

Ain Shams University  
Faculty of Engineering

**Autonomous & Manually Controlled Upper Humanoid Robot**

Submitted by:

|  |  |
| --- | --- |
| Robeir Remon Farid | 1500594 |
| Rami Wafik Attia | 1500560 |
| Hassan Sami Fahmy |  |
| Alaa Ayman Elremaily |  |
| Aya Ayman Elremaily |  |

Supervised by:

**Dr. Mohammed Ibrahim Dr. Shady Ahmed Maged**

Table of Contents

[1. Introduction 5](#_Toc42464083)

[1.1. Background 5](#_Toc42464084)

[1.2. Historical brief 6](#_Toc42464085)

[1.3. Motivation 8](#_Toc42464086)

[1.4. Aims 8](#_Toc42464087)

[1.5. Objectives 8](#_Toc42464088)

[2. System overview 9](#_Toc42464089)

[2.1. System schematic 9](#_Toc42464090)

[2.2. Human motion 9](#_Toc42464091)

[2.3. VDI engineering design model 10](#_Toc42464092)

[3. Conceptual design 11](#_Toc42464093)

[3.1. Design considerations 11](#_Toc42464094)

[3.1.1. DOF for arms 11](#_Toc42464095)

[3.1.2. Head functions 11](#_Toc42464096)

[3.1.3. Materials and weight 11](#_Toc42464097)

[3.1.4. Availability of market 11](#_Toc42464098)

[3.2. Requirements 12](#_Toc42464099)

[4. InMoov 13](#_Toc42464100)

[4.1. Introduction 13](#_Toc42464101)

[4.2. Mechanical 13](#_Toc42464102)

[4.2.1. Mechanical design 13](#_Toc42464103)

[4.3. URDF model 13](#_Toc42464104)

[4.4. Electrical design 19](#_Toc42464105)

[4.4.1. Circuit design 19](#_Toc42464106)

[4.4.2. PCB layouts 19](#_Toc42464107)

[4.4.3. Wiring 19](#_Toc42464108)

[4.4.4. Circuit schematic 19](#_Toc42464109)

[4.4.5. Power calculations 19](#_Toc42464110)

[4.5. Sensor selection 19](#_Toc42464111)

[4.6. GPU selection 19](#_Toc42464112)

[5. Manual arm control 21](#_Toc42464113)

[5.1. Introduction 21](#_Toc42464114)

[5.2. Exoskeleton 21](#_Toc42464115)

[5.2.1. Arm control 21](#_Toc42464116)

[5.2.2. Fingers control 21](#_Toc42464117)

[5.3. Wireless joystick 21](#_Toc42464118)

[5.4. Kinect control 21](#_Toc42464119)

[5.5. Head control using VR 21](#_Toc42464120)

[6. Autonomous arm control 23](#_Toc42464121)

[6.1. Catching a ball 23](#_Toc42464122)

[6.1.1. Introduction 23](#_Toc42464123)

[6.1.2. Ball detection and tracking 23](#_Toc42464124)

[6.1.3. Trajectory estimation 25](#_Toc42464125)

[6.1.4. Inverse kinematics 27](#_Toc42464126)

[6.1.5. Connecting Jetson Nano to the servos 27](#_Toc42464127)

[6.2. Object detection using neural networks 27](#_Toc42464128)

[6.2.1. Using SSD (Single Shot Multibox Detection) 27](#_Toc42464129)

[6.2.2. Testing the detection of a person and a dog 28](#_Toc42464130)

[6.3. Speech Recognition 28](#_Toc42464131)

[7. References 29](#_Toc42464132)

**Table of Figures**

[Figure 1 Complete view for our model 15](#_Toc46690405)

[Figure 2 URDF Model 1 16](#_Toc46690406)

[Figure 3 URDF Model 2 16](#_Toc46690407)

[Figure 4 URDF Model Left Hand 17](#_Toc46690408)

[Figure 5 URDF Model Right Hand 17](#_Toc46690409)

[Figure 6 URDF Model Head 18](#_Toc46690410)

[Figure 7 URDF Model in Rviz 1 18](#_Toc46690411)

[Figure 8 URDF Model in Rviz 2 19](#_Toc46690412)

[Figure 9 URDF Model in Rviz 3 19](#_Toc46690413)

[Figure 10 Depth using traingulate 10 inch 28](#_Toc46690414)

[Figure 11 Depth using triangulate 10 inch 28](#_Toc46690415)

[Figure 12 Detect Depth in 13 inch (D) 29](file:///C:\Users\Robeir\Desktop\Inmoov_graduation\Inmoov%20Thesis%202020.docx#_Toc46690416)

[Figure 13 Detect Box in 13inch depth 29](file:///C:\Users\Robeir\Desktop\Inmoov_graduation\Inmoov%20Thesis%202020.docx#_Toc46690417)

[Figure 14 Depth of the Box is D in inch 30](file:///C:\Users\Robeir\Desktop\Inmoov_graduation\Inmoov%20Thesis%202020.docx#_Toc46690418)

[Figure 15 Depth of box in 56 cm 31](#_Toc46690419)

[Figure 16 Depth of box using kinect 31](file:///C:\Users\Robeir\Desktop\Inmoov_graduation\Inmoov%20Thesis%202020.docx#_Toc46690420)

[Figure 17 Depth of box 240cm 32](file:///C:\Users\Robeir\Desktop\Inmoov_graduation\Inmoov%20Thesis%202020.docx#_Toc46690421)

[Figure 18 Detected depth for box is 243.89 cm 32](#_Toc46690422)

[Figure 19 Test Path Estimation Juggling 1 Figure 20 Test Path Estimation Juggling 2 33](#_Toc46690423)

[Figure 21 Test Path Estimation Juggling 3 Figure 22 Test Path Estimation Juggling 4 34](#_Toc46690424)

[Figure 23 Test Path Estimation Juggling 5 Figure 24 Test Path Estimation Catching 34](#_Toc46690425)

[Figure 25 PCA 9685 37](#_Toc46690426)

[Figure 26 Detect Dog with DetectNet Neural Network 40](#_Toc46690427)

[Figure 27 Detect Person with 98.8 % 41](#_Toc46690428)

[Figure 28 Detect Person with 98.4 % using DetectNet Netural Network on Jetson Nano 41](#_Toc46690429)

# Introduction

## Background

Humanoid robots, while being one of the smallest groups of service robots in the current market, have the greatest potential to become the industrial tool of the future.

It has been well documented that there will be increase in the number of robots over the next decade. According to the Boston Consulting Group, by 2025, robots will perform 25% of all labor tasks. This is due to improvements in performance and reduction in costs.

Before the coronavirus pandemic and the economic uncertainty, Statistics Market Research Consulting expected that the Global Humanoid Robot Market would reach $13 billion by 2026. While future market behavior is now unclear, robot usage is on the rise: Chinese companies were rushing to deploy robots and automation technology, as doctors were grappling with COVID-19.

Developing a humanlike robot involves copying the appearance of humans as well as emulating the capabilities, expression of emotions and possibly even having thoughts. Making such robots involves advances in many disciplines including mechanical and electrical engineering, materials science, computer science, artificial intelligence, and control. To make such smart machines that look and act like a human there is a need to integrate capabilities that are at the cutting edge of the related technology. The materials to be used need to be resilient, lightweight and multifunctional.

These robots need sensors to visualize the train, hear sound, as well as sense touch, pressure and temperature. The robots need to use light batteries or generator for power that can be operated over a long time without recharge. In addition, the robots need to interpret the information that is measured by the sensors to perceive and be aware of the surrounding terrain and to sense hazards and risks. Humanlike robots need to have effective control and artificial intelligence algorithms in order to be operated like humans and interact with its environment and humans.

Developing a humanlike robot involves copying the appearance of humans as well as emulating the capabilities,

expression of emotions and possibly even having thoughts. Making such robots involves advances in many

disciplines including mechanical and electrical engineering, materials science, computer science, artificial

intelligence, and control. To make such smart machines that look and act like a human there is a need to integrate

capabilities that are at the cutting edge of the related technology [Bar-Cohen and Breazeal, 2003; Bar-Cohen and

Hanson, 2009]. The materials to be used need to be resilient, lightweight and multifunctional. Further, the mobility

system that is used needs to allow walking via two legs and maintain stability while able to traverse complex terrains

(e.g. climb stairs and avoid obstacles). These robots need sensors to visualize the train, hear sound, as well as sense

touch, pressure and temperature. The robots need to use light batteries and/or generator for power that can be

operated over a long time without recharge. In addition, the robots need to interpret the information that is measured

by the sensors to perceive and be aware of the surrounding terrain and to sense hazards and risks. Humanlike robots

need to have effective control and artificial intelligence algorithms in order to be operated like humans and interact

with its environment and humans [Plantec, 2003]. The produced robots need to have body parts and related

functions similar to a human as much as possible

machines were driven by mechanical energy that was stored in a spring. Examples include the "The Flute Player"

that was produced by the French engineer Jacques de Vaucanson in 1737 and the “Writer” that was made by the

Swiss clockmaker Jacquet-Droz and completed in 1772. The era of robotics as we know it today where the machine

is equipped with artificial intelligence has began in 1946 with the first introduction of the digital computer, the

ENIAC computer, which was the first large-scale general-purpose electronic computer [McCartney 1999]. The first

time that the possibility of building thinking and learning machines was raised was in 1950 [Turing, 1950]. Progress

in developing powerful microprocessors with high computation speed, very large memory, wide communication

bandwidth, and more effective software tools made the most impact on the development of intelligent robots. With

the advancements in microelectronics and intelligent software more sophisticated robots have been emerging with

concepts and methodologies that are inspired and guided by nature [Arkin 1998; Bar-Cohen and Breazeal, 2003;

Bar-Cohen and Hanson, 2009; Gould, 1982].

With the advances in technology, humanlike robots are increasingly becoming easier to make as lifelike using

effective autonomous operation algorithms, humanlike materials, and the capability to emulate the movement and

functionality (seeing, hearing, smelling, etc.) of humans. Using state of the art microprocessors, materials, sensors,

software, and many other technologies are leading to increasingly more capable robots. These advances are

allowing them to perceive, interpret, respond, and adapt to their environment. Robotic products are already being

developed for entertainment, education, healthcare, home security, military, and many others. Currently,

entertainment applications are the most beneficiary of this technology where humanlike robotic toys are

commercially available in many stores. Further, industry has begun to collaborate with scientists to make their

characters in movies appear more realistic and to move more like people. Also, robotics researchers are increasingly

collaborating with artists to make their robots appear more expressive and believable.

3. MAKING A HUMANLIKE ROBOT

Developing a humanlike robot involves copying the appearance of humans as well as emulating the capabilities,

expression of emotions and possibly even having thoughts. Making such robots involves advances in many

disciplines including mechanical and electrical engineering, materials science, computer science, artificial

intelligence, and control. To make such smart machines that look and act like a human there is a need to integrate

capabilities that are at the cutting edge of the related technology [Bar-Cohen and Breazeal, 2003; Bar-Cohen and

Hanson, 2009]. The materials to be used need to be resilient, lightweight and multifunctional. Further, the mobility

system that is used needs to allow walking via two legs and maintain stability while able to traverse complex terrains

(e.g. climb stairs and avoid obstacles). These robots need sensors to visualize the train, hear sound, as well as sense

touch, pressure and temperature. The robots need to use light batteries and/or generator for power that can be

operated over a long time without recharge. In addition, the robots need to interpret the information that is measured

by the sensors to perceive and be aware of the surrounding terrain and to sense hazards and risks. Humanlike robots

need to have effective control and artificial intelligence algorithms in order to be operated like humans and interact

with its environment and humans [Plantec, 2003]. The produced robots need to have body parts and related

functions similar to a human as much as possible

Developing a humanlike robot involves copying the appearance of humans as well as emulating the capabilities,

expression of emotions and possibly even having thoughts. Making such robots involves advances in many

disciplines including mechanical and electrical engineering, materials science, computer science, artificial

intelligence, and control. To make such smart machines that look and act like a human there is a need to integrate

capabilities that are at the cutting edge of the related technology [Bar-Cohen and Breazeal, 2003; Bar-Cohen and

Hanson, 2009]. The materials to be used need to be resilient, lightweight and multifunctional. Further, the mobility

system that is used needs to allow walking via two legs and maintain stability while able to traverse complex terrains

(e.g. climb stairs and avoid obstacles). These robots need sensors to visualize the train, hear sound, as well as sense

touch, pressure and temperature. The robots need to use light batteries and/or generator for power that can be

operated over a long time without recharge. In addition, the robots need to interpret the information that is measured

by the sensors to perceive and be aware of the surrounding terrain and to sense hazards and risks. Humanlike robots

need to have effective control and artificial intelligence algorithms in order to be operated like humans and interact

with its environment and humans [Plantec, 2003].

Developing a humanlike robot involves copying the appearance of humans as well as emulating the capabilities,

expression of emotions and possibly even having thoughts. Making such robots involves advances in many

disciplines including mechanical and electrical engineering, materials science, computer science, artificial

intelligence, and control. To make such smart machines that look and act like a human there is a need to integrate

capabilities that are at the cutting edge of the related technology [Bar-Cohen and Breazeal, 2003; Bar-Cohen and

Hanson, 2009]. The materials to be used need to be resilient, lightweight and multifunctional. Further, the mobility

system that is used needs to allow walking via two legs and maintain stability while able to traverse complex terrains

(e.g. climb stairs and avoid obstacles). These robots need sensors to visualize the train, hear sound, as well as sense

touch, pressure and temperature. The robots need to use light batteries and/or generator for power that can be

operated over a long time without recharge. In addition, the robots need to interpret the information that is measured

by the sensors to perceive and be aware of the surrounding terrain and to sense hazards and risks. Humanlike robots

need to have effective control and artificial intelligence algorithms in order to be operated like humans and interact

with its environment and humans [Plantec, 2003]

The main applications for which these robots are being considered include health-care, entertainment, home or office security, and military. Humanoids are also suitable for some procedurally-based vocations such as operating like hospital workers, receptionists, guards, and more, and they can speak in various languages, dance to the sound of music, and play musical instruments.

