

**VISION-BASED SORTING SYSTEM USING ROBOTIC
MANIPULATOR FOR INDUSTRY 4.0
PROJECT AND VIVA-VOCE REPORT**

Submitted by

P SAI CHARAN	(17130029)
S GOWTHAM	(17130044)
M PARTHIBAN	(17130048)
R PRAVEEN	(17130071)

Under the guidance of

MS. N SEENU, M.E., (Ph. D.)

and

Dr. R M KUPPAN CHETTY, M.Tech., Ph.D.

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

MECHATRONICS ENGINEERING



DEPARTMENT OF MECHANICAL ENGINEERING

SCHOOL OF MECHANICAL SCIENCES

HINDUSTAN INSTITUTE OF TECHNOLOGY AND SCIENCE

PADUR 603 103

APRIL 2021



BONAFIDE CERTIFICATE

Certified that this Project Report titled "**VISION-BASED SORTING SYSTEM USING A ROBOTIC MANIPULATOR FOR INDUSTRY 4.0**" is the bonafide work of **Mr. P SAI CHARAN (17130029), Mr. S GOWTHAM (17130044), Mr. M PARTHIBAN (17130048), Mr. R PRAVEEN (17130071)** who carried out the project work under my supervision during the academic year 2021.

HEAD OF THE DEPARTMENT

Dr. D. Dinakaran, Ph.D.
Professor and Head
Centre for Automation and Robotics

SUPERVISORS

Ms. N. Seenu, M.E, (Ph.D.)
Assistant Professor
Centre for Automation and Robotics

Dr. RM. Kuppan Chetty, M.Tech., Ph.D.
Associate Professor
Centre for Automation and Robotics

INTERNAL EXAMINER

Name:_____

Designation:_____

EXTERNAL EXAMINER

Name:_____

Designation:_____

Project Viva-Voce conducted on_____

ABSTRACT

In the recent decade, vision-guided robot manipulation systems have automated several assembly line operations in industries. Our project is inclined towards Industry 4.0 to enhance material sorting by incorporating IoT communication and autonomous control. The drawbacks of conveyor-based sorting systems such as lack of object placement planning and material damage are overcome using a dexterous robot manipulator. The investigations carried out for robot manipulation include ROS-based software architecture development, robot teleoperation, and coordinate transformation. This project aims to extend the related work by interfacing a 6 DoF Kinova robot manipulator with a 2-D vision system to demonstrate color-based sorting operation. The object's color signature is acquired using blob detection and its position and orientation parameters are communicated to the API through ROS topics and services. The developed algorithm correlates the camera frame and robot frame, allowing the manipulator to interpret the object's location to establish Cartesian control. The manipulator's trajectory execution during the pick and place task is validated through real-time experimentation in a static work environment. Upon task completion, IoT communication is implemented to log the details and timestamps of the sorted boxes in a spreadsheet and a webpage embedded interactive dashboard. The users are notified of the product's status and summary via auto-generated emails after the sorting operation. The developed system could be used to automate complex material handling tasks using ROS-supported articulated robot manipulators.

Keywords: Machine vision, color-based sorting, motion planning, collision avoidance, robot manipulator, ROS, Industry 4.0, IoT

ACKNOWLEDGEMENT

First and foremost, we would like to thank the Lord Almighty for his presence and immense blessings throughout the project. It is a matter of pride and privilege for us to express profound gratitude to the management of HITS for providing the necessary facilities and support. We are highly elated in expressing sincere and abundant respect to the Vice-Chancellor, **Dr. S.N. Sridhara**, for giving us this opportunity to bring and implement our ideas in this project.

We wish to express our heartfelt gratitude to **Dr. D Dinakaran**, Professor and Head of Centre of Automation & Robotics, for much of his valuable support and encouragement in carrying out this work.

We want to thank our guides, **Ms. N. Seenu** and **Dr. R.M. Kuppan Chetty**, to continually guide and actively participate in our project by providing valuable suggestions to complete the project work. We thank all the technical and teaching staff of the Centre for Automation and Robotics, Mechanical Engineering Department for their support. We also thank our class teacher, Ms. Manju Mohan, for supporting us throughout the project.

P SAI CHARAN

S GOWTHAM

M PARTHIBAN

R PRAVEEN

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	
	LIST OF FIGURES	
	LIST OF TABLES	
1.	OVERVIEW OF THE PROJECT	1
	1.1 INTRODUCTION	1
	1.2 PROBLEM STATEMENT	4
	1.3 OBJECTIVES OF THE PROJECT	4
	1.4 SCOPE OF THE PROJECT	5
2.	LITERATURE SURVEY	6
	2.1 INTEGRATION OF ROBOT MANIPULATOR WITH VISION SYSTEM	6
	2.2 VISION-BASED ROBOT MANIPULATOR	7
	2.3 COLOR SORTING USING A ROBOT MANIPULATOR	8
	2.4 CONTROL OF ROBOT MANIPULATOR THROUGH ROS	10
	2.5 SUMMARY OF LITERATURE SURVEY	11
3.	COMPONENT DESCRIPTION	14
	3.1 COMPONENT SELECTION	14
	3.2 HARDWARE COMPONENTS	14

3.2.1	Kinova JACO ² 6 DoF Robot Manipulator	15
3.2.2	USB Camera	18
3.2.3	Adjustable Camera Mount	19
3.3	SOFTWARE USED	20
3.3.1	Robot Operating System	20
3.3.2	Gazebo 3-D Robot Simulator	23
3.3.3	RViz 3-D Visualization Tool	24
3.3.4	MoveIt Motion Planning Framework	26
3.3.5	Camera Configuration Utility	28
4.	METHODOLOGY AND EXPERIMENTAL SETUP	30
4.1	INTERFACING GAZEBO WITH MOVEIT	30
4.2	SETTING UP THE WORK ENVIRONMENT	31
4.3	CORRELATING THE CAMERA FRAME WITH TOOL FRAME	33
4.4	COLLISION AVOIDANCE	36
4.4.1	Modelling the camera mount in SolidWorks	38
4.4.2	Collision Checking using MoveIt	40
4.5	MOTION PLANNING	41
4.5.1	Setting up a MoveIt Planning Scene	42
4.5.2	Implementing Kinematics Solver	42
4.5.3	Joint-space trajectory execution using MoveIt	42

5.	IOT IMPLEMENTATION THROUGH APPS SCRIPTING	46
5.1	GOOGLE APP SCRIPT	46
5.1.1	Spreadsheet updation through app scripting	47
5.1.2	Developing auto-generated e-mail alerts	47
5.2	CREATING A DYNAMIC DASHBOARD	49
5.2.1	Sending spreadsheet data to webpage	49
5.2.2	Updating webpage dashboard using AJAX requests	49
6.	RESULTS AND DISCUSSION	52
6.1	PACKAGE DETECTION AND TRACKING	52
6.2	COLLISION AVOIDANCE USING MOVEIT	54
6.3	AUTONOMOUS PICK AND PLACE OPERATION	55
6.3.1	Joint-space trajectory execution	56
6.3.2	Cartesian-space trajectory execution	57
6.4	AUTOMATIC DASHBOARD UPDATION	59
6.5	SENDING AUTO-GENERATED EMAILS TO USER	60
7.	FUTURE SCOPE	63
	REFERENCES	65
	APPENDIX	68

LIST OF FIGURES

Figure No	Title of the figure	Page No.
Fig 1.1	Kinova JACO2 6 DoF robotic arm	2
Fig 2.1	Experimental setup to mimic arm gestures	6
Fig 2.2	Slide action gripper	7
Fig 2.2.1	Mounted web camera	7
Fig 2.3	Original image	8
Fig 2.3.1	Binary image	8
Fig 2.4	Experimental setup used for object sorting	9
Fig 2.4.1	Pick and place operation	9
Fig 2.5	Experimental setup used for ball balancing	10
Fig 3.1	Kinova JACO ² 6 DoF robotic manipulator	15
Fig 3.1.1	Base panel with power supply and interface	16
Fig 3.2	USB camera	19
Fig 3.3	Adjustable camera mount	20
Fig 3.4	Node-to node communication	21
Fig 3.4.1	Server-Client communication in ROS	22
Fig 3.5	Kinova manipulator imported in Gazebo GUI	23
Fig 3.6	Robot manipulator in Rviz GUI	25
Fig 3.6.1	MoveIt Rviz plug-in	26
Fig 3.7	Centralized communication of move group node	27
Fig 4.1	MoveIt-Gazebo Interface	31
Fig 4.2	Mechanical mounting of the robot manipulator	32
Fig 4.2.1	Experimental setup	32
Fig 4.3	Tool frame convention	33

Fig 4.4	Camera frame convention	34
Fig 4.4.1	Camera frame and tool frame association	35
Fig 4.4.2	Flowchart of FCL operation	37
Fig 4.5	Camera mount assembly in SolidWorks	38
Fig 4.5.1	Bottom portion of camera holder	39
Fig 4.5.2	Top portion of camera holder	39
Fig 4.5.3	Camera mount model in SolidWorks	40
Fig 4.6	At home position	41
Fig 4.7	Home position of robotic arm executed using MoveIt	43
Fig 4.8	Publishing robot joint states to MoveIt	45
Fig 5.1	Deploying apps script as a web application	46
Fig 5.2	Flowchart of AJAX method	50
Fig 5.3	Column headers of the dashboard	51
Fig 6.1	Package recognition	52
Fig 6.2	Position and orientation values of the package	53
Fig 6.2.1	Real-time package tracking	53
Fig 6.3	Goal position fixed by user	54
Fig 6.3.1	Avoiding collision to reach goal pose	54
Fig 6.3.2	Preview of collision object	54
Fig 6.4	Initial position of the robot	55
Fig 6.4.1	Assigning the goal state	55
Fig 6.4.2	Initiated motion planning	55
Fig 6.4.3	Reached goal state	55
Fig 6.4	Control of robotic arm using joint-space description	56
Fig 6.5	Picking pose of the robot	57
Fig 6.5.1	Wrist orientation relative to the box	57
Fig 6.5.2	Cartesian path traversal	57

Fig 6.5.3	Pre-defined motion along z-axis	57
Fig 6.6	Altering gripper pose relative to package orientation	58
Fig 6.6.1	Grasping and releasing the detected package	59
Fig 6.7	Automatic spreadsheet updation with google apps script	60
Fig 6.8	Updating the dynamic dashboard using AJAX requests	60
Fig 6.9	Auto-generated email as user notification	61

LIST OF TABLES

Table No.	Title of the table	Page No.
Table 2.1	Summary of literature survey	11
Table 3.1	List of components	14
Table 3.2	Various segments of the manipulator	16
Table 3.3	Specifications of the JACO ² Kinova arm	17
Table 4.1	Collision detection capabilities of FCL	37
Table 4.2	Pseudocode of ROS communication for motion planning	44
Table 5.1	Pseudocode of IoT communication during pick and place operation	48

CHAPTER 1

OVERVIEW OF THE PROJECT

1.1 INTRODUCTION

Machine vision technology has significantly stimulated the evolution of industrial automation during the recent decade. Presently, vision is widely employed in industrial assembly lines and supply chains to automate several operations such as material inspection, defect detection, component assembly, palletizing, labeling, barcode reading, sorting, and process control. A typical vision system comprises a camera, an illumination source, image processing software, and actuators. Industrial machines operated using conventional sensors for monitoring and process control proves efficient in operation. However, such prevailing pieces of machinery are constrained to the work environment and often demand immense downtime. Vision is broadly being integrated with industrial machines for the past few decades. To overcome such significant drawbacks. The controller processes the vision system's information to guide the actuators for performing the desired operation autonomously. In this project, a 2-D camera is employed as the sorting system's sensory unit to refine the picking operation relative to the object's position and orientation on the workspace. The sensor identifies and distinguishes boxes on a flat surface based on color. The boxes are color-coded as red, blue, and yellow to be placed in their respective containers. Vision systems integrated with industrial machines surpass conventional machinery types due to their higher accuracy, repeatability, operational flexibility, and decision-making capabilities.

Robotic systems associated with vision have revolutionized production and assembling operations by performing sequential tasks instantaneously and accurately. An economically feasible sorting system consists of a fixed camera overlooking a flat table surface. The camera recognizes and differentiates between several objects and sends the information to the controller. The controller issues command signals to actuate the conveyor toward the respective containers. Such conventional sorting

systems can efficiently sort rigid objects in bulk. However, it can inflict damage to fragile objects such as plastic and glass. This project is oriented towards developing a sorting system using a robotic manipulator to handle objects safely. A robot's work volume refers to the entire collection of points in space that the robot can reach by actuating the joints. The selection of an appropriate manipulator relies on several characteristics and specifications of the arm, such as payload, reach, and repeatability. However, the robot's work volume can be a significant constraint when employed for a particular task.



Fig 1.1 Kinova JACO2 6 DoF robotic arm

The Kinova JACO2 6 DoF robotic manipulator is a lightweight and dextrous robot. It uses six degrees of freedom to execute complex and optimal trajectories to move the object while orienting it in space. Like industrial and assistive robot manipulators, the Kinova robotic manipulator comprises two sections: the arm and wrist assembly. The manipulator can be programmed using C++ language and is compatible with Windows, Linux, and ROS. Collaborative robotics is gradually expanding as a wide area of research in the current decade. Collaborative robots referred to as 'Cobots,' can work hand-in-hand alongside humans to perform specific tasks. It is presently extensively employed in industries to speed up production and component assembling processes and lower labor costs and time consumption.

Usually, Cobots are constrained to work with relatively lesser payloads compared to autonomous industrial robots. Consecutively, such robots lack speed and often demand high setup costs and maintenance.

Most assistive and industrial robots are presently compatible with ROS (Robotic Operating System) for autonomous control, collision avoidance, and trajectory tracing. ROS provides numerous packages to ease significant problems such as motion planning, collision avoidance, inverse kinematics, and 3-D perception. In this project, ROS forms the basic framework to develop a collision-free optimal motion plan for the Kinova JACO2 6 DoF manipulator arm to carry out pick and place operations. ROS manipulator packages such as MoveIt and TrackIK are state-of-the-art packages recommended for ROS-supported manipulators. These software libraries are utilized to provide the shortest collision-free trajectory to reach the goal state. Consecutively, it can handle heavy computations involving inverse kinematic equations and n-number of intermediate locations from the current state to the goal state. Similarly, Cartesian, angular, and finger control of the manipulator can also be established through the ROS interface by communicating ROS messages among the respective nodes.

Industry 4.0 incorporates several advanced segments such as IoT (Internet of Things), HMI (Human Machine Interface), cloud computing, autonomous robots, and data analysis within an intelligent factory. The primary motive of the current industrial revolution is to promote de-centralization and connectivity to improve production and quality. The subtractive manufacturing methods performed at large in Industry 3.0 are gradually replaced with an additive manufacturing method such as 3-D printing. The material wastage remaining after processing is vastly minimized in Industry 4.0, thereby contributing towards environmental safety and working conditions. IoT plays a vital role in the advancement of Industry 4.0 by improving connectivity among sensor and software embedded machines for monitoring and data transfer. This technology enables rapid prototyping and manufacturing to elevate production efficiency. IoT application is extended to be implemented in intelligent

warehouses where autonomous robots solely perform sorting operations in this project. A color-based sorting system is developed along with the implementation of IoT to log the count and color of sorted boxes in a spreadsheet. The data log obtained after the sorting operation is used to send auto-generated emails to the respective customers.

1.2 PROBLEM STATEMENT

- Traditional sorting operations lead to an increase in labor costs and time consumption. Human errors due to manual sorting lessen production efficiency in industries.
- Sorting operations performed solely by conveyors can inflict damage on fragile objects. In such cases, a robotic arm can be employed to pick and place objects safely.
- Sorting systems with camera setup relative to the tool frame (moving frame) demands extensive computation.
- Manipulators with an onboard camera mounted on the end-effector have to be re-centered to the home position to initiate the picking operation.
- Collaborative robots (Cobots) lack speed, require high setup costs, and are confined to operating with low payloads. In case of a malfunction, it may inflict injuries to a nearby human.

1.3 OBJECTIVES OF THE PROJECT

- This project aims to develop a color-based sorting system with a vision sensor and a 6 DoF robotic arm for industrial applications.
- To associate the object's center coordinates with the Cartesian system of the robotic arm.
- To apply inverse kinematics on the robotic arm using TRACK-IK inverse kinematics solver.
- To perform motion planning using the MoveIt Rviz plug-in to place the object in a respective container.

- To implement IoT for logging the sorted objects in spreadsheets and auto-generate emails for customers.

1.4 SCOPE OF THE PROJECT

- The vision-based sorting system can be employed in Amazon warehouses to sort packaged boxes based on color in different containers.
- The IoT application automatically keeps a count of the sorted boxes in a spreadsheet and sends auto-generated mails to the respective customers.
- The developed algorithm is made applicable to all ROS-supported articulated robot manipulators.

CHAPTER 2

LITERATURE SURVEY

2.1 INTEGRATION OF A ROBOT MANIPULATOR WITH VISION SYSTEM

The approach involved in robotic manipulator teleoperation using human hand gestures is demonstrated in this paper. The authors have proposed integrating the Kinova arm with a Microsoft Kinect 3-D depth camera [1] using the existing ROS libraries. The pose made by the human hand is recognized and approximated to a structure comprising links and joints. The joint angle formed by the human arm's gesture is visualized and interpreted by the Kinect depth camera to tele-operate the Kinova robotic arm.

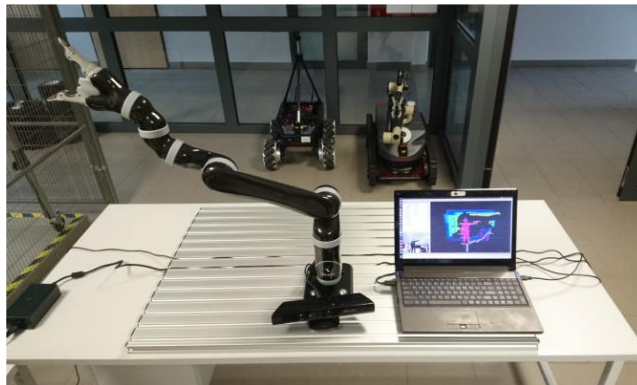


Fig 2.1 Experimental setup to mimic arm gestures

Manual teleoperation of the robot manipulator can be established using the manufacturer's joystick or a workstation running Windows or Linux Ubuntu. Fig 2.1 depicts the experimental setup consisting of the Kinova JACO robot arm, Microsoft Kinect 3-D depth camera, and a workstation running Linux. The overall weight of the arm with the two-fingered gripper is 5 kg. The maximum reach of the robot is approximately 68.5 cm. The Kinova API allows the user to program the actuation mode in two ways: Cartesian and joint. In Cartesian mode, the robot's frame is defined by the fixed reference frame's co-ordinate axes, i.e., x, y, and z. The orientation of the

end-effector remains fixed in this mode. The robot is actuated in Joint mode where each joint of the robot can be manually actuated To control the gripper explicitly.

