DE-41 (MTS) Usama, Fahad, Wajid

Assistive Feeding System

COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
2023

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-41 MTS
PROJECT REPORT

Assistive Feeding System

Submitted to the Department of Mechatronics Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Engineering
in
Mechatronics
Engineering
2023

Sponsoring DS:
Dr. Mohsin Islam Tiwana
Dr. Hamid Jabbar

Submitted By:
Usama Jahangir
M. Fahad Amir
Wajid Ali

ACKNOWLEDGEMENT

Firstly, we are extremely thankful to the Allah Almighty for enabling us to reach this stage of progress where we are today.

After that we would like to thank our parents for their appreciation, well wishes, prayers and strong support. Of course, we would not be able to make our progress without the help of our extremely supporting faculty members of NUST College of E&ME. We would especially like to thank Dr. Mohsin Tiwana and Dr. Hamid Jabbar for their full support in every issue we underwent.

We would also like to thank all those people who, though, not mentioned here but have helped us in the project directly or indirectly.

ABSTRACT

Old people and patients with limited or complete motor impairment had always face difficulty in their dailylife task in which most important and frequent is eating meals, such people need assistant like nurse to feed them. In this project we worked on an assistive feeding system based on 6 Degree of Freedom fixedbase open-source robotic arm with active assistance. The manipulator is 3D printed by fused deposition modelling technique using PLA+ material and a counterweight mechanism is introduced to reduce the motor torque requirements hence reducing the power consumption of the system. The system is sub-divided into 2 sub-systems which are vision system to get location of mouth of the person to feed and, scooping and delivery system which uses manipulator and control system to feed the food. The system is completely embedded on raspberry pi with wireless access for user and programmer. In the final system there are two main modes one is passive feeding and second is active feeding user can choose on runtime and change modes as per requirement.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Greek Letters

θ	Joint angle
τ	Torque
α	Twist angle
β	Link 2 angle

Acronyms

DOF Degrees Of Freedom
ADL Activity of Daily Life
DC Direct Current
AC Alternating Current
YOLO You Only Look Once
IOU Intersection over Union
RNN Recurrent Neural Network
DNN Deep Neural Network
VNC Virtual Network Computing
SSH Secure Shell
SSD Single Shot Detector
SLA Stereolithography
FDM Fuse Deposition Modelling
SLS Selective Laser Sintering
CNC Computerized Numerical Control
IC Integrated Circuit
SMPS Switching Mode Power Supply
USB Universal Serial Bus
SoC System-on-Chip
GPU Graphic Processing Unit
LAN Local Area Network
BLE Bluetooth Low Energy
PLA Polylacticacid
ML Machine Learning
OS Operating System
RAM Random Access Memory
GUI Graphical User Interface
CAD Computer Aided Design

Chapter 1: INTRODUCTION

A robotic manipulator arm is a programmable, multipurpose mechanical device used to move tools, components, or materials according to predetermined movements to carry out numerous operations. They are widely used in industries where automation is in play. Robotic manipulators can be classified as either fixed base manipulators or mobile base manipulators. Fixed base manipulators are on a fixed spot and then perform movements whereas mobile base manipulators can move with the help of wheels to some other location. The degrees of freedom of a robotic manipulator determines its number of independent movements in a plane and it is one of the most important parameters of a robotic manipulator.

Successful deployment of a robotic manipulator arm includes well designed structure (links), good path planning and motion planning, accurate sensors placement, a good controller, and robust motion algorithms. For active feeding, the coordinates will be determined on run-time and the motion of the joints will be determined with that. So, the controller should be capable of performing the required actuations after getting values from the sensors or camera (Computer Vision).

The objective of this paper is the development of a 6 Degrees of Freedom, Fixed base Robotic Manipulator for assistive feeding of patients. The controller used will be Raspberry Pi and NEMA 23, NEMA 14 and NEMA 17 stepper motors will be used for actuations. There is a predefined location of food, so the coordinates of food will be known, and the coordinates of user's mouth will be determined using computer vision. After that Inverse Kinematics will be used to find the joint angles. With help of motion algorithms, the Raspberry Pi will actuate motors to move the manipulator to pick up food and then move to the desired location.

Robotic Feeding systems can be used for commercial and personal uses; and with computer vision and customizable end effector, various types of food utensils can be used via the active feeding. The fact that they can work with the people suffering from contagious diseases such as COVID adds to their importance in medical industry and hospitals.

Chapter 2: LITERATURE REVIEW

2.1. Feeding

Activities of daily living (ADLs) such as eating, toileting, and clothing are critical for overall health and well-being. Nonetheless, without the support of a human caregiver, such chores can be difficult for many people with disabilities, particularly those with upper limb limitations. However, healthcare worker shortages and rising healthcare expenses generate an urgent need for technologies that make help more inexpensive and effective. Technology interventions can help bridge the gap between physical capabilitiesand required functional competence. To aid people with disabilities in performing ADLs on their own, numerous specialized assistive gadgets, including robots, have been developed. Each gadget often providesa specific type of help for people with specific limitations. Researchers have also used general-purpose mobile manipulators in several applications such as rescue, aid, and residential service. Robots with a mobile base and human-like arms (for example, the PR2 robot from Willow Garage and the Jaco arm witha mobile base from Fattal et al.) assist individuals in overcoming physical or perceptual constraints through teleoperation. Although mobile manipulators have the potential to deliver a wide range of supportive services, their complexity poses difficulties, including the danger of low usability. Meal preparation is anexample of an assistive chore, and it is an important ADL for staying healthy. People with upper-body andlimb limitations frequently struggle to feed themselves. Although there are a variety of commercially available specialist food-help robots (e.g., My Spoon, Bestic arm, and Mealtime companion), these robotsonly provide limited meal assistance. Notably, we refer to the type of assistance provided by these robotsas passive feeding aid, in which the robot delivers food to a specified point outside the users' mouths and the users consume the food using upper body and limb movement. This is owing in part to the robots' fixedbases (desk-mountable), low degrees of freedom (DOF) limbs, and restricted sensory capabilities. Instead,we employ a general-purpose mobile manipulator to provide active feeding assistance by autonomously delivering food within a user's mouth, leveraging the robot's superior physical and sensing skills.

2.1.1. Nursing

There are large number of old people and patients with disabilities who can’t eat independently and hence, needs assistance which is mostly provided by caregivers who arededicated for a particular person which ultimately is creating hurdle as large number of caregivers are needed, apart from that there is one more important hurdle which is concluded in a research paper as following:

“Most of the patients were striving to maintain independence and control, and they wantedto manage their food and meals themselves; however, they needed help. For some,becoming dependent on help from others was an inner struggle: ‘When you're used to doingeverything alone and suddenly find yourself in need of assistance to make a sandwich. Could you imagine? It's gloomy simply thinking about it.' Despite the patients' indicated desire for independence, no self-management help was provided by the care services. Theywere reliant on family caregivers, friends and neighbors for assistance in managing their daily lives, understanding information, and communicating with healthcare professionals,indicating a paternalistic approach to care.”

Figure 1. Example of nurse feeding a patient (Source: www.medicarelaboratories.com)

2.1.2. Assistive Feeding System

Assistive feeding devices are an alternative to nursing or human caregivers. These systems or gadgets aid in feeding people. We already employ partial assistive feeding tools like spoons, forks, and knives in our daily lives. However, autonomous systems are created to aid without a human requirement in the event of elderly individuals and sick who can't eat independently. Although several specialized meal-assistance robots are commercially available (e.g., My Spoon [1], Bestic arm [2], and Mealtime partner), but these robots provide limited meal assistance.

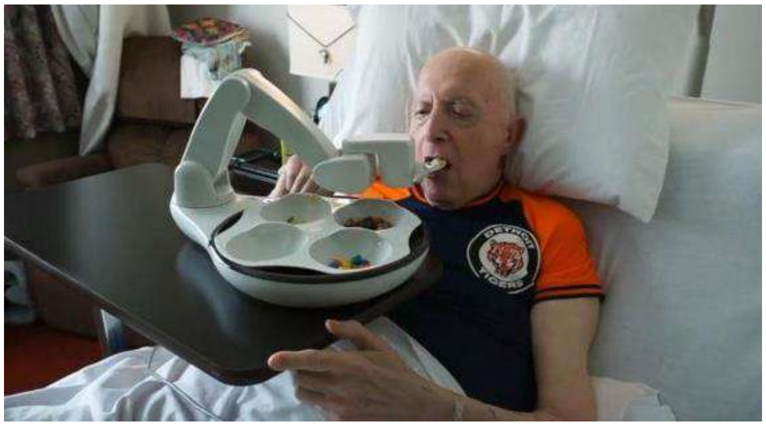


Figure 2. Example of robotic feeding system that is feeding a patient (www.sciencedirect.com)

2.1.2.1. Types based on motion method

2.1.2.1.1. Fixed base

Assistive robots with fixed bases are frequently positioned close to a person or a specific workspace. Early assistive robots were mounted to desktop computers by researchers to help with feeding, grooming, and hygiene. The professional vocational assistive robot (ProVAR) is a representative desktop manipulator placed in an office workspace [3]. Handy-1 is another adjustable table-mounted manipulator for ADLs such as eating, drinking, and washing applications. The mounted robots were designed to perform various ADLs using a general-purpose manipulator. However, the limited workspaces of the robots restrict the range of available activities. Alternatively, researchers have introduced various wheelchair-mounted robotic arms (WMRAs). For meal assistance, Maheu et al. showed that people with disabilities can feed themselves using a manually controlled JACO armmounted on a wheelchair [4]. Schroer et al. showed drinking assistance using a 7- DoF KUKA arm [5]. For object fetching, Kim et al. introduced the UCF-MANUSrobot, consisting of a wheelchair-mounted manipulator and interface [6].

2.1.2.1.2. Mobile base

A mobile base can expand a robot's workspace and the number of jobs it can undertake. Hawkins et al. discovered that movement of a mobile manipulator's base was required to assist with a shaving task since the PR2 could not otherwise reach the essential places [7]. Caretakers frequently need to reposition the fixed-base robot before or during the feeding activity. A fixed base limits the breadth of help duties. Robots with restricted mobility are limited to a small set of jobs and are unable to leave the human's close vicinity to assist elsewhere. For numerous assistive robotic operations, such as shaving, dressing, fetch-and-carry, and guiding duties, general purpose mobile manipulators have been developed recently. Our meal-assistance system has a mobile base that has the potential to enhance the quality of feeding assistance.

2.1.2.2. Types based on feeding method

2.1.2.2.1. Passive Feeding

Where the robot delivers food to a predefined location outside the users' mouth and users take the food by using their upper body and limb movement. This is due in part to the robots' (desk-mountable) fixed bases, low degree-of-freedom (DOF) arms, and limited sensing capabilities.

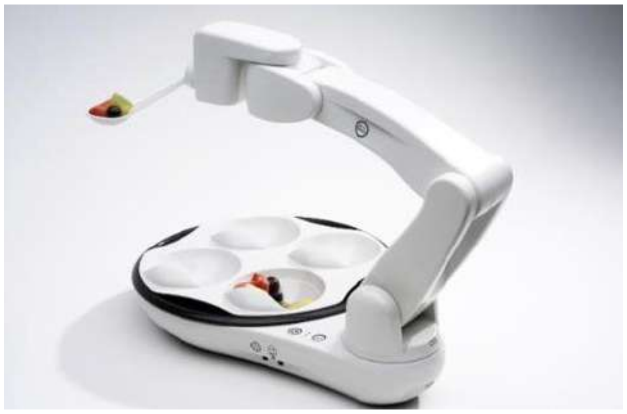


Figure 3. Example of a passive Feeding system – Omnibot (Source: Omnibot Technologies)

2.1.2.2.2. Active Feeding

General-purpose mobile manipulator to provide active feeding assistance thatautonomously delivers food inside a user's mouth, taking advantage of the robot's greater physical and sensing capabilities.

Figure 4. An active feeding robotic manipulator (Source: www.sciencedirect.com/science/article/pii)

2.2. Robotic Arms

Generally robotic arms can be both serial manipulators and parallel manipulators based on their link configurations and dependencies. In our project our main focus is on serial manipulators only.

2.2.1. Characteristics of Robotic Arms

2.2.1.1. Mechanical Design

Dynamics of robotic manipulators are highly coupled, time variable, and nonlinear. A robotic arm's mechanical design, which was inspired by the human hand, consists of a series of linkages that form what is known as a kinematic chain. The links' joints provide the system with the necessary rotational and translational capabilities. The end effector, also known as end of arm tooling, is the part of a robotic arm that interacts with its environment (EOAT). When developing an end effector, the three primary aspects to consider are the material of the end effector, its DOF, and its adaptability to different tools. For example, if a meal assistance robot is involved, the end effector material should be food grade, such as stainless steel or another material. Moreover, the parameters for or a better scooping/grabbing requires a 4 or 5 DOF robotic manipulator.

2.2.1.2. Workspace of a robot

The robot's workspace (also known as reachable space) is defined by the collection of all points which can be reached by the end effector. There are many variables on which the workspace depends such as the link lengths, rotational and translational limits, overall configuration of the mechanism, etc.

2.2.1.3. Actuation method

Actuators make up the joints of a robot. Actuation methods can be considered an important component of the hardware system for robots. To generate linear and rotational motions, a typical robot arm employs several actuation mechanisms. Pneumatic, hydraulic, and electrical actuators are the most prevalent actuation technologies used in robot arm systems. However, because the entire system is used to assist disabled people, meal assistance robots can be regarded as a separate category, according to previous research. Pneumatic and hydraulic actuation technologies may not be ideal for applications such as meal assistance robots due to their routine use in industrial environments that need significant forces. Therefore, meal assistance robots are designed with electrical actuation systems which provide more accurate and smooth controlling. Since actuations should be done in a safe manner, direct current (DC) and servo motors have mostly been used to control the motions of meal assistance robots. In terms of speed and accuracy, various feeding robots have used high speed stepper motors for accurate positioning and speed. One of the famous motors are NEMA 17, NEMA 23 as used in PR2, a 32-DOF Feeding arm.

2.2.2. General parameters of Robotic Arms

A robotic arm must work within its set parameters to ensure the best results and avoid any damage to its parts or the parts around it. Following are the parameters that are typically associated with robotic arms:

- ➔ **Number of axes:**

  The area required for the arm to work in. It might move linearly in two axes to pick and drop objects or work in three axes to make a 3D object.
- ➔ **Degrees of Freedom:**

  The number of points within which the robotic arm can be controlled. It usually has 6 degree of freedom.
- ➔ **Work envelop:**

  It simply refers to the robot's range of motion in all directions. Its operating envelope is the shape it produces when moving completely in all directions. Only items within this envelope can be worked by a robot.
- ➔ **Work space:**

  This is the area that the robotic arm's end effector can access. It is the area that the end effector can access; it is constrained to this little work area.

➔ **Kinematics:**

Robotic kinematics is the movement of the robot due to the arrangements and types of its joints.

➔ **Payload:**

It is the amount of weigh the robotic arm can lift and work with. Anything that exceeds the payload parameter might end up damaging the robotic arm.

➔ **Speed:**

The robotic arm's movement speed is very important. It can be described as either the overall speed or the angular speed. As an illustration, moving at an exceedingly slow pace on an assembly line.

➔ **Acceleration:**

A robotic arm cannot suddenly reach a specific speed. It necessitates a constant acceleration based on its other parameters and requirements.

➔ **Accuracy:**

Because of the maximum speed and acceleration, this is the best situation within the practical range. It is one of the most important elements in a robotic arm, because poor accuracy might result in misshaped products, similar to what happens in 3D printing.

➔ **Repeatability:**

The work a robotic arm can do a repeated basis. A user predefines its movements and it must follow the same movements again and again. Picking an object and dropping it into another place and doing this repeatedly is a common example.

➔ **Motion control:**

The robotic arm cannot go out of its working space or else it would damage the surrounding things. Its motion should be restricted within its working space to avoid any mishap.

➔ **Power source:**

A robotic arm can be driven by pneumatic, hydraulic or electric power sources. The most used in the modern age is one of the types of electric power sources, AC Servo. They are the most reliable and require very little maintenance.

➔ **Drive:**

The actuating system that moves the robotic arm into its desired positions. They may be attached directly to the segment or attached via gears or harmonically.

- ➔ **Compliance:**

  It is the control and monitoring of both the forces and positioning of the robotic arm. It is the measure of the angles and distance the arm will move when a force is applied.

2.2.3. Types of Robotic Arms

There are several types of robotic arms such as:

- ➔ **Articulated robot:**

  Used for assembly operations, die-casting, fettling machines, gas welding, arc welding and spray-painting. It is a robot whose arm has at least three rotary joints.
- ➔ **Parallel robot:**

  One application is a mobile platform that handles cockpit flight simulations. It is a robot with concurrent prismatic or rotational joints in its arms.
- ➔ **Cartesian robot / Gantry robot:**

  Mostly used for pick-and-place jobs, sealant application, assembly procedures, machinery handling, and arc welding. It is a robot with three prismatic joints on its arm, and each joint's axes coincide with a Cartesian coordinate.
- ➔ **Cobot:**

  Cobot, as opposed to traditional industrial robot applications that keep robots away from people, allow for human interaction. Commercial applications for Cobot include robotic research, dispensing, material handling, assembling, finishing, and quality assurance. Cobot safety may rely on rounded edges, lightweight building materials, and built-in speed and force constraints.
- ➔ **Cylindrical robot:**

  During assembly procedures, it is used for machine tool handling, spot welding, and die casting machine handling. It is a robot with axes forming a cylinder- shaped coordinate system.
- ➔ **Spherical robot / planner robot:**

  Used for machine tool manipulation, arc welding, gas welding, spot welding, die casting, and machine fettling. The axes of the robot in question form a polar coordinate system.
- ➔ **SCARA robot:**

  Used for pick and place work, application of sealant, assembly operations and handling machine tools. This robot features two parallel rotary joints to provide compliance in a plane.

➔ **Anthropomorphic robot:**

It is shaped in a way that resembles a human hand, i.e., with independent fingers and thumbs.

2.2.4. Open source Robots

There are several open-source robotic arms such as Thor, uArm swift, Kauda, BCN3D Moveo. All of them differ in terms of their cost, manufacturing methods, electronics and control systems. Among these few are discussed below:

2.2.4.1. Kauda

The Kauda robotic arm features a simple design and is completely open-source, including 3D printable parts, making it inexpensive. With a payload capacity of 300 grammes, this 5-DOF arm is powered by three stepper motors and two servo motors. When paired with an Arduino controller, it allows for quick and precise movements.

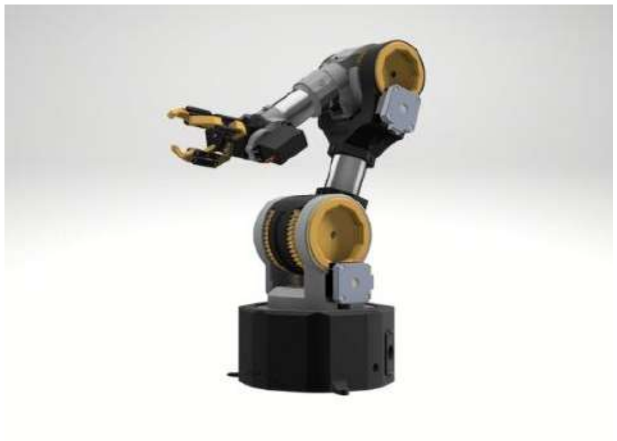


Figure 5. Kauda Robotic Manipulator (Source: https://www.instructables.com/KAUDA-Robotic-Arm/)

2.2.4.2. BCN3D Moveo

The BCN3D Moveo is an impressive 4-DOF arm controlled by an Arduino. It's fully 3D printed and open-source and has been tested as an educational tool, with many already operating in schools.

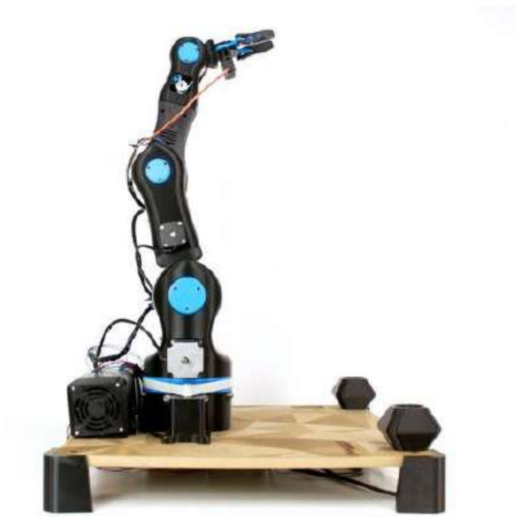


Figure 6. BCN-3D Moveo (Source: https://www.bcn3d.com/)

2.2.4.3. DOBOT Magician

A multifunctional robotic arm that performs functions such as 3D printing, laser engraving, writing and drawing. It is a 4-axis robotic arm that can lift up to 500 g worth of payload.

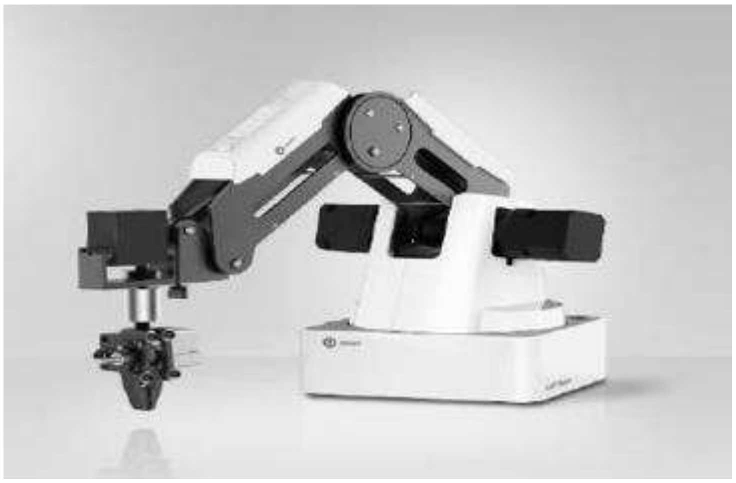


Figure 7. Dobot Magician (Source: www.dobot.com)

2.2.4.4. Thor

It is a 6 degree of freedom 3D printed robotic arm. Its configurations are those of a common market manipulator, yaw-roll-roll-yaw-roll-yaw. It can raise up to 0.75 kg and stands 0.625 m tall.