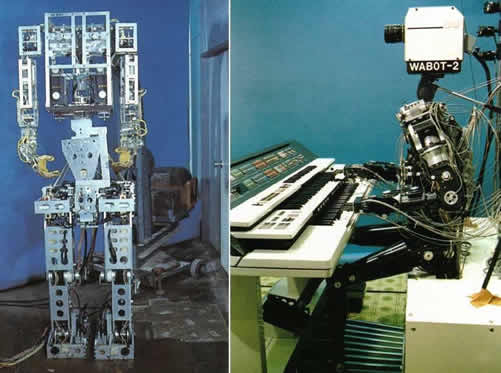
Companies like [**Softbank Robotics**](https://www.softbankrobotics.com/us/) have created human-looking robots to be used as medical assistants and teaching aids. Currently, humanoid robots are excelling in the medical industry, especially as companion robots.

However, companies are now using humanoid robots to fill engineering tasks. A four-year joint research project was conducted by [**Joint Robotics Laboratory**](https://jrl-umi3218.github.io/index.html)and [**Airbus Group**](http://www.airbus.com/) to use humanoid robotic technology in aircraft manufacturing facilities. By using humanoid robots on aircraft assembly lines, Airbus looks to relieve human operators of some of the more laborious and dangerous tasks. The human employers could then concentrate on higher value tasks. The primary difficulty is the confined spaces these robots have to work in and being able to move without colliding with the surrounding objects

## Historical brief

1967

Waseda University started the WABOT project in 1967. The WABOT-1 robot was completed in 1972 and was the world’s first full-scale android humanoid robot. It was the first robot able to walk and communicate with people in Japanese, navigate a room and grip and transport objects. They later went on to create WABOT-2 which was capable of reading a musical score and playing an electric keyboard



2010

[NASA and General Motors revealed Roboaut 2](https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-robonaut-58.html), a highly advanced humanoid robot that was part of the Discovery shuttle launch in 2011. Robonaut was designed to assist NASA with space walks and has enough dexterity to use tools and work alongside astronauts in future space expeditions.

NASA later went on to develop [Valkyrie](http://valkyrie.inf.ed.ac.uk/), a robot that has been developed with the ability to ultimately setup habitats on Mars prior to human arrival.

2014

[Softbank Robotics release Pepper](https://www.softbankrobotics.com/emea/en/pepper), which quickly became the leading commercially available social robot. Pepper was rolled out in Softbank’s mobile stores in Japan and has since been use in Carrefour and Renault stores across France.

2016

[Hanson Robotics release Sophia](https://www.hansonrobotics.com/sophia/), a social robot with silicone skin, and the ability to interact with people and display more than 50 facial expressions. Sophia has been covered my media around the world and has participated in many high-profile interviews. The Sophia robot is also the first ever robot to be granted citizenship of a country.

2020

At a time when healthcare workers at hospitals are prone to coronavirus infection, humanoid robots at AIIMS has mitigated risk by performing contactless monitoring of patients

Milagrow human tech has installed humanoid Milagrow ELF at dedicated COVID-19 wards.



## Motivation

Our motivation for this project is to advance research in this area as it is needed for our country at this moment especially at the coronavirus pandemic because as the virus spreads to the rest of the world, robots are being deployed in many countries.

This project can be used for research and space exploration, personal assistance and caregiving, education and entertainment, search and rescue, manufacturing and maintenance, public relations, and healthcare.

Also, to advance our abilities in the control theory, machine vision and artificial intelligence to be graduated as a modern mechatronics engineers, this project was perfect as it accomplishes this goal.

## Aims

The virtual presence robot can be controlled by a person who is far away from the robot which is used in education and medical use but the person can only see the environment , talk using the tablet and move the robot base but unfortunately he can’t take any action in the robot environment so our aim is to design, manufacture and control a robot that mimics human appearance which can be controlled from a remote location instantly providing a virtual presence and also can perform some human functions like object detection and catching objects on the fly taking into consideration the safety aspect as it will be working in the vicinity of humans which will be used exactly like the old virtual presence design robot also with more options and functions.

## Objectives

Our objective is to provide a relatively cheap humanoid robot with a differential base structure. It can be used among various medical and social or educational applications so that it can be used as a nurse for people infected with coronavirus and also can be used for catching flying objects.

# System overview

## System schematic

## Human motion

Human hand motion during point-to-point reaching was initially observed as straight lines in the task  
space, though this was later disputed by others, who observed curved trajectories. It has been proposed  
that the straight lines were in fact the result of experimental protocols in which the subjects were asked  
to manipulate an instrumented tool to facilitate data capture. Other researchers have suggested that the  
curved motions are due to gravitational effects. It has been observed during this literature review that in  
many of the straight-line motion cases, subjects had their arm movement (including redundant DOF)  
constrained to the horizontal plane, in which changes in gravitational effects are largely negated.

Optimization has been widely accepted as a driver of human motion, though the cost function has been  
disputed. It seems that a number of different cost functions have produced valid results in various scenarios. These seem to evolve over the course of research from purely kinematic considerations (often  
at the joint level) to dynamic quantities, such as muscle forces or change in joint torque.

Though muscle models are often considered as viable aspects of movement cost functions, their high  
contribution to increased model complexity and redundancy means that many researchers only  
implement highly simplified muscle representations in their simulations, in which the muscle quantity and  
placement do not match those of a human. The inclusion of even these factors adds a great deal of  
additional redundancy to the control of a limb.

It is also true that researchers tend to rely on simplified kinematic models of the arm, focusing often on  
2DOF or 3DOF planar systems. Some examples have included three-dimensional movements achieved  
with a 4DOF model of the human arm. Many studies have constrained human motion to planar horizontal  
tasks involving the manipulation of some tool. Generalized movement models have then been based on  
data collected in these specific, gravity-negated scenarios. Modern models that include dynamics and  
kinematics seem much more appealing than pure kinematic models, especially considering the results of  
neurological movement analysis.

There is evidence of both kinematic and dynamic limb and external object representations (models) within  
the human central nervous system. It has been proposed that after periods of learning, where these  
models are created or adapted, the mechanisms of the brain employ feedforward control schemes that  
utilize these models to produce movement abilities that would not be possible with the slow feedback  
mechanisms of the body.

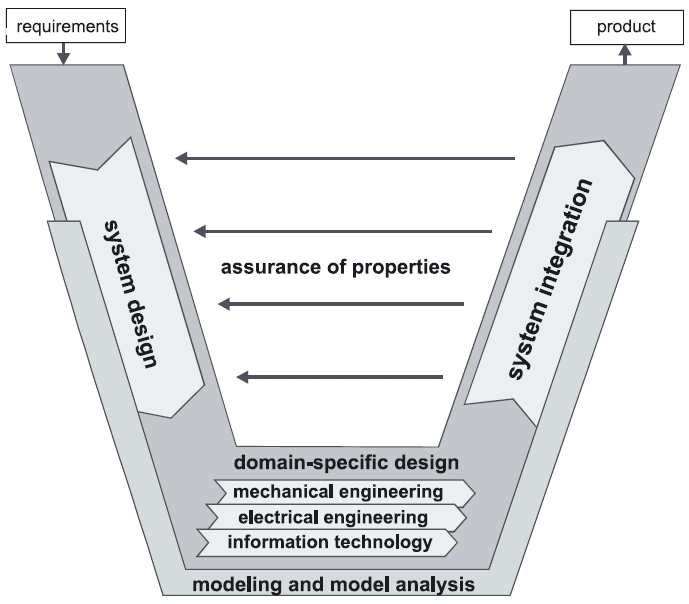
Considering the number of factors that influence human motion, it is proposed that non-exclusively planar  
motion paths are influenced by several aspects, including solving the redundancy problem by optimization of some physiological quantity with additional dependency on external dynamic factors, such as change of gravitational forces and direction of movement [2]

## VDI engineering design model

The German Society for Engineers has published a guideline for mechatronic systems,VDI 2206:2004, which includes a ‘V-model’ of design development by implication, the procedural models of VDI 2225:1975, VDI 2222:1977 and VDI 2221:1985 are included in the ‘domain specific design’. Blanchard [2004] shows a similar model with respect to software systems. We claim a similarity to the Procedural Model of Design Engineering [Hubka 1992b, 1992a, 1996]:

The ‘domain-specific design’ is represented by separate functions in the function structure, which may specify functions that can be realized by mechanical, electrical, chemical, software, or any other system.

‘integration’ can and should take place in any of the relevant structures, but is especially necessary in the constructional structure because cooperation among the specialists is especially necessary here, and The cycle of ‘substantiate, verify, improve’ at the end of each design stage in the Procedural Model [Hubka 1992a, 1992b, 1996] leads to a feedback to any previous stage, not just to the horizontally referenced level, although this level may be the most likely target.



# Conceptual design

The simplest design for virtual presence device is a stick with an iPad and a camera moving on a wheeled base but since we are going for more than that, the plan is to make it able to perform tasks like bring something from shelves move items around the room and for that it was chosen to be an upper body humanoid robot to do these tasks with design considerations based on the tasks performed by it then followed by the requirements to implement a design with this considerations.

## Design considerations

It should have dimensions similar to a human being with friendly looks and moving base on wheels for easy movement.

### DOF for arms

For a humanoid its preferable for the arm to be similar in looks to the human arm, The human arm including the shoulder, elbow, wrist joints and exclusion scapular motion has 7 Degrees of Freedom (DOF) while positioning of the wrist in space and orientating the palm is a task that requires 6 DOF.

As such it includes one more DOF than is needed to complete the task. Given the redundant nature of the arm, while in a robot you will not have such privilege due to design complexity issues, increase of cost for motors and material, as well as weight and inertia increase causing the end effector to be less accurate.

### Head functions

For total presence feeling of the absent user he will need to see hear and talk to the surroundings which will need to equip the head with motors for the head in order to engage better, see everything around by tilting right and left with camera feeding him the view.

For his hearing a Dolby sound surrounding to differentiate between voices from left side to these coming from his right side with dedicated mic to talk back.

### Materials and weight

For fast movement and reach of arms the materials can’t be metallic but rather plastics to be lighter in weight but also handle its own inertia for serial links on shoulders joints.

### Availability of market

For easy maintenances and for most market availability the whole system must be electric rather than hydraulic.

## Requirements

This table defines our system requirements to be satisfied in design according to Function, Performance, Operation, Manufacturing, Cost.

|  |  |
| --- | --- |
| Category | Requirements |
| Functional | 1- Safe to work and interact with people.  2- Can detect objects.  3- Can recognize people.  4- Can serve food and drinks.  5- Can help people in case of quarantined |
| Performance | 1- Velocity Ranges from 0 to 2 m/sec  2- Can handle a relatively light weight object (1kg).  3- Can move in the main 3 axis X, Y, Z |
| Operational | 1- Can move on flat surfaces.  2- Robot learns by imitation using AI.  3- Can handle a relatively small object with human-like speed. |
| Manufacturing | 1- Easy to be manufactured and relatively cheap.  2- The whole robot is an assembly of 3D printed parts.  3- Steel base |
| Environmental Conditions | 1- Can work in normal conditions of temperature and pressure |
| Cost | Up to 25,000 EGP |

# InMoov

## Introduction

## Mechanical

### Mechanical design

#### Upper body design

#### Base design

## URDF model

The URDF (Universal Robot Description Format) model is a collection of files that describe a robot’s physical description to ROS. These files are used by a program called ROS (Robot Operating System) to tell the computer what the robot actually looks like in real life. URDF files are needed in order for ROS to understand and be able to simulate situations with the robot before a researcher or engineer actually acquires the robot, and it’s a part in autonomous control of robots which is the perception.

While URDFs are a useful and standardized format in ROS, they are lacking many features and have not been updated to deal with the evolving needs of robotics. URDF can only specify the kinematic and dynamic properties of a single robot in isolation. URDF can not specify the pose of the robot itself within a world. It is also not a universal description format since it cannot specify joint loops (parallel linkages), and it lacks friction and other properties. Additionally, it cannot specify things that are not robots, such as lights, height maps, etc.

On the implementation side, the URDF syntax breaks proper formatting with heavy use of XML attributes, which in turn makes URDF more inflexible. There is also no mechanism for backward compatibility.

To deal with this issue, a new format called the [Simulation Description Format](http://sdformat.org/) (SDF) was created for use in Gazebo to solve the shortcomings of URDF. SDF is a complete description for everything from the world level down to the robot level. It is scalable, and makes it easy to add and modify elements. The SDF format is itself described using XML, which facilitates a simple upgrade tool to migrate old versions to new versions. It is also self-descriptive.

It is the intention of this author to make URDFs as fully documented and supported in Gazebo as possible, but it is relevant to the reader to understand why the two formats exist and the shortcomings of both. It would be nice if more work was put into URDFs to update them to the current needs of robotics.

URDF model can be made by writing your own xml file if the robot is not complicated and you reached at the end to the tree structure of your robot which is linking the links with the joint between each two links so you can start with the first link name then you have three major sections to describe your link which are the Visual , Inertia and Collision , in the Visual you describe the shape and the dimensions of this link, it has exactly the origin of the link , geometry which you can describe its shape or u can put your .stl file if u have and last part is the material of the link and its color .

Also the Inertia contains the origin and the mass.

The last part in the description of the link is the collision where you can prevent this link from colliding with any other links.

