

Multi-Robot Collaboration for Co-Assembly Tasks

Master’s Project submitted to

*The Department of Computer Science at*

*Montclair State University*

In partial fulfillment of the requirements for a degree in Master of Science in Computer Science

By: Rishikumar Patel

Faculty Advisor: Prof. Weitian Wang

May 11, 2022

Table of Contents

[**ABSTRACT**](#_heading=h.xftzfp3wkxjo) **3**

[**INTRODUCTION**](#_heading=h.rrinprmxlhdr) **4**

[**MULTI-ROBOT SIMULATION DEVELOPMENT**](#_heading=h.5jud8s4c5kec) **8**

[Ubuntu](#_heading=h.8vubb2swxme1) 8

[ROS – Robot Operating System](#_heading=h.eor5tcsmhco0) 8

[MoveIt](#_heading=h.9f8obxbr6x1q) 10

[Rviz](#_heading=h.159fpyjqxmf9) 12

[How to Deploy Multi-Robot Simulation?](#_heading=h.mbyy9bgi8my) 13

[URDF](#_heading=h.skmjs7gtyye8) 13

[MoveIt Setup Assistant](#_heading=h.hmb1tujqqavz) 14

[Launch Rviz](#_heading=h.htdynsike1zn) 19

[**COST COLLABORATION STRATEGY**](#_heading=h.th0p5fx6z1m) **20**

[Sorting Algorithms](#_heading=h.aw8jm558zp1i) 20

[What is Big Omega(Ω), Big Theta(Θ) & Big Oh(O) Notation?](#_heading=h.47bet8fblb2l) 21

[Merge-Sort](#_heading=h.sx4qy1g8d0b7) 22

[Implementation of the Merge-Sort algorithm](#_heading=h.cqqjavk7xoyv) 23

[**RESULTS & ANALYSIS**](#_heading=h.bhlpapkowk0f) **28**

[Experimental Setup](#_heading=h.hqytm95v1e55) 28

[Moving Robot-arm using poses](#_heading=h.iql7h7ur6o2x) 29

[Moving Robot-arm using cartesian motion](#_heading=h.s75krcgtzmnq) 30

[Task Scheduling using the “Merge-Sort”](#_heading=h.pobg76k0putl) 31

[**CONCLUSION**](#_heading=h.aa4xiim9m34z) **36**

[**Additional Information & Future Improvements**](#_heading=h.qayaamjl9uoa) **36**

[**Acknowledgment**](#_heading=h.m11mxz6dm3hg) **36**

# 

# ABSTRACT

Advances in the robotics sector over the last decade have led many businesses to transition from humans to robots to do repetitive jobs such as assembly tasks, owing to their larger load-bearing capability when compared to humans and lower maintenance costs. When compared to one single robot laboring to perform the task at hand, adopting a multi-robot scenario with a shared workspace can significantly boost throughput and efficiency. The nature of some assembly tasks be linear, where sub-tasks are required to get executed one after another, and they can be sorted by assigning priority to them to complete them in the desired sequence and provide the intended output. To demonstrate this, we have created a visualization of three robot arms with six degrees of freedom (DOF) each that share a common workspace. If all assembly activities in the shared workspace have been prioritized, the proposed platform allows a human to program and manage multiple robot arms to execute any assembly chores in the shared workspace. We go over the specifics of the software system's development and implementation. Additional experiments include the completion of an assembly task by the implementation of a generic method to schedule the given prioritized sub-tasks and programming robot arms to move using Python script. Experiments have shown that the established platform can effectively aid a person in coding/programming numerous robot arms for assembly tasks. This is encouraging.

# INTRODUCTION

Different types of robots are being utilized for several tasks in a range of industries these days, including industrial work [1], assembly work [2], cleaning, space exploration, medical procedures [3], and manufacturing [4], to mention a few. All these industries demand different functionality performed by their robots, each with its own set of capabilities. Although autonomous robots are desired to provide the best output possible without human intervention, they are not applicable in all workspaces due to various factors, such as humans and robots working in a shared environment [2, 5, 6], each object being worked on by a specific robot having a different layout which may require a human touch [3] or the site where the robot is working requires a human operator to operate the robot/s [1, 2, 5], and so on.

The 6DOF (six-degree-of-freedom) arm robots are the primary focus of this research. The term "mechanical" arm robot or "industrial" robot can also be used to describe arm robots. Robot arms have several joints that serve as axes, allowing for a certain amount of flexibility. Robot arms typically have numerous degrees of freedom (DOF), which allows them to perform activities such as pick and place/deliver. A robotic arm's ability to move freely increases in direct proportion to the number of rotational joints it possesses. A body's freedom of movement in a three-dimensional space is represented by the use of three degrees of freedom for movement along the positional axes of X (surge), Y (sway), and Z (heave), as well as three degrees of freedom for orientation, which are represented by the degrees of freedom Roll, Pitch, and Yaw, where Roll reflects a spinning around the X-axis, Pitch reflects a spinning around the Y-axis, and Yaw reflects spinning around Z-axis.  A robotic arm's freedom of movement is increased by the number of rotational joints it possesses, as suggested previously. However, this increases the complexity of the movement and makes controlling it more challenging [7].

Multiple-DOF robot arms are controlled using Inverse Kinematics (IK), which is a technique commonly used. Inverse kinematics is the application of kinematic equations to determine the joint parameters that will result in the desired configuration (position and orientation) for the end-effector of each robot's arm. In robotics, motion planning refers to the process of determining the movement of a robot arm so that its end-effector moves from an initial configuration to the desired configuration. There are numerous motion planners available for use in motion planning that can be customized to meet the needs of the user. Advanced users always have the option of building their motion planner.

There have been many studies conducted in the past that are focused on either multiple robots working in collaboration or multiple robots with human collaboration to increase throughput. A study showed an implementation of a vision-guided feedback system (using cameras) used to create personalized stent grafts produced according to the human anatomical structure using bimanual robots and limitations were noted to be cameras, higher definition cameras would provide better information for robots to process and plan their motions accordingly [3]. Another study proposed to improve the throughput of 3D printing technology by adopting collaborative 3D printing for manufacturing over the traditional approach and the limitations of this approach showed that better algorithms for scheduling, task segmentation, and sequencing can increase efficiency [4]. One of the studies done on collaborative assembly systems (CAS) evaluated its performance by assigning challenging assembly tasks in a multi-resource setting and limitations were again found to be task scheduling [8]. One of the conducted studies suggested that the robots can perform laborious and monotonous/repetitive tasks more efficiently than humans and it also suggested that more than one robot in the shared workspace can be used for load distribution among the robots for the tasks that required heavy lifting [9]. One of the studies was focused on working with different types of robots in collaboration to complete the desired task and the limitation of the study was later found to be the task infeasibility due to the decentralized approach to allocating tasks [10]. One study focused on Human-multi-robot collaboration to generate a real-time generic algorithm to adjust assembly scheduling of multi-robot according to human capabilities [2]. One study was focused on human attention modeling in multi-robot interactions to maximize multi-robot system performance [5]. One of the studies showed a network-based policy for multi-robot coordination that allowed efficient task scheduling for multiple robots in varying team sizes [11]. So we saw that there have been many studies that focused to develop an effective task scheduling in multi-robot collaboration in a shared environment or the study’s limitations were task scheduling instead.

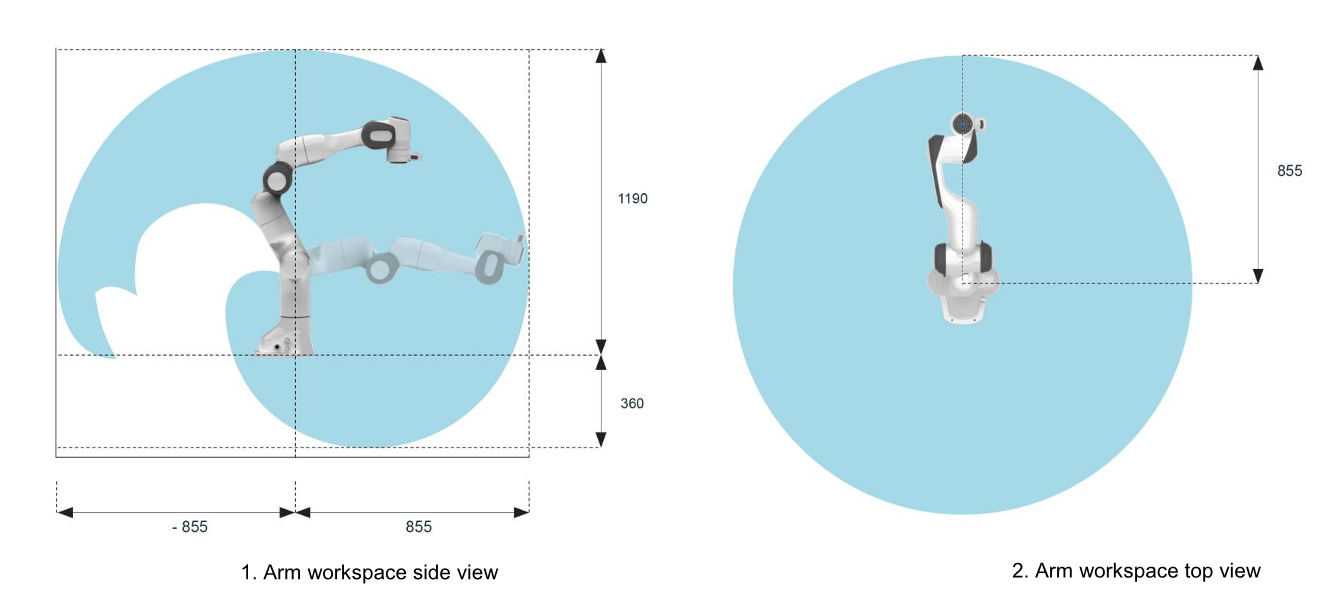


Figure 1 - A Panda Arm’s Workspace [12].

We can admit that when a complex assembly task needs to be completed such as sewing, stacking, or manufacturing, it is preferable to have more than one arm robot working in a shared workspace. The proposed model is centered on multiple robot arms collaborating in a shared workspace without the use of any additional equipment. Latter we have implemented a generic sorting algorithm in the project to effectively schedule the task in a certain order, given that the task has been prioritized. The model is a simulation of three Panda arm robots by Franka Emika (Fig. 1). Figure 1 also shows the workspace of a Panda arm, in the side-view as well as in the top-view, based on its position. As we can see the arm’s front is entirely reachable from the robot so, to best fit our project’s needs for the shared workspace we have made the three Panda arm robots face the center of the table as you can see in Fig. 2. Of course, it can be used with real Panda arm robots with little configuration changes. This eliminates the need to consider factors such as human safety in a shared workspace since there are no humans working in that workspace and it is also more cost-effective. The setup is depicted in Fig. 2.

Graphical user interface

Description automatically generated

Figure 2 – Three Panda Robot Arms Setup

In this paper, we describe the software and/or middleware that was used in the design and development of the multi-robot simulation depicted in Fig. 2 as well as the implementation of the generic sorting algorithm used for task scheduling. Aside from that, we experimented with implementing two (2) different methods, poses and cartesian movements, to manipulate the robot arms to identify which method is more appropriate for our setup and which method is less time-consuming overall. Later, we finalize a method of manipulation and experiment to determine whether the use of a generic sorting algorithm can result in effective task scheduling when the priorities of the sub-tasks are known. These three experimental methods have been observed and the results have been reported. Section 2 goes into great detail about the development of the multi-robot simulation system. Section 3 discusses the strategy for cost collaboration. Section 4 of this report contains the results and analysis. Section 5 concludes this project and brings it to a close.

This master's project's contributions can be summarized as follows:

1. A cheap and easy-to-use solution is proposed for task scheduling while working with multiple robot arms.
2. An easy-to-use method to develop applications for the Multi-robot arms working in the shared workspace.

# MULTI-ROBOT SIMULATION DEVELOPMENT

In developing our multi-robot simulation setup we used four (4) different components as listed below and after that, we’ll see how to deploy the entire setup:

1. Ubuntu - 20.04.
2. Robot Operating System - ROS
3. MoveIt
4. Rviz
5. How to deploy Multi-Robot Simulation?

Let us discuss the topics listed above in greater detail:

## Ubuntu

Linux distribution Ubuntu (based on the Debian operating system) is primarily comprised of free and open-source software and is based on the Debian kernel. Because Linux is a lightweight operating system, it is simple to use for robotics and IoT (internet of things) devices, among other things. The Ubuntu operating system, which is known for its flexibility and user-friendliness, has been the primary platform for ROS since the beginning. For the time being, Ubuntu is available in three editions: Desktop, Server, and Core. For our robotics project, we used the Desktop edition of Ubuntu (20.04). Developed by the British company Canonical and a community of other developers, Ubuntu is a free operating system. Following that, we'll learn about ROS.

