A Project Report on

# “Computer Vision Based Robotic Arm”

Submitted in partial fulfillment of the requirements
of the degree of
**Bachelor of Technology in Mechanical Engineering**
By


Mayuresh Kshirsagar 111810048

Anshul Chandekar 111810094

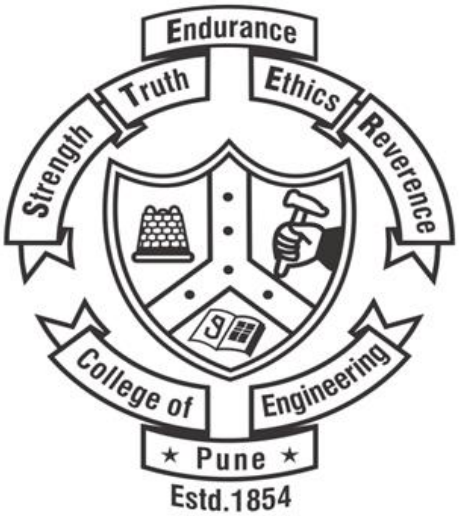
Tejas Ukey 111810115

Ankush Ukey 111810143


Under the Guidance of

**Prof. Dr.S.S.Ohol**

Department of Mechanical Engineering





Department of Mechanical Engineering

**COLLEGE OF ENGINEERING PUNE**


(2021-2022)

# CERTIFICATE


This is to certify that the thesis/dissertation/report entitled '**Computer Vision Based Robotic Arm**' submitted by Mayuresh Kshirsagar (MIS No. 111810048), Anshul Chandekar (MIS No. 11181094), Tejas Ukey (MIS No. 111810115), Ankush Ukey (MIS No. 111810143) in the partial fulfillment of the requirement for the award of the degree of Bachelor of Technology (Mechanical Engineering) of College of Engineering Pune, affiliated to the Savitribai Phule Pune University, is a record of their own work.

| | |
|---|---|
| **Dr. S. S.Ohol** | **Dr. M. R. Nandgaonkar** |
| **Guide** | **Head of the Department** |
| **Department of Mechanical Engineering** | **Department of Mechanical Engineering** |
| **College of Engineering Pune** | **College of Engineering Pune** |

Date:

Place:

**Approval Sheet**

This report entitled

# Computer Vision Based Robotic Arm

By


Mayuresh Kshirsagar 111810048

Anshul Chandekar 111810094

Tejas Ukey 111810115

Ankush Ukey 111810143


is approved for the degree of

**Bachelor of Technology**

of


**Department of Mechanical Engineering**

**College of Engineering Pune**

**(An autonomous institute of Govt. of Maharashtra)**


| **Examiners** | **Name** | **Signature** |
|---|---|---|
| 1. External Examiner | _____ | |
| 2. Internal Examiner | _____ | |
| 3. Supervisor (s) | _____ | |
| | _____ | |

Date:

Place:

# Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

| Name | MIS | Sign |
|---|---|---|
| Mayuresh Kshirsagar | 111810048 | _____ |
| Anshul Chandekar | 111810094 | _____ |
| Tejas Ukey | 111810115 | _____ |
| Ankush Ukey | 111810143 | _____ |

Date:

Place:

# ABSTRACT

Aim of this project is to program a robotic arm for fruit sorting operations. In this project, a fully automated robotic sorting system is proposed which can sort fruits based on their shape, color and a lot of other parameters that our brain uses to distinguish between different objects. A robotic arm is used to pick and place the object in their predefined place using ROS as an interface. This robotic arm is controlled by Raspberry pi. A single camera is used to capture images to be processed using Opencv. YOLO deep learning model is used for processing the robotic arm is used to pick and place the object in their predefined place using ROS as an interface. This robotic arm is controlled by Raspberry pi. A single camera is used to capture images to be processed using Opencv. YOLO deep learning model is used for processing the captured image frame and to determine the class of the object . This system can differentiate between two different classes which are apple and orange . Additional classes could also be added into the system by training datasets of those classes on YOLO architecture. Servo motors are used for the controlled movements of the robotic arm. The robot will serve as a research platform for extended functionality by adding and improving existing software environments and hardware devices

# CONTENT

# List of Figure

# List of Tables

# Chapter 1

# Introduction

Nowadays, robots are increasingly being integrated into working tasks to replace humans, especially to perform repetitive tasks. In general, robotics can be divided into two areas, industrial and service robotics. Service robot is an operational aid which operates semi- or fully autonomously to perform services useful to the wellbeing of humans and equipment, excluding manufacturing operations. These robots are currently used in many fields of applications including office, military tasks, hospital operations, dangerous environment and agriculture. Besides, it might be difficult or dangerous for humans to do some specific tasks like picking up explosive chemicals, defusing bombs or in worst case scenario to pick and place the bomb somewhere for containment and for repeated pick and place action in industries. Therefore a robot can replace humans to do work.

Here in our project, we aim to reduce human effort with minimal human intervention. Today's systems are reducing in size and cost, we interfaced our prototype of hand with a Raspberry Pi, a mini computer.

A robotic arm is a type of mechanical arm, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robot. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement. The links of the manipulator can be considered to form a kinematic chain. The terminus of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand. The end effector, or robotic hand, can be designed to perform any desired task such as welding, gripping, spinning etc., depending on the application. For example robot arms in automotive assembly lines perform a variety of tasks such as welding and parts rotation and placement during assembly. In some circumstances, close emulation of the human hand is desired, as in robots designed to conduct bomb disarmament and disposal. The first robotic arm was developed in the

1950s by a scientist named George Devol, Jr., before which robotics were largely the products of science fiction and the imagination. The development of robotics was slow for a while, with many of the most useful applications being involved with space exploration. The use of robots to aid in industrialization weren‘t fully realized until the 1980s, when robotic arms began to be integrated in automobile and other manufacturing assembly lines. While working in a fashion similar to the human arm, robot arms can still have a much wider range of motion since their design can be purely up to the imagination of their creator. The joint that connects the segments of a robotic arm, for example, can rotate as well as moving like a hinge. The end of the robotic arm designed to actually do the work that it was designed for is known as the end effector, and can be designed for practically any task, for example gripping like a hand, painting, tightening screws and more. These robots can be fixed in one place, for example along an assembly line, or they can be mobile so they can be transported to do a variety of tasks in different places. Autonomous robotic arms are designed to be programmed and then left alone to repeat their tasks independent of human control. Conversely, a robotic arm can also be designed to be.

## 1.1 The Robotic Arm

A typical robotic arm has the following components:

• Links and joints

• Actuators

• Controller

• End-effector

• Sensor

### 1.1.1 Links and joints

A link is considered as a rigid body that defines the relationship between two neighboring joint axes of a manipulator. Manipulators consist of rigid links, which are connected by joints that allow relative motion of neighboring links. The links move to position the end-effector.

### 1.1.2 Actuators

Actuators play the same role the muscles play in the human arm - they convert stored energy into movement. Actuators are used to move a robot’s manipulator joints. The three basic types of actuators currently are used in contemporary robots are pneumatic, hydraulic, and electrical actuators. Pneumatic actuators employ a pressurized gas to move the manipulator joint. They are inexpensive and simple, but their movement is not precise . Hydraulic actuators on the other hand employ a pressurized liquid to move the manipulator joint. They are quite common and offer very large force capability as well as high power-to-weight ratios. The main disadvantages of hydraulic actuators are their accompanying apparatus (fluid pumps and storage tanks) and problems with fluid leaks. Electric motor-driven actuators provide smoother movements, can be controlled very accurately, and are very reliable. However, these actuators cannot deliver as much power as hydraulic actuators of comparable mass. Nevertheless, for modest power actuator functions, electrical actuators are often preferred. The various types of electric motors used as actuators for robotic applications are direct current (dc) motors, stepper motors and servo motors .

### 1.1.3 Controller

The controller is the main device that processes information and carries out instructions in a robot. It is the robot's 'brain' and controls the robot's movements. It is usually a computer of some type which is used to store information about the robot and the work environment and to store and execute programs which operate the robot. It contains programs, data algorithms, logic analysis and various other processing activities which enable the robot to perform its intended function. Most robots incorporate computer or microprocessor-based controllers. These controllers perform computational functions and interface with and control sensors, grippers, tooling, and other peripheral equipment . Most modern servo motor actuators are designed and supplied around a dedicated controller module from the same manufacturer. Such controllers are usually developed around microcontrollers and common examples are the Pololu Mini Maestro 12- Channel USB Servo Controller and Lynxmotion's SSC-32 servo controller.

### 1.1.4 End-effector

In robotics, an end-effector is the device at the end of a robotic arm, designed to interact with the environment. The exact nature of this device depends on the application of the robot. Typical functions of the end-effector include grasping, pushing and pulling, twisting, using tools, performing insertions, welding and various types of assembly activities. Thus, the major types of robot end-effectors are:

• grippers - Grippers are the most common type of end-effectors. They can use different gripping methods (such as vacuum or use of fingers).

• material removal tools - These include cutting, drilling and deburring tools installed as robot tools.

• welding torches - Welding is a very popular robotic application. Welding torches have thus become very efficient end-effectors that can be controlled in a sophisticated way for optimized welding.

• tool changers - Tool changers are used when many different end effectors need to be used in sequence by one robot. They are used to standardize the interface between the robot flange and the base of the tool. They can be manual or automatic Surgical robots have end-effectors that are specifically manufactured for performing surgeries

### 1.1.5 Sensor

Sensors are physical devices that enable a robot to perceive its physical environment in order to get information about itself and its surroundings. They allow the robotic arm to receive feedback about its environment. They can give the robot a limited sense of sight and sound. The sensor collects information and sends it electronically to the robot controller. One use of these sensors is to prevent collision with obstacles (collision detection and avoidance). Another common application of sensors is the use of vision sensors to give a pick and place robotic arm the capability to differentiate between items to choose and items to ignore..

## 1.2 Applications of the Robotic Arm

According to Angelo (2007), robots now play a prominent and indispensable role in modern manufacturing. The use of robots has led to increased productivity and improved product quality. It has helped to keep manufacturing industries viable in high-labor cost countries. Today, many products we buy have been assembled or handled by a robot. The most common manufacturing robot is the robotic arm. Robotic arms are typically used in industry. Repetitive autonomous robots perform one task repeatedly based upon predetermined movements and specifically located objects. Start and stop commands are determined by position, acceleration, deceleration, distance, and direction. More complex actions are executed based upon sensor processing. If object orientation or position is unknown, arms are often paired with machine vision and artificial intelligence to identify the object and subsequently control the arm. Contemporary applications of the robotic arm range from doing an accurate and reliable job of spray-painting an automobile on an assembly line, to robotic surgery. The da Vinci surgical robot uses robotic arms equipped with scalpels and other instruments to more accurately target surgical objectives, allowing surgeons to use smaller, less invasive incisions.

Other applications of the robotic arm include:

• welding - arc welding, laser welding, plasma welding, welding automation, resistance welding and plasma cutting;

• material handling - dispensing, injection molding, machine loading, pick and place, packaging, parts transfer and press tending;

• painting automation;

• fiberglass cutting;

• assembly operations;

• foundry operations.

In the automobile industry, robotic arms are used in diverse manufacturing processes including assembly, spot welding, arc welding, machine tending, part transfer, laser processing, cutting,

grinding, polishing, deburring, testing, painting and dispensing. Robots have proved to help automakers to be more agile, flexible and to reduce production lead times. They can be programmed to perform precise intricate duties at much faster speeds than any human could be expected to. They also have the advantage of being able to withstand exposure to conditions adverse to humans such as extreme heat and presence of dangerous chemicals in the air

## 1.3 Project Aim

Aim of this project is to program a robotic arm for fruit sorting operations. In this project, a fully automated robotic sorting system is proposed which can sort fruits based on their shape, color and a lot of other parameters that our brain uses to distinguish between different objects. A robotic arm is used to pick and place the object in their predefined place using ROS as an interface. This robotic arm is controlled by Raspberry pi. A single camera is used to capture images to be processed using Opencv. YOLO deep learning model is used for processing the robotic arm is used to pick and place the object in their predefined place using ROS as an interface. This robotic arm is controlled by Raspberry pi. A single camera is used to capture images to be processed using Opencv. YOLO deep learning model is used for processing the captured image frame and to determine the class of the object . This system can differentiate between two different classes which are apple and orange . Additional classes could also be added into the system by training datasets of those classes on YOLO architecture. Servo motors are used for the controlled movements of the robotic arm. The robot will serve as a research platform for extended functionality by adding and improving existing software environments and hardware devices.

## 1.4 Approach

As for any project, the first step was clearly defining the problem statement. A well-defined description of the problem, to be solved by the project, was instrumental in defining the scope of the project and giving way to the further steps to be undertaken. The problem statement is as follows:

The arm uses six servo motors for movement along six degrees of freedom (DOF), and the gripper is designed to handle one package weighing 75-150 gms. The work envelope is a hemisphere of 45 cm radius, with the center being the base center. The arm is used to demonstrate sorting of fruits. After adequately defining the problem statement as above, a strategy plan was made, based on the project scope. A comprehensive description of scope of project comprised of the following aspects:

- Calculate torque for defined payload
- Select appropriate servos
- Select robotic arm
- Forward and inverse kinematics of the robotic arm using MATLAB
- Create object detection model using appropriate deep learning architecture
- Create URDF model
- Create Robot launch files using Moveit
- Use Gazebo to create robot simulations
- Use moveit commander library to get the joint angles of the robot
- Feed those joint angle to arm library to control the hardware

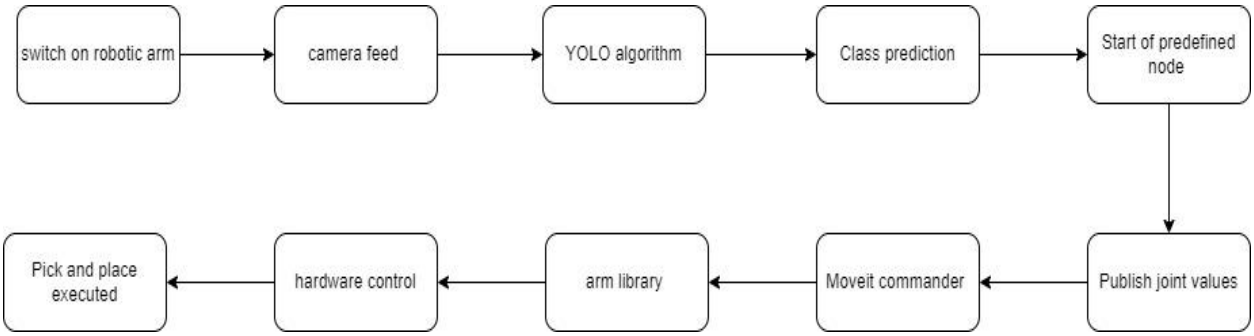



**fig 1.1**: Project approach

# Chapter 2

# Literature Review

## 2.1 Control of a robotic arm: Application to on-surface 3D-printing,M. R. de Gi[1]

3D-printing is a largely developing technique in manufacturing. The general aim of the research is to print 3D-structures on curved surfaces. The technique allows accurate and quick 3D-prints, requiring a 6-DOF manipulator for printing on curved surfaces. The UR5 robotic arm was used which has 6-DoF.The purpose of this research is grow a control strategy for printing on a double curved surface.

In 3D printing succeeding layers are printed on top of other till required product is formed.

The general aim of this research is to print 3D-structures on curved surfaces. The Drop-on-Demand print technique is used which has a print head with multiple nozzles. For this a 6-DOF robotic arm UR5 is used. The UR5 can be commanded with ROS via URControl and MATLAB via URControl and C-API.The robotic arm can be controlled at a maximum sampling frequency of 125 Hz.

Robotic arm UR5

The arm was controlled by MATLAB as other programs are developed in MATLAB hence ensuring fast communication between programs.The robotic arm can be controlled at a maximum sampling frequency of 125 Hz.the robot will be controlled by commanding the velocity for the joints.

The Workspace of the UR5

A kinematic model can be derived for UR5 using the Denavit-Hartenberg parameters. To avoid singular configurations velocity is checked so as to not exceed maximum velocity.Self-collision is avoided by restricting some predetermined angles.

Modeling of the UR5

To reduce drift in position of joints and end-effector, a control loop having a controller around internal controller UR5 is constructed. For designing this controller Model Predictive Control

(MPC) is used for better performance. Under what conditions the UR5 can be regarded as a Single-Input Single-Output (SISO) system instead of a Multiple-Input Multiple-Output (MIMO) system. In this research is chosen to divide the whole system in six separate systems. The whole UR5 is simplified to six joints and it is assumed that these joints will not interfere with each other. The cross correlation between all the joints was researched and it is justified to have six SISO systems.

Inverse Kinematics

The used method for the Inverse Kinematics of the UR5 is the technique where the inverse of the Jacobian is used. This method is very broad and can be used with any robotic arm.As velocities will be integrated to the position , there will be small variation in the solution which can be reduced by keeping step size small.

Control of the UR5

In this thesis MPC was used to control the UR5 as a final solution.Model predictive control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. The MPC controller is used without constraints.

## 2.2 Angeliki Topalidou-Kyniazopoulou(2017)[2]

The thesis introduces a method that can reduce implementation time and hence required computational resources of motion planning for a 6 DoF's robotic arm used for independent pick and place operations in a static environment. For this purpose MoveIt library and ROS framework were used .As MoveIt can do forward and backward calculations upto 6 DoF only ,a 6DoF arm was used. They were concerned with two robots namely UR 10 Robot and Fanuc M-20iA/20M Robot. Both can be run manually and autonomously. The middleware used was ROS(Robot Operating System) for implementation and its libraries were used.For motion planning and control Moveit was used.It has MoveIt Setup Assistant, a GUI for configuring a new robot. The aim was efficient motion planning and control of a robotic arm aimed at doing pick and place tasks in a stationary environment with a high success rate and minimum computing time.

## 2.3 Khasim A. Khana, Revanth R. Kondab and Ji-Chul Ryu[3]

Robotics has been a growing field.When commercial robots are used,to increase user-customized functionality, the robots can be accessed in the form of GUI (Graphical User Interface) or APIs (Application Program Interfaces). The GUI provided by manufacturers is often not capable of fully exploiting the abilities of the robot and dealing directly with the APIs may be unwieldy.To overcome these inadequacies, middleware packages have been developed.ROS is an open source middleware package that works between multiple platforms and performs OS-like functionality. Quigley et al. (2009) first developed ROS with the design goals of "Peer to Peer, Multi-lingual, Tools-based, Thin, Free and Open-sources." ROS is gaining popularity for simple as well as complex systems.

The objective was to command a 7-DOF manipulator arm,having 7 servo motors (Dynamixel), by ROS having external vision camera system and illustrating balancing of a ball on plate-type end effector.The ball had a camera(Sony PlayStation Eye) for feedback control of balancing, with the tracking algorithm written in C++ using OpenCV libraries.The plate had two axis linearized motion.The developed system can help students to learn ROS with control theories in robotics.The employed system can be divided into three modules:vision system package, manipulator system package, and planning and control package.As ROS is open source, drivers and drivers for Dynamixel and USB cameras are available.The object tracking to get the position of the ball is done by a series of image processing steps. Manipulator package is created to make connections between the actuators.The planning and control node brings all the components of the system together and forms a bridge between the vision tracking node and the manipulator system node.

**System modeling and control:**

1.Equations of motion

First, the equations of motion of a ball-on-beam is derived using the Lagrangian method. The two plate motions are rotations about X- and Y-axes. the equations of motion for each rotational motion are calculated.

2.Linearization around the center of the plate

The aim is to balance the ball around the center of the plate. Hence the rotation angles will be very small. The equations of motion are further simplified by applying linear approximation.

3.Controller design

The linear equations derived previously are now used for controller design.
The paper introduced a 7-DOF manipulator arm using ROS interface, having a vision tracking system.This system is such that the components communicate with each other to receive the state feedback from the vision system and Dynamixel servo motors, and then compute and send control commands to perform prescribed control tasks. This system can be used for more complicated tasks having more joint controls and also using robotics control theories used for learning ROS programming.

## 2.4 Robotic Jaw for Object Sorting using Raspberry pi[4]

Shraddha More et. al. made use of the robotic arm which is capable of detecting and placing in specified objects. The robotic arm detects the pre-defined object and segregates them based on the RGB color. The goal of the system was to detect the object based on color and sort them accordingly.The paper presents to an apparatus and method for classifying in and sorting small-sized objects.A robot that has the capability to pick a pre-specified object and place it in separate divisions based on color was made.Raspberry pi was used for color detection of the object and image processing . The ultrasonic sensor calculates the distance using its transmitter and receiver. The web camera senses the object and gives the outcome to the pi. After taking the results from the raspberry pi and the sensor, the robotic arm moves accordingly and picks and sorts the objects in their pre-specified position.The results were better than previously existing systems. This can be controlled automatically and used for industrial purpose.It can identify the specific color of the object and pick it and place it in a specified area as desired by the user with the help of RGB values by detecting the color of the object.
Shraddha More. " Robotic Jaw for Object Sorting using Raspberry pi" IOSR Journal of Computer Engineering (IOSR-JCE) 21.2 (2019): 01-05.

## 2.5 Mohammad Monirujjaman Khan et al.(2020)[5]

Planning and execution of a robotic vision system using interactive GUI application was introduced by this paper.Minimum operator training requirement was kept in mind.Objects are filtered on the basis of colour, shape and size. Generally separate algorithms are used for the three.For determining the robotic arm’s joint positions for picking the object inverse kinematic

algorithm is used.The joint coordinates are forwarded to a microcontroller that sets the arm's joint angle at each position.IN this paper OpenCV was used to create a computer vision sorting system for the robotic system thus multiple algorithms for colour, shape and size were not required.The filtering along the three parameters is done by employing a Hue-Saturation-Value (HSV) mode. The images captured by the camera are in the Red, Green, and Blue (RGB) color space and must be converted to the HSV. Once converted to HSV, the image is binarized such that the desired color is allotted a value of one, and the remainder of the image is allotted a value of 0, and so a contour can be drawn concerning the desired object. The contour created during color sorting was approximated to polygon. Objects were sorted according to shape by approximating polygon to objects placed in the workspace. To determine size, area of approximated polygon is used to identify targeted objects. The application enables users to decide their desired object, which is then picked up and placed by a robotic arm into the target location.

Muhatasim Intisar[1], Mohammad Monirujjaman Khan[1], Mohammad Rezaul Islam[1] and Mehedi Masud[2]

## 2.6 Integration of an industrial robot manipulator in ROS to enhance its spatial perception capabilities,Dornbirn, November 30, 2020[6]

This thesis attempts to tackle collision problems by equipping such a robot with extra sensor hardware for perceiving environmental objects. The robot used within this thesis is a KUKA LBR iiwa 7 R800. The goal is a robot capable of moving in an unseen environment without colliding with obstacles nearby.

During the research, different sensor options used by others were presented. The research also covered robots moving in cramped areas as well as algorithms and simulation topics. Software platforms and libraries used for the implementation

After evaluating the available sensor options, multiple infrared sensors were directly installed onto the robot manipulator. The extra sensors and the robot were integrated into the ROS middleware to create an application capable of sensing the robots' environment and plan collision-free paths accordingly.

## 2.7 Fruit sorting robot based on color and size for an agricultural product packaging system[7]

To overcome the problem of dressing manpower in the agriculture sector, this paper introduces the application of a 4-DOF fruit sorting robot based on color and size in a packaging system. The sorting is done by image processing where color is identified by HSV analysis, and the diameter is known in the grayscale image and setting the thresholding.This project helped in creating a robot imitating the human arm for the sorting system. The fruit sorted were red and green tomatoes and red and green grapes..The fruits are classified into small green and red, and big green and red. Experiments were conducted to show the effectiveness of the proposed method. The robot applied in this study is a 4 DOF arm robot manipulator where each joint is moved by a servo motor.The sorting robot system is equipped with a camera attached near to the initial fruit position and a proximity sensor attached to the end-effector to sense the distance between the end-effector and the fruit.The end-effector designed is a gripper to pick and place the fruit according to its size and color.The main controller is an ATMega2560 responsible for moving the servo motors, and the processor for image processing is Raspberry Pi.

Fruit sorting algorithm design

The image processing is conducted in two steps, deciding the colors and recognizing the size. The color recognition is using HSV analysis, and the size recognition is conducted by calculating the diameter of the object/fruit in the grayscale image and setting the thresholding.The color analysis is by converting the RGB image to HSV.Therefore, as the IP camera in the packaging system captures the RGB images from the online video, the Raspberry PI converted those captured images into HSV value. The HSV values are used to determine the x-y coordinate position of the object and decide the color of the object between red and green. The red detection is for reddish color, starting from orange to very dark red.The next step of fruit detection is the determine the size of the fruit.The size is decided by using ellipsoidal shape achieved from the grayscale image.The diameter of the fruit gives the size of the fruit.

The experiment was conducted ten times for each fruit. The effectiveness of sorting, picking, and placing the fruit is 80% for red and 90% for green tomatoes, 70% for red, and 60% for green

grapes.The experimental result shows that the arm robot is applicable for a fruit sorting robot based on the proposed method in this study.