After finishing the links you have to describe the joints between these links ,

It contains the joint name and the parent and child links which are the two links which are linked to this joint then you write the origin and the axis of this joint .

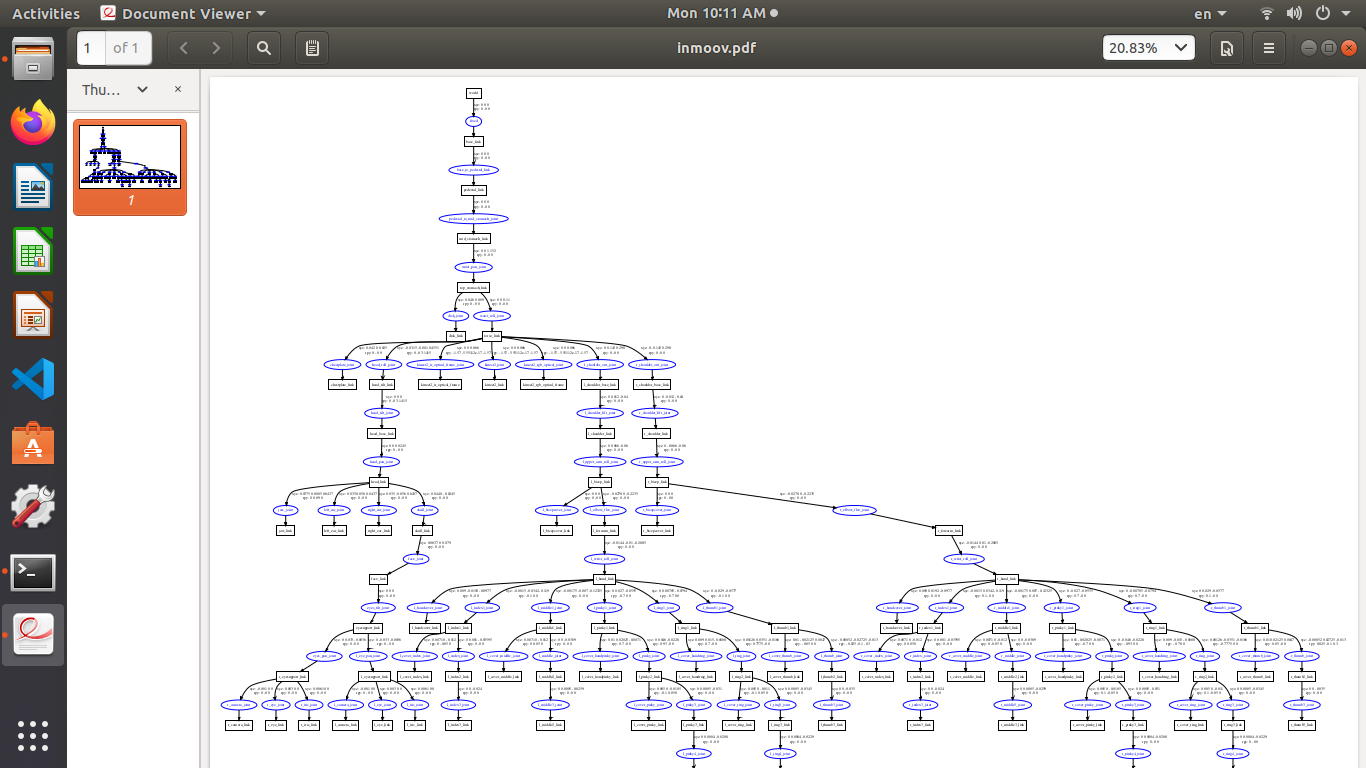


Figure Complete view for our model

This is how we can write a URDF model for our robot there is another way to do the URDF model by extracting it from the solid works or any other CAD program and it generates the file to use it in any Visualization like RVIZ or in Simulation in programs like Gazebo or Vrep .

There is another language or format for writing the URDF files than what we have just mentioned which is XACRO ( XML Macro Lanuage ) used for URDF simplification and u can write many xacro files and combine them in a final file but in urdf file you have to write your own code in only one file and it will be more complicated.

So we have created or own model for Inmoov Robot to use it in the autonomous tasks and we used it in simulation using Vrep and checked the model by several ways first by checking the syntax errors in the xml file by using check\_urdf which is a ros command to check the syntax errors not semantics in the URDF files and also you can see the tree structure of the file ,

Also there is another test to the file by using ( urdf\_to\_graphiz my\_robot.urdf ) which is generate a file contains all the joints and links to see them clearly and you can run this file using evince file\_name.pdf