The Microsoft Kinect 3-D depth camera consists of several sensory components such as a depth detection sensor, IR emitter, color sensor, and a tilt motor. The IR emitter emits the captured user's hand's infrared image and uses built-in software algorithms to transform the IR data into point cloud images. Teleoperation is established to control the arm via human arm gestures. The angles made by the shoulder and elbow of one hand are computed and are transformed to actuate the hip and shoulder joints of the Kinova arm using Joint mode. In the future, Cartesian control is mentioned to be established for manipulating the arm.

2.2 VISION-BASED ROBOT MANIPULATOR

In this paper, the authors have developed a color-based sorting system for industrial applications using machine vision. A real-time demonstration of robot-vision integration is presented by integrating the Scorbob-ER 9 Pro industrial manipulator with a vision sensor [2]. Fig 2.2 illustrates the slide action gripper with a parallel gripping mechanism constrained by the slider space. The drawbacks of conventional industrial vision-based robots with a fixed camera setup demands re-calibration when the work environment is modified. To overcome this significant drawback, the authors have integrated the vision system relative to the tool frame of the 2-fingered gripper, as shown in Fig 2.2.1. The industrial robot arm used for demonstration consists of 5 degrees of freedom to mimic human operations.

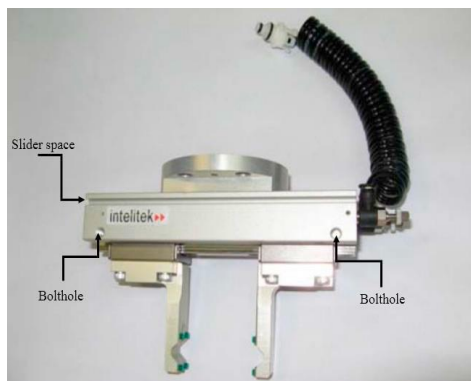


Fig 2.2 Slide action gripper

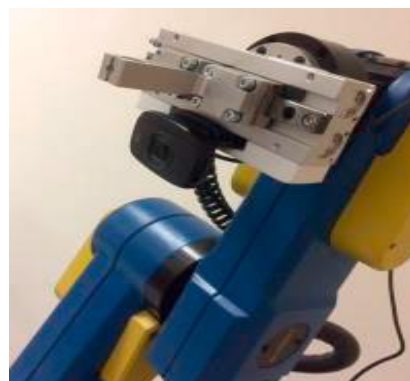


Fig 2.2.1 Mounted web camera

The vision-based approach for object recognition involves image thresholding and binary image conversion to capture the object of interest's average size and shape. The mode of robot actuation is electric and pneumatic; it consists of an electronic controller and a drive system.

The research was oriented towards developing a vision-based intelligent sorting system by integrating vision into the existing Scorbot manipulator. The camera mount is modeled in Solidworks and fabricated with a 3-D printer. The mounting location was selected right underneath the gripper to eliminate any fingers' hindrance during object perception, as shown in Fig 2.2.1. Image processing is carried out in Matlab to accommodate data acquisition and further processing. The objects' center coordinates are determined from the web camera while the objects get converted to grayscale. Fig 2.3.1 shows the binary images of the circular objects. The conversion is carried out to visualize the centroids of the circular objects.



Fig 2.3 Original image

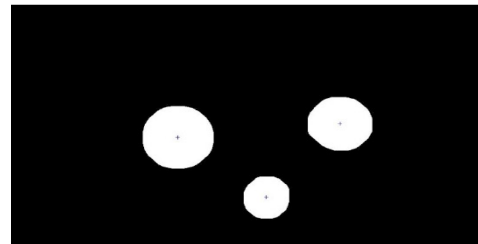


Fig 2.3.1 Binary image

The arm and gripper assembly is modeled, and a color-based sorting operation is simulated in Scorbace. Upon success, the experimentation was carried out in a real-time environment to pick and place circular objects using the slide action gripper. To summarize, image processing was carried out using Matlab for object recognition. The Scorbot-ER 9 Pro is interfaced with Matlab, and circular objects were picked and placed based on size and color.

2.3 COLOR SORTING USING A ROBOT MANIPULATOR

The drawbacks encountered in manual sorting are addressed to establish a vision-based robotic sorting system. A Phantom X Reactor custom robotic arm is integrated with a single RGB web camera to perform color-based sorting operations [3]. Camera

calibration is performed by capturing and recording the square size of a chessboard pattern. The focal lengths and optical center parameters are obtained in a matrix format to convert the pixel coordinates generated by the RGB camera to Cartesian coordinates corresponding to the fixed reference frame.

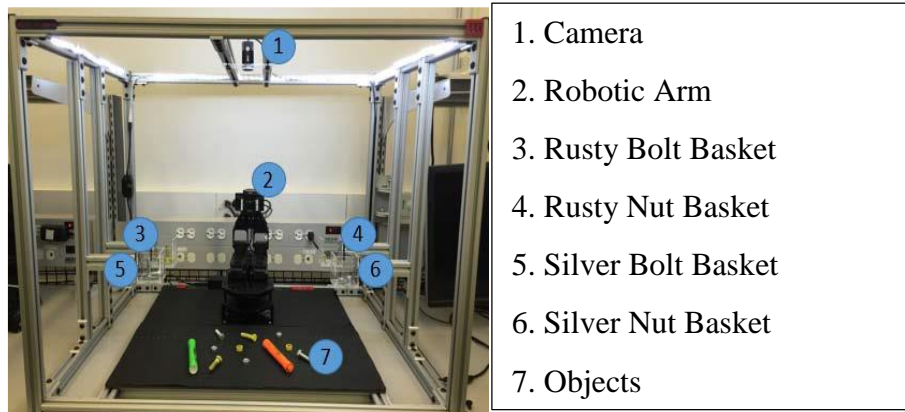


Fig 2.4 Experimental setup used for object sorting

The center position of the object placed on a 2-D plane is then calculated and utilized to determine the robotic arm's inverse kinematic solution to perform the pick and place operation. Fig 2.4 shows the experimental setup to perform the color-based sorting operation. The web camera is mounted above the robot arm to overlook the 2-D plane. The robot places the object in any one of the four baskets based on color.



Fig 2.4.1 Pick and place operation

Image processing involves segmentation and color detection. The flow of processes can be described as image acquisition, processing, localization, and serial communication. Fig 2.4.1 depicts the sorting operation undertaken in a real-time environment to drop a bolt in its respective basket. The background of the 2-D plane is kept dark to visualize and differentiate between the colors better. Furthermore, the

objects are differentiated based on size and shape after performing morphological operations to remove noise and improve images' texture. The inverse kinematic computation is carried out by the Arduino microcontroller using several input variables such as the object's centroid, orientation, size, and color. The joints of the 4 DoF robot are actuated using the inverse kinematic solution computed using the DH method. The sorting system is employed to distinguish between nuts and bolts of varying sizes and colors.

2.4 CONTROL OF ROBOT MANIPULATOR THROUGH ROS

In this paper, the authors have demonstrated ROS-based control of a Robai Cyton Gamma 300 robot integrated with a Sony Playstation Eye camera [4]. The advantages of utilizing ROS over other inadequate GUIs (Graphical User Interface) and APIs (Application Programming Interface) have been discussed. Presently, ROS is widely accepted as a software framework for programming the functionality of robots ranging from domestic household robots to complicated aerial and underwater robots. This paper's significant contribution is to establish a ROS-based software framework for the Robai Cyton Gamma 300 robot manipulator to perform a complicated task such as balancing a ball on a flat plate, as shown in Fig 2.5.

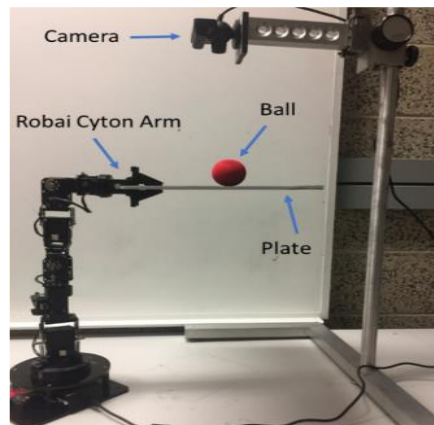


Fig 2.5 Experimental setup used for ball balancing

The 7 DoF robot manipulator uses 7 Dynamixel servo motors for joint actuation. The end-effector consists of a 2-fingered gripper to hold the flat plate for balancing the ball. The vision system is incorporated by mounting a Sony PlayStation

Eye camera onto an adjustable stand overlooking the 2-D surface of the plate at the height of 33 cm. The camera unit is capable of capturing framerates of video feeds from 60-120Hz. The USB camera captures the RGB image of the red ball shown in Fig 2.5 to initiate image processing. The images are converted to HSV color space, followed by grayscale conversion and thresholding to obtain binary images using the OpenCV image processing library. The software framework provided by the Roboi manufacturer was insufficient to fully exploit the functionalities of the 7 DoF manipulator to perform the ball-balancing task. Thus, the authors have developed a software architecture in ROS Indigo to segment the entire chunk of operations involved in the balancing task into several individual ROS nodes that can communicate with one another and used independently.

During experimentation, the manipulator is controlled in velocity mode using the system packages provided in the official ROS Wiki documentation. The ROS master is started to initiate the vision tracking node to obtain the ball's position coordinates on the plate. Simultaneously, the planning and control node is initiated to deduce the motor angle of the Dynamixel servos to calculate the required control velocity. The position and velocity parameters are implemented while initializing the manipulator system node to send velocity input to the servo motors.

2.5 SUMMARY OF LITERATURE SURVEY

Table 2.1 Summary of literature survey

S. No	Title of the paper	Authors	Publication	Inference
1.	Integration of vision system and robotic arm under ROS	Baranowski, L., W. Kaczmarek, J. Panasiuk, P.	2016 3rd International Conference on Systems and Informatics (ICSAI) 2016	This paper presents the integration of a 3-D Kinect vision sensor with a 6 DOF Kinova robotic arm using ROS [1]. Teleoperation of the robotic arm has been performed. The

		Prusaczyk, and K. Besseghieu r	Nov 19 (pp. 422-426). IEEE.	camera perceives depth information of the object as point-cloud images.
2.	Vision-based Robot Manipulator for Industrial Applications .	Md. Hazrat Ali*, Aizat K., Yerkhan K., Zhandos T. and Anuar O.	Procedia Computer Science, Volume 133, 2018.	A color sorting system is presented in this paper. An external vision sensor is integrated with the Scorbot-ER 9 Pro industrial robot's gripper assembly to perform autonomous color-sorting operations [2]. Image processing is carried out using Matlab. Sorting operation is performed in the RoboCell simulator and a real-time environment.
3.	Real-Time Color-Based Sorting Robotic Arm System	Yonghui Jia, Guojun Yang and Jafar Saniie	2017 IEEE International Conference on Electro Information Technology (EIT) 2017 May 14 (pp. 354-358). IEEE.	A camera module is mounted on top of the 5 DoF Phantom X Reactor robotic arm [6]. Image processing is carried out, and the object is detected based on color. ArbotiX-M microcontroller controls the servo motors to position the arm using inverse kinematics by the D-H method.
4.	Workpieces sorting	Xia K and Weng Z.	2016 3rd International	The working of an industrial sorting robot through machine

	system based on the industrial robot of machine vision		Conference on Systems and Informatics (ICSAI) Nov 19, 2016	vision is presented. The paper focuses on image processing using the Canny algorithm, Probabilistic Hough Transform, and Freeman code for experimenting with accuracy [3].
5.	Sorting System of Robot Based on Vision Detection	Qin Qin, Dongdong Zhu, Zimei Tu, and Jianxiong Hong	International Workshop of Advanced Manufacturing and Automation Sep 2017 Singapore.	The paper presents an industrial robot simulation that uses machine vision to recognize QR code labels [5]. LabVIEW is implemented for image acquisition, image processing, and coordination transformation.
6.	ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task	ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task	Advances in Robotics Research Volume 2, Number 2, June 2018	An experimental platform is developed and controlled by a 7 DOF robotic arm (Robai Cyton Gamma 300) in this paper using ROS Indigo [4]. The end-effector carries out a ball-balancing operation. The arm is actuated with 7 servo motors. A camera is used for acquisition, and OpenCV to process the images for tracking the object. Simulation and real-time results are obtained.

CHAPTER 3

COMPONENT DESCRIPTION

3.1 COMPONENT SELECTION

This chapter describes the selection and specifications of the components incorporated in the project. The sensory component selection is carried out by carefully considering the project's objectives and problem statements addressed in Chapter 1. A USB camera is mounted to overlook the workspace to perform color-based sorting to implement vision into the sorting setup. Due to the drawbacks of material damage seen in conveyor-based traditional sorting techniques, a robotic manipulator is employed in this project to carry out the sorting operation. Table 3.1 contains the list of hardware components and software used in this project.

Table 3.1 List of components

Hardware components	Software used
1. Kinova JACO ² robot manipulator	1. Robot Operating System
2. USB camera	2. Camera configuration Utility
3. Adjustable camera mount	3. Gazebo 3-D robot simulator
	4. MoveIt Motion Planning framework
	5. Rviz 3-D robot visualisation tool

3.2 HARDWARE COMPONENTS

The hardware setup consists of a Kinova JACO² robot manipulator interfaced with a USB camera mounted on an adjustable stand overlooking the work space. The components shown in Table 3.1 are integrated to carry out the autonomous color-

based sorting operation in a controlled work environment. The following sub-sections explain the functions and specifications of the hardware components in detail.

3.2.1 KINOVA JACO² 6 DOF ROBOT MANIPULATOR

Kinova manufactures modular robotic systems involving assistive robots to aid physically challenged people and service robots for medical and extensive research purposes. The company is based in the United States, and it also manufactures grippers, actuators, and controllers associated with the robot manipulators. The manipulators produced by Kinova Robotics are dextrous, lightweight, and have a high payload to weight ratio. At present, the JACO and MICO manipulators are widely used for research in trending technology aspects such as artificial intelligence and machine vision. The manipulator is compatible with WindowsOS and UbuntuOS and can thus be configured according to the user's needs using the existing libraries. Both models use stepper motors for joint actuation and are available in 4 DoF and 6 DoF manipulators [8]. Similarly, the controller position is located at the base of both models and the USB interface and power supply for better compactness.



Fig 3.1 Kinova JACO² 6 DoF robotic manipulator

In this project, a 6 DoF JACO² robot arm with a three-fingered gripper is employed to carry out pick and place operations. The manufacturer recommended stock firmware JACOSOFT can perform joint and torque control for basic trajectories.

However, the existing APIs and GUIs in ROS were used in this project for optimal motion planning and collision avoidance.



Fig 3.1.1 Base panel with power supply and interface

Fig 3.1.1 shows the base panel of the JACO2 robotic arm. The USB interface with a machine running Windows or Linux and the joystick interface for manual control are depicted. The power supply wire is connected across the right below the power switch. The JACO2 manipulator has 6 actuators, as seen in Table 3.2, to control all six joints independently and the maximum reach of the arm is approximately 90 cm.

Table 3.2 Various segments of the manipulator

PART ID	PART NAME
1	Fixed base
2	Actuator 1 (KA-75+)
3	Base
4	Actuator 2 (KA-75+)
5	Arm
6	Actuator 3 (KA-75+)
7	Forearm
8	Actuator 4 (KA-58)
9	Wrist 1
10	Actuator 5 (KA-58)
11	Wrist 2
12	Actuator 6 (KA-58)
13	End-effector
14	Fingers

The arm can carry a maximum payload of 2.6 kg at mid-range without the gripper. The JACO model significantly exceeds MICO in terms of load-to-weight ratio, reach, and payload. The entire assembly is made out of carbon fiber to withstand the payload and for durability. The actuators can move the links at a velocity of 20 cms^{-1} . The average power required by the unit during operation is 25 watts. Position sensors are embedded within the robot manipulator to detect the absolute position and torque generated at every joint to provide precise actuation and tracing complicated trajectories. The robot's base resides the tri-axial accelerometer, power supply port, gripper interface port, and USB interface port for serial communication. The wrist assembly uses optical position encoders for precise object orientation. The actuators used in the JACO2 variant are brushless DC motors. Actuators 1, 2, and 3 embedded in the arm along the base, as shown in Table 3.2, use KA-75+ motors to provide 12 Nm torque for heavy-duty operation. However, joints 4, 5, and 6 use K-58 motors for lesser torque and better positional control. Table 3.2 summarizes the technical specifications of the JACO2 6 DoF robot manipulator.

Table 3.3 Specifications of the JACO² Kinova arm

S.No	Category	Specification	
1.	General	Operating temperature range:	-10°C to 40°C
		Build material:	Carbon fiber (links), Aluminium (actuators)
		Number of joints:	6
		End-effector used:	3-fingered gripper
2.	Mechanical	Overall weight:	5.2 kg (3-fingered gripper) 4.4 kg (without gripper) 5 kg (2-fingered gripper)
		Maximum reach:	90 cm
		Maximum linear arm speed:	20 cms^{-1}
		Actuators:	KA-75+, KA-58

		No-load speed (K-75+):	12.3 RPM
		Torque (K-75+):	12 Nm
		Peak torque (K-75+):	30.5 Nm
		Actuator weight (K-75+):	570 grams
		No-load speed (KA-58):	20.3 RPM
		Torque (KA-58):	3.6 Nm
		Peak torque (KA-58):	6.8 Nm
3.	Electrical	Supply voltage:	18-29 V (DC)
		Peak power:	100 W
		Operating power:	25 W
		Standby power:	5 W
		Communication protocol:	RS-485
		Joystick controller:	1 Mbps
		USB interface:	12 Mbps
4.	Software	Control:	Cartesian, Force, Angular
		Compatibility:	WindowsOS, UbuntuOS, ROS
		Programming API:	C++
		Vendor provided firmware:	Jacosoft

3.2.2 USB CAMERA

The USB camera consists of an on-board dual-core image processor and an image sensor. The 2-mega pixel camera can recognize and track objects by detecting the color signatures associated with blob detection. The unit is compatible with Arduino and RaspberryPi and supports USB, UART, SPI, I2C interfaces. It can recognize and store up to 7 distinct color signatures such as red, orange, yellow, green, cyan (light blue), blue, and violet. The overall camera assembly dimensions are (42x 38x 15) mm, weighing up to 70 grams. The camera comes with an IDC (Insulation

Displacement Conductor) cable that makes it easier for users to interface the camera with popular microcontroller boards such as Arduino UNO and Mega.