## 2.8 Literature Review Summary

Table 2.1 Literature Review Summary

| Sr No | Author's Name | Key Findings |
|---|---|---|
| 1 | M. R. de Gier(April 6, 2015) | 3D printing using using a 7-DOF robotic arm was studied |
| 2 | Angeliki Topalidou-Kyniazopoulou(2017) | The pre-computed motion plans in a cell grid reduces planning time for a specific manipulation task |
| 3 | Khasim A. Khana, Revanth R. Kondab and Ji-Chul Ryu( 2018) | Successful demonstration of 7-DOF manipulator arm using ROS interface, having a vision tracking system |
| 4 | Alexander Dengg( 2020) | Sensors and robot were integrated into the ROS for sensing the robots' environment and plan collision-free paths |
| 5 | Shraddha More(2019) | Detection of pre-defined objects and segregating them based on the RGB color. |
| 6 | Mohammad Monirujjaman Khan et al. (2020) | Design and implementation of a robotic vision system operated using an interactive Graphical User Interface (GUI) application |
| 7 | Tresna Dewi(2020) | Image processing for sorting based on the color and size with HSV analysis |

# Chapter 3

# Robotic Manipulator

## 3.1 Parts

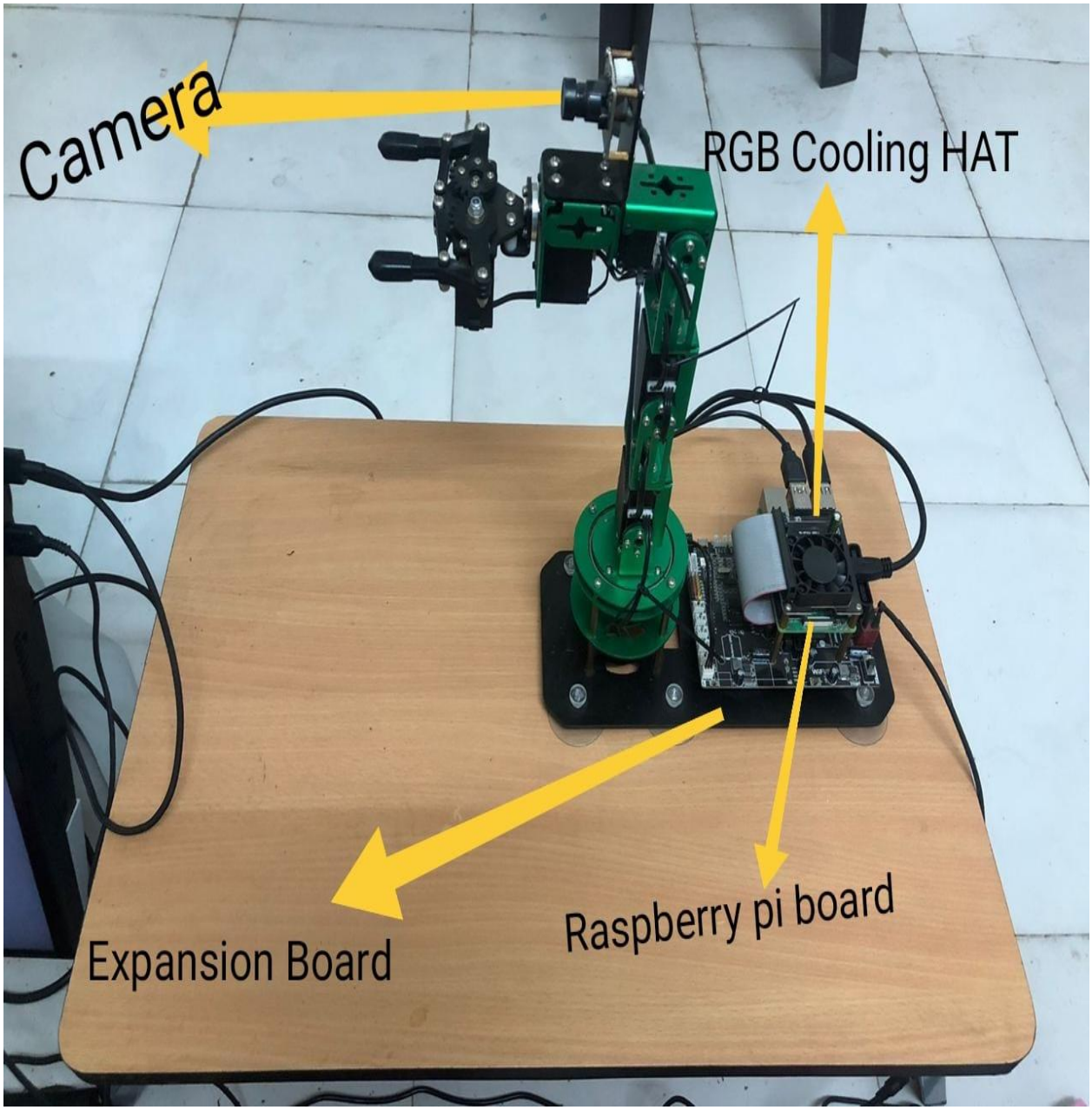



**fig.3.1** Robotic Arm

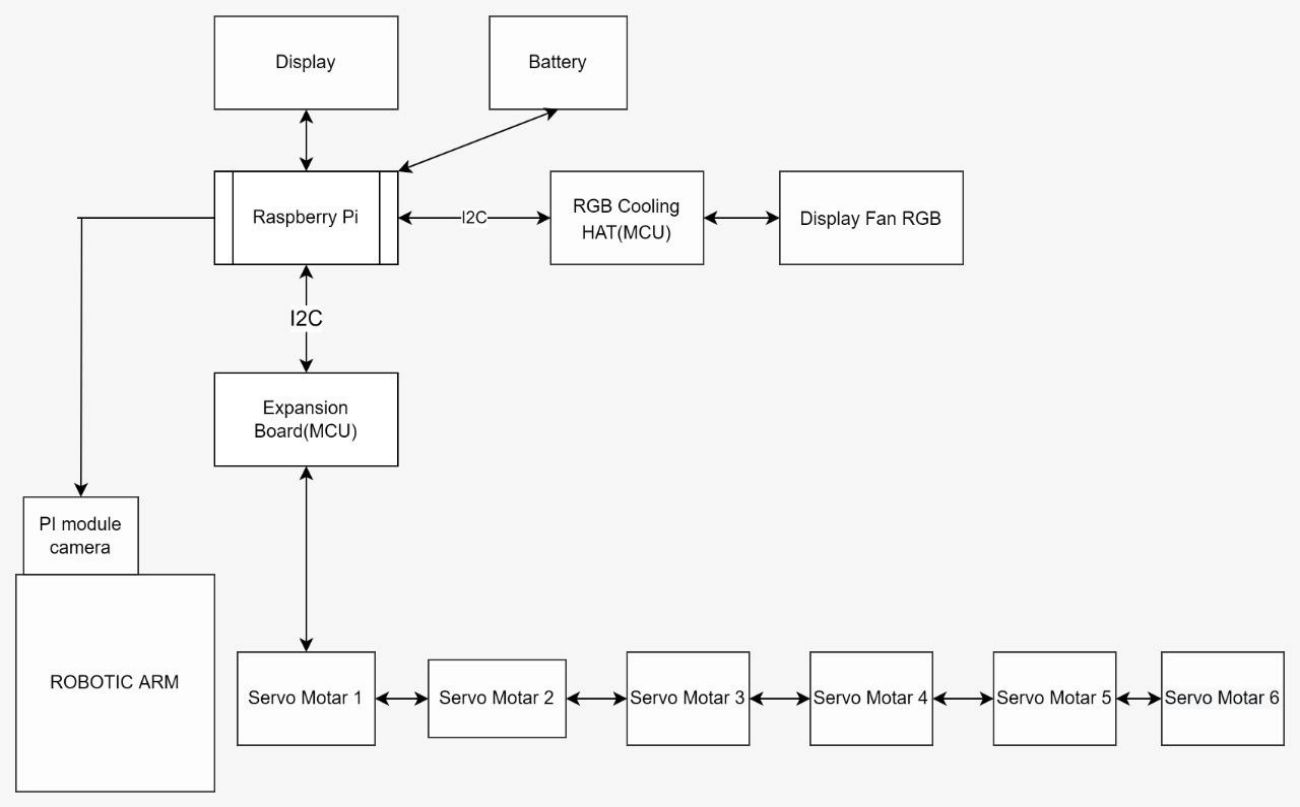



**fig.3.2** Diagrammatic Representation (hardware connection)

### 3.1.1 Servo motors

Serial bus smart servo motor with bracket

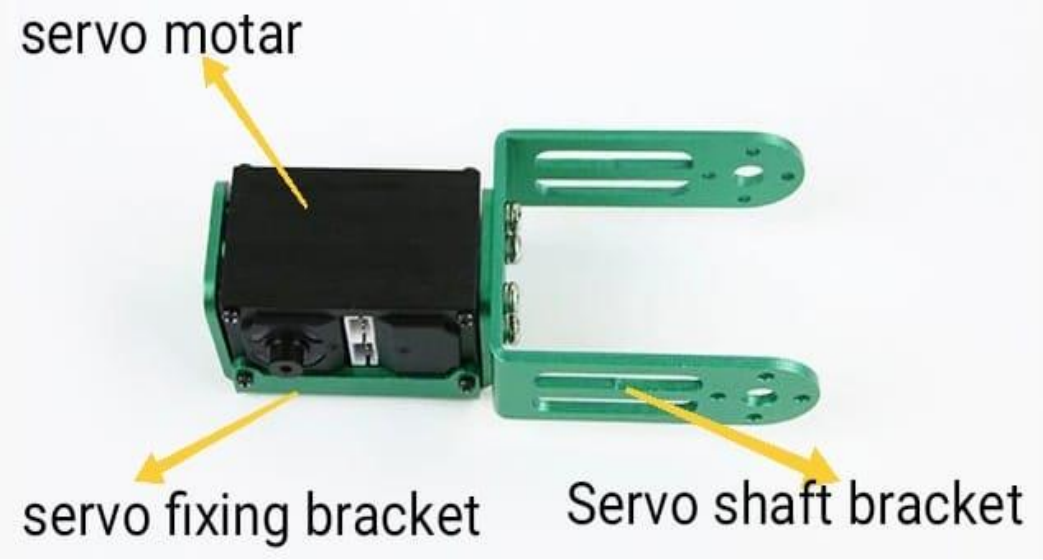



**fig 3.3** Servo Motor

Servo Size- 44.37*23.06*12mm

Servo shaft bracket- 43*21*57mm

Servo fixing bracket - 50*30*28mm

Servo interface output voltage- 6.3V(max 7.4V)

Servo interface output current-1.2A (max 2A)

A servomotor is a rotary actuator that allows for precise control of angular position. It consists of a motor coupled to a sensor for position feedback, through a reduction gearbox. The servomotor is actually an assembly of four things: a normal DC motor, a gear reduction unit, a position-sensing device (usually a potentiometer—a volume control knob), and a control circuit.

This serial bus smart servo possesses 15KG torque and adopts all-metal gears inside, strong and durable. It can be controlled by serial port commands, multiple servos can be cascaded at the same time, which can help us create robotic arms and bionic robots. Compared with ordinary servos on the market, it has the characteristics of reading back the angle, adjusting the rotation speed, and modifying the identification ID number. With Yahboom servo driver board, it is easy to drive this servo. And we will also provide PC debugging software and some programs

### 3.1.2 RASPBERRY PI and Extension board

Raspberry Pi is a low-cost, small - sized device that is connected into a PC screen, and uses a general mouse and keyboard. It really is a small object which enables individuals of different ages to discover computer technology as well as gain knowledge on how to program in languages such as Scratch and Python. An SD card inserted in the board slot acts as the Raspberry Pi hard drive. This is operated by USB and you can connect the video output to a standard Amplifier TV box, an extra contemporary display, or perhaps a Pc to use the HDMI port. The Pi consists of various versions and here the version used is Raspberry 4 Model B. It consists of 40 vertical GPIO pins. Devices can be connected to these GPIO pins Based on the high and low command of the user, the particular device moves.

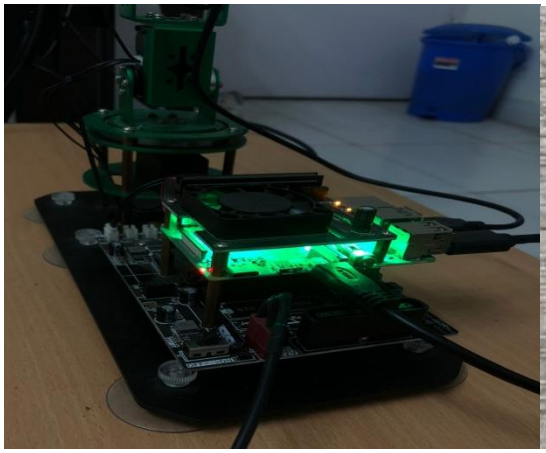
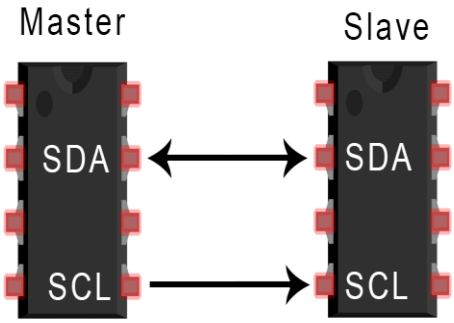


**fig.3.4** Raspberry Pi and Expansion Board

**fig.3.5** Raspberry Pi

These two components communicate through I2C (Inter-integrated Circuit) bus and the functionality of the robot is dependent on the continuous exchange of information. The “brain” of the mobile robot platform is Raspberry Pi which is a small single-board computer that can be programmed to manage the robot operation (mapping, navigation, obstacle detection and avoidance/transport). The Raspberry Pi is not directly connected to the servo. We connect it to the chip on the expansion board through the i2c on the Raspberry Pi, and then drive the servo by sending commands to the chip on the expansion board.

### 3.1.2.1 I2C communication protocol



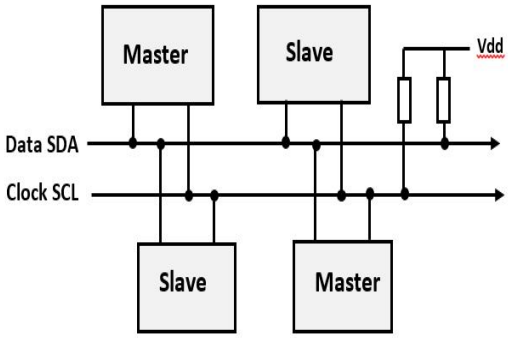




**fig 3.6** I2C master and slave

**fig.3.7** I2C data transmission

The information exchange between RPiI and Expansion Board takes place through the I2C synchronous communication protocol. Raspberry Pi is the "Master" component that initiates the data transfer to the "Slave" component SDA (Serial Data) – The line for the master and slave to send and receive data. SCL (Serial Clock) – The line that carries the clock signal.I²C uses only 2 wires to transmit data between devices to connect master to multiple slaves or multiple master to multiple slaves

### 3.1.3 Rgb cooling HAT

Board can be directly plugged onto Raspberry Pi board via its 40 pin interfaces, and it has an expanded 40 pin header for connecting to other devices without affecting the use of GPIO ports on Pi board. Furthermore, the place for the OLED screen is reserved on the board to display, CPU usage, hard disk space, memory, and CPU temperatur IP address in real-time.

### 3.1.4 Other parts

- Ethernet cable: Connected to Extension board
- Connecting wires: Connected to Extension board
- Keyboard & Mouse: Connected to Extension board
- Camera Module: For running the computer vision algorithms and to carry out the handover process, a camera was used in this project that could interface with the Raspberry Pi 4B microcontroller. The camera is mounted on the manipulator and is used to detect the presence of a hand, as and when required.
- Install the Raspberry pi camera by inserting the cable into the Raspberry pi. The cable slots into the connector situated between the Ethernet and HDMI ports, with the silver connector facing the HDMI ports

## 3.2 Torque calculation

We have accounted for the force of gravity when calculating our torque requirements. However, we assumed that the links have no weight. In reality they do.

- Neglecting link weight for easy calculation and camera module weight also neglected

To calculate the torque requirement, we start at the end effector (i.e. gripper) of the robotic arm and then work our way to the base of the robot. We need to account for all the pieces of the robot that are impacted by gravity:

- Weight of the link
- Weight of the joint
- Weight of the object being lifted (e.g. the box below)

**Table 3.1**. Link Lengths

| Sr. No | Length no | Length of links |
|---|---|---|
| 1 | L1 | 57 |
| 2 | L2 | 71.77 |
| 3 | L3 | 71.77 |
| 4 | L4 | 73.47 |
| 5 | L5 | 33.07 |
| 6 | L6 | 0 |

Torque Required By a Servo Motor = (Torque Due to Force of Gravity on Links and Payload) …..(1)

Let the Mass of box be 250g and calculate torque at each mortar

- The force of gravity acting on the box is equal to $m_{box}$ * g. (where m means mass).
- The torque due to the box is therefore $L_j$ * $m_{box}$ * g, j is the length of each link accordingly

Servo Shaft bracket -17.55g  Servo fixing bracket- 16.74g

Servo Motor- 50g

**Motar 6** L6 =0 ,T6=0 so torque produced will be zero

**Motar 5** L5=33.07mm

Torque =Box weight + motar 6 weight …..(2)

=250g*9.81(m/s2)*33.07mm+66.74mm*9.81(m/s2)*33.03

=102729.5568 **T=0.102Nm**

**Motor 4** L4=73.47mm

Torque =Box weight + motar 6 weight+motar 5 weight ….(3)

=250g*9.81(m/s2)*(33.07+73.47)mm+66.74mm*9.81(m/s2)*(33.07+73.47)+

=73.47*9.81*66.74

**T=0.379145Nm**

**Motor 3** L3=71.77mm

Torque =Box weight + motar 6 weight+motor 5 weight+motar4 weight ….(4)

=250g*9.81(m/s2)*(33.07+73.47+71.77)mm+66.74mm*9.81(m/s2)*(33.07+73.47+71.77)+(73.47+71.77)*9.81*66.74+73.47*9.81*66.74

**T =0.695Nm**

**Motar 2** L2=71.77mm

Torque =Box weight + motor 6 weight+motor 5 weight+motar 4 weight+ motar 3 weight

=250g*9.81(m/s2)*(33.07+73.47+71.77+71.77)mm+66.74mm*9.81(m/s2)*(33.07+73.47+71.77+71.77)+(73.47+71.77+71.77)*9.81*66.74+(73.47+71.77)*9.81*66.74+71.77*9.81*66.74

**T=1.016020Nm**

**Motor 1 L1**=57mm

Torque=Box weight + motor 6 weight+motor 5 weight+motor 4 weight motar 3 weight+motor 2 weight

=250g*9.81(m/s2)*(33.07+73.47+71.77+71.77+57)mm+66.74mm*9.81(m/s2)*(33.07+73.47+71.77+71.77+57)+(73.47+71.77+71.77+57)*9.81*66.74+(71.77*2+57)*9.81*66.74+(71.77+57)*9.81*66.74 +57*9.81*66.74 **T=1.38577Nm**

**Table 3.2** Torque values for each servo motor

| Servo motor | Torque value(Nm) |
|---|---|
| 1 | 1.3857 |
| 2 | 1.0162 |
| 3 | 0.695 |
| 4 | 0.379 |
| 5 | 0.102 |
| 6 | 0 |

## Conclusions

After calculating the torque on each motor while assuming the mass of 250g box is taken(because for fruit sorting the maximum load of apple and orange is between 70-150g) ,the maximum torque obtained is 1.3857Nm .So we choose the servo motors of each torque capacity greater than 1.3857Nm .

Therefore the servo motor used has 1.5Nm torque(15kgf.cm) ,so when we are using apples and oranges whose normal mass is approx (70 -150g),it can be easily lifted by the motor.

## 3.3 Kinematics Analysis

Kinematics is the motion description of a rigid body. [8]The kinematic chain is a grouping of links connected by joints, as illustrated in Figure 1. In the kinematic chain, the number of DoF(degree of freedom) is equal to the number of joints. Maintaining a strong connection between the two joints is called the kinematics function of a link, Homogeneous transformation is commonly used as a definition of kinematics model particularly for chain mechanism, as described below. **D-H (Denavit–Hartenberg)** model[12] is the most popular mode for developing the kinematic model, matrix is used to represent the pose (position and orientation) of one body with respect to another.

$$^{0}_{n}T = T_1 * T_2 * T_3 \dots \dots . T_i \dots . T_n \quad (1)$$

Where n is the total no. of links, Ti is link transformation from the ith joint, and $^{0}_{n}T$ is the final position for the end-effector with respect to base.

There are two main types of kinematic models: forward kinematics and inverse kinematics. Several models are developed for kinematic modeling, but the D-H (Denavit–Hartenberg) model is the most popular mode

### 3.3.1 Forward Kinematics

Forward kinematics is the study of the manipulator to find out its tip or end-effector position and orientation by using joint values of the manipulator.The length of each links and angles of each joint is given, through that, position of any point(x,y,z) can be found

### Analytical approach

The first step of performing the forward kinematics is to label link lengths. The transformation matrix for a link i is described as follows:

Now taking transformation matrix from each link as shown in eq. (2)

$$A_I = \begin{bmatrix} cos\theta i & -sin\theta i * cos\alpha i & sin\theta i * sin\alpha i & \alpha i cos\theta i \\ sin\theta i & cos\theta i * cos\alpha i & -cos\theta i * sin\alpha i & \alpha i sin\theta i \\ 0 & sin\alpha i & cos\alpha i & di \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

A1 is the transformation matrix $T^0{}_1$

$$T_1^0 = \begin{bmatrix} cos\theta 1 & -sin\theta 1 * cos\alpha 1 & sin\theta 1 * sin\alpha 1 & \alpha 1 cos\theta 1 \\ sin\theta i & cos\theta i * cos\alpha i & -cos\theta i * sin\alpha i & \alpha 1 sin\theta 1 \\ 0 & sin\alpha 1 & cos\alpha 1 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

We will perform from base to end effector ,transformation based on each theta

Transformation matrix from base to end-effector is

$$T_6^0 = T_1^0 * T_2^1 * T_3^2 * T_4^3 * T_5^4 * T_6^5 \quad (4)$$

$$T_6^0 = \begin{bmatrix} r11 & r12 & r13 & Px \\ r21 & r22 & r23 & Py \\ r31 & r32 & r33 & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$T_6^0 = \begin{bmatrix} R06 & P06 \\ 0 & 1 \end{bmatrix} \quad (6)$$

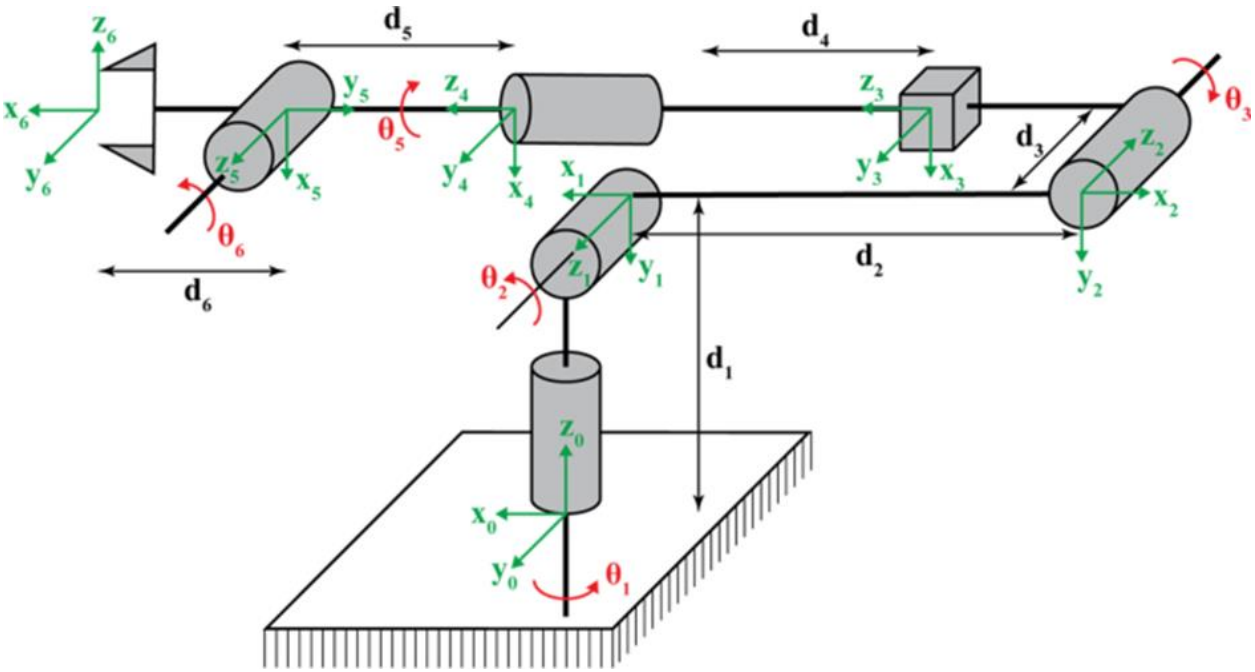



**fig. 3.8** Forward Kinematics

### 3.3.2 Inverse Kinematics

Here the length of each link and position of some points(x,y,z) is given and the angle of each joint is needed to find

**Analytical approach**

This is according to Pieper's approach [17], in which the manipulator is divided to analyze the inverse kinematics [10][14]

An important assumption!

The first three joints are entirely responsible for POSITIONING the end effector, and any additional joints are responsible for ORIENTING the end effector

- Step 1: Draw a kinematic diagram of only the first 3 joints, and do inverse kinematics for position
- Step 3: Find the inverse of the RO_3 matrix
- Step 4: Do forward kinematics on the last three joints and pull out the rotation part, R3_6
- Step 5: Specify what you want the rotation matrix RO_6 to be

- Step 6: Given a desired X, Y, and Z position, solve for the first three joints using the inverse kinematics equations from Step 1
- Step 7: Plug in those variables and use the rotation matrix to solve for the last three joints

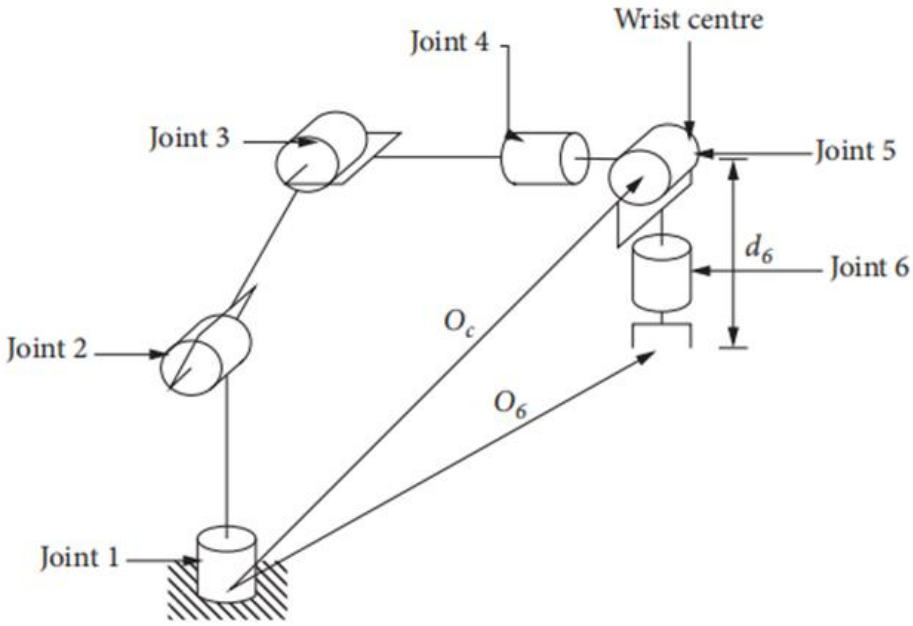



**fig. 3.9** Inverse Kinematics

### 3.3.3 Using MATLAB Simulation

The GUI Based Graphical interface development for Forward & Inverse Kinematics analysis of 6 DOF Robot Using MATLAB