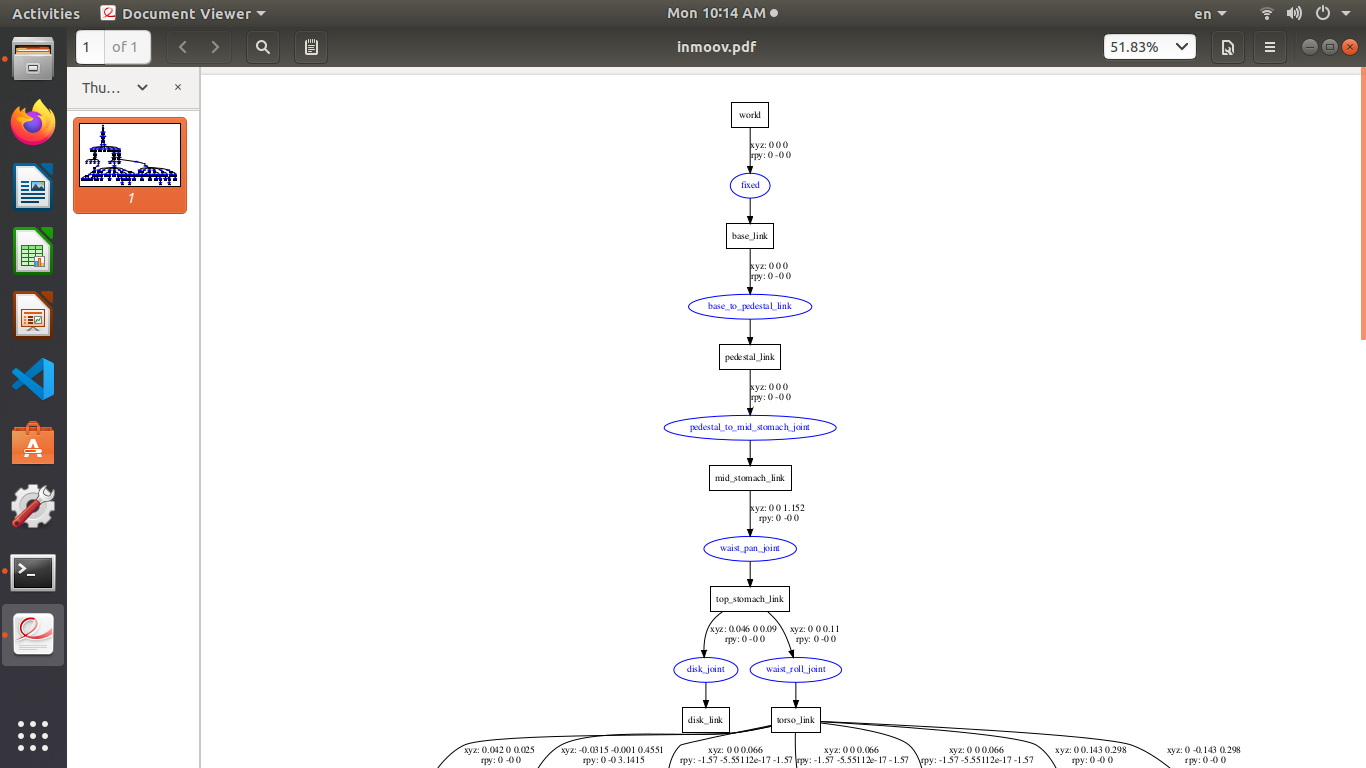


Figure URDF Model 1



Figure URDF Model 2

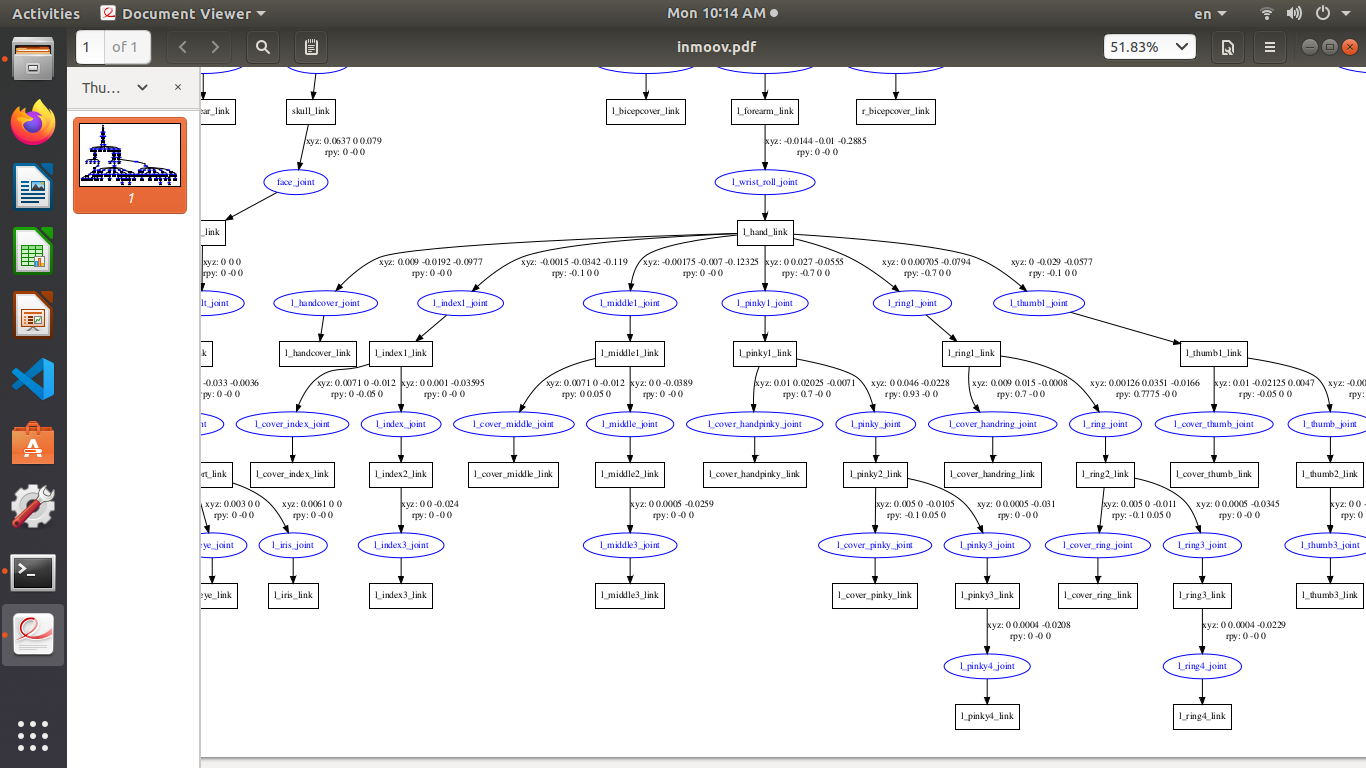


Figure URDF Model Left Hand

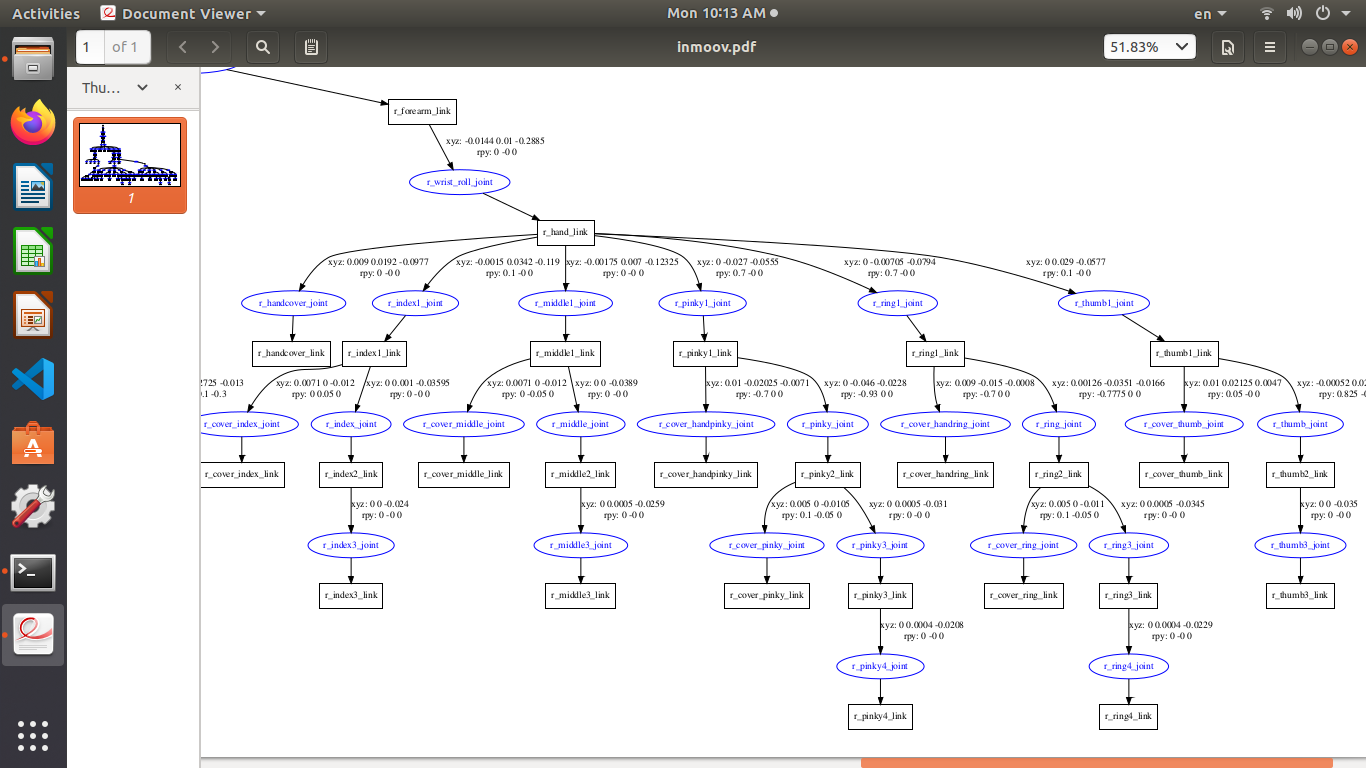


Figure URDF Model Right Hand

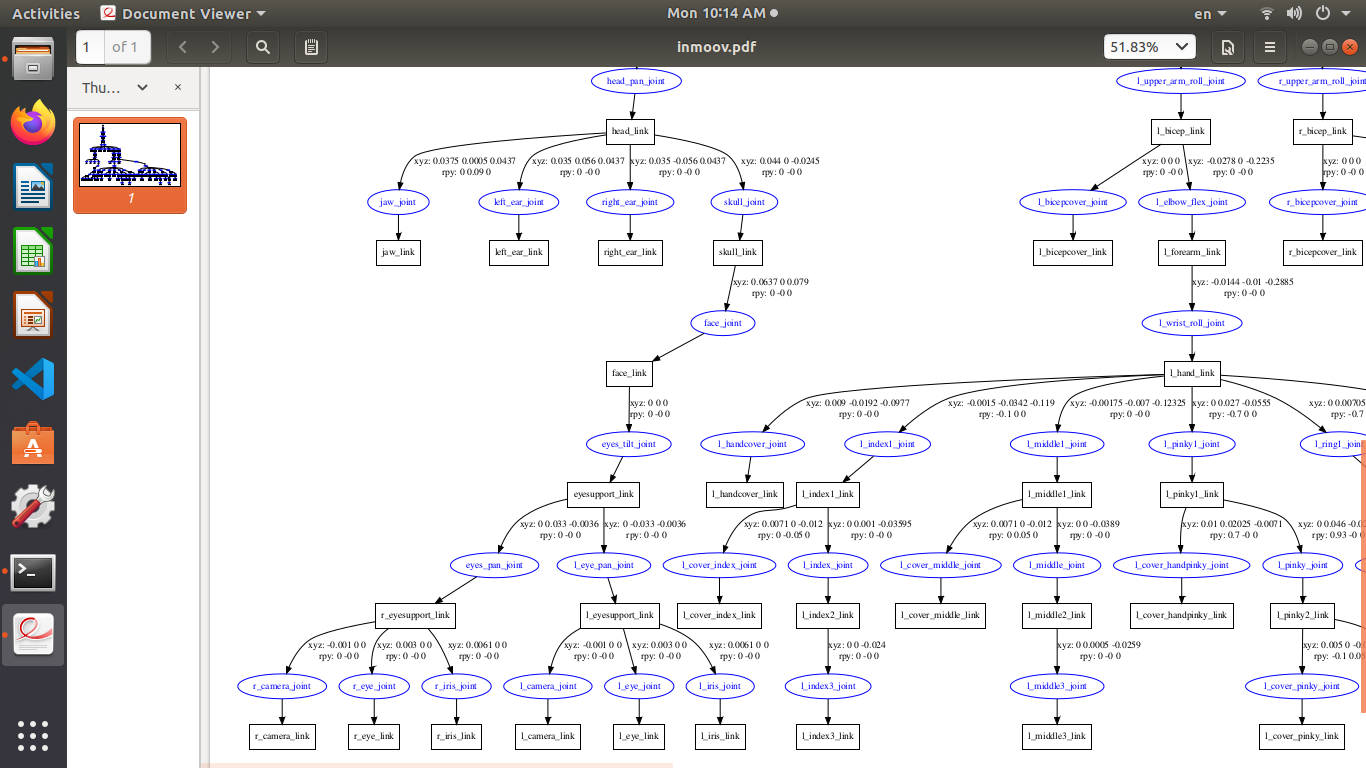


Figure URDF Model Head

Finally the last test to our robot by using Rviz Visualization to see the complete model connected to each other .

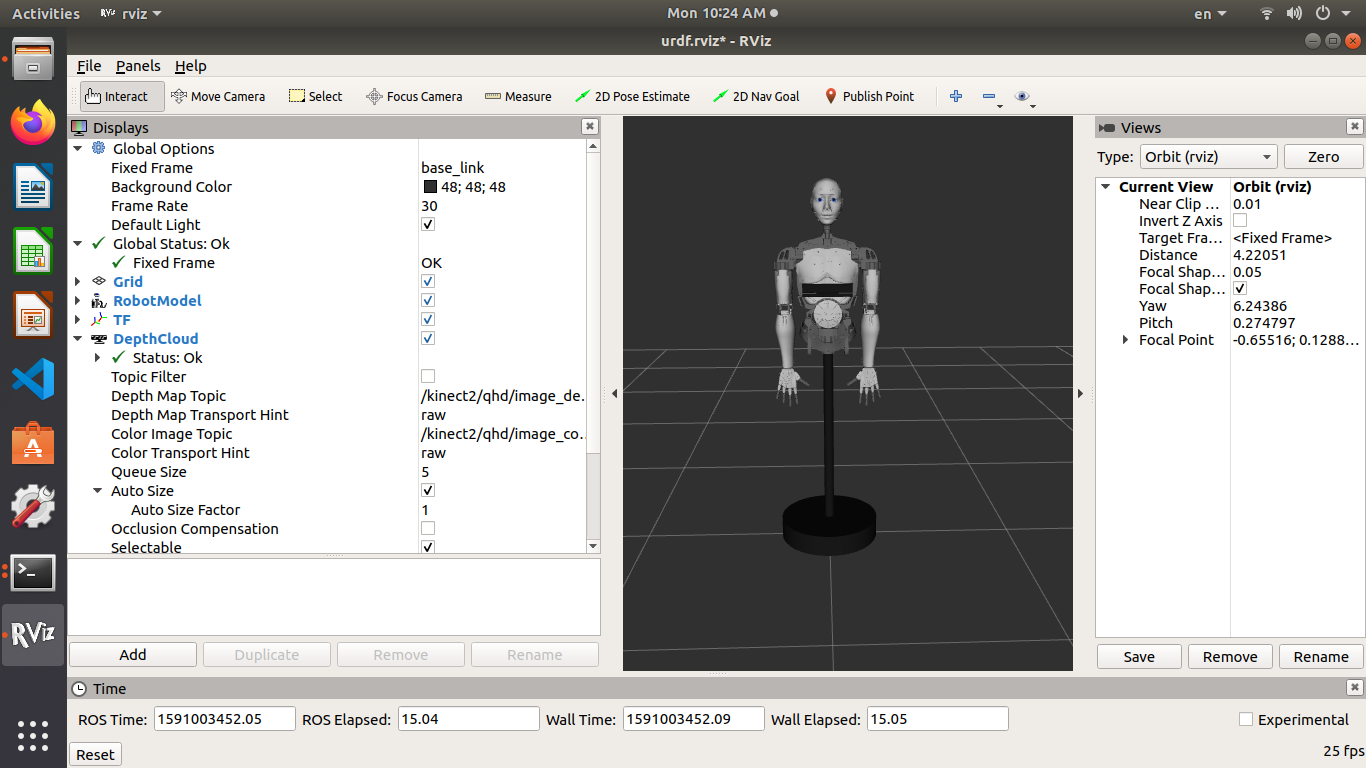


Figure URDF Model in Rviz 1

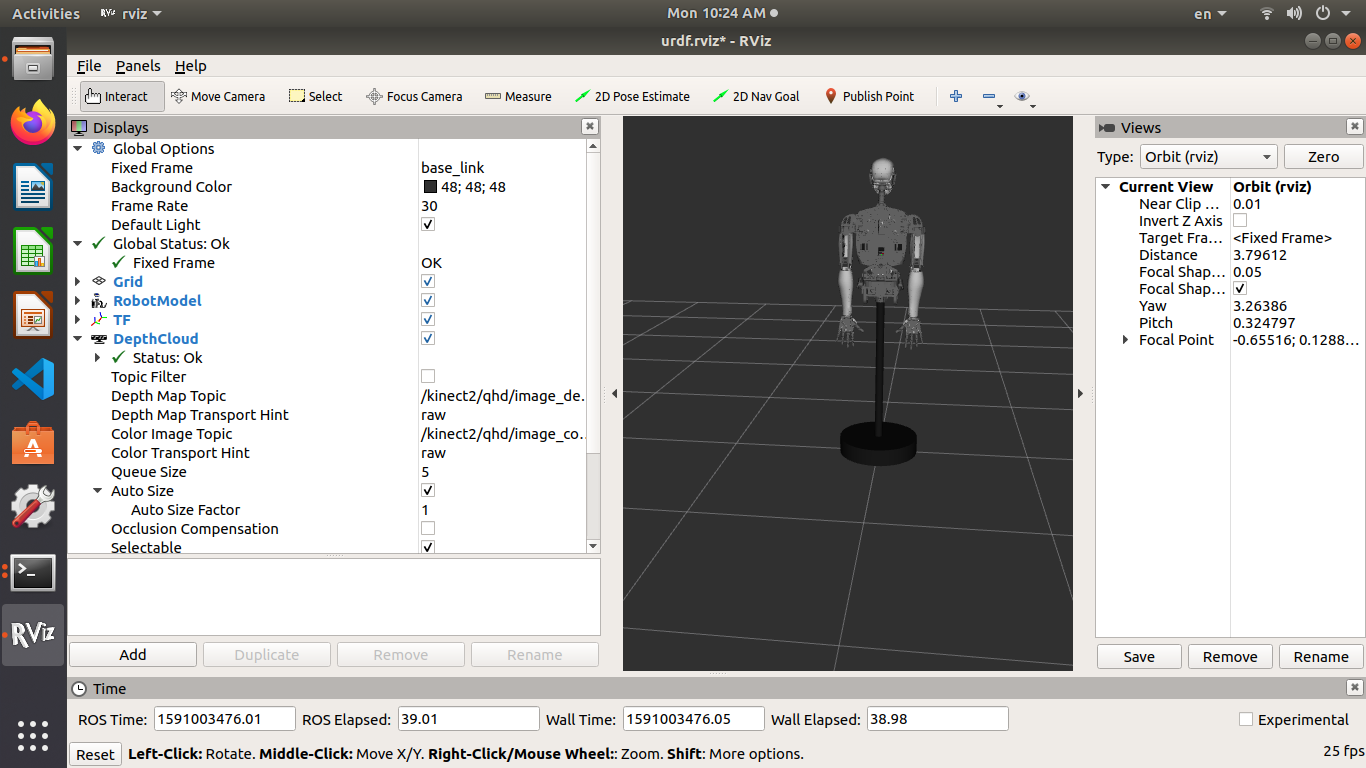


Figure URDF Model in Rviz 2

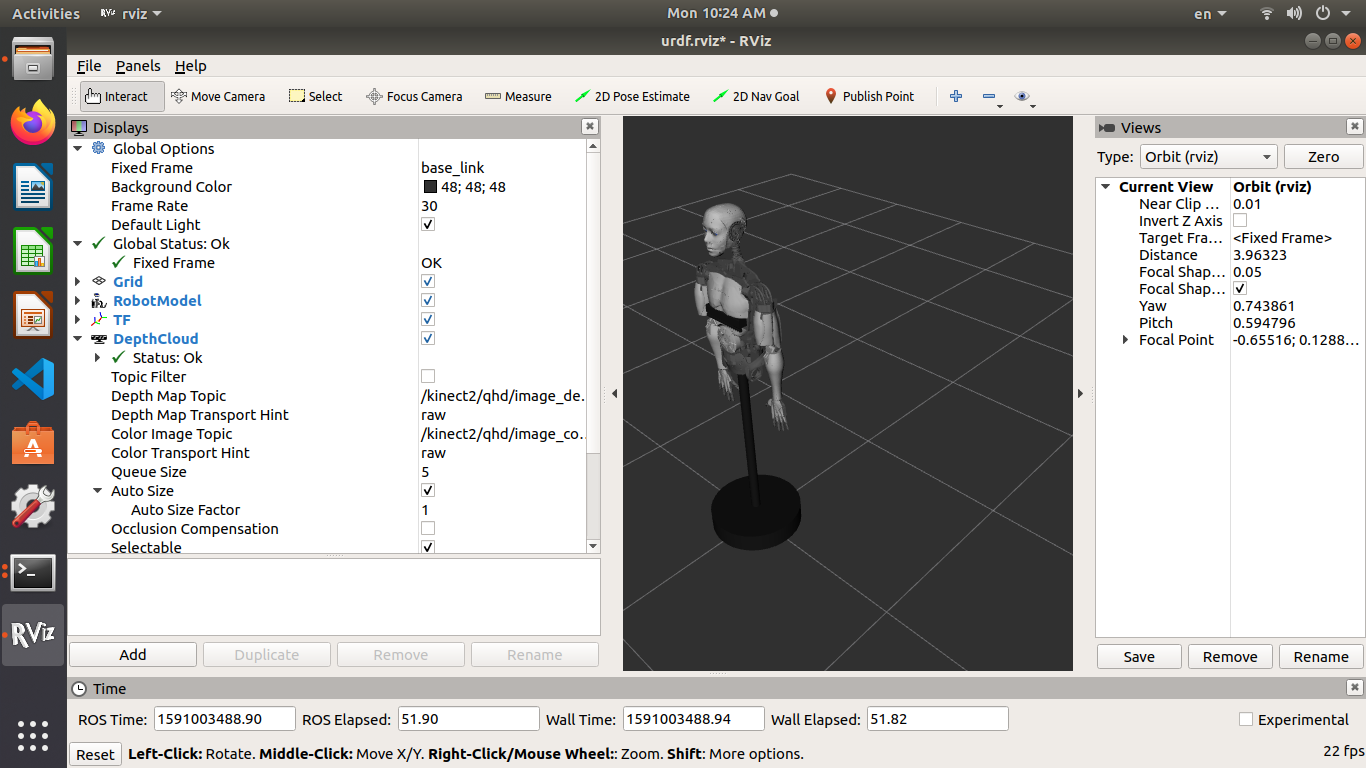


Figure URDF Model in Rviz 3

## Electrical design

### Circuit design

### PCB layouts

#### Hand PCB layout

#### Wrist PCB layout

#### Shoulder and elbow PCB layout

#### Head PCB layout

#### Voltage regulating PCB layout

#### Final PCB layout

### Wiring

### Circuit schematic

### Power calculations

## Sensor selection

## GPU selection

Most of the Autonomous and Artificial Intelligence (AI) systems are now using graphics processing unit (GPU) to make the system faster in processing and for our project which is autonomous we decided to choose the cheapest GPU made by Nvidia but it is still more expensive than the Raspberri Pi so we decided to make a comparison between Raspberry pi and Nvidia Jetson Nano and we conclude that because of the heating of raspberry pi during work and there is a heat sink in jetson nano and also its much more faster due to the GPU and RAM we chose the Nvidia Jetson Nano Development Board.