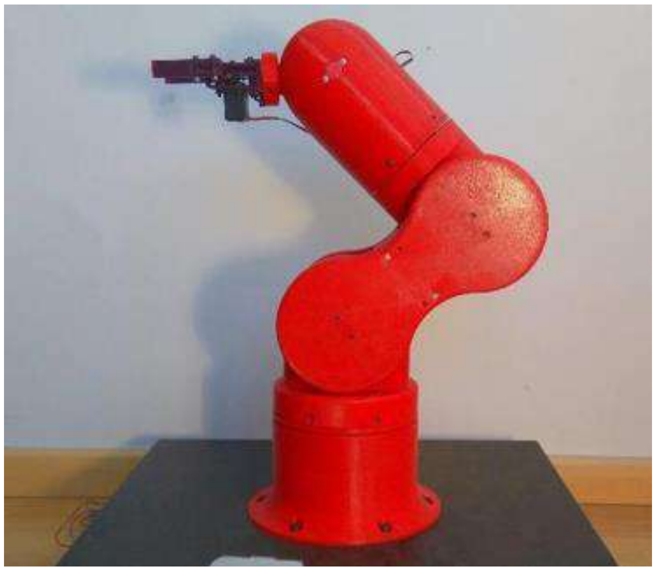


Figure 8. Thor Robotic Manipulator (Source: www.thormechanics.com)

2.3. Computer Vision

2.3.1. Object Detection

Object detection is a computer vision technology that allows us to recognize and pinpoint certain objects in an image or video. Object detection can be used to count the items in a scene, as well as find and track them in real time while precisely labelling them, using this type of localization and identification. To be more specific, object detection draws bounding boxes around the objects it discovers, allowing us to establish their placement inside (or how they move across) a scene. Object detection is critical since it allows us to partition or segment our object of interest. Reducing the amount of computing necessary to process photos and videos. Real-time monitoring and object detection made possible by quick object detection opens doors in connected businesses.

2.3.1.1. RCNN

The technique of locating and classifying items in an image is called object detection. Rectangular region suggestions and convolutional neural network features are combined in one deep learning method called regions with convolutional neural networks (R-CNN).

A two-stage detection algorithm is R-CNN.

1) A subset of regions in a picture that may contain an item are found in the first step.

2) The object is categorized in each region in the second stage.

To begin, the R-CNN detector generates region suggestions using an algorithm similar to Edge Boxes. To remove the proposal parts, the image is resized and cropped. CNN then classifies the resized and cropped regions. Finally, the region proposal bounding boxes are refined by a support vector machine (SVM) trained with CNN features.

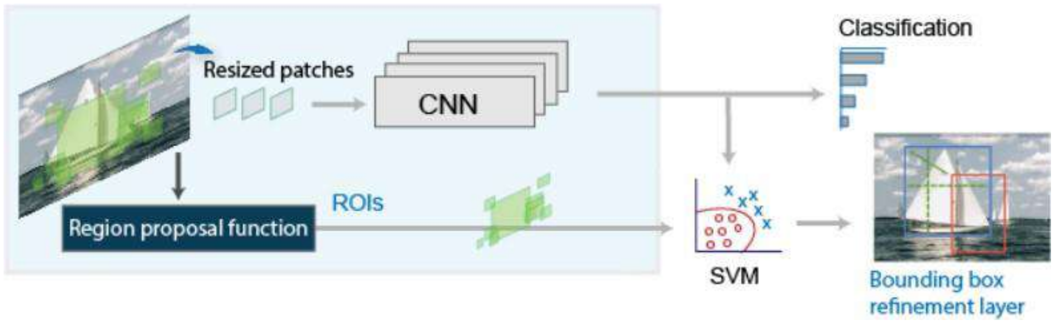


Figure 9. RNN Architecture (Source: https://towardsmachinelearning.org/)

2.3.1.2. SSD

Instead of sliding windows, SSD divides the image into grid cells, and each grid cell is in charge of identifying things in that part of the image. Object detection is just predicting the kind and location of an object within that area. SSD speeds up the procedure by eliminating the need for a regional proposal network. To compensate for the loss in accuracy, SSD implements a few enhancements such as multi-scale features and default boxes. These developments allow SSD to compete with faster R-accuracy CNNs while using lower-quality images, considerably increasing performance.

2.3.1.3. YOLO

An object detection framework requires only a single look. It captures the entire image in one shot. Grids are utilized to divide the image into pieces, and each grid is used to categories and localize the image. Once YOLO has predicted and generated the bounding boxes around the objects, we select the box with the highest probability for each class.

IOU: Non-suppression aids in determining whether the outcome is intersection or union. It is provided by the anticipated bounding box is satisfactory. It determines how the projected bonding box and the actual bounding box intersect over union. IOU stands for Intersection and Union Area.

IOU = Area of the intersection / Area of the union

1) If an IOU's value is greater than 0.65 or 0.7, they overlap.

2) If the IOU value is less than 0.65, they are non-overlapping.

The idea of anchor boxes is used when a grid cell's center contains two objects. The number of object centers in a grid determines how many anchor boxes are needed.

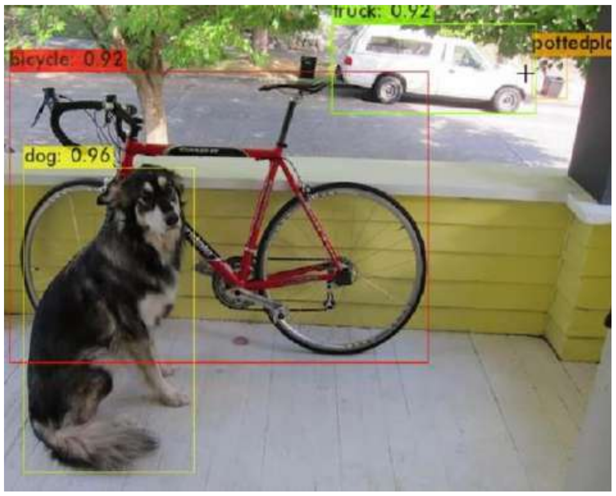


Figure 10. Bounding Boxes in YOLO (Source: https://www.section.io/)

2.3.1.4. Face detection

Face detection is a computer technology that uses artificial intelligence to find and recognize faces in digital photographs. Algorithms and machine learning are used by applications that locate human faces in larger photographs that frequently include non-face features such as landscapes, buildings, and other human body parts such as feet or hands. Because human eyes are one of the simplest qualities to perceive, face detection algorithms usually begin by looking for them. The computer may then attempt to recognize the iris, mouth, nose, and nostrils. When the algorithm determines that it has discovered a facial region, it does additional tests to confirm that it has spotted a face.

2.3.1.5. Open CV

OpenCV is a popular approach for detecting faces. Before employing the AdaBoost approach as a face detector, it first extracts feature images from a large sample set by extracting the face Haar features from the image. The algorithm's capacity to adapt to difficult settings, such as poor illumination and background blur, significantly improves face detection accuracy. Different training sets are constructed for use in subsequent work by varying the distribution probabilities of each sample in a collection of training sets. Then, for each training set, a weak classifier is trained and weighted. For instance, if each sample has a training class assigned to it, a new training set may be generated by modifying the distribution probability based on how successfully the training set was classified. The higher the classification accuracy rate, the smaller the distribution probability. The new training set is trained to generate a classifier, and the process is repeated to generate a large number of classifiers, increasing the weight of each classifier as classification accuracy increases.

2.3.2. Object Tracking

Once the initial position of the target object is known, object tracking refers to the capability to estimate or forecast the position of the target object in each subsequent frame of a video [12].

2.3.2.1. Image tracking

Image tracking is the process of automatically detecting and tracking images. It is most used in the realm of augmented reality (AR). When it receives a two-dimensional image as input from a camera, for example, the approach can be used to overlay a 3D graphical object on top of it. After superimposing the 3D graphic on top of the 2D planar surface, the user can adjust the camera without losing sight of either

2.3.2.2. Video tracking

Video tracking is the process of following a moving object in a video. Video tracking attempts to connect or link target things as they appear in each frame of the video. In other words, video tracking entails gradually evaluating the video frames and anticipating and constructing a bounding box around the item to connect its previous and current locations.

2.3.2.3. Object tracking camera

The real-time video stream of any camera can be used to apply contemporary object tracking algorithms [25]. As a result, object tracking can be performed with help of the video stream from a USB camera or an IP camera by providing the individual frames to a tracking algorithm. Real-time video feeds from one or more cameras can be used to improve object tracking performance by frame skipping or parallelized processing [28] [29].

2.3.3. Localization

Finding the chosen object's location inside an image or video sequence is referred to as localization. Following object detection in the image detection process, localization using a rectangular border [13] is required.

Depending on the granularity of the localization, this topic can be separated into two subtasks: object detection and object segmentation. In contrast to object detection methods, which identify the item and its location in an image by drawing a bounding box around it, the purpose of object segmentation is to categorize all of the pixels in an image in order to locate the targets [14]. While object detection looks for all of the things and their borders, localization looks for the object that is most visible in an image. Alex Net was the first neural network to be utilized for object detection or localization [30].

2.3.4. Depth Perception

The ability to visually perceive the world in three dimensions (or 3D) and to calculate the separation/depth of an object from the source is known as depth perception.

The input to our brain is in two dimensions because although the world we see is three dimensional, the image created on the human retina is only two dimensional (2D). But we can still see the world in three dimensions. The outcome of human evolution is the capacity of our brains to execute depth perception. It provides information about the depth of each thing, or more precisely, the relative proximity of each object to our eyes [26]. It is essential to daily living and keeps us from running into objects. We can use it to estimate an object's relative speed as well.

Stereo vision on the other hand helps humans to determine how far away items are by using simply the relative positions of an object in the two eyes. Sensory and motor skills are both required. Stereo vision is used in three- dimensional environments to assess how far an object is from a tool or machine. It is really nothing more than the human visual system of eyeballs. Two images, one from each eye, are obtained. When compared to the image on the right, the image on the left shows a tiny shift, which suggests depth. We shot two photos with two or more cameras to calculate depth information from the photographs. The fundamental idea is to examine at a scene from two or more perspectives and then use the disparity to explain the positioning, organization, and relationships of the elements in the scene. The geometry of stereo vision is as follows:

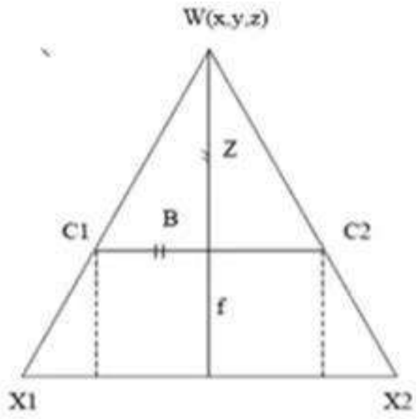


Figure 11. Geometry of stereo vision (Source: https://www.semanticscholar.org/)

- B is baseline
- C1 and C2 are camera centers
- F is focal length
- X1, X2 are image location in left and right cameras
- Z is depth

2.4. Control

In many robot manipulator applications, the control goal is to command the end-effector motion to obtain a desired response. Control inputs are applied to manipulator joints, and the intended position and orientation are commonly expressed in terms of a Cartesian coordinate frame coupled to the robot end-effector in relation to the base frame (i.e., the so-called task-space variables) [18]. As a result, a mapping (i.e., the inverse kinematics solution) is necessary to translate the desired task space trajectory into a form that can be used by the joint space controller. The control challenge for a robotic mechanism is typically characterized as follows: given an expected route, a numerical model of the mechanism and its interactions with the environment, a control method that sends force or torque signals to the actuators is found. As a result, the robotic mechanism can perform the predicted movement [22]. There are two important steps in the control design of a serial mechanical system.

First, a robotic end-effector travel path is specified, such as moving the end-effector from position A to position B. The motions of the joints can thus be computed based on the end-movement effector's trajectory and by using inverse kinematics to construct the required trajectory for the end-effector. The next step is to determine how much torque should be applied to joints for them to perform the desired motion. Inverse dynamic equations can be used to compute torque. Because the robotic system is exceedingly nonlinear, controlling the robotic manipulator to behave in a specific manner is difficult [24]. In the case of robotic systems, the coefficients of dynamic equations include joint and payload variables. These variables may be unknown or may change during the task. When a robotic mechanism is in motion, the joint variables change, causing the dynamic equation of the robotic system to change throughout the task.

2.4.1. Kinematics

- ➔ Inverse Kinematics

  Once a kinematic model has been developed, the kinematics must be inverted to acquire the desired actuator or configuration space variables. This is quite simple and has been extensively investigated for rigid manipulators. It can be accomplished via differential inverse kinematics (IK) [8] by direct inversion or by optimization [9].

- ➔ Paul's Method

  The systematic approach to solve kinematic equations, proposed by Paul, rewrites the transformation equation of base with respect to tool with the desired location. Thus, both sides of the FK matrix equation are left and right multiplied by inverse transformation matrices. In doing so, one gets equivalent equations having the same solutions. We get rid of sub transforms matrix and then by algebraic techniques, we get the solution.

- ➔ ANN Based

  The use of a neural network-based control method for joint tracking control of a conventional robot manipulator is a well-known idea [23]. Neural network controllers for a wide range of robot manipulator models, including rigid link manipulators and flexible joint manipulators, were developed in [10] and the references therein.

2.4.2. Path Planning and Motion Planning

In robotics, motion planning refers to the act of breaking down a desired movement job into discrete motions that satisfy movement limitations while potentially optimizing some component of the movement.

Consider a mobile robot moving within a building to a distant waypoint. It must complete this mission while avoiding walls and avoiding falling downstairs. A motion planning algorithm would take these tasks as input and generate the speed and turning commands that would be issued to the robot's wheels. Motion planning algorithms may be applied to robots with more joints (industrial manipulators), more complex tasks (e.g., object manipulation), different constraints (e.g., a car that can only drive forward), and uncertainty [21].

2.5. 3D Printing

2.5.1. Methods

In 3D printing, a person creates a design of an object using software, and the 3D printer creates the object by adding layer upon layer of material until the shape of the object is formed. The object can be made using several printing materials, including plastics, powders, filaments and paper.

There are several 3D printing technologies such as:

- **Stereolithography (SLA)**

  A liquid plastic is used in stereolithography as the source material, and this liquid plastic is gradually turned into a 3D object. Resin liquid is poured into a transparent-bottomed vat. To cure and harden a layer of the resin, a UV (Ultraviolet) laser traces a pattern on the liquid resin from the bottom of the vat. A lifting platform gradually raises the formed structure as the laser generates a distinct pattern on each layer to produce the required 3D shape.

- **Fused Deposition Modelling (FDM)**

  Production-grade thermoplastics can be used to construct products with the help of this technology. By melting a thermoplastic filament and layer-by-layer extruding the molten plastic, objects are constructed [15]. Complex structures can be made using specialized methods. A second substance that will act as support material for the object being made throughout the printing process, for instance, could be extruded by the printer. Later, the support material might be eliminated or dissolved.

- **Selective Laser Sintering (SLS)**

  SLS and stereolithography are somewhat comparable. SLS, on the other hand, uses a powdered substance that is poured into a vat [33]. In order to build up the object to be made, a layer of powdered material is applied using a roller to the top of the layer beneath it [16]. The powdered material is then laser sintered in accordance with a predetermined pattern. It's interesting that the part of the powdered material that isn't sintered can be used to create the support structure and that material can be taken out once the object is produced to be reused [17].

2.5.2. Issues with 3D Printing

3D printing, also known as additive manufacturing, is becoming popular with manufacturers [31]. The demand is growing due to some of the revolutionary benefits that it can provide. Like almost all technologies it has its own drawbacks that need considering

- Currently, 3D printers have small print chambers that limit the size of parts that can be printed. Anything larger will need to be printed in separate parts and assembled afterward. Because the printer must print more parts before manual labor is used to join the parts together, this can increase costs and time for larger parts [20].

- Although large pieces, as previously indicated, necessitate post-processing, most 3D printed products necessitate some form of cleanup to remove support material from the build and smooth the surface to get the desired finish. For post-processing, water jetting, sanding, a chemical soak and rinse, air or heat drying, assembly, and other methods are considered. A multitude of factors, including the size of the component being produced, the intended application, and the type of 3D printing technique used for manufacturing, influence the amount of post-processing required. As a result, while 3D printing allows for rapid part creation, post-processing may slow down the manufacturing process [32].
- Another key difficulty with 3D printing is that it is directly related to the equipment or method used, with certain printers having lower tolerances, indicating that the finished result may differ from the original design. This can be fixed in post-production, but keep in mind that it will extend the production time and cost.

Chapter 3: METHADOLOGY

3.1. Hardware specifications

The hardware of the project is inspired by BCN3D Moveo. The robotic manipulator will have 6 Degrees of Freedom. There will not be use of original end effector, but a custom-built end effector will be used. Moreover, instead of Arduino Mega, this project will use Raspberry Pi. The maximum reach of this manipulator is 400mm in all the three axis. There is a base motor which will control the ground link. It is a NEMA 17 stepper motor. A timing belt connects the motor with the ground link. For Link 2, there are 2 NEMA 23 stepper motors followed by a NEMA 17 Reductor motor with a 4:1 gearbox to increase the torque. The joint of end effector will be controlled by a NEMA 14 motor for controlling a customizable end effector. To control the motors TB6560 stepper motor drivers will be used.

3.1.1. Original CAD Model

BCN-3D Moveo is an open-source robotic arm developed by designed by BCN-3D Technologies. Its CAD model with components and Bill of Materials is available on GitHub [34].

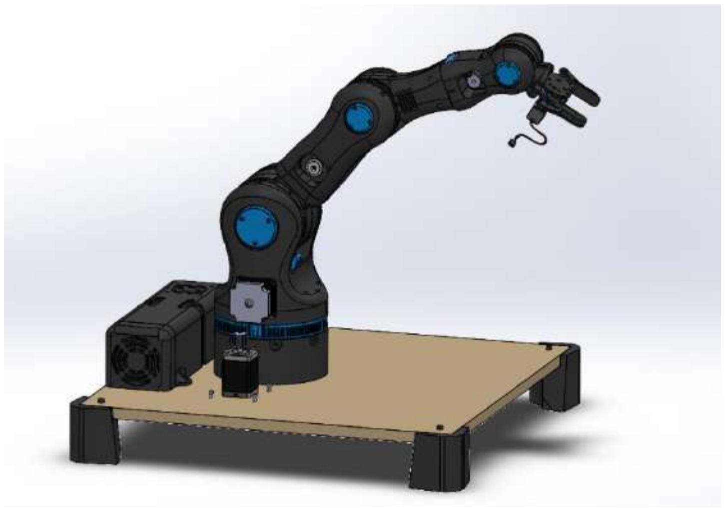


Figure 12: BCN3D Moveo CAD model

3.1.2. Design modifications in CAD

Apart from multiple design changes in different parts of the robot for easy manufacturing and time limitations two major changes are introduces which includes:

3.1.2.1. End-Effector

The end effector is designed keeping in view that engineering design considerations, which is to keep the weight and cost minimum.

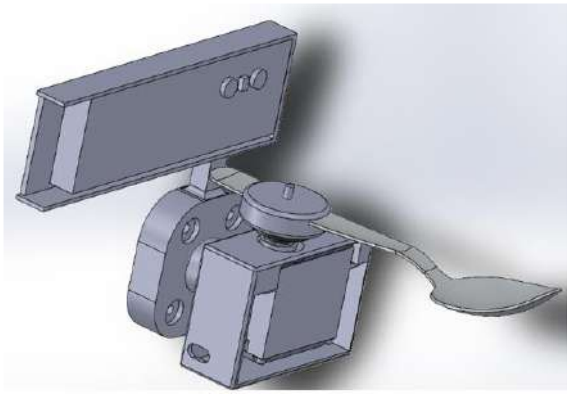
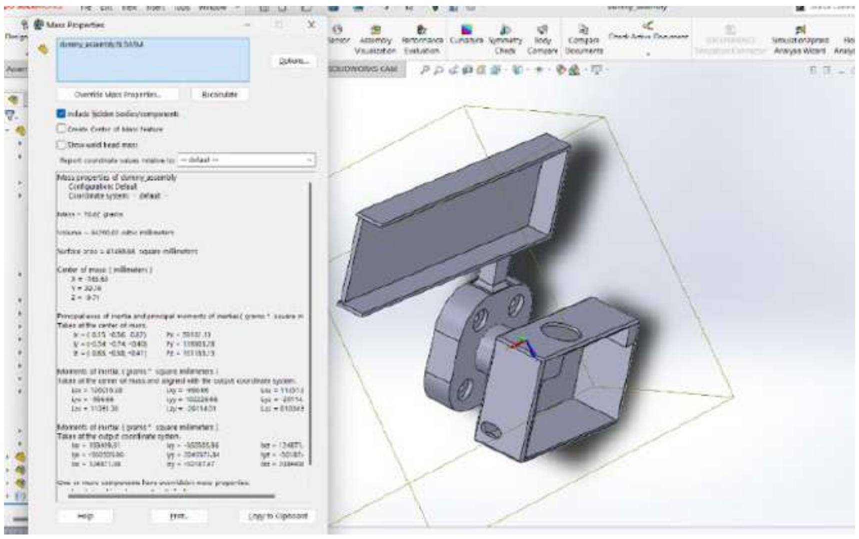


Figure 13. Custom End Effector

The original end effector weighs about 70.68 grams without a camera holder, whereas our own designed end effector with camera holder and utensil holder has a weight of 70.62 grams.