There are 7 Steps to make this work

1. **Creating the New GUI - Interfaces**

Write guide in MATLAB

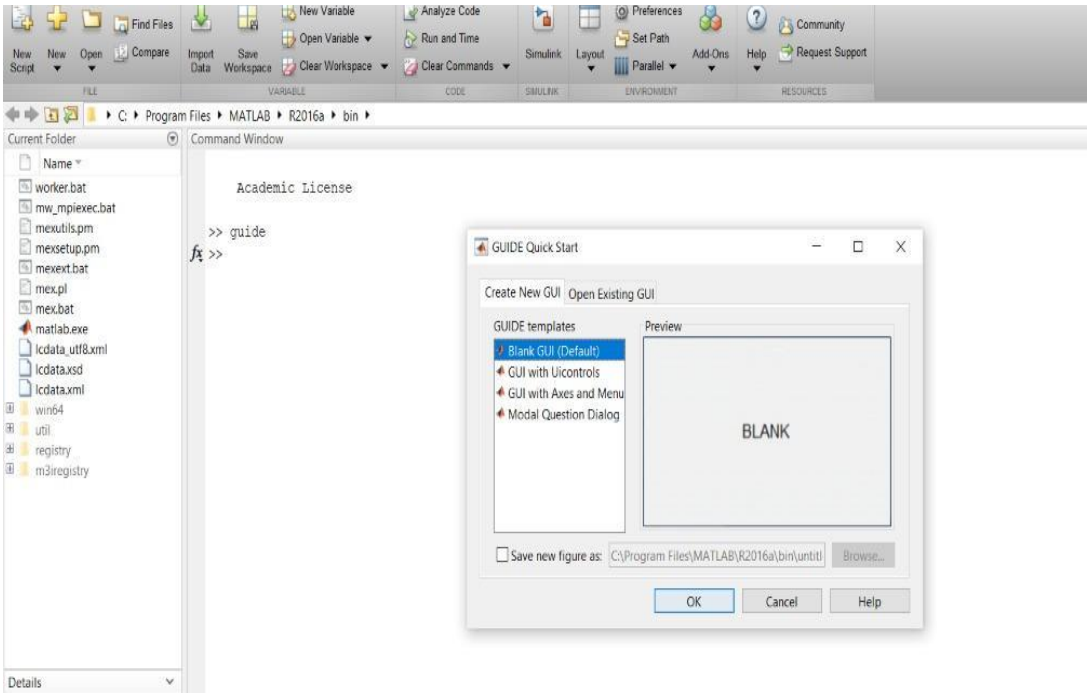



**fig 3.10** MATLAB

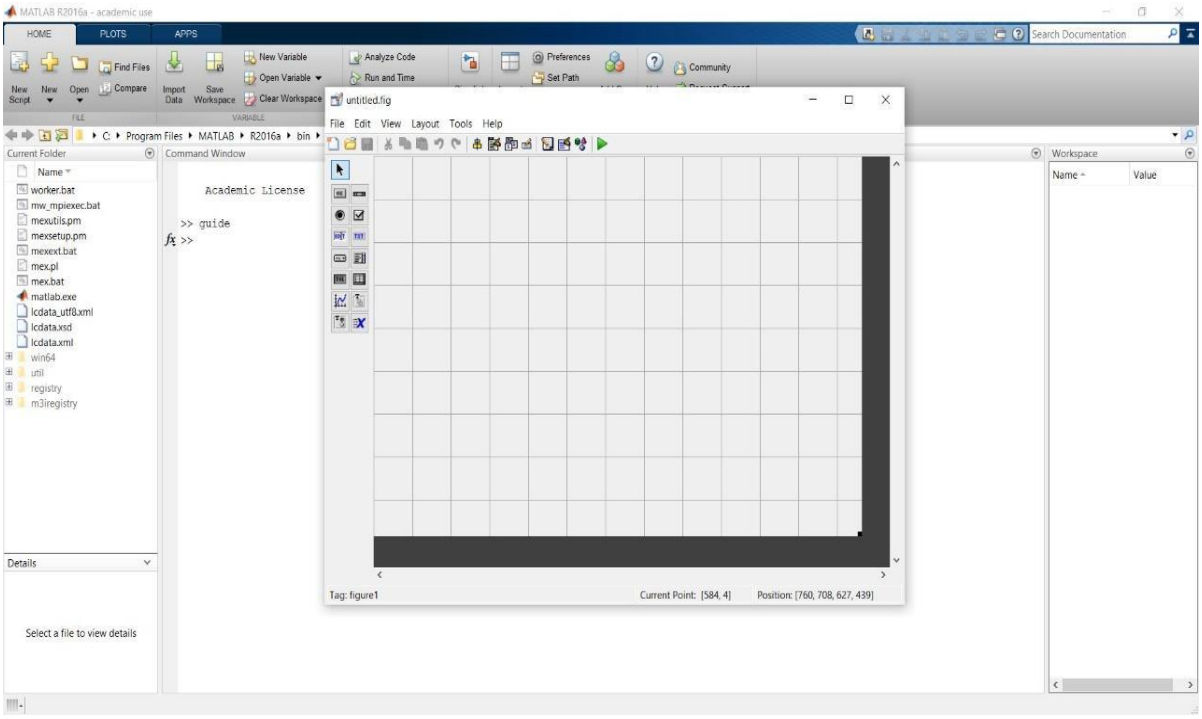



**fig3.11** GUI interface

## 2. Adding GUI Components

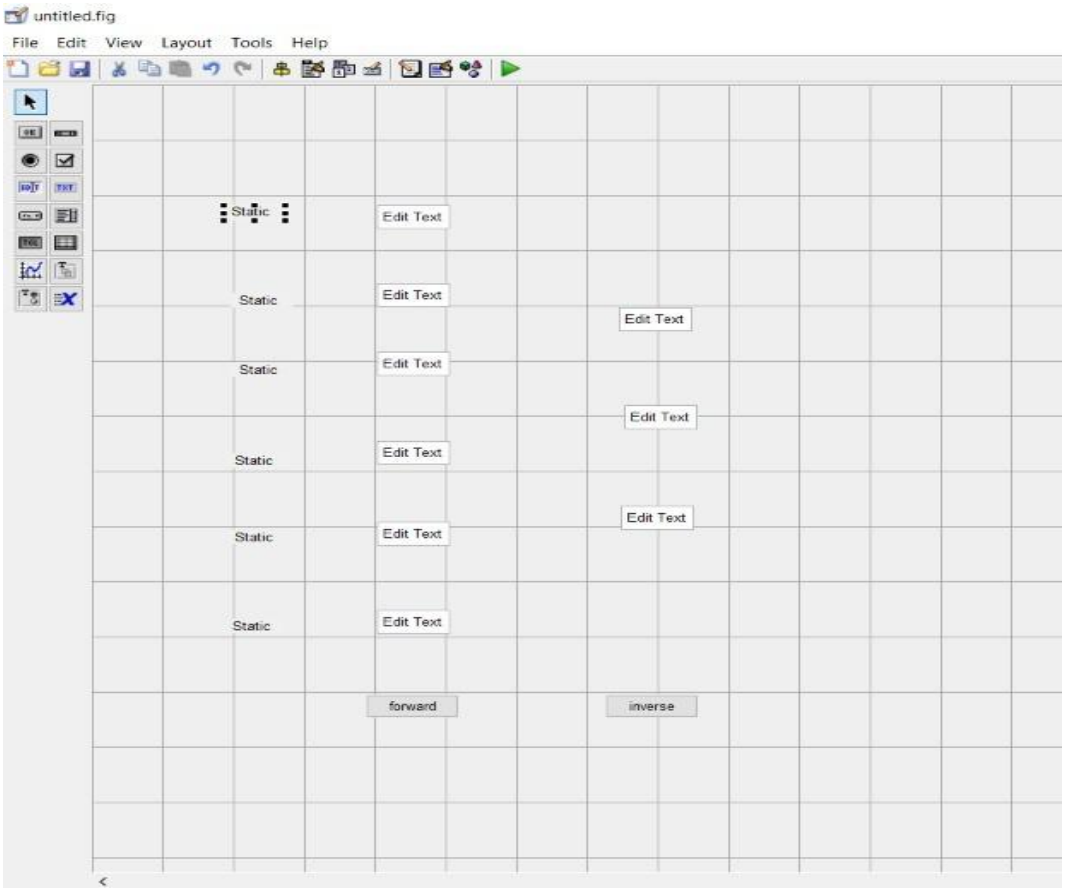



**fig3.12** Creating components in GUI interface

**3. Setting Control Properties**: Change the properties of each boxes accordingly in properties and change names

**4. Forward Kinematics**: Code inside the callback function writing the code on the basis on link length on table 3.1

```
Th_1 = str2double(handles.Theta_1.String)*pi/180;
Th_2 = str2double(handles.Theta_2.String)*pi/180;
Th_3 = str2double(handles.Theta_3.String)*pi/180;
Th_4 = str2double(handles.Theta_4.String)*pi/180;
Th_5 = str2double(handles.Theta_5.String)*pi/180;
Th_6 = str2double(handles.Theta_6.String)*pi/180;

L_1 = 57;
L_2 = 71.77;
```

```
L_3 = 71.77;
L_4 = 77.47;
L_5 = 33.07;
L_6 = 0;
L(1) = Link([0 L_1 0 pi/2]);
L(2) = Link([0 0 L_2 0]);
L(3) = Link([0 0 L_3 0]);
L(4) = Link([0 0 L_4 -pi/2]);
L(5) = Link([0 0 L_5 pi/2]);
L(6) = Link([0 L_6 0 0]);

Robot = SerialLink(L);
Robot.name = 'Six_R_Robot';
Robot.plot([Th_1 Th_2 Th_3 Th_4 Th_5 Th_6]);

T = Robot.fkine([Th_1 Th_2 Th_3 Th_4 Th_5 Th_6]);
handles.Pos_X.String = num2str(floor(T.t(1)));
handles.Pos_Y.String = num2str(floor(T.t(2)));
handles.Pos_Z.String = num2str(floor(T.t(3)))
```

**5. Running Forward Kinematics**-Using MATLAB simulation, with the help of GUI Based Graphical interface development, the end effector position has been found out by taking random value of joint angles

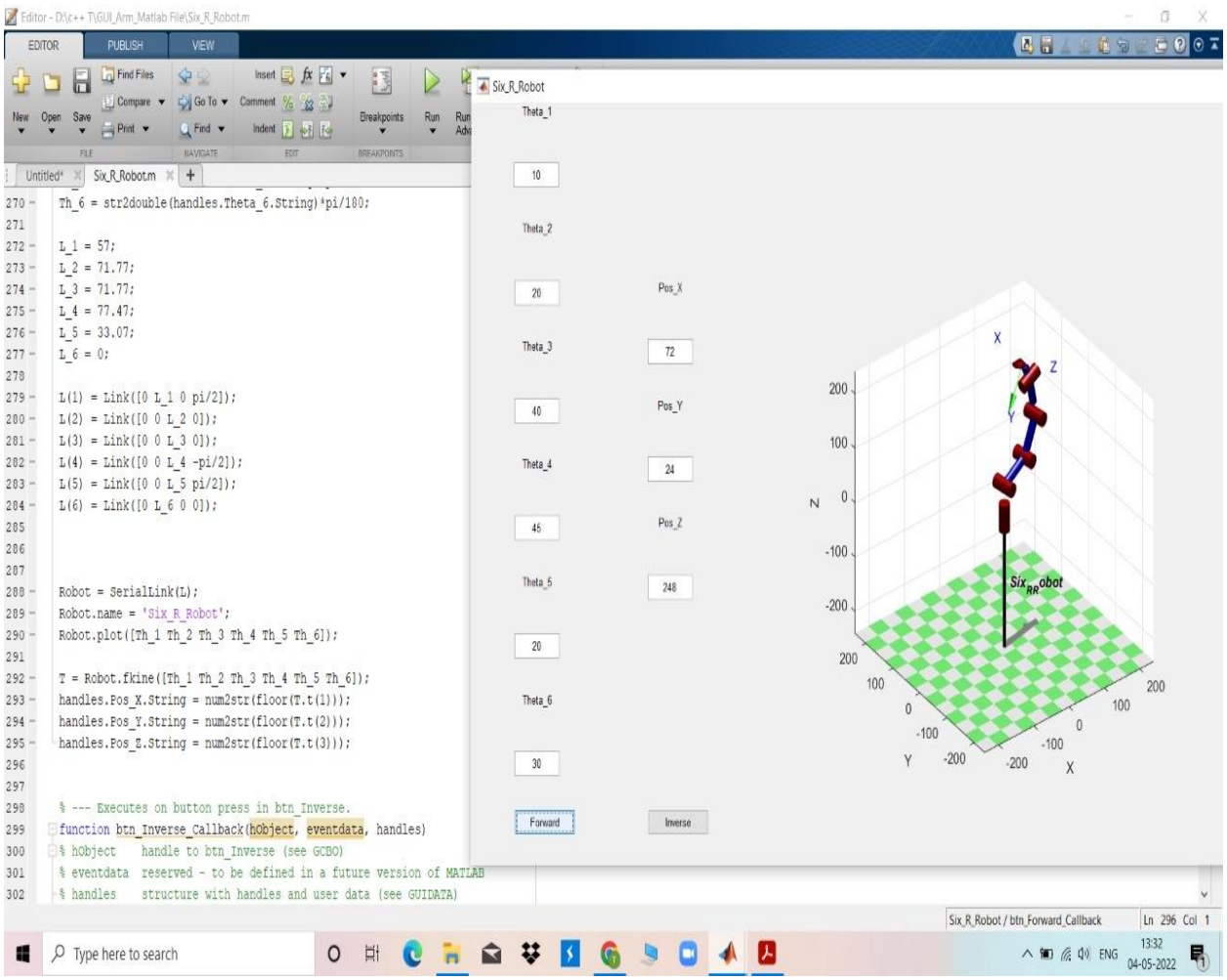



**fig.3.13** Forward kinematic using MATLAB

## 6. Inverse Kinematics Code

```
PX = str2double(handles.Pos_X.String);
PY = str2double(handles.Pos_Y.String);
PZ = str2double(handles.Pos_Z.String);

L_1 = 57;
L_2 = 71.77;
L_3 = 71.77;
L_4 = 77.47;
L_5 = 33.07;
L_6 = 0;

L(1) = Link([0 L_1 0 pi/2]);
L(2) = Link([0 0 L_2 0]);
```

```
L(3) = Link([0 0 L_3 0]);
L(4) = Link([0 0 L_4 -pi/2]);
L(5) = Link([0 0 L_5 pi/2]);
L(6) = Link([0 L_6 0 0]);
Robot = SerialLink(L);
Robot.name = 'Six_R_Robot';

T = [ 1 0 0 PX;
      0 1 0 PY;
      0 0 1 PZ;
      0 0 0 1];
  J = Robot.ikine(T,[0 0 0],'mask',[1 1 1 0 0 0])*180/pi;
handles.Theta_1.String = num2str(floor(J(1)));
handles.Theta_2.String = num2str(floor(J(2)));
handles.Theta_3.String = num2str(floor(J(3)));
handles.Theta_4.String = num2str(floor(J(3)));
handles.Theta_5.String = num2str(floor(J(3)));
handles.Theta_6.String = num2str(floor(J(3)));

Robot.plot(J*pi/180);
```

**7. Running Inverse Kinematics Code:**Using MATLAB simulation, with the help of GUI Based Graphical interface development, the joint angles have been found out by taking the random value of end-effector position.

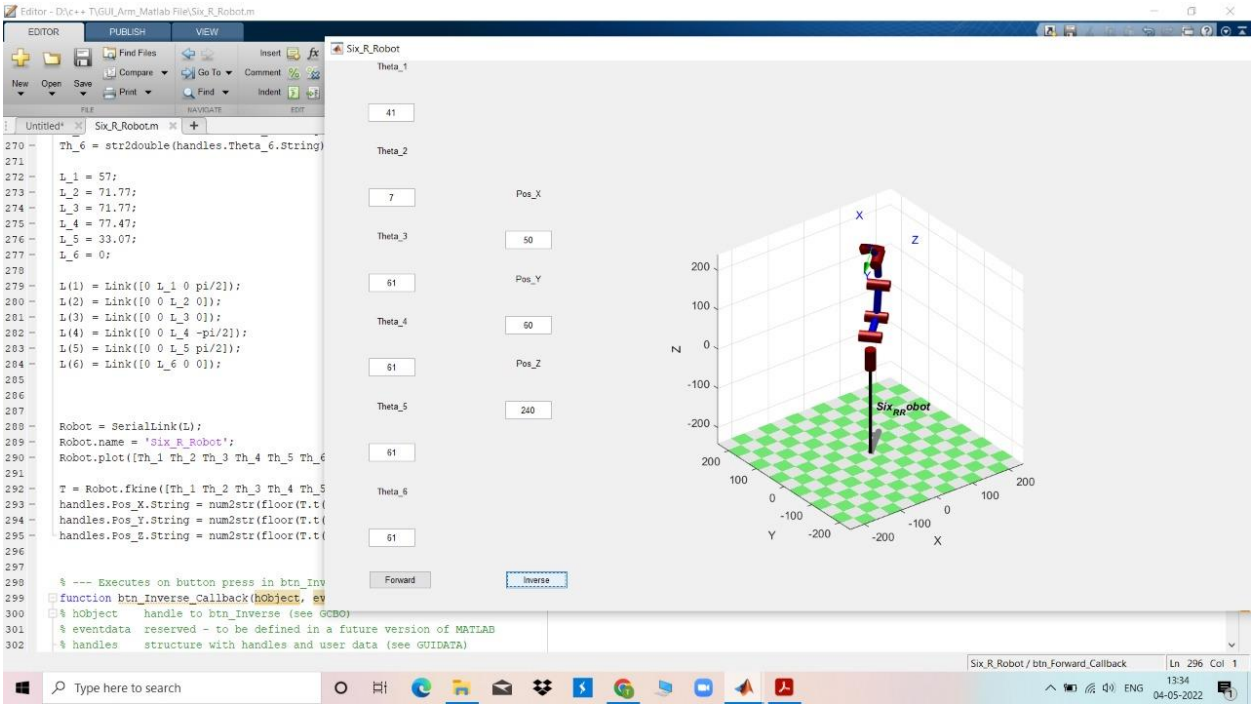



**fig.3.14** Inverse kinematics using MATLAB

# Chapter 4

# Deep Learning Architecture

## 4.1 Workflow

Following figure describes our workflow for creating object detection model

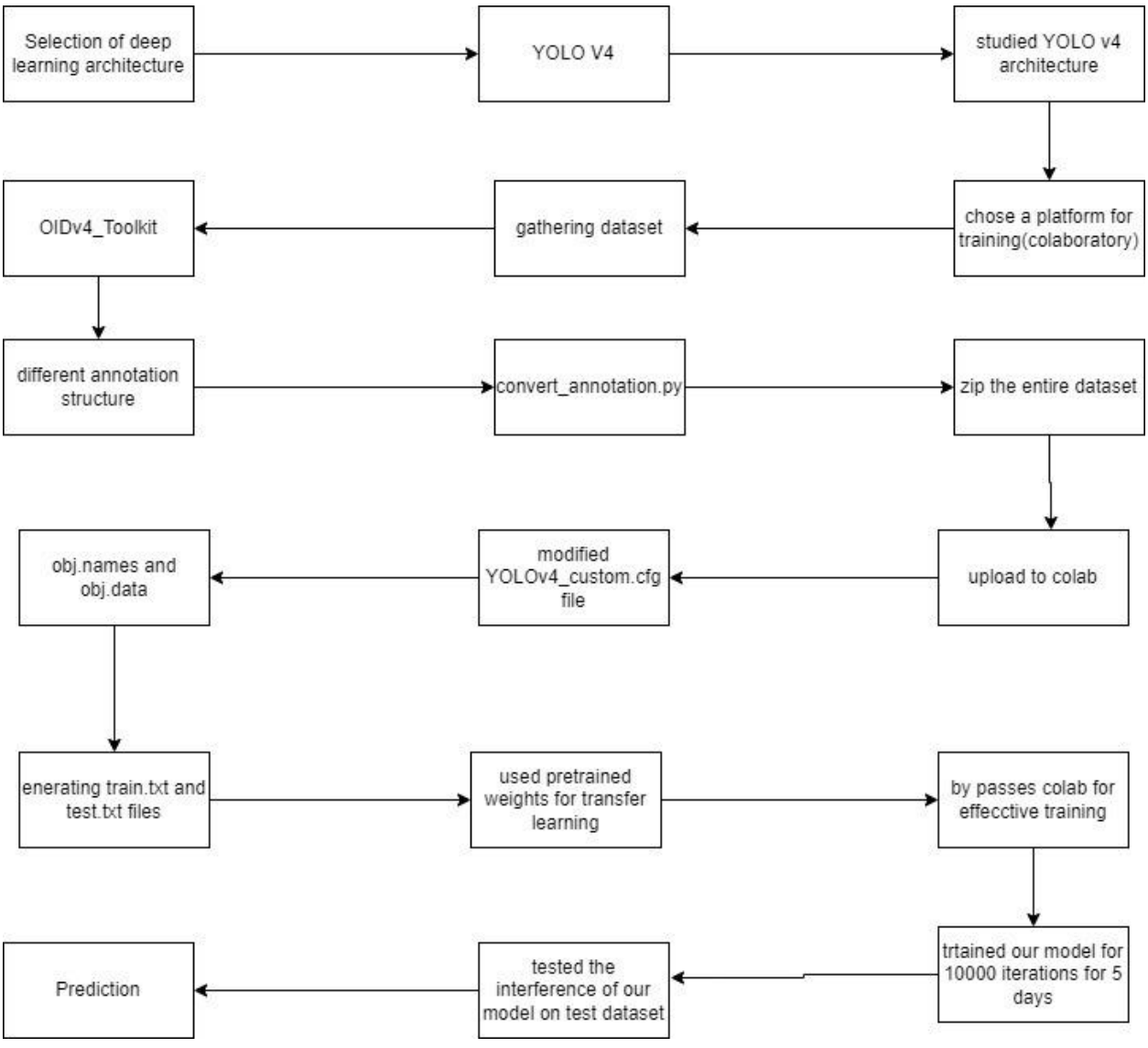



**Fig 4.1**: Workflow

## 4.2 Selection of deep learning architecture

YOLO is a futuristic recognizer that has faster FPS and is more accurate than available detectors like EfficientDet, ATSS, ASFF and CenterMask. The detector can be trained and used on a conventional GPU which enables widespread adoption. Following chart of average precision vs frames per second best describes the performance of YOLO with respect to other available detectors.

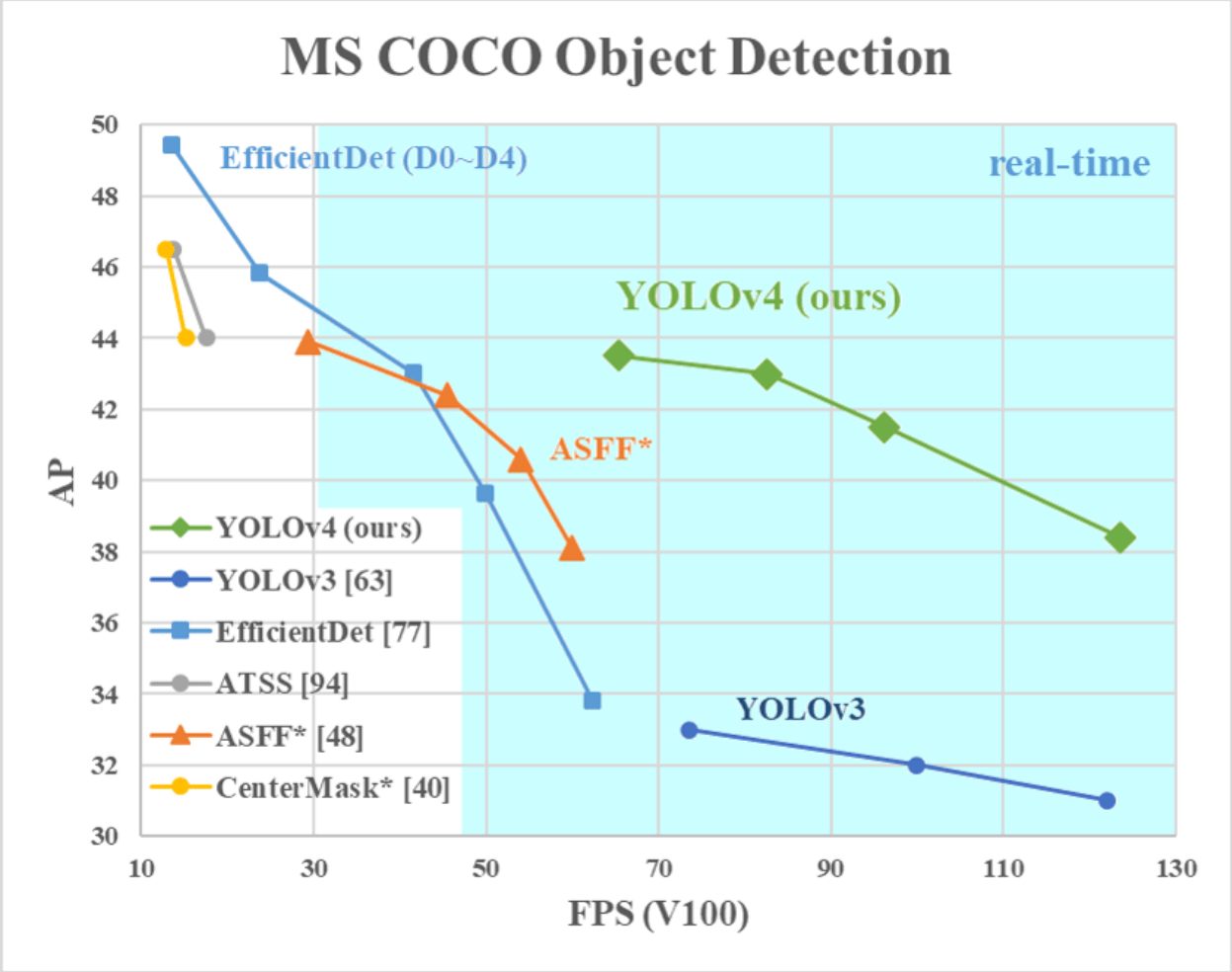



**fig 4.2** AP vs. FPS chart

We also studied a comparative study of object detection algorithms [13]. The result analysis of a system or algorithm is based upon some set of parameters. Most common parameters are performance, time taken, resources needed, accuracy etc. which are undertaken in almost all analysis. Where the performance is parameter indicating how well the algorithm does perform. Time taken is the parameter which represents the time taken by the algorithm to output the result. Resources needed are defined as the amount of resources required by the algorithm. Accuracy

defines the promising factor of the algorithm which is the percentage of the correct output generated by the algorithm.

**Table 4.1:** Comparison result of different models

| Sr. No | Model | Latency | mAP | FPS | Real Time |
|---|---|---|---|---|---|
| 1 | R-CNN | High | ~ 60 | <1 | NO |
| 2 | Fast R-CNN | Medium | ~ 70 | <1 | NO |
| 3 | Faster R-CNN | Medium | ~ 70 | 7 | NO |
| 4 | YOLO | Low | ~ 60 | 46 | YES |

Our robot is required to perform real time detection for fruit sorting operations, so YOLO is the best suitable algorithm for object detection that we could incorporate in our project.

## 4.3 YOLO Algorithm

### 4.3.1 Definition

YOLO or You Only Look Once, is a popular real-time object detection algorithm. YOLO combines what was once a multi-step process, using a single neural network to perform both classification and prediction of bounding boxes for detected objects. As such, it is heavily optimized for detection performance and can run much faster than running two separate neural networks to detect and classify objects separately. It does this by repurposing traditional image classifiers to be used for the regression task of identifying bounding boxes for objects.. Although the subsequent iterations feature numerous improvements, the basic idea behind the architecture stays the same. YOLOv1 referred to as just YOLO, can perform faster than real-time object detection at 45 frames per second, making it a great choice for applications that require real-time detection. It looks at the entire image at once, and only once hence the name We Only Look Once, which allows it to capture the context of detected objects. This halves the number of false-positive detections it makes over R-CNNs which look at different parts of the image separately.

Additionally, YOLO can generalize the representations of various objects, making it more applicable to a variety of new environments.

### 4.3.2 Working

YOLO is based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions S×S, like so:

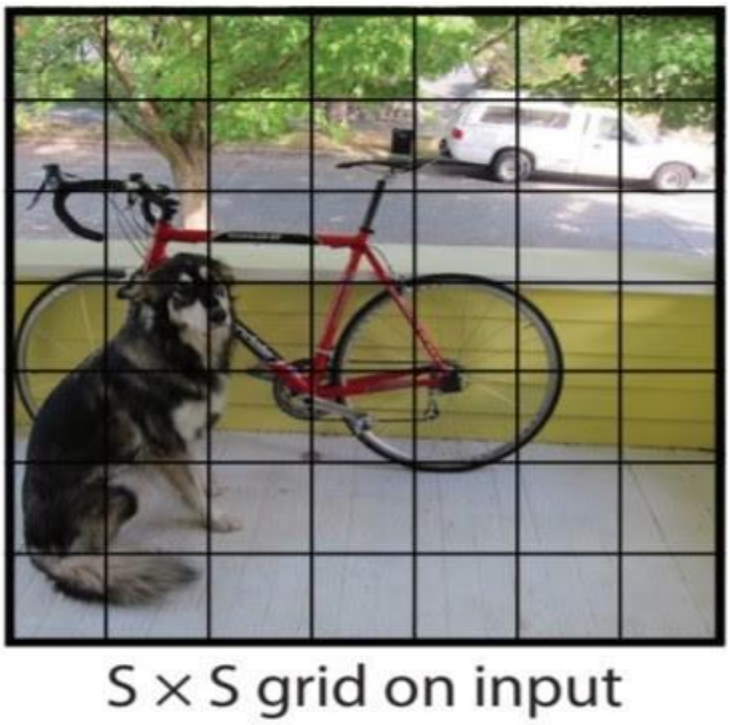



**fig 4.3:** Input image

The cell in which the center of an object, for instance, the center of the dog, resides, is the cell responsible for detecting that object. Each cell will predict B bounding boxes and a confidence score for each box. The default for this architecture is for the model to predict two bounding boxes. The classification score will be from `0.0` to `1.0`, with`0.0` being the lowest confidence level and `1.0` being the highest; if no object exists in that cell, the confidence scores should be `0.0`, and if the model is completely certain of its prediction, the score should be `1.0`. These confidence levels capture the model's certainty that there exists an object in that cell and that the

bounding box is accurate. Each of these bounding boxes is made up of 5 numbers: the x position, the y position, the width, the height, and the confidence. The coordinates `(x, y)` represent the location of the center of the predicted bounding box, and the width and height are fractions relative to the entire image size. The confidence represents the IOU between the predicted bounding box and the actual bounding box, referred to as the ground truth box. The IOU stands for Intersection Over Union and is the area of the intersection of the predicted and ground truth boxes divided by the area of the union of the same predicted and ground truth boxes.