Comparison between Nvidia Jetson Nano and Raspberry pi

|  |  |  |
| --- | --- | --- |
| Point of comparison | Nvidia Jetson Nano Dev Board | Raspberry pi 3 |
| CPU | 1.43 GHz 64-bit quad-core ARM Cortex –A57 | 1.4 GHz 64-bit quad-core ARM Cortex –A53 |
| GPU | 128-Core Nvidia Maxwell | Broadcom Video Core IV |
| GPIO Header | 40-pin | 40-pin |
| RAM | 4GB LPDDR4 | 512 MB LPDDR2 SDRAM |

NVIDIA Jetson Nano enables the development of millions of new small, low-power AI systems. It opens new worlds of embedded IoT applications, including entry-level Network Video Recorders (NVRs), home robots, and intelligent gateways with full analytics capabilities.

NVIDIA Jetson uses AI to power the future of robotics, intelligent video analytics, The World's Smallest AI Supercomputer for Embedded and Edge Systems.



A new dimension in AI at just 70 x 45 mm, the Jetson Nano module is the smallest Jetson device. This production-ready System on Module (SOM) delivers big when it comes to deploying AI to devices at the edge across multiple industries—from smart cities to robotics.

Big computing performance, the Jetson Nano delivers 472 GFLOPs for running modern AI algorithms fast. It runs multiple neural networks in parallel and processes several high-resolution sensors simultaneously, making it ideal for applications like entry-level Network Video Recorders (NVRs), home robots, and intelligent gateways with full analytics capabilities.

Small power demands as Jetson Nano frees you to innovate at the edge. Experience powerful and efficient AI, computer vision, and high-performance computing at just 5 to 10 watts.

# Manual arm control

## Introduction

## Exoskeleton

### Arm control

### Fingers control

## Wireless joystick

This is a way of improving the manual control of the robot by using a wireless joystick which is dualshock 4 which is the playstation 4 joystick as its used with a Bluetooth to connect it from a 10 m distance from the robot we can control any motor in the robot so we can control base motion or full hand motion , the hand movement joints with the analog joysticks in the dualshock controller to control the range of the movement also the fingers to open and close them to hold anything we want and can put it anywhere else,

The first part was connecting the joysick with Bluetooth to the GPU or the microcontroller used , then the second part was connecting the servos to the board using the PCA9685 board which is servo driver can control up to 16 PWM pin using i2c communication .

The code used for programming is python and there are two main libraries used one for the joystick to define how to use the buttons and the analog , also the range of outputs of them as the analog ranged from -1 to 1 and we mapped the output in coding to match each servo limit in the arm there are many libraries to control the joystick but we chose the approxeng.input there is also pyps4 , the other main library in our code is the ServoKit library from Adafruit which is responsible for connecting the PCA9685 to the GPU by i2c bus .

## Kinect control

## Head control using VR

Many manufacturing jobs require a physical presence to operate machinery. But what if such jobs could be done remotely? So our system presented a virtual-reality (VR) system that lets you teleoperate a robot using a VR heasset which can be Google cardboard headset or Oculus Rift headset but in our project we are going to try with the Google Cardboard VR.

The system embeds the user in a VR control room with multiple sensor displays, making it feel like they are inside the robot's head. By using gestures, users can match their movements to the robot's to complete various tasks.

"A system like this could eventually help humans supervise robots from a distance". "By teleoperating robots from home, blue-collar workers would be able to tele-commute and benefit from the IT revolution just as white-collars workers do now."

The researchers even imagine that such a system could help employ increasing numbers of jobless video-gamers by "gameifying" manufacturing positions.

**How it works**

There have traditionally been two main approaches to using VR for teleoperation.

In a "**direct**" model, the user's vision is directly coupled to the robot's state. With these systems, a delayed signal could lead to nausea and headaches, and the user's viewpoint is limited to one perspective.

In the "**cyber-physical**" model, the user is separate from the robot. The user interacts with a virtual copy of the robot and the environment. This requires much more data, and specialized spaces.

In case of using the Oculus Rift headset we can use Oculus' controllers, users can interact with controls that appear in the virtual space to open and close the hand grippers to pick up, move, and retrieve items. A user can plan movements based on the distance between the arm's location marker and their hand while looking at the live display of the arm.

But in case of using Google Cardboard headset we will use a Kinect or joystick or a sleeve with sensors to control the arms and motion of the robot.

The cyber-physical model is too hard to develop right now it needs a research team of graduated engineers to develop it so we will develop the other model which is the direct model

This task is divided into two parts :\_

The first part is sending the live stream of the webcam to an ip address which has to be on a local network to see it from the smartphone which will be inside the Google Cardboard.

The second part is sending back the gyro-sensors inside the smartphone which are the pitch , yaw and roll which are the orientation about x , y, z by sending these values back we can use them to move the servo motors which are responsible for moving the head so now by moving my own head which wearing the VR that contains the smartphone that will leads to change the raw values of gyro sensor to move the robot’s head.

So finally we can see what the robot sees using a VR also controlling the head movement for a better view for the human who control it

The VR headset technology can combine two photos into a single obvious and clear photo but in the smartphone it has to be two photos but we don’t have to use both cameras to send them we can only use only one webcam and duplicate it to be in the place of the two windows in the smartphone and then it will be seen as a single shot

The webcams we are using are Microsoft LifeCam HD-3000 they are HD cams which there resolution can be 1280\*720 but in this application we can control the resolution we use to send to the smartphone inside the VR headset but unfortunately as long as the resolution increase the stream become more slower in broadcasting so the best resolution for a live stream is 320 \* 240 which hasn’t any lag and as long as we increase we get a slower stream .

# Autonomous arm control

## Catching a ball

### Introduction

For centuries, humans have been fascinated by the idea of creating machines similar to existing life. Although robots have replaced the human worker at repetitive tasks in industrial scenarios for many years, they do not have yet reached the capability and autonomy to accomplish everyday human tasks. This becomes even clearer when considering what variety of dynamic activities humans are able to perform, e. g. when doing sports.

Catching a thrown ball with a hand is not easy – neither for humans nor for robots. It demands for a tight interplay of skills in mechanics, control, planning and visual sensing to reach the necessary precision in space and time. Because of this, ball catching has been used for almost 20 years now as a challenging benchmark system to develop and test robotics key technologies

In all the works the general setup is in principle the same: a stereo vision system tracks the ball and predicts the balls trajectory, then the point and time, where and in which orientation the robot should  
intercept the ball on its trajectory, is determined. Next, the robot configuration to reach the catch point is computed and finally a path is generated, which brings the robot from its start configuration to the desired catch configuration.

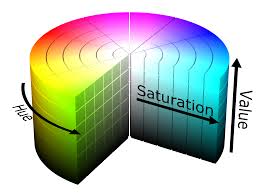
### Ball detection and tracking

#### Ball detection using opencv

The first part of this task is to detect the place of the ball in the image and tracking its place as it’s moving in the image, so we used the OpenCV library which is an open source library using a computer vision technique.

The first step is to change the RGB image into HSV image (Hue, Saturation, Value) which is better in detecting colors in an image.

The HSV color wheel sometimes appears as a cone or cylinder, but always with these three components:



HUE

Hue is the color portion of the model, expressed as a number from 0 to 360 degrees:

**Red** falls between 0 and 60 degrees.

**Yellow** falls between 61 and 120 degrees.

**Green** falls between 121 and 180 degrees.

**Cyan** falls between 181 and 240 degrees.

**Blue** falls between 241 and 300 degrees.

**Magenta** falls between 301 and 360 degrees.

SATURATION

Saturation describes the amount of gray in a particular color, from 0 to 100 percent. Reducing this component toward zero introduces more gray and produces a faded effect. Sometimes, saturation appears as a range from 0 to 1, where 0 is gray, and 1 is a primary color.

VALUE (OR BRIGHTNESS)

Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0 to 100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.

So to detect a ball in an image we have to set a lower and upper boundaries for the color of our ball’s color.

Second part is to draw a contour on the detected parts of the same color we have just defined to track the ball in the next frames to detect it in a real time video and also to calculate the centroid of that contour to use it in calculations and predict the path of the ball.

Contour: Edge (Contour) Detection is an image processing technique for finding the boundaries of objects within images.

Contours are a curve joining all the continuous points along the boundary, having same color or intensity.

Algorithm for finding the contours is by using “cv.findContours()” applied on binary image, then process the contours (find its areas ,enclosing circle and centroid)

Third part is to solve a problem of the noises to prevent finding objects in the frame with the same colors we have just declared so we can find the area of all contours using cv.contourArea() and choose to track the biggest area only.

Fourth part is to draw the ball contour around our object which is the ball using :

cv.drawContour(image , contours , index , color , thickness)

index : the index no. of the contours found if index =-1 then draw all contours, and color is the color of the contour line .

Then the Last step is to get\_contour\_center : to use it to find the pos and velocity of the ball in next steps of predicting the path of the ball.

#### Calculating the depth of the ball

After detecting the ball and tracking it in the video we have to know the real depth of the ball from the camera in cm.

So we found that there is only two methods to find the depth : \_  
First method is to use a stereo vision technique using two or more web cams and also there are two ways to find depth by using two or more cams which are using the depth map and the triangulation technique.

The triangulation technique is based on some trigonometry and two well calibrated cameras.

First, the camera calibration. The cameras must have the same orientation which means that both are looking in a forward direction and neither one of them is tilted in the x or y direction. Also the distance between the two cameras must be known.

If the object is places in front of 2 cameras as shown in the figure below on the right, knowing the two angles θ1 and θ2 and the distance l between the two cameras, the distance between the object and the line between the two cameras d can be calculated by the following equation:

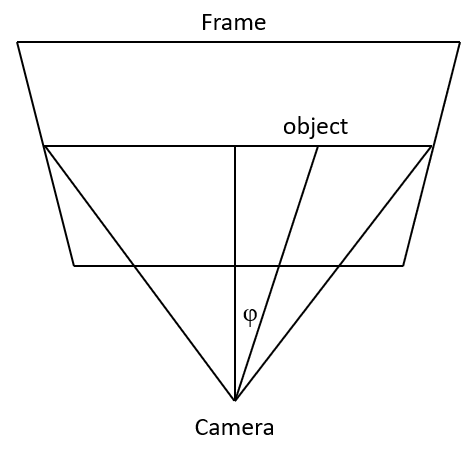
d = l / (1 / tan (θ1) + 1 / tan (θ2)

But the real problem is how to find those two angles of the two cameras. Assuming that the object appears in the frame captured by the camera as shown in the figure below the left one, the angle φ is the angle that object makes with the line drawn between the camera and the center of the frame and it is the complement of the angle θ we want to calculate.

So knowing the resolution of the camera (m x n) and the pixel representing the center of the object in the captured frame (a pixels from the center of the frame) and also the angle of the camera in the x direction (ω) which is specific to each camera, the angle θ can be calculated using the following equation:

tan(φ) = a x tan(ω) / (m/2)

θ = 90 – φ



The triangulation method has one limitation which is that the accuracy of the depth is proportional with the distance between the two cameras so in order to get a bigger range for the depth, the distance between the two cameras must get bigger which was a problem for us because of the design of the head the distance between the two cameras placed as the eyes of the robot did not allow us to widen the distance between the two cameras. As a result, the triangulation method gave an accuracy of about 60 cm and that was not suitable for our application because we needed at least 250 cm. So we used another method for the depth calculation.