Fig 3.2 USB camera

The module can capture raw and processed video feeds at high frame rates with a 5V power supply. The camera lens has a 60° field of view horizontally and a 40° vertical field of view. The viewing angle is crucial to calculate the camera coverage to determine the work area along the length and breadth of the 2-D plane. The object can be manually trained with a signature associated with its color using the push button. However, training objects through the GUI is much practical due to custom camera configuration. The video feed is paused during training as the user defines the object's boundaries to be recognized. The camera unit provides live video output and the moving object's coordinates in x and y axes. Consequently, the dimensions of the bounding box encompassing the object are also displayed. The following section describes the 3-D printed camera enclosure and an adjustable camera mount to position the camera above the workspace.

3.2.3 ADJUSTABLE CAMERA MOUNT

Object recognition and tracking require a perception module to capture the entire work environment. The module can either be mounted on the end-effector or an adjustable stand as a stationary setup. Both configurations have pros and cons associated with them and are highly dependent on the task and the work area they are employed. In the case of the camera mounted on the end-effector, it is cost-effective and uses a much smaller workspace. However, if the camera is associated with the

manipulator's tool frame, it demands heavy kinematic computations as the frame moves at every instant while tracing the trajectory.

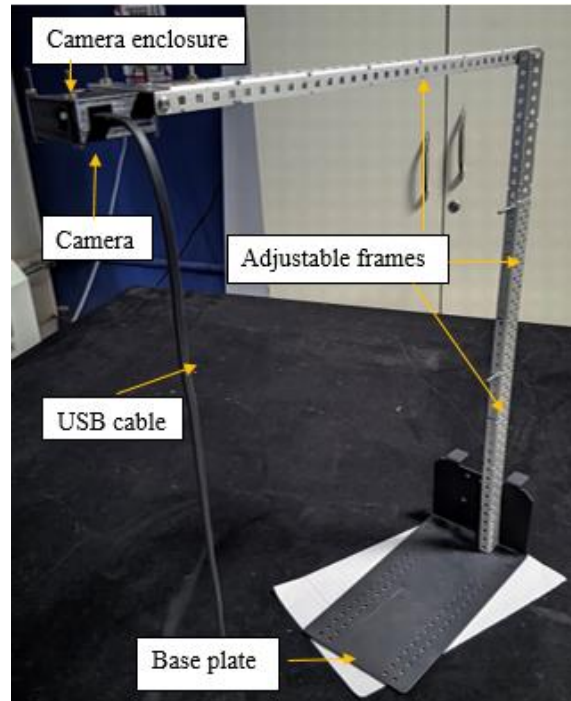


Fig 3.3 Adjustable camera mount

The sorting operation is carried out with a low-cost stationary camera setup overlooking the work area for a fixed reference frame. The custom-made mount was incapable of holding the camera relatively flat to the surface of the work area. A separate camera enclosure had to be mounted to resolve proper fitting to ensure that the camera is entirely parallel to the work area's surface. An additional enclosure is modeled, and 3-D printed to serve as a perfect fit for the bottom counterpart to ensure that the camera is horizontal to the work area.

3.3 SOFTWARE USED

3.3.1 ROBOT OPERATING SYSTEM

ROS is an open-source software framework that runs on Linux-based Operating Systems such as Ubuntu. ROS can perform many operations such as task scheduling, monitoring, and real-time visualization of robot simulation. Currently, ROS is regarded as the apex of robot programming frameworks due to its compatibility with

a wide range of robots, and the vendor's functionalities do not constrain it. ROS consists of several layers of components to establish a fully functional API for robot programming. The client library supports the execution of multiple programming languages such as C++, Python, and JAVA. Since its inception, ROS has a substantial user community and developers to make extensive improvements with every version to bring about a better GUI and API for its users. The catkin workspace paves the base framework for the ROS build system. The workspace consists of Cmake and .xml files to initiate the functionality of robots. The CMake macro file contains the list of messages, services, and action files. ROS's existing interfaces provide control over robot hardware for teleoperation and autonomous control by receiving and transmitting data using ROS nodes.

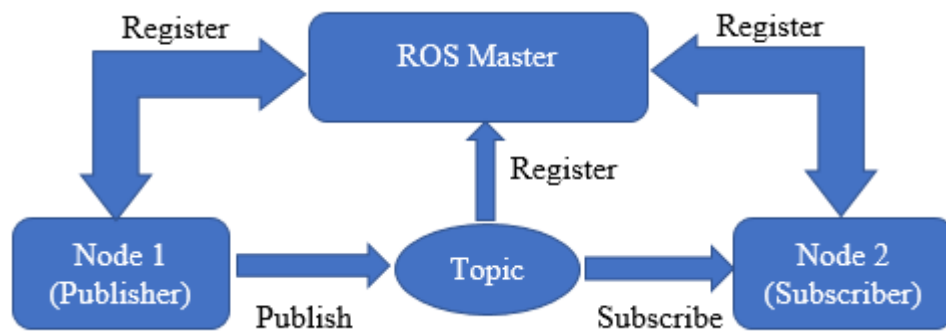


Fig 3.4 Node-to node communication

Other application frameworks include the MoveIt motion planner and Rviz visualization tool. Fig 3.4 represents a publisher-subscriber form of node-to-node communication in ROS [7]. Communication can be established between two or more nodes that control the hardware components; messages, topics, and services are available. ROS supports the installation of countless hardware drivers through the terminal for better compatibility during operation.

The meta-OS provides a customizable simulation environment such as the well-known 'Gazebo' simulator and other software development tools such as Rviz and catkin. ROS is compatible with numerous IDEs (Integrated Development Environment); the commonly used programming tool Visual Studio Code is utilized in this project for programming the manipulator control. The ROS master server is a

centralized source to establish a connection between several ROS nodes for message communication. ROS nodes represent executable programs such as C++ and Python scripts. Nodes are widely incorporated in mobile robots to organize every individual functionality that the robot must perform. Functions such as acquiring input, conversion of sensor data, obstacle detection, and motor actuation are valuable mentions. ROS nodes can be configured to act as publishers or subscribers to send and receive information from other nodes, respectively.

A publisher node transmits messages and waits until the ROS master establishes a connection with a corresponding subscriber node. However, the subscriber node performs the opposite function by receiving the publisher node's information through ROS topics. Consequently, the information gets transmitted as messages via topics, services, parameters, and actions. A ROS topic acts as a bridge between node-to-node communication, similar to the analogy of ROS master [7]. However, topics form a direct connection between two nodes and allow a passage for transmitted or received information.

On the other hand, services can communicate bi-directionally to terminate the process and save sensor configuration when another operation runs simultaneously. Fig 3.4.1 shows the server-client form of node communication in ROS. Node 1 sends a request message to the server, which redirects the request to Node 2. The service server in Node 2 sends back a message response to the service client. Similarly, several service client nodes can be registered to the same service server node.

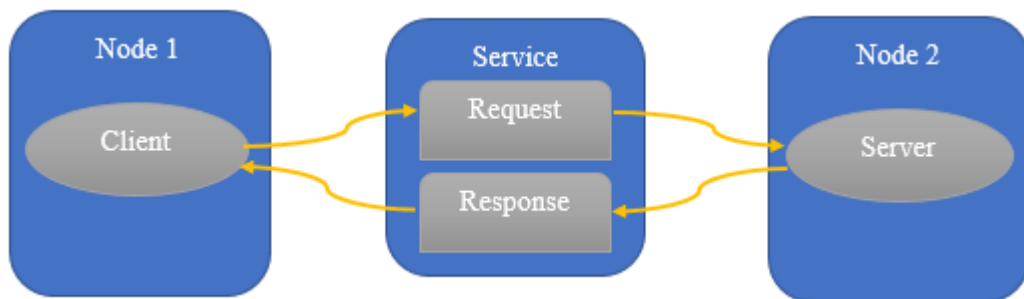


Fig 3.4.1 Server-Client communication in ROS

In addition to applicability among a wide range of mobile robots, ROS incorporates convenient software file formats such as URDF (Unified Robot

Description Format) and XML (Extensible Markup Language) to visualize robot manipulators during model a robot from scratch and visualize its operation in the Rviz tool. ROS is also supported with popular development boards such as Arduino and Linux-based computers such as Raspberry Pi. By establishing a URDF, the user can fully customize the joints and links and view the same in Rviz by initializing the program through a launch file.

3.3.2 GAZEBO 3-D ROBOT SIMULATOR

The gazebo is a commonly utilized robot simulator platform by the users of ROS. Ever since the inception of ROS, the simulator was compatible with the meta-OS as the developers own the community. The gazebo is an open-source robot simulator running in ROS to provide optimized and realistic simulations of robots and environments. At present, various robot simulators have 3-D graphics embedded as default. However, Gazebo's rendering engines provide a realistic experience to the users to incorporate lighting, shadow effects, and texture modifications similar to real-time environments.

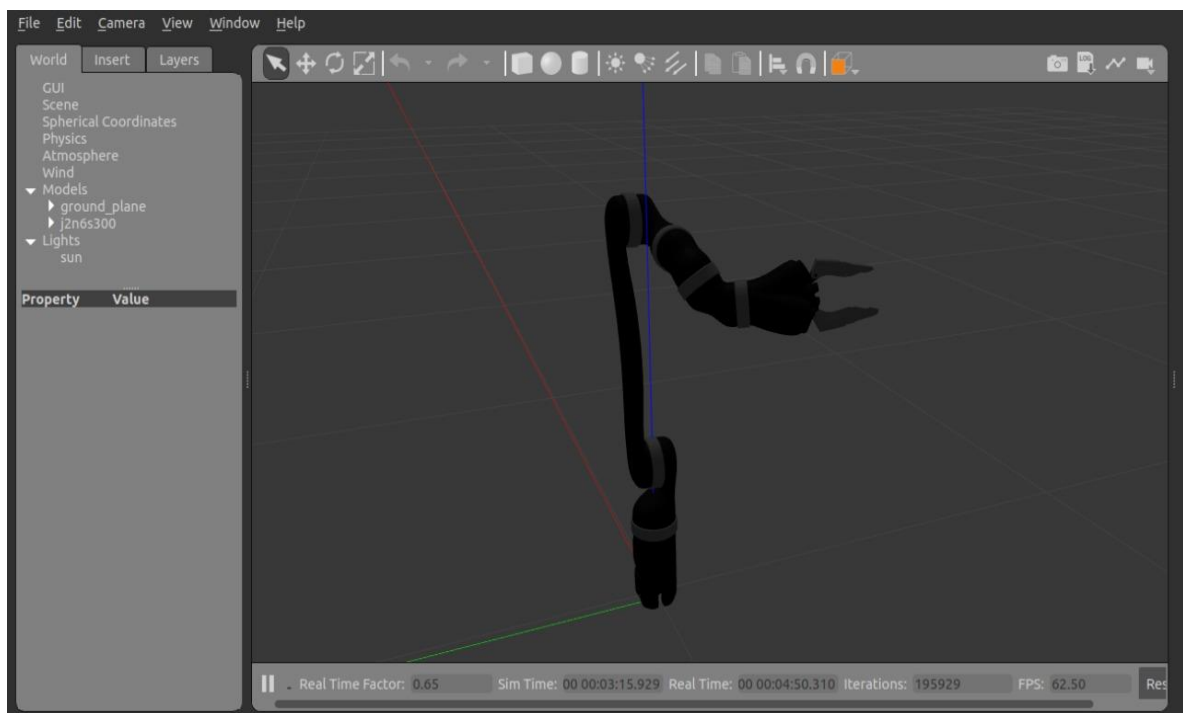


Fig 3.5 Kinova manipulator imported in Gazebo GUI

The Gazebo simulator shown in Fig 3.5 has built-in physics engines since version 1.9. Currently, the DART engine runs along with other physical engines in the latest version to simulate complex robots and environments. A significant advantage of this platform is to control the range and influence of sensors on the robot's operation and the preferences given to users for adding external noise into the environment to simulate the real-time performance of robots. The Gazebo GUI enables the users to modify the environment and add several sensory and actuator components. The simulator also consists of standard robot simulation models such as Turtlebot and industrial manipulators to provide better user exposure. ROS users are independent to execute robot models in a defined virtual environment through executable scripts as well. The surrounding environment, obstacles, and goals for the robot can be customized in the GUI. The simulator presents various viewing modes such as wireframe, joints, and center of mass/inertia. Mobile robots can be generated by defining their URDF parameters can be tele-operated using a twist keyboard in the terminal window and publishing to a topic such as `cmd vel` to configure the linear and angular parameters (pose) of the robot. The following section elucidates Rviz, ROS's visualization tool, where the robot's behaviour in real-time environments can be inferred.

3.3.3 RVIZ 3-D VISUALIZATION TOOL

Rviz is a built-in 3-D robot visualization tool in ROS. As programmers feed and receive data between two or more nodes, it tends to be cumbersome to analyze terminal outputs' operation outcomes. However, ROS has eased this problem by providing a 3-D visualization tool to view the robot's movements in an environment. Using rviz, one can visualize the sensor output data and point cloud generation in real-time environments simultaneously in autonomous mobile robots. The rviz tool can quickly be initiated in ROS by running the 'roscore' command in a terminal window and executing the rviz command. The grid portion depicts a 3-D window where all sensor output data and robot models can be visualized [7]. The display section shows the entity to be displayed from various ROS topics. The global frame

topic is displayed as a default to define the robot's relative position on the 2-D plane. The menu panel resides above the display section where robot configuration files can be imported and preferences can be set. The tools manager displays the shortcut tools to control the robots' position, orientation, and movement.

Additionally, the pose estimate tools aid the robot in navigation by setting the start and goal states. The GUI also presents several viewpoints such as first-person, third-person, orthographic, et cetera. About robot manipulators, the URDF model of the robot can be imported into rviz to visualize the functioning of joints and links of the robot.

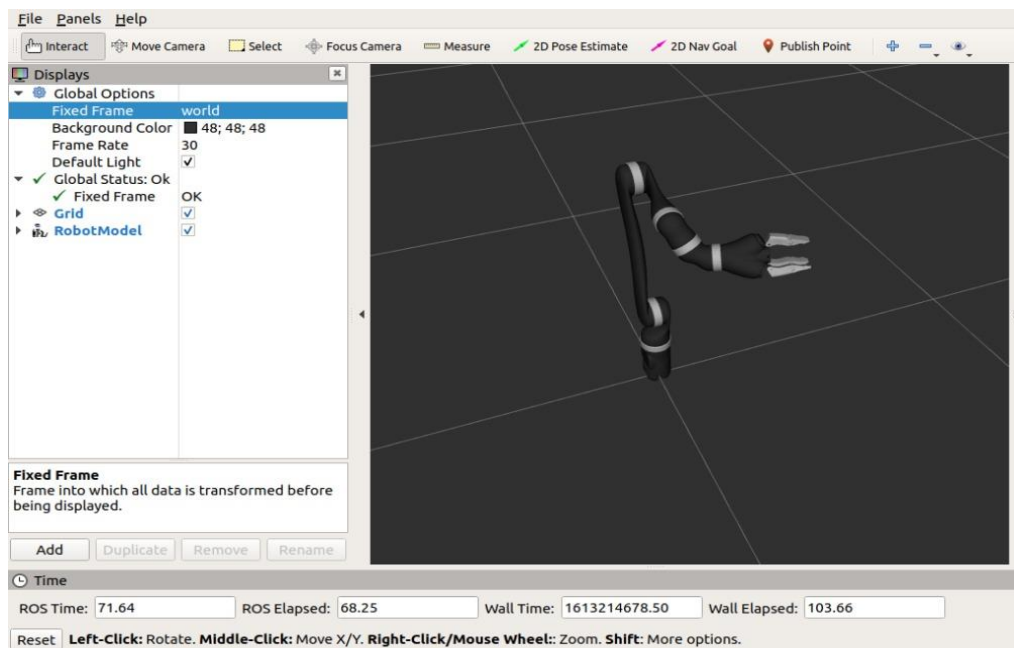


Fig 3.6 Robot manipulator in Rviz GUI

The robot's coordinate transformation is also displayed in color-coded XYZ axes using the transformation (TF) icon. In this project, the rviz tool shown in Fig 3.6 is interfaced with a motion planning plug-in called MoveIt to incorporate obstacle avoidance and observe the motion plan traced by the arm in a real-time environment. The MoveIt rviz-plug-in utilizes the resources of rviz to display the JACO arm in an environment defined by the user. The following section shows the MoveIt framework's efficacy in incorporating motion planning and collision avoidance in ROS-supported robot manipulators.

3.3.4 MOVEIT MOTION PLANNING FRAMEWORK

MoveIt is a state-of-the-art Rviz plug-in extensively used to offer numerous functions for robotic manipulators. The framework eases heavy computations involving inverse kinematics and advanced algorithm implementation. The framework has been employed for mobile manipulation in the NASA agency and technology giants such as Google and Microsoft. The GUI presents us with interactive markers for end-effector pose control. It consists of a triad (XYZ axes) for translation and three color-coded rings for rotation. The robot motion simulated by MoveIt can be simultaneously experimented with and developed in a physics-based simulator such as Gazebo. This library uses a centralized node "move group" to communicate with the sensory components, controllers, parameter server, and ROS's general user interface. The ROS parameter server contains all the information to be shared among the nodes in general. Besides, ROS nodes can add and modify the data stored in the parameter server as well. It is popular because, unlike other integrated robot manipulator utility tools, MoveIt provides C++, Python APIs, and a Rviz supported GUI.