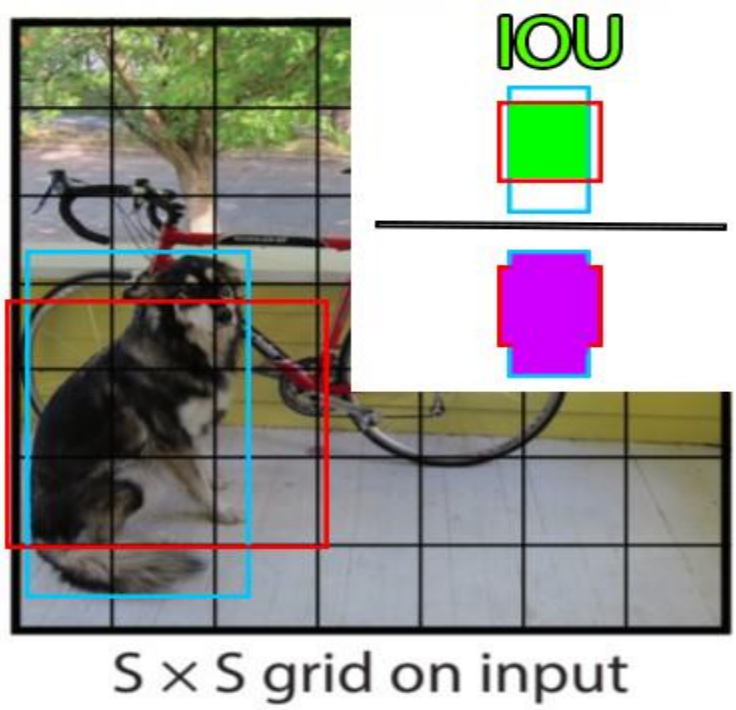



**fig 4.4:** Intersection over Union

The area of intersection of the ground truth and predicted box in green divided by the area of the union of the two boxes, in purple. This will be between 0 and 1, 0 if they don't overlap at all, and 1 if they are the same box. Therefore, a higher IOU is better as it is a more accurate prediction.

In addition to outputting bounding boxes and confidence scores, each cell predicts the class of the object. This class prediction is represented by a one-hot vector length C, the number of classes in the dataset. However, it is important to note that while each cell may predict any number of bounding boxes and confidence scores for those boxes, it only predicts one class. This is a limitation of the YOLO algorithm itself, and if there are multiple objects of different classes in one grid cell, the algorithm will fail to classify both correctly. Thus, each prediction from a

grid cell will be of shape ***C + B * 5***, where ***C*** is the number of classes and ***B*** is the number of predicted bounding boxes. ***B*** is multiplied by 5 here because it includes ***(x, y, w, h, confidence)*** for each box. Because there are ***S × S*** grid cells in each image, the overall prediction of the model is a tensor of shape ***S × S × (C + B ∗ 5)***.

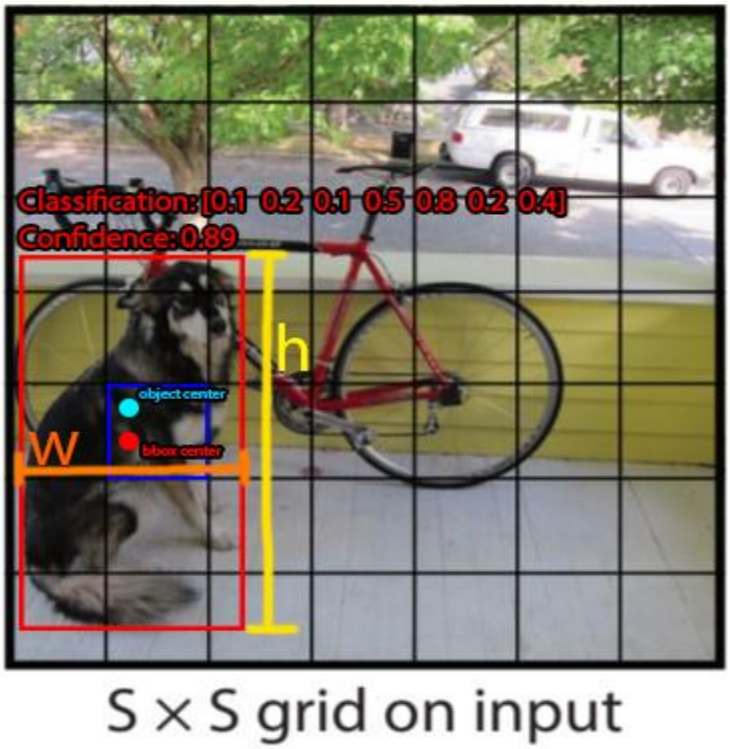



**fig 4.5:** Confidence vector

Here is an example of the output of the model when only predicting a single bounding box per cell. In this image, the dog's true center is represented by the cyan circle labeled 'object center'; as such, the grid cell responsible for detecting and bounding the box is the one containing the cyan dot, highlighted in dark blue. The bounding box that the cell predicts is made up of 4 elements. The red dot represents the center of the bounding box, ***(x, y)***, and the width and height are represented by the orange and yellow markers respectively. It is important to note that the model predicts the center of the bounding box with widths and heights rather than top left and bottom right corner positions. The classification is represented by a one-hot, and in this trivial example, there are 7 different classes. The 5th class is the prediction and we can see that the model is quite certain of its prediction. Keep in mind that this is merely an example to show the

kind of output that is possible and so the values may not be accurate to any real values. Below is another image of all the bounding boxes and class predictions that would actually be made and their final result.

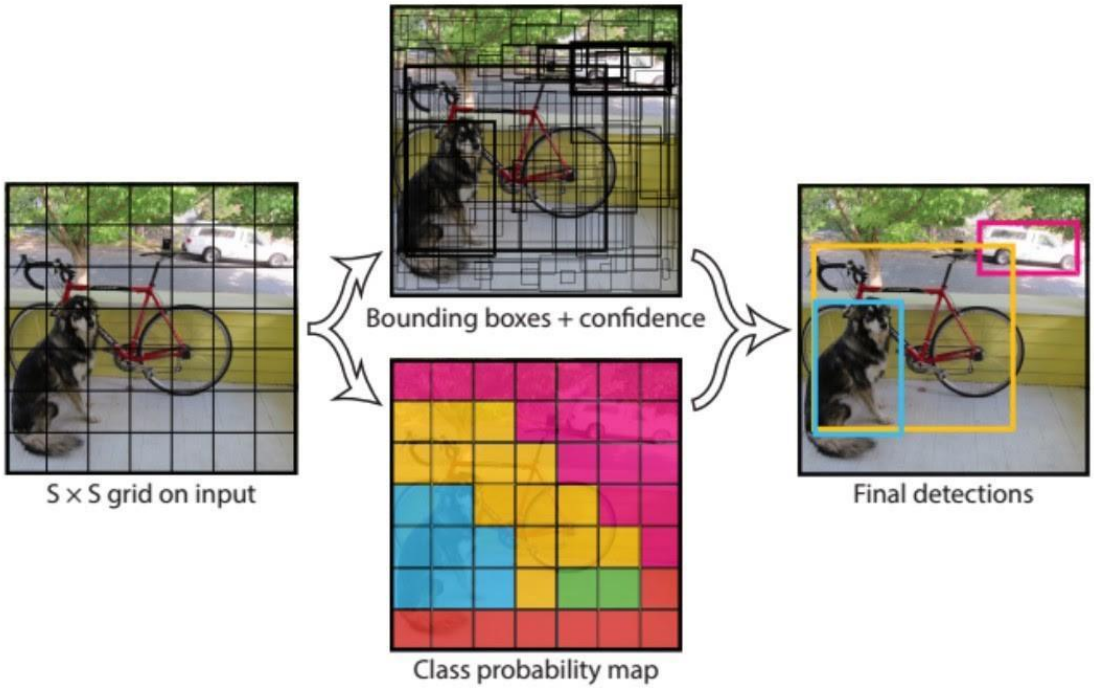



**fig 4.6:** Final detection

### 4.3.3 Architecture

The YOLO model is made up of three key components: the head, neck, and backbone. The backbone is the part of the network made up of convolutional layers to detect key features of an image and process them. The backbone is first trained on a classification dataset, such as ImageNet, and typically trained at a lower resolution than the final detection model, as detection requires finer details than classification. The neck uses the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates. The head is the final output layer of the network which can be interchanged with other layers with the same input shape for transfer learning. As discussed earlier, the head is

an $S \times S \times (C + B * 5)$ tensor and is $7 \times 7 \times 30$ in the original YOLO research paper with a split size S of 7, 20 classes C, and 2 predicted bounding boxes B. These three portions of the model work together to first extract key visual features from the image then classify and bound them.

### 4.3.4 Training

The backbone of the model is pre-trained on an image classification dataset. The original paper used the ImageNet 1000-class competition dataset and pre-trained 20 out of the 24 convolution layers followed by an average-pooling and fully connected layer. They then add 4 more convolutions to the model as well as 2 fully connected layers as it has been shown that adding both convulsions and fully connected layers increases performance. They also increased the resolution from $244 \times 244$ to $448 \times 448$ pixels as detection requires finer details. The final layer, which predicts both class probabilities and bounding box coordinates, uses a linear activation function while the other layers use a leaky ReLU function. The original paper trained for 135 epochs on the Pascal VOC 2007 and 2012 datasets using a batch size of 64. Data augmentation and dropout used to prevent overfitting, with a dropout layer with a rate of 0.5, used between the first and second fully connected layers to encourage them to learn different things (preventing co-adaptation). There are more details available on the learning rate scheduling and other training hyperparameters in the original paper.

The loss function is the simple squared sum, but it must be modified. Without modification, the model will weight localization error, the difference between predicted and true bounding box coordinates, and class prediction error the same. Additionally, when a grid cell doesn't contain an object, its confidence score tends towards 0 which can overpower the gradients from other cells that do contain objects. Both issues are solved by using two coefficients, $\lambda coord$ and $\lambda noobj$, which multiply the loss for the coordinates and the object losses respectively. These are set to $\lambda coord = 5$ and $\lambda noobj = 0.5$, increasing the weight of detection and decreasing the importance of no object loss. Eventually, to weight small bounding box equality as much as large boxes, the width and height difference is square-rooted rather than used directly. This makes sure that the error is treated the same as in large and small boxes, which would otherwise discourage the model from predicting large boxes. For example, if the predicted width of the bounding box is 10 and the actual width is 8, and we use this equation

$$(w_i - \hat{w}_i)^2 \tag{4.1}$$

we find the loss is 4. When we scale up to a predicted width of 100 and an actual of 98, the loss is 4 again. However, a difference of 2 out of the true 98 is negligible compared to a difference of 2 out of 8. Therefore, the loss between 10 and 8 should be much larger than the loss between 100 and 98. So we use this equation instead:

$$(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 \tag{4.2}$$

Using this new equation, the loss for 10 and 8 is 0.111 while the loss for 100 and 98 is 0.010. Looking at loss as a number by itself is meaningless, but the difference between values is meaningful. So the fact that 0.111 is much smaller than 4 doesn't matter, but what does matter is that the difference between loss for the large and small widths is 0% for the squared difference while the difference is 90.99% for the squared rooted difference. That's why the square root is important: we want to treat big and small bounding boxes the same.

Each grid cell predicts multiple bounding boxes, but only one bounding box is responsible for detecting the object. The responsible bounding box is determined by choosing the predicted bounding box with the highest IOU. The effect of this is that certain bounding boxes will improve at predicting certain shapes and sizes of bounding boxes while others will specialize in other shapes. This occurs because of the following: if there is a large object when the multiple bounding boxes predict bounds, the best one is rewarded and continues to improve predicting large boxes. When a small object comes up, the previous predictor fails at predicting a good fit as its bounding box is too large. However, another predictor has a better prediction, and it is rewarded for bounding the small object well. As training goes on, the predictions of various bounding boxes diverge to specialize in the tasks they oure good at early in training time.

The loss function:

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{4.3}
$$

Here

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B}$$

The double summation merely means to sum across all of the grid cells (an ***S*** × ***S*** square) and all of the bounding boxes ***B***.

$$\mathbb{1}_{ij}^{\text{obj}}$$

This is an identity function, set to 1 if there is an object in cell ***i,*** *and* bounding box ***j*** is responsible for the prediction (it is responsible if it has the highest IOU).
As given in equation 4.1 it represents the squared difference between the actual x coordinate and the predicted coordinate in cell ***i***. This is repeated for both x and y coordinates, finding the squared difference between the overall midpoint. Finally, the identity function is `0` when there is no object or the current bounding box isn't the responsible one. In other words, we only calculate the loss for the best bounding box. So the first line of the equation 4.3 is the sum of squared differences between the predicted and actual midpoints of the objects in all grid cells which have an object in them and are the responsible bounding box.

The second line of the equation 4.3 is the sum of the squared differences between the square roots of the predicted and actual widths and heights in all grid cells which have an object in them. These are square rooted for reasons explained earlier.

The third line of the equation 4.3 is just the squared difference between the predicted class probabilities and the actual class probabilities in all cells that contain an object.

The fourth line of the equation 4.3 is the same but for all cells that don't have an object in them. These two lines are summed across all bounding boxes because each bounding box also predicts a confidence score in addition to coordinates. The reason these two are split up is so that we can multiply the fourth line by the noobj coefficient to punish the model less severely if misclassifies when there is no object present. As in line 4 we see the identity function for no object but it is not clear which bounding box is the responsible one to make the identity function a one. The research paper says the responsible box is the one with the highest IOU, but if there is no object there is no ground truth box, and consequently, no IOU values. One could train all bounding boxes or just the worst or any other combination, but the original paper doesn't specify which they did.

In the last part of the equation 4.3, first summation goes through every grid cell which has an object in it. Then, for that single grid cell, the squared difference between the predicted class vector and the actual vector is found. End part is the squared difference between the predicted and actual class for all cells that have an object in them, which is basically just checking how far off the classification was. This is calculated just across grid cells and not across each bounding box as each cell predicts only a single classification regardless of the number of bounding boxes it also predicts. All of these are summed together, and the first two lines are multiplied by a coordinate coefficient to weigh them more heavily and line four is multiplied by a smaller no object coefficient to weigh it less.

### 4.3.5 Limitations

YOLO can only predict a limited number of bounding boxes per grid cell, 2 in the original research paper. And though that number can be increased, only one class prediction can be made per cell, limiting the detections when multiple objects appear in a single grid cell. Thus, it

struggles with bounding groups of small objects, such as flocks of birds, or multiple small objects of different classes.

## 4.4 YOLO v4 Algorithm

All the YOLO models are object detection models, and with the release of YOLOv4, there is a significant increase in the inference time of the model. It can be trained on a single GPU.

Object detector architecture breakdown

- Backbone, neck, head
- Bag of freebies (BoF)
- Bag of specials (BoS)
- YOLOv4 architecture selection
- YOLOv4 BoF and BoS selection

YOLOv4 is a two-stage detector with several components to it. Each component will be broken down further in the later section of the blog.

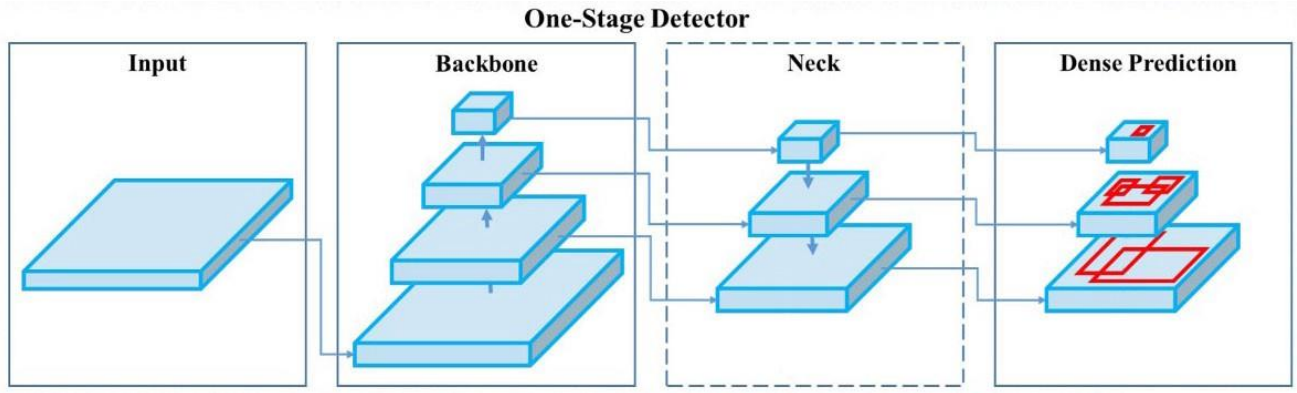



**fig 4.7:** YOLOv4 architecture

Following are the different building blocks of YOLOv4.

Input: Image, patches, Pyramid

Backbone: VGG16, ResNet-50, SpineNet, EfficientNet-B0-B7, CSPResNext50, CSPDarknet53.

Neck:

- Additional Blocks: SSP, ASPP, RFB, SAM
- Path-aggregation blocks: FPN, PAN, NAS-FPN, Fully-connected FPN, BiFPN, ASFF, SFAM

Heads:

- Dense prediction (one-stage):
    1. RPN, SSD, YOLO, RentinaNet (anchor-based)
    2. CornerNet, centerNet, MatrixNet (anchor-free)
- Sparse prediction:
    1. Faster R-CNN, R-FCN, Mask R-CNN (anchor-based)
    2. RepPoints (anchor-free)

YOLOv4 can be implemented in any combination of input, backbone, neck, and head. Backbone is the deep learning architecture that basically acts as a feature extractor. All of the backbone models are basically classification models. VGG16 is one of the earliest deep learning classifiers. There are three more models that we can use in backbone other than the models mentioned above namely SqueezeNet, MobileNet, ShuffleNet, but all of them are meant for CPU training only. Whereas we will be using Colab for training which gives us a free 13 GB GPU. Neck is a subset of the bag of specials, it basically collects feature maps from different stages of the backbone. In simple terms, it’s a feature aggregator. Head is also known as the object detector, it basically finds the region where the object might be present but doesn't tell about which object is present in that region. We have two-stage detectors and one stage-detectors which are further subdivided into anchor-based and anchor-free detectors.

**Bag of freebies (BoF)**

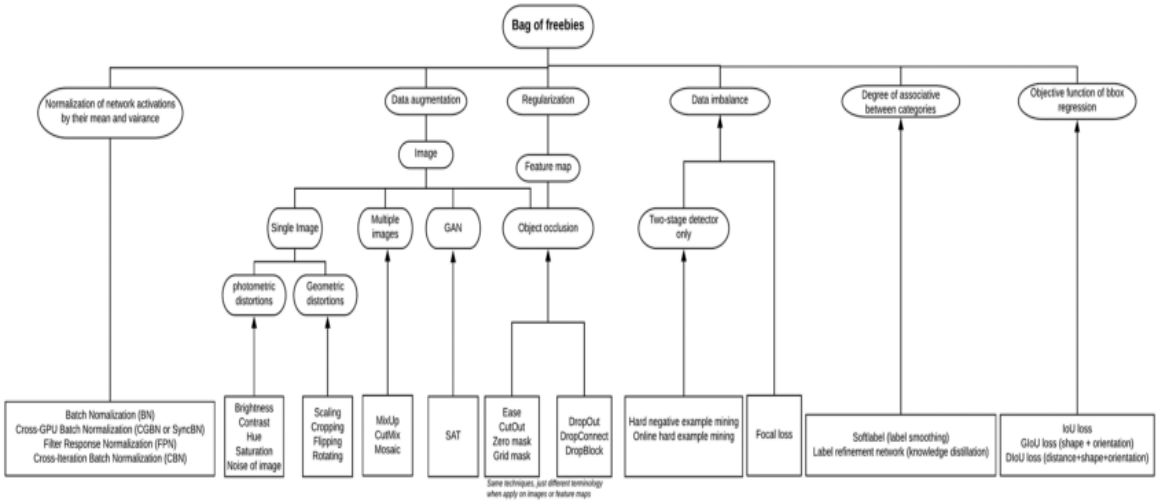



**fig 4.8:** Bag of Freebies

Bag of freebies are those methods that only change the training strategy or only increase the training cost.

**Bag of specials (BoS)**

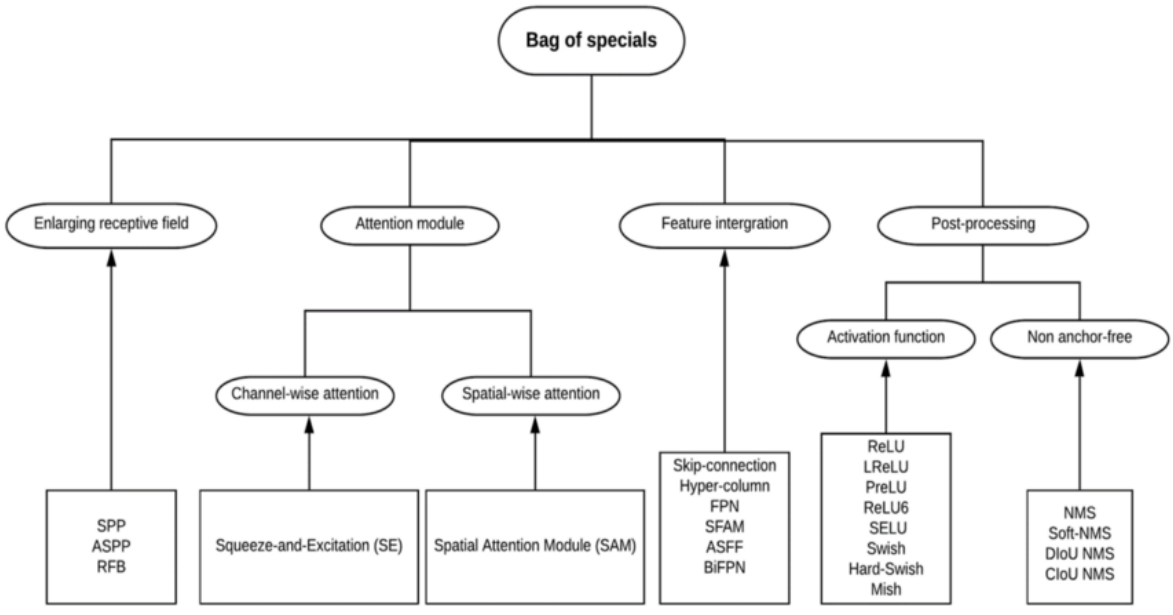



**fig 4.9:** Bag of Specials

Bag of specials are those Plugin modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy.

### 4.4.1 Architecture selection

There are a lot of possibilities in choosing the architecture. Selection criteria are based on the optimal balance between input network resolution (input image size), number of convolution layers, number of parameters, and number of output layers (filters). Also, we have an additional block for increasing the receptive field (bag of specials) and the best method from different backbone levels to different detector levels (Necks).

The final architecture presented in the paper is

CSPDarkNet53 (Head) => SSP + PANet (Neck)=> YOLOv3 (head)

Other close competitors were CSPResNext50 and EfficientNet-B3 but CSPDarkNet gave higher FPS than the rest.

Table 1: Parameters of neural networks for image classification.

| Backbone model | Input network resolution | Receptive field size | Parameters | Average size of layer output (WxHxC) | BFLOPs (512x512 network resolution) | FPS (GPU RTX 2070) |
|---|---|---|---|---|---|---|
| CSPResNext50 | 512x512 | 425x425 | 20.6 M | **1058 K** | 31 (15.5 FMA) | 62 |
| CSPDarknet53 | 512x512 | 725x725 | **27.6 M** | 950 K | 52 (26.0 FMA) | **66** |
| EfficientNet-B3 (ours) | 512x512 | **1311x1311** | 12.0 M | 668 K | 11 (5.5 FMA) | 26 |

**fig 4.10:** Different architectures

Generally, more parameters or BFLOPS mean less FPS but that is not the case here definitely. So, that's why the authors choose CSPDarkNet53. In contrast to the classifier, the detector requires a higher input network size for detecting multiple small-sized objects. It also needs more layers for a higher receptive field to cover the increased size of the input network. In short, it needs more parameters for the greater capacity of a model to detect multiple objects of different sizes in a single image.

The impact of receptive field size:-

The influence of the receptive field with different size

- Up to the object size — allows viewing the entire object

- Up to the network size — allows viewing the context around the object
- Exceeding the network size — increases the number of connections between the image point and the final activation

## 4.4.2 Techniques

### 4.4.2.1 Self-Adversarial Training (SAT)

Deep learning performs super bad with adversarial data and thus YOLOv4 uses SAT such that it introduces the *X* amount of perturbation to the training data till the predicted label remains the same as the original class. This helps the model to become more generalized.

### 4.4.2.2 CSP: Cross-Stage Partial Connection

As the number of layers grows, the last layers have a lesser and lesser context of the features learned in the initial layers. In order to solve that, researchers introduced the idea of skip connections such that gradients can backpropagate to the initial layers with relative ease. DenseNet was one such network that skipped connections from every layer to another. But the problem with DenseNet is that it has too many parameters, making it harder to train and infer. The CSPResNext50 and the CSPDarknet53 are both based on DenseNet.

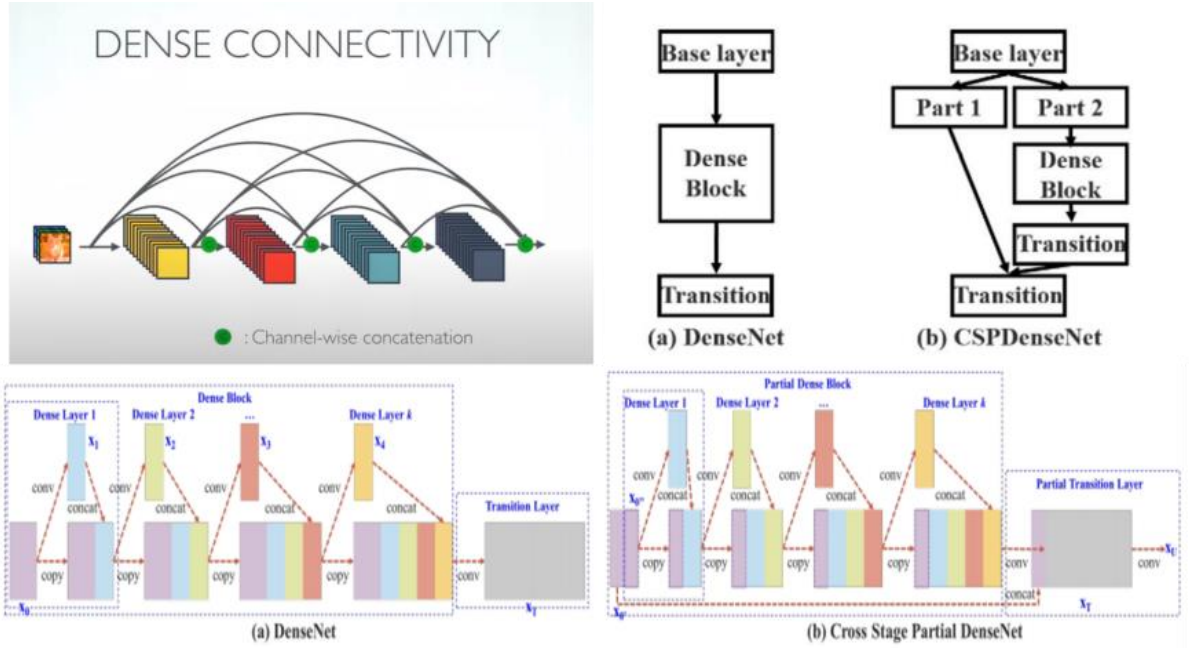



**fig 4.11:** CSP connection

#### 4.4.2.3 DenseNet and Modified DenseNet

In CSPResNext50 and CSPDarknet53, the DenseNet has been edited to separate the feature map of the base layer by copying it and sending one copy through the dense block and sending another straight on to the next stage. The idea with the CSPResNext50 and CSPDarknet53 is to remove computational bottlenecks in the DenseNet and improve learning by passing on an unedited version of the feature map.

#### 4.4.2.4 CmBN: Cross-Iteration mini Batch Normalization

Cross batch normalization is normalizing the data with 4 batches in training. So, during the training data is passed as a batch and the entire batch is normalized but it has no correlation with the previous batch or batch coming after it. CBN takes into account the mean and standard deviation of the past 4 batches and uses that to normalize the current batch. CmBN takes it even a step further, it introduces the idea of CBN in a single batch with mini-Batch normalization.