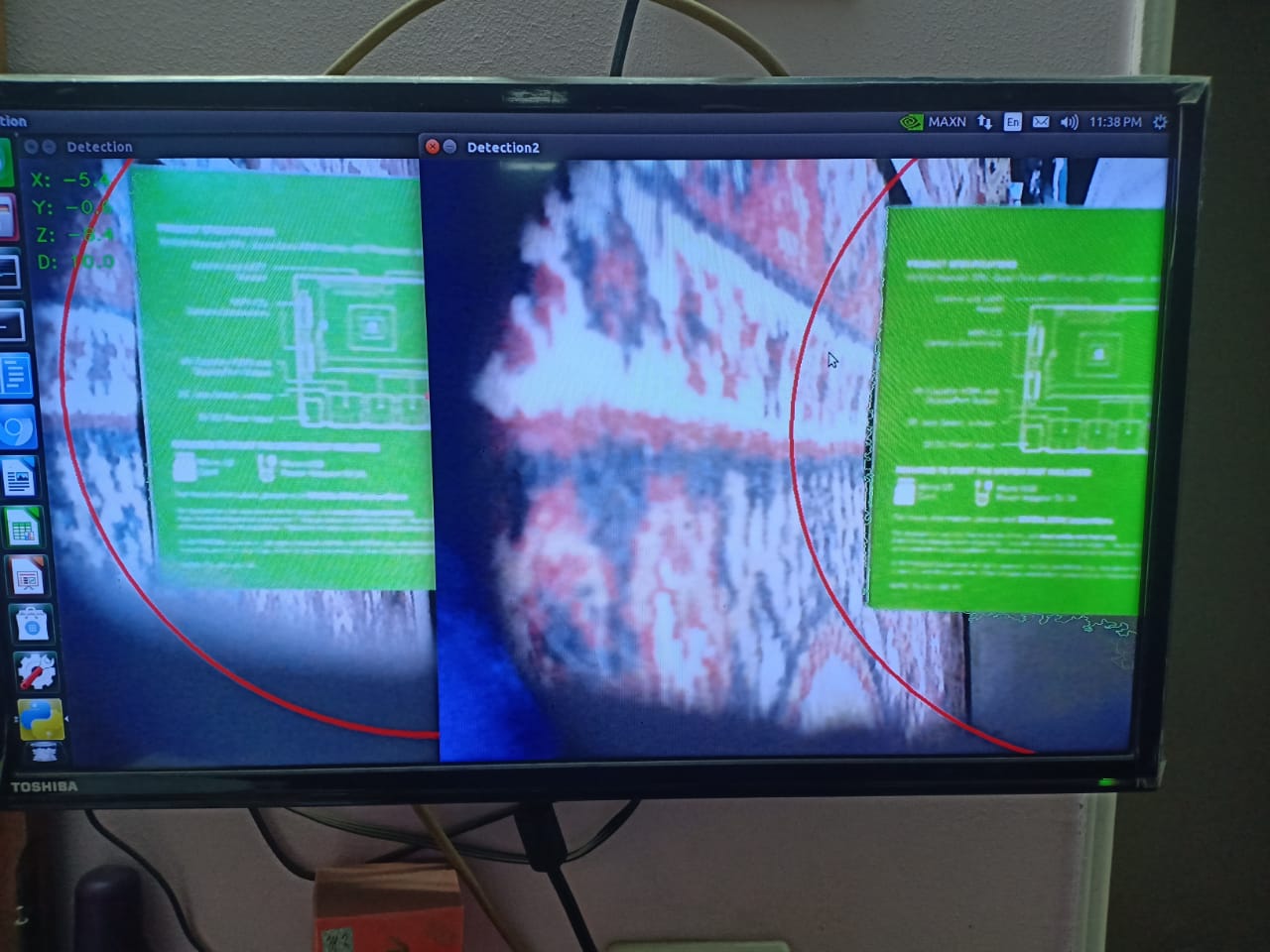


Figure Depth using traingulate 10 inch



Figure Depth using triangulate 10 inch

Figure Detect Depth in 13 inch (D)



Figure Detect Box in 13inch depth

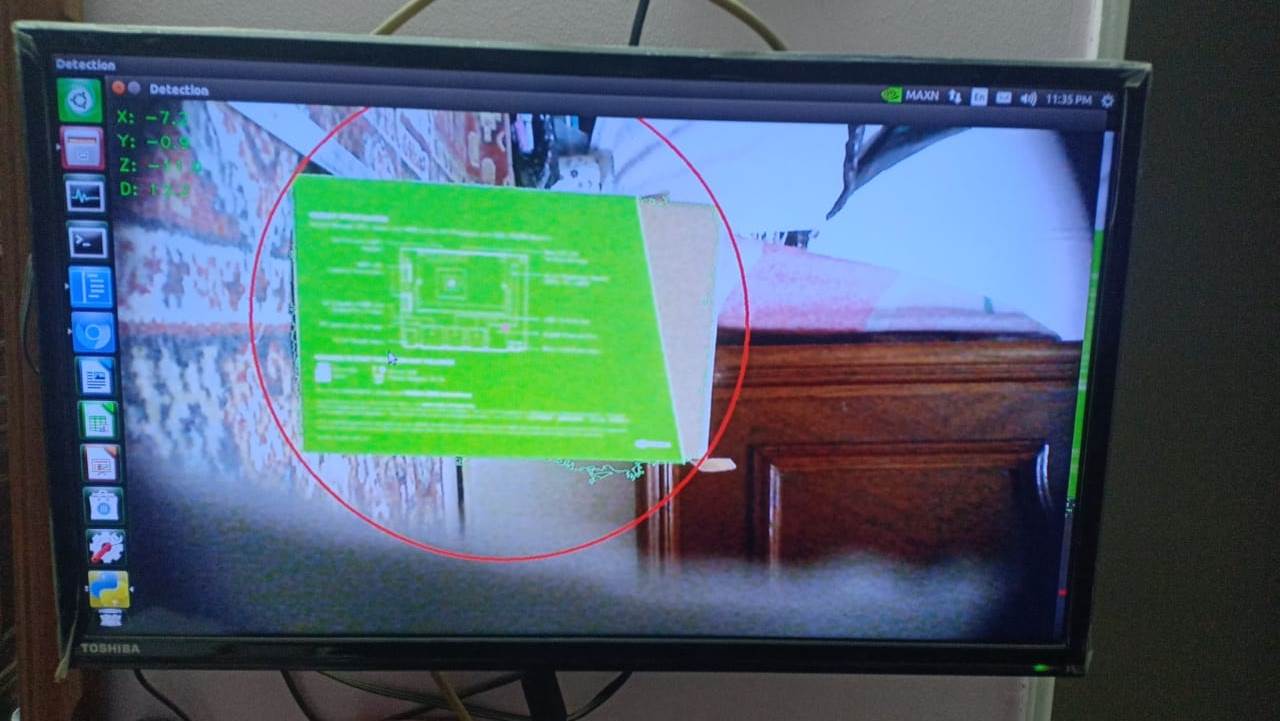
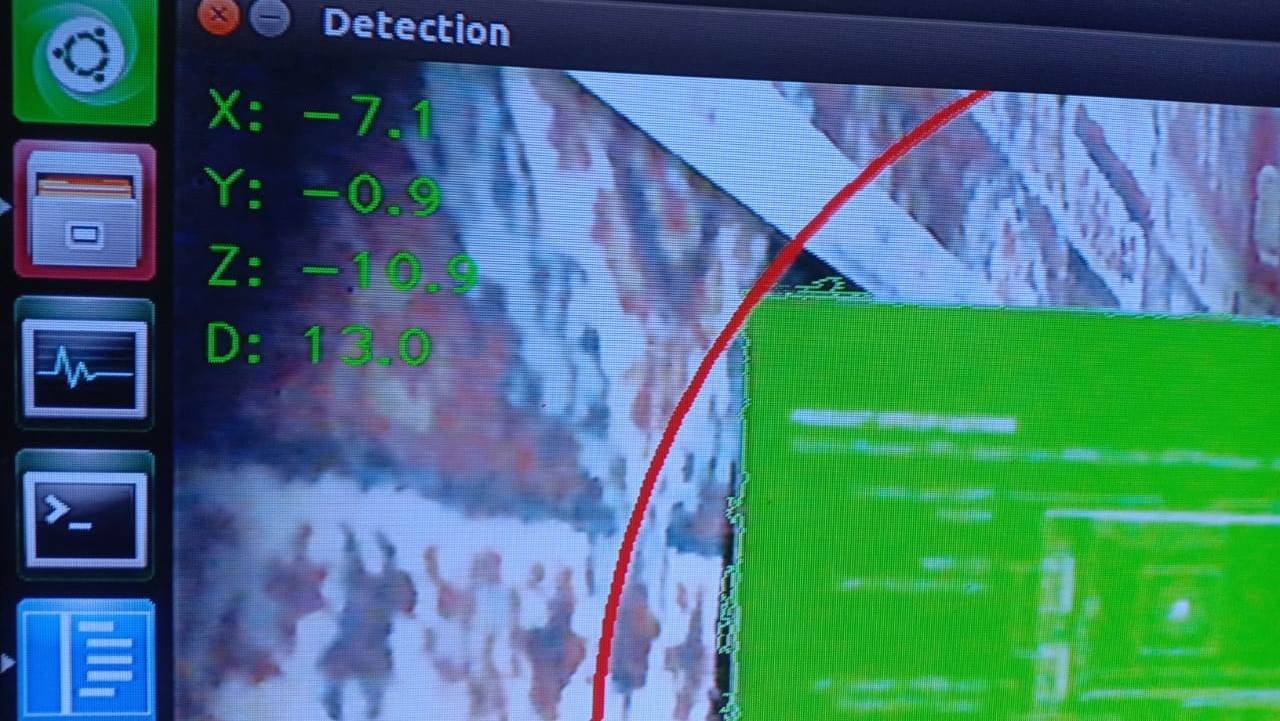


Figure Depth of the Box is D in inch



The second method is using a stereo camera which has an IR emitters and receivers to detect the depth of an object from the camera. We used both Microsoft Xbox 360 Kinect Sensor and Intel RealSense d435 but the second was better because our application is a real time application and we needed a camera with at least 60 fps and the maximum fps of the Kinect was 30. Both work with the same concept using the IR emitters and receivers and stereo vision we were able to calculate the distance of the center of the ball to camera given the pixel representing the center from the detection algorithm.

