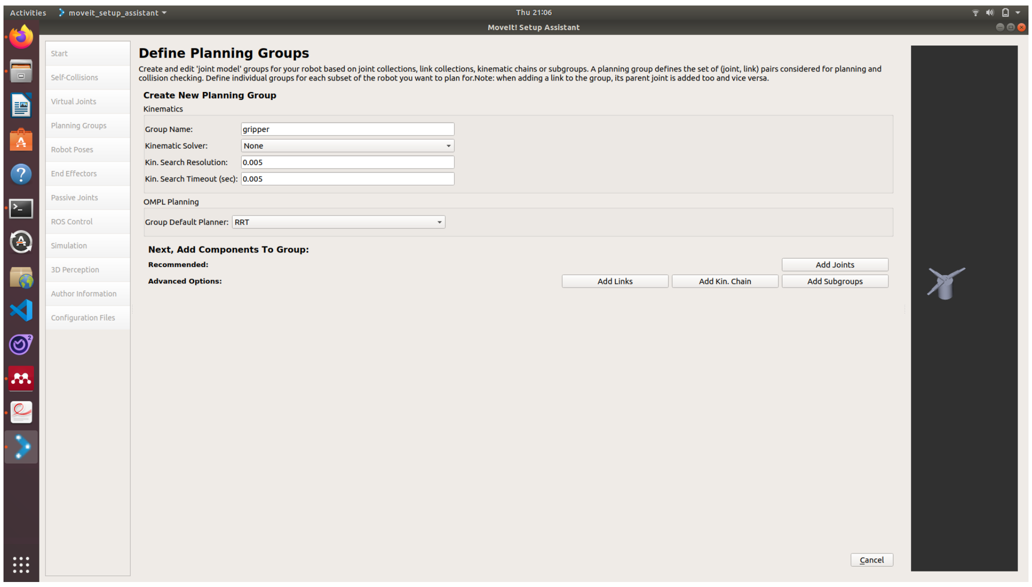
Virtual Joint Name: base

Child Link: world

Parent Frame Name: odom_combined

Joint Type: fixed

Figure 4.3: Defining Virtual Joints



Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.Note: when adding a link to the group, its parent joint is added too and vice versa.

Create New Planning Group

Kinematics

Group Name: gripper

Kinematic Solver: None

Kin. Search Resolution: 0.005

Kin. Search Timeout (sec): 0.005

OMPL Planning

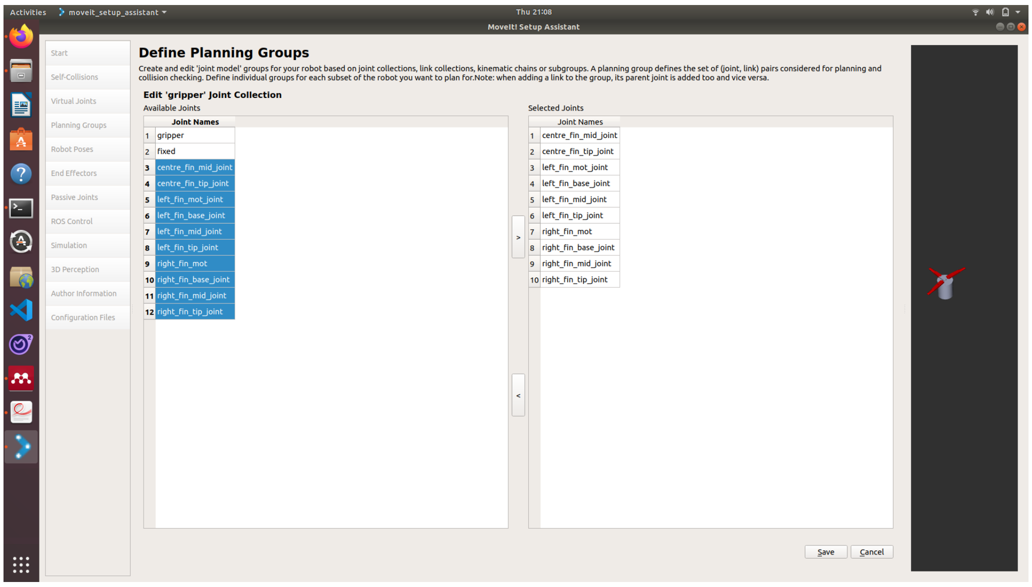
Group Default Planner: RRT

Next, Add Components To Group:

Recommended:

Advanced Options:

Figure 4.4: Planning groups in MoveIt



Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.Note: when adding a link to the group, its parent joint is added too and vice versa.