#### 4.4.2.5 SPP: Spatial Pyramid pooling

YOLOv4 uses an SPP block after CSPDarknet53 to increase the receptive field and separate out the most important features from the backbone. Spatial pyramid pooling is that we take an input image and use Conv layers to extract its feature map, then use the max pool of window size_1 to generate a feature set, then again use the max pool of window size_2, repeat this *n* times and we would have different feature maps in height and width dimension, thus we make a pyramid. YOLOv4 takes this to the next step, instead of applying SPP it divides the feature along depth dimension, applies SPP on each part, and then combines it again to generate an output feature map.

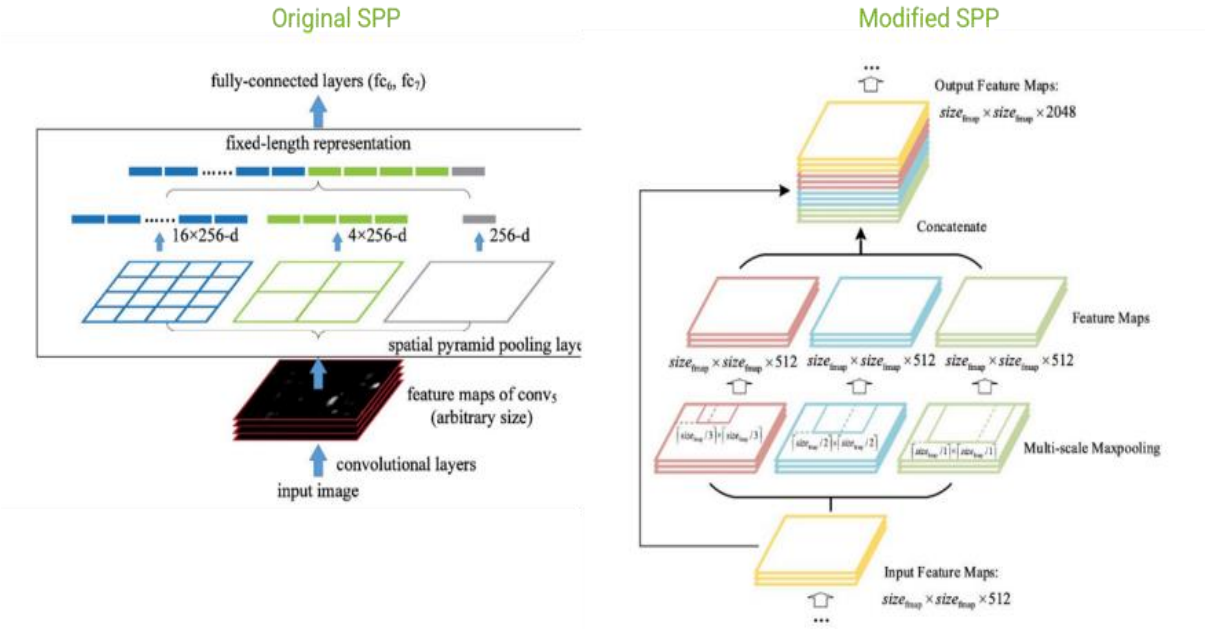



**fig 4.12:** Original and modified pooling

#### 4.4.2.6 SAM: Spatial Attention Module

Special attention Module is derived from SENet (Squeeze and Excitation Network). What SENet basically does is find which channel is more important and which is less important in a feature map block. In SAM we don't squeeze each filter to a point rather use both max pool and average pool thus we get a reduced size (width and height wise) feature block which is fed to a sigmoid activation and then crosswise multiplied to the original feature map in order to obtain the feature importance of each filter in the feature map. So, this attention mechanism is along the depth/filter, but YOLOv4 applies the attention along width and height that is called spatial attention. It basically uses an attention mechanism along with both filters (depth) and spatial (width and height)

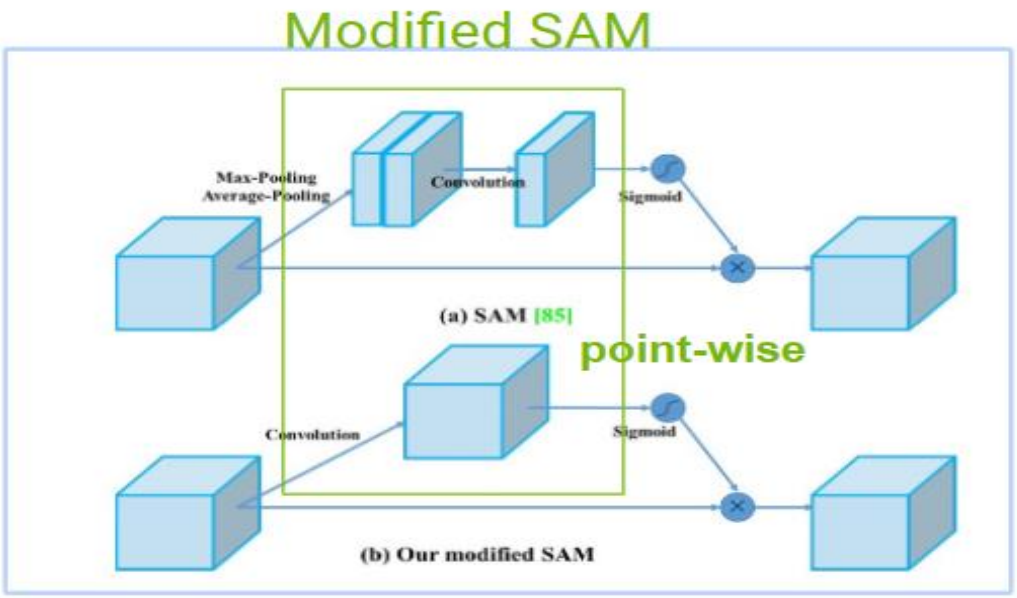



**fig 4.13:** Modified SAM

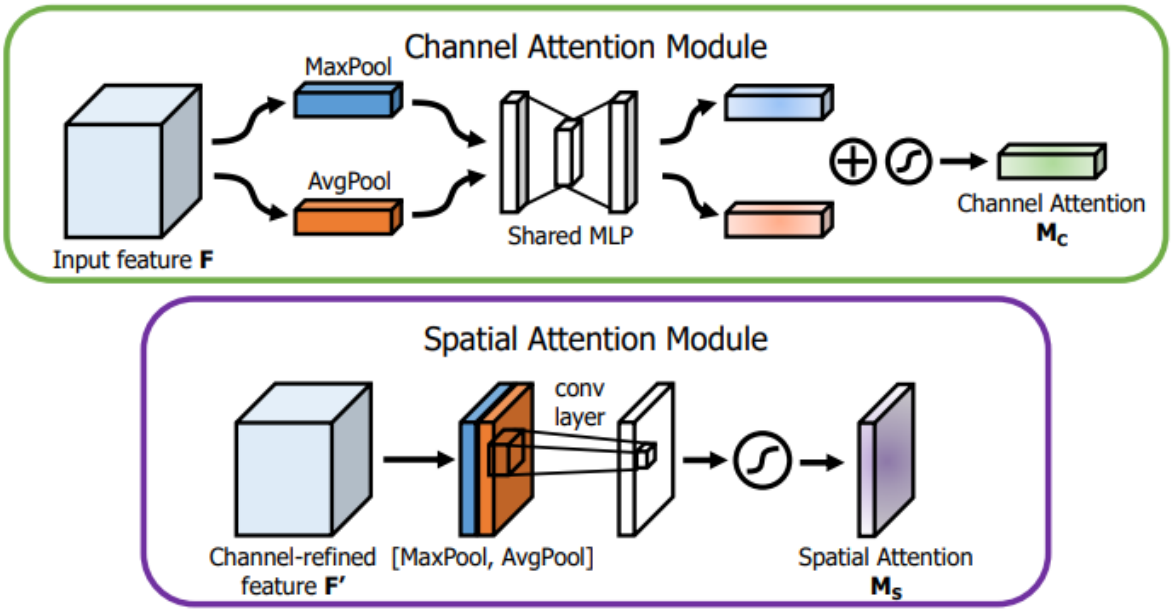



**fig 4.14:** CAM & SAM

#### 4.4.2.7 PAN Path — Aggregation Block

YOLOv4 chooses PANet for the feature aggregation of the network. They don't write much on the rationale for this decision, and since NAS-FPN and BiFPN written concurrently, this is presumably an area of future research. The thing that they modified in the YOLOV4 is that instead of using addition in the original PAN, they used concatenation. PANet is basically a Feature pyramid network that extracts important features from the backbone classifier. It uses SPP to make this FPN possible.

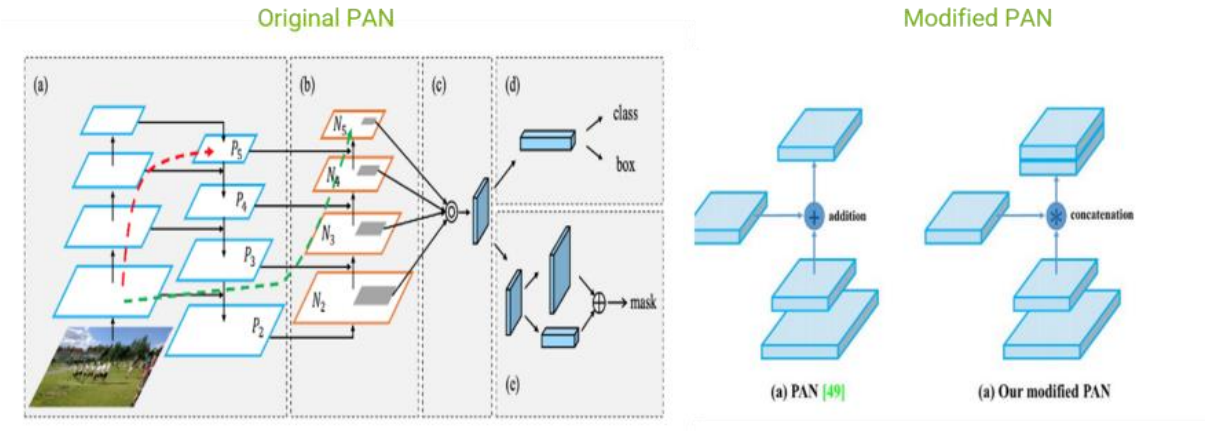



**fig 4.15:** Original PAN & modified PAN

## 4.5 Fruit detector:

In order to create our fruit detector we gone through following steps:

- Gathering dataset
- Downloaded pretrained weights for the convolutional layers
- Trained our fruit detector.
- Predictions

### 4.5.1 Gathering dataset

In order to get our object detector off the ground, we needed to first collect training images. We narrowed the domain that our model must handle as much as possible to improve our final model's accuracy. As there are different datasets present on the internet one of them was the COCO dataset – it is provided by Microsoft and called Common Objects in Context . It encompasses 158 classes but for our fruits sorting algorithms we only needed 2 classes that are apple and orange . Eventually we got a dataset from Kaggle. After intense research we also came across the Open Images Dataset. Open Images Dataset is called the Goliath among the existing computer vision datasets. It has ~9M images annotated with image-level labels, object bounding boxes, object segmentation masks, visual relationships, and localized narratives. It contains a total of 16M bounding boxes for 600 object classes on 1.9M images, making it the largest existing dataset with object location annotations.

#### 4.5.1.2 Package description:

The openimages package comes with one "download" module which provides an API with two download functions and a corresponding CLI (command-line interface) including script entry points that can be used to perform downloading of images and corresponding annotations from the OpenImages dataset.

We used OIDv4_ToolKit to download images for object detection. The ToolKit permits the download of our dataset in the folder we want . The folder can be imposed with the argument --Dataset so we can make different dataset with different options inside. Firstly, the ToolKit can be used to download classes in separate folders. The argument --classes accepts a list of classes or the path to the file.txt (--classes path/to/file.txt) that contains the list of all classes one for each lines

```
python3 main.py downloader --classes Apple Orange --type_csv validation
```

The algorithm will take care to download all the necessary files and build the directory structure like this:

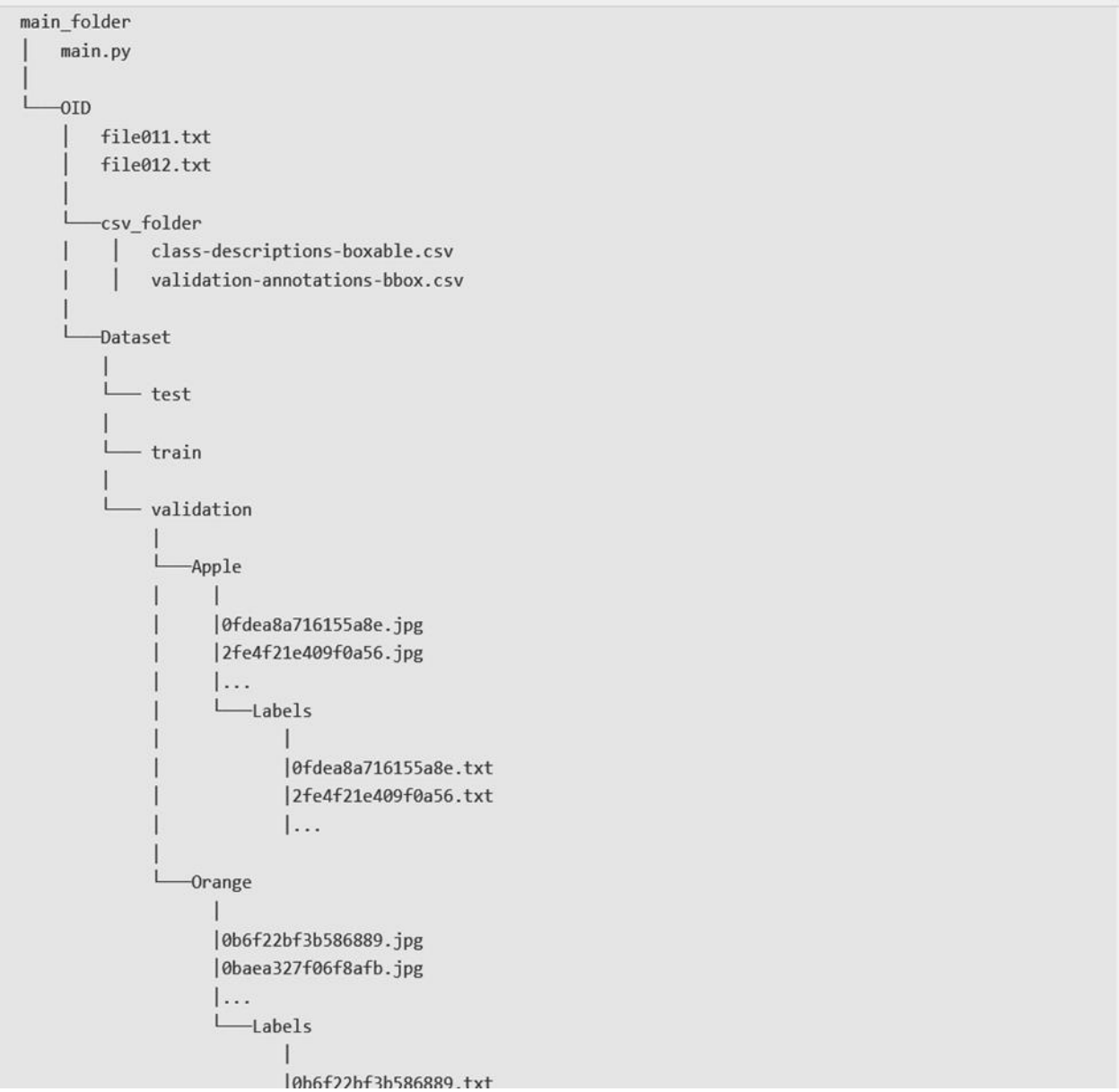



**fig 4.16:** Directory structure

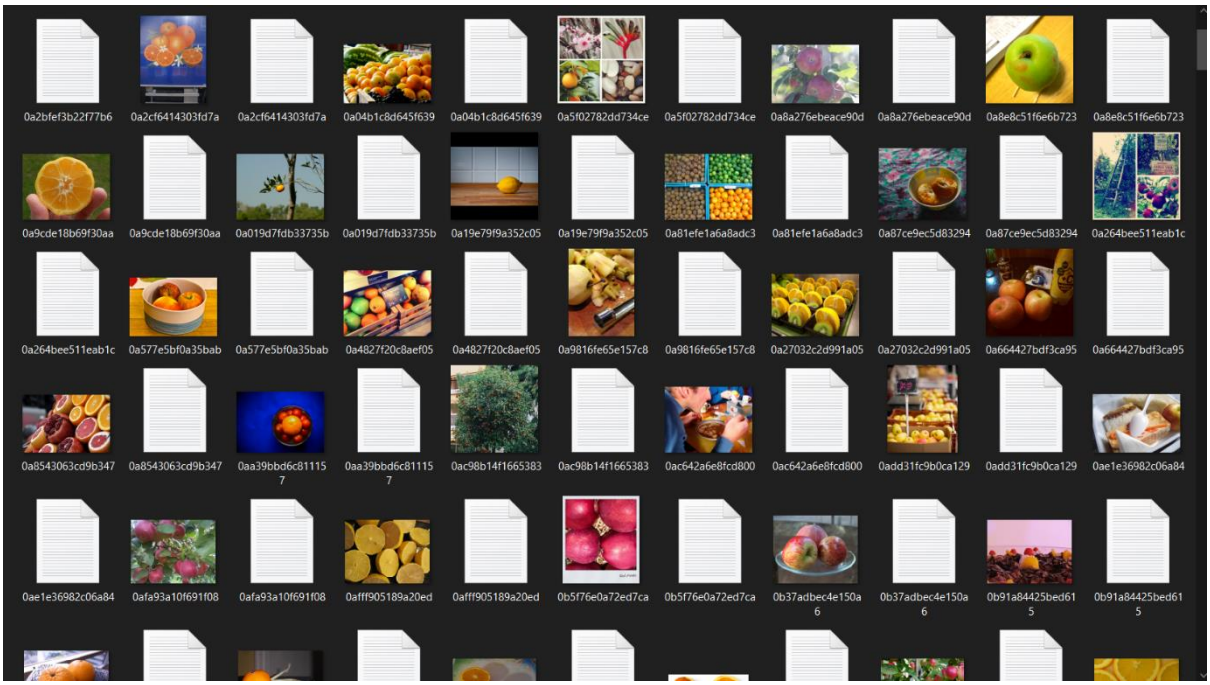


**fig 4.17**: Images and corresponding annotation

In the above figure you could see the structure of the dataset where each image is followed by its annotation file and both are given the same ID. Annotation file contains the class id and its corresponding bounding box coordinates of each object present in the image.

#### 4.5.1.3 Annotations :-

In the original dataset the coordinates of the bounding boxes are made in the following way:

XMin, XMax, YMin, YMax: coordinates of the box, in normalized image coordinates. XMin is in [0,1], where 0 is the leftmost pixel, and 1 is the rightmost pixel in the image. Y coordinates go from the top pixel (0) to the bottom pixel (1). However, in order to accommodate a more intuitive representation and give the maximum flexibility, every .txt annotation is made like:

name_of_the_class left top right bottom

`where each coordinate is denormalized. So, the four different values correspond to the actual number of pixels of the related image.

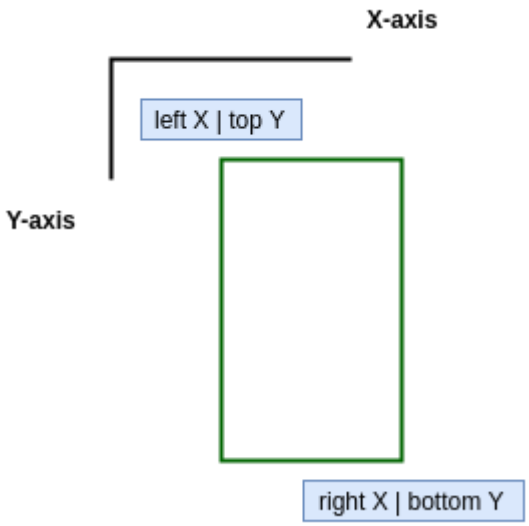



**fig 4.18:** Annotation geometry

The labels that we get from the toolkit are not in the proper YOLOv4 format. We edited classes.txt to have the classes we just downloaded, one per line. So we created a convert_annotations.py script. This converts all labels to YOLOv4 format which can now be used by darknet to properly train our custom object detector.

```
import os
import cv2
import numpy as np
from tqdm import tqdm
import argparse
import fileinput

# function that turns XMin, YMin, XMax, YMax coordinates to normalized
yolo format
def convert(filename_str, coords):
    os.chdir("..")
    image = cv2.imread(filename_str + ".jpg")
    coords[2] -= coords[0]
    coords[3] -= coords[1]
    x_diff = int(coords[2]/2)
    y_diff = int(coords[3]/2)
    coords[0] = coords[0]+x_diff
    coords[1] = coords[1]+y_diff
    coords[0] /= int(image.shape[1])
    coords[1] /= int(image.shape[0])
    coords[2] /= int(image.shape[1])
    coords[3] /= int(image.shape[0])
    os.chdir("Label")
    return coords

ROOT_DIR = os.getcwd()

# create dict to map class names to numbers for yolo
classes = {}
with open("classes.txt", "r") as myFile:
    for num, line in enumerate(myFile, 0):
        line = line.rstrip("\n")
        classes[line] = num
    myFile.close()
# step into dataset directory
os.chdir(os.path.join("OID", "Dataset"))
DIRS = os.listdir(os.getcwd())

# for all train, validation and test folders
for DIR in DIRS:
    if os.path.isdir(DIR):
        os.chdir(DIR)
        print("Currently in subdirectory:", DIR)

        CLASS_DIRS = os.listdir(os.getcwd())
        # for all class folders step into directory to change
annotations
        for CLASS_DIR in CLASS_DIRS:
            if os.path.isdir(CLASS_DIR):
                os.chdir(CLASS_DIR)
                print("Converting annotations for class: ", CLASS_DIR)

                # Step into Label folder where annotations are
generated
                os.chdir("Label")

```

#### 4.5.1.4 Moving our dataset into cloud VM

So after our datasets were properly formatted to be used for training and validation, we moved them into this cloud VM so that we can actually train and validate our model. We renamed the training dataset folder with our images and text files on our local machine to be called '**obj**' and then created a .zip folder of the 'obj' folder. Then we uploaded the zip to our Google Drive. So we should now have obj.zip someplace in our Google drive. We did the same with our validation dataset but named it 'test'. So we should now have test.zip also uploaded to our Google Drive. This greatly reduced the time it took to transfer our dataset into our cloud VM. we copied in the zips and unzips them in our cloud VM.

#### 4.5.1.5 Configuring files for training

This step involves properly configuring our custom .cfg, obj.data, obj.names, train.txt and test.txt files. It is important to configure all these files with extreme caution as typos or small errors can cause major problems with our custom training.

1. Cfg file:

Yolov4 repository provides us with a yolov4-custom.cfg file that we can edit for our custom detection. It looks like this

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500500
policy=steps
steps=400000,450000
scales=.1,.1

#cutmix=1
mosaic=1

#:104x104 54:52x52 85:26x26 104:13x13 for 416

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=mish

# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[route]
layers = -2
```

This is how we configured our Variables:

width = 416

height = 416 (these can be any multiple of 32, 416 is standard, we can sometimes improve results by making value larger like 608 but will slow down training)

max_batches = (# of classes) * 2000 (but no less than 6000 so if we are training for 1, 2, or 3 classes it will be 6000, however detector for 5 classes would have max_batches=10000). In our case there are 2 classes so max_batches = 6000

steps = (80% of max_batches), (90% of max_batches) (so if our max_batches = 10000, then steps = 8000, 9000). In our case it would be 4800, 5400

filters = (# of classes + 5) * 3 (so if we are training for one class then our filters = 18, but if we are training for 4 classes then our filters = 27) . In our case number of filters would be 21

Many times we ran into memory issues or found the training taking a super long time. In each of the three yolo layers in the cfg, we changed one line from random = 1 to random = 0 to speed up training but it will also slightly reduce accuracy of model.

2. Obj.names and obj.data

We created a new file within a code or text editor called obj.names where it has one class name per line in the same order as our classes.txt from the dataset generation step. Following is the obj.data file. Where we defined the number of classes , train path, validation path, obj.names path and backup path. It is shown in below

Classes = 2

Train = data/train.txt

Valid = data/test.txt

Names = data/obj.names

Backup = /mydrive/yolov4/backup

This backup path is where we save the weights of our model throughout training. Created a backup folder in google drive. Later both of these files were uploaded to google drive.

3. Generating train.txt and test.txt

The last configuration files needed before we train our custom detector are the train.txt and test.txt files which hold the relative paths to all our training images and validation images. We have created scripts that easily generate these two files with proper paths to all images.

Generate_train.py file :-

```
import os

image_files = []
os.chdir(os.path.join("data", "obj"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/obj/" + filename)
os.chdir("..")
with open("train.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

Generate_test.py file:-

```
import os

image_files = []
os.chdir(os.path.join("data", "test"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/test/" + filename)
os.chdir("..")
with open("test.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

We uploaded them to our Google Drive so we can use them in the Colab Notebook.

### 4.5.2 Downloaded pretrained weights for the convolutional layers.

In this step we downloaded the weights for the convolutional layers of the YOLOv4 network. By using these weights it helped our custom object detector to be way more accurate and not had to train as long. We didn't actually have to use these weights but we knew it would help our model converge and be accurate way faster. It is based on transfer learning. Following figure best describes the difference between the traditional machine learning approach and transfer learning approach that we have used in our training.

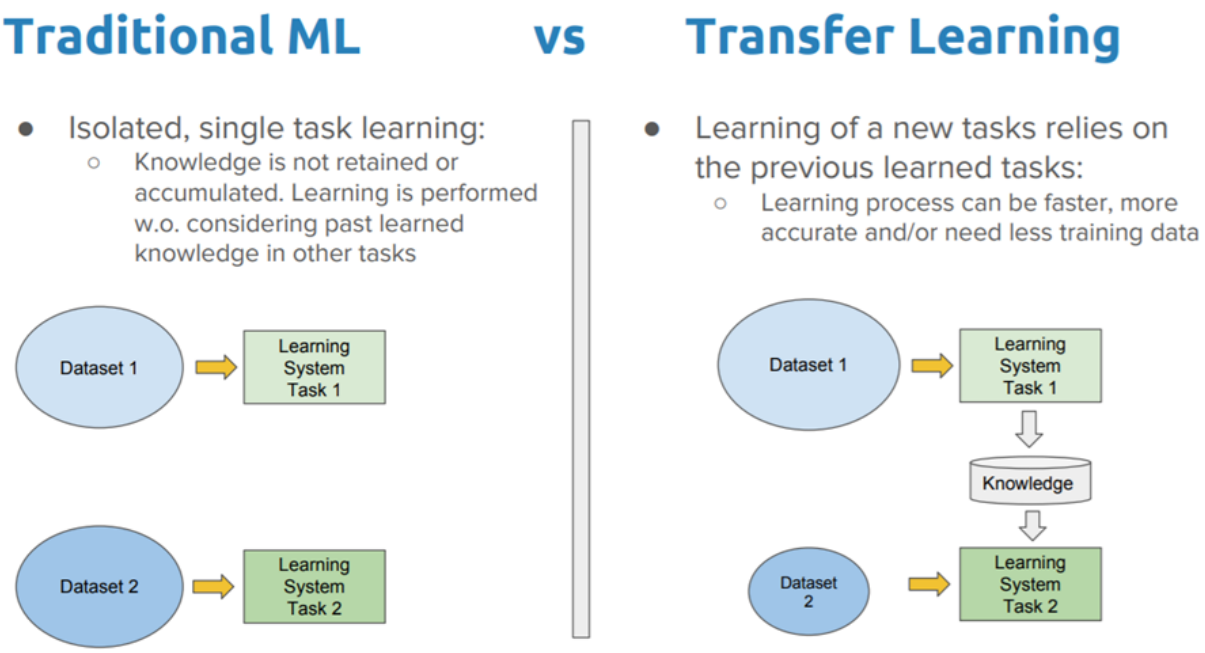



**fig 4.19**: Difference between traditional ML & Transfer learning

### 4.5.3 Trained our fruit detector

We did our training in Google Colaboratory. Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that we create can be simultaneously edited by team members. It is especially well suited to machine learning, data analysis, etc. But Colab resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate. This training could take several hours depending on how many iterations you chose in the .cfg file. However, Colab Cloud Service kicks anyone off its VMs if we are idle for too long (30-90 mins). To avoid this we went through multiple hacks and found one that basically clicks the screen every 10 minutes so that we can't be removed.