## ROS – Robot Operating System

The first application that was installed on the Ubuntu 20.04 was Robot Operating System (ROS). Robot Operating System - ROS is not an operating system as the name suggests, nor it is a framework, but it is an open-source robotics middleware suite. It provides a structured communications layer between the operating system and the software applications running on it [13]. Ubuntu is most preferred when working with ROS due to its openness. ROS is partitioned into over 2,000 packages, each of which offers specific functionality. The framework's greatest significance is possibly the number of technologies it connects to. ROS has features such as hardware abstraction, device drivers, process communication across multiple machines, testing and visualization tools, and much more.

**Roscore**: Roscore is the core of the system providing a collection of nodes and programs. It contains three main functional module suits as shown in Fig. 3(a). ROS Master provides naming and registration to nodes in the ROS system [14].

**ROS Node**: A ROS node is a process that performs computation. It is an executable program running inside your application. We can write many nodes and put them into packages. Nodes are combined into a graph and communicate with each other using ROS topics, services, actions, etc. as shown in Fig. 3 (b, c, and d) respectively [14].

**ROS Topic**: ROS handles all the communications through topics, Topics are named buses over which nodes exchange messages [13].

**ROS Services**: One-way request & response communications can be done with the use of ROS services. Which requires a couple of messages, one for the request and another for the response [15].

**ROS Action Server**: ROS actions have a client-to-server communication relationship with a specific protocol. For the action server to communicate with the action client five (5) messages are generated when the action server is launched. Those messages are Goal, Cancel, Status, Feedback, and Result.

Software in the ROS Ecosystem can be separated into three groups [16]:

* language and platform-independent tools used for building and distributing ROS-based software.
* ROS client library implementations such as roscpp, rospy, and roslisp .
* Packages containing application-related code which uses one or more ROS client libraries.

Diagram

Description automatically generated

Figure 3 – ROS concepts. (a) Core computation, (b) Using topics to enable communication between different nodes using message publisher and subscriber, (c) Service request and response communication between server and client, (d) action communication between server and client [14].

## MoveIt

MoveIt is open-source software, it is widely used as a robotic manipulation platform that allows users to develop complex manipulation applications using ROS. MoveIt is the project that started at Willow Garage and is now led by PickNik Robotics. The high-level system architecture for the primary node provided by MoveIt, which is designated as "move\_group", is depicted in Fig. 4. This node performs the function of an integrator, bringing all of the individual components together to provide a set of ROS actions and services for users to take advantage of.

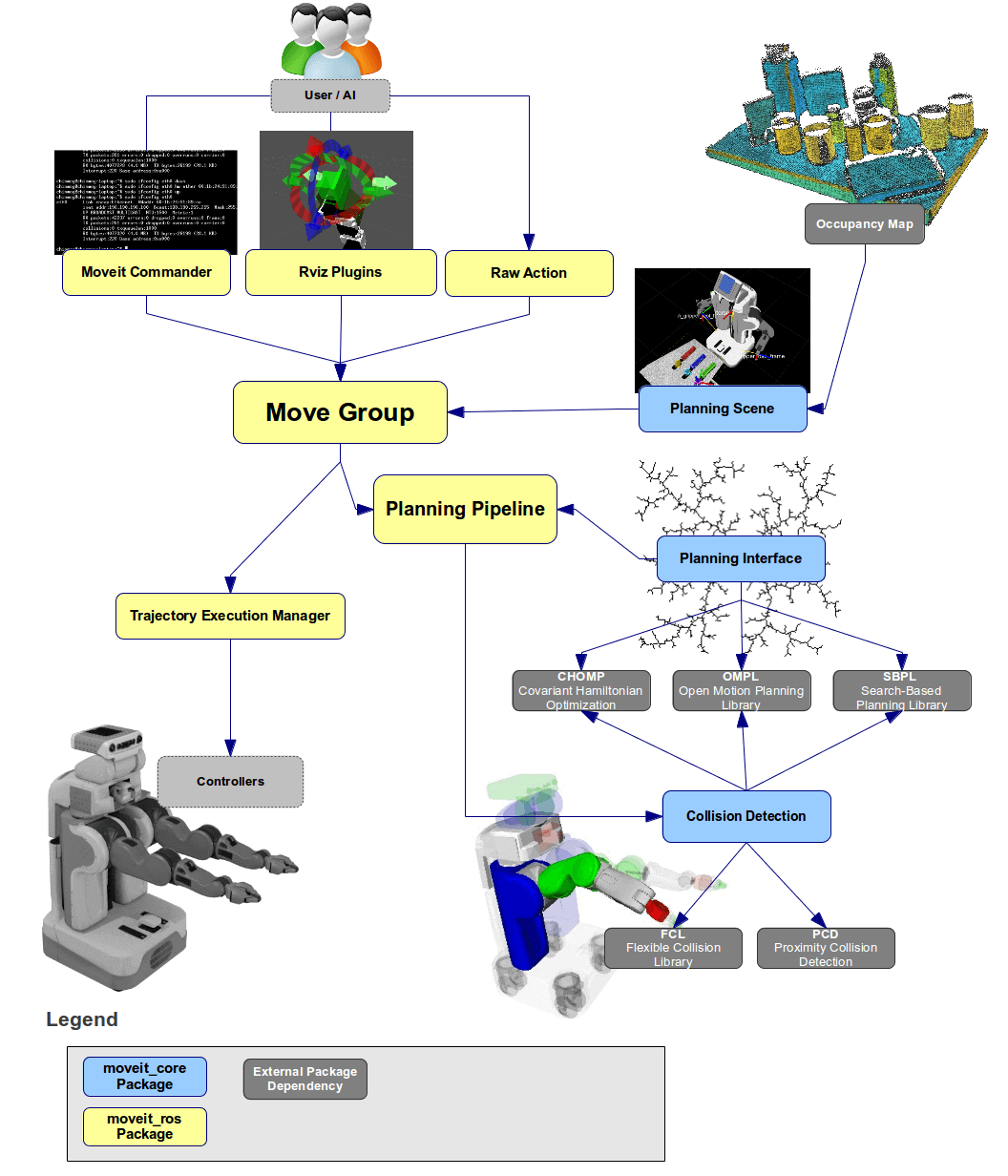


Figure 4 – High-level system architecture of the *“move\_group”* node [17].

MoveIt is a primary source of the functionality for manipulation in ROS. There are three (3) ways in which users can gain access to the actions and services provided by the *‘move\_group’* node as shown in Fig. 5:

* **In C++** - Using the *“move\_group\_interface”* package that provides an easy to setup C++ interface to *‘move\_group’* [17].
* **In Python** – Using the *“moveit\_commander”* package [17].
* **Through GUI** – Using the Motion planning plugin to Rviz [17].

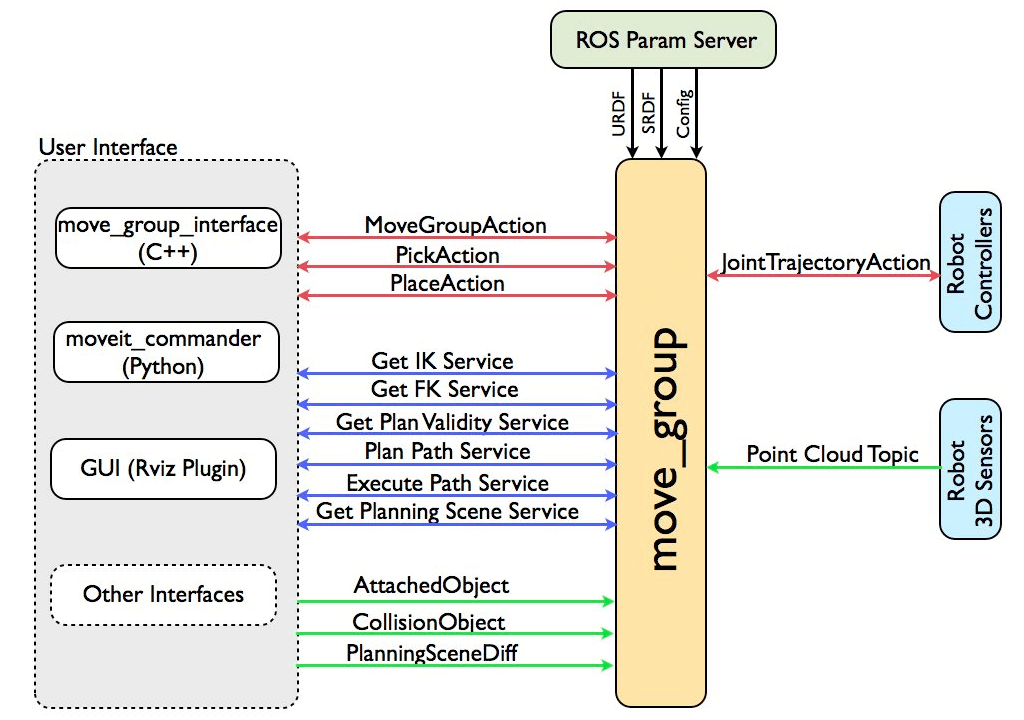


Figure 5 – User interface supported by the *“move\_group”* node [17].

MoveIt is built on top of the ROS messaging and build systems, and it makes use of some of the most popular ROS tools, such as the ROS Visualizer (Rviz) and the ROS robot format (URDF) [18]. MoveIt is a common entry point into ROS, especially through the use of the “MoveIt Setup Assistant” for configuring new robots. For this project, we have manipulated a URDF of the Panda arm-robot developed by Franka Emika to configure the setup and used Python to access the *‘move\_group’* node provided by MoveIt. MoveIt offers a variety of features for working with real robot visualizations and developing applications that can be run on real robots without the need for a physical robot.

## Rviz

Rviz is a graphical interface for ROS that allows you to visualize a large amount of data using plugins for a variety of topics. It makes it easier to view and understand what the robot is seeing and doing in real-time. It also provides MoveIt’s Motion planning Plugin to plan and execute motions on the simulated robots. A general robotics simulation architecture can be roughly classified into four key functional modules as shown in Fig. 6. These are Modeling, Planning, Visualization, and Control [14].

Diagram

Description automatically generated

Figure 6 - General robotic simulation architecture and module relationship [14]

## How to Deploy Multi-Robot Simulation?

To launch multi-robot simulation, we’ll need the ROS package generated by MoveIt Setup Assistant, in our case the package is named **“triple\_panda\_moveit\_config”**, and the package containing URDFs, which is usually the *“franka\_description”* package although in our case it is **“3\_panda\_arms\_moveit\_config”** package since we had to edit the URDF file to fit our project needs.

### URDF

Since we already saw all the components and understood what each of them is used for, we can begin to discuss how to set up and run the simulated environment. Before we jump into “Quick Start” for MoveIt [19], we’ll have to create URDF which describes the three-dimensional geometric representation of a robot, its kinematics as well as other properties relevant to robotics may be included such as the geometric visualization meshes, courser-grained collision geometry of the robot used for fast collision checking, joint limits, sensors, and dynamic properties such as mass, moments of inertia, and velocity limits [18]. Creating an accurate URDF from a scratch is a difficult task. URDF models for many robots already exist, that can be used directly or manipulated to fit the project’s needs. We had to manipulate the URDF to fit our project’s needs and it can be found in the *“User’s Manual”* of the project.

### MoveIt Setup Assistant [19]

The MoveIt Setup Assistant is the major feature that provides out-of-the-box guidance for beginners. The MoveIt Setup Assistant is a graphical user interface (GUI) that guides new users through the initial configuration requirements of using any robot with the motion planning framework by employing its URDF [20]. It automatically generates many configuration files necessary for the initial operation of MoveIt. Such as generating a self-collision matrix (Fig. 7(b)), virtual joints list (Fig. 7(c)), setting up planning groups (Fig. 7(d)), setting up robot poses (Fig. 7(e)), configuring end effectors (Fig. 7(f)), generating URDF for Gazebo simulation, passive joints list, setting up controllers (Fig. 7(g)), and generating entire ROS package according to the configurations (Fig. 7(h)). A *“demo.launch”* script is also generated, during ROS package generation, to launch a visualization tool (Rviz) with the new robot loaded and ready to code/run motion planning in a non-physics-based simulation. Fig. 7(a) shows the first page of the MoveIt Setup Assistant, and the large navigation pane on the left can be used by the user to move back and forth through the setup process as needed.