[✓ Include hidden bodies/components]
Create Center of Mass feature
Show weld bead mass
Report coordinate values relative to: -- default --

```
Mass properties of dummy_assembly
    Configuration: Default
    Coordinate system: -- default --

Mass = 70.62 grams

Volume = [illegible] cubic millimeters

Surface area = [illegible] square millimeters

Center of mass: ( millimeters )
    X = -[illegible]
    Y = [illegible]
    Z = [illegible]

Principal axes of inertia and principal moments of inertia: ( grams * square m
Taken at the center of mass.
    Ix = ( 0.15, [illegible], [illegible])     Px = [illegible]
    Iy = ( [illegible], [illegible], [illegible])     Py = [illegible]
    Iz = ( 0.85, [illegible], [illegible])     Pz = [illegible]

Moments of inertia: ( grams * square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = [illegible]     Lxy = [illegible]     Lxz = [illegible]
    Lyx = [illegible]     Lyy = [illegible]     Lyz = [illegible]
    Lzx = [illegible]     Lzy = [illegible]     Lzz = [illegible]

Moments of inertia: ( grams * square millimeters )
Taken at the output coordinate system.
    Ixx = [illegible]     Ixy = [illegible]     Ixz = [illegible]
    Iyx = [illegible]     Iyy = [illegible]     Iyz = [illegible]
    Izx = [illegible]     Izy = [illegible]     Izz = [illegible]

One or more components have overridden mass properties.
```

Figure 14. Mass of designed End Effector on Solidworks

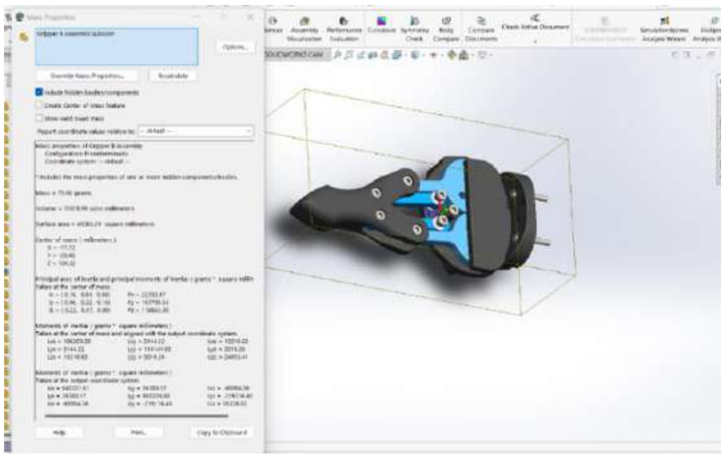


Figure 15. Mass of original End Effector on Solidworks

3.1.2.2. Counterweight mechanism

The rated motors (NEMA 23) at link 2 have a holding torque of 2.7N.m each to hold the weight of rest of the links. As we could not find the rated motors, our NEMA 23 motors have a torque of 1.8 N.m each because of which the robot was unable to bear the load of entire assembly. To cater to this problem, we built a counterweight mechanism. A counterweight mechanism is a device used to balance or counteract an object's weight or force. It is frequently applied in a variety of settings to provide stability, regulate motion, or lessen the effort needed to move large things.

3.1.2.2.1. Design

For the design of counterweight mechanism, the factors of consideration were:

- The mass of counterweight
- The amount of torque that the counterweight can reduce, to make it suitable for our application.

For counterweight, we selected two Polycarbonate sheets of thickness 5mm to be attached on both sides of link 2 above each motor. First, a free body diagram was made for the static structure based on the dimensions shown below. These dimensions were considered based on the safety of the structure and to make the motors provide the required torque a specific angle.

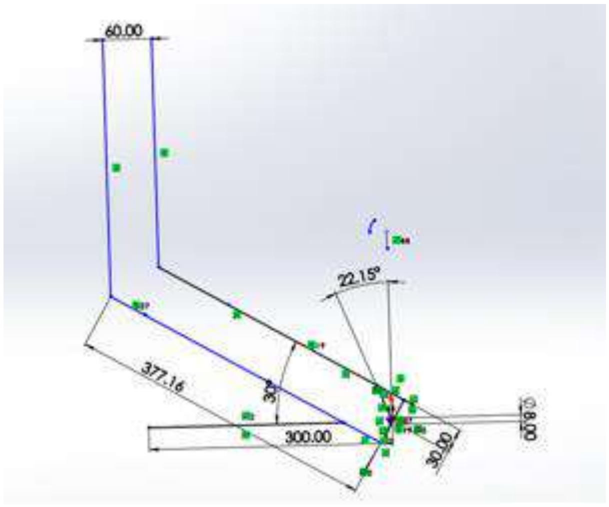


Figure 16. Counterweight mechanism sketch

The weight to be hung from the mechanism weighed 500 grams. The own weight of the mechanism is approximately 70 grams. From the above dimensions and free body diagram, the torque was calculated as:

$$T = r.F.cos\ \alpha$$

Here the length r =377mm=0.377m and weight is 0.500*9.8 and α=30 degrees:

$$T = 0.377 * 0.500 * 9.8 * cos\ 30$$

$$T = 0.377 * 0.500 * 9.8 * cos\ 30$$

$$T = 1.599\ N.m$$

It means that at the home position of the robot, on each side of Link 2, there acts a torque of 1.599/2= 7.49 N.m.

When the robot is at soft home, the counterweight touches the ground. In that case, there is no load on the motors for forward movement.

3.1.2.2.2. Torque requirement

For pulling up link 2, we have used fishing wire which is attached to center of gravity of link 2, so that it can efficiently provide the required torque.

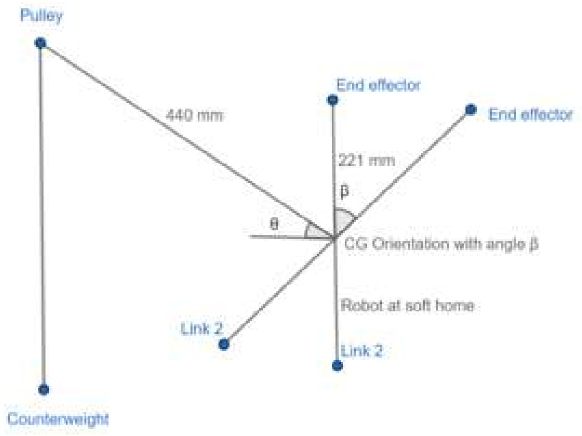


Figure 17: Free body diagram of counterweight mechanism

As the counterweight mechanism is attached to link 2, the length from link 2 to center of gravity of rest of structure is 221 mm or 0.221m. The weight of the entire structure acting at the Centre of gravity is calculated to be (2.41 kg) *(9.8m/s2).

Considering the dimensions of link 2, the torque at center of gravity, oriented at angle β can be calculated as:

$$T = 0.221 * (2.41) * 9.8 * cos(90 - \ \beta)$$

$$T = 0.221 * (2.41) * 9.8 * sin\ \beta$$

$$T = 5.219 * sin\ \beta$$

When the link 2 moves, making an angle β, the required torque increases. For example, at 60 degrees, the required torque is about 4.519 N.m.

3.1.2.2.3. Counter torque

The holding torque of motors combined is 3.6N.m. The counterweight applies torque in opposite direction. The weight is 500 grams which means a force of (0.5 kg) *(9.8m/s2) will be applied. The angle of wire to the pulley will is θ.

$$T = 0.440 * (0.5) * 9.8 * cos(\theta)$$

$$T = 2.156 * sin\ \theta$$

From experiments, the angle θ=25 degrees, when β=60 degrees, the torque provided by counterweight is 2.025 N.m.

It means the motors torque and counterweight torque will be added and provide a total torque of 5.625 N.m.

3.1.2.2.4. Counterweight limitation

When the weight exceeds 500 grams the forward torque also increases, which means this is the design limitation. To be on the safe side, maximum angle achieved by the robot is 60 degrees, because at that position, the required torque for movement of link 2 is 4.519N.m and counterweight mechanism provides 5.625 N.m. It means there is a margin of 1.10 N.m for the robot to avoid jerks.

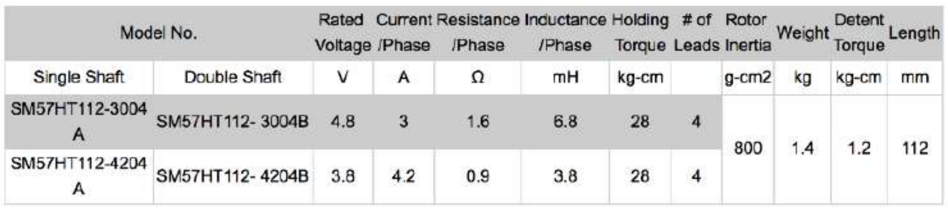
The weight is limited because if we exceed this limit the counter torques becomes more than the motor holding torque hence will not be useful.

3.1.3. Actuators

In this project total of 7 rotary electric actuators are used. Among which 6 are stepper motors and one is servo motor. Types of stepper motors used in the project are discussed below:

3.1.3.1. Nema 17 (First joint motor)

The required motor model was SM42HT33-1334. This motor will be used to rotate the link 1 of BCN3d Moveo. The torque of this motor is 2200 gram.cm [35].



Model No.		Rated Voltage	Current /Phase	Resistance /Phase	Inductance /Phase	Holding Torque	# of Leads	Rotor Inertia	Weight	Detent Torque	Length
Single Shaft	Double Shaft	V	A	Ω	mH	kg-cm		g-cm2	kg	kg-cm	mm
SM57HT112-3004 A	SM57HT112- 3004B	4.8	3	1.6	6.8	28	4	800	1.4	1.2	112
SM57HT112-4204 A	SM57HT112- 4204B	3.8	4.2	0.9	3.8	28	4				

Figure 18: Parameters of Nema 17 for joint one (Source: www. https://www.alldatasheet.com/)

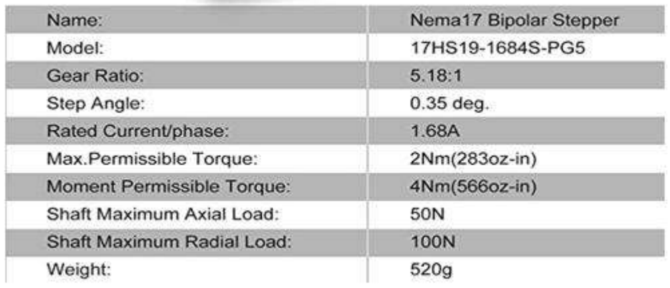
3.1.3.2. Nema 23 (Second joint motor)

Nema 23 SM57HT112-3004A motor was required to be used to carry the weight of overall structure of BCN3D Moveo. There are 2 NEMA 23 motors attached on the base link (Link 1). The torque of this motor is 23 Kg.cm.

Because of unavailability of the motor locally we have used second hand alternative motor which had lesser torque hence, for this reason counter weight mechanism was designed to get along.

3.1.3.3. Nema 17 (Third joint motor)

Stepper 17HS19-1684S-PG5 motor is attached on Link 2 of the structure for the third joint. The torque is 2Nm. There is a gear box with gear ratios 4:1 attached to it, so that there is velocity reduction and increment of torque [36]



Name:	Nema17 Bipolar Stepper
Model:	17HS19-1684S-PG5
Gear Ratio:	5.18:1
Step Angle:	0.35 deg.
Rated Current/phase:	1.68A
Max.Permissible Torque:	2Nm(283oz-in)
Moment Permissible Torque:	4Nm(566oz-in)
Shaft Maximum Axial Load:	50N
Shaft Maximum Radial Load:	100N
Weight:	520g

Figure 19. Nema 17 Stepper 17HS19-1684S-PG5 Specifications (Source: https://www.alldatasheet.com/)

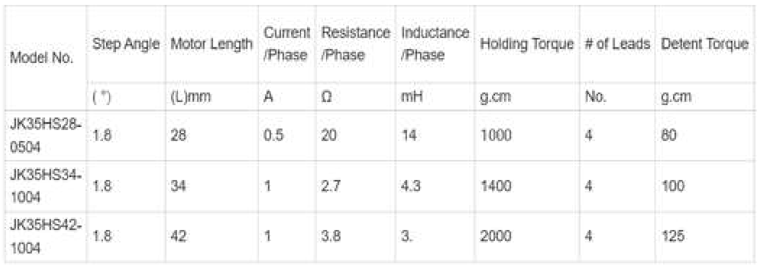
3.1.3.4. Nema 23 (Forth joint motor)

Stepper Motor SM42HT47-1684 is for the rotation of joint 4. The torque of this motor is 3600 g.cm with 1.68A of current per phase.

Figure 20. Stepper Motor SM42HT47-1684 Specifications (Source: https://www.alldatasheet.com/)

3.1.3.5. Nema 14 (Fifth joint motor)

SM35HT36-1004A motor will be used for the rotation of the link that holds the end effector. It has a torque of 1400 g.cm.



Model No.	Step Angle	Motor Length	Current /Phase	Resistance /Phase	Inductance /Phase	Holding Torque	# of Leads	Detent Torque
	(°)	(L)mm	A	Ω	mH	g.cm	No.	g.cm
JK35HS28-0504	1.8	28	0.5	20	14	1000	4	80
JK35HS34-1004	1.8	34	1	2.7	4.3	1400	4	100
JK35HS42-1004	1.8	42	1	3.8	3.	2000	4	125

Figure 21. Nema 14 SM35HT36-1004A Dimensions and Specifications (Source https://www.alldatasheet.com/)

3.1.3.6. MG995 Servo (End-effector motor)

Plastic servo is used for the actuation of the sixth degree of freedom which is at the end-effector.

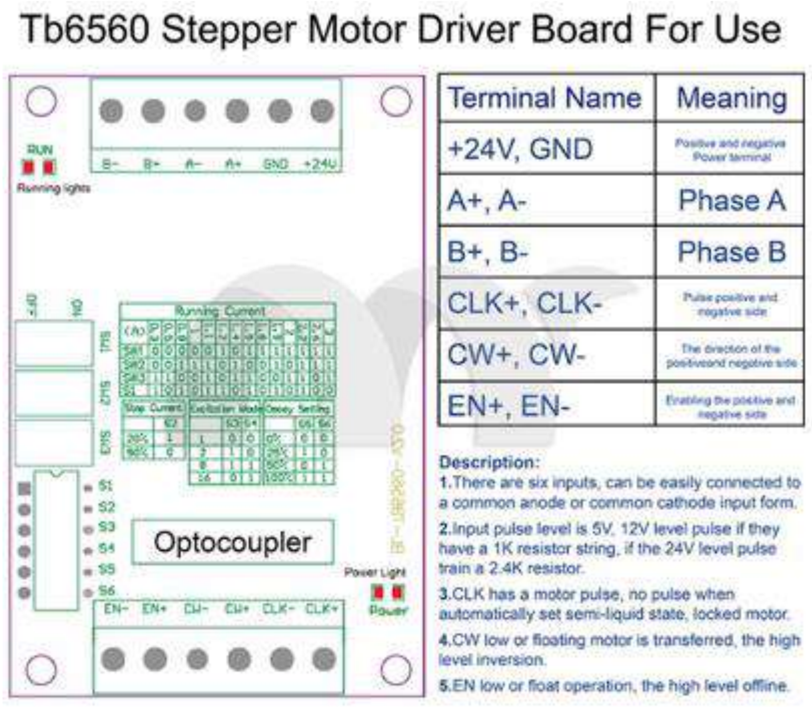
Figure 22: Servo motor for end-effector (Source: www.amazon.in)

3.1.4. Motor driver

Bipolar stepper motors may be driven with up to 3.5 amps per phase using the commonly used stepper motor driver IC known as the TB6560. It is frequently used in CNC machines, 3D printers, and other applications requiring precise motion control. Micro stepping capabilities offered by the TB6560 driver provide smoother and more accurate motor control [37]. The motor can move in extremely small steps because to its capability for 1/16 micro stepping.

This is crucial for high precision applications like CNC milling and 3D printing. The TB6560 can handle an input voltage range of 9V to 42V DC. The maximum output current of the TB6560 is 3.5A, making it suitable for driving a wide range of stepper motors. It supports up to 1/16 micro stepping, which allows for smoother motion and higher precision. Moreover, it includes a built-in overheat protection circuit that will shut down the driver if the temperature exceeds a certain threshold. There is a short-circuit protection to prevent damage to the driver in case of a short circuit.

The TB6560 can accept pulse input frequencies up to 100 kHz. The maximum power consumption of the TB6560 is 25W.



Terminal Name	Meaning
+24V, GND	Positive and negative Power terminal
A+, A-	Phase A
B+, B-	Phase B
CLK+, CLK-	Pulse positive and negative side
CW+, CW-	The direction of the positiveand negative side
EN+, EN-	Enabling the positive and negative side

Description:
1.There are six inputs, can be easily connected to a common anode or common cathode input form.
2.Input pulse level is 5V, 12V level pulse if they have a 1K resistor string, if the 24V level pulse train a 2.4K resistor.
3.CLK has a motor pulse, no pulse when automatically set semi-liquid state, locked motor.
4.CW low or floating motor is transferred, the high level inversion.
5.EN low or float operation, the high level offline.

Figure 23. TB6560 Specifications and Pinout (Source: https://projectiot123.com/)

3.1.5. Power supply

AC to DC power supply rated 120/240 VAC 20A input with a maximum output power of 320 watts, 24 VDC. It is frequently utilized in a range of commercial and industrial applications, including driving electrical devices, lights, and motors. The efficiency rating is a crucial factor to consider when choosing a power supply. A greater efficiency rating indicates that the power source will use less energy during conversion, which can eventually save money and have a less negative impact on the environment.

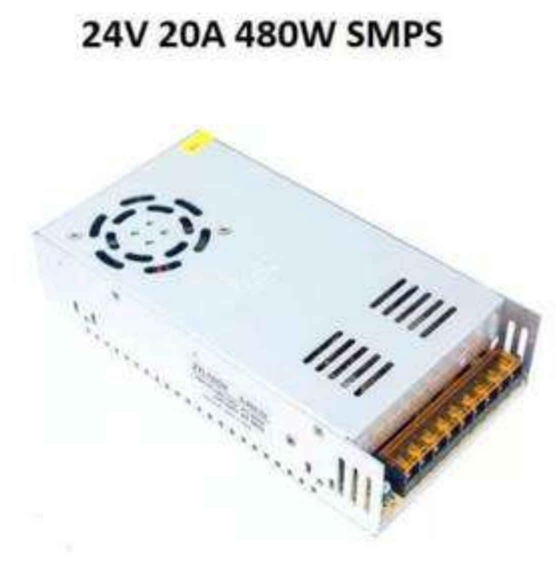


Figure 24: Power supply (Source: https://projectiot123.com/)

3.1.6. Laser Distance sensor

The VL53L0X is a laser distance sensor module that measures a target's distance precisely using Time-of-Flight technology. It offers precise distance measurements regardless of the target reflectance, in contrast to traditional technologies. This beam strikes the object's surface and deflects back. The time it takes for a laser beam to strike an object's surface and bounce back to the sensor is referred to as the "time of flight." Even if an object's surface is very reflective, the VL53L0X can still measure its distance range.

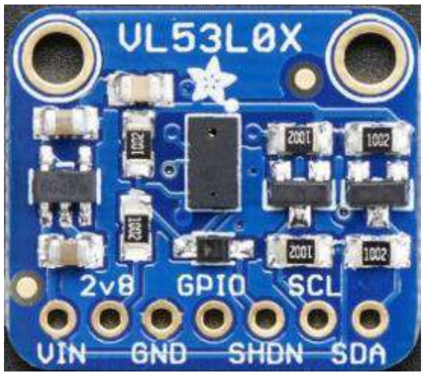


Figure 25: VL53L0X Laser sensor (Source: www.electrobes.com.pk)

VIN	Connected to the anode of the power supply
GND	Connected to the ground wire
SCL	12C SCK
SDA	I2C serial data wire
XSHUT	Reset pins only available under low level
GPIO1	Interrupt pins

Table 1. Pin Configuration of VL53L0X Laser sensor

3.1.7. Etron esp8702 camera

According to Etron, the left and right lenses can both take pictures at a rate of more than 30 frames per second because to the integration of the eSP870 single-chip image processor and two CMOS sensor chips into one module. The solution also includes a built-in 3D depth detecting controller that is suitable for somatosensory gaming.

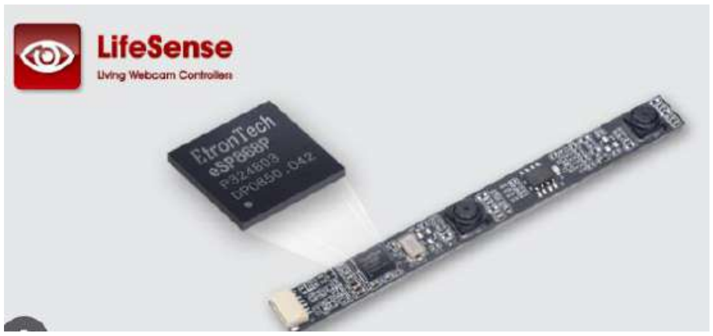


Figure 26: Etron stereo camera (Source: www.etron.com)

3.1.8. Limit Switches

In this project, no hardware based feedback is taken from the actuators. Hence, in order to set a hard home for the robot we implemented limit switches at the maximum operating limits of the joints.

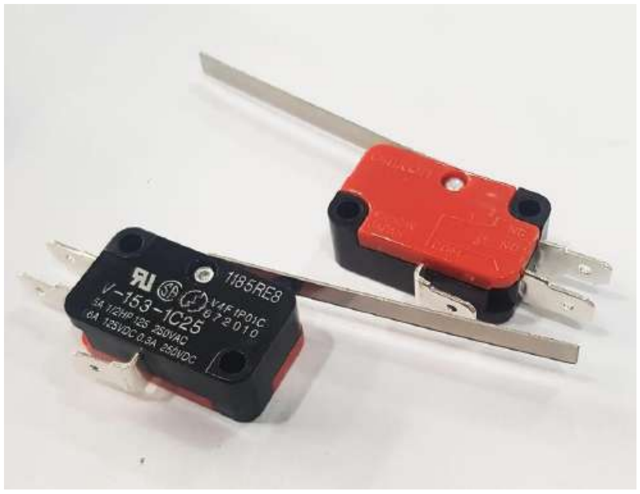


Figure 27: Mechanical limit switches (Source: www.electrobes.com.pk)

3.1.9. Buck Convertor

As the complete system is developed as one device powered from one power supply. Different modules required different power rating mostly 24V which is already available and secondly 5V for which we used buck convertor to step down DC voltage to the required limit.