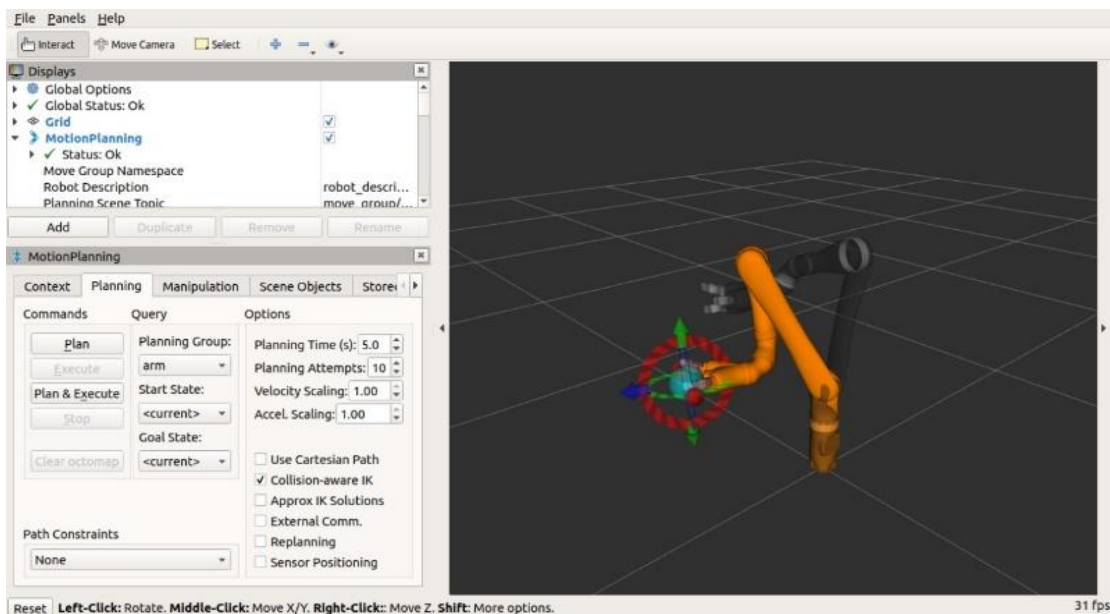


Fig 3.6.1 MoveIt Rviz plug-in

The JACO² robot manipulator model is launched in the MoveIt-Rviz plug-in, as shown in Fig 3.6.1. It illustrates the menu screen of the MoveIt Rviz plug-in under

the motion planning section. The planning groups (arm and wrist) are selected and interacted by specifying the arm's start and goal state using this utility. The interface provides collision avoidance and an option to enable Cartesian trajectory during motion planning. Interactive markers are provided to position the end-effector precisely in XYZ axes and the desired orientation.

Furthermore, MoveIt provides standardized icons such as start state, goal state, planning, and execution tabs. During trajectory execution, the velocity and acceleration parameters can be scaled to lower or quicken the planning routine. The arm's greyed-out model represents the current state, whereas the orange model depicts the goal state to be reached by the arm. This way, users can visualize the operation of the manipulator in virtual and real-time environments. Fig 3.7 shows the centralized communication established by the move group node between other multiple nodes. The robot's configuration consisting of the type and number of joints and links is provided to the MoveIt library using the server's URDF file. The robot sensors (torque and position) send messages containing robot joints' position through the joint state topic. The user interface provided by MoveIt contains APIs and the widely used Moveit Rviz GUI plug-in to initiate the planning scene and carry out the mathematical computation. By default, the GUI provides pre-loaded poses of the arm such as vertical, home, retracted, current, and previous positions.

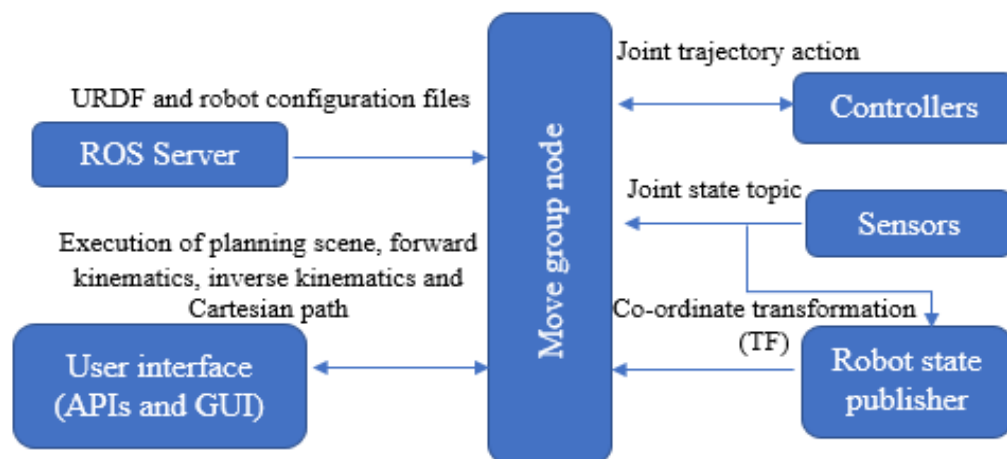


Fig 3.7 Centralized communication of move group node

In the default Rviz scene, the greyed-out model perfectly coincides with the orange model indicating that the arm has completed the motion plan and the goal state becomes its current state. The robot controller (joint trajectory controller) is enabled by default for the MoveIt framework to command the robot. It is used to execute joint trajectories by controlling a set of joints. The trajectory of a robot manipulator is a group of numerous waypoints or positions to be reached by the arm at a given time instant. The controller execution is carried out by configuring the position, velocity, and acceleration parameters to send trajectories. The API uses an action interface to send and receive information of the robot's state via the move group node to notify the robot's status. It is subscribed to the joint trajectory topic and publishes the controller's state to the controller state topic.

The robot state publisher node is subscribed to the joint state topic to receive information of the robot's joints to publish the coordinate transformation to the move group node. The MoveIt setup assistant allows users to model robot manipulators from scratch using the URDF format. In this format, users can define the material, color, number, and type of individual joints and links desired in the robot. During the setup, the user is prompted to upload the URDF file for further customized configuration. Furthermore, MoveIt creates additional YAML, XML, and SRDF files to simulate the model in Gazebo and visualize the same using the Rviz tool. The planning groups are defined as arm and gripper; the former configures the motion of joints and the robot's links, whereas the latter configures the end-effector's motion. The type of robot controller is selected as a joint trajectory controller for the arm and wrist assembly. In case of obstacle collision and self-collision detection, MoveIt warns the user by highlighting the collision area in red color and subsequently fails to execute the motion plan.

3.3.5 CAMERA CONFIGURATION UTILITY

The camera module can be configured to recognize and track objects based on color, line tracking, pan and tilt mechanism of servos, barcode reading, and marker recognition. It is compatible with Windows, Mac, and few Linux-based operating

systems such as Mint and Ubuntu. The camera's functionality can be programmed using Python, C, and C++ languages. In case of lack/excess lighting, the application triggers the LED lamp to be toggled on or off. The application allows the user to view live and processed video feed and encapsulates blocks/pixels from the background. In this project, the camera is interfaced with ROS and trained to sort boxes based on color to distinguish between red, blue, and yellow.

The object's coordinates in pixel notation are further converted to Cartesian position coordinates to actuate the arm. The program icon allows the user to select four modes of operation such as color detection and tracking, line tracking, pan and tilt mechanism, and QR code recognition. A color-connected component program is utilized to differentiate and track red, yellow, and blue colors individually. The program allows the user to select and store 7 color signatures. The CC signature option stores an object having dual colors to identify its orientation (Φ) on a 2-D plane. The camera is trained by capturing the object of interest under sufficient lighting conditions. The video feed containing the object is paused, and the user selects and defines the boundaries of the object to be tracked by the camera. A familiarized signature intensity is elevated by adjusting the slider bar provided in the GUI to avoid flickering during tracking. The configuration panel contains slider bar settings to modify the camera brightness and other settings under the advanced menu to vary white balance and auto-exposure. The object's video is captured and displayed at a rate of 60 frames per second for efficient and lag-free object tracking, which is ideal for the project. In the next chapter, the selected components and software are interfaced to establish the robot manipulator's autonomous control.

CHAPTER 4

METHODOLOGY AND EXPERIMENTAL SETUP

4.1 INTERFACING GAZEBO WITH MOVEIT

In the previous section, using a physics-based simulator such as Gazebo, the robotic arm's trajectory execution is tested in a virtual environment [7]. The performance of the developed code is ensured, and the actual Kinova robot manipulator is interfaced with ROS via USB cable to load the controller program for motion planning. The significant difference in trajectory execution between both platforms is that MoveIt uses virtual arm controllers and virtual gripper controllers for trajectory execution and visualization. For real-time motion planning, a joint-space trajectory controller is required. ROS serves as an interface to establish communication with the Kinova robot manipulator. Before testing and validating the real-time robotic arm's performance, it is safer to ensure the robot's performance via simulation to prevent unexpected damages to the robot during operation.

Firstly, the controller configuration file for MoveIt is defined to communicate with Gazebo's trajectory controller. The configuration file must consist of the arm and gripper interfaces. After establishing the trajectory controller, an action interface is provided for the arm and gripper to communicate during the simulation, as shown in Fig 4.1. In the MoveIt directory configuration folder, a YAML file is created to define the controller topic used in Gazebo and the robot's joints from the URDF. In this case, it refers to the action server "follow the joint trajectory." Similarly, the gripper portion is also defined with the finger variable names in the URDF.

The next step involves creating the YAML file that defines all the joints controlled by the robot controller. The third step involves creating a launch file that loads the controller.YAML file as well as the robot's joint names in the parameter server. The topic where the robot states get published in the 'joint states' topic. The particular topic is included in the launch file to send the joint information to the

simulator. Fig 4.1.1 (on the right) shows the robot model with interactive markers in the MoveIt rviz platform; the Gazebo simulator is shown on the right.

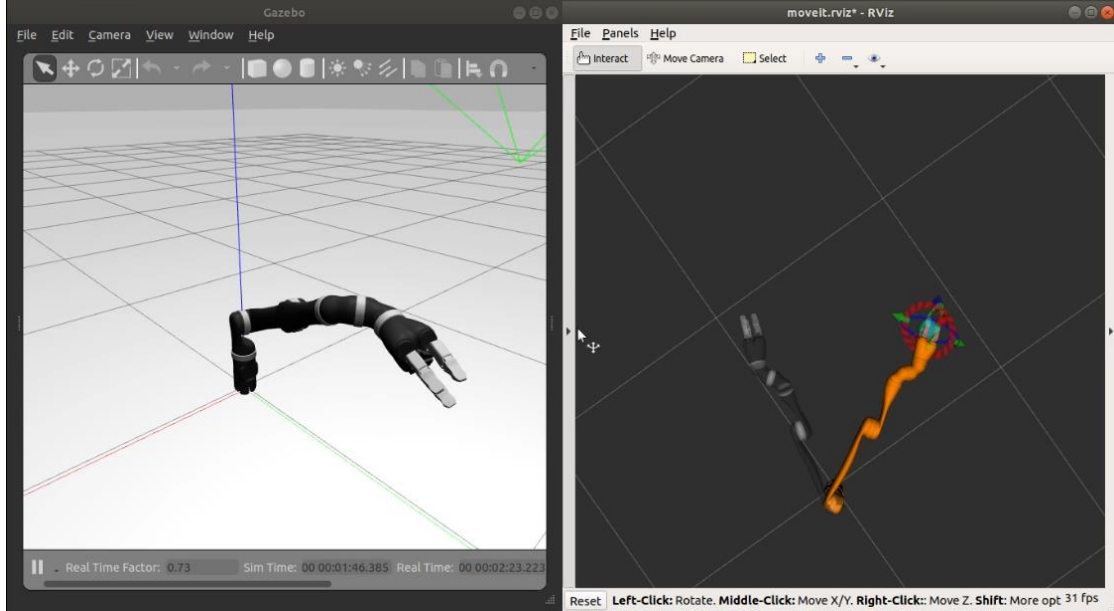


Fig 4.1 MoveIt-Gazebo Interface

The joint trajectory execution of the robot arm through MoveIt has been described in the preceding section. Although the robot arm model constructed from URDF can execute joint-space trajectories in the virtual environment generated by Rviz, it is inadequate to deduce the robot's performance in real-time environments. Thereby, the operations performed in MoveIt are tested in a physics-based simulator such as Gazebo. The section describes the interfacing of the MoveIt framework with the Gazebo 3-D robot simulator. The pose defined by the user is executed with MoveIt controllers and sent to Gazebo via an action server. Upon, trajectory execution the same path traced in MoveIt is visible in Gazebo, as shown in later sections.

4.2 SETTING UP THE WORK ENVIRONMENT

This section elucidates the mechanical mounting of the Kinova robot manipulator and the positioning of the camera mount in the work environment. The mounting kit provided with the manipulator consists of a mounting plate and mounting post to provide rigid support to the robot manipulator on a flat table surface.

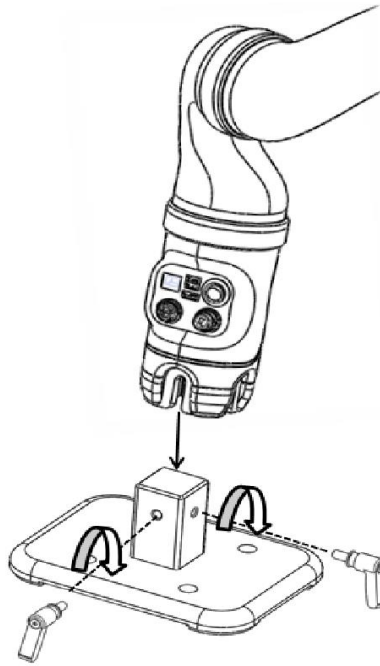


Fig 4.2 Mechanical mounting of the robot manipulator

The mounting post is inserted into the manipulator base rigidly fixed with M8 screws on the post, as shown in Fig 4.2. The mounting post is screwed to the base plate's square cavity with an 8mm hex key. The assembled mounting kit is firmly clamped to the table surface using a pair of IRWIN quick grip clamps. The camera mount is positioned directly in front of the robot arm such that the center of the robot base aligns with that of the mount's base plate, as shown in Fig 4.2.



Fig 4.2.1 Experimental setup

The camera is located at the height of 56 cm from the table surface and placed at 45 cm away from the robot's mounting plate. The spacing ensures efficient installation of a conveyor system between the base and camera mount. In the experimental setup, a yellow-colored box is placed at the origin of the robot's tool frame. The box is detected by the USB camera, as shown in the development computer's camera configuration utility. The box's location relative to the camera frame is obtained as image coordinates in pixel notation. However, the coordinates must be converted into the Cartesian system for the manipulator to interpret the location for picking the box. Thus, the coordinate frame conversion to represent the image coordinates relative to the tool frame is described in the next section.

4.3 CORRELATING THE CAMERA FRAME WITH TOOL FRAME

Generally, there are five commonly used coordinate frames to define a set of locations in space. The world frame or global reference frame is located at an arbitrary point, neither a part of the robot nor the vicinity object. However, the position of all objects in space can be determined relative to the fixed world frame. The robot's frame or base frame is located at the base of the robot. It is a commonly used frame in robotics because it is a fixed reference frame relative to which other entities' descriptions can be made at ease by absolute coordinate representation.



Fig 4.3 Tool frame convention

Fig 4.3 shows the convention followed to define the x-y axes of the tool frame. When the user faces the supply panel, the left direction denotes the +x axis. The tool frame corresponds to the robotic arm's gripper portion, where the Cartesian coordinates are represented in meters. The tool frame is located at the tip of the end-effector, in this case, a three-fingered gripper. In this project, the tool frame is set as a reference frame w.r.t which the object's position is determined. Let us consider two coordinate frames in the overall sorting setup to execute the picking operation. The camera frame corresponds to the image coordinate system of the perception module (pixel convention).

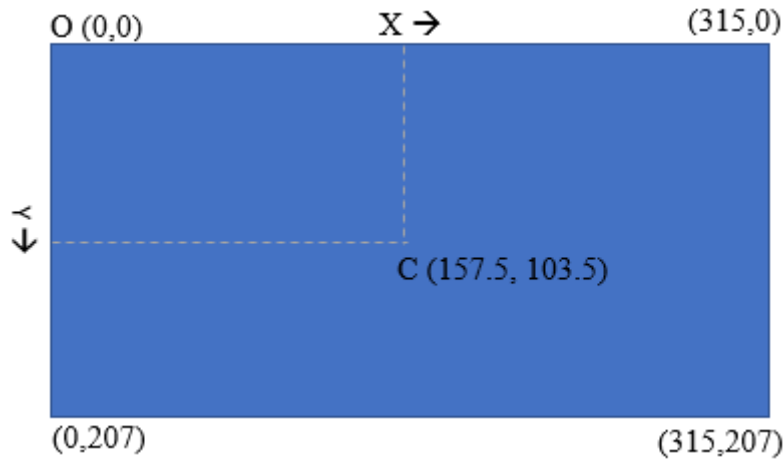


Fig 4.4 Camera frame convention

Conventionally, the origin of the camera frame is located at the top left corner of the frame. The +x axis direction is along the right side of the frame, and the +y axis runs along the downward direction, as shown in Fig. The camera frame must be converted to a tool frame for the arm to identify the object's location (box) w.r.t the gripper on a 2-D plane. In addition to frame translation and rotation, the pixel notation must be converted to meters for Cartesian position control. Co-ordinate frame transformation is a commonly used operation to attain descriptions from a specific frame of interest. For instance, if an object's location is known relative to the base frame, the same can be computed relative to the base frame through frame transformation. To carry out the picking task, we convert the camera frame to a tool frame using a logical approach by applying mathematical operations on the camera frame. As illustrated in Fig 4.4.1,

our objective is to merge the origin of camera frame C (0,0) to point C (157.5, 103.5) such that it coincides with the origin of tool frame T (0,0).