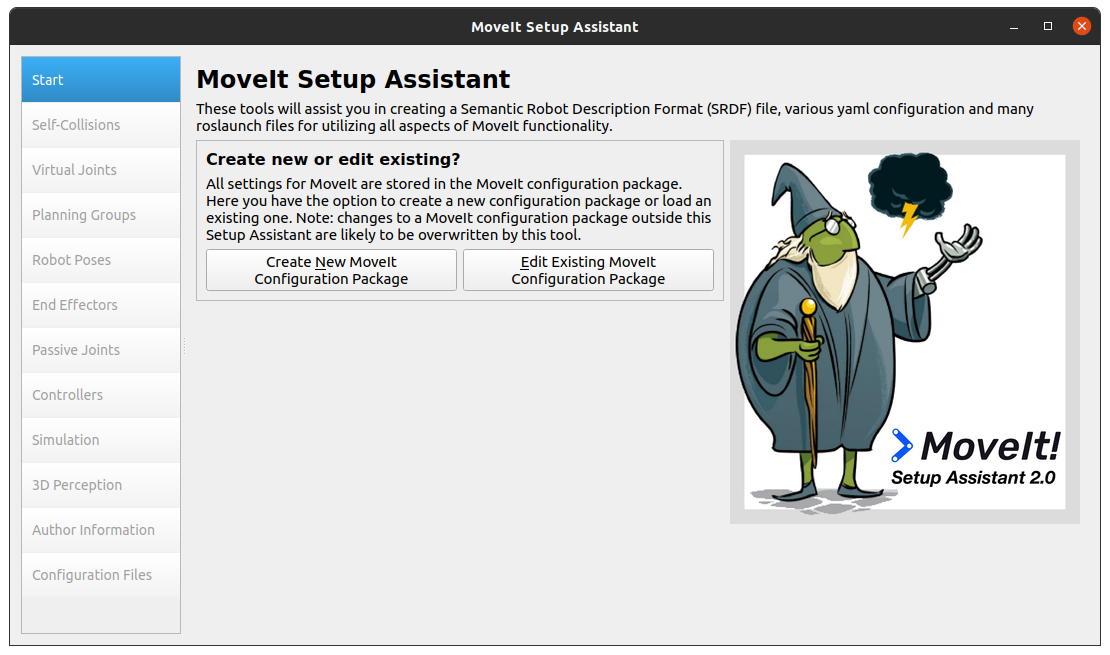


Figure 7(a) – MoveIt Setup Assistant - Starting Pane

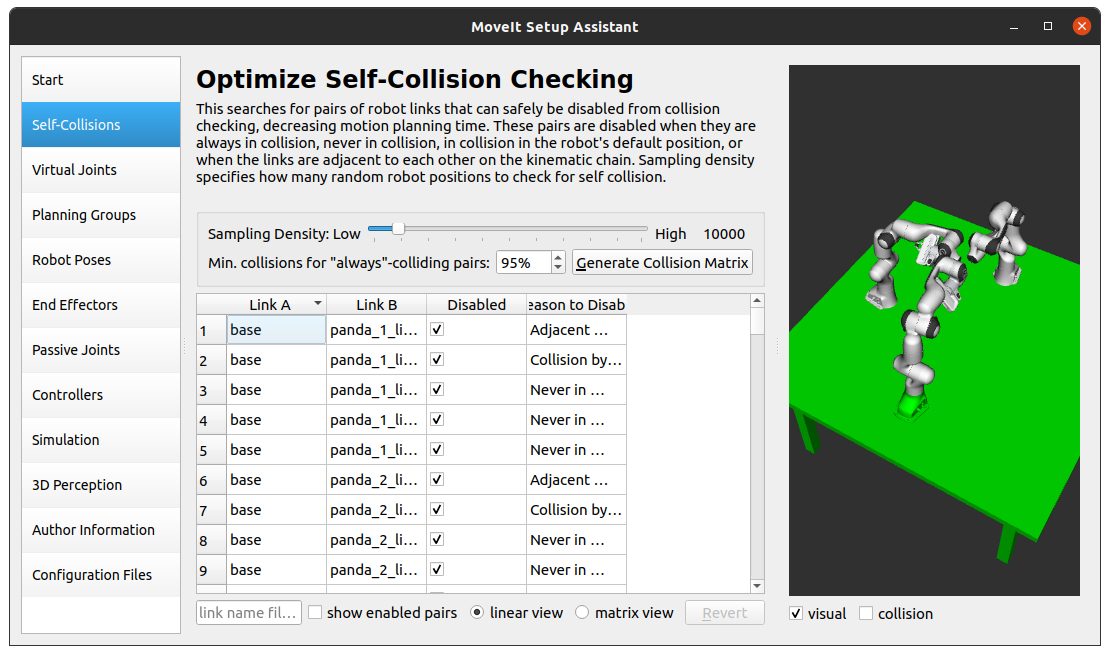


Figure 7(b) – Self-Collision Pane

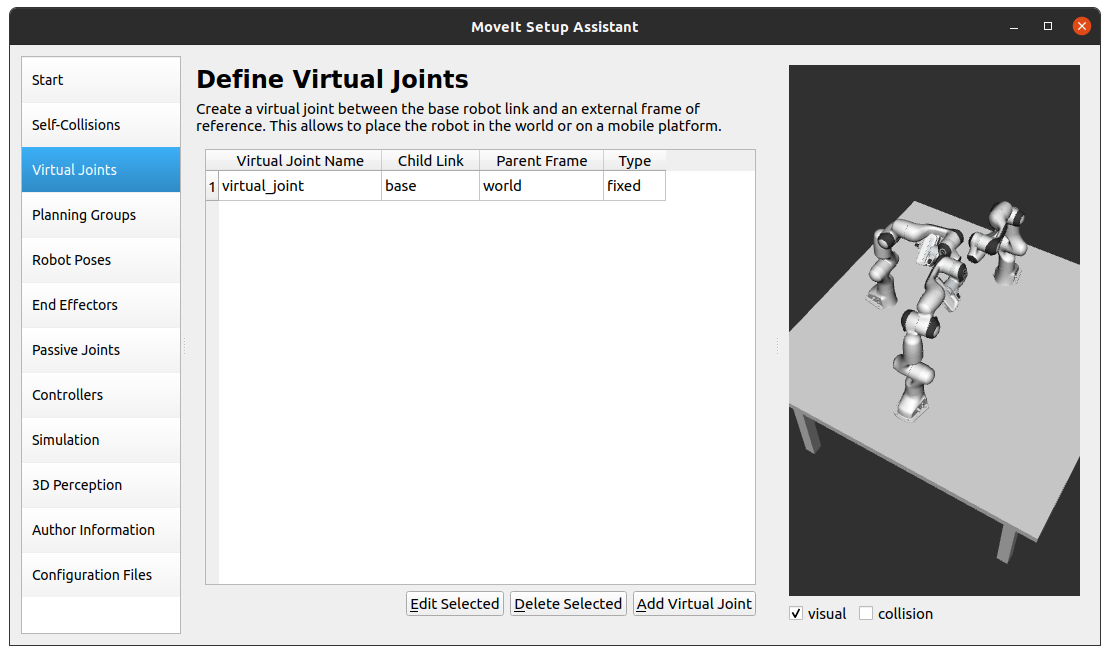


Figure 7(c) – Virtual Joint Pane

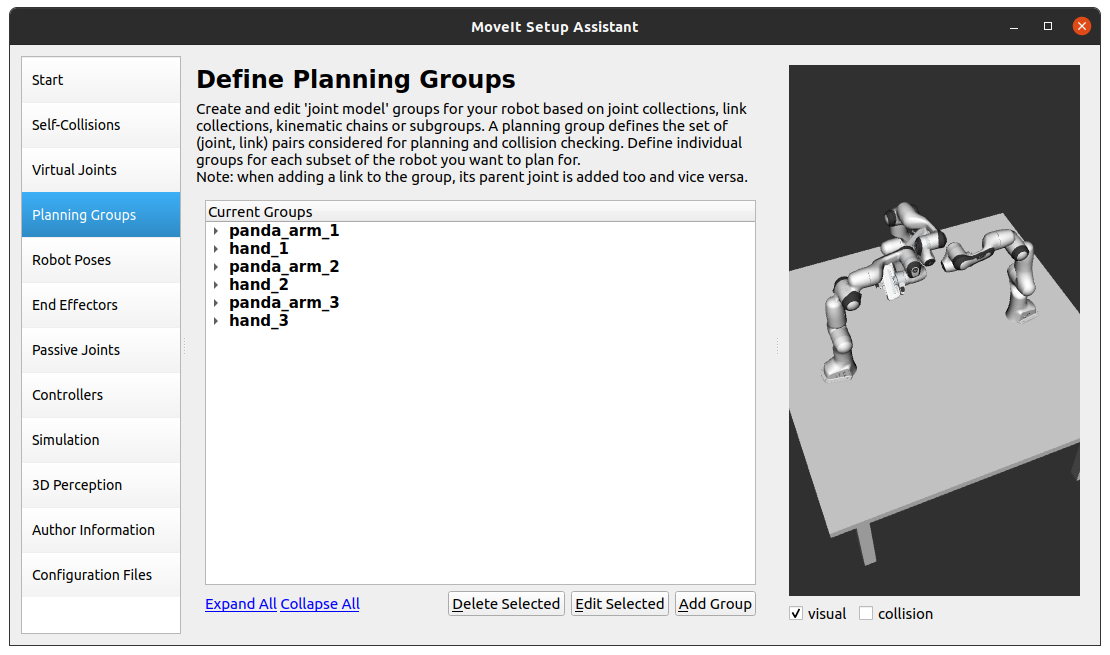


Figure 7(d) – Planning Group Pane



Figure 7(e) – Robot Poses Pane

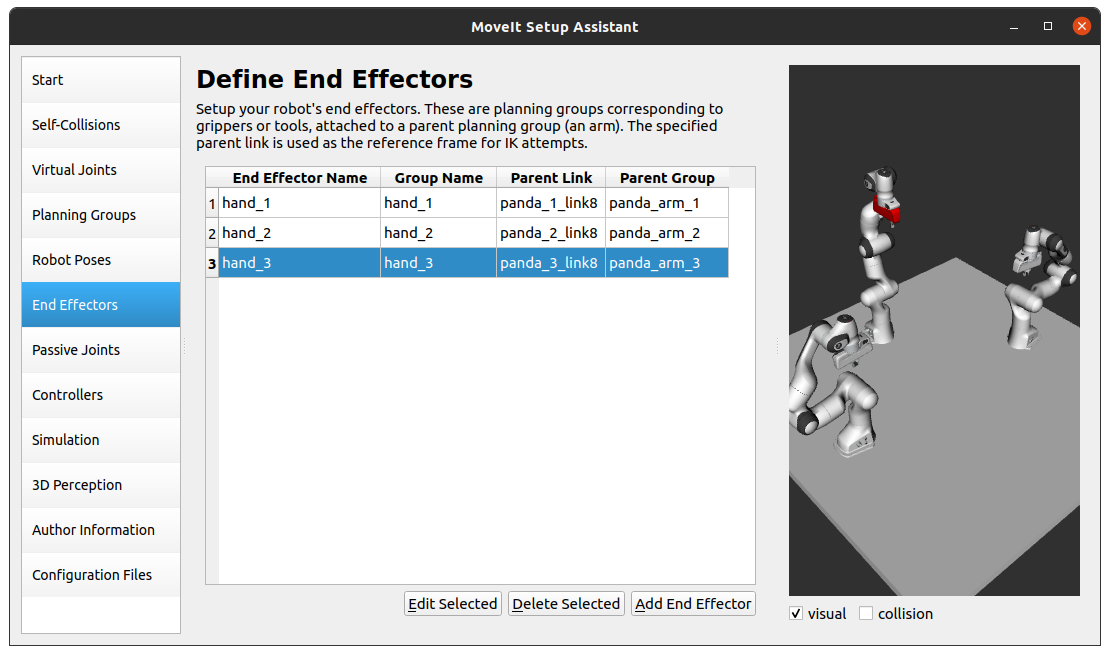


Figure 7(f) – End Effector Pane

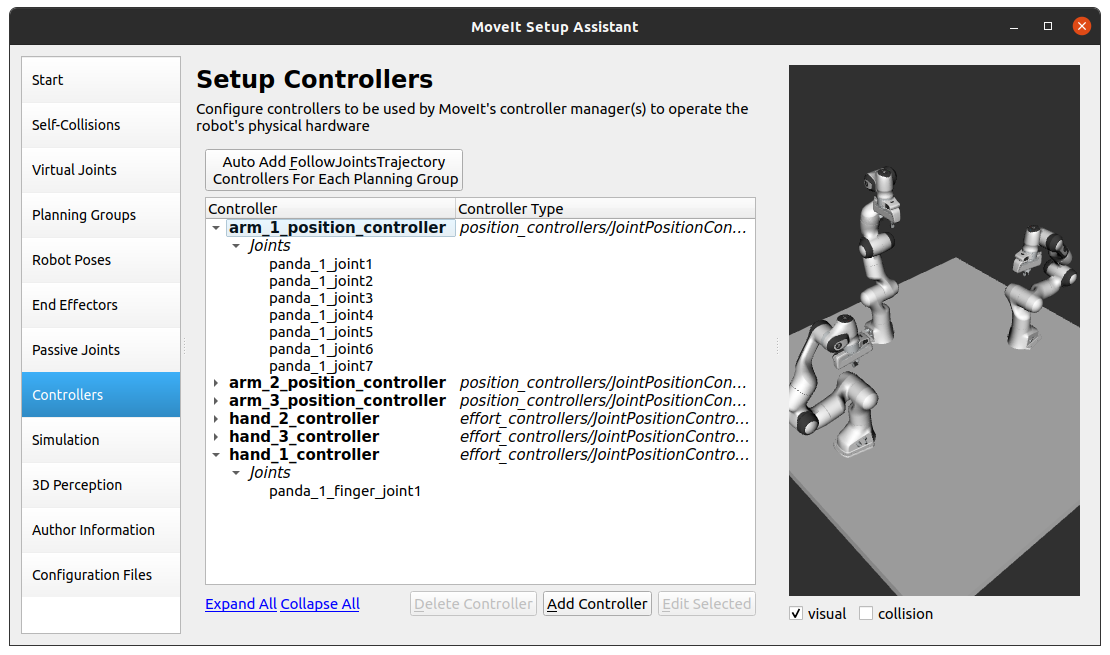


Figure 7(g) – Controllers Pane

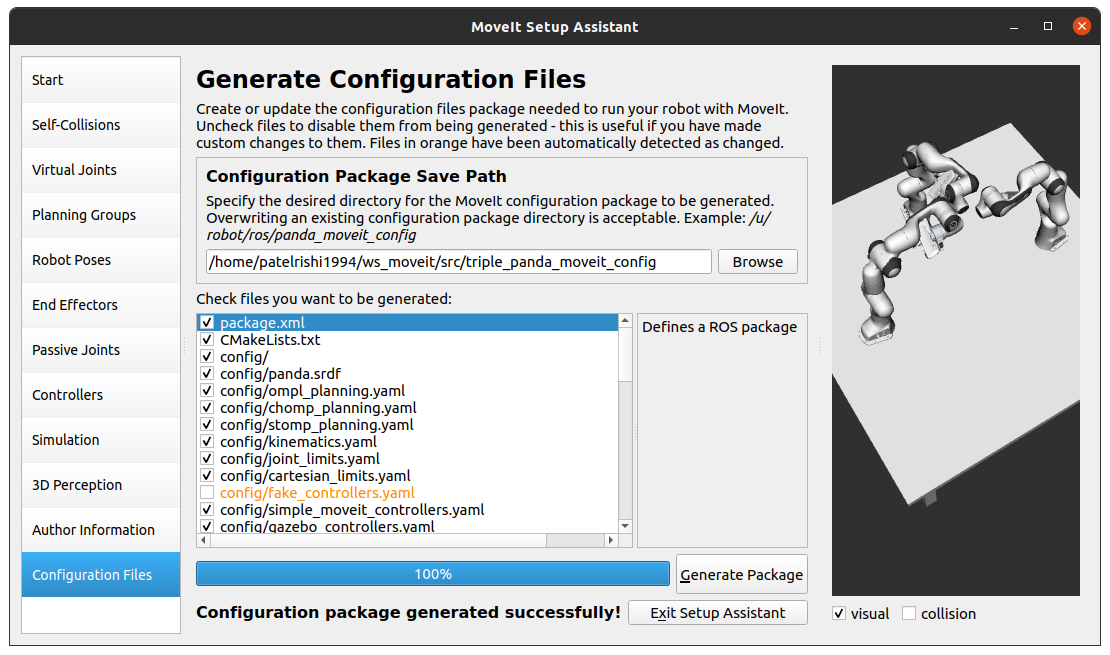


Figure 7(h) – Generate Configuration Files/ROS package Pane

### Launch Rviz

After the user has gone through all the steps of the MoveIt Setup Assistant and finished generating the ROS package. The *“demo.launch”* file can be launched in the following manner by using a terminal to launch the simulation in Rviz. Rviz provides many functionalities that allow beginners to learn and conduct experiments. Rviz can also be used as debugging tool for users who like to code in C++ / Python.

roslaunch <ros\_package\_name> demo.launch

In our case, the command will look like the following, and Rviz is launched as per shown in Fig 8. The setup was configured while going through the MoveIt Setup Assistant and working with URDF.

roslaunch triple\_panda\_moveit\_config demo.launch

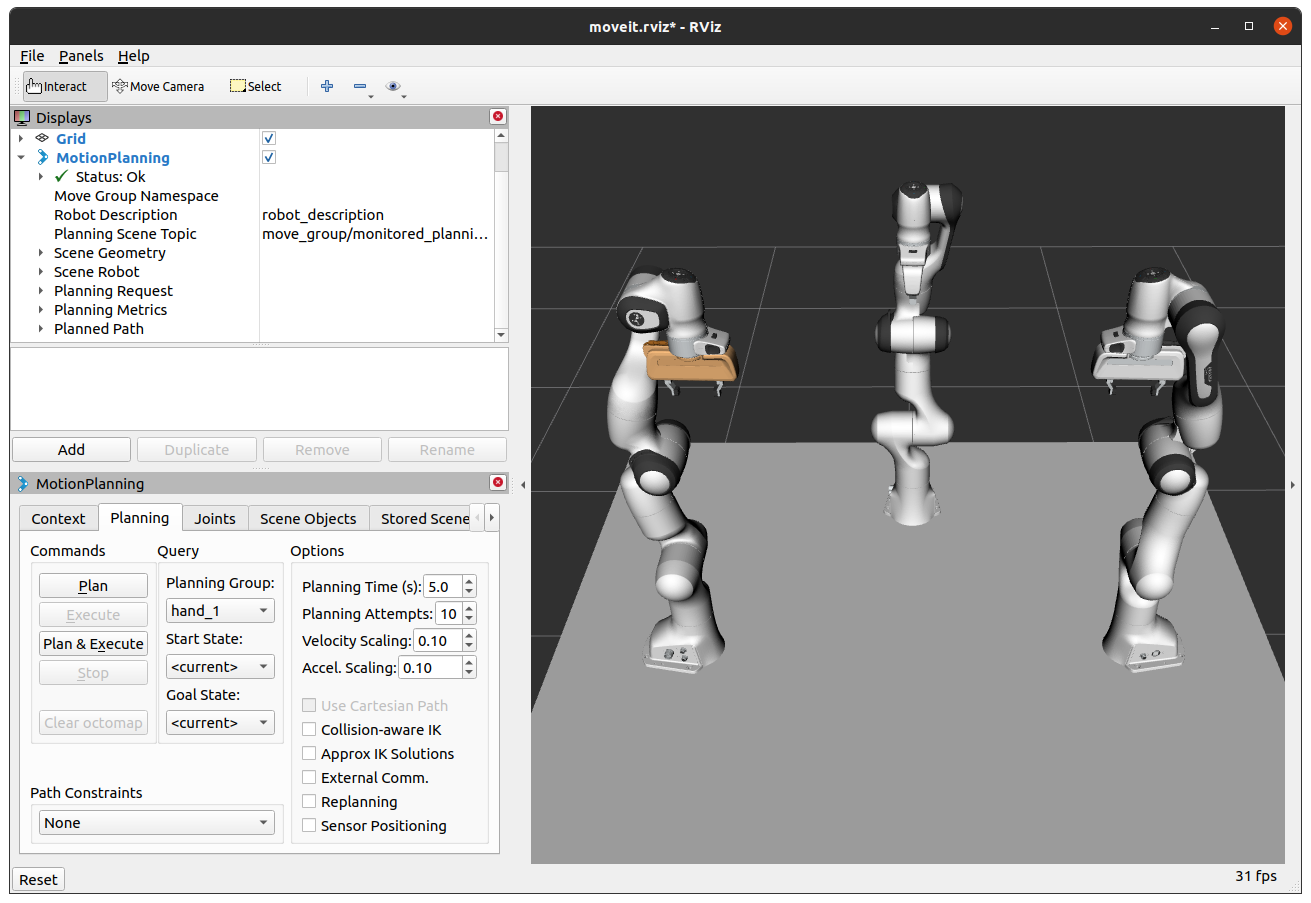


Figure 8 – Running *“demo.launch”*

# COST COLLABORATION STRATEGY

As previously stated, many firms have invested in the implementation of various ways for scheduling multi-robot collaboration, including the use of artificial intelligence. We have proposed a more cost-effective technique of scheduling jobs according to their relative importance, which we shall discuss in greater detail in Section 3.3. Before we go any further, let us consider why we chose the technique that we did.

## Sorting Algorithms

A sorting algorithm is an algorithm that puts elements of the list/array into an order. Frequently used orders are numerical, lexicographical, ascending, and descending. For optimum efficiency, input data should be stored in a data structure that allows random access rather than the one that only allows sequential access.

The output of the sorting algorithm must satisfy two conditions [21]:

1. The output is in monotonic order.