```
function ClickConnect(){
console.log("Working");
document
  .querySelector('#top-toolbar > colab-connect-button')
  .shadowRoot.querySelector('#connect')
  .click()
}
setInterval(ClickConnect,60000)
```

We executed the above code in the console window.

The model was trained for 23 hours. It took us nearly 5 days to complete the training due to the mandatory 18 hours of interval after 6 hours of training. After training we could observe a chart of how our model did throughout the training process, it is the average loss vs. iterations chart. We got our best mean average precision to be 59%. And the average loss is 0.13.

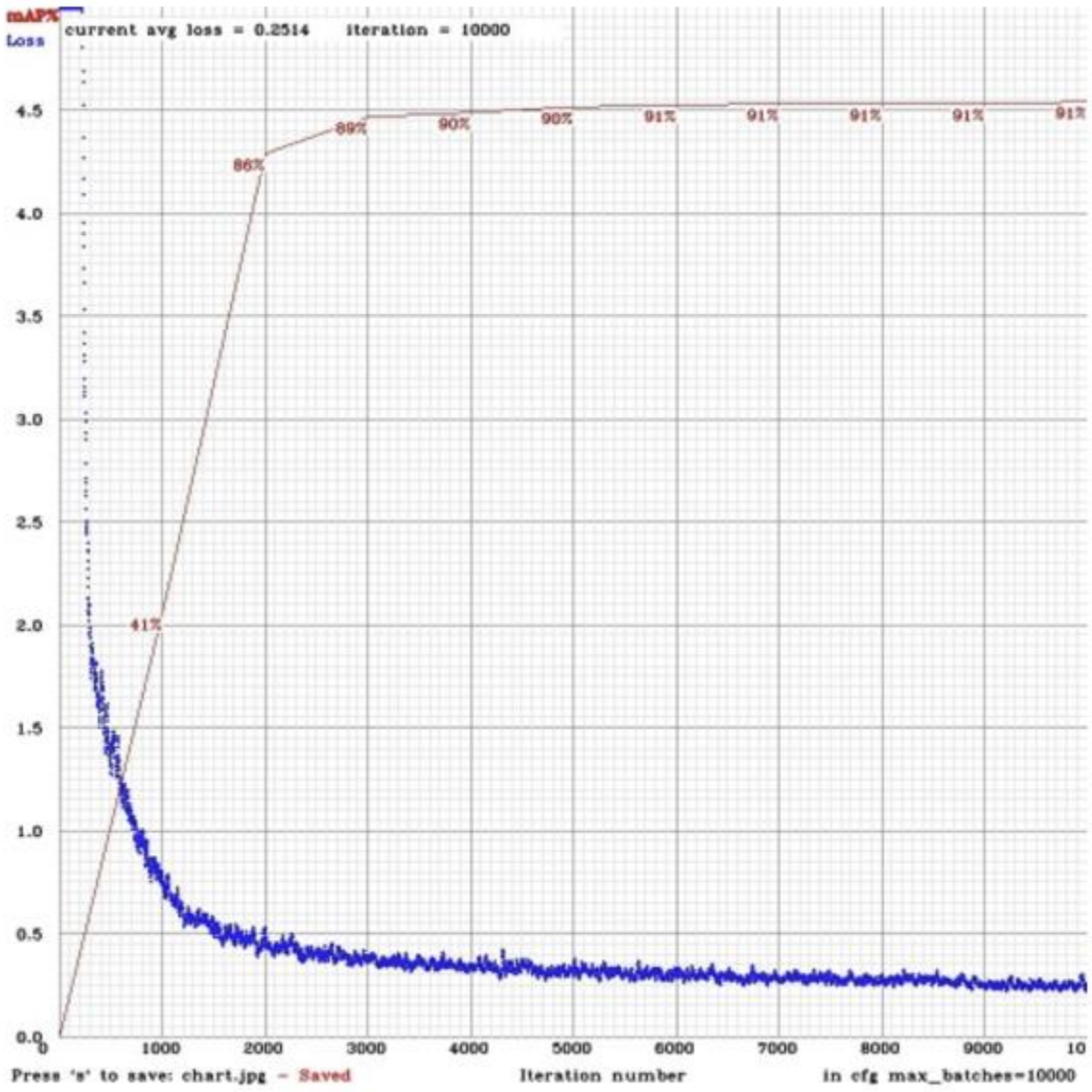



**fig 4.20:** Mean average precision vs. iterations map

## 4.6 Results

Input Prediction


**fig 4.21 :** Predictions

The experiments environment configured in this project is as follows: The operating system is Ubuntu 18.04. The CPU is Intel Xeon E5-2678 v3 with 2.5 GHZ main frequency. The GPU provided by colab is the NVIDIA GeForce GTX 1080Ti. In order to make full use of the GPU to accelerate the network training, the CUDA 10.1 and its matching CUDNN are installed in the system. The deep learning framework is PyTorch. We use the same parameters for different methods. The batch size, epoch, learning rate, momentum and decay are 16, 273, 0.001, 0.973 and 0.0005 for all methods, respectively. The mAP (mean value of average precision) , FPS (Frames per second) and GPU utilization are used to quantitatively evaluate the performance of different methods. The mAP is the mean value of average precision for the detection of all classes. FPS denotes the number of images that can be detected successfully in one second. GPU utilization denotes used GPU memory in testing the different detection methods. The mAP of our method is 51% .The FPS of our method is 120. The GPU utilized for the detection is 1223 MB. We randomly selected 3 images from the test dataset. Predictions of those images are shown in the figure 4.23. Bounding boxes are drawn around the detected object and predicted class name as well as confidence scores are also mentioned on top of the bounding boxes.

# Chapter 5

# ROS - Robotic Operating System

Software programming of robots is difficult and continues to be so as the scope and scale of robotics continue to grow, different robots have majorly varying hardware and sensors which renders the reusability of code futile. Also, the code is very huge in size starting straight from driver level of software and is almost impossible for a single researcher to support large scale integrations.

To meet these requirements, many robotics researchers have created a wide variety of frameworks to manage the complexity and facilitate rapid prototyping software for experiments, resulting in rapid growth of the field.

ROS is a Linux-based, open-source, middleware framework for modular use in robot applications. ROS, originally designed by Willow Garage and currently maintained by the Open-Source Robotics Foundation, is a powerful tool because it utilizes object-oriented programming, a method of programming organized around data rather than procedures in its interaction with data and communication within a modular system [18].

## 5.1Levels of ROS

### 5.1.1 Filesystem Level

The filesystem level is the organization of the ROS framework on a machine. At the core of the ROS's organization of software is the package. A package may contain ROS runtime execution programs, which are called nodes, a ROS-independent library, datasets, configuration files, third-party software, or any software that should be organized together [19].

The package provides easy to use functionality in an organized manner so that the same software may be used for multiple different projects or prototypes. All this along with object-oriented programming allows the packages to act as modular blocks that work together to accomplish a desired end state; hence code reusability comes into play. Packages follow a common structure

and contain the following elements: package manifests, message types, service types, headers, executable scripts, build file and runtime process [19].

Package manifests provide metadata about the package such as name, author, version, description, license information and any dependencies. Message types define the structure of data for messages sent within ROS (i.e., describe the data values that ROS nodes publish). Service types which define the request and response data structures for services. There are also repositories in the filesystem level, With the help of both the packages and repositories we can see that they make ROS a modular system.

### 5.1.2 Computational Graph Level

The computational graph level is where ROS processes data within a peer-to-peer network. The basic elements of ROS's computational graph level are nodes, topic, messages, bags, services, master and parameter server.

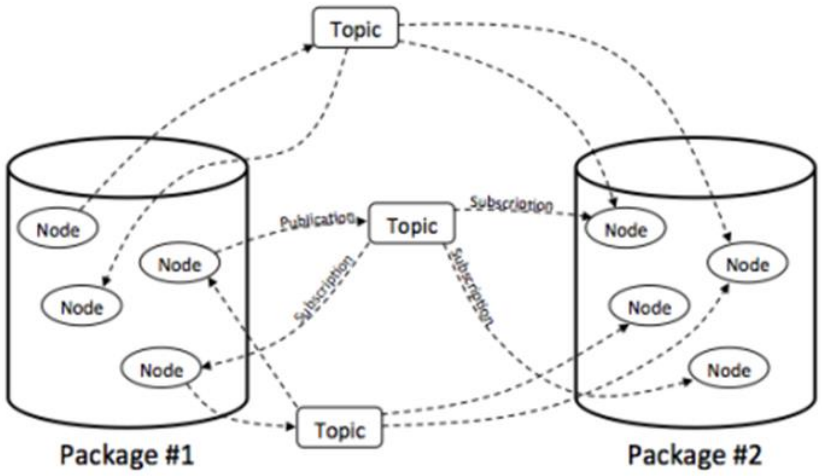



**figure 5.1:** Package Communication

#### 5.1.2.1 NODES

A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic

[20]. Nodes subscribe to a topic to receive information, perform computation, control sensors and actuators and publish data to topics for other nodes to use.

The rosnode tool is a useful command line tool for displaying information about ROS nodes. The command **rosnode list** displays all the active nodes currently running on ROS master.

A package usually contains many nodes within it to accomplish a set of actions and computations in which they all communicate with each other through topics and services via messages.

The primary method of transfer of data between nodes is by publishing messages to topics. A message is structuring of data so it is in a useful and standard format for the other nodes to use. Standard types such as integer,floating point, Boolean as well as arrays are supported. The rosmsg list prints all the messages available to the ROS master.

The main reason what makes ROS modular is the method in which the nodes communicate with each other. Rather than communicating with each other, ROS nodes communicate via topics, Topics can be considered as hubs in which nodes can publish and/or subscribe. Nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a single topic.

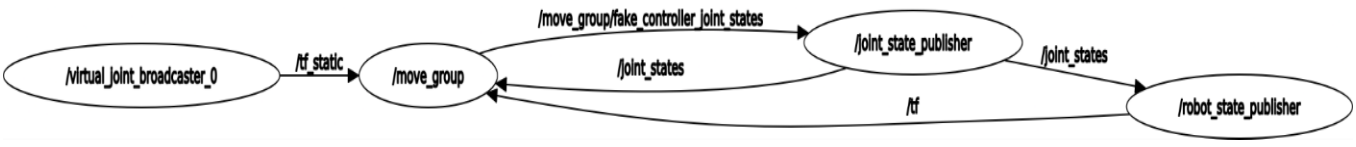



**figure 5.2:** Various nodes communicating

**The move_group node**

This node serves as an integrator: pulling all the individual components together to provide a set of ROS actions and services for users to use.

move_group talks to the robot through ROS topics and actions. It communicates with the robot to get current state information (positions of the joints, etc.), to get point clouds and other sensor data from the robot sensors and to talk to the controllers on the robot.

**joint_state_publisher node**

Publishes JointState message with values read from various sources, including GUI.

robot_state_publisher node

robot_state_publisher uses the URDF specified by the parameter robot_description and the joint positions from the topic joint_states to calculate the forward kinematics of the robot and publish the results via tf.

**5.1.2.2TOPICS**

Topics allow nodes of different packages to work with each other even though they may have different origins or functions [21]. rostopic is a useful tool for displaying debugging information about a topic. Following are some useful rostopic commands -

•rostopic list – Displays all active topics.
•rostopic info <topic_name> – Prints the message type accepted by the topic and the subscribe and publisher node.
•rostopic echo <topic_name> – Prints the messages published to a topic.
•rostopic pub <topic_name> – Manually publish data to a topic.
•rostopic hz <topic_name> – Displays publishing rate used by a topic.
•rostopic bw <topic_name> – Displays bandwidth used by a topic.

```
mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers:
 * /joint_state_publisher (http://mayuresh:35101/)

Subscribers:
 * /robot_state_publisher (http://mayuresh:41311/)
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /attached_collision_object
Type: moveit_msgs/AttachedCollisionObject

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /collision_object
Type: moveit_msgs/CollisionObject

Publishers: None

Subscribers:
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /execute_trajectory/goal
Type: moveit_msgs/ExecuteTrajectoryActionGoal

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /execute_trajectory/cancel
Type: actionlib_msgs/GoalID

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)
```

**fig 5.3** – Terminal output of topics published by Moveit part1

**joint_states**

This topic describes the state of set of torque-controlled joints through a received message, sensor_msgs/Jointstate.msg, the state of each joint is defined by the position of the joint (rad or m), the velocity of the joint (rad/s or m/s) and the torque or the effort that is applied to the joint (Nm).

The information received by the topic can be published further for use in manipulation of the robot.

**attached_collision_object**

Receives message about collision object that is attached to a link, it receives information about actual shapes and poses for the collision object, the set of links that are allowed to touch, if certain links were in a particular position to keep the object attached, the posture necessary to release the object and the weight of the object.

**Collision_object**

Receives message about the object type in the known object database, the collision geometry associated with the object, solid geometric primitives, meshes, bounding planes, adding the object to the planning scene, removing the object from the planning scene. move_group node subscribes to the topic for important information.

**execute_trajectory/goal**

Receives message about trajectory execution published from moveit and can be further used in code for manipulation of the robot by subscribing to this topic

**execute_trajectory/cancel**

Receives message which stores the time at which goal was requested and is used by the action server when it tries to preempt all goals requested before a certain time. Id provides a way to associate feedback and result messages with specific goal requests. The id specified must be unique.

```
mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /execute_trajectory/feedback
Type: moveit_msgs/ExecuteTrajectoryActionFeedback

Publishers:
 * /move_group (http://mayuresh:39315/)

Subscribers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /execute_trajectory/result
Type: moveit_msgs/ExecuteTrajectoryActionResult

Publishers:
 * /move_group (http://mayuresh:39315/)

Subscribers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /pickup/goal
Type: moveit_msgs/PickupActionGoal

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /pickup/cancel
Type: actionlib_msgs/GoalID

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /pickup/feedback
Type: moveit_msgs/PickupActionFeedback

Publishers:
 * /move_group (http://mayuresh:39315/)

Subscribers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)
```

**fig 5.4**: Terminal output of topics published by Moveit part2

**execute_trajectory/feedback**

Receives feedback describing the progress the mechanism is making upon following the trajectory. Published by move_group node.

**execute_trajectory/result**

Receives message about the result of executed trajectory, published by move_group node.

**pickup/goal**

Receives information about the action for picking up an object, includes various information about name of the object to pick up, which group should be used to plan for pickup, which end effector to be used for pickup, list of possible grasps to use, name of the support surface, whether collision between end effector and support surface should be acceptable, names of links the object to be attached is allowed to touch, name of the motion planner to use and the maximum amount of time the motion planner is allowed to plan for.

**pickup/cancel**

Receives message which stores the time at which goal was requested and is used by the action server when it tries to preempt all goals requested before a certain time.

**pickup/feedback**

receives message about the information of the internal state that the pickup action currently is in.

```
mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /pickup/result
Type: moveit_msgs/PickupActionResult

Publishers:
 * /move_group (http://mayuresh:39315/)

Subscribers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /planning_scene
Type: moveit_msgs/PlanningScene

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /rosout
Type: rosgraph_msgs/Log

Publishers:
 * /virtual_joint_broadcaster_0 (http://mayuresh:44725/)
 * /robot_state_publisher (http://mayuresh:41311/)
 * /move_group (http://mayuresh:39315/)
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)
 * /joint_state_publisher (http://mayuresh:35101/)
 * /moveit_python_wrappers_16516469968287521 27 (http://mayuresh:34569/)

Subscribers:
 * /rosout (http://mayuresh:37169/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /rosout_agg
Type: rosgraph_msgs/Log

Publishers:
 * /rosout (http://mayuresh:37169/)

Subscribers: None
```

**fig 5.5:** Terminal output of topics published by Moveit part3

**pickup/result**

Receives message about the overall result of the pickup attempt, the full starting state of the robot at the start of the trajectory, the trajectory that moved group produced for execution, the performed grasp and whether if attempt was successful.

**planning_scene**

receives message about the name of planning scene, the full robot state, the name of the robot this scene is for, additional frames for duplicating tf, full allowed collision matrix, all links padding, all links scales, attached objects, collision objects and collision map.

**rosout**

Each log message can appear as output on the console, as a message on the rosout topic, and as an entry in a log file.

**rosout_agg**

rosout_agg is an aggregated feed for subscribing to console logging messages. This aggregated topic is offered as a performance improvement: instead of connecting to individual ROS nodes to receive their console messages, the aggregated message feed can instead be received directly from the rosout node.

```
mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /rosout_agg
Type: rosgraph_msgs/Log

Publishers:
 * /rosout (http://mayuresh:37169/)

Subscribers: None


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /tf
Type: tf2_msgs/TFMessage

Publishers:
 * /robot_state_publisher (http://mayuresh:41311/)

Subscribers:
 * /move_group (http://mayuresh:39315/)
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /tf_static
Type: tf2_msgs/TFMessage

Publishers:
 * /virtual_joint_broadcaster_0 (http://mayuresh:44725/)
 * /robot_state_publisher (http://mayuresh:41311/)

Subscribers:
 * /move_group (http://mayuresh:39315/)
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)


mayuresh@mayuresh:~/ros_ws/coep_ws$ rostopic info /trajectory_execution_event
Type: std_msgs/String

Publishers:
 * /rviz_mayuresh_5132_1603547477841333595 (http://mayuresh:36985/)

Subscribers:
 * /move_group (http://mayuresh:39315/)
```

**fig 5.6**: Terminal output of topics published by Moveit part4

#### 5.1.2.3 tf

tf is a package that receives and keeps track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time[22].

**tf_static**

tf_static receives broadcasted static frames and uses lesser bandwidth, These frames are latched, which means that the last message sent on /tf_static will be sent to new subscribers.

#### 5.1.2.4 SERVICES

In addition to publishing messages to topics, nodes can also exchange a request and response message as part of a ROS service. This is useful if the publish and subscribe (many-to-many) communication method is not appropriate, such as a remote procedure call. A ROS node that provides data offers a service under a string name, and a client node that requires data calls the service by sending the request message and awaiting the response [23]. Active services can be displayed by utilizing the command rosservice list, and information about a service can be found by using rosservice info <service_name>.

#### 5.1.2.5 BAGS

Another powerful tool is the bags, bags are a method of storing and recording ROS message data. It allows users to store, process, analyze and visualize the flow of data.

Bags are created using the ‘rosbag’ tool, which subscribes to one or more topics and stores message data as they are received. This stored can be replayed in ROS to the same topics as if the original nodes were sending the messages. This is especially useful for testing different algorithms, sensors, controllers and actuators using a controlled data stream. Some useful commands –

rosbag record <topic_name> – records data.
rosbag info <bag_file> – displays information about a bag file.
rosbag  play <bag_file> – Publish messages from topics as if were being played the first time

#### 5.1.2.6 LAUNCH FILE

Launch file is a method of launching multiple ROS nodes at the same time, as well as establishing parameters on the ROS Parameter server. It is useful in large scale projects where multiple packages, nodes, libraries, parameters as well as other launch files which can all be started by a single launch file rather than individually running each node separately.

roslaunch tool uses XML (extensible markup language) file that describes the nodes that should be run, parameters that needs to be set and any other attributes of launching a collection of ROS nodes [6]. The roslaunch tool is utilized by using the command **roslaunch <package_name> <file.Launch>**.

### 5.1.3 Community Level

The ROS community level consists of ROS distributions, repositories, the ROS wiki and ROS answers, which all enable researchers, hobbyists and industries to exchange ideas, software and knowledge in order to progress robotics worldwide.

ROS distributions are a collection of versioned ROS stacks, allows to utilize different versions of ROS software frameworks which means that even if ROS continues to be updates, users can maintain their projects with older and more stable versions.

## 5.2 Other ROS concepts

### 5.2.1 Unified Robot Description Format

The unified robot description format (URDF) package contains an XML file that has all the data about the robot model. URDF is yet another tool which helps in making ROS modular, rather than creating nodes for all the different robot, nodes are created without a regard for the robot that will use them, the URDF file comes into play here giving necessary, robot-specific information so that the nodes may conduct their procedures.

A URDF file is written such that each link of the robot is the child of a parent link, with joints connecting each link and joints are defined with their offset from the reference frame of the parent link and the axis of rotation. A tree diagram of the URDF file can be visualized using the **urdf_to_graphiz** tool as shown below.

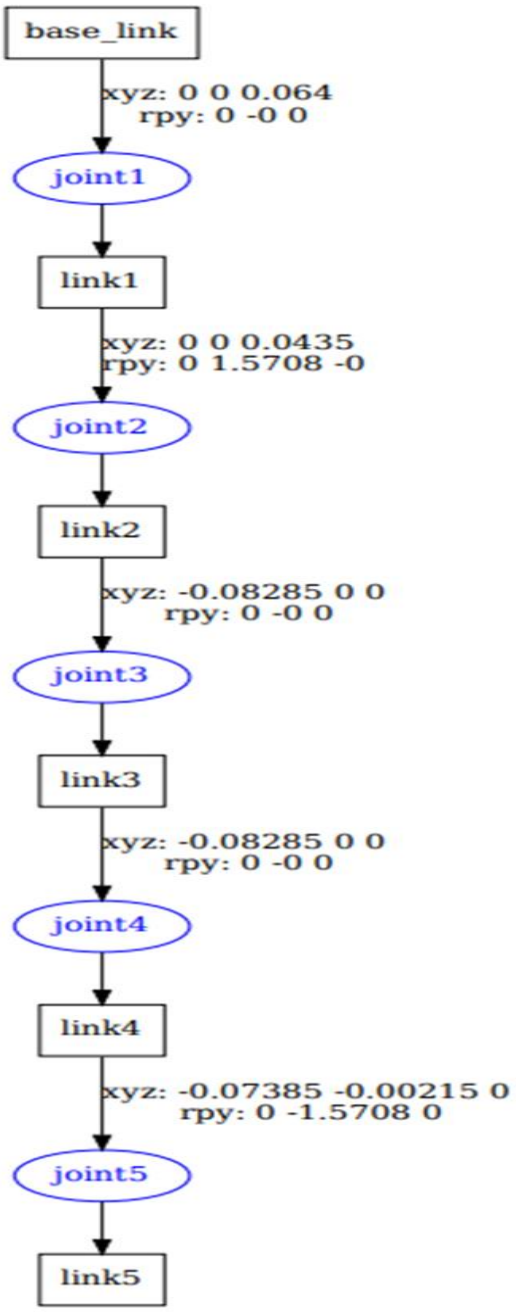



**fig 5.7**: URDF tree

### 5.2.2 ROS Installation

ROS is currently supported on Ubuntu and experimentally supported on OS X, Arch Linux and Debian Wheezy. For this project ROS on Ubuntu was used, before installing ROS, Ubuntu must be properly configured to accept the four types of repository components: main, officially supported software; restricted, supported software not available under a completely free license; universe, community-maintained software; and multiverse, software that is not free. This is done from the Software Sources interface, which can be accessed through the Ubuntu Software Center or from the terminal.

Following are the commands used to install ROS Noetic

```
# Set up the system to accept software packages from packages.ros.org
  $ sudo sh -c 'echo "deb <a href="http://packages.ros.org/ros/ubuntu" rel="nofollow"> http://packages.ros.org/ros/ubuntu
    </a> $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

# Set up the keys
  $ sudo apt install curl
  $ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add

# Update the debian package index
  $ sudo apt-get update

# Install ROS
  $ sudo apt-get install ros-noetic-desktop-full

# Add ROS variables to bash automatically every time new shell is launched
  $ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
  $ source ~/.bashrc

# install tools and other dependencies for building ROS packages
  $ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential