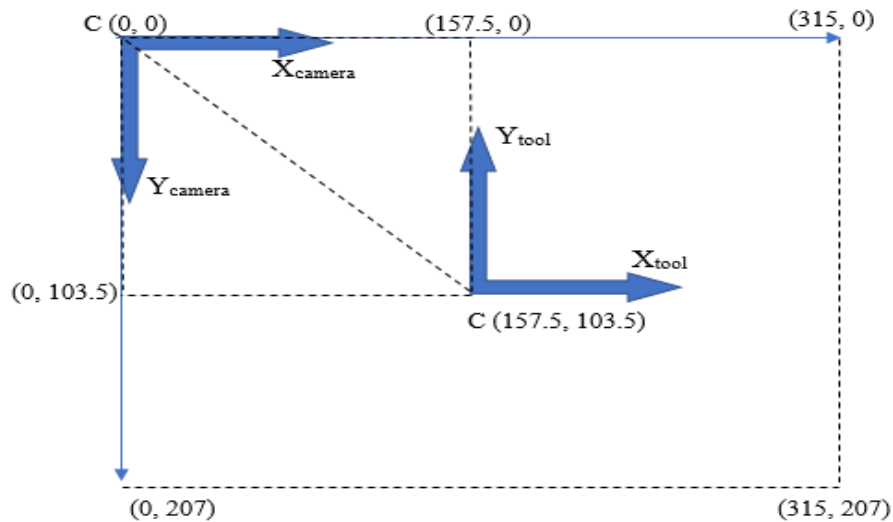


Fig 4.4.1 Camera frame and tool frame association

Our approach aids in a much simpler frame association compared to the conventional frame transformation methods as the pixel coordinate values are available to offset the camera frame's origin to the tool frame's origin. Fig. 4.4.1 illustrates the association between camera frame and tool frame through pixel coordinates. The calculations involved in the camera frame to tool frame conversion are shown as follows:

Maximum value of x-axis in the camera frame = 315 pixels

Maximum value of y-axis in the camera frame = 207 pixels

Center co-ordinates C (X_0 , Y_0) of camera frame is calculated from dividing the maximum value of pixels along both axes by 2, as shown below:

$$X_0 = 315 / 2 = 157.5 \text{ pixels}$$

$$Y_0 = 207 / 2 = 103.5 \text{ pixels}$$

The origin of the camera frame is then offset from point C (0, 0) to C (157.5, 103.5) to merge with the origin point of the tool frame T (0,0):

$$\mathbf{X}_{\text{new}} = \mathbf{X}_{\text{input}} - \mathbf{X}_0 \quad (4.1)$$

$$\mathbf{Y}_{\text{new}} = \mathbf{Y}_{\text{input}} - \mathbf{Y}_0 \quad (4.2)$$

Pixel resolution is the least pixel value along x and y axes generated by the USB camera to be converted into Cartesian co-ordinates. The maximum reach of the arm (in meters) is divided with the equivalent pixel resolution along x and y axes respectively:

$$\text{X-axis: } 1 \text{ unit} = \text{Maximum reach along x} / X_0 = 0.25 / 157.5 = 0.0015\text{m}$$

$$\text{Y-axis: } 1 \text{ unit} = \text{Maximum reach along y} / Y_0 = 0.12 / 103.5 = 0.0011\text{m}$$

Accordingly, the pixel values generated by the USB camera representing the object's location relative to the tool frame is multiplied with the pixel resolution and converted to the metric system. From equations (4.1) and (4.2), we can deduce the object's location as follows:

$$X_1 = X_{\text{new}} * 0.0015 \text{ (in meters)} \quad (4.3)$$

In Fig 4.4.1, the x-axis of the camera frame and tool frame are oriented in the same direction. However, from the right-hand thumb rule, in the camera frame, the z-axis points into the page, whereas in the tool frame, the z-axis points upward from the page. Due to this orientation, the y-axis is misaligned in both frames. A factor of -1 is multiplied in the y-axis value because it is equivalent to a 180° rotation about the x-axis of the camera frame.

$$Y_1 = Y_{\text{new}} * (-1) * 0.0011 \text{ (in meters)} \quad (4.4)$$

From Equations (4.3) and (4.4), we obtain the location of the package as (X_1, Y_1) . The gripper executes a Cartesian-space trajectory to pick the package relative to the tool frame.

4.4 COLLISION AVOIDANCE

Collision checking is an essential pre-requisite for motion planning to avoid self-collision and collision with obstacles present in the work area. MoveIt incorporates the FCL (Flexible Collision Library) package to detect and avoid standard 3-D objects such as boxes, cones, spheres, and custom obstacles imported as mesh files [9]. As shown in the flowchart in Fig 4.4.2, the collision obstacle can be considered mesh files or standard 3-D shapes. The traversal node determines various strategies to

implement optimal collision avoidance by detecting collision objects and estimating their proximity [9].

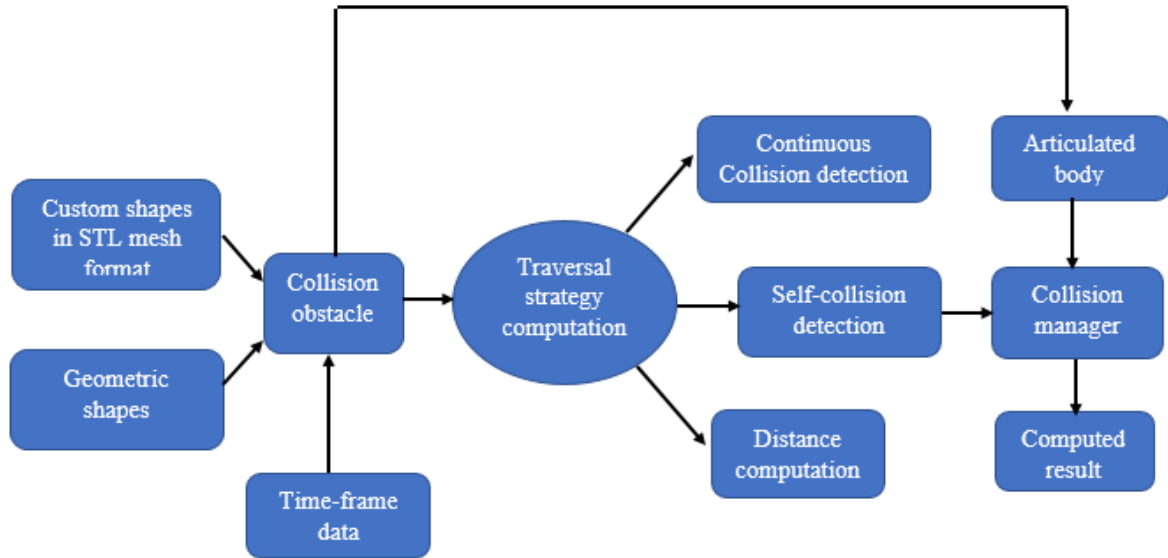


Fig 4.4.2 Flowchart of FCL operation

In the case of self-collisions, the library checks for narrow phase contact between adjacent joints and links. Similarly, broad phase collision checks are performed on the obstacles. Complicated structures are approximated to standard 3-D models, and bounding boxes are applied to the objects accordingly to approximate the object's structure into standard 3-D shapes.

Table 4.1 Collision detection capabilities of FCL

Collision manager query	Rigid obstacles	Deformable obstacles
Continuous collision detection	✓	X
Self-collision detection	✓	✓
Distance/proximity calculation	✓	✓
Broad-phase collision detection	✓	✓

The FCL library performs collision checking for self-collision and collision with other user-defined obstacles in the environment. A set of planning scenes can be established to incorporate collision avoidance in different scenarios. We establish a planning scene consisting of the adjustable camera mount and the table surface on which the manipulator is mounted as collision objects in MoveIt. The real-time dimensions of the camera mount are applied and modeled in SolidWorks. The resulting SLDASM file is converted to STL format and imported into the MoveIt workspace.

4.4.1 MODELLING THE CAMERA MOUNT IN SOLIDWORKS

The planning scene must contain a description of the collision object in the near vicinity to the arm. Thereby, the camera mount is considered as a collision object in our first planning scene. The real-time dimensions of the mount are applied in the SolidWorks model for precise collision checking.

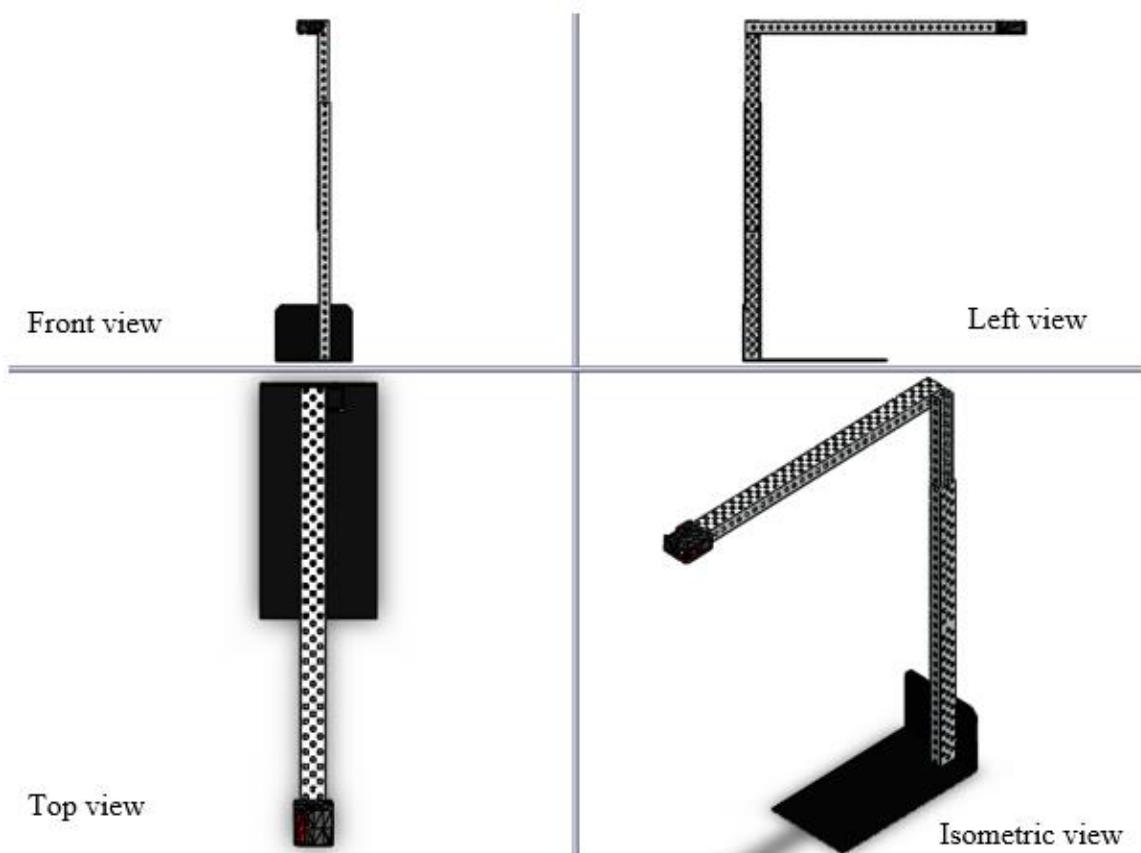


Fig 4.5 Camera mount assembly in SolidWorks

Fig 4.5 shows four standard views of the camera mount: top, front, left, and isometric. The individual frames in the assembly have the dimensions 445x35x20mm. The dimensions of the base plate are 265x140mm. The camera enclosure is located at the height of 585mm from the base plate. The dimensions of the camera enclosure are 47x52mm. The horizontal frame supports the camera enclosure (black), and the level is verified to orient the camera's view parallel to the base. The SLDASM file is then converted to STL mesh format to be imported in MoveIt workspace.

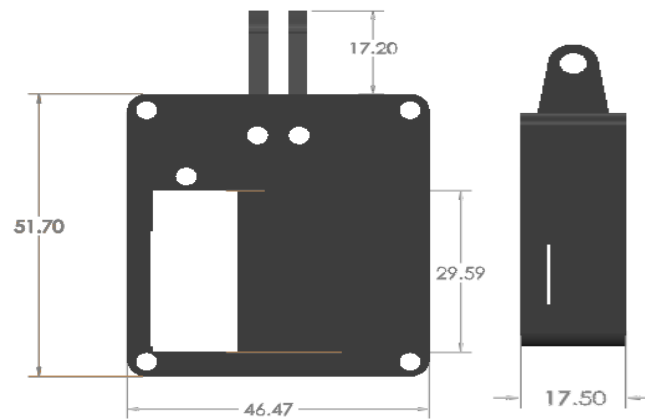


Fig 4.5.1 Bottom portion of the camera enclosure

Fig 4.5.1 shows the front and side views of the bottom portion of the camera enclosure with all dimensions represented in mm. A 30mm rectangular cut-out is given to allow access to the I/O port. The dimensions of the bottom enclosure are roughly 52x46.5x17.5 mm. This portion provides firm contact with the horizontal frame for rigid support.

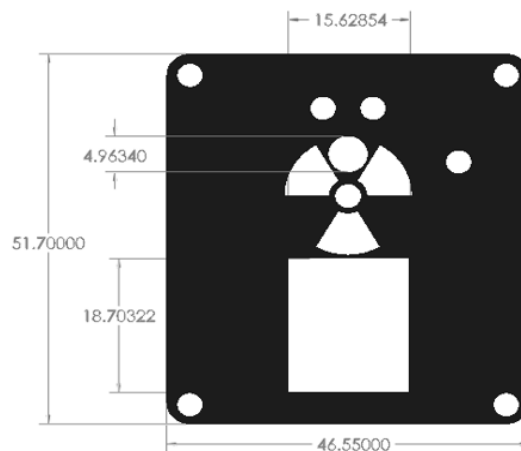


Fig 4.5.2 Top portion of the camera enclosure

All the dimensions of the top portion of the camera enclosure shown in Fig 4.5.2 are represented in mm. The dimensions of the portion are the same as the bottom part for mating. A central 19mm rectangular cut-out makes room for the camera lens and circular holes are defined such that both portions can be fastened with M3 screws. Fig 4.5.3 shows the camera's overall assembly after mating the camera enclosure with the SolidWorks assembly model's stand.

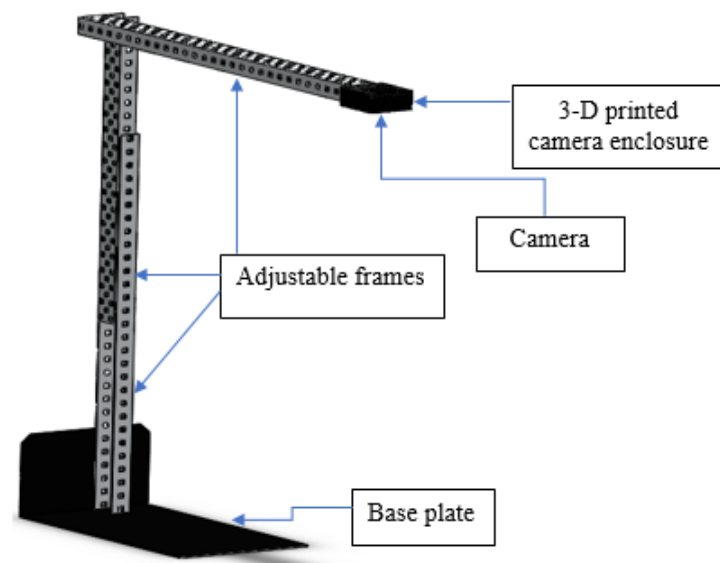


Fig 4.5.3 Camera mount model in SolidWorks

The assembly file is converted to STL format and imported into the MoveIt workspace as a mesh file for collision checking.

4.4.2 COLLISION CHECKING USING MOVEIT

In the MoveIt planning scene, the camera mount and 2-D plane are regarded as collision objects. The mount's mesh file is added into the environment, as shown in Fig 4.6. The floor is set as a collision object by defining a box node that covers the entire grid. The gripper's pose is defined using interactive markers that allow users to control the gripper's position and orientation in the virtual space. The camera base frame is placed 46 cm away from the robot's base so that the links don't hinder the camera view. In this demonstration, the motion plan is initiated from the robot's home position (start state). In Fig 4.6, the green-colored entities represent the collision

objects: table surface and camera mount. The orange-colored manipulator model represents the goal state, whereas the black-colored model represents the robot manipulator's current state (home position).

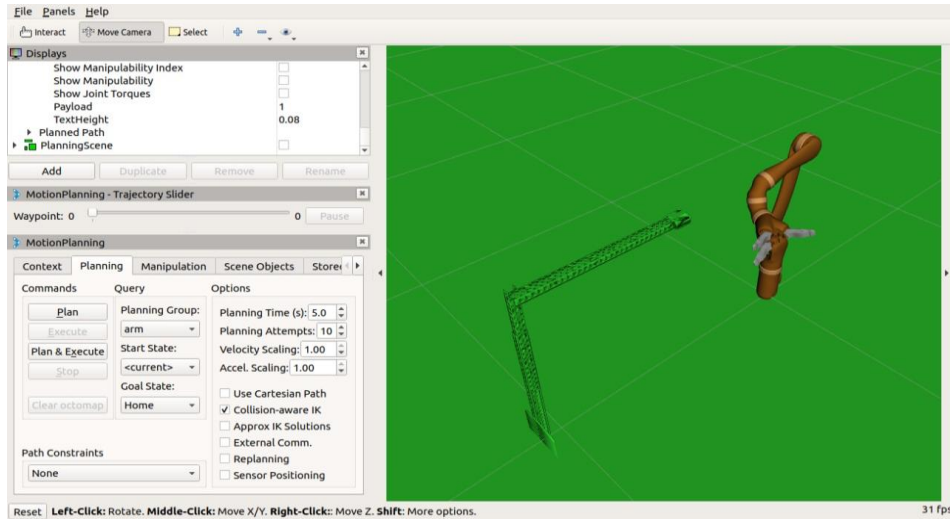


Fig 4.6 At home position

The MoveIt Rviz plugin features a trajectory slider that allows us to view the manipulator's trajectory at various waypoints. The trajectory can be set to execute a Cartesian path with collision-aware inverse kinematics to displace the end-effector along a straight path. The goal pose is set right below the horizontal frame of the camera mount for collision checking. The robot arm avoids collision with the mount and reaches the desired goal state. The MoveIt planning scene shows a preview of the trajectory to be executed and the collision object's presence as a red-colored indication. In this case, the motion plan fails to execute when a collision with a scene object is detected.

4.5 MOTION PLANNING

Motion planning is when a robotic manipulator traverses an optimal path from the start pose to the goal pose. The basic requirements of motion planning include the robot's start pose, goal pose, geometric description of the robot, and its surrounding environment. In the previous section, motion planning was carried out in a physics-based simulator such as Gazebo to verify the robot's real-time performance compatibility. A planning scene has to be generated by the user to undertake motion

planning in real-time environments. The planning scene must consist of the robot model and its surrounding collision obstacles. The dimensions of the collision object should be precisely defined to ensure better motion planning and collision avoidance.