Figure 28: Buck convertor 5-40 V 9A (Source: https://learnabout-electronics.org/PSU/psu31.php)

3.1.10. Controller

The Raspberry Pi was created by the Raspberry Pi Foundation to offer a cost-effective platform for programming exploration and instruction. It is a computer the size of a credit card. The third iteration of the Raspberry Pi is called the Model B. The Raspberry Pi can perform many tasks that a typical desktop computer can, including programming, word processing, spreadsheets, high-definition video, gaming, and high-definition video. The board's four USB ports may be used to connect USB accessories like keyboards and mice. There are several online resources for the Raspberry Pi and more than twelve million Raspberry Pi's have been sold. It is built around the BCM2837 system-on-chip (SoC), which has a potent VideoCore IV GPU and an ARMv8 quad-core processor running at 1.2 GHz [38]. Along with Microsoft Windows 10 IoT Core, the Raspberry Pi can run the whole spectrum of ARM GNU/Linux distributions, including Snappy Ubuntu Core, Debian, Fedora, and Arch Linux.

Features

- 1.2 GHz quad-core BCM2837 ARMv8 64bit CPU
- 1 GB RAM
- VideoCore IV 3D graphics core
- Ethernet port
- 802.11n Wireless LAN (Wi-Fi)
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- Four USB ports
- Full-size HDMI output
- Four-pole 3.5 mm jack with audio output and composite video output
- 40-pin GPIO header with 0.1″-spaced male pins that are compatible with our 2×20 stackable female headers and the female ends of our premium jumper wires.
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)

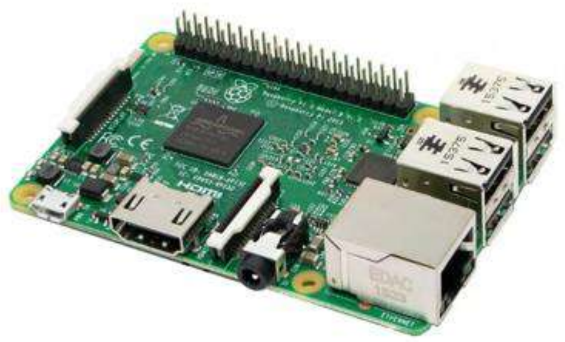


Figure 29: Raspberry pi 3B (1GB RAM) (Source: https://learnabout-electronics.org/PSU/psu31.php)

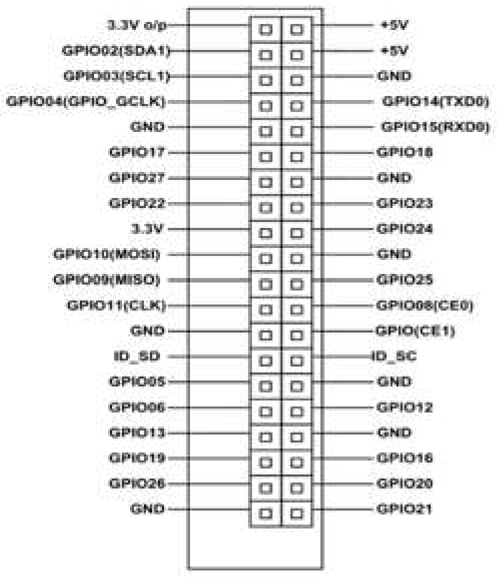


Figure 30: Raspberry pi 3B pinouts (Source: https://learnabout-electronics.org/PSU/psu31.php)

3.2. Mechanical analysis and simulations

The mechanical properties of the Robot were evaluated on Ansys Workbench.

3.2.1. Material selection and properties

The material used for hardware of Robot is PLA+. It was added to the Ansys Workbench manually by adding the properties in the Engineering Data Section. The Physical properties of the material are as follows:

- Tensile breaking strength: 57.8 MPa.
- Modulus of longitudinal elasticity: 3.3 GPa.
- Flexing strength: 55.3 MPa.
- Rockwell hardness: R70-R90.
- Extrusion temperature: 205-230 °C.
- softening temperature: 50 °C.

3.2.2. Static Analysis

For Static Analysis, the robot was positioned in 3 configurations which could have the maximum load on the base motors. These were the most vulnerable positions for failure to occur.

There were several assumptions such as:

- The Structure is a Solid-Body (which means the infill is 100%)
- The Force is uniformly distributed.
- The weight of end effector with food in spoon is about 300 grams (which is equal to 2.94N or approximately 3N.
- The mesh size was kept as 0.002m.

3.2.2.1. First Configuration

The first configuration is shown in the figure below. The load due to end effector while picking up the food was assumed to be 3N. Therefore, a force of 3N was applied at that joint.

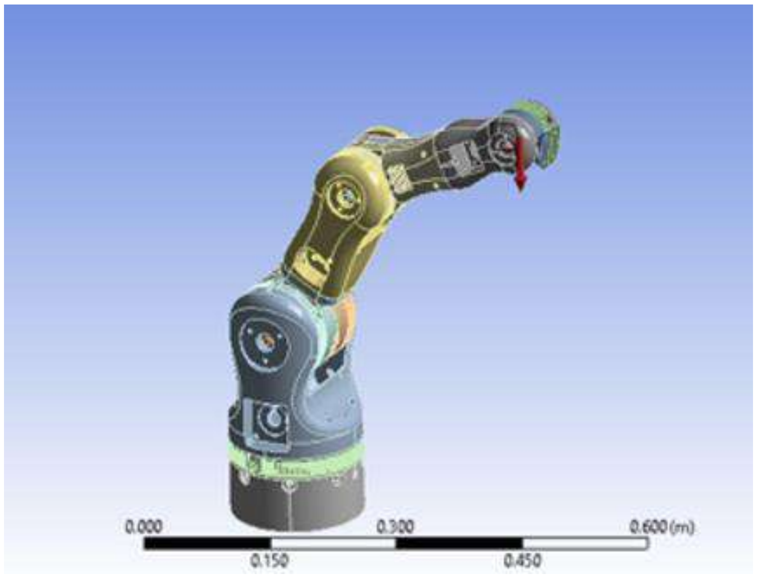


Figure 31. BCN3D Moveo placed in Configuration 1

3.2.2.2. Second Configuration

The second configuration is shown in the figure below. A force of 3N was applied at the joint of end effector.

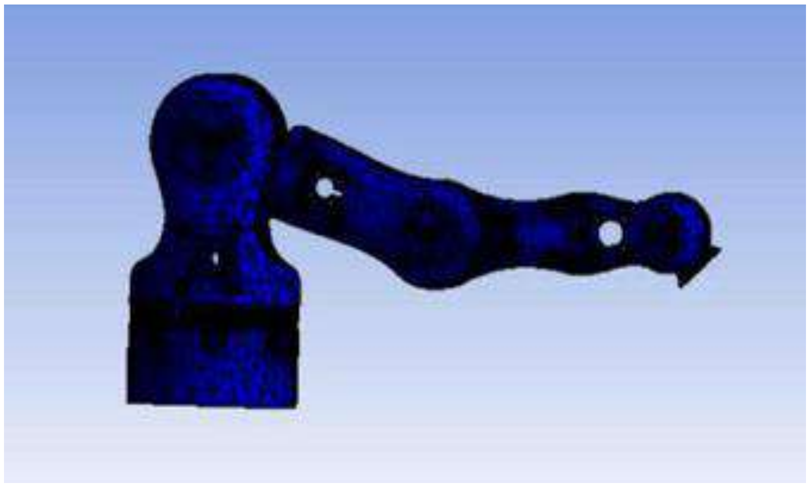


Figure 32. Equivalent Stress in Configuration 2

3.2.2.3. Third Configuration

The third configuration is shown in the figure below. Same as the first two configurations, a force of 3N was applied at the joint of end effector and the evaluated results are shown.

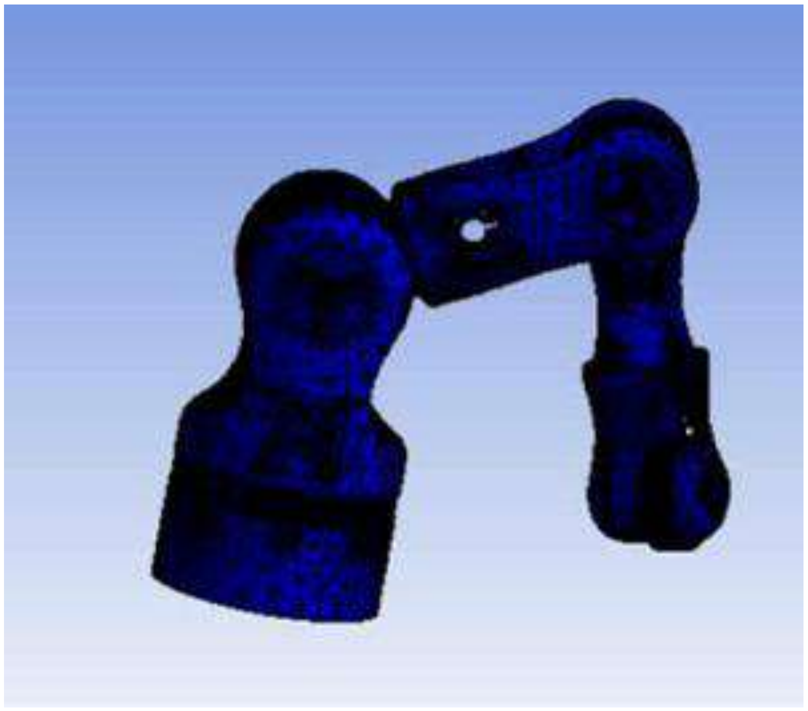


Figure 33. Equivalent Stress in Configuration 3

3.2.3. Dynamic Analysis

For Dynamic Analysis, a time varying force (cyclic force) must be applied. A time varying force (between 3N and -3N was applied) at the end effector joint, and the results for fatigue life of the material were obtained. They are shown below.

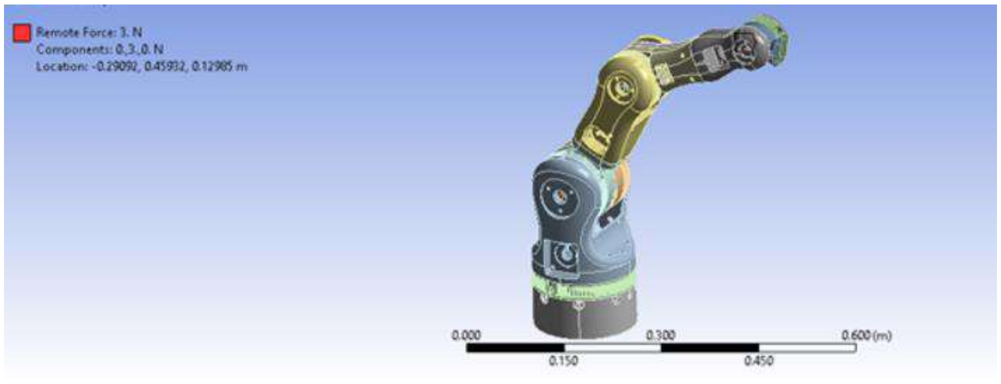


Figure 34. Time Varying Force between 3N and -3N applied on End Effector

3.3. Manufacturing

The whole structure of robot was manufactured using 3d Printing. Apart from counter weight mechanism which is made using polycarbonate material and manual machining process.

3.3.1. 3D Printers

Two printers based on FDM (Fused Deposition Modelling) [40] were used. The printers used were:

➔ FlashForge Creator Pro 2

➔ Creality Ender 5

3.3.1.1. FlashForge creator pro 2

Majority of printing is done on this printer. The specifications of this printer are [41]:

- Extruder Quantity: 2
- Nozzle Diameter: 0.4 mm
- Maximum Extruder Temperature: 240℃ (464°F)
- Print Speed: 30-100mm/s
- Maximum platform Temperature: 120℃ (248°F)
- Filament Compatibility: PLA, HIPS, ABS, PVA
- Filament Diameter: 1.75mm (0.069IN)
- Print Volume: 200*148*150mm (7.9*5.8*5.9IN)
- Layer Thickness: 0.1mm-0.4mm
- Print Precision: ±0.2mm

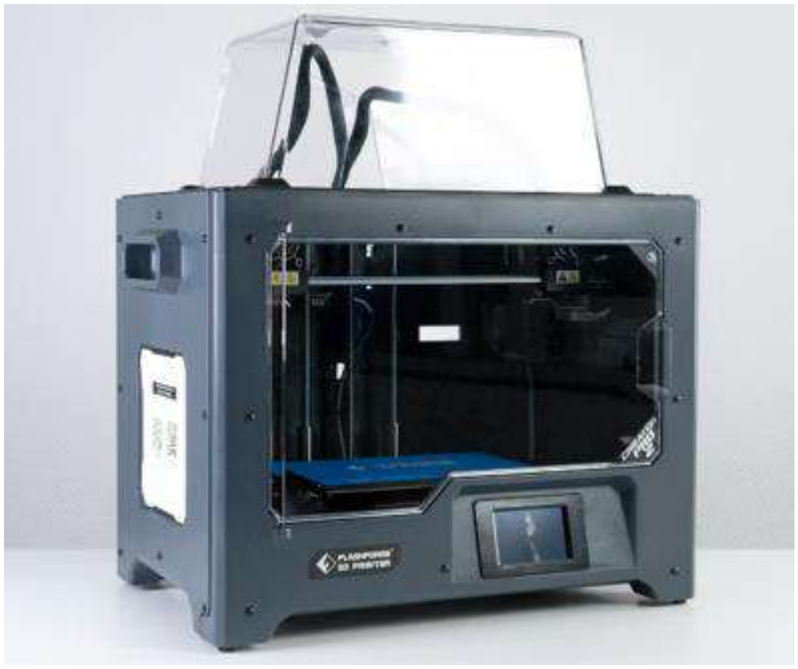


Figure 35: FlashForge creator pro 2 FDM dual extruder 3D printer (Source: https://www.flashforge.com/)

3.3.1.2. Creality Ender 5 Plus

This printer was used to print limited parts due to its bigger printing area and to save printing time. The specifications of this printer are:

- Extruder Quantity: 2
- Nozzle Diameter: 0.4 mm
- Maximum Extruder Temperature: 260℃
- Maximum platform Temperature: 110℃
- Filament Compatibility: PLA, PETG, ABS, PVA
- Filament Diameter: 1.75mm (0.069IN)
- Print Volume: 350*350*400mm
- Layer Thickness: 0.1mm-0.4mm
- Print Precision: ±0.1mm)

3.3.2. 3D Printing parameters

The parameters for printing of components varied a little bit for small components as compared to large components. The table below shows different parameters settings that we adjusted depending upon the part and conditions:

Fill Density	10% to 30%
Extrusion Temperature	216 Degrees Celsius
Base Print Speed	60 mm/s
Shell count	4 (depending upon the part)
Brim	Enables with 1.5 mm margin
Layer Height	0.13 mm

Table 2: 3D printing parameters

3.3.3. Interlocking joints

Interlocking joints are a common method for connecting components that are regularly assembled and disassembled. Due to limited bed dimensions of FlashForger Creater Pro 2 bed, we made interlocking joints for parts of BCN3D moveo that were taking more than 8 hours to print using the required settings.

3.3.3.1. Design factors

There are 2 forces to consider when designing interlocking joints:

➔ Friction - the vital force keeping the joint in place. The higher the friction and harder it will be to pull apart, the tighter the joint is.

➔ Tension - the force that acts to pull the joint apart.

Due to 10% infill, there was a significant reduction in weight of the components, so we designed the interlocking joints as key substrate type.

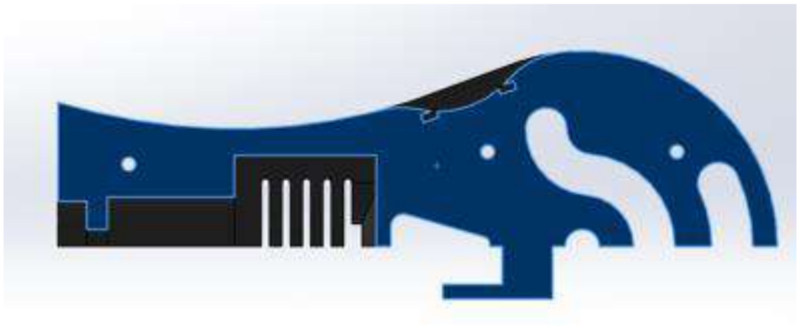


Figure 37. A Part 3M2C with Interlocking joint

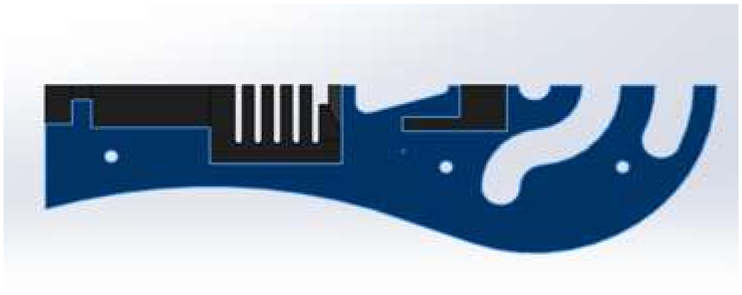


Figure 38. A Part 3M2C with other Interlocking joint

3.3.3.2. Structural analysis

All the interlocking joints were tested to keep the extruded part of the key model with minimum area so that volume is minimum and printing time is also less. It also saved time as the time limit was 8 hours. For example, a part named as 2M2HA on Link 2 will bear a weight of 22.54N for the rest of structure. It was sliced in 3 lock and key type models. The resultant stresses were:

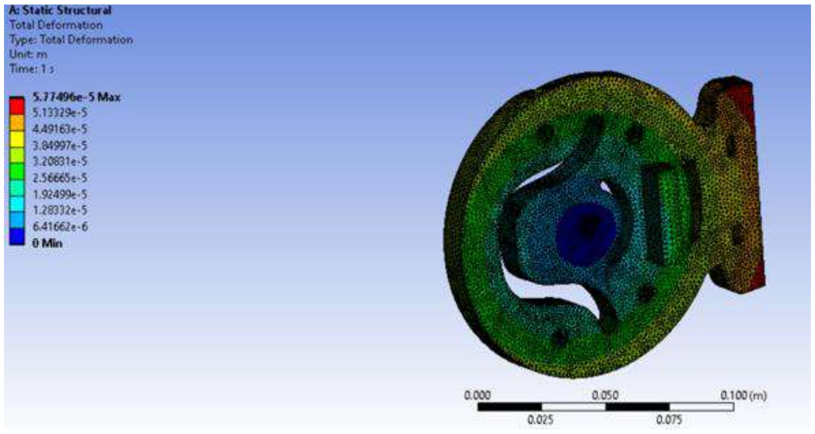


Figure 39. Total Deformation of joint 2M2HA

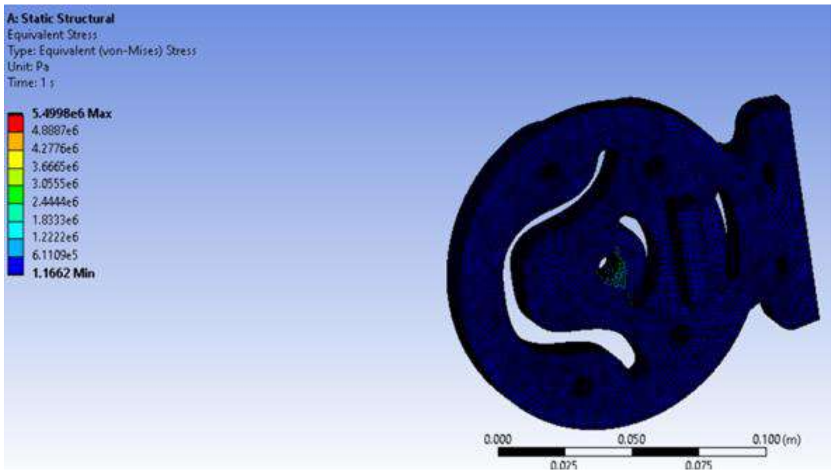


Figure 40. Equivalent Stresses on joint 2M2HA

The results on Ansys showed that that equivalent stress on whole body was 1.1662 Pa, except for the hole supporting the rod that connects other links. The Stress on the hole was 3.06665 Pa approximately. Similarly, the maximum deformation was 0.0000577496 m.

3.4. Robot Assembly

The robot assembly was done following the assembly manual provided by the BCN3D for this open source robotic arm. For the upgraded designs the assembly was done as seemed feasible.

3.5. Face detection and localization

For face detection of the patient, the Etron esp870U camera model is used to get the video feed. Etron esp870U is a single chip image processor plus CMOS sensor integrated to capture images at a rate of more than 30 fps. Image data is transferred by USB 2.0 port. [42]

3.5.1. Open CV

To get the video feed from camera and perform further processing Open CV library in python is used. OpenCV is an open-source computer vision and machine learning library. OpenCV provides infrastructure for computer vision application and accelerates the use of machine learning. The built in algorithms are used for detection and recognition of faces, identification of objects, tracking of camera movements, tracking of moving objects, 3D object extraction and produce 3D point clouds from stereo cameras etc. To work on a deep learning module in OpenCV it has a module named DNN.

3.5.2. DNN

To run the Deep Learning Algorithm using OpenCV we used the DNN module for real time evaluation. Deep neural networks (DNNs) can be used with OpenCV by using the library known as the OpenCV DNN module. It enables the usage of well-known DNN models, like those from TensorFlow and Caffe, and their execution on a range of hardware platforms, including CPUs, GPUs, and mobile devices. We are using the Caffe model.