# Initialize rosdep
  $ sudo apt install python-rosdep
  $ sudo rosdep init
  $ rosdep update
```

**fig 5.8:** ROS installation process

### 5.2.3 MoveIt

MoveIt has been used on over 126 robots by the community from deep sea to outer space, from hobbyists to industrial applications. One of the basic things that MoveIt does is to create necessary trajectories for the arm to put the end effector in a user defined place.

In simple words we need a Motion Plan, basically creating a sequence of movements that all joints have to perform in order for the end effector to reach the desired position, Motion planning is a hectic task and MoveIt provides the plan that the joints have to follow. Moveit requires a URDF file of the robot

### 5.2.3.1 MoveIt Installation and setup

**1.We first install moveit package using command**

$ sudo apt install ros-noetic-moveit

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

$ rosrun moveit_setup_assistant_moveit_setup_assistant

**2.Load URDF model**

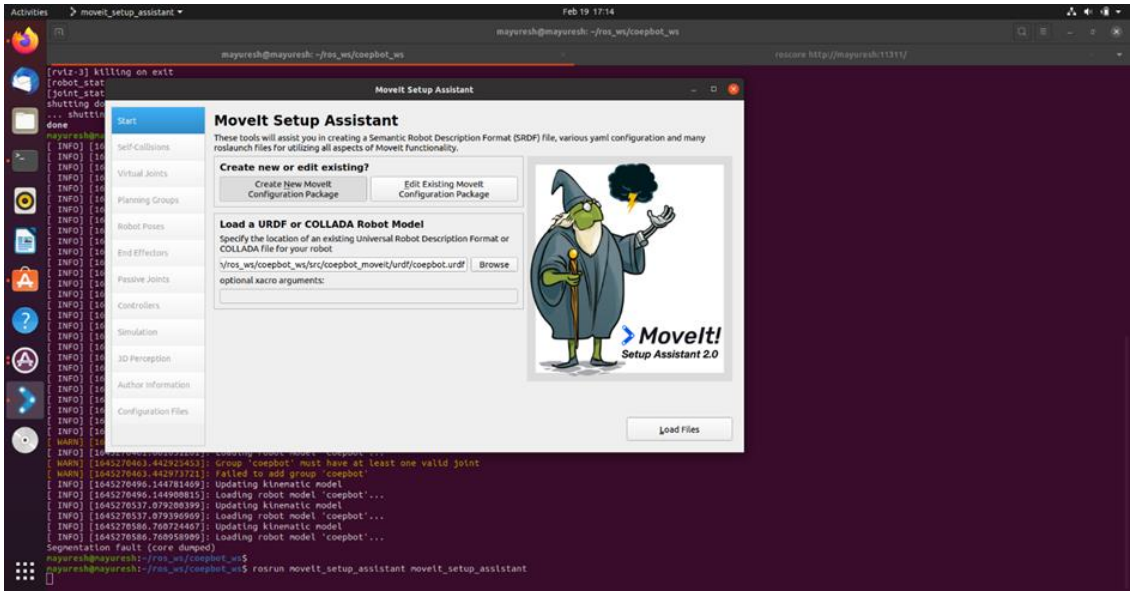



**fig 5.9:** Loading URDF model

Click the Browse button, find the URDF model file and load it.

**3.Create collision avoidance matrix**

click the “Generate Collision Matrix” button. The Default Self-Collision Matrix Generator searches for pairs of links on the robot that can safely be disabled from collision checking, decreasing motion planning processing time. These pairs of links are disabled when they are always in collision, never in collision, in collision in the robot’s default position or when the

links are adjacent to each other on the kinematic chain. The sampling density specifies how many random robot positions to check for self-collision. Higher densities require more computation time while lower densities have a higher possibility of disabling pairs that should not be disabled. The default value is 10,000 collision checks. Collision checking is done in parallel to decrease processing time.

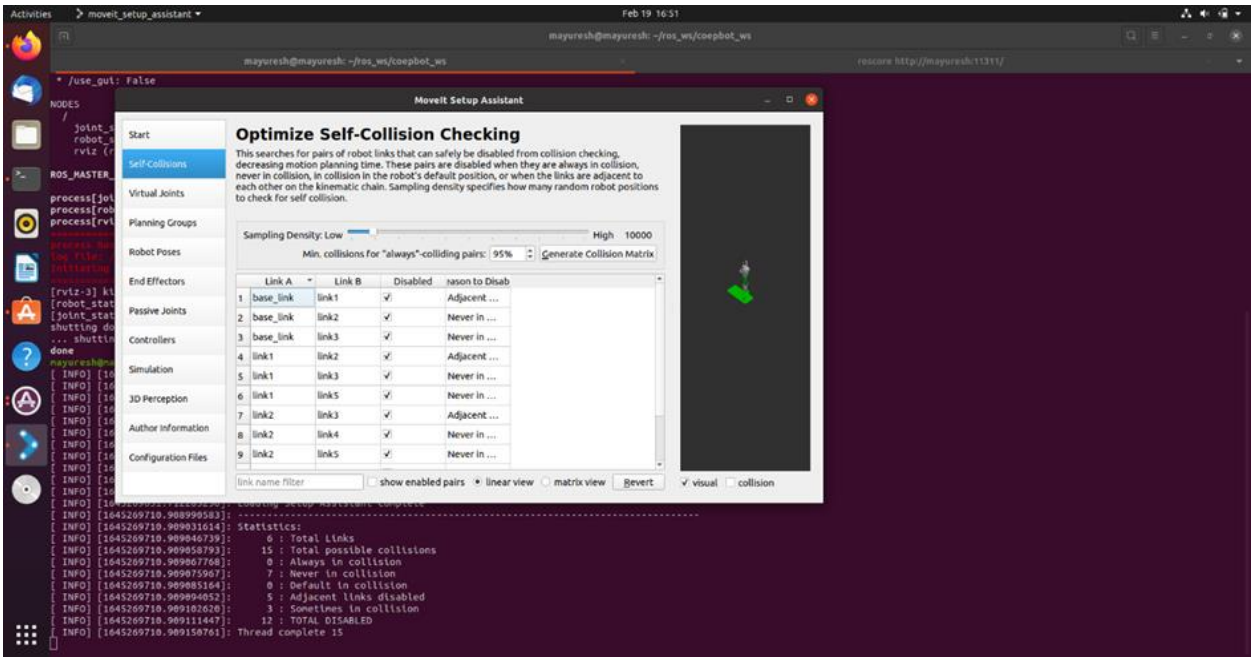



**fig 5.10:** Self Collision Checking

**4.Add virtual joints.**

Virtual joints are used primarily to attach the robot to the world, we named Virtual Joint virtual_joint. Child Link refers to the part where we want to connect the ‘world’ to the robot, and we choose base_link.

Parent Frame Name, is the name of the world coordinate, generally called world in ROS.

For joint Type we select Fixed. It means that the robot is fixed relative to the world.

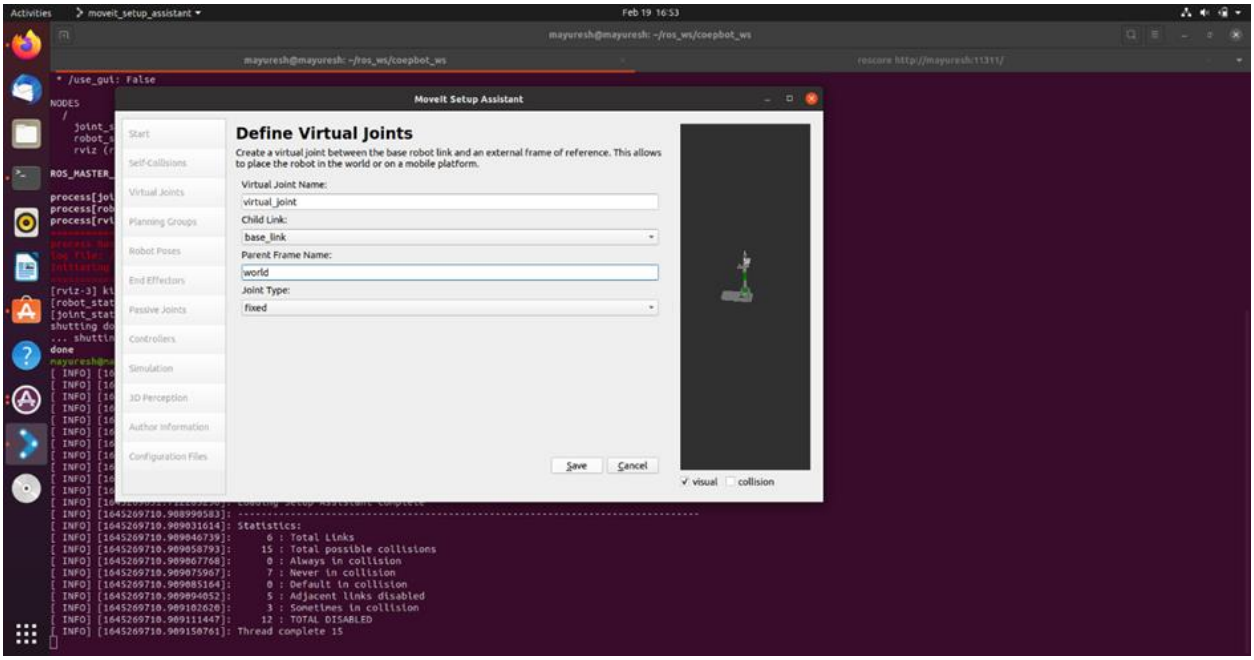



**fig 5.11:** Define Virtual Joints

### 5. Define planning group

Planning groups are used for semantically describing different parts of your robot, such as defining what an arm is, or an end effector. They help the kinematics planner only focus on a part of the robot to plan and execute robot motion.

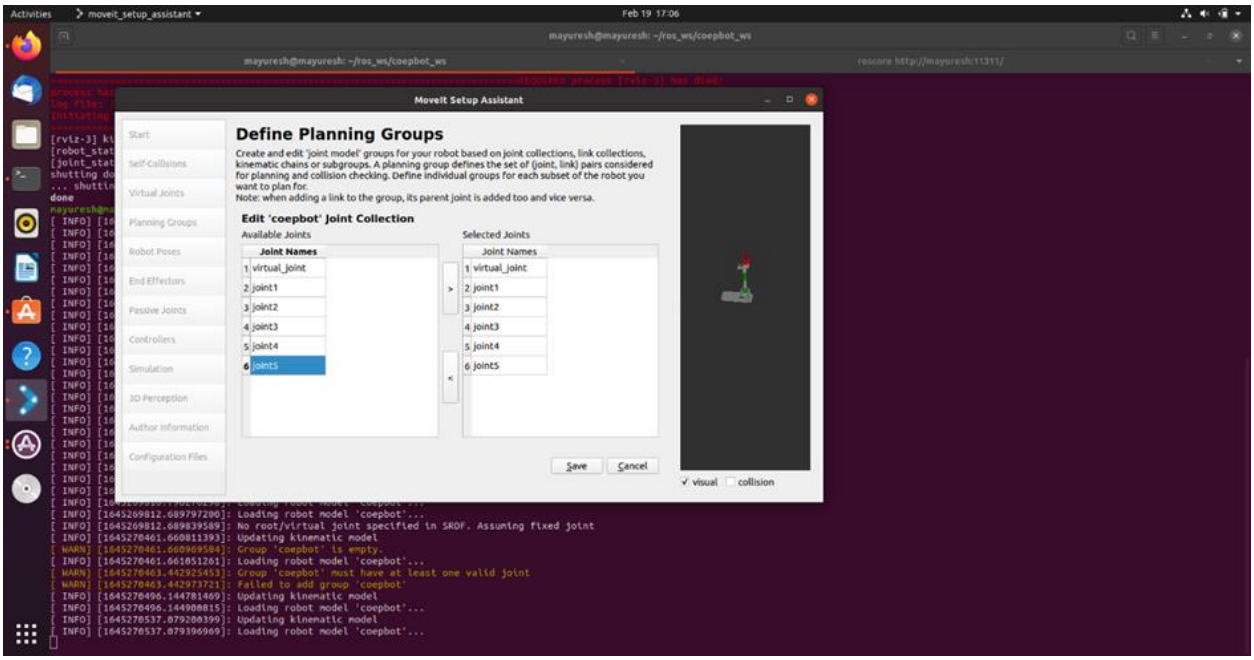



**Fig 5.12:** Define Planning group

### 6. Add robot poses

For creating preset robot poses. The Setup Assistant allows you to add certain fixed poses into the configuration. This helps in defining a certain position of the robot as a Home position.

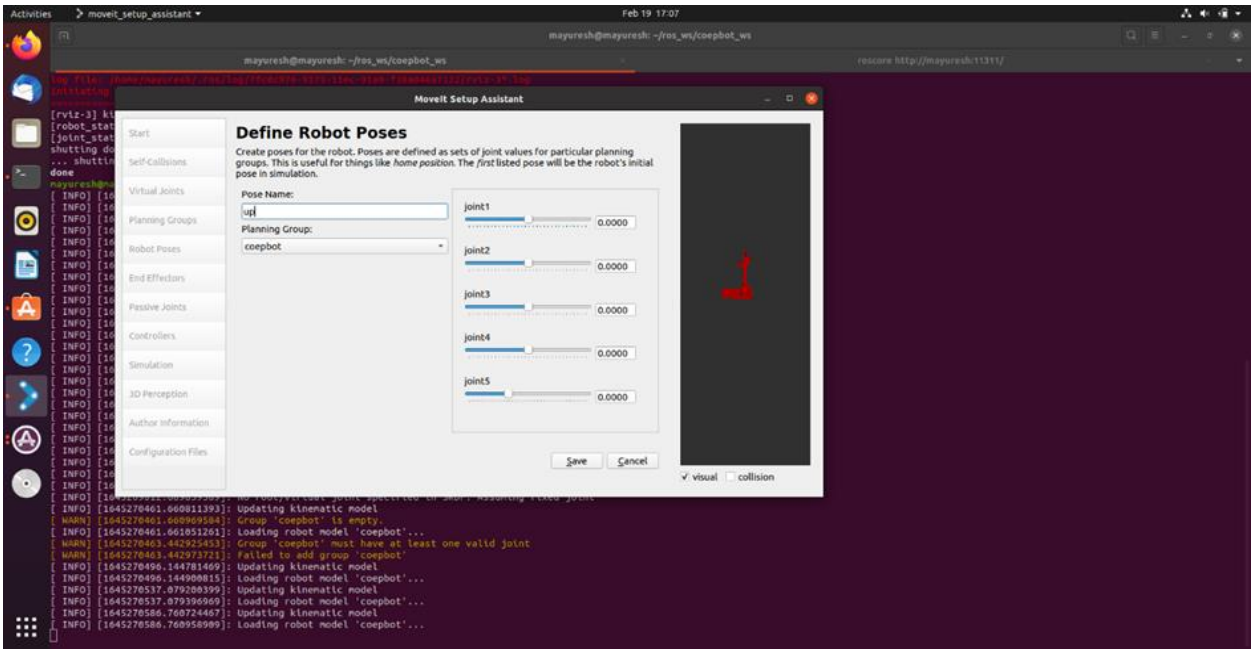



**Fig 5.13:** Define Robot Pose(up)

If the parameter is set to 0, it will become a vertical upward pose and if we slide joint2 for far left it will become down pose.

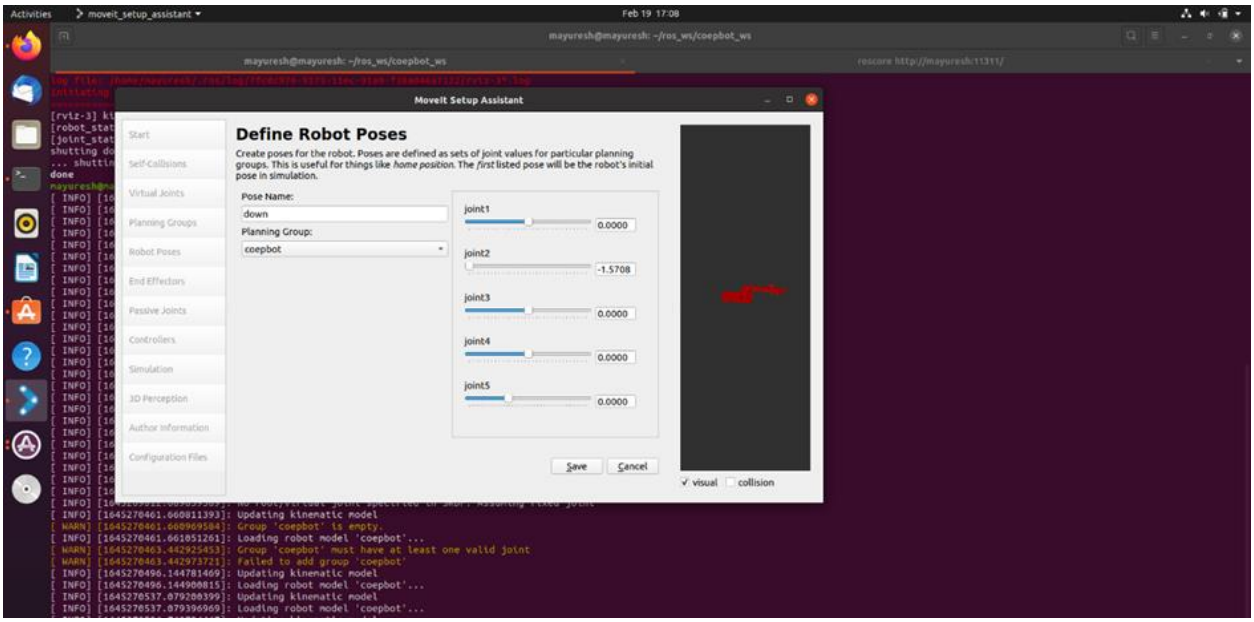



**Fig 5.14:** Define Robot Pose(down)

### 7.Establish ROS controller

ROS Control is a set of packages and tools that allow you, basically, to send commands and communicate with the joints of your robot in order to be able to control them, ROS Control tab can be used to auto generate simulated controllers to actuate the joints of your robot.

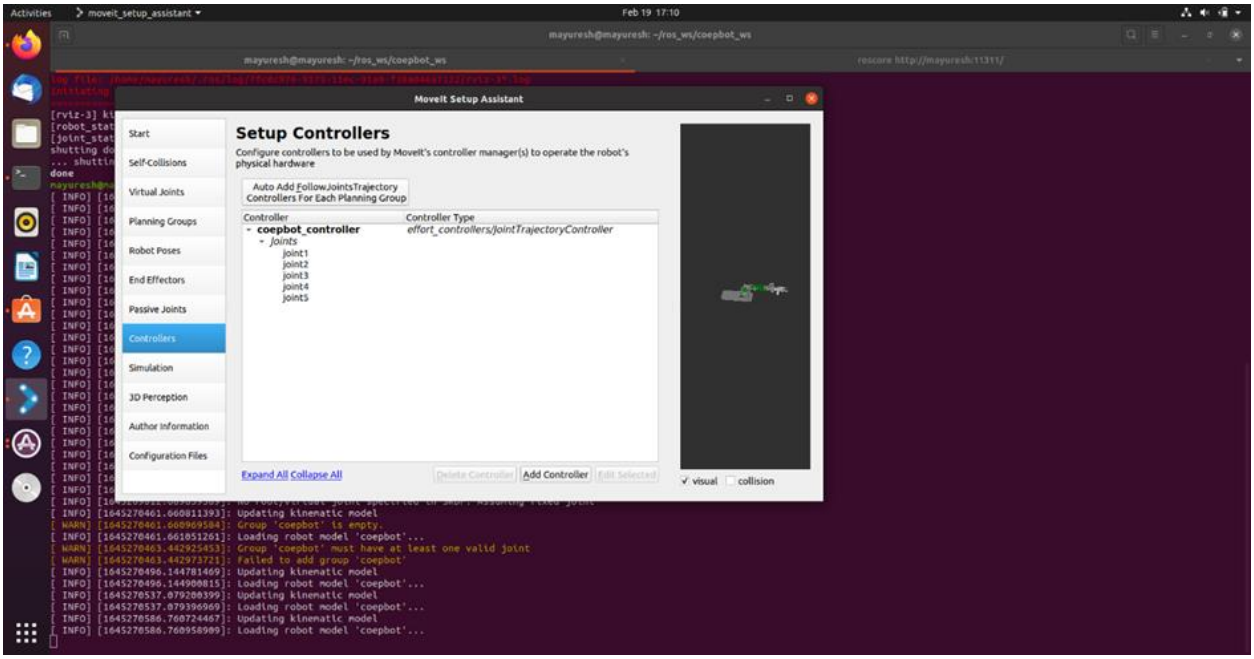



**Fig 5.15:** Setup Controllers

### 8. Add available gazebo simulation

The Simulation tab can be used to help simulate robot with Gazebo by generating a new Gazebo compatible URDF when needed.

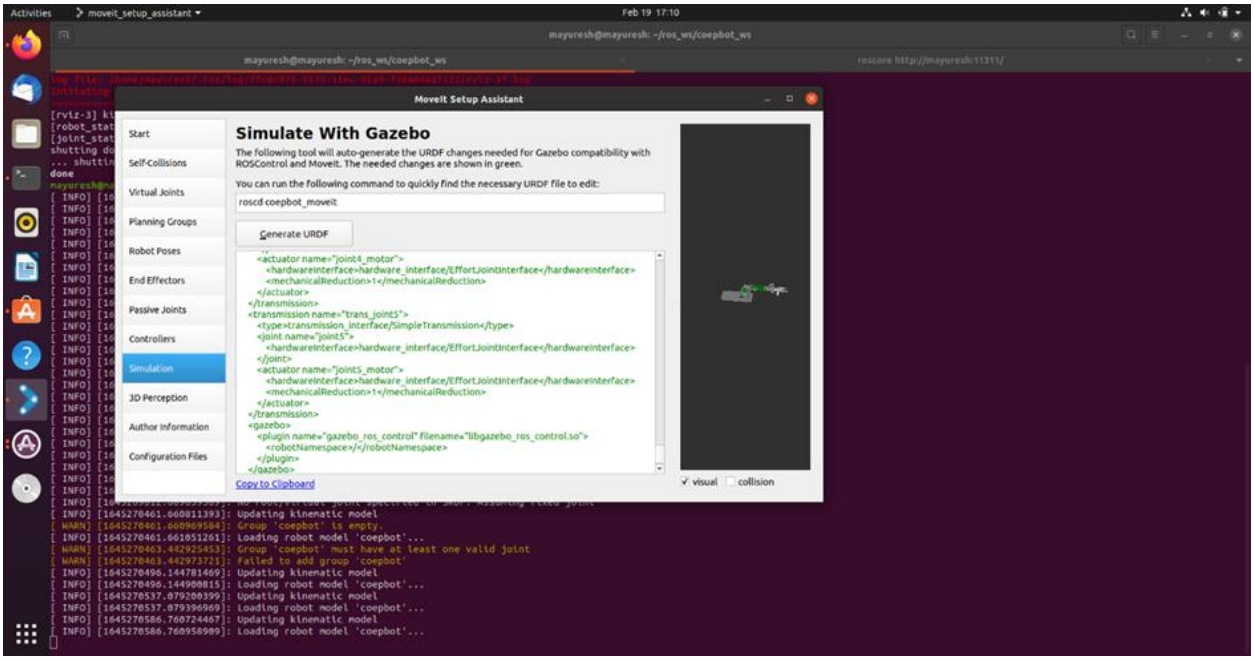



**Fig 5.16:** URDF to Simulate with Gazebo

## 9. Add author information

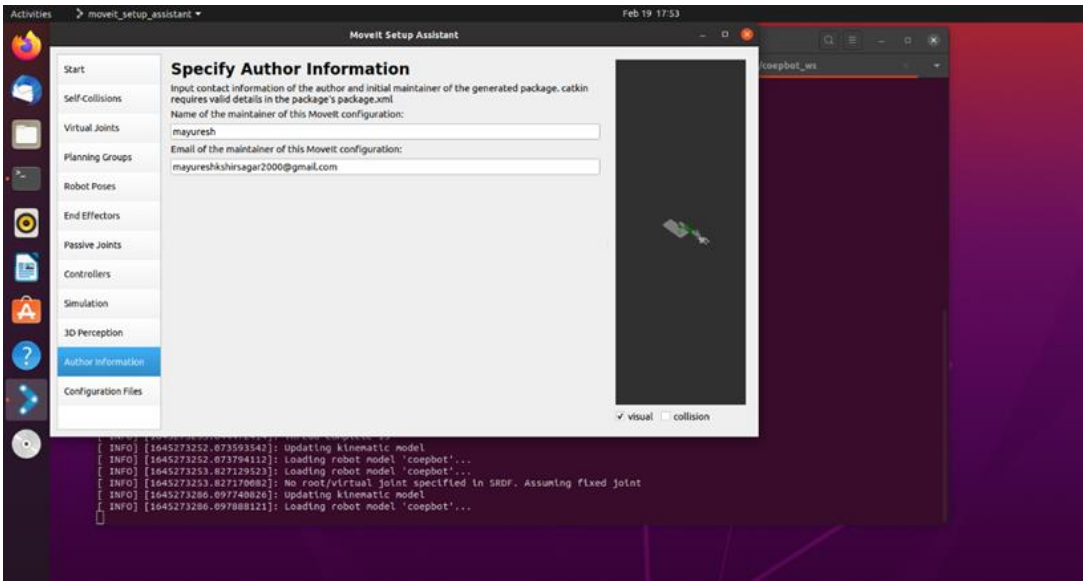



**Fig 5.17:** Author Information

## 10. Generate configuration files

Create a new config folder, put the configuration file in this folder,

Selected the folder, clicked the “Generate Package” button, and generate the configuration file.

#### 5.2.3.2 MoveIt configuration package

If we open the folder just created, we find that there are two folders, config and launch.

**config folder**

**fake_controllers.yaml：**This is a virtual controller configuration file, which is convenient for us to run MoveIt without a physical robot or even any simulator (such as gazebo).

**joint_limits.yaml：**Here is the limit of the position, velocity and acceleration of each joint of the robot, which will be used in future planning.

**kinematics.yaml：**Things set by the motion planning group, used to initialize the kinematics solution library.

**dofbot.srdf：**This is an important MoveIt configuration file.

**ompl_planning.yaml：**There are various parameters for configuring various algorithms of OMPL.

**SRDF file：**SRDF is the configuration file of moveit and is used in conjunction with URDF.

**launch folder**

**demo.launch：**Demo is the summary point of the operation. Open it and we can see that it includes other launch files.

**move_group.launch：**Its function is to make a planning group move.

**planning_context.launch：**We can see that the urdf and srdf files used are defined, as well as the kinematics solution library. We can't change these contents manually.

**setup_assistant.launch：**If we need to change some configuration, we can run it directly.

### 5.2.4 Visualization and Simulation

While designing and debugging robotics software, it often becomes necessary to observe some state while the system is running, ROS being modular, we can tap into any message stream on the system, we can write simple programs which subscribe to a particular topic and plot the data, however a more powerful visualization program which utilizes a plugin architecture is rviz which is distributed with ROS.

Rviz stands for ROS visualization, it is a package developed by willow garage and is an extremely powerful visualization tool which allows the user to visualize data using the plugins for many kinds of available topics. It can be used to view a large variety of data types, such as images, point clouds, geometric primitives, render robot poses and trajectories, etc. Rviz subscribes to sensor topics and additionally the URDF file can be utilized to visualize the robot in a three-dimensional space.

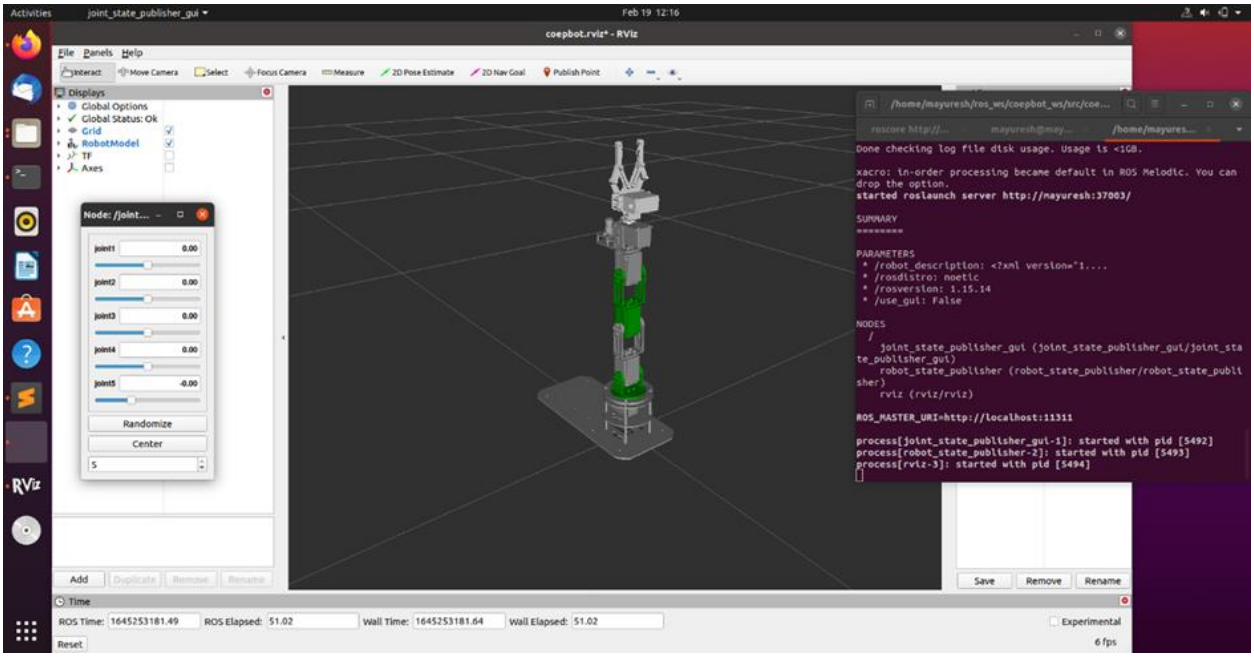



**Fig 5.18:** Visualization using MoveIt and RViz Plugin

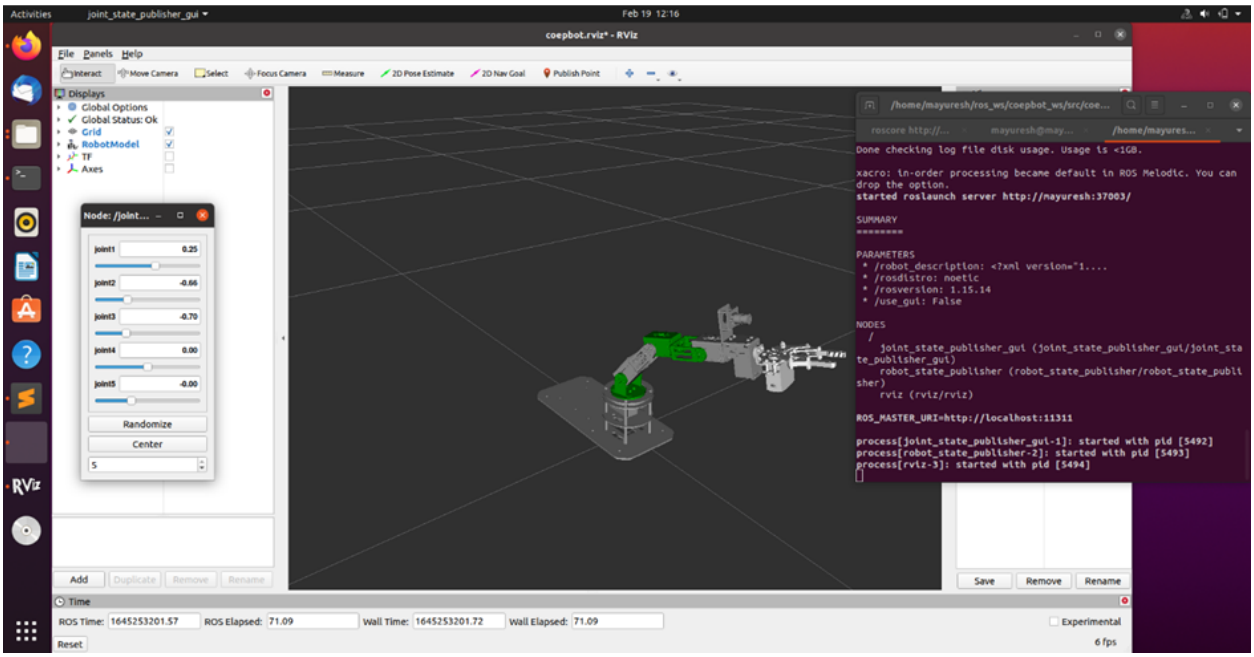



**Fig 5.19:** Motion Planning

**Gazebo** is a powerful real time robot simulator used by professionals and researchers that accounts for real world physics while generating sensor data. Gazebo couples the visualization with the simulation.

Not every situation requires simulation and gazebo integrates visualization along with simulation which makes it tedious to use in cases where it is only required to visualize the data, hence Rviz and Gazebo have their own applications and are both necessary.

For simulation, we wrote python scripts to simulate certain behavior in MoveIt, the primary user interface is through the MoveGroup class. It provides easy to use functionality for most operations that a user may want to carry out, specifically setting joint or pose goals, creating motion plans, moving the robot, adding objects into the environment and attaching/detaching objects from the robot. We used the moveit_commander library. The moveit_commander Python package offers wrappers for the functionality provided in MoveIt. Simple interfaces are available for motion planning, computation of Cartesian paths, and pick and place.

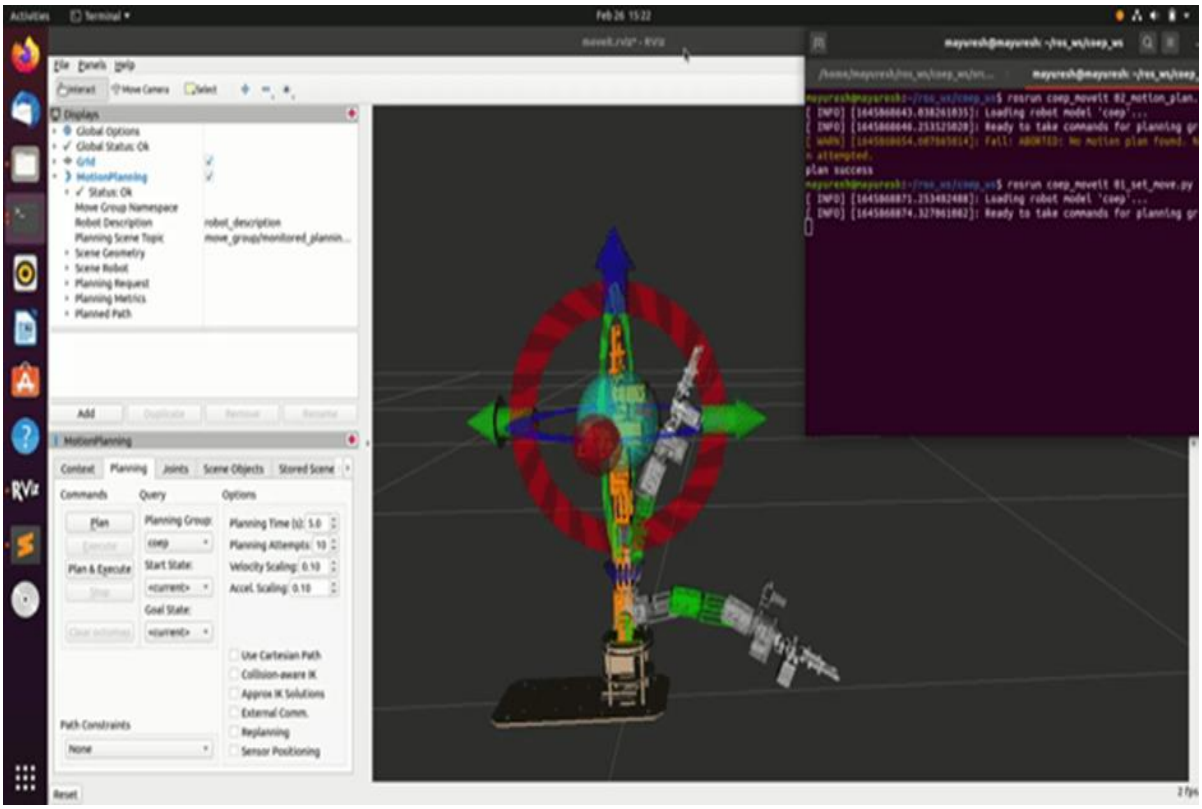



**Fig 5.20:** random valid point simulation

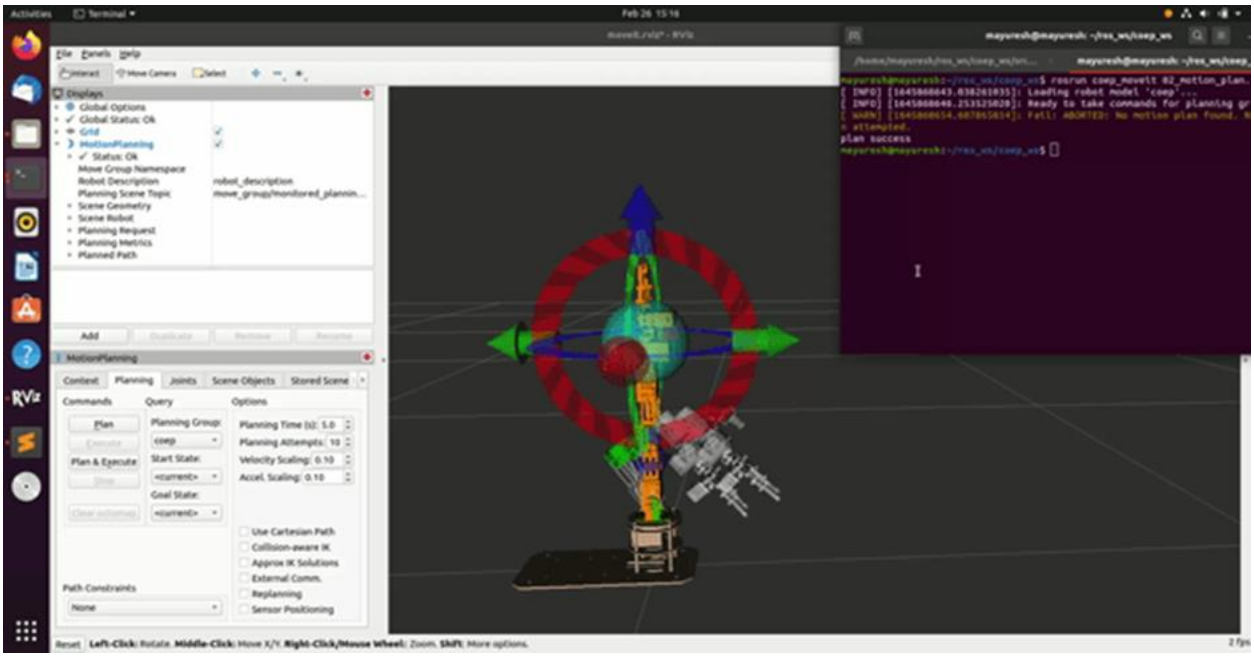


**Fig 5.21:** Planned motion simulation

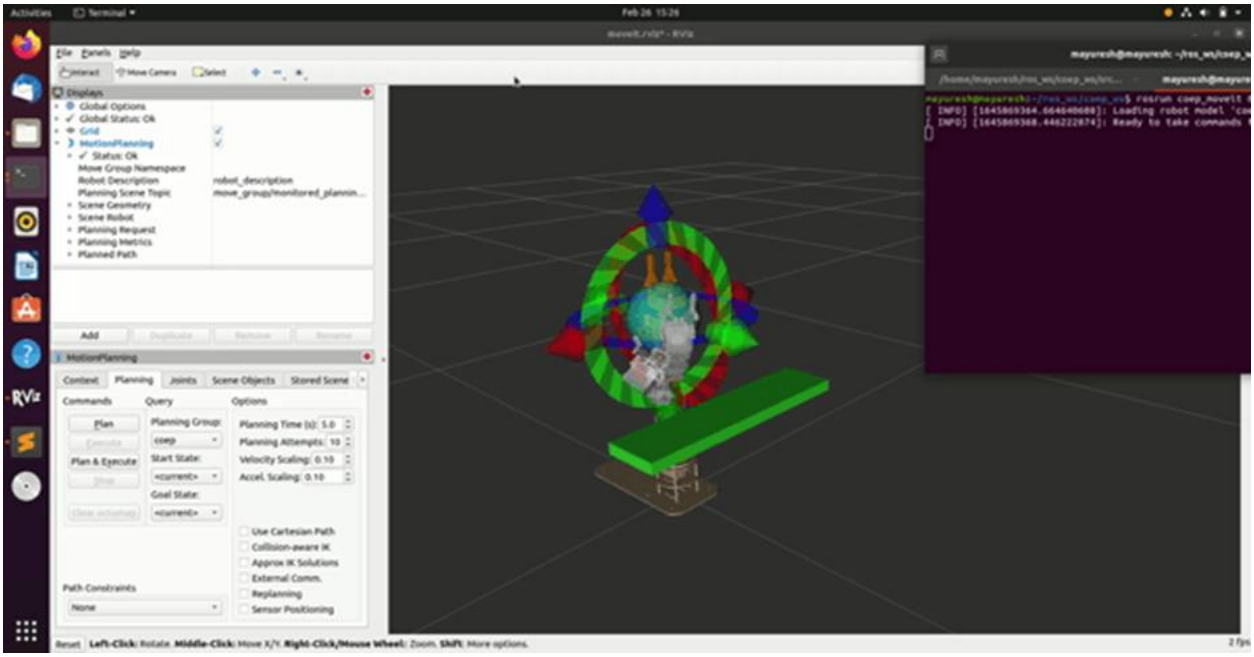


**Fig 5.22:** Object avoidance

### 5.3 Move-Commander Library

The moveit_commander Python package offers wrappers for the functionality provided in MoveIt. Simple interfaces are available for motion planning, computation of Cartesian paths, and pick and place. The moveit_commander package also includes a command line interface, moveit_commander_cmdline.py.

We connect the publishing nodes of the moveit_commander package to the input of the arm library. The arm library has various functions and commands in it which help to control the joints of the robot, controlling the servos.

So the basic flow is

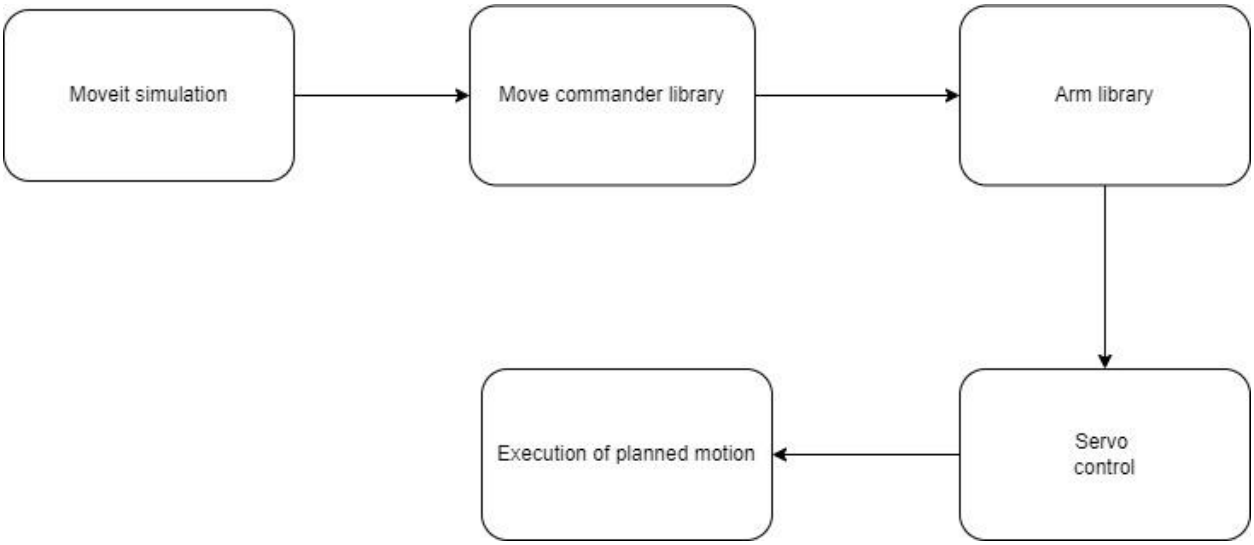



**Fig 5.23:** Hardware control

### 5.4 ROS commands summarized

**roscore** – Starts ROS master.

**rospack** – A tool for inspecting packages.

**roscd** – Changes directories to a package.

**rosls** – Lists package information.

**roscreate-pkg** – Creates a new ROS package.

**rosmake** – Builds a ROS package.

**rospack** find <package_name> – Prints file path to package.

**rxdeps** – displays package structure and dependencies.

**rosrun** <package_name> <node_name> – Starts an executable node.

**roslaunch <package_name> <launch_file>** – Starts launch file.

**rostopic list** – Lists all active topics

**rostopic info </topic_name>** – Provides data on topic such as type, subscriber and publisher.

**rostopic echo </topic_name>** – Prints the topic message to screen.

**rosnode list** – Lists all the active nodes.

**rosnode info <node_name>** – Provides data on nodes.

**rosrun rqt_graph rotograph** – Graphical representation of active packages, nodes and topics.

**rosrun rviz rotograph** – Starts Rviz (Visualization Tool).

**rosbag record -a** – record all topics.

**rosbag record <topic_1> <topic_2>** – Records only the required topics

## 5.5 Results

Our proposed pick and place approach has as key advantage the ability of being easily adaptable to the ROS framework. Since ROS is becoming one of the most useful tools in robotics nowadays, the possibility of using a methodology able to be expressed in ROS allows for the development of a standard approach to pick and place operations. Another advantage of our proposed pick and place approach is the ability to have a robot safely and efficiently inserted in an unknown environment. We successfully performed a pick and place operation using an object detection model. So when we place an apple or orange in front of the robotic arm, it uses the usb camera to capture the image of the environment and passes it to the deep learning model. Our fruit detection model then predicts the class of the fruit, here it is either apple or orange. Depending on the predicted class ROS launches a node to pick an object from the position to a predefined location. Joint values from the published node are passed to the arm library by move commander to control the hardware as described in figure 5.24. Following four figures shows the entire pick and place operation.

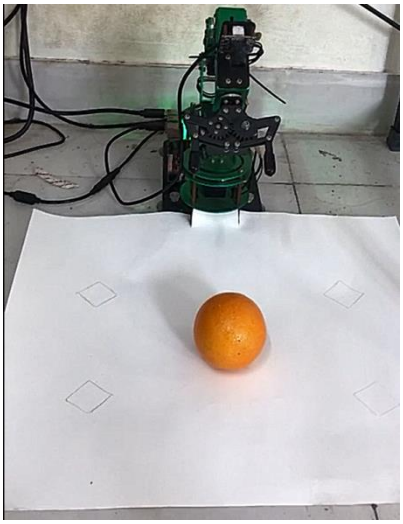


**Fig 5.24(a):**Detects orange

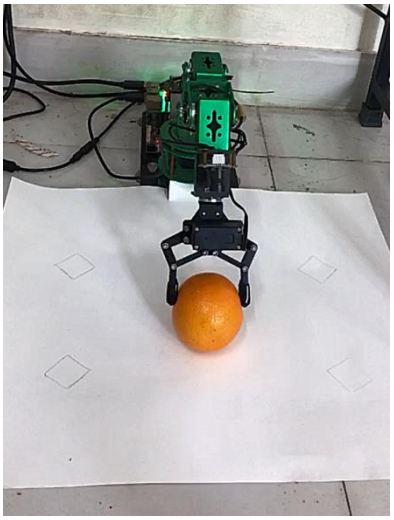


**Fig 5.24(b):**Picks(middle)

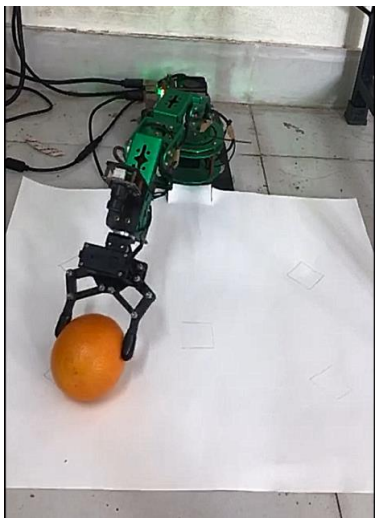


**Fig 5.24(c):**Places(right)

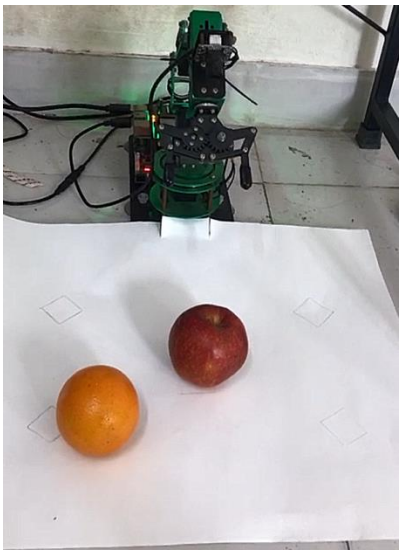


**Fig 5.24(d):**Detects apple

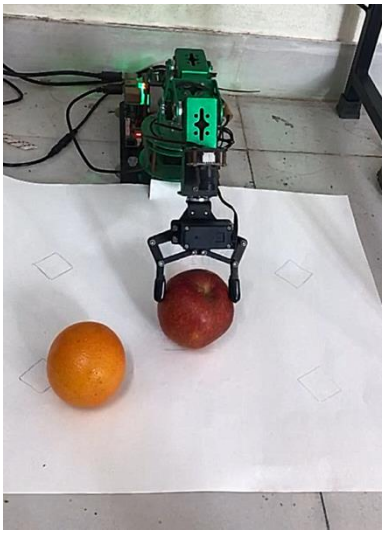


**Fig 5.24(e):**Picks(middle)

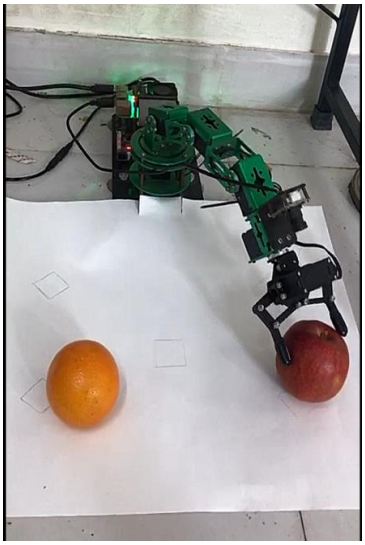