4.5.1 SETTING UP A MOVEIT PLANNING SCENE

A planning scene refers to storing the robot's state to its surrounding environment and considering the collision objects in the vicinity. The planning scene monitor comprises the topic of the joint state. The user specifies various constraints such as position, orientation, visibility, and joints to deploy a motion planning request.

4.5.2 IMPLEMENTING KINEMATICS SOLVER

The MoveIt framework incorporates KDL (Kinematics and Dynamics Library) by defaulting inverse kinematics on the robot arm. The solver uses Inverse-Jacobian principles to determine the joint limits. Other kinematic solvers have preceded this solver developed recently, such as IK Fast and TRAC IK. The solver works well in identifying joint limits of greater than or equal to 6 DoF robotic arms. However, the time taken to solve the problem is high when compared to the recently developed plug-ins.

In this project, the TRACK IK plug-in is implemented for kinematic computation. TRAC IK uses two approaches, the first being a Newton Convergence method to determine joint limits. Moreover, the joint-state trajectory execution can be carried out by solving 3rd and 4th order polynomials. The approach involves repetitive approximation of algebraic equations for computing the roots. Secondly, it uses SQP (Sequential Quadratic Programming) to compute trajectories from current joint states.

4.5.3 JOINT-SPACE TRAJECTORY EXECUTION USING MOVEIT

The MoveIt RViz plug-in provides an interactive virtual environment for the user to visualize the robot controller's motion plan. However, before implementing motion planning in real-time environments for industrial or assistive operations, the robot's performance must be ensured in such environments through simulation. The RViz

plug-in lets us visualize the planning and execution sequence considering the robot's work envelope.

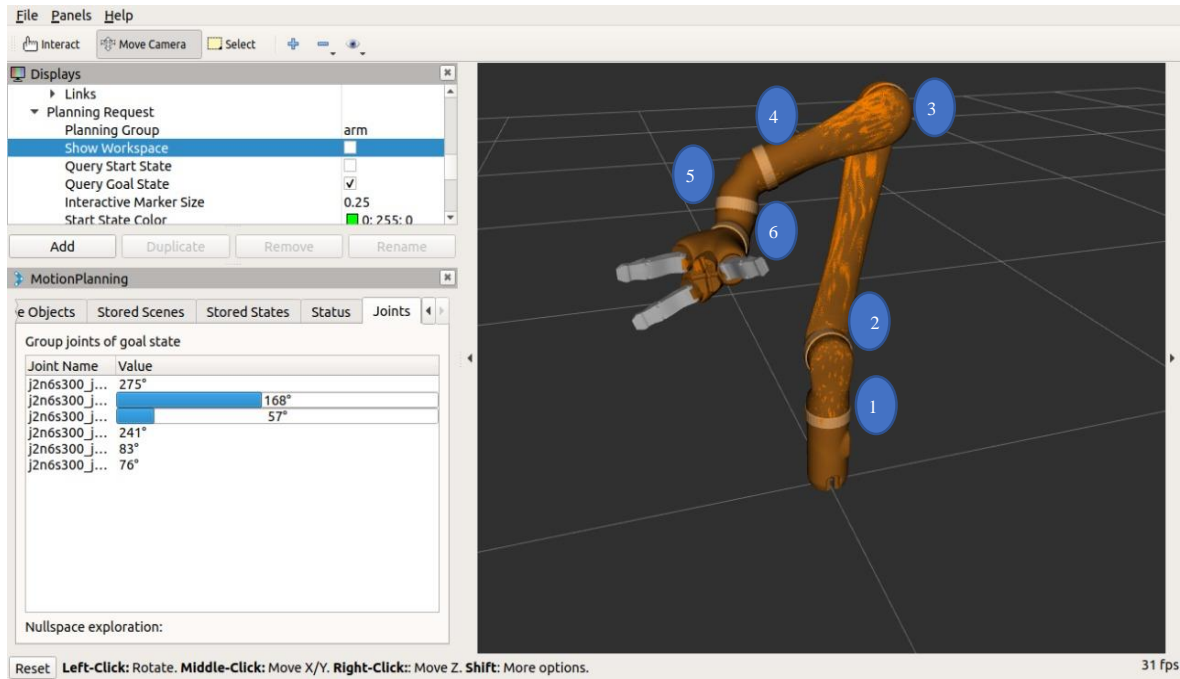


Fig 4.7 Home position of robotic arm executed using MoveIt

The MoveIt framework performs trajectory execution using its virtual default controllers for linear and circular interpolation. The controllers embedded in the framework only allow the robotic arm to execute joint-space trajectories (uncontrolled path). The arm is controlled solely by commanding the joints. In other words, a joint-space description is provided to the robot to reach the goal state. Fig 4.7 shows the joint angles of the robot's goal state, which is the home position of the JACO2 arm. In the GUI, the planning group is selected as 'arm' to command all six manipulator joints. The robot's joint angles are displayed under the 'joints' section of the GUI before and after the trajectory execution. To lessen the computational load and provide a quicker trajectory for visualization, MoveIt calculates the most optimal trajectory to reach the goal pose defined by the user. However, in this form of execution, the robot's motion between the start and goal state is unpredictable.

Complex robot manipulators constrained to work in a smaller environment require a controlled trajectory to follow a user-defined motion. Thus, such robots use Cartesian-space trajectory controllers to allow the joint motors to actuate

proportionately to align the end-effector in space uniformly. The entire operation requires many computations as the path between the robot's current and goal state is broken down into numerous segments/portions. The controller computes inverse kinematic equations at every intermediate segment, resulting in joint angles to reach the next consecutive segment. This way, joint angles are calculated at every segment to ensure the end-effector's uniform motion in space.

The `moveit_commander` package provides the Python interface needed to carry out controller programming and establish centralized communication with the `move_group` node. The following table elucidates the control flow of motion planning nodes starting from the acquiring package information such as color signature, orientation, and position coordinates until placing the package in the desired location.

Table 4.2 Pseudocode of ROS communication for motion planning

Motion planning for pick and place operation	
1:	Input: Color signature, orientation, and position coordinates of the package
2:	Output: Trajectory execution of the manipulator
3:	begin
4:	Initialize the <code>moveit_commander</code> and <code>move_group</code> node.
5:	Subscribe to the topic published by the package tracking node
6:	while True:
7:	if object is present then
8:	Publish joint-space description to MoveIt to execute picking pose
9:	Get package coordinates and orientation relative to camera frame
10:	Convert pixel coordinates to metric units relative to robot frame
11:	Request Cartesian control node to position the gripper at location
12:	Request finger positioning node to grasp the object
13:	if color signature == current signature then
14:	Publish joint-space description to reach the respective location
15:	Request finger positioning node to release the object

```
16:     else
17:         Stop
18:     else
19:         End
```

The robot's joint-space description contains all six joints' displacement angles needed to localize the end-effector at a particular pose. The math library is imported to convert the angles specified in degrees to radians using the radians function from the math dependency package. Subsequently, the joint angle input is directly fed to the motors via the robot controller.

The velocity and acceleration parameters must be specified to execute a controlled trajectory to reach the goal pose. As seen in Fig 4.8, the robotic arm's position sensors generate the current position parameters in joint angles. The computed joint angles must be published to the mainframe so that the TF library can initiate coordinate transformation and relate the same with other multiple frames of the robotic arm.

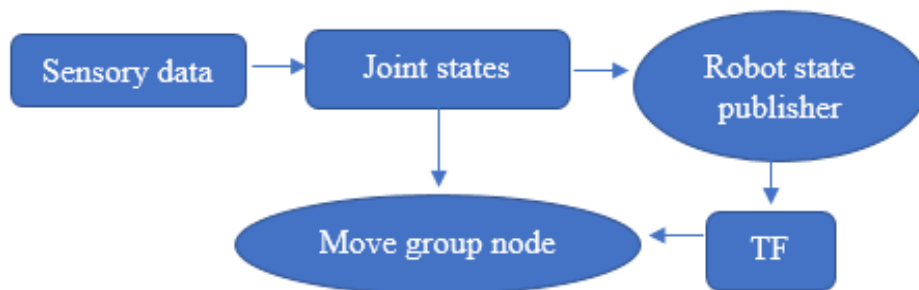


Fig 4.8 Publishing robot joint states to MoveIt

The robot state publisher node carries out the state transmission, and the states are utilized in the TF library to generate coordinate frames for the arm. The joint trajectory action node plays a crucial role in trajectory execution. This action interface sends trajectory goals to the robot controller to initiate trajectory execution. The controller also receives the after-execution messages to report the success/failure status of trajectory execution.

CHAPTER 5

IOT IMPLEMENTATION THROUGH APP SCRIPTING

5.1 GOOGLE APPS SCRIPT

Google apps script is a versatile platform promoting the development of web applications embedded in several Google platforms. This scripting platform acts as an intermediate between the webpage and ROS API to establish IoT communication. Users can define the cell parameters and spreadsheet layouts through apps script and upload the logged data into an HTML web page. Formula acceleration saves time and mitigates errors by surfacing relevant formulation as data gets logged into the sheet. The APIs in the app script shown in Fig 5.1 allows users to build, read, and edit google sheets programmatically.

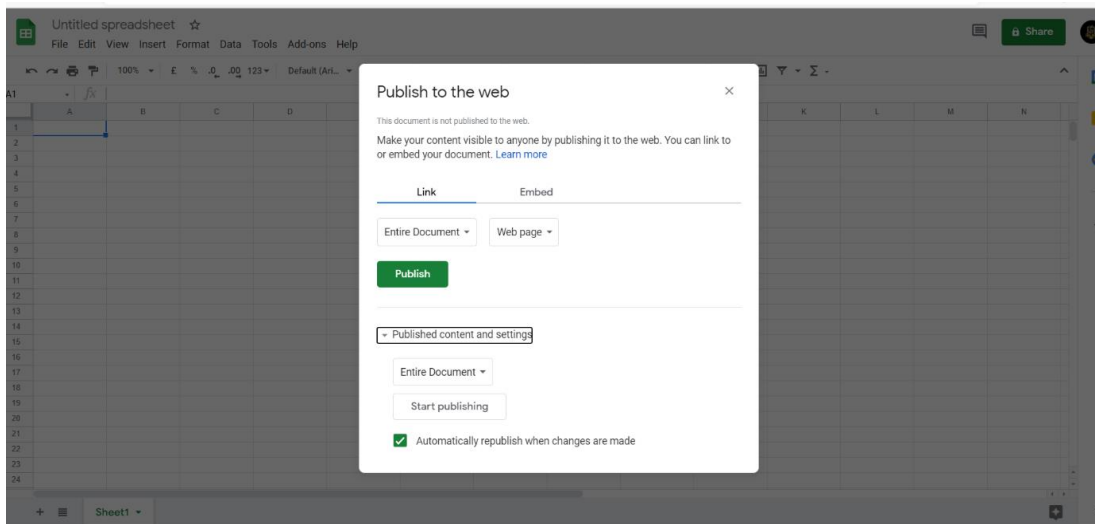


Fig 5.1 Deploying apps script as a web application

Users can click on the explore button to customize visualizations and access other available tendencies in a less strenuous way to navigate data. Sheets contain all of the resources required to evaluate, visualize and draw insights from the formulated results. Without explicitly defining the contents and formatting changes of the spreadsheet, the apps script allows users to target specific cells by creating custom functions and macros. While defining a spreadsheet, the respective column headers

to log data of sorted packages are labeled as Product name, Product ID, Quantity, sorted status, and timestamp.

5.1.1 SPREADSHEET UPDATION THROUGH APP SCRIPTING

Additional APIs and connector software help us sort records from several sources, whether in CSV, Excel, or another file format. As ROS API sends a request to the application, the app script runs the doGet(). The app script code is deployed as a web app that accesses the spreadsheet information using GET requests. The app script can communicate with google sheets in two ways: the spreadsheet is modified to adjust the user interface or perform other tasks using the apps script editor.

The python script sends the instruction to the google app script from the ROS API using the request library. HTTP method has two GET and POST libraries, but one of the most common HTTP methods is GET. The GET method indicates that we are trying to retrieve data from a specified resource, in this case, the Google spreadsheet. Dependencies such as 'requests' and 'datetime' are imported in the script send an HTTP request and log the current date and time under the timestamp column. The spreadsheet is deployed as a web application, and the URL is passed into the 'get' function to log the parameters in the spreadsheet. Then we take the URL of the spreadsheet as a web application and attach it to the python code as an address line (i.e., product id, quantity, sorted, sorted timing) for sending mail. The mail app class sends mail containing the package's parameters using the built-in python library through HTTP request.

5.1.2 DEVELOPING AUTO-GENERATED E-MAIL ALERTS

In the previous section, the developed python script serves as a software interface between ROS and Google Apps Script. The request library retrieves the package parameters from the apps script and sends an auto-generated email to the user's Google mail ID. The mail contains information on parameters such as product ID, quantity, sorted status, and sorted time. After sending the mail, sorted data is automatically updated in the spreadsheet. The apps script code snippet shown in Fig

5.3 performs the spreadsheet's data logging activity using the parameters retrieved from the python script.

A conditional structure is developed to detect the data log in the spreadsheet to send the auto-generated email. The user's Gmail ID is stored in the 'to' variable indicating the recipient similarly; similarly, the subject is stored in the message variable. The email status is printed in the development computer terminal for notifying the user to check his/her inbox.

Table 5.1 Pseudocode of IoT communication during pick and place operation

Incorporating IoT communication into the sorting system

```

1:  Input: Gripper status and color signature of the package
2:  Output: Auto-generated email
3:  begin
4:      Publish package's color signature to a topic from vision tracking node
5:  while True:
6:      if gripper status == 'open' then
7:          Subscribe to the topic published by vision tracking node
8:          Initiate callback function to retrieve color signature from topic
9:          if color signature == 12 then
10:             Log PCB parameters with current timestamp in spreadsheet
11:             Send auto-generated email with PCB package summary
12:          elif color signature ==13 then
13:             Log controller parameters with current timestamp in spreadsheet
14:             Send auto-generated email with controller package summary
15:          elif color signature ==23 then
16:             Log motor parameters with current timestamp in spreadsheet
17:             Send auto-generated email with controller package summary
18:          else
19:             Stop

```

20:	else
21:	End

5.2 CREATING A DYNAMIC DASHBOARD

This topic interprets the development of the KINOVA dashboard to log information of the sorted packages; its purpose is to update the dashboard data when the robotic arm completes the pick and place operation in the respective bins. The primary step is to give column headers such as Timestamp, Product ID, Product Name, Quantity, and sorted status in the google spreadsheet. Google app script acts as an intermediate between the google spreadsheet and the python. Then deploying the google script code to the cloud after deploying it provides a link using request library from python.

5.2.1 SENDING SPREADSHEET DATA TO WEBPAGE

The concept is to use the google sheet as a Jason file, the foremost step is creating a new sheet in the google spreadsheet, and after all the script work, we are publishing the spreadsheet to the web and then it generates a JSON endpoint and finally using our google sheet as a JSON endpoint and also we can make our google sheet public for collaboration and data entry purpose. An overview of how to use the spreadsheets data API's JSON output format to view a list of cells feed for a specific worksheet in a spreadsheet. In this view, it is acceptable to use the JSON file as it gives a better way to replace IPs, passwords, et cetera and changes the size and number of components. We can use the completed worksheet or a JSON file after deploying it to the cloud builder appliance.

5.2.2 UPDATING WEBPAGE DASHBOARD USING AJAX REQUESTS

CSS stands for cascading file sheets; HTML and CSS are the two of the most popular web page building technologies. HTML and CSS, along with graphics and scripting, are the foundation for developing web pages and web applications. CSS may be used to describe the padding around images and other objects, as well as the design, thickness, and color of a table's border. CSS allows web designers more precise

control over the appearance of pages than HTML does. CSS is used to make the display more vibrant by giving some gradient colors to the dashboard. The AJAX library is the most used in javascript its stands for asynchronous javascript XML, which is a technique to make fast dynamic web pages. You can regard ajax as a part of javascript; it is running on javascript. So, it can be seen as a way to replace data using a server and update web page sections without reloading the entire page.

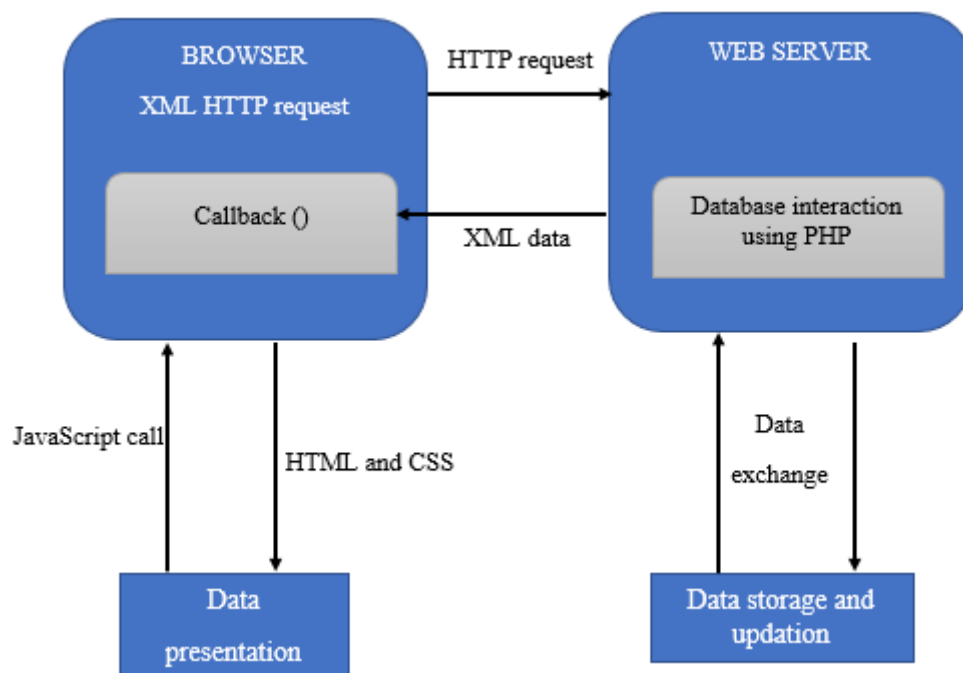


Fig 5.2 Flowchart of AJAX method

Fig 5.4 shows the workflow of the AJAX technique commonly employed in web development applications. The user sends JavaScript call request to the XML HTTP request block containing the callback function in the browser block. The HTTP request is then sent to the web server, and the server does database interacting using a scripting language such as PHP to retrieve the data back to the callback function. The control flows back to the browser block to display HTML and CSS data in the browser. This way, the AJAX technique is used to automatically send and retrieve information in the browser background without refreshing the web page. Afterward, updating the parameters like the product, product id, which are in the columns after updating the parameters, then publishing the parameters to the google app script.

product ID	Product	Quantity ▼	Sorted	Sorted Time
CL01	Controller	1	YES	2021-03-24 15:25:24
PS01	PCB	1	YES	2021-03-24 15:25:29
CL01	Controller	1	YES	2021-03-24 15:25:35
PS01	PCB	1	YES	2021-03-24 15:25:41
CL01	Controller	1	YES	2021-03-24 15:25:47
CL01	Controller	1	YES	2021-03-24 15:25:52

Fig 5.3 Column headers of the dashboard

After all these steps, the publishing sheet to the web and generate a unique link with that link afford the parameters of the sheets and then publish in the HTML web page. The HTML table is a sortable kind that can sort the entries alphabetically, numerically, and based on sorted time. In the next chapter, the experimental results obtained after applying the methodologies and pseudocodes are discussed.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 PACKAGE DETECTION AND TRACKING

The USB camera mounted over the workspace detects objects based on color signatures using blob detection. The detected packages are enclosed with bounding boxes as shown in Fig 6.1 to implement continuous tracking. The color-connected components option in the configuration utility allows the camera to register two color signatures as a single object. Three different packages identical in dimensions but dual-color combinations are detected by the camera, as shown in Fig 6.1. The 's' quantity contains 2 digits representing the primary and secondary color signatures in sequential order. Simultaneously, ' Φ ' represents the angle at which the package is positioned on a flat surface from the difference in orientation of individual color components. Color signature is an integer data type published to the topic '/pixel_data' from which the email-generator script (subscriber) retrieves the information to send emails containing parameters of the respective packages automatically.