2. The output can be in a different order than the input although it must contain all the elements of the input.

There are many different sorting algorithms available, and they can be classified into different categories although we are more interested in their performance. So to evaluate sorting algorithms on their performances Big O Notations are used. The comparison of different sorting algorithms based on big O notation is shown in Fig. 9. We’ll discuss the notations in section (3.1.1) although at this time if you take a look at the picture, you’ll find out why we choose to go with the “Merge-sort” algorithm. Its because the Merge-sort algorithm has the best, average, and worst time complexity as **n log (n)**. Meaning it will take the same amount of time to finish sorting in each case. The reason we did not choose “Heapsort” is that it’s unstable.

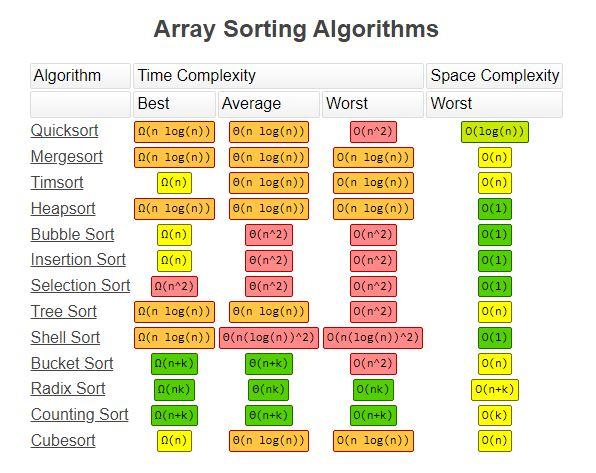


Figure 9 – Time & Space Complexity of Sorting Algorithms [22].

### What is Big Omega(Ω), Big Theta(Θ) & Big Oh(O) Notation? [23]

Big Omega, Theta, and Oh notation are a way of describing how your code’s performance depends on the amount of data its processing (Fig. 9)). Big Omega, Theta, and O allow us to calculate the best, average, and worst possible runtime of an algorithm respectively. These notations show how long an algorithm takes by **time complexity** and how much space an algorithm uses by **space complexity**. The fastest to slowest clarification of these notations can be written as:

**1 < log(n) < √n < n < n log(n) < n² < 2n< 3n < nn**, as shown in Fig. 10.

1. O(**1**) – Speed is constant. It doesn’t change with the size of the data set.
2. O(**log n**) – Logarithmic change, is a little more time-consuming than O(1).
3. O(**n**) – Linear change, Time taken is directly proportional to the size of the data set.
4. O(**n log n**) – Linearithmic change, Time increases by a multiplier of 1 with the size of the data set doubles.
5. O(**n²**) – Quadratic change, time consumed increases by four times more when the size of the data set doubles.
6. O(**2n**) – Exponential change, time takes twice long for every new element added to the data set.
7. O(**n!**) – Factorial change, ([n! Times](https://en.wikipedia.org/wiki/Factorial)) longer time consuming directly proportional to the size of the data set.