**Fig 5.24(f):**Places(left)

**Fig 5.24:** Pick and Place operation

The insertion of intelligent methodologies such as dictionaries and decision trees allowed the development of a ROS node where all the system is correctly identified.

# Chapter 6
# Future Scope

A multimodal teleoperation system could be developed to control the robotic arm. Various android applications could also be developed to teleoperate the robot.

1. Thick skin could be utilized in the future. It is basically an electronic skin that gives robots a sense of touch. Known as 'Wootzkin' and developed at the University of Edinburgh, the electronic skin is made up of nanostructures and includes underlying electronics that can be used in targeted drug delivery or in gripper technology. Like human skin, it can give the robot feedback on force, pressure, temperature and humidity, making it easier for robots to complete tasks that require a high level of dexterity. The sensitive area size can be changed, between 50 microns and 12 inches, tailoring the robot to the specific needs of the application. Wootzkin operates in temperatures from 0 – 180°C, which means that a robot can perform dexterous tasks under conditions that humans cannot endure, while maintaining a delicate approach to fragile items.
2. Augmented reality can be incorporated in the manipulator system. Augmented Reality (AR) technology has emerged as a novel, effective method to interact with and control robots, while planning the path and orientation of the robotic arm. The AR-based approach also helps users to create control points and define orientation of the arm linked with each control point. The AR technology is significantly enabling robotic arm to accomplish pick and place tasks through a collision-free patch. As part of technology, convergence between augmented reality and robotics in recent years has led manufacturers to employ programing in various collaborative robots. Augmented reality also helps the user to digitally plan precision movements of the robotic arm and place the component or object in the exact place. Moreover, previewing the path of robotic end of arm tools also allows the user to plan for interference and collisions.
3. The robotic arm could also be controlled by electric signals from the brain. This new type of robotic arm technology is controlled by electronic signals that are sent from muscles in the human body. A robot arm, a machine-learning algorithm and a brain-computer interface have been combined to perform the task. In tests, it proved effective for

providing gripping strength for turning keys in locks and other similar tasks. The person's brain activity was monitored by an EEG cap - which effectively scans the electrical activity inside person’s head. These brain waves would then be sent through a computer to be interpreted by the machine-learning algorithm. The algorithm translates the brain signals when the person notices an error, inferring automatically when the brain doesn't like a certain action.

4. The proposed robot doesn’t have the ability to grasp unknown objects that could be altered by incorporating deep reinforcement learning. It is performed by pose estimation of objects, which is the specific task of determining the pose of an object in an image The pose estimation problem can be solved in different ways depending on the image sensor configuration, and choice of methodology. Analytic or geometric methods: Given that the image sensor (camera) is calibrated and the mapping from 3D points in the scene and 2D points in the image is known. If also the geometry of the object is known, it means that the projected image of the object on the camera image is a well-known function of the object's pose. Once a set of control points on the object, typically corners or other feature points, has been identified, it is then possible to solve the pose transformation from a set of equations which relate the 3D coordinates of the points with their 2D image coordinates. Algorithms that determine the pose of a point cloud with respect to another point cloud are known as point set registration algorithms, if the correspondences between points are not already known. Genetic algorithm methods: If the pose of an object does not have to be computed in real-time a genetic algorithm may be used. This approach is robust especially when the images are not perfectly calibrated. In this case, the pose represents the genetic representation and the error between the projection of the object control points with the image is the fitness function. Learning-based methods: These methods use artificial learning-based systems which learn the mapping from 2D image features to pose transformation. In short, this means that a sufficiently large set of images of the object, in different poses, must be presented to the system during a learning phase. Once the learning phase is completed, the system should be able to present an estimate of the object's pose given an image of the object.

# Chapter 7
# Conclusions

1. Our approach based on three tiers is proposed: (1) recognition / perception / sensing; (2) movement / grabbing / using end effector and (3) the control / adaption tier. Such an approach is flexible.
2. Improvements in perception can be made in the training process like data augmentation, class imbalance, cost function, soft labeling etc. to advance accuracy.
3. The accuracy of our model is 51% that could be improved to above 90%.To improve accuracy, we can design a deeper network to extend the receptive field and to increase model complexity. And to ease the training difficulty, skip-connections can be applied. We can expand this concept further with highly interconnected layers.
4. We could use the evolutionary algorithm to select the right set of hyperparameters for training our model. The Evolutionary Algorithms are an educated guess method. It follows the concept of survival of the fittest. For example, we select 100 sets of hyperparameters randomly. Then, we use them for training 100 models. Later, we select the top 10 performed models. For each selected model, we create 10 slightly mutated hyperparameters according to its original. We retrain the models with the new hyperparameters and select the best models again. As we keep the iterations, we should find the best set of hyperparameters.
5. The loss of our model is 0.12, that could be further reduced by training on a large dataset and more iterations. We performed 10000 iterations to train the model.
6. Our model is able to run at a speed of 120 fps and that could be increased above 200 fps by use of Faster R-CNN models. We can also get a higher frame rate by reducing the image size to a multiple of 32.
7. The fruit sorting robot arm has been implemented with minimizing jittering issues caused by the number of servo motors used. We managed to use an existing motion planning library (MoveIt) and exploit all of its desirable features and compensate for its shortcomings.

8. We used MoveIt collision space computation module, which allows adding and removing collision objects and defining allowed collisions with obstacles if necessary. In addition, MoveIt can detect and avoid self-collision for any robot and plan collision free motion trajectories.
9. We managed to use joint constraints that are computationally inexpensive in contrast to orientation constraints and generate motions that mimic the effect of an orientation constraint in a reasonable time frame.

In conclusion, the work presented formalizes an easy and efficient approach to pick and place problems. The proposed approach can be valuable in the field of robotics and can be potentially applied in multiple tasks.

# References

[1] D. Gier and M. Rojerur "Control of a robotic arm: Application to on-surface 3D-printing." (2015).

[2] T. Kyniazopoulou, T. Angeliki, and S. Behnke. "Motion Planning Strategy For a 6-DOFs Robotic Arm In a Controlled Environment." no. August (2017).

[3] A. Khasim, M. Khan, R. Konda, and J. Ryu. "ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task." Advances in robotics research 2.2 (2018): 113.

[4] S. More, et al. "Robotic jaw for object sorting using raspberry pi." vol 21: 1-5.

[5] K. Intisar, R. Muhatasim, et al. "Computer Vision Based Robotic Arm Controlled Using Interactive GUI." INTELLIGENT AUTOMATION AND SOFT COMPUTING" 27.2 (2021): 533-550.

[6] V. Dengg and M. Alexander. Integration of an industrial robot manipulator in ROS to enhance its spatial perception capabilities. Diss. FH Vorarlberg (Fachhochschule Vorarlberg)

[7] R. Dewi, U. Tresna, P. Risma, and Y. Oktarina. "Fruit sorting robot based on color and size for an agricultural product packaging system." Bulletin of Electrical Engineering and Informatics 9.4 (2020): 1438-1445

[8] Z. Gan and Q. Tang, "Robot kinematic calibration," in Visual Sensing and its Applications, pp. 166–192, Springer Berlin Heidelberg, Berlin, Germany, 2011

[9] M. Ceccarelli, Fundamentals of Mechanics of Robotic Manipulation Tle, Kluwer Academic Publishers, Dordrecht, Netherlands, 27th edition, 2004.

[10] M. Shimizu, H. Kakuya, and W. Yoon, "Analytical inverse kinematic computation for 7-DOF redundant manipulators with joint limits and its application to redundancy resolution," IEEE Transactions on Robotics, vol. 86, 2008

[11] J. Denavit, "A kinematic notation for lower-pair mechanisms based on matrices," Journal of Applied Mechanics, pp. 215–221, 1955.

[12] J. B Zhuang, "Review of drug treatment for Down's syndrome persons 'a complete and parametrically continuous kinematic model for robot manipulators', robotics and automation," American Journal of Mental Deficiency, vol. 80, no. 4, pp. 388–393, 1976

[13] C. Robots, Coamu NM45 Specifications, Industrial-Robots, Robots, Bedford, MA, USA, 2015, http://industrial-robotics.co.uk/comau/nm_spec.htm.

[14] J. Craig, Introduction to Robotics: Mechanics and Control, Pearson Education, Inc. Pearson Prentice Hall, London, UK, 2005

[15] M. Summers, “Robot capability test and development of industrial robot positioning system for the aerospace industry,” SAE Technical Paper Series, vol. 114, no. 1, pp. 1108–1118, 2005.

[16] M. Spong, S. Hutchinson, and M. Vidyasagar, “Robot Modeling and Control,” John Wiley & Sons, Inc., Hoboken, NJ, USA, 2006.

[17] S. Megahed, “Inverse kinematics of spherical wrist robot arms: analysis and simulation,” Journal of Intelligent and Robotic Systems, vol. 5, no. 3, pp. 211–227, 1992

[18] C Pathanjali, P. Kumar , P. Somvanshi , , V. Garg”A Comparative Study of Object Detection Algorithms in A Scene”,vol 08, MAY 2019

[19] I. Moll, L. Sucan, M. Kavraki,”The open motion planning library”, IEEE Robotics Automation Magazine 19(4) (Dec 2012)

[20] A. Farhadi, J. Redmon , R. Girshick, S. Divvala “YOLO research paper”: https://arxiv.org/pdf/1506.02640v5.pdf

[21] More information on the yolo research paper and their other publications : https://pjreddie.com/publications/

[22] A. Bochkovskiy, C. Wang and H. M. Liao “YOLOv4: Optimal speed and accuracy of object detection”: https://arxiv.org/abs/2004.10934

[23] YOLOv4 medium article : https://medium.com/aiguys/yolo-v4-explained-in-full-detail-5200b77aa825

[24] Open Image Dataset : https://storage.googleapis.com/openimages/web/index.html

[25] A. Dattalo. (2018, August 08). ROS/introduction. [Online].

[26] I. Saito. (2019, April 14). Packages. [Online].

[27] D. Woods. (2021, September 08). Understanding Nodes [Online].

[28] T. Foote (2019, February 20). Topics [Online].

[29] T. Foote, E. Marder-Eppstein, W. Meeussen. tf [online]

[30] A. Koubaa (2019, July 18). Services [Online]

[31] ROS literature : https://wiki.ros.org/