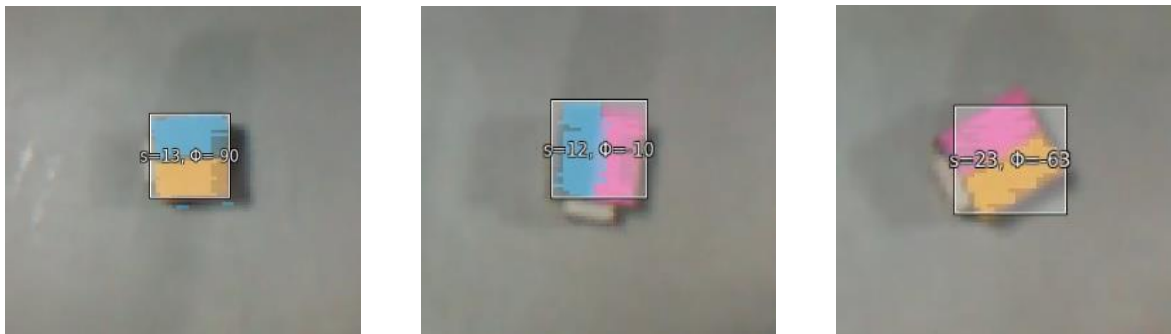


Fig 6.1 Package recognition

Fig 6.1 represents the recognized image of three different objects with different signatures and angles. That recognized object is tracked using the colors here; two colors are used not only for tracking but also for finding the object's orientation. The vision system's output contains the signature, x, y, width, height, and angle of the object. The recognized object gives the position in pixel resolution that origin starts

from top left of the frame and (315, 207) to the frame's bottom right. Though the origin is changed to the center of the frame by (157.5, 103.5), both the robotic Cartesian path and camera frame coincide. The object's position from the camera frame is manipulated with a new origin which is in pixels. Fig 6.2 represents the object's position in pixels, the object's color signature, and its orientation in degrees.

```
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -150.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 17, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 98, 'Pixel y = ', 15, 'Orientation in degree = ', -145.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 100, 'Pixel y = ', 16, 'Orientation in degree = ', -149.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -151.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -149.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 15, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -142.0)
('Position of the object in pixel', 'sig = ', 11, 'Pixel X = ', 99, 'Pixel y = ', 16, 'Orientation in degree = ', -146.0)
```

Fig 6.2 Position and orientation values of the package

Fig 6.2.1 represents the pixel's converted value to the meter by implementing the algorithm discussed in Chapter 4. The pixel values generated along the x and y axes are converted to meters and stores in the variables 'x-meter' and 'y-meter' as shown. The package orientation is converted to radians multiplying a factor of ($\pi / 180$) with the generated degree values to rotate the gripper while picking the package automatically.

```
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15873015873015872, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -150.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -147.0)
('position of the object in meters', 'X-meter = ', 0.15873015873015872, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -149.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15873015873015872, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -145.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -150.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -150.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01971014492753623, 'Orientation degree = ', -150.0)
('position of the object in meters', 'X-meter = ', 0.15555555555555556, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -145.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -151.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15873015873015872, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -149.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -142.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.01855072463768116, 'Orientation degree = ', -146.0)
('position of the object in meters', 'X-meter = ', 0.15714285714285714, 'Y-meter = ', 0.017391304347826087, 'Orientation degree = ', -142.0)
```

Fig 6.2.1 Real-time package tracking

Now, the package's position and orientation parameters are known relative to the robot's frame to initiate an optimal motion plan. During trajectory execution, it must be made sure that the manipulator does not collide with the table surface and the camera mount. In the next section, real-time collision checking is carried out in the MoveIt framework before trajectory execution to ensure the robot manipulator's safety.

6.2 COLLISION AVOIDANCE USING MOVEIT

FCL (Flexible Collision Library) is used to identify the model's collision, whether it is blocking its path or the robot's final goal position is colliding 3D model imported in Rviz [9]. Trac_IK library is used to solve inverse kinematics and plans the robot's path, but before planning FCL library also checks for collision status in MoveIt. In case of collision, FCL will inform Trac-IK to plan a different path to reach the destination.

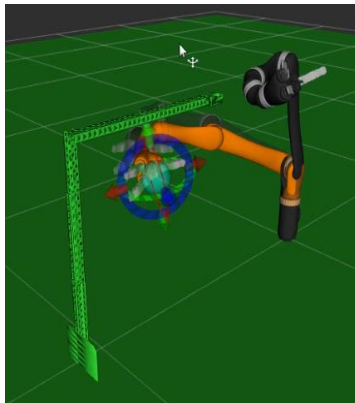


Fig 6.3 Goal position fixed by user



Fig 6.3.1 Avoiding collision to reach goal pose



Fig 6.3.2 Preview of the collision object

In Fig 6.3.1, the robot arm took the longest path to reach the destination by avoiding the obstacle and avoiding any collision. The ultimate goal of the FCL library is to avoid self-collisions and collision with neighboring objects for the safety of the robotic manipulator. Fig 6.3.2 shows the robot's final goal position is made to collide with the obstacle in Rviz, but the FCL library recognizes the collision it stopped the Trac-Ik from generating path. The library's precautionary features make the robot work alongside other robots and humans safely.

6.3 AUTONOMOUS PICK AND PLACE OPERATION

TRAC-IK inverse kinematics library is used to plan the path using both forward and inverse kinematics which uses KDL Newton's – convergence algorithm and sequential quadratic programming. But in this project, we have used inverse kinematics for the robot to calculate the joint angles needed to reach the goal position. Inverse kinematics goal position is given as each link's joint angle is defined in degree or radian, it is fed into the TRAC-IK solver to generate the path.

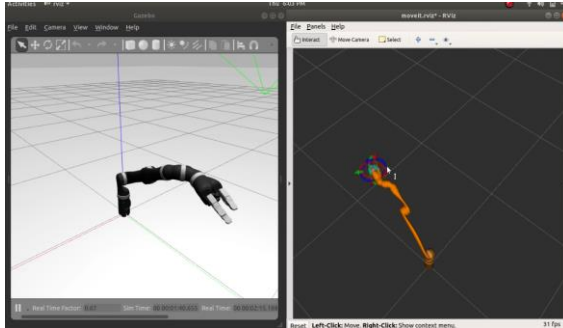


Fig 6.4 Initial position of the robot

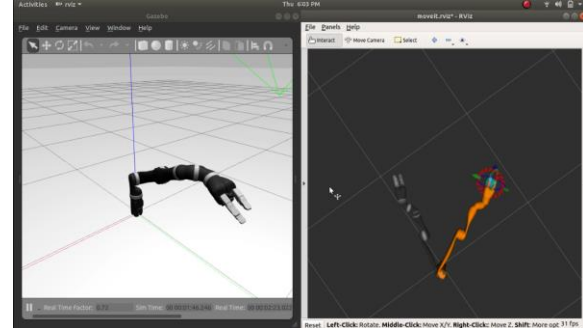


Fig 6.4.1 Assigning the goal state

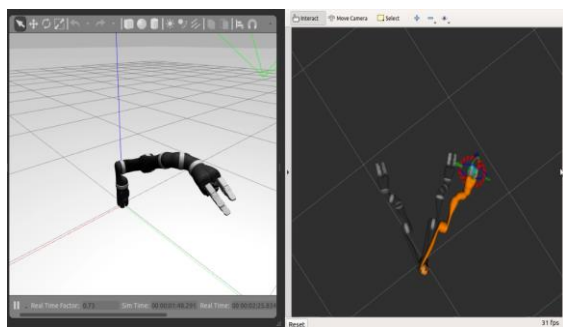


Fig 6.4.2 Initiated motion planning

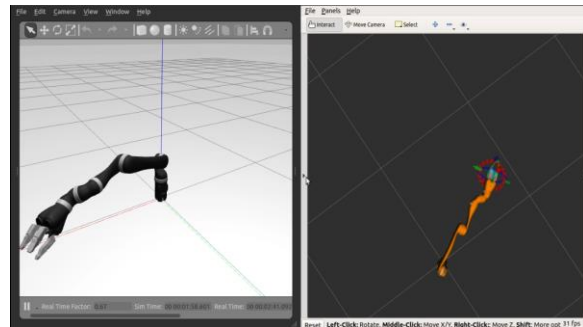


Fig 6.4.3 Reached goal state

The above figures show a side-by-side comparison of the Gazebo and MoveIt Rviz plug-in to visualize the motion planning and trajectory execution of the manipulator. Fig 6.4 current position and goal position is the same, so there is no inverse solver required. In Fig 6.4.1, the goal position is set at a specific position where the robot's joint's goal position is obtained then fed to the TRAC-IK solver. After TRAC-IK plans the shortest trajectory to the goal position, there is no obstacle in the planned path. Fig 6.4.3 represents the position of the robot reaching the goal destination after executing the planned path. TRAC-IK planned trajectory represents the joint values from the current joint angle to the goal position with several joint values. Those joint angles from the planned path are fed to the robot to perform the same path generated by the inverse kinematics library.

6.3.1 JOINT-SPACE TRAJECTORY EXECUTION

The Kinova JACO² robotic arm is interfaced to PC with USB through which the data is transferred using Udev rules to provide read and write permission from the robotic manipulator.



Fig 6.4 Control of robotic arm using joint-space description

The Udev rules are set whenever the robotic arm is interfaced with the development computer allowing data transmission to automatically send a joint-space description and retrieve the current pose of position sensors in the robotic manipulator. The robotic arm is further interfaced with ROS MoveIt to incorporate additional features such as communication and collision avoidance. Fig 6.4 represents the different poses of the robot with different joint angles. Here, a joint-space description is provided to control the robot using MoveIt python API. The data published through Move_group node is sent to TRAC_IK inverse kinematics solver to incorporate motion planning.

6.3.2 CARTESIAN-SPACE TRAJECTORY EXECUTION

Cartesian control of the manipulator enables the controller to estimate a straight-line path between the end-effector's current and goal pose. Control is established by dividing the straight-line path into numerous sections such that inverse kinematics is computed consecutively for the manipulator to reach each section.



Fig 6.5 Picking pose of the robot



Fig 6.5.1 Wrist orientation relative to the box

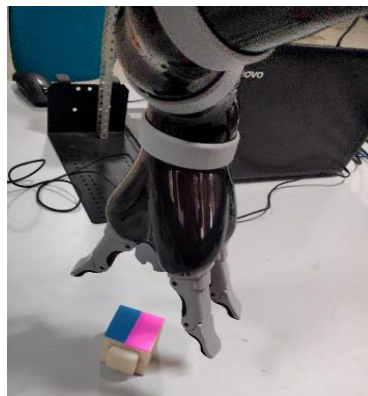


Fig 6.5.2 Cartesian path traversal

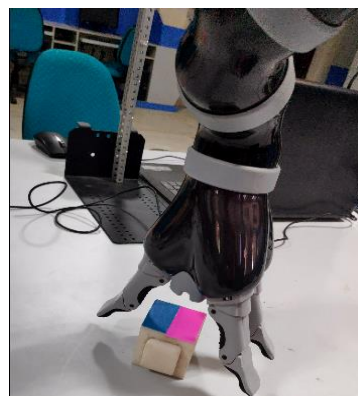


Fig 6.5.3 Pre-defined motion along z-axis

Figure 6.5 represents the robotic arm positioned at its origin in which the gripper is coinciding with the camera pixel coordinates $C (157.5, 103.5)$. The parameters required to execute a Cartesian path contain the end-effector goal position (x,y,z) in meters and its orientation about x , y , and z axes in degrees. In Fig 6.5.1, to orient the finger, we need to vary the gripper orientation about the z -axis to turn the robot's wrist according to the box's orientation. Fig 6.5.2 represents the tracking of the object in 2D workspace using vision sensor X , Y values of pixel values of the recognized object is converted into the meter and fed into X , Y position to the Cartesian path robotic manipulator then it reaches the box with the position.

The misalignment of y -axes of the camera frame and robot frame causes inversion of co-ordinate values along the robot's y -axis due to USB camera positioning. A factor of (-1) is multiplied with the resulting co-ordinates value in meters to negate the sign discussed in Chapter 4. The robot manipulator's up and down motion is controlled about the z -axis. Fig 6.6 represents the robot's vertical motion according to pre-defined vertical movement because the 2-D camera does not provide depth of the object located on the table surface.

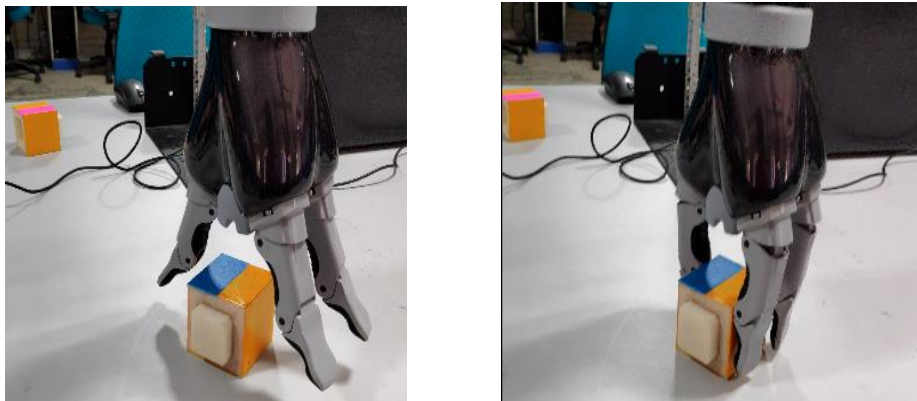


Fig 6.6 Altering gripper pose relative to package orientation

The ROS Move_Group node has 2 types of manipulation which are “arm” and “gripper.” The arm is used to execute trajectories on the 6 DoF Kinova manipulator. The “gripper” move_group is used to control the three-fingered gripper; the positions can be provided either millimeter or percentage. For instance, the gripper's position is relative; for instance, if the finger is closed by 30%, -30% is fed to the gripper client

to open the fingers by 30%. Here, the opening and closing of fingers are done twice to grasp the object and then release the object at the desired position.



Fig 6.6.1 Grasping and releasing the detected package

The gripper's grasp and release capacity is assigned as 60% in the ROS API finger positioning python script by keeping into account its dimensions. The opening and closing actions are shown in Fig 6.6.1 are assigned for all three fingers. The opening and closing of the finger are controlled by establishing service communication in ROS. Whenever the main script passes the “close” string to the service, the finger control script closes all three fingers by -60%, and if the service gives “open” command, the fingers open by +60%, compensating for grasping action to bring the gripper back to its initial state.

6.4 AUTOMATIC DASHBOARD UPDATION

Although the project primarily targeted industry 4.0 applications, it is essential to develop an interactive dashboard to keep track of components and packages sorted by the workplace's manipulator. The Google Apps Script web application is utilized to log the sorting information into a Google spreadsheet such that it indicates the product summary and sorting timestamps of each package. After dropping a package to the goal destination, the request library from python sends the google apps script's required parameters, updating the Google spreadsheet values. Fig 6.7 represents the updated values sent by HTTP request in the python script. The following section discusses automatic email generation based on the color signature of the package.

	A	B	C	D	E	F
1	Timestamp	productid	Product	quantity	sorted	sortedTime
2	Wed Mar 24 2021, 05:55:25	CL01	Controller	1	YES	2021-03-24 15:25:24
3	Wed Mar 24 2021, 05:55:30	PS01	PCB	1	YES	2021-03-24 15:25:29
4	Wed Mar 24 2021, 05:55:37	CL01	Controller	1	YES	2021-03-24 15:25:35
5	Wed Mar 24 2021, 05:55:42	PS01	PCB	1	YES	2021-03-24 15:25:41
6	Wed Mar 24 2021, 05:55:48	CL01	Controller	1	YES	2021-03-24 15:25:47
7	Wed Mar 24 2021, 05:55:53	CL01	Controller	1	YES	2021-03-24 15:25:52
8	Wed Mar 24 2021, 05:55:58	LM00	Motor	1	YES	2021-03-24 15:25:58
9	Wed Mar 24 2021, 05:56:04	CL01	Controller	1	YES	2021-03-24 15:26:03
10	Wed Mar 24 2021, 05:56:09	LM00	Motor	1	YES	2021-03-24 15:26:08
11	Wed Mar 24 2021, 05:56:15	CL01	Controller	1	YES	2021-03-24 15:26:14
12	Wed Mar 24 2021, 05:56:20	PS01	PCB	1	YES	2021-03-24 15:26:19
13	Wed Mar 24 2021, 05:56:26	PS01	PCB	1	YES	2021-03-24 15:26:25

Fig 6.7 Updating spreadsheet with google apps script

An additional python script is developed to load new data entries in the web page during sorting operation after a given time delay. Instead of manually reloading the webpage URL to view the updated data, we have developed a python script that uses HTTP requests to extract data from the spreadsheet automatically in the background. In JavaScript and HTML table, it is made sortable in alphabetical and numerical order, and CSS provides an interactive and pleasant background, as illustrated in Fig 6.8.