Chart

Description automatically generated

Figure 10 – Big O Complexity chart [22]

## Merge-Sort

The “Merge Sort” algorithm is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer. It divides the problem into smaller sub-problems, solves sub-problems by calling them recursively until solved, and then combines the sub-problems to get the final solution of the whole problem.

For example, if an array of integers needs to be sorted by using the Merge sort algorithm. First, the algorithm will divide the array into two (2) halves, calls itself recursively to divide those two (2) halves into smaller subproblems, until each sub-problem consists of no less than a single element, Now when the sub-problems cannot be divided into smaller pieces, It begins to solve the problem by sorting while merging the sub-problems, to do so merge sort compares each element before it merges them to form an array, For that reason, Merge-sort is also classified as comparison-based sorting, and then calling merge recursively until we get the final result of a sorted array of integers. Fig. 11 shows the complete operation of the Merge-sort algorithm on an array of integers.

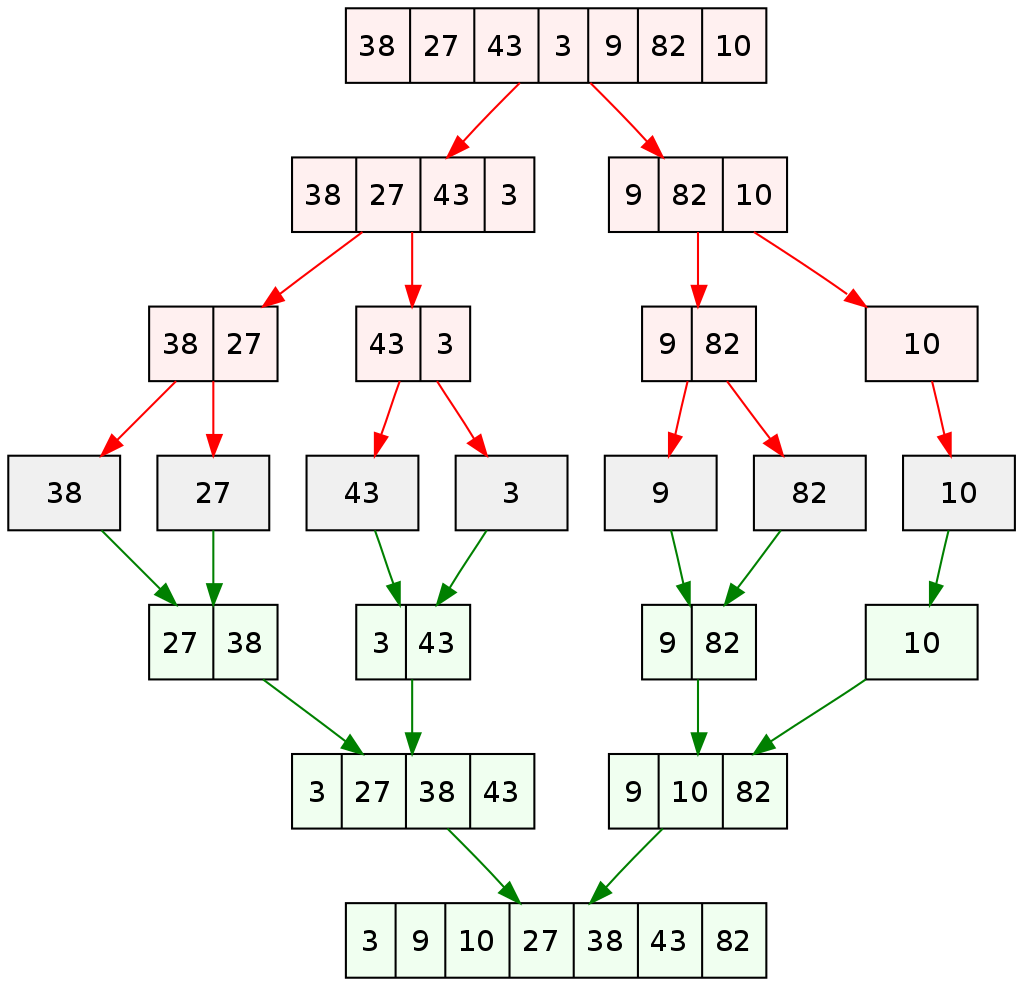


Figure 11 – A recursive Merge Sort algorithm is used to sort an array of 7 integers [21].

## Implementation of the Merge-Sort algorithm

In this project, we are asking that the order of the task be known or should be assigned with priority so that it can be ordered accordingly using the Merge sort algorithm. Assigned the priority before using merge sort: In our case, we have a suffix for the object names from 1 to 9 which describes priority in a way that 1 is of the highest priority level and 9 is of the lowest priority level. Now to divide the tasks into three separate groups to assign them to each arm to make sure that each arm does the same amount of work, seems trivial although we did it. Later it can be assigned different tasks while waiting for the current workspace to become available, to plan its next task.

### 

The “Merge Sort” algorithm has been implemented in this project and the method of implementation is depicted in the pseudocode below. We have used a top-down approach to implementing the Merge sort which required two functions *“mergeSort()”* and *“merge()”*. The *“mergeSort()”* function is used to break down the given problem into smaller sub-problems, i.e., A list is split into two sub-lists first and then calling it recursively on those two (2) sub-lists to break them into even smaller pieces until the size of the sub-list becomes one (1). After that, the *“merge()”* function comes into play and starts merging the sub-lists while sorting the elements of it until the list has been completely sorted and merged.

**Pseudocode:**

**function** mergeSort(list, listFirstIndex, listLastIndex)is

// Check if there is more than one element in the given list

**if** listFirstIndex is less than listLastIndex then

// To be able to call mergeSort() recursively

// To divide the list into equal size of sub-lists

// To Find the middle index by following expression

// Same as, (listFirstindex + listLastIndex // 2)

listMidIndex = listFirstIndex + (listLastIndex - listFirstIndex) // 2

// Call mergeSort recursively on the first & second half of the list

// To split it into smaller sub-lists

mergeSort(list, listFirstIndex, listMidIndex)

mergeSort(list, listMidIndex + 1, listLastIndex)

// Calling merge() function to sort & merge-

// the sub-lists in to the completely sorted list

merge(list, listFirstIndex, listMidIndex, listLastIndex)

**function** merge(list, listFirstIndex, listMidIndex, listLastIndex):

// Assigning two variables to keep count of number of elements of the temporary lists

**var** leftTempListElementCount = listMidIndex - listFirstIndex + listFirstIndex

**var** rightTempListElementCount = listLastIndex - listMidIndex

// Initializing two temporary lists to split the list in two portions

leftTempList = [0] \* leftTempListElementCount

rightTempList = [0] \* rightTempListElementCount

// Copy list's data into two temporary lists named leftTempList & rightTempList

**for** i in range(0, leftTempListElementCount):

// Copy all the elements of list starting from index 0 up to leftTempListElementCount

leftTempList[i] = list[listFirstIndex + i]

**for** j in range(0, rightTempListElementCount):

// Copy remaining elements of the list into rightTempList

rightTempList[j] = list[listMidIndex + 1 + j]

// Not we initialize three variables just to iterate over three lists

// i to iterate over leftTempList

// j to iterate over rightTempList

// k to iterate over list

**var** i, j, k = 0

// To sort and merge the temporary lists into a list

**while** i < leftTempListElementCount and j < rightTempListElementCount:

// Compare i'th element of leftTempList to j'th element of rightTempList to sort.

**if** leftTempList[i] <= rightTempList[j]:

// Add the i'th element of the leftTempList at the k'th index of list

// And Increase i's value by adding 1

list[k] = leftTempList[i]

i = i + 1

**else**:

// Add the j'th element of the rightTempList at the k'th index of the list

// And Increase j's value by adding 1

list[k] = rightTempList[j]

j = j + 1

// Increase k's value by adding 1.

k = k + 1

// To copy remaining elements of leftTempList to end of the list

**while** i < leftTempListElementCount:

list[k] = leftTempList[i]

i = i + 1

k = k + 1

// To copy remaining elements of rightTempList to end of the list

**while** j < rightTempListElementCount:

list[k] = rightTempList[j]

j = j + 1

k = k + 1

We have applied the Merge sort to sort four lists in total. Those are three lists of objects assigned to each arm and the list of all the objects in the shared workspace. We’ve explained the diagram representation of the top-down merge sort algorithm application on the list of objects in Fig 12(a). The red-colored text represents that the list is being split into smaller pieces using the function *“mergeSort()”*, recursively. The green-colored text represents that the elements of the lists are being sorted and merged using the function *“merge()”*. And the blue-colored numbers indicate the order in which the steps are processed.

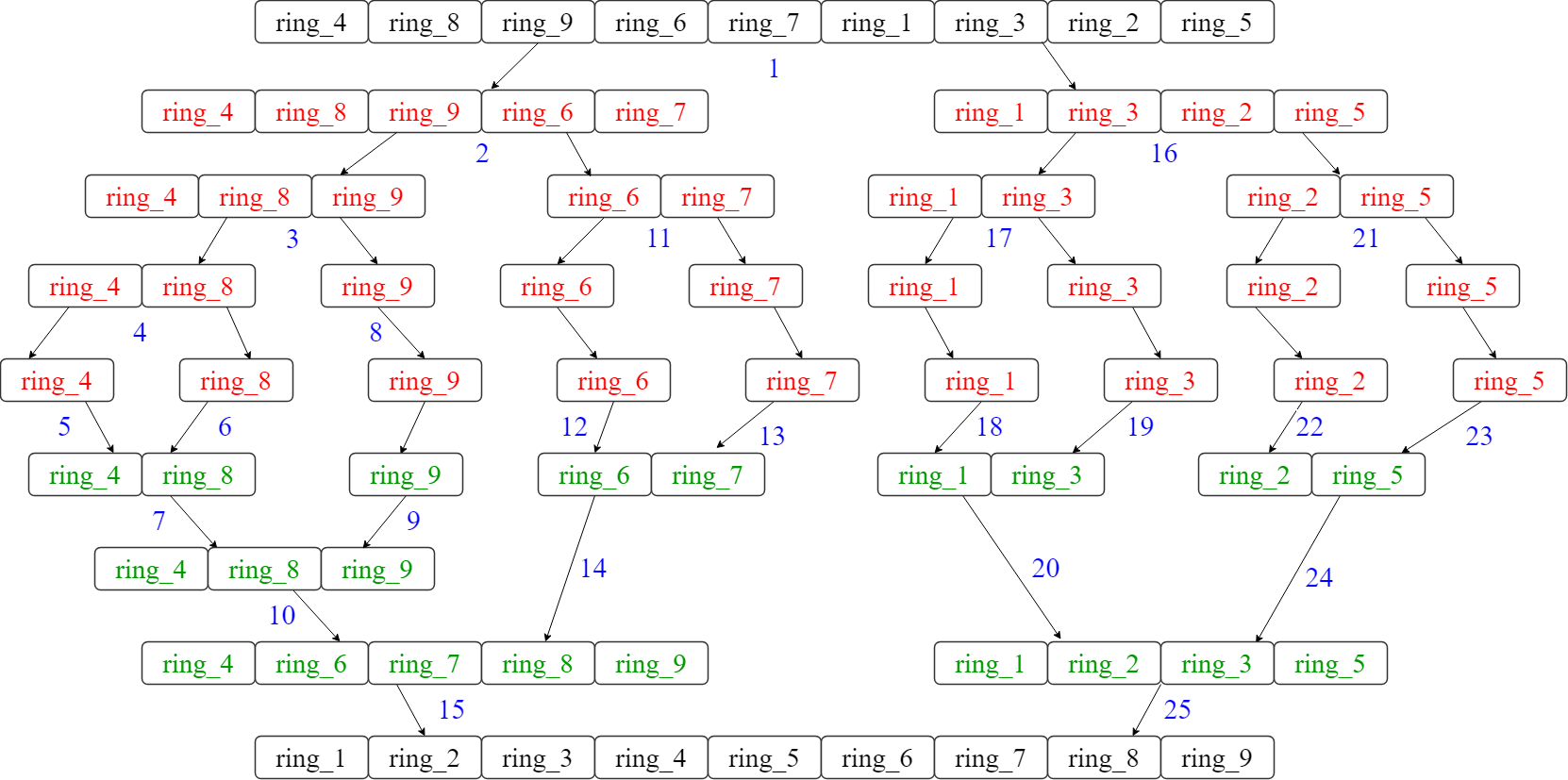


Figure 12(a) - Step-by-step sorting of the list of objects

The list is recursively being divided into two (2) halves till the size of the sub-list becomes one (1) (Fig. 12(a)). Then the *“merge()”* process starts sorting and merging the list till the complete list is sorted & merged. The Merge Sort algorithm is used in this project to sort the list of objects and the list of assigned objects to each arm in ascending order and the results can be viewed in Fig. 12(b). So, we can allow each robotic arm to verify that the task is in the desired order before it goes in for planning and execution.