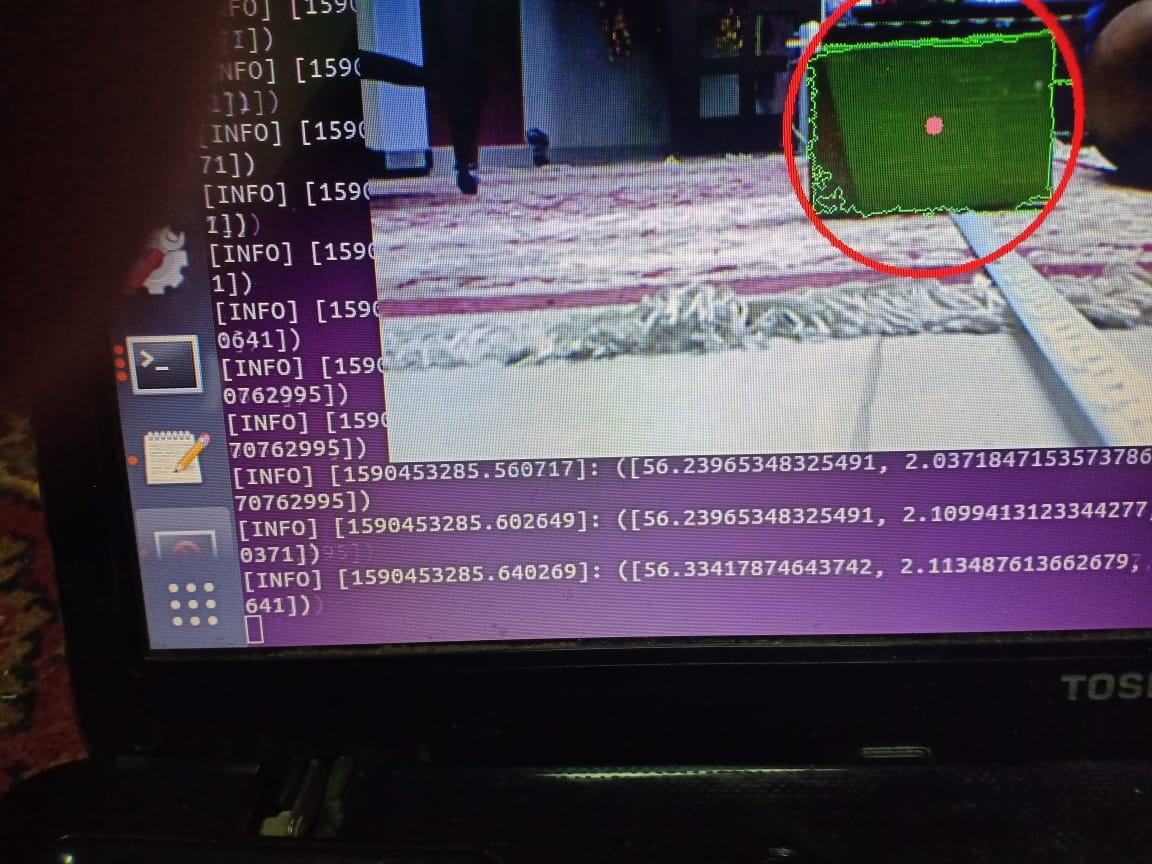


Figure Depth of box in 56 cm



Figure Depth of box using kinect

Figure Depth of box 240cm

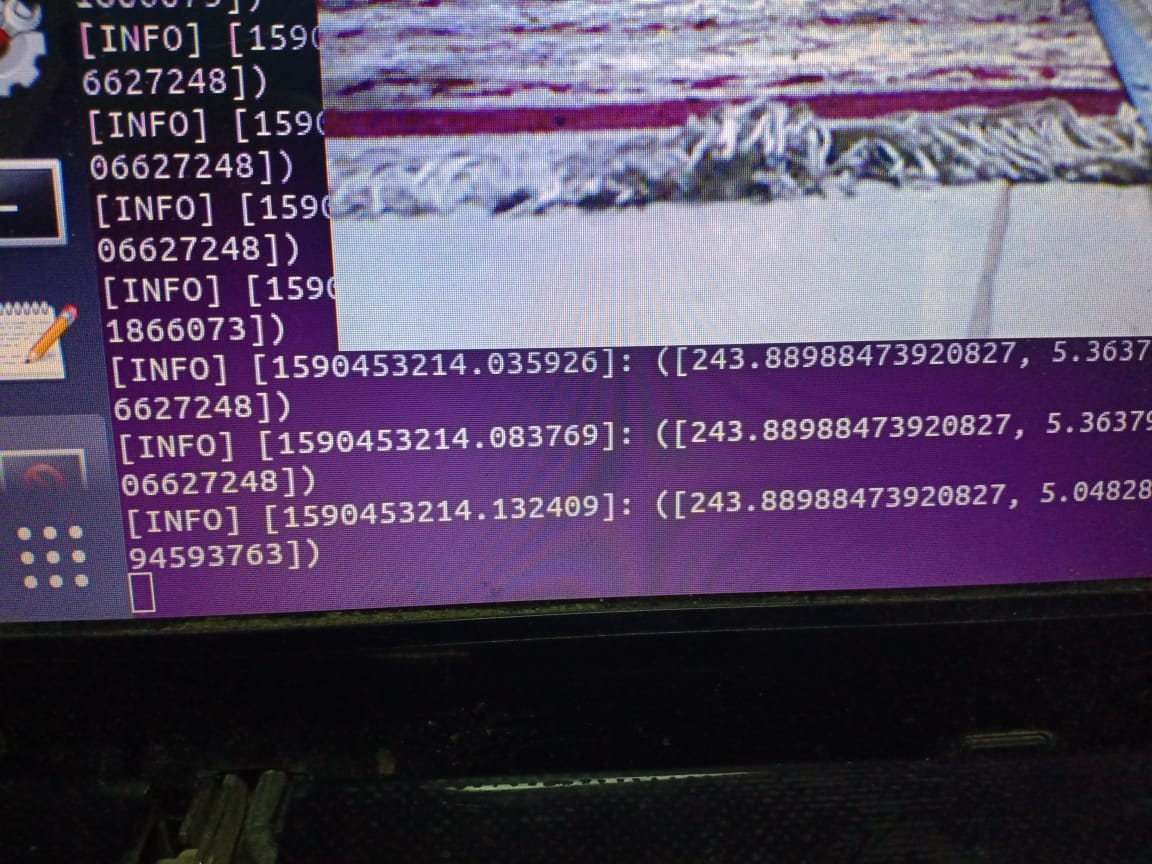
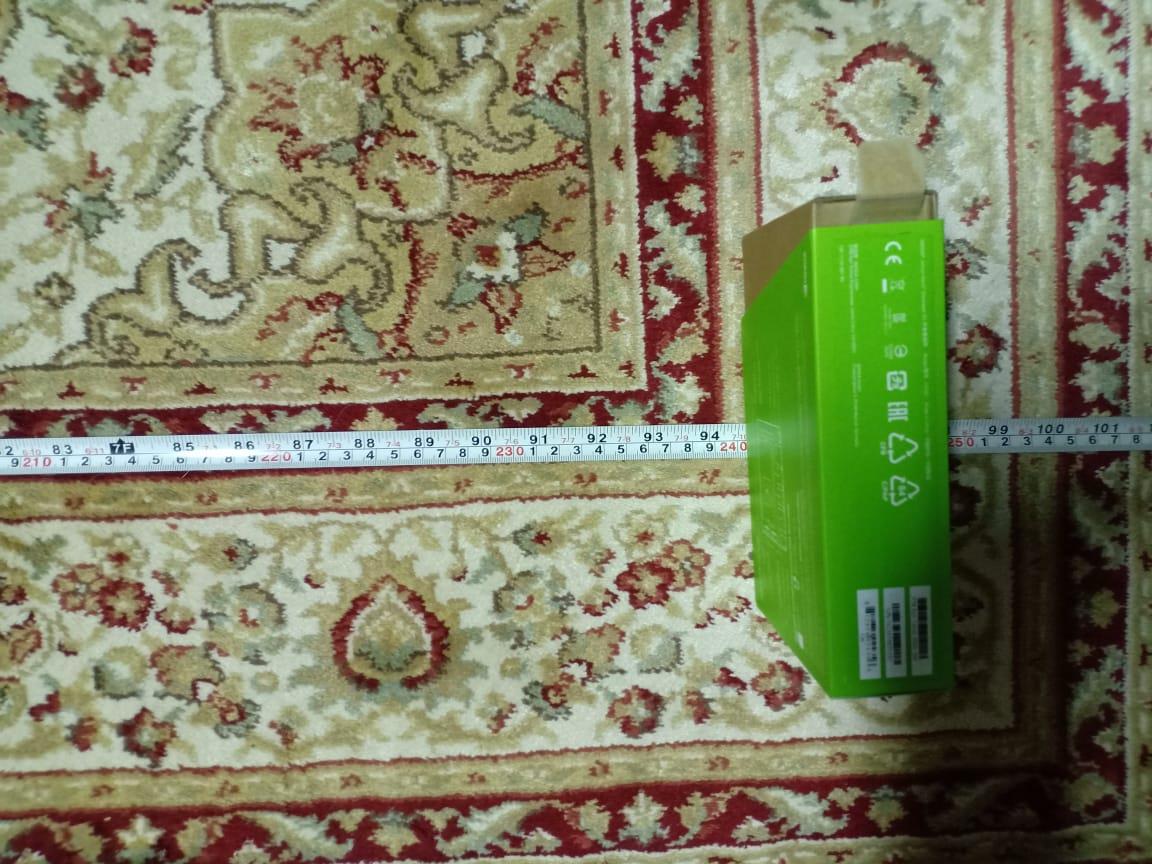


Figure Detected depth for box is 243.89 cm

### Trajectory estimation

#### Using Euler method

Euler's method approximates an exact solution to a differential equation as an iterative algebraic equation. Look at it like this - given a differential equation like

**dx / dt = x2 −t**

We can loosen the definition of d = lim (Δ→0) Δ and instead revert back to using Δ with a nonzero limit and refer to it as the step size - by how much are we jumping in time between iterations.

**(dx / dt) ≈ Δx / Δt = xi+1 − xi / Δt**

And for reference

**d2x / dt2 ≈( xi+1 − 2xi + xi−1 ) / Δt2**

Using the first, we can approximate the solution to the differential equation

**(xi+1 − xi / Δt) = x2i − ti ⟹ xi+1 = xi + Δt (x2i − ti)**

Notice that to begin this iterative process, we first need to know the initial condition to calculate the next position. Likewise, using the second derivative approximation requires two initial conditions.

This is extremely useful when plotting solutions/gathering data to differential equations that are difficult to solve, as it approximates the solution one step at a time.

In Our application we have our centroid of our object in each frame so by using the distance and velocity between each two frames we can use the differential equations to predict the trajectory path.



Figure Test Path Estimation Juggling 1 Figure Test Path Estimation Juggling 2



Figure Test Path Estimation Juggling 3 Figure Test Path Estimation Juggling 4



Figure Test Path Estimation Juggling 5 Figure Test Path Estimation Catching

### Inverse kinematics

#### Moveit

Moveit is a Motion Planning Framework used by ROS , we implement the URDF ( Unified Robot Description Format ) model inside it so we can use as visualization or simulation for the robot and it has many applications but we just used it in Inverse Kinematics .

First of all we have to build our URDF model and we have already about this part in this thesis.

Second We use the Moveit ! Setup assistant to allow solving inverse kinematics of the robot, after implementing the URDF model we can visualize in in moveit assistant to make sure that the robot is connected to each other correctly , then we Generate Self-Collision Matrix to prevent joints from colliding with each other ,then we have to add the planning group or the joints that we want to solve them together, also we have to clarify the end effector link which we will give the coordinates related to it , also we can make a rest or Pose Position to the group we have just made so when we start connecting the robot we see it in this position , Finally we implement the IK solver we want to use there are two main types the analytical and the numerical ,

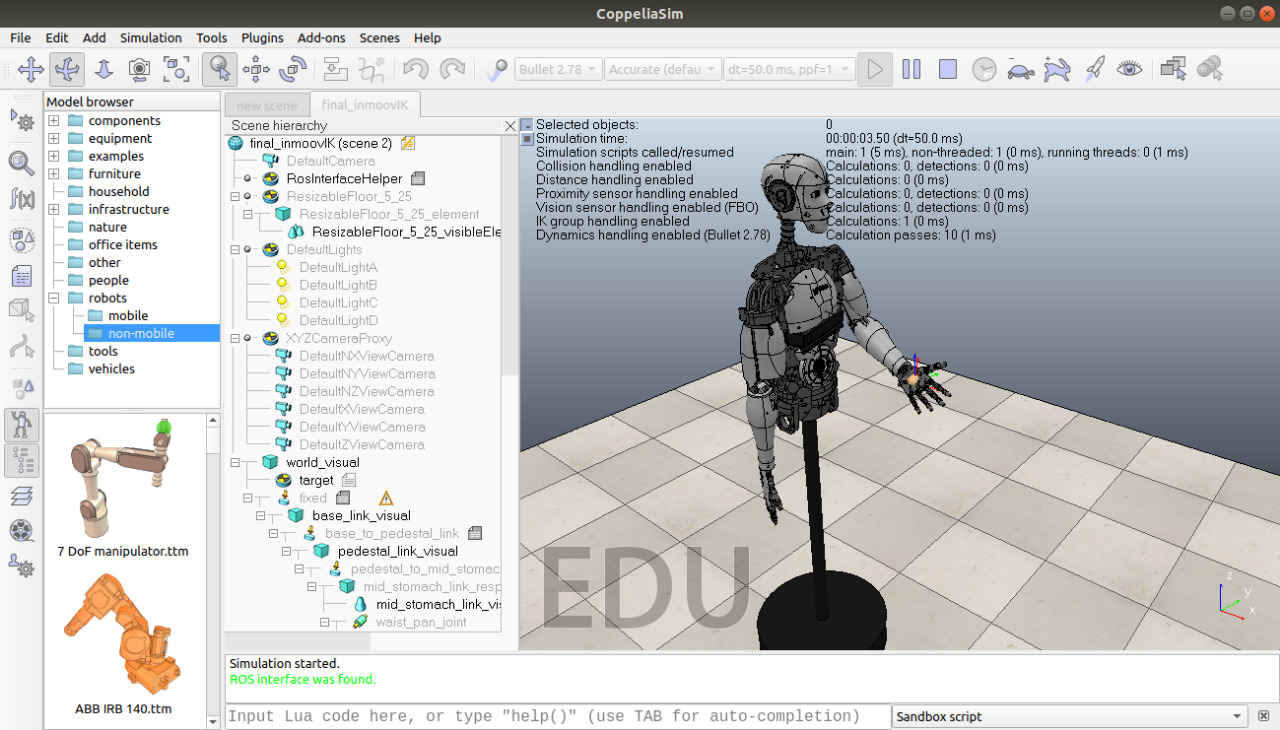
The analytical is mush better than the numerical and there is only one which is IK Fast which is faster and more accurate , but the numerical is much more easier to use for the robots of maximum of 3 DOF and there are many types of the numerical solvers eg. TRAC-IK , KDL Kinematics Plugin .

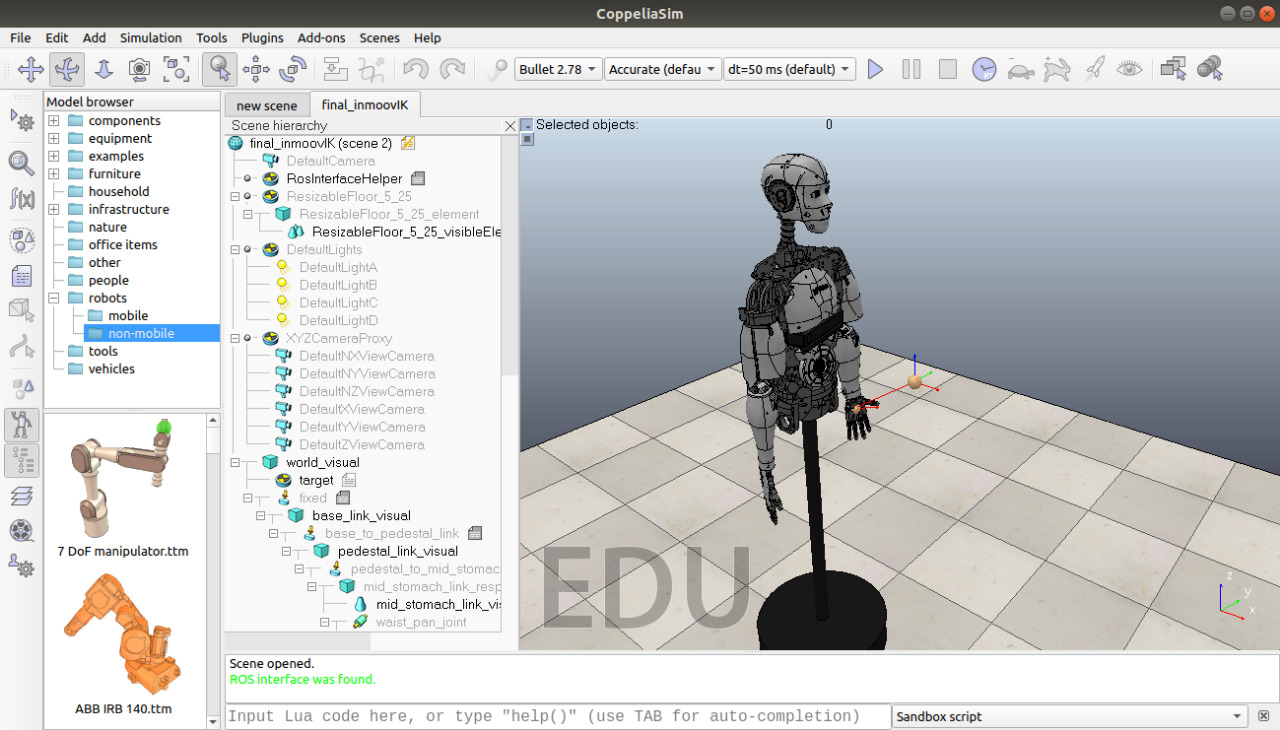
After finishing the Moveit assistant setup we have to send the position of the end effector to moveit to solve the joints we have two options now :  
The first one in which we can use Move Group Python or C++ Interface , which is interface between ROS (Moveit) and python code that we will write which is very simple in our situation there is already some functions like Pose or PoseStamped where we can put the location of end effector we want and there are some nodes which will be running after finishing the move it assistant setup using the demo.launch file which are joint\_State\_Publisher , Robot\_state\_publisher , Move\_group . then we can use the functions to set our pose then send it to the robot to solve the IK and return the Joint Variable .  
The Second option is using a Server called Compute\_IK which will also run with the demo.launch file of our model so we can use this service , server and client . so the client will send the request of the end effector then the server will respond with the joint variables .