3.5.3. Media Pipe

MediaPipe is an open-source framework that offers a pipeline of pre-made parts and tools for creating different kinds of multimedia processing applications. [43] It provides an adaptable and effective infrastructure for managing media data, including video and audio, as well as carrying out real-time processing tasks like object detection, pose estimation, facial recognition, hand tracking, and more. The desktop/server, Android, iOS, and embedded devices like the Raspberry Pi and Jetson Nano are all supported by this cross-platform framework.

3.5.4. Face Mesh

Face Mesh is a system that estimates 468 3D face landmarks in real-time even on mobile devices. It uses machine learning (ML) to infer the 3D facial surface and only needs one camera input—a specialized depth sensor is not required. The method provides the real-time speed necessary for live experiences by combining GPU acceleration across the pipeline with lightweight model architectures.

The two real-time deep neural network models in our ML pipeline—a detector that acts on the entire image and computes face locations and a 3D face landmark model that uses those locations to forecast an approximation of the 3D surface through regression—work together to create our system [44]. The necessity for typical data augmentations such affine transformations, which include rotation, translation, and scale adjustments, is significantly reduced when the face is precisely cropped. Instead, it enables the network to focus most of its resources on accurate coordinate prediction. The crops in our pipeline can also be produced using the face landmarks from the previous frame, and the face detector is only activated to delocalize the face when the landmark model is unable to do so.

3.5.5. Face detection

You can find faces in images and videos with the MediaPipe Face Detector job. This assignment can be used to identify faces and facial characteristics within a frame. A machine learning (ML) model is used for this task, and it can process either a single image or a stream of images continuously. The task generates face locations as well as the left eye, right eye, nose tip, mouth, left eye region, and right eye region facial important points.[45]

The following processes are involved in MediaPipe face detection.

3.5.5.1. Pre-processing

The input frame of an image or video is preprocessed to improve its quality and prepare it for face detection like image rotation, resizing, normalization, and color space conversion.

3.5.5.2. Face detection model

A deep learning model is used by MediaPipe to find faces in the preprocessed image. To understand the patterns and features that separate faces from other objects, the model is often trained on a sizable dataset of annotated photos. We are using Blaze Face (short-range) model. It is a compact model for identifying individual or group faces in selfie-like pictures taken with a smartphone or webcam. The form is tailored for close-up photographs captured by front-facing phone cameras. The Single Shot Detector (SSD) convolutional network approach is used in the model architecture along with a unique encoder.

3.5.5.3. Anchor generation

MediaPipe makes use of a notion known as anchors or priors to enable detection at various scales and places. In different sizes and aspect ratios, anchors are predetermined bounding boxes that represent possible face locations. The following stages use these anchors as reference points.

3.5.5.4. Landmark detection

MediaPipe offers landmark detection, which entails locating essential face features like the eyes, nose, and mouth. Additional features like face tracking and facial expression analysis are made possible by this phase.

In the Face Landmark Model, we are using 4 landmark points for our application. After detection of these facial landmarks, they can be extracted. To access each facial point from the facial landmark detection key points data the following numbers are used.[46]

Left Eye	landmark [0]
Right Eye	landmark [1]
Nose	landmark [2]
Mouth	landmark [3]

After the extraction of these facial points, they are localized. For the localization of X and Y coordinates (x, y) of each point is shape function of NumPy is used to extract X coordinate using “0” and Y coordinate using “1”.

3.5.6. Depth perception

The ability to visually perceive the world in three dimensions (3D) and to calculate the separation/depth of an object from the source is known as depth perception. The input to our brains is in two dimensions (2D), even though the world we see is three dimensional. This is because the image created on the human retina is only two dimensions.[59] But we can still see the world in three dimensions. The outcome of human evolution is the capacity of our brains to execute depth perception. It provides information about the depth of each thing, or more precisely, the relative proximity of each object to our eyes. Although still challenging, measuring distance in relation to a camera is essential to enabling innovative applications like autonomous driving, 3D scene reconstruction, and augmented reality.[61] In robotics, depth is a crucial requirement for carrying out a variety of activities like sensing, navigation, and planning.

3.5.6.1. Monocular depth estimation

Monocular depth estimation is a task to predict a pixel-wise depth map from a single image to understand the 3D geometry of a scene. In computer vision it involves predicting the depth information of a scene from a single image. In other words, it is the process of estimating the distance of objects in a scene from a single camera viewpoint.[60]
Monocular depth estimation, which infers the 3D information of a scene without the use of extra equipment, has grown in importance as a research area because only one camera is typically used in applications [50]. In numerous applications, such as 2D to 3D image/video conversion, augmented reality, autonomous driving, surveillance, and 3D CAD model development, the distance from a scene point to the camera gives crucial information. It is a difficult process since the model must comprehend the intricate connections between scene objects and the related depth data, which can be influenced by elements like texture, occlusion, and illumination.

3.5.6.2. Pinhole camera model

The pinhole camera model explains the mathematical connection between a point's coordinates in three dimensions and its projection onto the image plane of a perfect pinhole camera. [62]

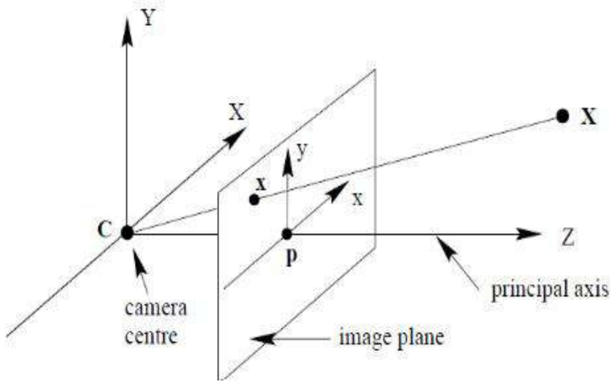


Figure 41. Pinhole camera model (Source: https://en.wikipedia.org)

There are no lenses used to concentrate light, and the camera's aperture is stated as a point. The model excludes effects like geometric distortions or blurring of out-of-focus objects brought on by lenses and apertures with fixed sizes. Additionally, it ignores the fact that most real-world cameras can only capture discrete picture coordinates. This means that the mapping from a 3D scene to a 2D image using the pinhole camera model can only be done at the first order. In general, when lens distortion effects rise, its validity diminishes from the center to the edges of the image depending on the camera's quality. [63]

Monocular depth estimation using the pinhole camera model involves utilizing the geometry and properties of a pinhole camera to infer depth information from a single image. The pinhole camera model approximates the behavior of a real-world camera with a small aperture (the pinhole) and no lens distortion.

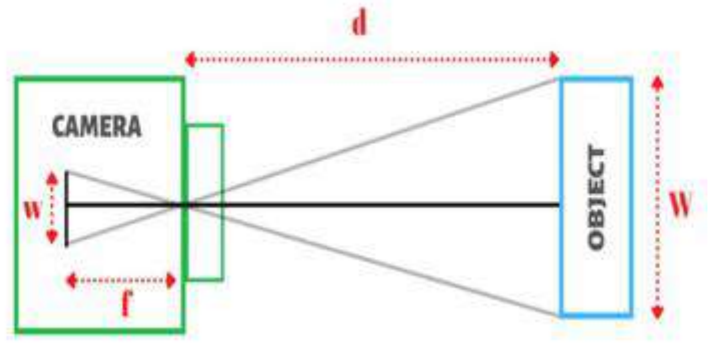


Figure 42. Pinhole camera models relation with real world camera (Source: https://en.wikipedia.org/wiki/Pinhole_camera_model)

Where W is the distance between both eyes in centimeters, "w" is the distance between eyes in pixel on the image plane, "f" is the distance between the optical center of the lens and the camera sensor, where the light information is recorded, and "d" refers to the distance between the object being photographed and the camera's image plane.

3.5.6.3. Triangulation

In a pinhole camera model, triangulation is the process of measuring a point's projections onto various two-dimensional (2D) image planes to determine its three-dimensional (3D) location in space. The triangulation law in the pinhole camera model is based on the principle of similar triangles. It asserts that if two image points and their matching camera projection rays are known, the 3D point that project onto those image points lies at the junction of the two rays.

Using the similar triangles concept.

$$\frac{\left(\frac{W}{2}\right)}{f} = \frac{\left(\frac{W}{2}\right)}{d-f}$$

$$\frac{w}{f} = \frac{W}{(d-f)}$$

$$(d-f) \times w = f \times W$$

$$d-f = \frac{f \times W}{w}$$

$$d = (f \times W)/w + 1$$

$$d = f \times (W/w + 1)$$

The value of depth or z-coordinate is equal to the depth plus focal length in that direction.

$$Z - coordinate \ = \ d \ + \ f$$

The units of the parameters used are in centimeters and pixels. To get the final desired units of focal length we need to perform simplification.

$$depth \ = \frac{(f \times W)}{w}$$

$$depth \ = pixel \times \frac{centimeters}{pixel}$$

$$depth \ = \ \ centimeters$$

The units of depth will be in centimeters. As the face is moved towards the screen the focal length is reduced and "w" value is increased on the image plane and vice versa.

3.5.6.4. Calculations

Taking one person at a time helped in knowing the actual height of the object whose depth estimation is required. The distance between the eyes of the human is fixed. Taking the W of the person to be experimented.

$$W = \ \ 2.51 \, inches \times 2.54$$

$$W = 6.985 \, centimeters$$

$$W = 7 \, centimeters$$

The average distance between the eyes of humans globally is also 7 centimeters. To calculate the value of focal length in pixel and table of comparison of physically measured depth value against the pixel values of distance between the eyes on the image plane by "w".

	"d" (centimeters)	"w" (pixels)
1	25.5	92
2	20	119
3	30	77
4	35	65
5	15	157
Average		
	63.754	102

Table 3: Relation of distance with pixels

The formula for calculation of focal length for depth estimation using pinhole camera model is as follows.

$$f = (w \times d)/W$$
$$f = (102 \times 63.754)/6.985$$
$$f = 930.98$$

The units of the focal length using this process will be as follows.

$$f = (centimeters \times pixel)/centimeters$$
$$f = 930.98\, pixels$$

For the calculation of depth “W” and “f” values become constant in pinhole camera model. The value of depth is estimated from the value of distance between the eyes on image plane “w”. As the person moves forward the distance “w” is increased on image plane and hence the depth value “d” is decreased and vice versa.

3.5.6.5. Camera calibration

Camera calibration is a method used to find the intrinsic and extrinsic properties of a camera to precisely measure and interpret the relationship between the 3D world and the 2D image acquired by the camera.[48]

The two types of parameters are:

➔ Intrinsic parameters: These parameters of the camera/lens system. E.g., focal length, optical center, and radial distortion coefficients of the lens.

➔ Extrinsic parameters: This refers to the orientation (rotation and translation) of the camera with respect to some world coordinate system.

Using a set of known 3D points (Xw, Yw, Zw) and their corresponding image coordinates (u, v), the calibration procedure aims to discover the 3×3 matrix K, the 3×3 rotation matrix {R}, and the 3×1 translation vector {t}. The camera is said to be calibrated when we obtain the values of the intrinsic and extrinsic parameters.

3.5.6.6. Calibration pattern

Before starting any calculations, firstly we need to prepare our data for further analysis. It uses a checkerboard image pattern and recognizes its corners. I photographed the chessboard from different angles. Using 12 different images can make the unique estimation more robust.[49] First, to find chessboard patterns, use the “cv2.findChessboardCorners()” function. You also need to specify what type of pattern you are looking for. In our example, it's an 8x11 rectangular grid. The size of each black and white square within the rectangular pattern is 15 millimeters.[50] This function returns the vertices for each frame. Once you find them, refine them with “cv2.cornerSubPix()” and draw the pattern with “cv2.drawChessboardCorners()”.

We can compute the intrinsic camera matrix, rotation matrix, translation vector, and distortion vector once we have computed the image points and object points.
The “cv2.calibrateCamera()” function can be used to accomplish this. With 8×11 checkerboard size 15mm

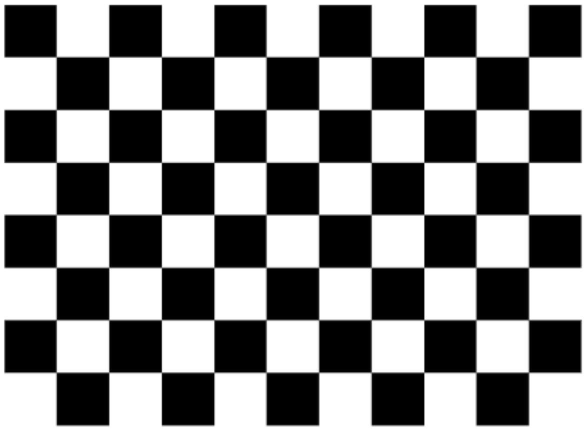


Figure 43. Calibration Pattern checkerboard (Source: Calibration Checkerboard Collection | Mark Hedley Jones)

“cv2.getOptimalNewCameraMatrix()”, we can additionally improve the camera matrix based on a free scaling parameter. Reprojection error can be used to validate the estimates of the parameters found. Compute the average Euclidean distance between pixels and projected pixels using the Eigen matrix. Accounting for quirks, distortions, rotations, and translations, you can convert object points to image points using “cv2.projectPoints()”.

3.5.6.7. Calibration matrix

A camera is mapping between the 3D world and the 2D image. To project the 3D world information onto 2D image plane we need to go through different frames [51]. Starting from 3D world frame to 3D camera frame, from 3D camera frame to pixel frame and frame pixel to 2D image plane.

$$x = PX$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Figure 44: Relation of Homogenous image/2D image point (matrix on left hand side) with homogenous world point (Right one on the right side) by camera matrix (Left one on the right side)

The camera matrix is made up of intrinsic and extrinsic parameters. The intrinsic parameters include information related to the focal length of the camera and principal focus whereas the extrinsic parameters include the information related to the rotation matrix which is 3×3 and a translation vector which is 3×1 [52].

$$P = K[R \mid t]$$

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc:c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{array} \right]$$

Figure 45: Equation having intrinsic parameters (Left matrix) and extrinsic parameters (Right matrix)

Extrinsic parameters consist of rotation matrix and translation vector.

$$\mathbf{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

Figure 46: 3D Rotation (R matrix) and 3D translation (t matrix)

In our case all we need to find in the intrinsic parameters of camera to proceed further with our calculations. So, dissecting the intrinsic parameters we get the following matrix. [54] The K matrix converts camera coordinate system to pixel coordinate system in two steps.

1) Camera to image
2) Image to pixel

$$k = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

The camera matrix K we obtained after performing calibration on checkerboard images from different poses and obtaining image points and object points.

$$k = \begin{bmatrix} 995.33 & 0 & 396.34 \\ 0 & 1002.32 & 265.16 \\ 0 & 0 & 1 \end{bmatrix}$$

Where:

fx and fy are the focal lengths of the camera in the x and y directions, respectively. Determines the scaling factor between 3D-world coordinates and 2D image coordinates.

The camera's optical centers, cx and cy, define the primary point where the optical axis of the camera and the image plane cross. They depict any shifts or distortions caused by the lens and take into consideration the offset of the image center from the top-left corner [53]. Typically, the matrix's values are given in terms of pixels. Convention dictates that the central point should be roughly in the middle of the image, therefore cx should be near to the image's width divided by 2, and cy should be close to the image's height divided by 2.

3.5.7. Image plane to 3D world plane

It is impossible to turn a 2D point into a 3D point. It can be transformed into a ray of points pointing in the direction of your imagined three-dimensional point. However, a two-view can provide a three-dimensional point utilizing an essential or fundamental matrix. If and only if you know the real height of the object, you can also determine its depth. [55]

If the height of the object is known and the object, then a 2D image point can be converted to the X and Y of the 3D world coordinates. For that conversion we need to know the focal length of the camera and height of the object.

Assumption:

We assume that pinhole camera model and that the ground plane is parallel to the image plane.

The formula to obtain X and Y of the 3D world coordinates from 2D image coordinates if the height of object and object is known.[16]

$$X = ((x - cx) \times Z)/fx$$

$$Y = ((y - cy) \times Z)/fy$$

Where X, Y, and Z are the coordinates of the 3D point in the camera coordinate system. cx and cy are the coordinates of the principal point (also known as the optical center) in the image plane. fx and fy are the focal lengths of the camera in the x and y directions, respectively. Z is the height of the object above the ground plane.

Suppose we have a camera with the following intrinsic parameters:

$$focal\ length\ fx\ =\ 500\ pixels$$

$$focal\ length\ fy =\ 500\ pixels$$

$$principal\ point\ cx = 320\ pixels$$

$$principal\ point\ cy = 240\ pixels$$

We also have an image of an object with the following pixel coordinates:

$$x = 400\ pixels$$

$$y = 300\ pixels$$

We know that the object is at a height of Z = 2 meters above the ground plane.

Using the formula [56], we can calculate the 3D coordinates of the object as follows:

$$X = ((x - cx) \times Z)/fx$$

$$X = ((400 - 320) \times 2)/500$$

$$X = 0.64\ meters$$

$$Y = ((y - cy) \times Z)/fy$$

$$Y = ((300 - 240) \times 2)/500$$

$$Y = 0.48\ meters$$

$$Z = 2\ meters$$

Therefore, the 3D coordinates of the object in the camera coordinate system are (0.64, 0.48, 2.0) meters.

Situation:

The camera is attached to the end effector of robot and moves with the end effector. There becomes a situation where the camera is very close to the human face and the human face starts to fall out of the field of view of camera. [57] The face is not in the field of view of the camera and the detection of human face and tracking is lost. As there is no data for the robot in this situation to make the decision and there is a possibility that robot might hit the patient.[58] To cater to this issue, we are using a laser sensor. The level 1 laser distance sensor, which is biomedically safe to use in human computer interactions. The depth will be shifted to laser distance sensor when face is not in the field of view of camera.

3.5.8. Laser distance sensor

VL53L0X is a laser ranging sensor. Using VL53L0X, level 1 laser distance sensor with 2D camera used for monocular depth estimation, the system became more reliable. The key objectives achieved are as follows.

→ The system’s safety increased and risk of robot hitting the patient is reduced by incorporating VL53L0X with 2D camera.

→ The precision in data increased as the sensor can measure value up to 1mm of distance.

→The situation in which patient's face is not in the field of view of camera is also tackled

3.6. Control and Electronics

For controls, Raspberry Pi 3B is used for the desired application.

3.6.1. Actuators control

In this project we used 2 types of motors which are stepper and servo motor respectively. Stepper motor were used in order to meet high torque requirement coupled with gear ratio mechanism and better precision keeping the system open loop generally but closed loop from software domain.

3.6.1.1. Stepper motors

3.6.1.1.1. Configuration

As only Bi-Polar stepper motors were used in the robotic arm hence all the stepper drivers were tb6560 which have two h bridges to control the two coils of the motor based on given set configuration and pulse input from the microcontroller. The tb6560 stepper driver allows multiple section configurations which includes current limitation, micro stepping up to 1/16, stop current and decay setting. Tables shows the configuration which was used.

Functionality	Configuration used
Max current	3A
Micro stepping	Full step ($\mathbf{1.8^0}$)
Stopping current	25 %
Decay	50 %

Table 4: Tb6560 Stepper driver configuration

3.6.1.1.2. Angle calculations

Stepper motors move single step as per driver setting on a pulse from the controller. In our case as mentioned in Table full stepping configuration was used hence each pulse from the controller results in 1.8^0 of rotation of the motor rotor. Also the motors are connected to the link via gear mechanism based on pulley and belts to increase the torque which in parallel decrease the angle achieved per step of the motor. Calculation for steps to resultant angles are as follows:

- Joint 1 relation

  Link 1 has gear ratio as follows:

$$gear\ ratio = 100{:}\,10$$

  Which means 100 revolutions of stepper motor will make 1 revolution of link.

$$100\ rev = 3600^0$$

$$1\ rev = \frac{3600^0}{100}$$

$$1\ rev = 36^0$$

  As step angle is 1.8^0 then,

$$360^0 = 36^0$$

$$200\ steps = 36^0$$

$$1\ step = \frac{36^0}{200}$$

  For given angle X:

$$X^0 \approx \frac{200}{36}\ steps$$

- Joint 2 relation

  Gear ratio for link 2 is 60:10.

- Joint 3 relation

  Gear ratio for link 3 is 55:10. With gear box of additional 4:1.

- Joint 4 relation

  There is no gear ratio as the stepper motor shaft is directly coupled with the next link.

- Joint 5 relation

  Gear ratio for link 5 is 30:10

As the sixth joint has servo motor which can be positioned controlled directly from the code it has not given.

3.6.1.2. Servo motor

The end-effector of our system uses servo motor. There are two reasons to use this motor one is that because of plastic body servo motor is lights than stepper in this case which requires lesser load on the previous link stepper motor. Secondly servo motor provides closed loop feedback to ensure the completion of the given task.

3.6.2. Actuation feedback

Overall assistive feeding system is closed loop system because of runtime feedback from camera and laser sensor but in case of actuation the system is open loop and the fact that stepper motors are used a software based closed loop system is generated.

3.6.2.1. Software base closed loop

For the feedback of stepper motors no encoder or mechanical feedback mechanism is used. Stepper motor actuate step wise based on control signal and using this functionality we implemented software based closed loop.

Stepper motors are being controlled as instances of class of stepper motor and in every instance/object of the class it creates a private variable of angle which stores the value of angle which is passed to motor to move and updated accordingly.

There is a limitation to this design, as when the code ends and restarted it will assume the current position of the motors to be zero states and this issue is resolved using hard home feature which will be discussed later. Secondly if the motors instances or object go out of scope and gets deleted this will also generate new instances every time the scopes changes. This limitation is resolved by ensuring that motor instances should be generated in the main function and passed to the required function as per need this also improves the accessibility of changing motor parameters from the main function only.