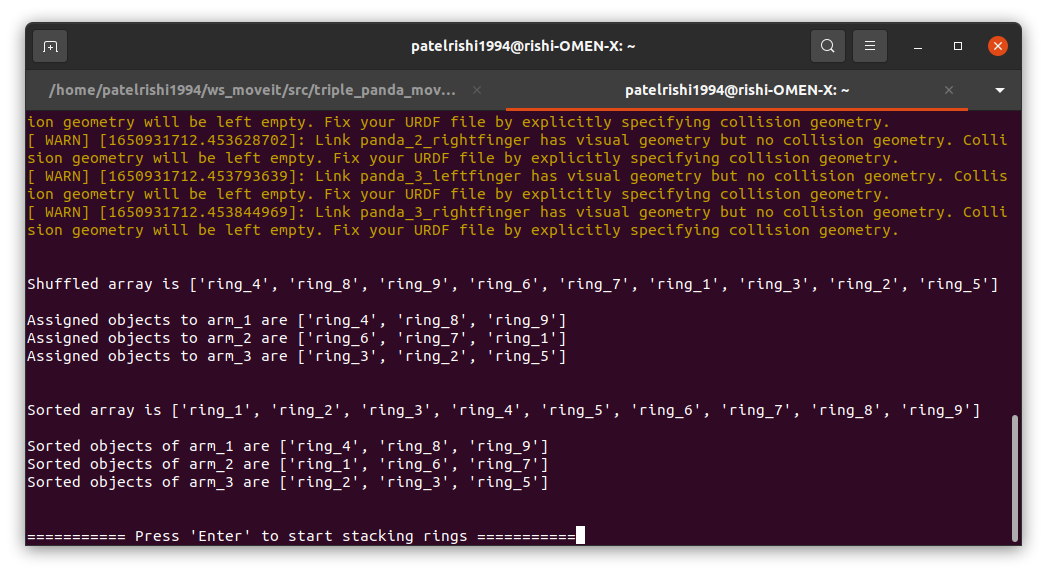


Figure 12(b) – “Merge Sort” list of objects and Assigned objects.

# RESULTS & ANALYSIS

## Experimental Setup

To set up the environment for the experiment, we did not use any hardware except a PC (with Windows 10 installed) and an SSD. Ubuntu-20.04 (desktop version) was installed first onto the SSD and the PC was booted through SSD to run Ubuntu. Then first, we downloaded ROS – Robot Operating System on Ubuntu and then MoveIt. To create the desired setup, we also downloaded the ROS package named *“franka\_description”* just to get the URDF for the Panda Arm-Robot. We edit the URDF according to mentioned in the *User’s Manual*, to add two additional arms and the table with 4 legs as the base frame for those three Panda Arm-Robots. Then MoveIt Setup Assistant was run and the edited URDF was loaded into it to configure the ROS package named **“triple\_panda\_moveit\_config”** with desired controllers (Fig. 7(g)), arm groups (Fig. 7(d)), end-effectors (Fig. 7(f)), and poses (Fig. 7(e)). Then the *“demo.launch”* file was launched to get the simulation running for our experiments which were conducted through Python scripts. The setup for the experiment can be seen in Fig. 13. The first approach shows the movement of an arm through poses with seven elements in a list and the second approach shows the movement of an arm through cartesian motion using three elements in a list.

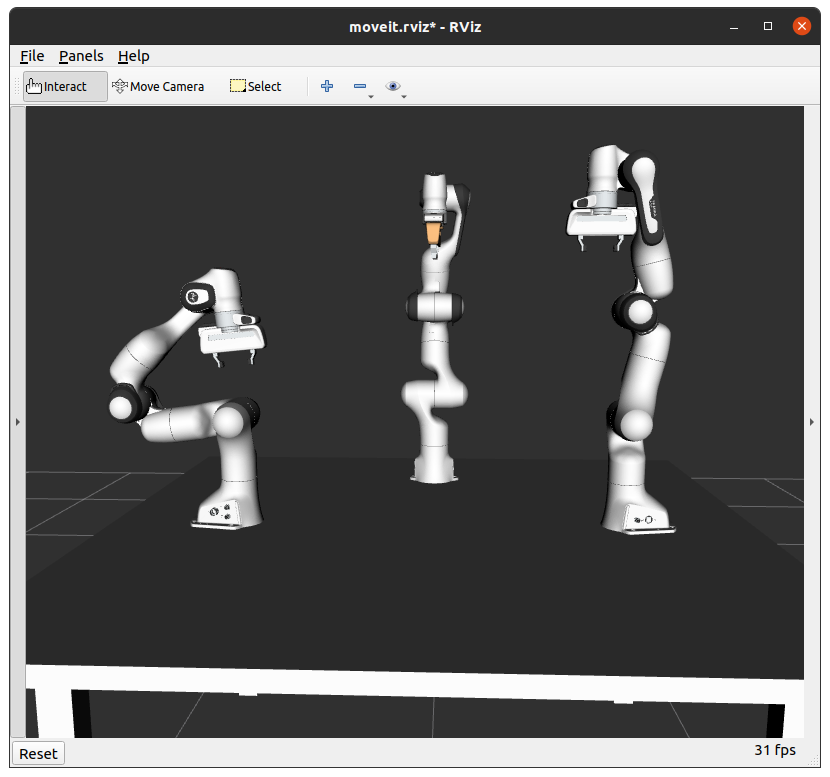


Figure 13 – Setup to conduct Experiment

## Moving Robot-arm using poses

To conduct this experiment, we prepared a Python script that used arm pose which consists of a list of seven elements for which the first three were positional axis X, Y, and Z & remaining four elements were for orientation X, Y, Z, and W. During this experiment non-necessary arm movements was noted quite often, which showed that the movement of the robot-arm is not the movement which we desire to do the task in our shared workspace because of its the extra movements it might collide with the base or the other robot-arms.

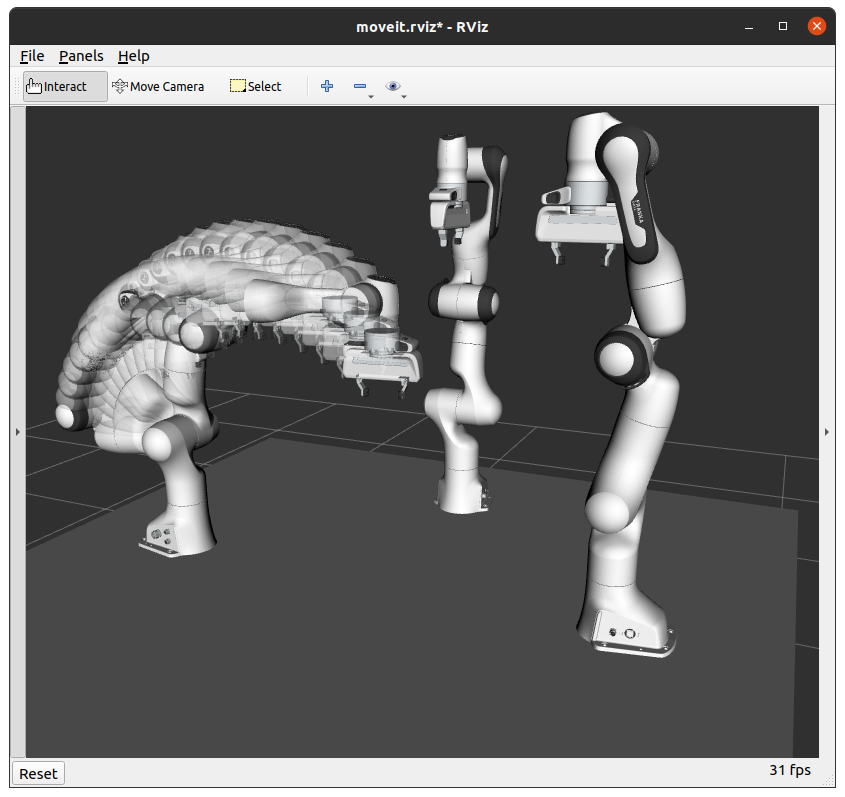


Figure 14 – Arm Pose Goal (Planned & Executed)

## Moving Robot-arm using cartesian motion

To conduct this experiment, we prepared a Python script that used poses the same as we used in the previous approach. Where we used seven elements of the list, the first three for position and the latter four for orientation. Although due to motion planning using cartesian motion we were able to eliminate any extra movements from the robot arm (Fig. 15) and it also took less time to finish the planning and the execution of the motion. Therefore, we decide to move ahead with this method to implement in our project.

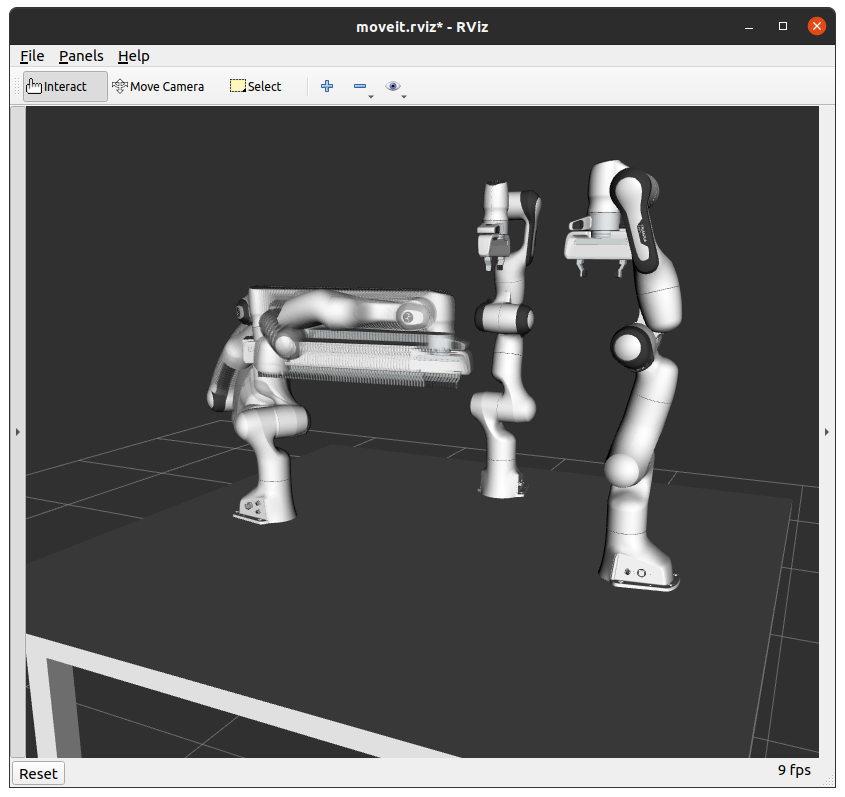


Figure 15 – Cartesian Motion (Planned & Executed)

## Task Scheduling using the “Merge-Sort”

To conduct this experiment, we created two python scripts. First script to generate nine (9) collision objects into the shared workspace resting on the tabletop (Fig. 16). The second script included the implementation of the “Merge-Sort” algorithm and function to stack those nine (9) collision objects using three (3) Panda Robot arms in a simulated environment.

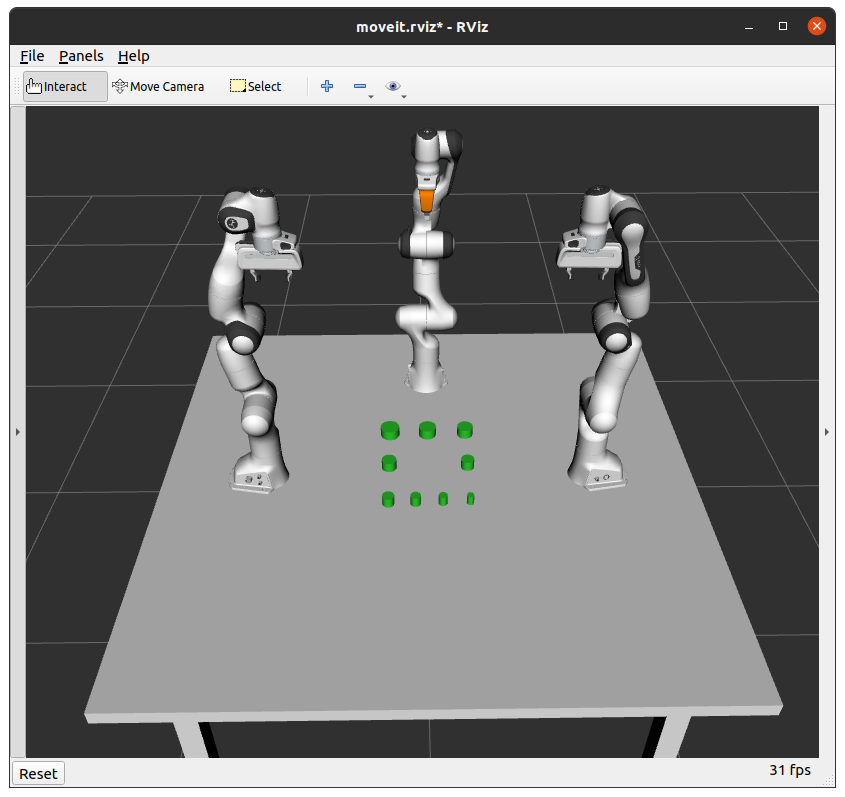


Figure 16 – Collision object resting on the tabletop.