Kinova Dashboard				
product ID	Product ▼	Quantity	Sorted	Sorted Time
CL01	Controller	1	YES	2021-03-24 15:25:24
PS01	PCB	1	YES	2021-03-24 15:25:29
CL01	Controller	1	YES	2021-03-24 15:25:35
PS01	PCB	1	YES	2021-03-24 15:25:41
CL01	Controller	1	YES	2021-03-24 15:25:47
CL01	Controller	1	YES	2021-03-24 15:25:52
LM00	Motor	1	YES	2021-03-24 15:25:58
CL01	Controller	1	YES	2021-03-24 15:26:03
LM00	Motor	1	YES	2021-03-24 15:26:08
CL01	Controller	1	YES	2021-03-24 15:26:14
PS01	PCB	1	YES	2021-03-24 15:26:19
PS01	PCB	1	YES	2021-03-24 15:26:25

Fig 6.8 Updating the dynamic dashboard using AJAX requests

6.5 SENDING AUTO-GENERATED EMAILS TO USER

The Google spreadsheet is updated using the “doGet” function in apps script editor to access the python script contents using a GET request. Subsequently, an additional, conditional code snippet is developed inside the ‘doGet’ function to check for data

log and retrieve data entries from the spreadsheet. After retrieval, an auto-generated mail is sent to the user after every package/item is sorted by the manipulator, as shown in Fig 6.9.

The package’s color information is retrieved from the ‘/pixel_data’ topic using the subscriber node's callback function. Upon completion of the sorting operation, the subscriber node checks for the gripper status, i.e., open/close, to initiate the email generation process. If the gripper status condition reads open, the node is initiated, and the callback function is called by passing the color signature as an argument. The detected color signatures discussed in Section 6.1 are incorporated in the conditional structure to log information of the respective sorted package by checking its color-signature Fig 6.9. The ‘unregister’ function ensures that once a color signature is retrieved and an email is sent, the node unsubscribes from the topic until a new color signature is detected to avoid notification spam in the inbox. After retrieving the package's respective parameters, the ‘send_email’ function shown in Fig 6.9.1 is called within the callback function using the class method.

The information about the sorted package is sent to the web application URL to log the spreadsheet automatically. For instance, if the published color signature is ‘12’, the PCB parameters are logged in the respective spreadsheet columns to send an auto-generated email, as shown in Fig 6.9.2.

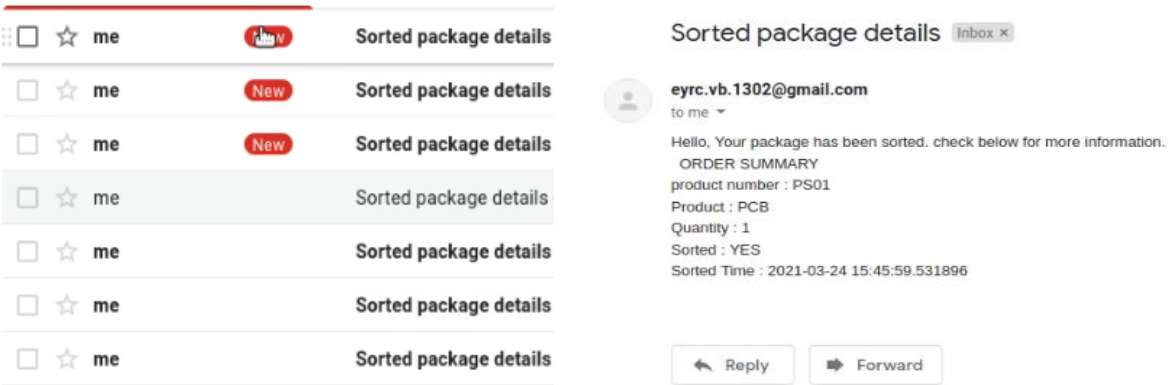


Fig 6.9 Auto-generated email as user notification

In the spreadsheet and user email, the packages are named “PCB,” “Motor,” and “Controller,” representing that sorting operation was carried out on dual-colored boxes shown in Section 6.1. After sorting each package, parameters such as product number, product ID, Product name, quantity, and the timestamps are displayed under “ORDER SUMMARY” in the body of the e-mail. The data is updated either to the customer or the managing person based on the email ID provided in the apps script.

CHAPTER 7

FUTURE SCOPE

This project presents the successful integration of the Kinova JACO² 6 DoF robot manipulator with a low-cost USB camera through ROS to automate color-based sorting operations. The perception module computes the package's position and orientation parameters irrespective of its location on a flat surface in its field of view. The developed algorithm effectively correlates the image coordinate system's pixel values into metric units relative to the robot's frame. Throughout the project, collision avoidance was ensured by implementing the Flexible Collision Library (FCL) in ROS and importing models of the neighboring obstacles in the MoveIt workspace. ROS nodes developed as python scripts in the API send the package coordinates for independent control of the manipulator's arm and gripper through ROS topics and service communication.

Optimal trajectories were generated with the TRAC IK inverse kinematics solver to reach the goal state's coordinates. The manipulator is programmed to execute a Cartesian-space trajectory to reach the goal state and pick the package by traversing in a straight path. Consecutively, packages varying in color are sorted into their respective containers by executing a joint-space trajectory. Using IoT in the vision-based sorting setup, the names and timestamps of sorted packages are automatically logged and updated in a spreadsheet through Google Apps Script. The user is notified of the completion of the sorting operation with a detailed summary through auto-generated e-mails. This sorting system's scope can be incorporated in Industry 4.0 and extended using QR codes in an intelligent warehouse setup. A centralized server can be established within an industry, allowing several mobile robots, robotic manipulators, and aerial robots to communicate remotely with each other and carry out material handling processes.

Upon real-time experimentation, the scope of the existing project can be significantly expanded to promote broader and reliable applications in industrial

sectors. Currently, the Kinova robotic manipulator is programmed to carry out the autonomous color-based sorting operation using input from a low-cost 2-D perception module through ROS communication. However, the meta-operating system provides convenient physics-based simulators and a robust programming API to effectively carry out communication, perception, and manipulation tasks. The future work to be incorporated to yield the performance and applicability of ROS-supported robot manipulators falls into the following perspectives:

- To integrate the robot manipulator with a 3-D vision system for depth perception.
- To promote dynamic object tracking to enable the manipulator to pick objects from a recycling conveyor system.
- To impart autonomous detection and sorting of objects labeled with QR codes.
- To establish autonomous control between mobile robots, manipulators, and aerial robots through a centralized ROS server.
- To ensure control compatibility with all assistive and industrial ROS-supported articulated robot manipulators.
- To develop a robust software architecture in ROS for carrying out autonomous component assembly operations.

REFERENCES

1. Baranowski, L., W. Kaczmarek, J. Panasiuk, P. Prusaczyk, and K. Besseghieur, Integration of vision system and robotic arm under ROS. *23rd International Conference Engineering Mechanics, Syratka, Czech Republic*, 15 – 18 May 2017.
2. Ali, Md Hazrat, K. Aizat, K. Yerkhan, T. Zhandos, and O. Anuar, Vision-based robot manipulator for industrial applications. *Procedia computer science*, 2018, 133, 205-212.
3. K. Xia and Z. Weng, Workpieces sorting system based on industrial robot of machine vision. *3rd International Conference on Systems and Informatics (ICSAI), Shanghai, China*, 2016, pp. 422-426.
4. Khan, Khasim A., Revanth R. Konda, and Ji-Chul Ryu, ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task. *Advances in robotics research*, 2018, 2, pp. 113-127.
5. Qin, Qin, Dongdong Zhu, Zimei Tu, and Jianxiong Hong, Sorting system of robot based on vision detection. *In International Workshop of Advanced Manufacturing and Automation, Springer, Singapore*, 2017, pp. 591-597.
6. Y. Jia, G. Yang and J. Saniie, Real-time color-based sorting robotic arm system. *2017 IEEE International Conference on Electro Information Technology (EIT), Lincoln, NE*, 2017, pp. 354-358.
7. W. Qian et al., Manipulation task simulation using ROS and Gazebo. *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), Bali, Indonesia*, 2014, pp. 2594-2598.
8. Campeau-Lecours, H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire, and L.-J. C. L'Ecuyer, Kinova Modular Robot Arms for Service Robotics Applications. *Rapid Automation*, 2019, pp. 693–719.
9. J. Pan, S. Chitta and D. Manocha, FCL: A general purpose library for collision and proximity queries. *2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA*, 2012, pp. 3859-3866.

10. P. Yadav, M. Uikey, P. Lonkar, S. Kayande and A. Maurya, Sorting of Objects Using Image Processing. *2020 IEEE International Conference for Innovation in Technology (INOCON), Bangluru, India, 2020*, pp. 1-6.
11. Jih-Gau Juang, Y.-Ju Tsai, and Yang-Wu Fan, Visual Recognition and Its Application to Robot Arm Control. *Applied Sciences*, 2015, 5, pp. 851–880.
12. Djajadi, Arko, Fiona Laoda, Rusman Rusyadi, Tutuko Prajogo, and Maralo Sinaga, A model vision of sorting system application using robotic manipulator. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 2010, 8, pp. 137-148.
13. Kale, Vishnu R., and V. A. Kulkarni, Object sorting system using robotic arm. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2013, 2, pp. 3400-3407.
14. Kumar, Rahul, Sunil Lal, Sanjesh Kumar, and Praneel Chand, Object detection and recognition for a pick and place robot. *In Asia-Pacific world congress on computer science and engineering, IEEE*, 2014, pp. 1-7.
15. J. Chen and K. Song, Collision-Free Motion Planning for Human-Robot Collaborative Safety Under Cartesian Constraint. *2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 2018*, pp. 4348-4354.
16. H. Abouaïssa and S. Chouraqui, On the control of robot manipulator: A model-free approach. *Journal of Computational Science*, 2019, 31, pp. 6–16.
17. H Ka, D Ding and RA Cooper, Three Dimensional Computer Vision-Based Alternative Control Method for Assistive Robotic Manipulator. *International Journal of Advanced Robotics and Automation*, 2016, 1, pp. 01–06.
18. Neerparaj Rai, Bijay Rai, and Pooja Rai, Computer vision approach for controlling educational robotic arm based on object properties. *2014 2nd International Conference on Emerging Technology Trends in Electronics, Communication and Networking, Surat, India, 2014*, pp. 1-9.

19. V. Kumar, Q. Wang, W. Minghua, S. Rizwan, S. M. Shaikh, and X. Liu, "Computer Vision Based Object Grasping 6 Dof Robotic Arm Using Picamera," 2018 4th International Conference on Control, Automation and Robotics (ICCAR), Apr. 2018.
20. S. Hajari and S. A. Shankarpurkar, Personal Computer Based Robotic Arm with Vision, *International Journal of Image Processing and Vision Science*, 2013, pp. 203–206.
21. N. Rofalis, L. Nalpantidis, N. A. Andersen, and V. Krüger, Vision-based Robotic System for Object Agnostic Placing Operations. *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2016.
22. K. Radlak and M. Fojcik, Integration of robotic arm manipulator with computer vision in a project-based learning environment. *2015 IEEE Frontiers in Education Conference (FIE)*, Oct. 2015.
23. Lan Ye and Chao Sun, Trajectory planning of 7-DOF redundant manipulator based on ROS platform. *2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA), Chongqing, China, 2020*, pp. 733-736.
24. E. Najafi and M. Ansari, Model-Based Design Approach for an Industry 4.0 Case Study: A Pick and Place Robot. *2019 23rd International Conference on Mechatronics Technology (ICMT), Salerno, Italy, 2019*, pp. 1-6.

APPENDIX

List of Programs

Program No.	Title of the program	Page No.
1.	Joint-space description of the manipulator	69
2.	Algorithm implementation for Cartesian movement	69
3.	Acquiring and sending detected package details to Google Apps Script	70
4.	Logging package details in spreadsheet and auto-generating emails	71

Code snippet 1: Joint-space description of the manipulator

```
lst_joint_angles_1 = [math.radians(629), # Picking home pose
                      math.radians(237),
                      math.radians(106),
                      math.radians(-19),
                      math.radians(780),
                      math.radians(-1009)]

lst_joint_angles_2 = [math.radians(173), # Goal pose description
                      math.radians(125),
                      math.radians(275),
                      math.radians(-181),
                      math.radians(388),
                      math.radians(78)]
```

Code snippet 2: Algorithm implementation for Cartesian movement

```
def cam_datas(self, inp):
    global pixel_meter_x
    global pixel_meter_y
    half_x = 315/2
    half_y = 207/2
    center_x = inp.pixel_x - half_x
    center_y = inp.pixel_y - half_y
    # print("reference to center co-ordinate",center_x,center_y)
    pixelx_1 = 0.28/157.5
    pixely_1 = 0.12/103.5
    pixel_meter_x = center_x * pixelx_1
    pixel_meter_y = (-1) * center_y * pixely_1
    # print("pixel to meter", pixel_meter_x, pixel_meter_y)
```


Code snippet 3: Acquiring and sending detected package details to Google Apps Script

```
def send_email(self, val):
    URL = "https://script.google.com/macros/s/AKfycbzVgDnGLwIT718C-
          mhBQmkeO2GZv0Vh80McTm_vKL1YlMTseeZbGZXz25xyPSNqr
          I5w/exec"
    mail_output = requests.get(URL, params=val)
    print(mail_output.content)
def callback(self, color_sign):
    print("Value in msg file is {}".format(str(color_sign.mail)))
    if color_sign.mail == 12:
        parameters= {"id": "kinova_sheet",
                      "productid": "PS01",
                      "Product": "PCB",
                      "quantity": "1",
                      "sorted": "YES",
                      "sortedTime": str(self.current_date_and_time)}
        self.send_email(parameters)
    elif color_sign.mail == 13:
        parameters= {"id": "kinova_sheet",
                      "productid": "CL01",
                      "Product": "Controller",
                      "quantity": "1",
                      "sorted": "YES",
                      "sortedTime": str(self.current_date_and_time)}
        self.send_email(parameters)
    elif color_sign.mail == 23:
        parameters= {"id": "kinova_sheet",
```

```

        "productid":"LM00",
        "Product":"Motor",
        "quantity":"1",
        "sorted":"YES",
        "sortedTime": str(self.current_date_and_time)}
    self.send_email(parameters)
    self.sub.unregister()
def mail_Subscriber(self):
    if self.finger_status == 'open': #Gripper dropped the package
        while not rospy.is_shutdown():
            print("Generating email")
            rospy.init_node('Sub_testing',anonymous=True)
            self.sub = rospy.Subscriber('/pixel_data',cam,self.callback)
            rospy.spin()
    else:
        print("finger status is : 'closed' ")

```

Code snippet 4: Logging package details in spreadsheet and auto-generating emails

```

if(e.parameter["id"] == "kinova_sheet")
{
    var lastRow = sheet.getLastRow();
    data = "product number :" + e.parameter["productid"]
    + "\nProduct Name :" + e.parameter["Product"]
    + "\nNo. of sorted packages :" + e.parameter["quantity"]
    + "\nSorted status :" + e.parameter["sorted"]
    + "\nSorted at time : " + e.parameter["sortedTime"];
    if(data)
    {





```

```
var to = "praveenrajendran1200@gmail.com";  
var message = "Hello, your package has been sorted."  
+ ""+"Check below for more information"  
+ "\n ORDER SUMMARY \n"  
+ data + "\n";  
MailApp.sendEmail(to, "Sorted package details",message);  
}  
}
```

Document Information

Analyzed document	8TH SEMESTER BATCH 1_FINAL REPORT.docx (D102278236)
Submitted	4/20/2021 11:35:00 AM
Submitted by	Kuppan Chetty R M
Submitter email	kuppanc@hindustanuniv.ac.in
Similarity	1%
Analysis address	kuppanc.hits@analysis.orkund.com

Sources included in the report

W	URL: https://www.researchgate.net/publication/327870982_ROS-based_control_for_a_robot_m ... Fetched: 4/20/2021 11:36:00 AM		1
W	URL: https://www.ims.ut.ee/www-public2/at/2020/msc/atprog-courses-magistrit55-loti.05.0 ... Fetched: 12/16/2020 4:16:04 AM		2
W	URL: http://www.techno-press.org/fulltext/j_arr/arr2_2/arr0202001.pdf Fetched: 4/20/2021 11:36:00 AM		1
W	URL: https://ru.b-ok2.org/book/2677479/a6715e Fetched: 4/7/2020 9:36:29 AM		1

Hit and source - focused comparison, Side by Side

Submitted text	As student entered the text in the submitted document.
Matching text	As the text appears in the source.

1/5	SUBMITTED TEXT	25 WORDS	84% MATCHING TEXT	25 WORDS
	ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task		ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task ArticlePDF Available ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task •	
	W	https://www.researchgate.net/publication/327870982_ROS-based_control_for_a_robot_manipulator_with ...		

2/5	SUBMITTED TEXT	9 WORDS	100% MATCHING TEXT	9 WORDS
	ROBOT OPERATING SYSTEM ROS is an open-source software framework		Robot Operating System (ROS) [41] is an open-source software framework	
	https://www.ims.ut.ee/www-public2/at/2020/msc/atprog-courses-magistrit55-loti.05.036-muhammad-usm ...			

3/5	SUBMITTED TEXT	14 WORDS	100% MATCHING TEXT	14 WORDS
	ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task."		ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task	
	<div>W</div> http://www.techno-press.org/fulltext/j_arr/arr2_2/arr0202001.pdf			

4/5	SUBMITTED TEXT	12 WORDS	100% MATCHING TEXT	12 WORDS
	Qian et al., "Manipulation task simulation using ROS and Gazebo," 2014		Qian et al. "Manipulation Task Simulation using ROS and Gazebo".	
	https://www.ims.ut.ee/www-public2/at/2020/msc/atprog-courses-magistrit55-loti.05.036-muhammad-usm ...			

[illegible]