3.6.2.2. Servo feedback

The feedback system in the MG995 servo motor works involving a potentiometer or rotary encoder. The feedback mechanism measures the rotation angle of the motor as it rotates and sends this information back to the control system. This allows the control system to compare the desired position to the actual position and make the necessary adjustments to accurately achieve the desired movement.

3.6.2.3. Hard home feedback

As the actuation system is open loop from hardware side and closed loop only from software side this introduces a limitation that when the robot start from an arbitrary state then it will assume the current state as the home position.

To solve this issue we implemented limit switches at every joint except the end-effector as that uses closed loop servo motor. Using this whenever the robot restarts, it first follow a default code to move until it reaches the limit switches after that it move to a soft home condition.

3.6.3. Controller

Raspberry pi 3B was used because we required a controller which has good memory and could perform computer vision task runtime. Although there were other better options like Jetson Nano but cost was second major factor of decision.

3.6.3.1. OS

64 bit Linux operating system is installed on the controller. One of the requirement was the compatibility of the operating system and computer vision libraries specifically Mediapipe and its specific frameworks.

3.6.3.2. Libraries

To implement computer vision section and do complex matrix calculations for forward and inverse kinematics and implementing graphical user interface we used multiple libraries as mentioned in table.

Library	Application
Mediapipe	Facial landmark detection
Numpy	Matrix calculations
Scipy	Numerical solution of two joint angles
Tkinter	GUI

Table 5: Libraries setup on controller for system

3.6.3.3. Virtual environment

Raspberry pi operating system has couple of virtual environments for programming. In our case as the RAM of the system was limited to 1 Gigabytes and we have to run computer vision algorithm and calculate numerical solutions apart from graphical user interface hence we used these virtual environment for coding.

In this project majorly “Thonny” was used for the purpose of programming and testing the codes on the controller but there was a major limitation in using these virtual environments which is the fact that virtual environment for the above OS setup and configurations can only run python3.8 or above while the computer vision algorithm uses mediapipe library which is compatible with python version less than 3.8. This issue was solved using the OS terminal and downgrading its python version to python3.7.

From above solution the codes still could not run on the virtual environment hence a main file is generated and accessed through the terminal and run using the downgraded python which starts the GUI and rest of the interface is shifted to the GUI.

3.6.3.4. Power

For the complete embedded and control system one main power supply is used to power are the system which is rated 24 V 20 A.

The power supply directly powers the motor drivers as all the stepper motors are working on 24V. Secondly for the remaining lower voltage system buck converter is used and 24V is applied to the buck convertor.

3.6.3.5. Camera power

Initially camera was directly connected to raspberry pi via USB cable. But when the computer vision algorithm starts as it require high processing power the raspberry pi gets heated and the power consumption of stereo camera increased that which ultimately resulted in shutdowns.

To solve the aforementioned problem, we separated the power lines and data line of the USB cable and connected the power lines with the buck convertor and data lines to the USB data line port of the raspberry pi as ground was common data transfer was successful and issue was resolved.

3.6.4. Graphical User Interface

To facilitate the user and the programmer as well we connected a touch screen with the raspberry pi which communicates using I2C protocol. We implemented GUI using tkinter library in python. The follow of GUI is shown in the figure below:

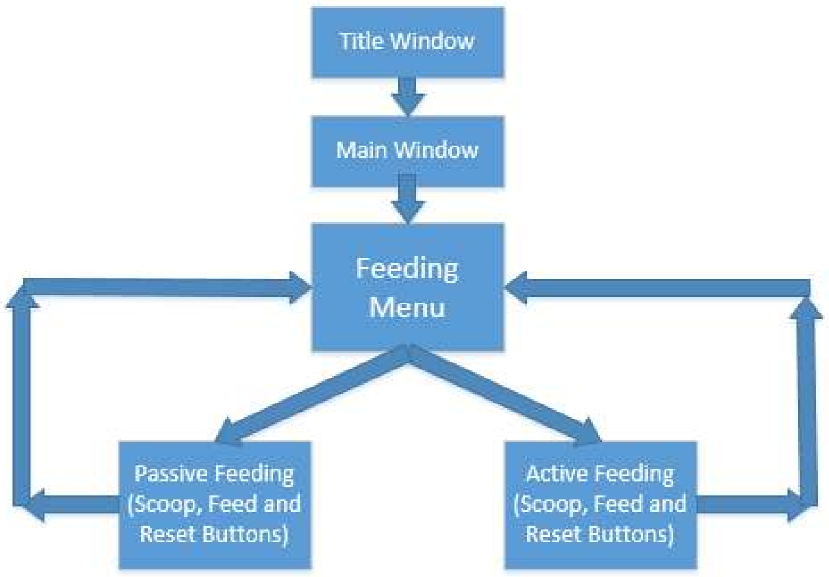

Title Window
Main Window
Feeding Menu
Passive Feeding (Scoop, Feed and Reset Buttons)
Active Feeding (Scoop, Feed and Reset Buttons)

Figure 47: Flowchart of GUI for assistive feeding system

3.6.5. Wireless communication

For the remote access for both programmer and the user we have used wireless communication to update the firmware and remotely control the system.

3.6.5.1. VNC

Virtual Network computing is a software that allows to remotely control the desktop interface of the raspberry pi over the Wi-Fi using any mobile device or another computer. As it gives access to desktop interface it is a limitation of VNC that the Raspberry pi OS should be setup with a GUI. In this project VNC is used for the user or guardian to get the remote access and monitoring option for the robotic system.

VNC is setup from pi configuration section and enabling VNC. A VNC account is needed to connect the host and the guest together. Once the account is made in the host which is pi in our case then we need to get the host name and IP address apart from the VNC account.

In any mobile device or computer which we want to connect from we have to setup VNC viewer app and add the login details. When everything is ready enter the IP of the host device if the device is available a window will popup which requires the guest to enter the host device password. Finally the GUI will be visible and can be tinkered with.

3.6.5.2. SSH

Secure Shell (SSH) is a network protocol that allows the guest to securely access the remote computer. This is often used to access the raspberry pi from the same network as of the host.

Unlike VNC which give remote access to GUI, SSH give access to the terminal of the raspberry pi. In this project this was an advantage as computer vision module works on degraded python on the terminal which was discussed in detail previously which can be done using SSH as it give the access of the terminal. Hence, SSH in this project is used for remote access for the programmer for updates and alteration in the design system.

SSH is also setup from raspberry pi configuration. For the guest only host name and password are required to access via SSH protocol. Guest can setup remote access extension in visual studio code and turn on the extension then setting up SSH configuration file. Once configuration file is ready initialize the SSH protocol and it will search for the availability and prompt for the host OS which in our case is Linux and then finally password of the host and terminal access is granted if the above credentials are correct.

3.6.6. Forward kinematics

To solve the kinematic problem of this 6 DOF robotic arm we used Modified Denavit-Hartenberg Parameter. For the sake of kinematic solution and resolving the joint angles from required target pose matrix we reduced the number of degrees of freedom we will solve the system for, to make the system less complex. The final kinematic solution is given using 5 degrees of freedom after fixing the end-effector servo.

3.6.6.1. Frame attachment

As mentioned previously we used Modified DH parameters for kinematic solution hence the frame attachment was done in such a way that frame of a given link (i.e. link i) was assigned to the end which is near to the next link (i.e. i+1). Frame attachment was done using the following set rules

1. Identify the joint axes and imagine (or draw) infinite lines along them. For steps 2 through 5 below, consider two of these neighboring lines (at axes i and i + 1).

2. Identify the common perpendicular between them, or point of intersection. At the point of intersection, or at the point where the common perpendicular meets the ith axis, assign the link-frame origin.

3. Assign the Zi axis pointing along the ith joint axis.

4. Assign the Xi axis pointing along the common perpendicular, or, if the axes intersect, assign Xi to be normal to the plane containing the two axes.

5. Assign the Yi axis to complete a right-hand coordinate system.

6. Assign {0} to match {1} when the first joint variable is zero. For {N}, choose an origin location and XN direction freely, but generally so as to cause as many linkage parameters as possible to become zero.

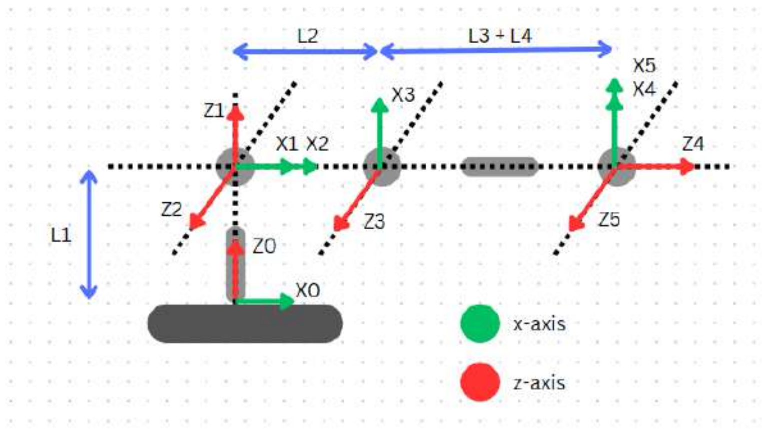


Figure 48: Frame attachment

3.6.6.2. DH parameters table

The Denavit-Hartenberg parameter tables consist of four variables. Two variables represent link length and link twist angle, whereas the other two represents joint angle link offset. The D-H parameter table template for a robotic arm with four reference frames. After the frame attachment, all four parameters for each link were devised based on following conditions:

$$a_i = the\ distance\ from\ \hat{Z}_i\ to\ \hat{Z}_{i+1}\ measured\ along\ \hat{X}_i$$

$$\alpha_i = the\ angle\ from\ \hat{Z}_i to\ \hat{Z}_{i+1}\ measured\ about\ \hat{X}_i$$

$$d_i = the\ distance\ from\ \hat{X}_{i-1}\ to\ \hat{X}_i\ measured\ along\ \hat{Z}_i$$

$$\theta_i = the\ angle\ from\ \hat{X}_{i-1}\ to\ \hat{X}_i\ measured\ about\ \hat{Z}_i$$

Using above equations the DH parameters were measured as shown in the table.

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	L1	θ_1
2	90	0	0	θ_2
3	0	L2	0	θ_3
4	90	0	L3 + L4 +L5	θ_4
5	-90	0	0	θ_5

Table 6: DH parameters for the 5DOF system

Where link lengths are as follows:

$$L1 = 0.221\,m$$

$$L2 = 0.223\,m$$

$$L3 + L4 + L5 = 0.0\,m$$

3.6.6.3. Modified DH formula

The original Denavit-Hartenberg (DH) parameters used in robotics and kinematics have been extended by modified DH parameters. The improved DH parameters fix several issues and offer a more adaptable way to describe the kinematics of robotic systems [65]. It includes the following modifications:

The offset parameter (d) specifies the angle along the common normal between the z-axes of successive coordinate frames. This offset was calculated using the original DH values along the current frame's z-axis. The offset is instead measured along the x-axis of the current frame in the adjusted DH settings. With this adjustment, there is more freedom in expressing intricate robot geometries.

The angle between the z-axes of succeeding coordinate frames is represented by the twist parameter. This twist was calculated using the original DH parameters and was centered on the current frame's z-axis. The twist is measured about the x-axis of the preceding frame in the adjusted DH parameters. This adjustment enhances the depiction of joint rotations and accommodates robots with non-revolute joints.

The length of the common perpendicular between the z-axes of succeeding coordinate frames is represented by the link length parameter (a). This parameter is unmodified from the original DH parameters and continues to indicate the same physical distance.

The rotation about the shared normal between successive coordinate frames is represented by the joint angle parameter. This value indicates the same joint angle as in the original DH parameters and is unaltered in the amended DH parameters.

Equation 5 was used to get the transformation matrix of every configuration required.

$$\boldsymbol{T_i^{i-1}} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{5})$$

3.6.6.4. Equations

To get the Base to Wrist transformation matrix we first calculated transformation matrices of every link with respect to its previous link.

Substituting the values from DH parameter table to the equation 5 returns the following matrices:

$$\boldsymbol{T_1^0} = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{6})$$

$$\boldsymbol{T_2^1} = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{7})$$

$$\boldsymbol{T_3^2} = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & L2 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{8})$$

$$\boldsymbol{T_4^3} = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & 0 \\ 0 & 0 & -1 & -(L3 + L4 + L5) \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{9})$$

$$\boldsymbol{T_5^4} = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s\theta_5 & -c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{10})$$

For $\boldsymbol{T_5^0}$ transformation matrix, following property of transformation matrices was used:

$$\boldsymbol{T_C^A} = \boldsymbol{T_B^A} \times \boldsymbol{T_C^B} \quad (\boldsymbol{equ}: \mathbf{11})$$

We used MATLAB to multiply the matrices based on equation 11 to get the final matrix which came out to be:

$$\boldsymbol{T_5^0} = \begin{bmatrix} r1 & r2 & r3 & p1 \\ r4 & r5 & r6 & p2 \\ r7 & r8 & r9 & p3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{12})$$

Where r1 to r9 are equations of rotation matrix and p1 to p3 are equations for linear transformation which are as follows:

$$\boldsymbol{r1} = \cos(Q5)*(\cos(Q4)*(\cos(Q1)*\cos(Q2)*\cos(Q3)) - \cos(Q1)*\sin(Q2) * \sin(Q3)) + \sin(Q1)*\sin(Q4)) - \sin(Q5)*(\cos(Q1)*\cos(Q2) * \sin(Q3) + \cos(Q1)*\cos(Q3)*\sin(Q2)) \quad (\boldsymbol{equ}{:}\ 13)$$

$$\boldsymbol{r2} = -\cos(Q5)*\big(\cos(Q1)*\cos(Q2)*\sin(Q3)+\cos(Q1)*\cos(Q3)*\sin(Q2)\big) - \sin(Q5) * \Big(\cos(Q4) * \big(\cos(Q1)*\cos(Q2)*\cos(Q3)-\cos(Q1)*\sin(Q2)*\sin(Q3)\big) + \sin(Q1)*\sin(Q4)\Big) \quad (\boldsymbol{equ}{:}\ 14)$$

$$\boldsymbol{r3} = \cos(Q4)*\sin(Q1) - \sin(Q4) * \big(\cos(Q1)*\cos(Q2)*\cos(Q3) - \cos(Q1)*\sin(Q2) * \sin(Q3)\big) \quad (\boldsymbol{equ}{:}\ 15)$$

$$\boldsymbol{p1} = \cos(Q1)*\big(L3*\sin(Q2 + Q3) + L2*\cos(Q2)\big) \quad (\boldsymbol{equ}{:}\ 16)$$

$$\boldsymbol{r4} = \cos(Q5) * \Big(\cos(Q4) * \big(\cos(Q2)*\cos(Q3)*\sin(Q1) - \sin(Q1)*\sin(Q2)*\sin(Q3)\big) - \cos(Q1)*\sin(Q4)\Big) - \sin(Q5) * \big(\cos(Q2)*\sin(Q1)*\sin(Q3) + \cos(Q3)*\sin(Q1) * \sin(Q2)\big) \quad (\boldsymbol{equ}{:}\ 17)$$

$$\boldsymbol{r5} = -\sin(Q5) * \Big(\cos(Q4) * \big(\cos(Q2)*\cos(Q3)*\sin(Q1) - \sin(Q1)*\sin(Q2)*\sin(Q3)\big) - \cos(Q1)*\sin(Q4)\Big) - \cos(Q5) * \big(\cos(Q2)*\sin(Q1)*\sin(Q3) + \cos(Q3)*\sin(Q1) * \sin(Q2)\big) \quad (\boldsymbol{equ}{:}\ 18)$$

$$\boldsymbol{r6} = -\sin(Q4)*\big(\cos(Q2)*\cos(Q3)*\sin(Q1) - \sin(Q1)*\sin(Q2)*\sin(Q3)\big) - \cos(Q1)*\cos(Q4) \quad (\boldsymbol{equ}{:}\ 19)$$

$$\boldsymbol{p2} = \mathbf{sin(Q1) * (L3 * sin(Q2 + Q3) + L2 * cos(Q2))} \quad (\boldsymbol{equ}: \mathbf{20})$$

$$\boldsymbol{r7} = \mathbf{cos(Q2 + Q3) * sin(Q5) + sin(Q2 + Q3) * cos(Q4) * cos(Q5)} \quad (\boldsymbol{equ}: \mathbf{21})$$

$$\boldsymbol{r8} = \mathbf{cos(Q2 + Q3) * cos(Q5) - sin(Q2 + Q3) * cos(Q4) * sin(Q5)} \quad (\boldsymbol{equ}: \mathbf{22})$$

$$\boldsymbol{r9} = \mathbf{-sin(Q2 + Q3) * sin(Q4)} \quad (\boldsymbol{equ}: \mathbf{23})$$

$$\boldsymbol{p3} = \mathbf{L1 - L3 * cos(Q2 + Q3) + L2 * sin(Q2)} \quad (\boldsymbol{equ}: \mathbf{24})$$

3.6.7. Target pose matrix

The desired location and orientation of a robot's end-effector or tool in the robot's coordinate system are represented by the target pose matrix in robotics. Translation and rotation information are commonly combined in a 4x4 homogeneous transformation matrix. With the use of the target pose matrix, the robot can precisely reach and control items in its surroundings to achieve a certain objective.

For target pose matrix in case of serial robotic manipulator we first require base to wrist transformation $\boldsymbol{T_W^B}$, then we can use Cartesian transforms to calculate the target to base matrix $\boldsymbol{T_T^B}$.

3.6.7.1. Base to wrist matrix

The base-wrist matrix, used in robotics, is a transformation matrix that explains the connection between a robot manipulator's base frame and wrist frame [66]. It symbolizes the total transformation from the base coordinate system of the robot to the end-effector or wrist coordinate system. A 4x4 homogeneous transformation matrix called the base-wrist matrix combines rotation and translation. To ascertain the position and orientation of the wrist frame with respect to the base frame, it takes into account the DH parameters, joint angles, and link lengths of the robot's kinematic chain.

Forward kinematics, or the process of identifying the location and orientation of the end-effector given the joint angles, depends on the base-wrist matrix. Inverse kinematics, which entails determining the joint angles that lead to a desired end-effector position and orientation, also use it.

In our case $\boldsymbol{T_W^B}$, is given as $\boldsymbol{T_5^0}$ which is calculated in equation 12.

3.6.7.2. Wrist to camera matrix

In our project we mounted the camera on the end-effector rather than setting up to some fixed location. This adds additional feature that this system can be deployed at any location without camera position calculation which are generally required in the previous case.

For sake of simplicity in the design and calculations the camera was mounted with just linear displacements hence the relative position can be calculated without incorporating the rotation matrix. This can be given as a translation vector between camera and the wrist.

$$\boldsymbol{P_W^C} = \begin{bmatrix} \frac{11.93}{1000} \\ \frac{30}{1000} \\ \frac{31}{1000} \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{25})$$

Or the transformation matrix can be made keeping the rotation matric as unity which is given in equation 26.

$$\boldsymbol{T_W^C} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \frac{11.93}{1000} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \frac{30}{1000} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \frac{31}{1000} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{26})$$

3.6.7.3. World to camera matrix

In this project as previously mentioned that 2D image is taken from the camera and computer vision algorithm detects the $\boldsymbol{X_W}$ and $\boldsymbol{Y_w}$ coordinates of the desired location then using a fix value of eye distance and pinhole camera model $\boldsymbol{Z_w}$ is estimated. These three are linear parameters of the target from the camera and the rotation parameters like pitch, yaw and roll are discarded because while feeding the spoon needs to be upright and the person needs to be in the same pose but there might be linear transformation which the robot will cover to feed. Hence keeping the assumption the world to camera matric can be given as:

$$\boldsymbol{T_T^C} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \boldsymbol{X_w} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \boldsymbol{Y_w} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \boldsymbol{Z_w} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad (\boldsymbol{equ}: \mathbf{27})$$

3.6.7.4. World to base matrix

Finally the world to base matrix is calculated which give the required pose to the robot. This can be calculated using the Cartesian transformation and given as:

$$\boldsymbol{T_T^B} = \boldsymbol{T_C^W}^{-\mathbf{1}} \times \boldsymbol{T_B^C}$$

3.6.8. Inverse kinematics

To compute the joint angles or joint locations necessary to obtain a particular end-effector pose (position and orientation), robotics uses inverse kinematics. It is identifying the joint variables that adhere to the mechanical limitations placed on the robot as well as the geometric constraints. There are several methods to perform inverse kinematics, in this project we used three of them which are discussed below:

3.6.8.1. Algebraic Method

The inverse kinematics problem is majorly solved using the algebraic method, which entails putting up and resolving a system of algebraic equations. It is frequently applied to robots with straightforward kinematic designs that enable the direct computation of joint variables. In our project we used the equations derived from the forward kinematics and a target matrix to calculate the unknown joint angles.