When we execute the second script the unordered list of objects and list of assigned objects to each arm are printed in the terminal. Later those lists are sorted using the recursive “Merge Sort” algorithm and then the sorted list of nine (9) objects and the lists of assigned objects to each arm are printed onto the terminal (Fig. 17(a)). Since the nine (9) objects have been split into three parts and assigned to each arm, we know which arm is going to move to a place which objects even before they start execution (Fig. 17(a).

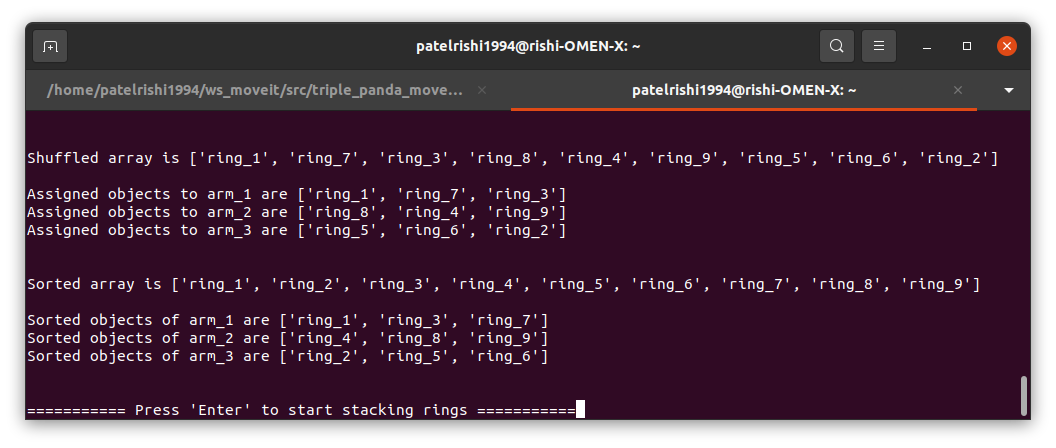


Figure 17(a) – Unsorted List and Sorted list using “Merge Sort”

This information can be useful for experienced programmers. Once we hit enter the stacking of objects starts in ascending order through objects 1 to 9. It can be seen in Fig. 17(b-j). Fig. 17(k) shows the final result after the script has finished running.

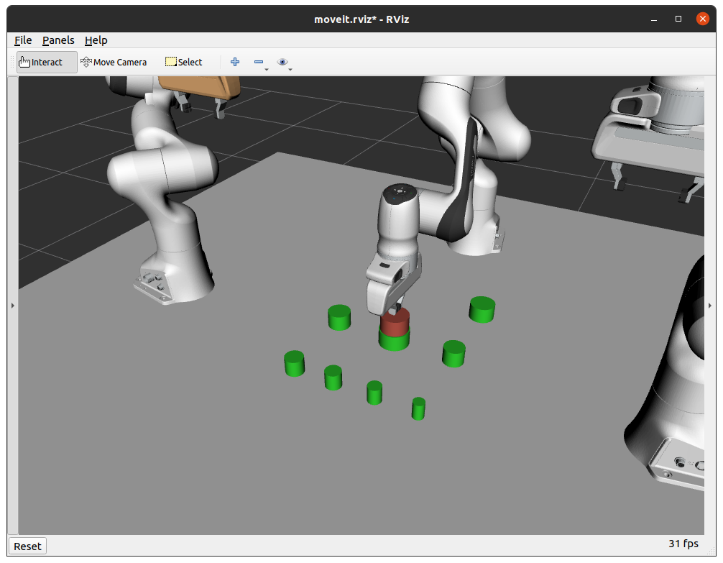
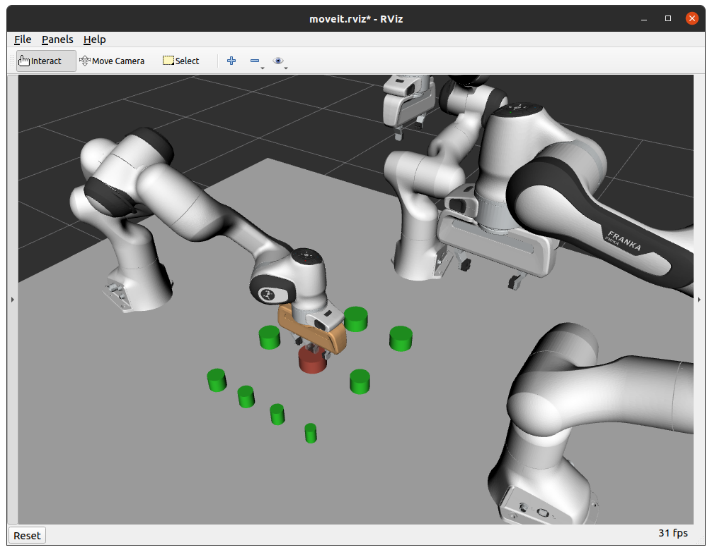


Fig. 17(b) – Arm – 1 placing the 1st Object Fig. 17(c) – Arm – 3 placing the 2nd Object

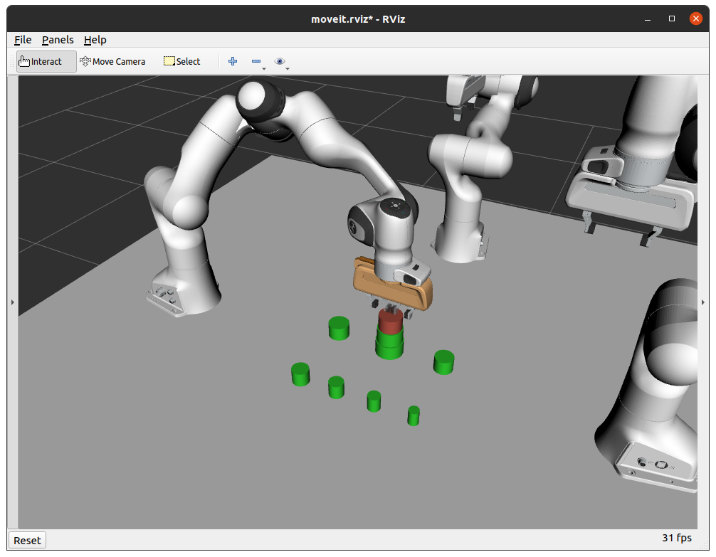
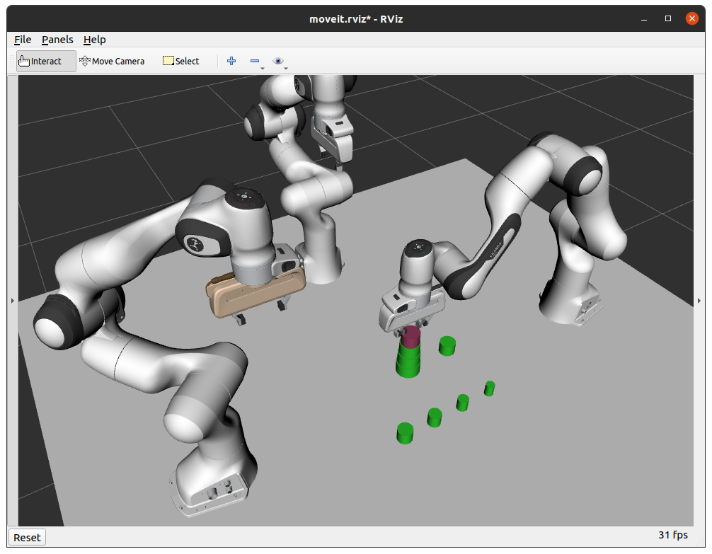
 

Fig. 17(d) – Arm – 1 placing the 3rd Object Fig. 17(e) – Arm – 2 placing the 4th Object

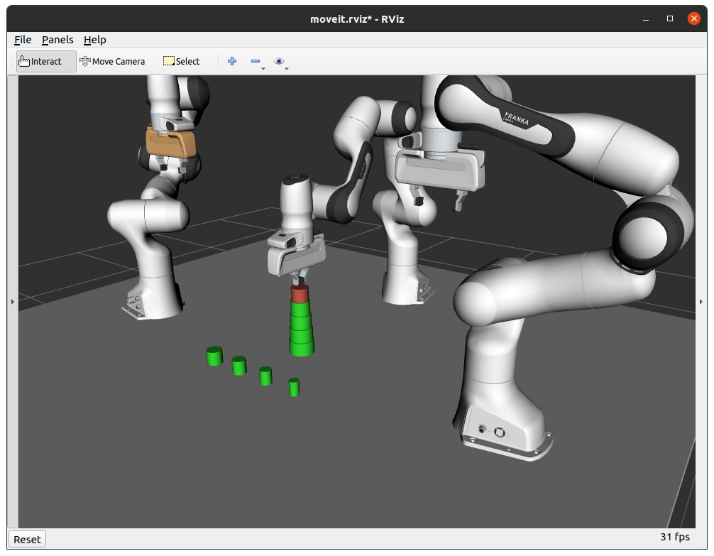
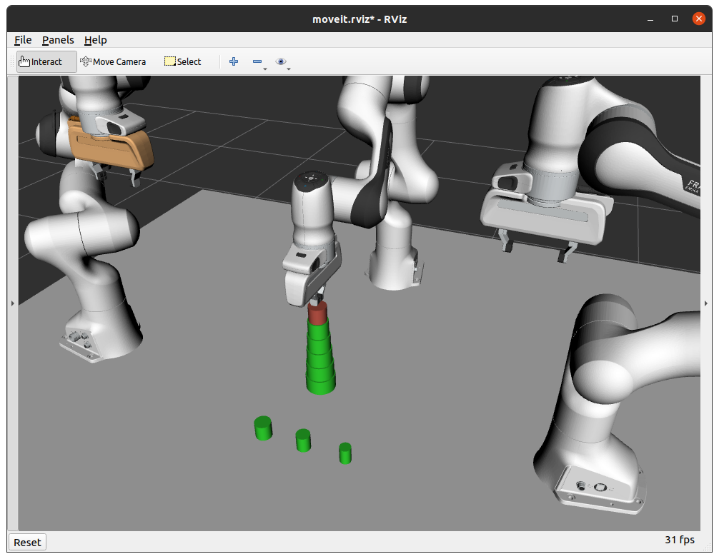
 

Fig. 17(f) – Arm – 3 placing the 5th Object Fig. 17(g) – Arm – 3 placing the 6th Object