Edit 'gripper' Joint Collection

Available Joints

	Joint Names
1	gripper
2	fixed
3	centre_fin_mid_joint
4	centre_fin_tip_joint
5	left_fin_mot_joint
6	left_fin_base_joint
7	left_fin_mid_joint
8	left_fin_tip_joint
9	right_fin_mot
10	right_fin_base_joint
11	right_fin_mid_joint
12	right_fin_tip_joint

Selected Joints

	Joint Names
1	centre_fin_mid_joint
2	centre_fin_tip_joint
3	left_fin_mot_joint
4	left_fin_base_joint
5	left_fin_mid_joint
6	left_fin_tip_joint
7	right_fin_mot
8	right_fin_base_joint
9	right_fin_mid_joint
10	right_fin_tip_joint

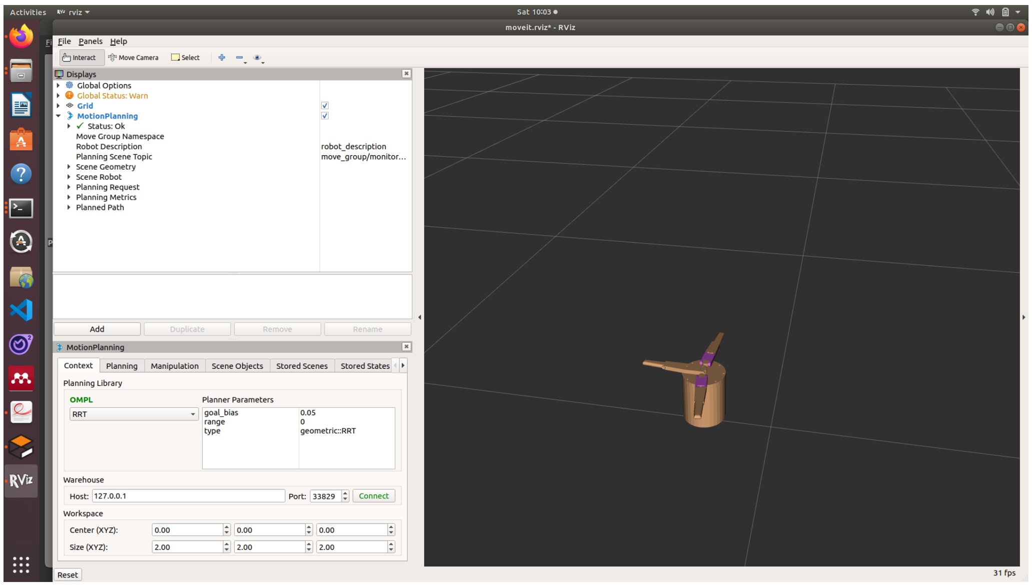
Figure 4.5: Defining Planning Groups

- After browsing the required folder, click on the **Generate Package** button, this will save the files into the chosen folder.
- Click on **Exit Setup Assistant**, which will exit us from the tool.

4.3 Motion planning of gripper in RViz

Gazebo is a multi-robot simulator for outdoor environments. Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid body physics). By realistically simulating robots and environments code designed to operate a physical robot can be executed on an artificial version. Numerous researchers have also used Gazebo to develop and run experiments solely in a simulated environment. Controlled experimental setups can easily be created in which subjects can interact with manipulators in a realistic manner. There is a big difference between RViz visualisation and Gazebo simulation, the former one is used to display relative position of links, but the latter one can be regarded as an experimental copy of real robot in virtual world.

- **RViz** allows to create new planning scenes where robot works, generate motion plans, add new objects, visualize the planning output and can directly interact with the visualized robot. MoveIt! package includes configuration files and launch files to start motion planning in RViz.
- Next step is visualization of the robot for that demo launch file is launched the command is **roslaunch gripper_ demo.launch**.
- **RViz** window is opened which cosists of several tabs, on **context** tab from **OMPL** drop down list select **RRT**.
- Next is **Planning** tab where, the start state, goal state plan a path, and execute the path of the robot can be assigned.



Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitor...

Planning Library

OMPL: RRT

Planner Parameters

goal_bias	0.05
range	0
type	geometric::RRT

Warehouse

Host: 127.0.0.1 Port: 33829

Workspace

Center (XYZ):	0.00	0.00	0.00
Size (XYZ):	2.00	2.00	2.00

Figure 4.6: Motion planning of robot in RViz

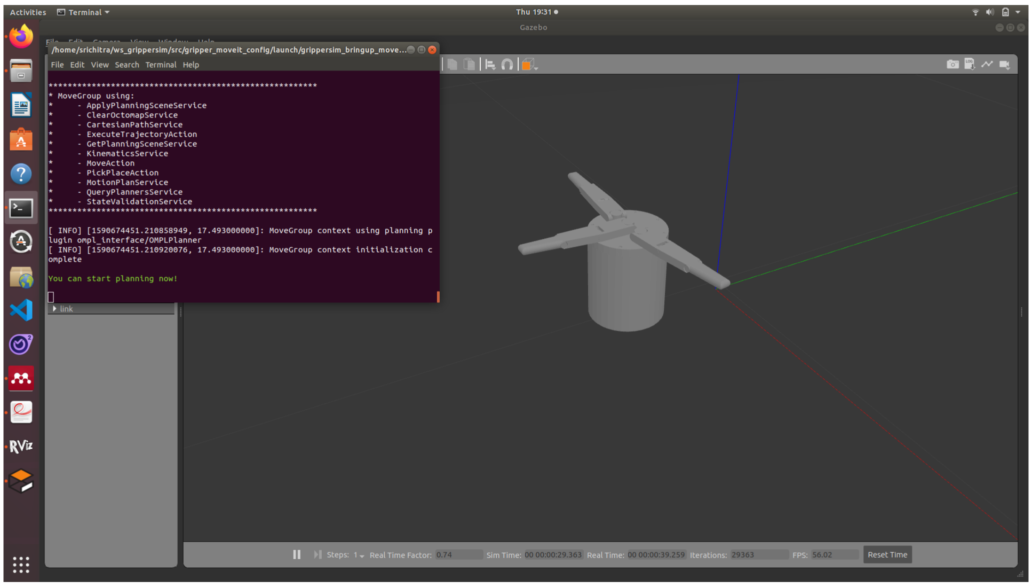
Plan button, is used to plan the path from the start to the goal state, and if the planning is successful, path can be executed by using **Plan and Execute** button.