#### Vrep

Vrep's inverse kinematics calculation module is very powerful and flexible. It allows handling virtually any type of mechanism in inverse kinematics mode (IK mode) or forward kinematics mode (FK mode).

By inserting the URDF file in vrep simulator and then adding two dummies one is attached to the last object in the considered kinematic chain which is the end-effector (tip) and the other is in the workspace (target). The target is always the dummy that represents the position and orientation the tip should adopt during simulation.  The target dummy should be linked to a tip dummy and the link should be an IK, tip-target link type. By running the IK calculation module the corresponding joint angles of inmoov arm will be computed.





#### analytical method

### Connecting Jetson Nano to the servos

#### Using Servo Driver PCA9685 from Adafruit

PCA 9685 is an i2c-controlled PWM driver with a built in clock. That means that, unlike the TLC5940 family, we do not need to continuously send it signal tying up our microcontroller, its completely free running!

It is 5V compliant, which means we can control it from a 3.3V microcontroller and still safely drive up to 6V outputs (this is good for when we want to control white or blue LEDs with 3.4+ forward voltages)

It’s 6 address select pins so you can wire up to 62 of these on a single i2c bus, a total of 992 outputs - that's a lot of servos or LEDs

Adjustable frequency PWM up to about 1.6 KHz

12-bit resolution for each output - for servos, that means about 4us resolution at 60Hz update rate

Configurable push-pull or open-drain output

Output enable pin to quickly disable all the outputs



Figure PCA 9685

This board is used to controlled multiple servos on a single i2c bus and it’s easily used with the jetson nano board because we can’t connect the servos directly to the pins as there isn’t any pwm pins so the only way to connect servos to the board is through the two pins which are serial Clock & serial Data pins which are used in i2c communication and on the board there are two buses in the board pins (27,28) and (3,5) which are (SDA,SCL) consequently so we have two i2c buses and we connect the pca board which are made from Adafruit company to only one bus and its easily used in Ubuntu or the jetson nano os which contains a library made by adafruit to control servos using this board , Its name is ServoKit library

We can connect up to 16 motor to the board so we can control all of them with PWM but they will need an external power supply to connect it to the board to supply the motors but the board itself is powered by the vcc or v++ pins from the jetson nano directly as it needs low current to be powered but for the motors each one will need a maximum of 2 Amperes for each motor so its preffered to use an external power supply at using a lot of motors.

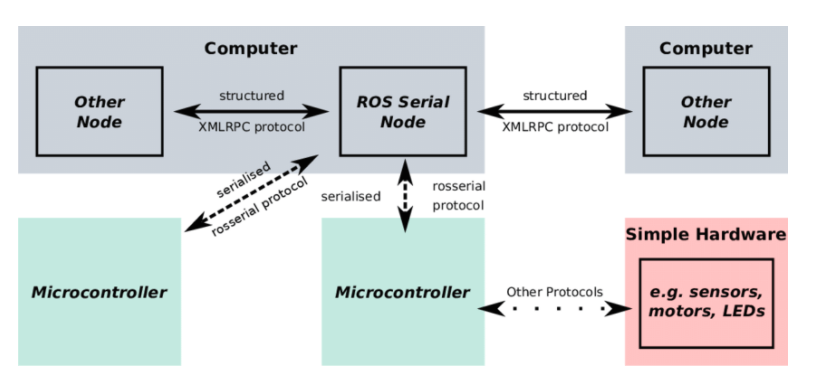
#### Using Arduino

##### Ros and Arduino bridge communication

For Ros:

Rosserial package is a protocol to send data through a serial interface. In a client-server rosserial implementation, a rosserial-server is a computer running ROS and a rosserial-client is the microprocessor that receives sensors’ data and transports it to the server in the form of ROS messages. rosserial-server in this implementation is a publishing node while rosserial-client is a subscriber node, although this can sometimes be the other way round.

Rosserial-client package is available for several microprocessor types including Arduino, STM32, embeddedlinux and [others](http://wiki.ros.org/rosserial#Client_Libraries). While rosserial-server package is available in Python or C++ versions.



For Arduino:

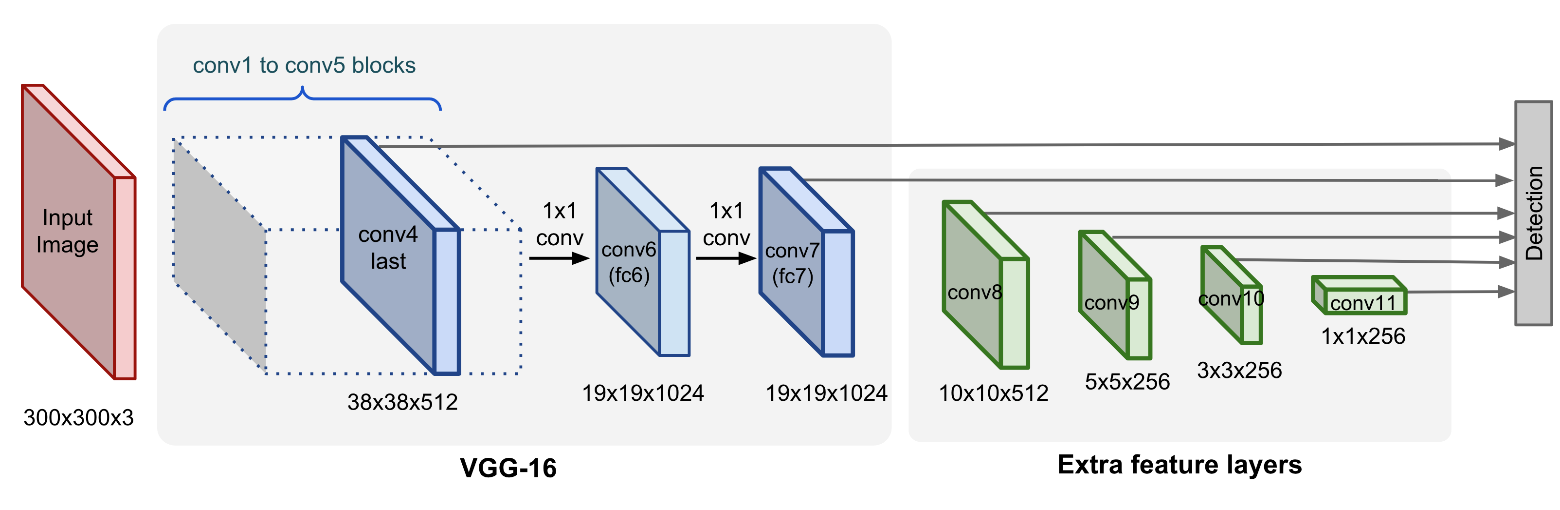
 To be able to communicate with ROS, ros\_lib must be installed. So the servo motors could be controlled through an Arduino code which will also set up a node that subscribes to a topic that streams each position of the servo motors.

## Object detection using neural networks

### Using SSD (Single Shot Multibox Detection)

The **Single Shot Detector (SSD**; [Liu et al, 2016](https://arxiv.org/abs/1512.02325)) is one of the first attempts at using convolutional neural network’s pyramidal feature hierarchy for efficient detection of objects of various sizes.

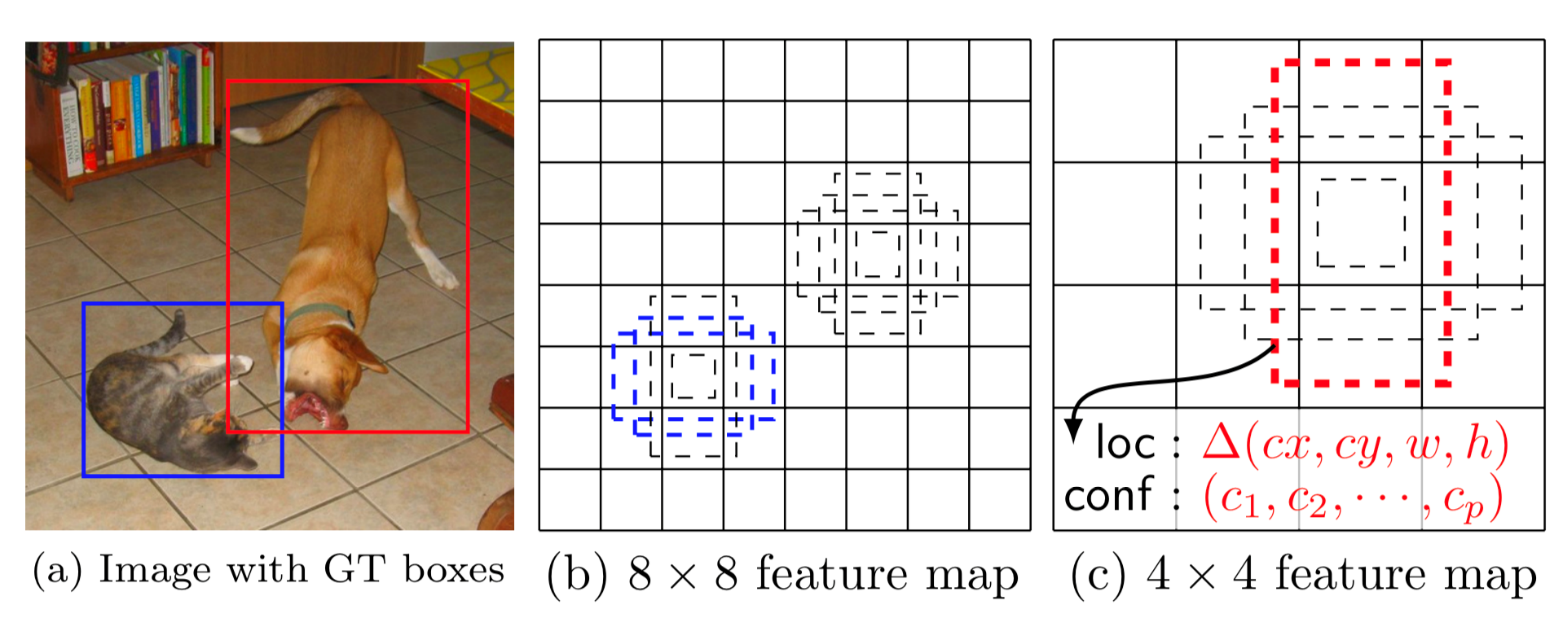
SSD uses the [VGG-16](https://arxiv.org/abs/1409.1556) model pre-trained on ImageNet as its base model for extracting useful image features. On top of VGG16, SSD adds several conv features layers of decreasing sizes. They can be seen as a pyramid representationof images at different scales. Intuitively large fine-grained feature maps at earlier levels are good at capturing small objects and small coarse-grained feature maps can detect large objects well. In SSD, the detection happens in every pyramidal layer, targeting at objects of various sizes.



SSD does not split the image into grids of arbitrary size but predicts offset of predefined anchor boxes (this is called “default boxes” in the paper) for every location of the feature map. Each box has a fixed size and position relative to its corresponding cell. All the anchor boxes tile the whole feature map in a convolutional manner.

Feature maps at different levels have different receptive field sizes. The anchor boxes on different levels are rescaled so that one feature map is only responsible for objects at one particular scale. For example, in figure shown the dog can only be detected in the 4x4 feature map (higher level) while the cat is just captured by the 8x8 feature map (lower level).

SSD also can detect the desired object at any orientation and at any scale so it eliminates the orientation and scaling problem and it maps the pixels corresponding to the detected object to a vector of four floating numbers that represent a rectangle that contains the object so it predicts the precise location if the object



SSD is already trained on detecting a huge variety of objects (classes) and can be easily trained to detect even more objects. We did not use the SSD in catching the ball as it was easy detecting the ball using only opencv but SSD was tested and it will be very useful in the future development of the InMoov robot when performing other autonomous tasks that will require the handling of different and more complicated objects than the ball.

### Testing the detection of a person and a dog

We tested object detection using DetectNet Neural Network on Nvidia Jetson Nano for detecting a dog and person.



Figure Detect Dog with DetectNet Neural Network



Figure Detect Person with 98.8 %



Figure Detect Person with 98.4 % using DetectNet Netural Network on Jetson Nano

## Speech Recognition

# References

* <https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html#ssd-single-shot-multibox-detector>