Using equation 16 and 20:

$$p1 = \cos(\mathrm{Q1}) * \big(\mathrm{L3} * \sin(\mathrm{Q2} + \mathrm{Q3}) + \mathrm{L2} * \cos(\mathrm{Q2})\big) \ (\boldsymbol{equ}: \mathbf{16})$$

$$p2 = \sin(\mathrm{Q1}) * \big(\mathrm{L3} * \sin(\mathrm{Q2} + \mathrm{Q3}) + \mathrm{L2} * \cos(\mathrm{Q2})\big) \ (\boldsymbol{equ}: \mathbf{20})$$

Dividing equation 20 by 16:

$$\frac{p2}{p1} = \frac{sin(Q1)}{cos(Q1)}$$

Simplifying equation we get:

$$\tan(Q1) = \frac{p2}{p1}$$

Taking Q1 on one side:

$$\mathrm{Q1} = \arctan2\left(\frac{p2}{p1}\right)$$

For Q2 and Q3 calculations algebraic method failed. These will be discussed in the next section. For Q4 we used equation 23:

$$r9 = -\sin(\mathrm{Q2}\ +\ \mathrm{Q3}) * \sin(\mathrm{Q4}) \quad (\boldsymbol{equ}\text{: }\mathbf{23})$$

Solving for Q4:

$$\sin(Q4) = \frac{r9}{-\sin(Q2 + Q3)}$$

$$Q4 = \arcsin\left(\frac{r9}{-\sin(Q2 + Q3)}\right)\ (\boldsymbol{equ}\text{: })$$

For Q5 we used equation 21 and 22:

$$r7 = \cos(\mathrm{Q2}\ +\ \mathrm{Q3}) * \sin(\mathrm{Q5}) + \sin(\mathrm{Q2}\ +\ \mathrm{Q3}) * \cos(\mathrm{Q4}) * \cos(\mathrm{Q5}) \ \ (\boldsymbol{equ}\text{: }\mathbf{21})$$

$$r8 = \cos(\mathrm{Q2}\ +\ \mathrm{Q3}) * \cos(\mathrm{Q5}) - \sin(\mathrm{Q2}\ +\ \mathrm{Q3}) * \cos(\mathrm{Q4}) * \sin(\mathrm{Q5}) \quad (\boldsymbol{equ}\text{: }\mathbf{22})$$

Let $\cos(Q2 + Q3) = a$ amd $\sin(Q2 + Q3) \times \cos(Q4) = b$ and substitute in the above equations we get,

$$r7 = a \times \sin Q5\ + b \times \cos Q5$$

$$r8 = -b \times \sin Q5 + a \times \cos Q5$$

Now, let $\sin Q5 = X$ and $\cos Q5 = Y$ and substitute in the above equations we get,

$$r7 = a \times X + b \times Y$$

$$r8 = -b \times X + a \times Y$$

Multiply equation of r7 and r8 by a and b respectively we will get,

$$a \times r7 = a^2 \times X + a \times b \times Y$$

$$b \times r8 = -b^2 \times X + a \times b \times Y$$

Subtract r8 equation from r7 we get,

$$a \times r7 - b \times r8 = a^2 X + b^2 X$$

$$a \times r7 - b \times r8 = X(a^2 + b^2)$$

Substitute the value of X and rearrange the equation:

$$\sin Q5 = \frac{a \times r7 - b \times r8}{a^2 + b^2}$$

Separate Q5 and substitute the values of a and b, we get the equation for Q5:

$$Q5 = \arcsin(\frac{a \times r7 - b \times r8}{a^2 + b^2})$$

$$Q5 = \arcsin\left(\frac{r7 * \cos(Q2 + Q3) - r8 * \sin(Q2 + Q3) * \cos(Q4)}{(\cos(Q2 + Q3))^2 + (\sin(Q2 + Q3) * \cos(Q4))^2}\right)$$

3.6.8.2. Paul’s Method

Paul's method offers a methodical solution to the issue by variously rewriting the forward kinematics equations [67]. Creating a coordinate system for the end-effector is the first stage in Paul's technique. Normally, the axes of the end-effector are aligned with the end-effector coordinate system. The next action is to establish a coordinate system for each manipulator connection. The axes of the linkages are normally aligned with the link coordinate systems.

The forward kinematics equations for the manipulator must be written as the last step. The location and orientation of the end-effector and the axis of the links are related by the forward kinematics equations to the joint angles.

General equation to implement Paul’s method is given as:

$$(T_i^0)^{-1} \times T_W^B = T_{i+1}^i \times T_{i+2}^{i+1} \; ....$$

In our case we need to calculate the values of Q2 and Q3 which can be done taking $i = 2$ as at this stage we already have the value of Q1 from the algebraic method. Note that T_W^B in our case is named as T_T^B .Updated equation will be:

$$(T_2^0)^{-1} \times T_T^B = T_3^2 \times T_4^3 \times T_5^4$$

Using MATLAB to calculate both sides of the equations which came out to be:

$$\boldsymbol{T_3^2} \times \boldsymbol{T_4^3} \times \boldsymbol{T_5^4} = \begin{bmatrix} w1 & w2 & w3 & w4 \\ w5 & w6 & w7 & w8 \\ w9 & w10 & w11 & w12 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where equations for w1 to w12 are as follows:

$$w1 = cos(Q3) * cos(Q4) * cos(Q5) - sin(Q3) * sin(Q5)$$

$$w2 = -\cos(Q5) * \sin(Q3) - \cos(Q3) * \cos(Q4) * \sin(Q5)$$

$$w3 = -\cos(Q3) * \sin(Q4)$$

$$w4 = (223 * sin(Q3))/1000 + 221/1000$$

$$w5 = cos(Q3) * sin(Q5) + cos(Q4) * cos(Q5) * sin(Q3)$$

$$w6 = cos(Q3) * cos(Q5) - cos(Q4) * sin(Q3) * sin(Q5)$$

$$w7 = -sin(Q3) * sin(Q4)$$

$$w8 = -(223 * cos(Q3))/1000$$

$$w9 = \cos(Q5) * \sin(Q4)$$

$$w10 = -sin(Q4) * sin(Q5)$$

$$w11 = \cos(Q4)$$

$$w12 = 0$$

From the above equation we will use equations of w3, w7, w4 and w8. The left hand side equations are very long so we will only mention the corresponding four equation just for this solution:

$$(T_2^0)^{-1} \times T_T^B = \begin{bmatrix} e1 & e2 & e3 & e4 \\ e5 & e6 & e7 & e8 \\ e9 & e10 & e11 & e12 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where equations for e3, e7, e4 and e8 are as follows:

$$\begin{aligned} e3 = {} & -(R2 * R9 * cos(Q2) - R3 * R8 * cos(Q2) - R5 * R9 * sin(Q2) + R6 * R8 \\ & * sin(Q2))/(R1 * R5 * R9 * cos(Q2)\hat{}2 - R1 * R6 * R8 * cos(Q2)\hat{}2 \\ & - R2 * R4 * R9 * cos(Q2)\hat{}2 + R2 * R6 * R7 * cos(Q2)\hat{}2 + R3 * R4 \\ & * R8 * cos(Q2)\hat{}2 - R3 * R5 * R7 * cos(Q2)\hat{}2 + R1 * R5 * R9 \\ & * sin(Q2)\hat{}2 - R1 * R6 * R8 * sin(Q2)\hat{}2 - R2 * R4 * R9 \\ & * sin(Q2)\hat{}2 + R2 * R6 * R7 * sin(Q2)\hat{}2 + R3 * R4 * R8 \\ & * sin(Q2)\hat{}2 - R3 * R5 * R7 * sin(Q2)\hat{}2) \end{aligned}$$

$$\begin{aligned} e4 = {} & (96 * R2 * R9 * cos(Q2) - 96 * R3 * R8 * cos(Q2) - 96 * R5 * R9 * sin(Q2) \\ & + 96 * R6 * R8 * sin(Q2) - 625 * P3 * R2 * R6 * cos(Q2)\hat{}2 + 625 \\ & * P3 * R3 * R5 * cos(Q2)\hat{}2 + 625 * P2 * R2 * R9 * cos(Q2)\hat{}2 \\ & - 625 * P2 * R3 * R8 * cos(Q2)\hat{}2 - 625 * P1 * R5 * R9 \\ & * cos(Q2)\hat{}2 + 625 * P1 * R6 * R8 * cos(Q2)\hat{}2 - 625 * P3 * R2 \\ & * R6 * sin(Q2)\hat{}2 + 625 * P3 * R3 * R5 * sin(Q2)\hat{}2 + 625 * P2 * R2 \\ & * R9 * sin(Q2)\hat{}2 - 625 * P2 * R3 * R8 * sin(Q2)\hat{}2 - 625 * P1 * R5 \\ & * R9 * sin(Q2)\hat{}2 + 625 * P1 * R6 * R8 * sin(Q2)\hat{}2)/(625 * (R1 \\ & * R5 * R9 * cos(Q2)\hat{}2 - R1 * R6 * R8 * cos(Q2)\hat{}2 - R2 * R4 * R9 \\ & * cos(Q2)\hat{}2 + R2 * R6 * R7 * cos(Q2)\hat{}2 + R3 * R4 * R8 \\ & * cos(Q2)\hat{}2 - R3 * R5 * R7 * cos(Q2)\hat{}2 + R1 * R5 * R9 \\ & * sin(Q2)\hat{}2 - R1 * R6 * R8 * sin(Q2)\hat{}2 - R2 * R4 * R9 \\ & * sin(Q2)\hat{}2 + R2 * R6 * R7 * sin(Q2)\hat{}2 + R3 * R4 * R8 \\ & * sin(Q2)\hat{}2 - R3 * R5 * R7 * sin(Q2)\hat{}2)) \end{aligned}$$

$$e7 = (R1*R9*cos(Q2) - R3*R7*cos(Q2) - R4*R9*sin(Q2) + R6*R7 *sin(Q2))/(R1*R5*R9*cos(Q2)^2 - R1*R6*R8*cos(Q2)^2 - R2*R4*R9*cos(Q2)^2 + R2*R6*R7*cos(Q2)^2 + R3*R4 *R8*cos(Q2)^2 - R3*R5*R7*cos(Q2)^2 + R1*R5*R9 *sin(Q2)^2 - R1*R6*R8*sin(Q2)^2 - R2*R4*R9 *sin(Q2)^2 + R2*R6*R7*sin(Q2)^2 + R3*R4*R8 *sin(Q2)^2 - R3*R5*R7*sin(Q2)^2)$$

$$e8 = -(96*R1*R9*cos(Q2) - 96*R3*R7*cos(Q2) - 96*R4*R9 *sin(Q2) + 96*R6*R7*sin(Q2) - 625*P3*R1*R6 *cos(Q2)^2 + 625*P3*R3*R4*cos(Q2)^2 + 625*P2*R1 *R9*cos(Q2)^2 - 625*P2*R3*R7*cos(Q2)^2 - 625*P1 *R4*R9*cos(Q2)^2 + 625*P1*R6*R7*cos(Q2)^2 - 625 *P3*R1*R6*sin(Q2)^2 + 625*P3*R3*R4*sin(Q2)^2 + 625 *P2*R1*R9*sin(Q2)^2 - 625*P2*R3*R7*sin(Q2)^2 - 625 *P1*R4*R9*sin(Q2)^2 + 625*P1*R6*R7*sin(Q2)^2)/(625 *(R1*R5*R9*cos(Q2)^2 - R1*R6*R8*cos(Q2)^2 - R2*R4 *R9*cos(Q2)^2 + R2*R6*R7*cos(Q2)^2 + R3*R4*R8 *cos(Q2)^2 - R3*R5*R7*cos(Q2)^2 + R1*R5*R9 *sin(Q2)^2 - R1*R6*R8*sin(Q2)^2 - R2*R4*R9 *sin(Q2)^2 + R2*R6*R7*sin(Q2)^2 + R3*R4*R8 *sin(Q2)^2 - R3*R5*R7*sin(Q2)^2))$$

As all the equation of e are functions of Q2 only, we will denote them as $e_i(Q2)$. Now comparing equations:

$$e3(Q2) = w3$$

$$e4(Q2) = w4$$

$$e7(Q2) = w7$$

$$e8(Q2) = w8$$

Using e3 and e7:

$$e3(Q2) = -\cos Q3 \times \sin Q4$$

$$e7(Q2) = -\sin Q3 \times \sin Q4$$

Dividing the above equation we get:

$$\frac{e7(Q2)}{e3(Q2)} = \frac{\sin Q3}{\cos Q3}$$

Now using e4 and e8:

$$e4(Q2) = -L3 \times \cos Q3$$

$$e8(Q2) = L2 + L3 \times \sin Q3$$

Separating Q3 and dividing the equation we get:

$$\cos Q3 = \frac{e4(Q2)}{-L3}$$

$$\sin Q3 = \frac{e8(Q2) - L2}{L3}$$

$$\frac{\sin Q3}{\cos Q3} = -\frac{e8(Q2) - L2}{e4(Q2)}$$

Now using the above two equations of Q3 and equating them gives equation of Q2 only which is then calculated using MATLAB but the resulting solution is not possible as it contains logarithmic integrals.

3.6.8.3. Numerical solution

Closed form solution of joint angles Q2 and Q3 are not possible and tested previously using algebraic and Paul's method which failed. Hence, we shifted to numerical solution for these angles.

Simultaneous solution method was used to solve the remaining angles and implemented on both MATLAB and python. Final implementation as on raspberry pi was on python using scipy library the results will be discussed in upcoming result section.

Chapter 4: RESULTS

4.1. Ansys Results

All the simulations were done assuming the CAD models as solid bodies whereas in actual the bodies are 3D printed and have infill. Hence, the results are supposed to differ and, in this case, will be worse than the simulations. Simulation results of the system at different configurations mentioned in the methodology section are as follow:

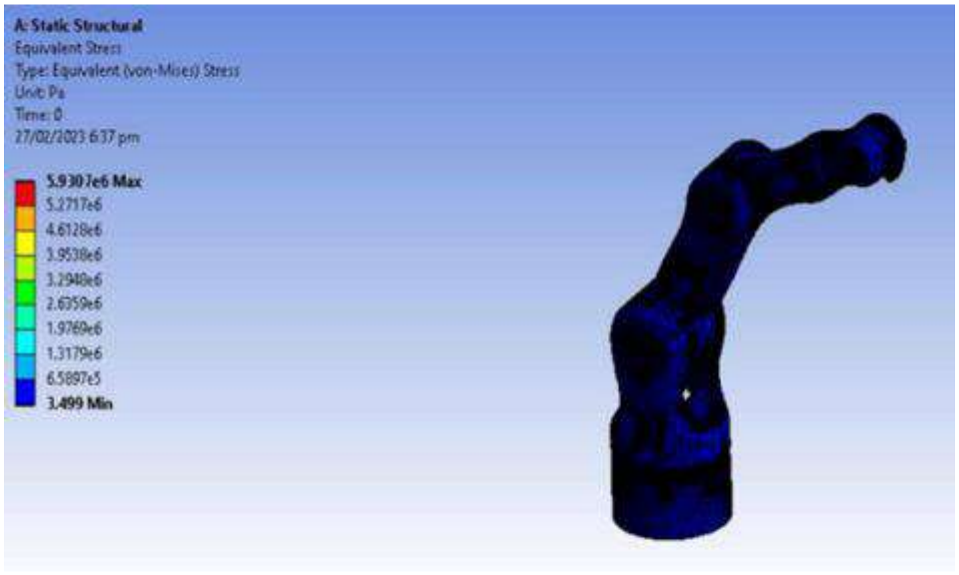


Figure 49. Equivalent Stress in Configuration 1

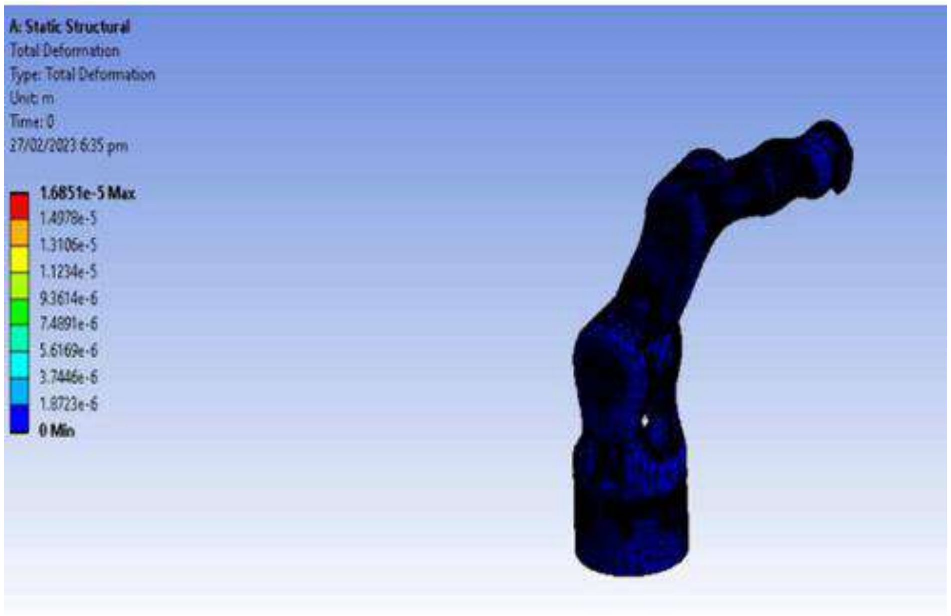


Figure 50. Maximum Deformation in Configuration 1

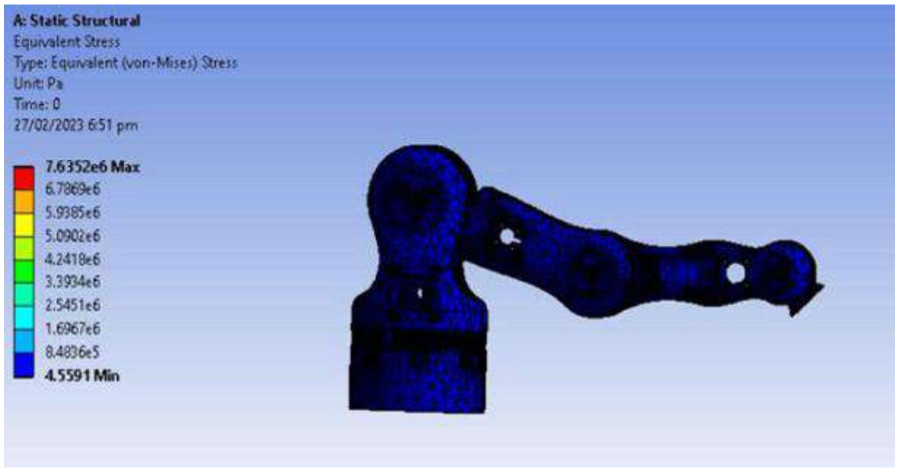

A: Static Structural
Equivalent Stress
Type: Equivalent (von-Mises) Stress
Unit: Pa
Time: 0
27/02/2023 6:51 pm

Figure 51: Equivalent stress in configuration 2

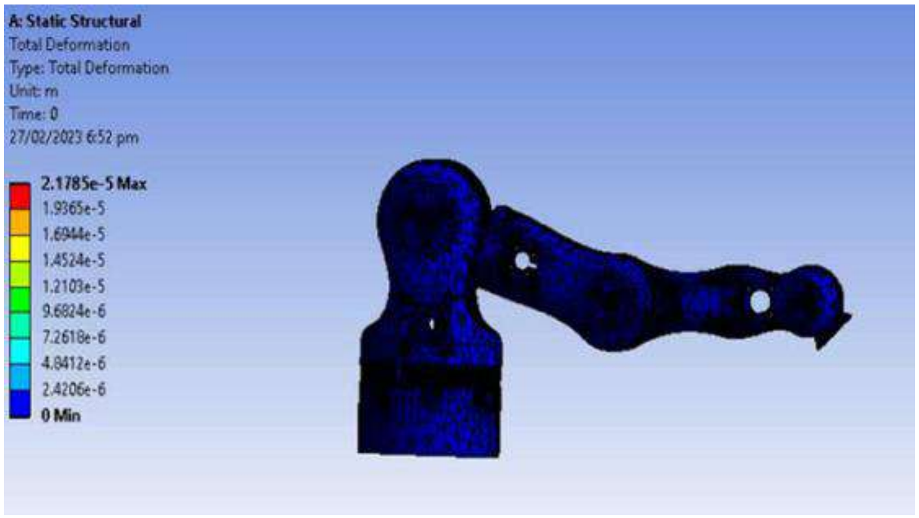

A: Static Structural
Total Deformation
Type: Total Deformation
Unit: m
Time: 0
27/02/2023 6:52 pm

Figure 52: Maximum deformation in configuration 2

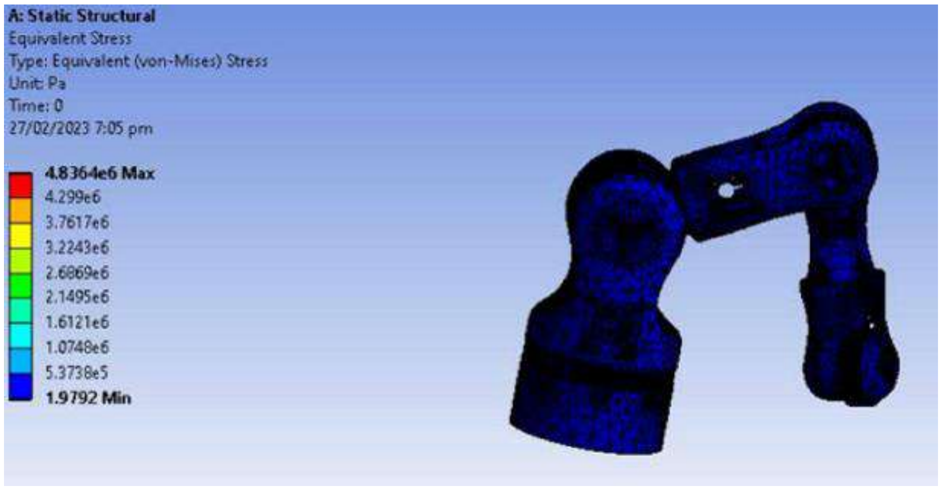

A: Static Structural
Equivalent Stress
Type: Equivalent (von-Mises) Stress
Unit: Pa
Time: 0
27/02/2023 7:05 pm