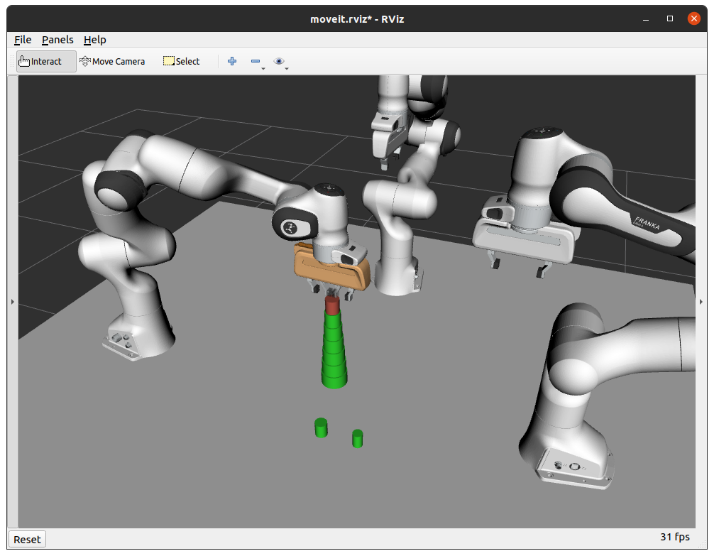
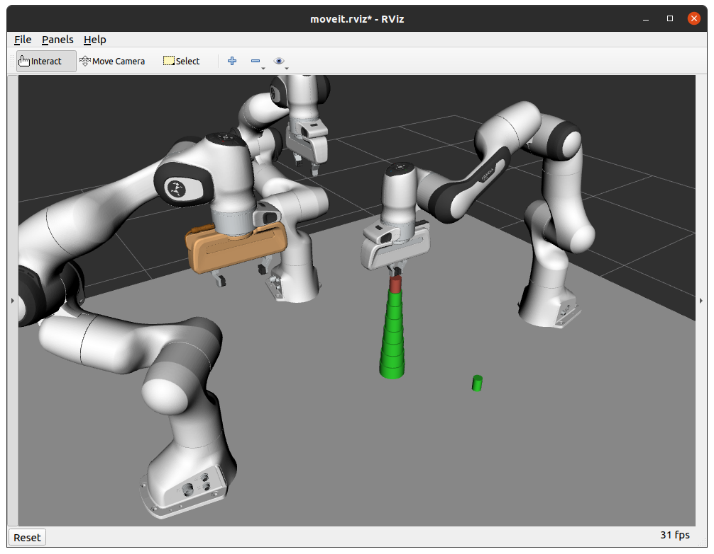
 

Fig. 17(h) – Arm – 1 placing the 7th Object Fig. 17(i) – Arm – 2 placing the 8th Object

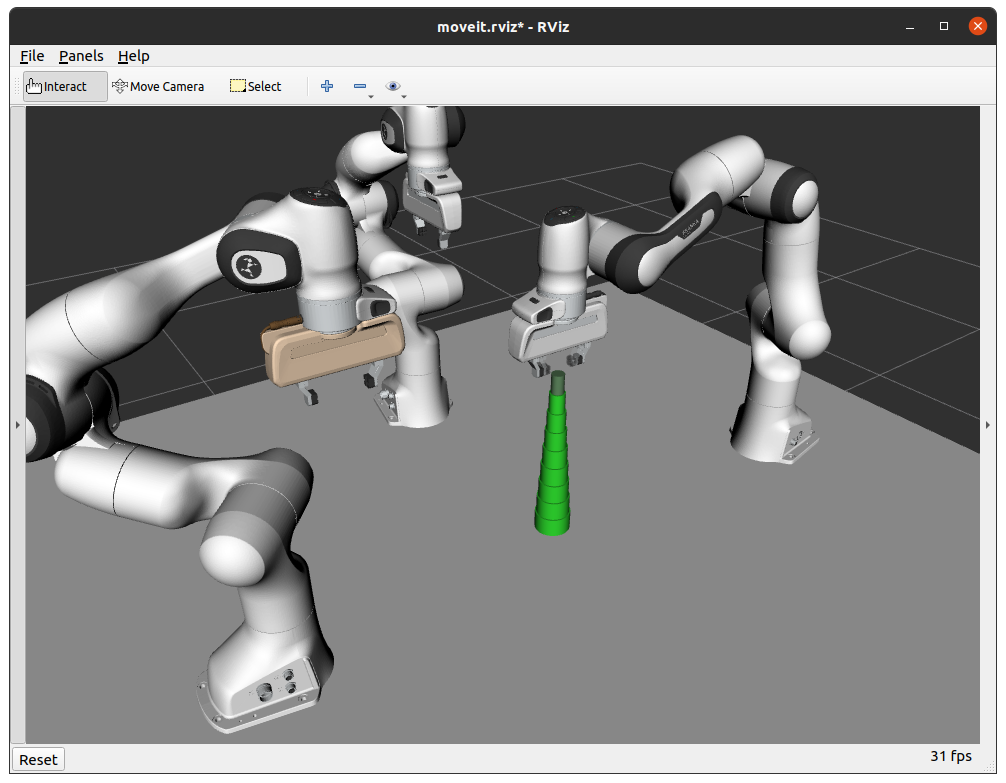


Figure 17(j) – Arm 2 placing the 9th object

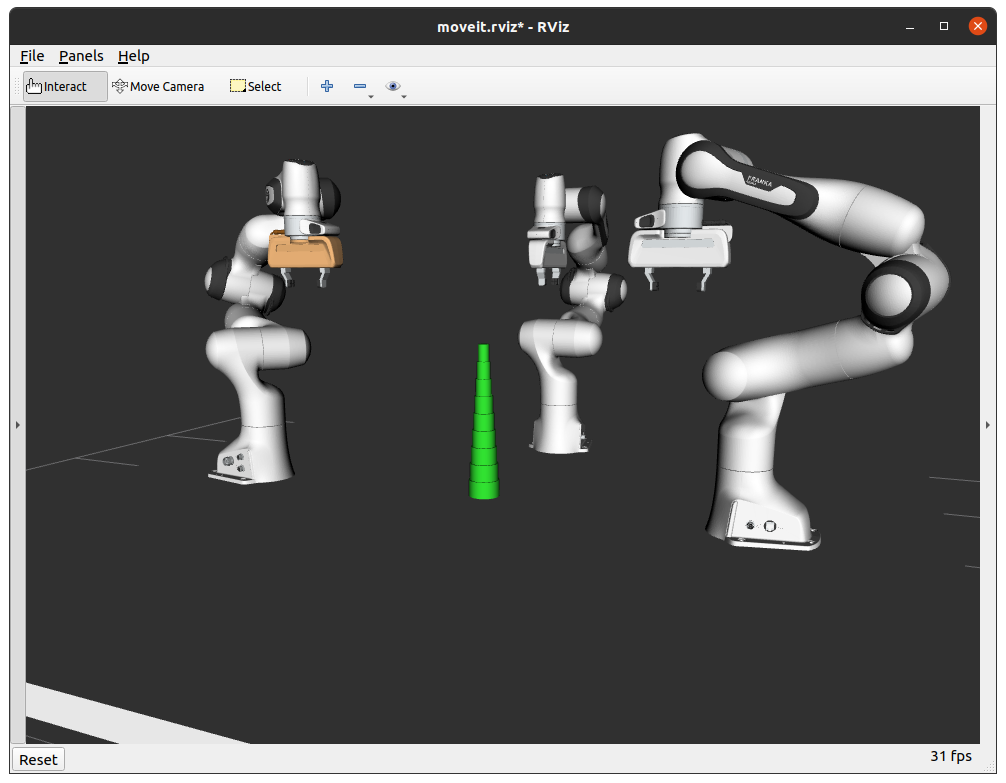


Figure 17(k) – All arms are at Rest, all tasks are complete

# CONCLUSION

In this paper, we presented the development of the Multi-Robot Collaboration for assembly tasks in steps. First, we briefed about each component being used in the project, then elaborated the setting up of the simulation of three Panda robot arms on top of a table using MoveIt setup assistant, then we saw how to launch the simulation in Rviz, then we elaborated on cost collaboration strategy for the project and discussed about a generic sorting algorithm named “Merge sort”. After that we saw the implementation of the top-down approach of the algorithm and also discussed about how it works with in the project using diagram representation and pseudocode. Then we showed how the setup was prepared to conduct the experiment and then we experimented using two different methods available for manipulating simulated robot arm using Python scripts. then we finalize the method of manipulation and conduct another experiment after implementing the “Merge-Sort” sorting algorithm in top-down manner to schedule prioritized tasks to assign each arm with same amount of assembly sub-tasks and later we saw how the task was being done by each Panda arm in proper manner. Through experiments, we were able to show how each part has been set up the way it is and how it works perfectly to complete the multi-robot collaboration assembly task in the shared workspace.

# Additional Information & Future Improvements

* The application of these robots can be performed the way it has been shown in this project or each arm can be mobilized to work in different places by employing of RPCs.
* Future improvements can include the solution, where the arm which is not currently moving to complete the task won’t have to wait for other arms to finish moving. Maybe by implementing dynamic collision checking for real-time collision detection.

# Acknowledgment

Without the assistance and encouragement of several people, this endeavor would not have been possible. As a result, I'd want to express my gratitude to Dr. Weitian Wang, my Project Advisor, for his unwavering support and assistance. At every stage of this endeavor, I was given appropriate guidance. He gave me all of the materials, pdfs, and websites that I needed for my project. In general, I'd want to express my gratitude to the faculty and students at Montclair State University's Department of Computer Science for their collaboration.

# REFERENCES

[1] H. Ding, M. Schipper, and B. Matthias, "Collaborative behavior design of industrial robots for multiple human-robot collaboration," in *IEEE ISR 2013*, 2013: IEEE, pp. 1-6.

[2] S. Zhang, Y. Chen, J. Zhang, and Y. Jia, "Real-time adaptive assembly scheduling in human-multi-robot collaboration according to human capability," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020: IEEE, pp. 3860-3866.

[3] B. Huang, M. Ye, Y. Hu, A. Vandini, S.-L. Lee, and G.-Z. Yang, "A Multi-Robot Cooperation Framework for Sewing Personalized Stent Grafts," *arXiv e-prints,* p. arXiv: 1711.03195, 2017.

[4] H. Shen, L. Pan, and J. Qian, "Research on large-scale additive manufacturing based on multi-robot collaboration technology," *Additive Manufacturing,* vol. 30, p. 100906, 2019.

[5] N. Yao and F. Zhang, "Optimal real-time scheduling of human attention for a human and multi-robot collaboration system," in *2020 American Control Conference (ACC)*, 2020: IEEE, pp. 30-35.

[6] S. Zhou and L. Tian, "Would you help a sad robot? Influence of robots’ emotional expressions on human-multi-robot collaboration," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*: IEEE, pp. 1243-1250.

[7] L. Trebaol. "The Robotics Glossary : 9 (more!) Robotics Terms You Need to Know." <https://www.energid.com/blog/the-robotics-glossary-9-more-robotics-terms-you-need-to-know> (accessed.

[8] G. Boschetti, M. Bottin, M. Faccio, and R. Minto, "Multi-robot multi-operator collaborative assembly systems: a performance evaluation model," *Journal of Intelligent Manufacturing,* vol. 32, no. 5, pp. 1455-1470, 2021.

[9] L. Yan, T. Stouraitis, and S. Vijayakumar, "Decentralized ability-aware adaptive control for multi-robot collaborative manipulation," *IEEE Robotics and Automation Letters,* vol. 6, no. 2, pp. 2311-2318, 2021.

[10] P. Schillinger *et al.*, "Adaptive heterogeneous multi-robot collaboration from formal task specifications," *Robotics and Autonomous Systems,* vol. 145, p. 103866, 2021.

[11] Z. Wang and M. Gombolay, "Learning scheduling policies for multi-robot coordination with graph attention networks," *IEEE Robotics and Automation Letters,* vol. 5, no. 3, pp. 4509-4516, 2020.

[12] "FRANKA EMIKA - PANDA." <https://wego-robotics.com/franka-emika-panda/> (accessed.

[13] K. S. B. Hasan. "What, Why and How of ROS." <https://towardsdatascience.com/what-why-and-how-of-ros-b2f5ea8be0f3> (accessed.

[14] H. Deng, J. Xiong, and Z. Xia, "Mobile manipulation task simulation using ROS with MoveIt," in *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2017: IEEE, pp. 612-616.

[15] O. Robotics. "Robot Operating System Documentation." Open Robotics. <https://wiki.ros.org/> (accessed.

[16] O. Robotics. "Robot Operating System." Wikipedia The Free Encyclopedia. <https://en.wikipedia.org/wiki/Robot_Operating_System#cite_note-7> (accessed.

[17] P. Robotics. "MoveIt - Concepts." PickNik Robotics. <https://moveit.ros.org/documentation/concepts/> (accessed.

[18] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785,* 2014.

[19] P. Robotics. "MoveIt Setup Assistant." PickNik Robotics. <https://ros-planning.github.io/moveit_tutorials/doc/setup_assistant/setup_assistant_tutorial.html> (accessed.

[20] D. T. Coleman IV, "Methods for Improving Motion Planning Using Experience," University of Colorado at Boulder, 2017.

[21] "Merge Sort." Wikipedia. <https://en.wikipedia.org/wiki/Merge_sort> (accessed.

[22] E. Rowell. "Know Thy Complexities!" DISQUS. <https://www.bigocheatsheet.com/> (accessed.

[23] R. Edwards. "Time Complexity: Big O Notation Fastest to Slowest Time Complexity." <https://knovhov.com/big-o-notation-fastest-to-slowest-time-complexity/> (accessed December, 27, 2020.