4.4 Gripper simulation using Gazebo

- Gazebo is a multirobot simulator for complex indoor and outdoor robotic simulation. We can simulate complex robots, robot sensors, and a variety of 3D objects.
- The simulation model for the gripper by updating the existing robot description can be created by adding simulation parameters.
- To launch the existing simulation model, add **world.launch** and **bringup.launch** files needs to be added to moveit! package.
- Now open a new terminal to launch Gazebo using the code **roslaunch gripper_moveit_config gazebo.launch**.
- **RViz** is used to control the robot on Gazebo, which creates a robotic environment similar to the real world.
- After Gazebo, use the move_group command and then launch RViz using the moveit_rviz.launch command. The motion path planned on the RViz can now be achieved on Gazebo.
- When the actual robot is fabricated, Gazebo is replaced by the robot and the robot can be controlled by using Rviz.

After completing all the above steps the robot can achieve the desired poses and rest of the interfacing using arduino can be seen in the next chapter.

4.4.1 Simulation in Gazebo

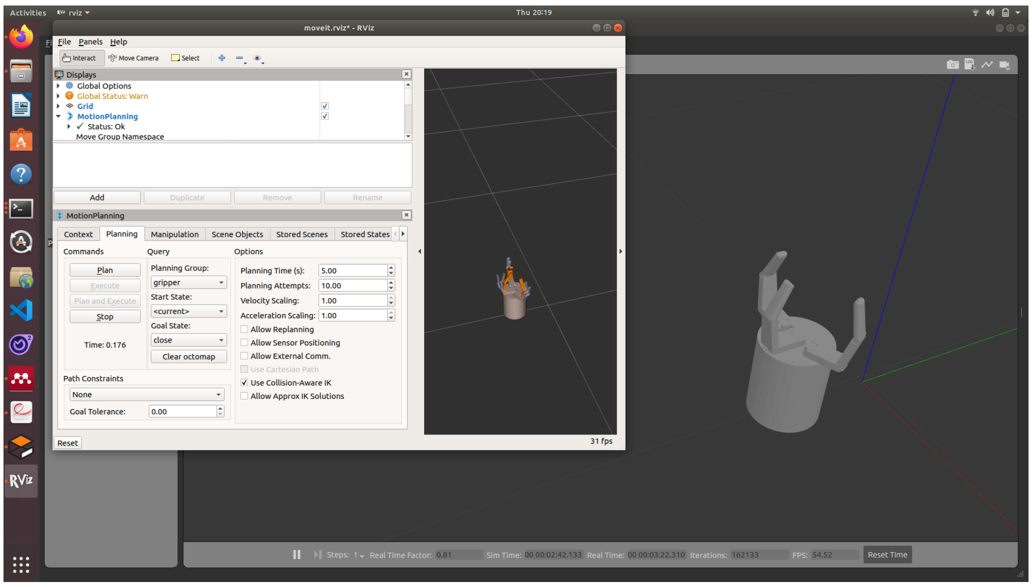


```
*******************************************************
* MoveGroup using:
*     - ApplyPlanningSceneService
*     - ClearOctomapService
*     - CartesianPathService
*     - ExecuteTrajectoryAction
*     - GetPlanningSceneService
*     - KinematicsService
*     - MoveAction
*     - PickPlaceAction
*     - MotionPlanService
*     - QueryPlannersService
*     - StateValidationService
*******************************************************

[ INFO] [1590674451.210858949, 17.493000000]: MoveGroup context using planning p
lugin ompl_interface/OMPLPlanner
[ INFO] [1590674451.210920076, 17.493000000]: MoveGroup context initialization c
omplete

You can start planning now!
```

Figure 4.7: Launching the model in Gazebo



Planning Group: gripper
Start State: <current>
Goal State: close
Time: 0.176
Path Constraints: None
Goal Tolerance: 0.00
Planning Time (s): 5.00
Planning Attempts: 10.00
Velocity Scaling: 1.00
Acceleration Scaling: 1.00
[ ] Allow Replanning
[ ] Allow Sensor Positioning
[ ] Allow External Comm.
[ ] Use Cartesian Path
[✓] Use Collision-Aware IK
[ ] Allow Approx IK Solutions

Figure 4.8: Corresponding execution in Gazebo

Chapter 5

Simulation Results

Chapter 6

Conclusions

Chapter 7

References