Figure 53: Equivalent stress in configuration 3

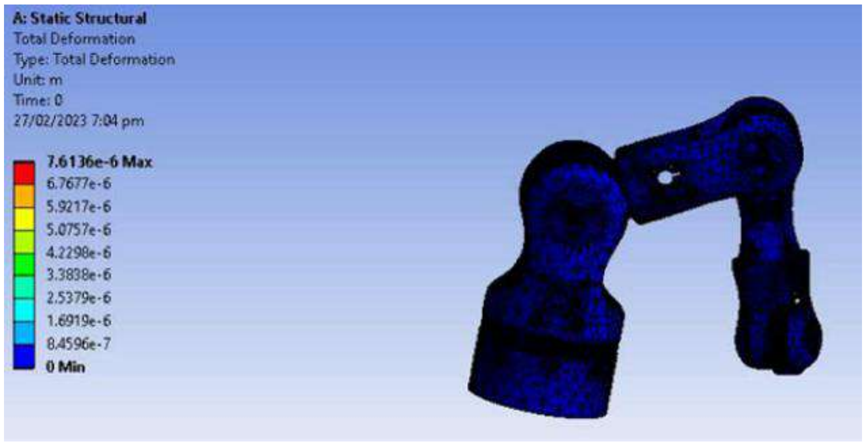


Figure 54: Maximum deformation in configuration 3

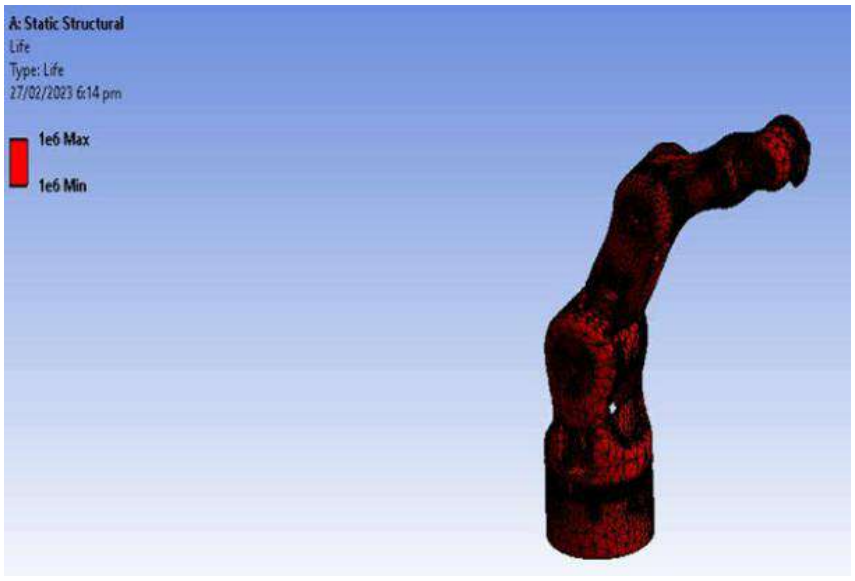


Figure 55: Fatigue life in configuration 1

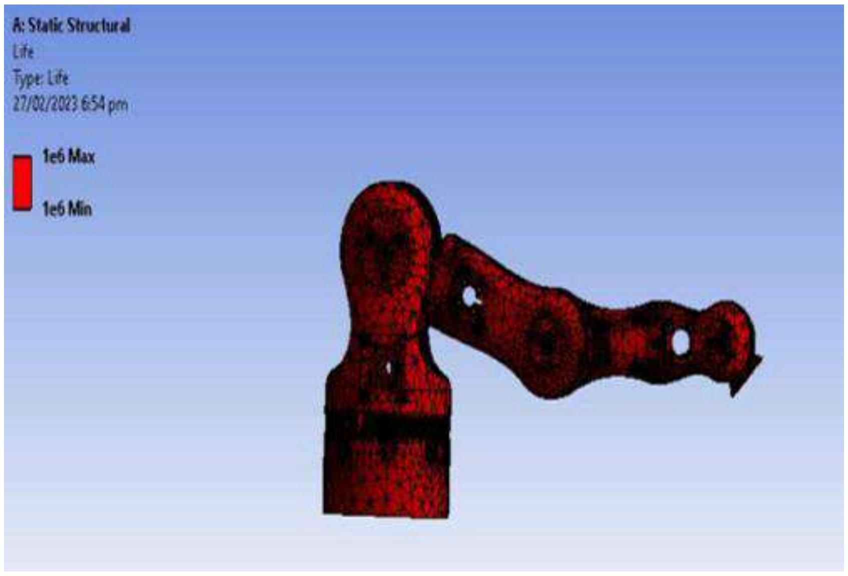


Figure 56: Fatigue life in configuration 2

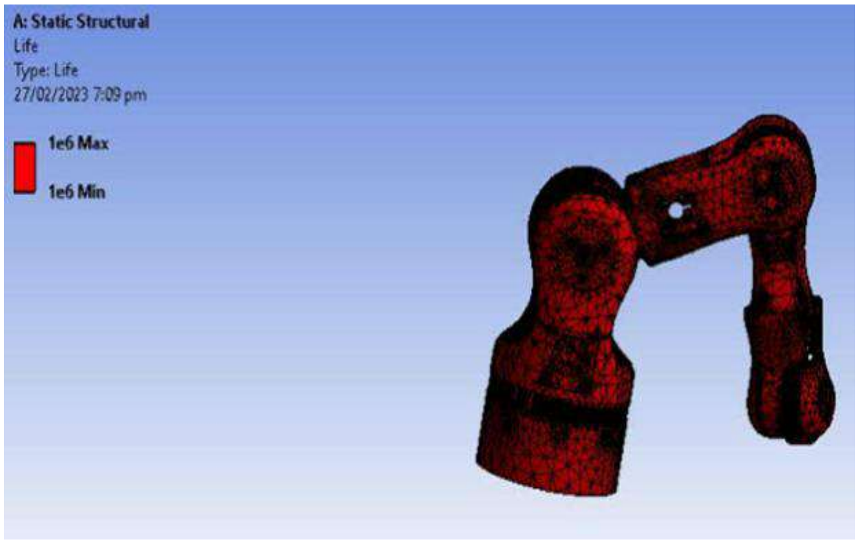


Figure 57: Fatigue life in configuration 3

The factor of safety in all the three configurations is 15. One of the configurations is shown below:

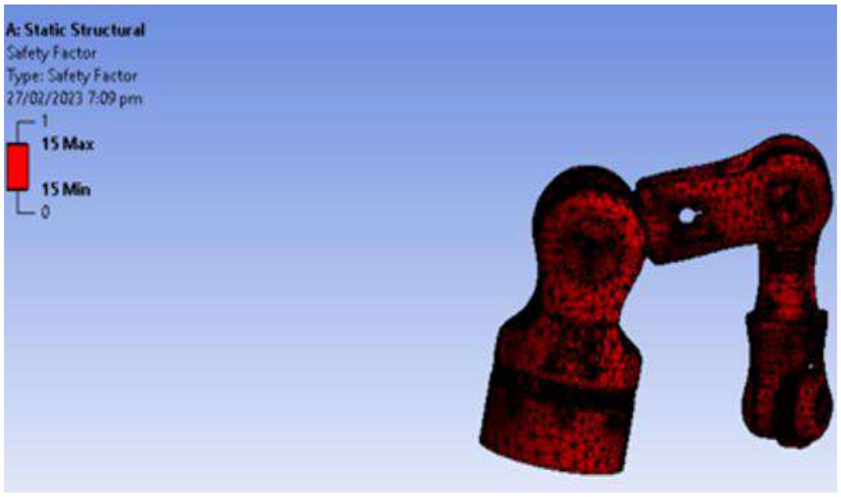


Figure 58: Factor of safety

4.2. Final Assembly

Complete assembly of the serial robotic manipulator with the improved end-effector is as follows:

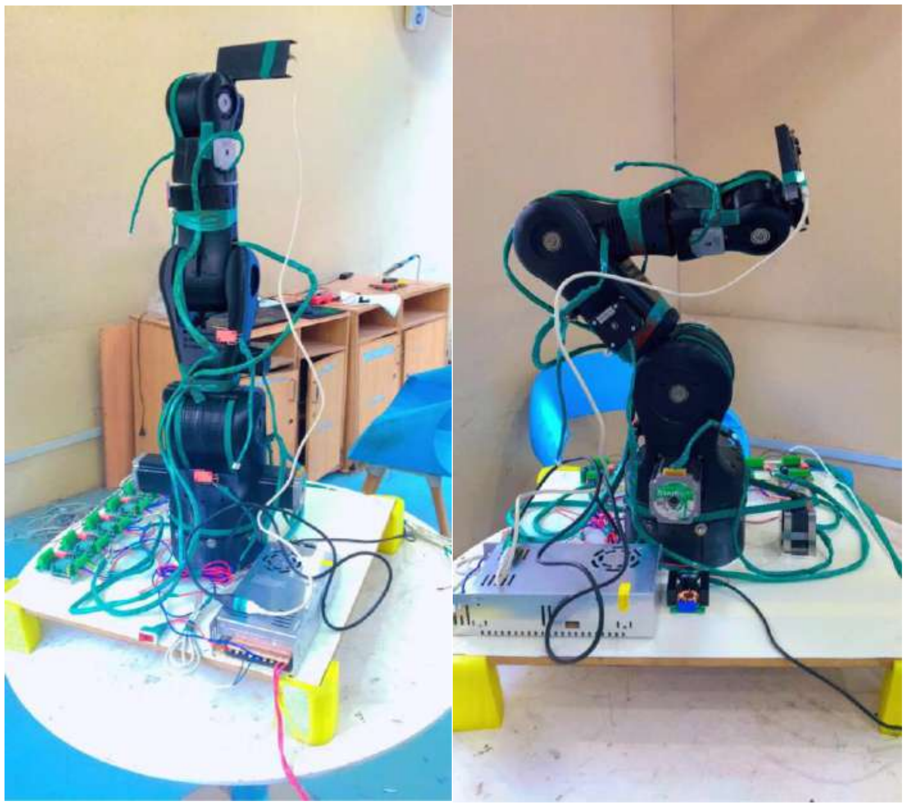


Figure 59: Final assembly in vertical position (Left side) and bended (Right side)

4.3. Computer vision

The results of computer vision algorithm were first taken from the laptop camera which came out to following:

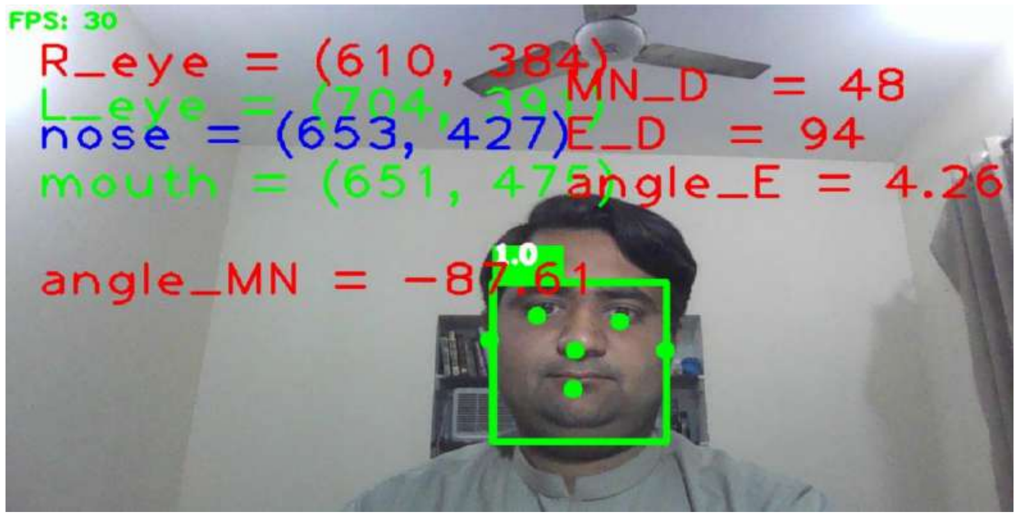


Figure 60: Face detection, localization and depth estimation on laptop

In figure 57 there are four angles as well which were measured initially but later on discarded as we don't want the spoon to be at certain angle while feeding.

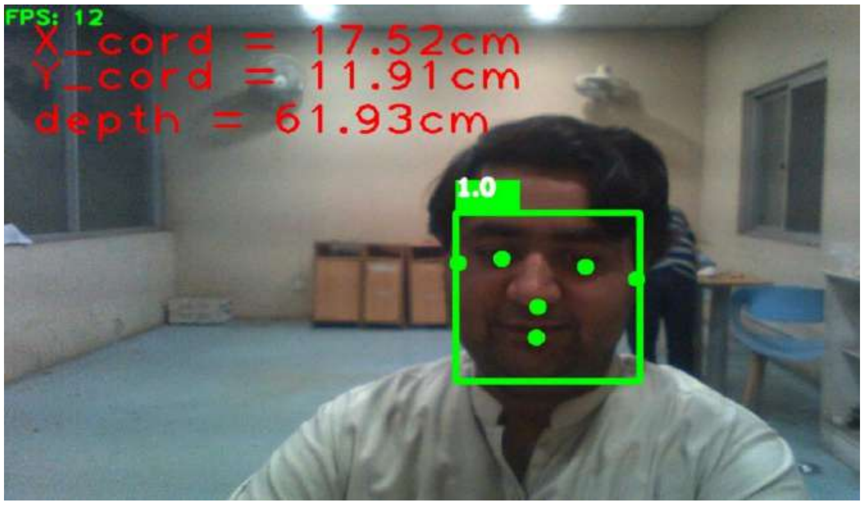


Figure 61: Final detection, localization and depth estimation results on laptop camera

Overall the results were satisfactory and the final error table for the estimation of depth is given below:

Actual/physical (cm)	Measured/computed. (cm)	Error = (Computed – actual)	%Error = (Computed – actual) x 100
46	48.17	2.17	4.71%
51	52.87	1.87	3.6%
73	75.62	2.62	3.58%
40.5	42.5	2	4.94%
60	63.75	3.75	6.25%
		Average = 2.48cm	Average = 4.63%

Table 7: Error calculations for depth estimation

The final results on the raspberry pi are below:

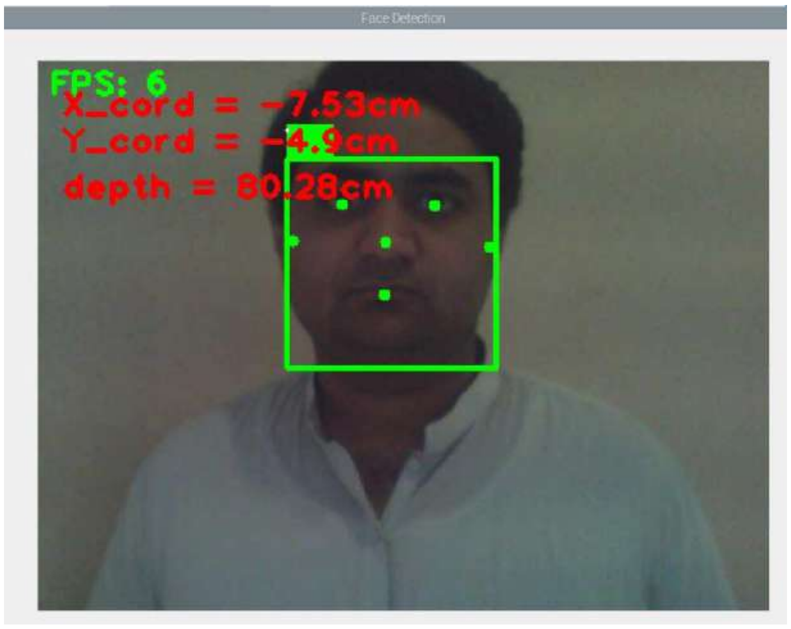


Figure 62: Face detection, localization and depth estimation on controller (1)

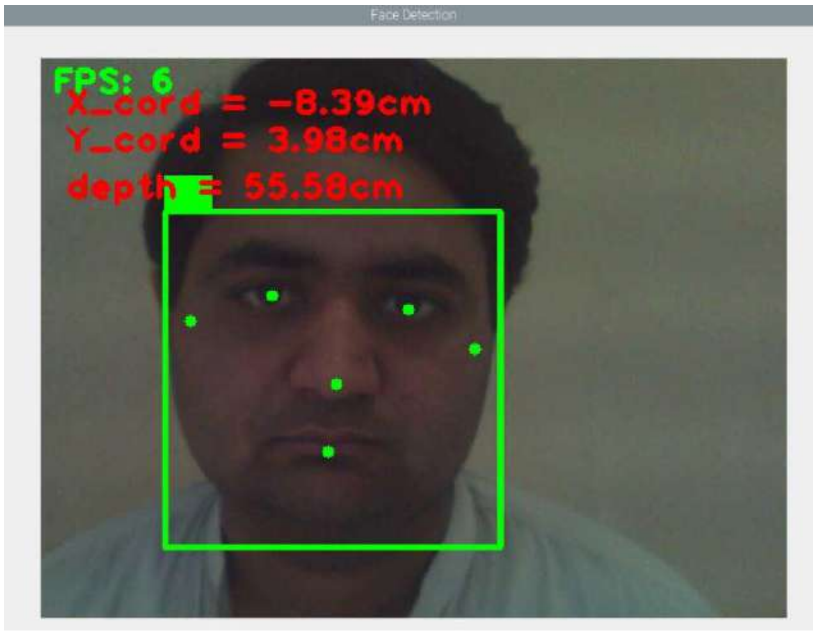
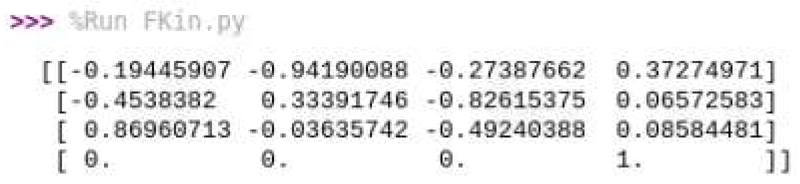


Figure 63: Face detection, localization and depth estimation on controller (2)

4.4. Kinematics

The kinematics was testing by given the joint angles in case of forward kinematics which gave the transformation matrix. Using the transformation matrix from forward kinematics inverse kinematics was implemented which calculated the same results as given for the forward kinematics.

The results for forward kinematics at given angles 10, 20, 30, 40 and 50 degrees are as follows:

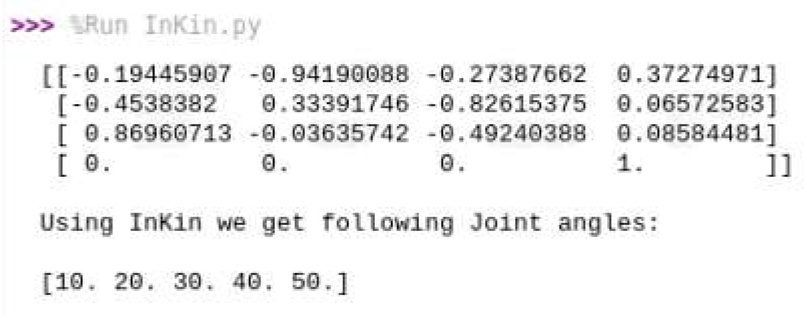


```
>>> %Run FKin.py

 [[-0.19445907 -0.94190088 -0.27387662  0.37274971]
  [-0.4538382   0.33391746 -0.82615375  0.06572583]
  [ 0.86960713 -0.03635742 -0.49240388  0.08584481]
  [ 0.          0.          0.          1.        ]]
```

Figure 64: Forward kinematics results on controller

Time for forward kinematics is less than one second. The results for inverse kinematics where the given transformation matrix is the above one is below:



```
>>> %Run InKin.py

 [[-0.19445907 -0.94190088 -0.27387662  0.37274971]
  [-0.4538382   0.33391746 -0.82615375  0.06572583]
  [ 0.86960713 -0.03635742 -0.49240388  0.08584481]
  [ 0.          0.          0.          1.        ]]

 Using InKin we get following Joint angles:

 [10. 20. 30. 40. 50.]
```

Figure 65: Inverse kinematics results on controller

The solution is accurate as per our settings. The overall time for inverse kinematics is around 1 second normally.

4.5. GUI

The final GUI windows are as follows:

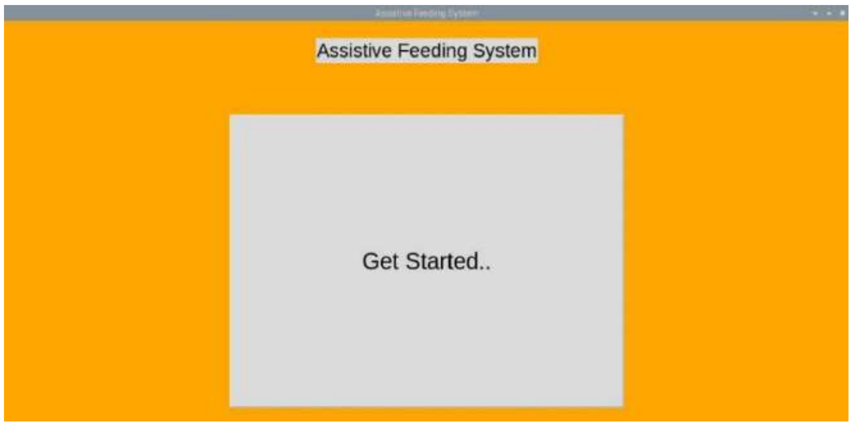


Figure 66: GUI Title window

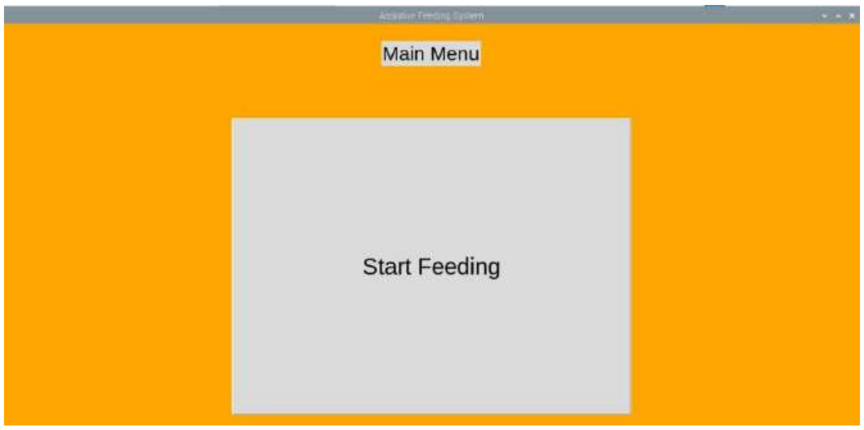


Figure 67: GUI Main window

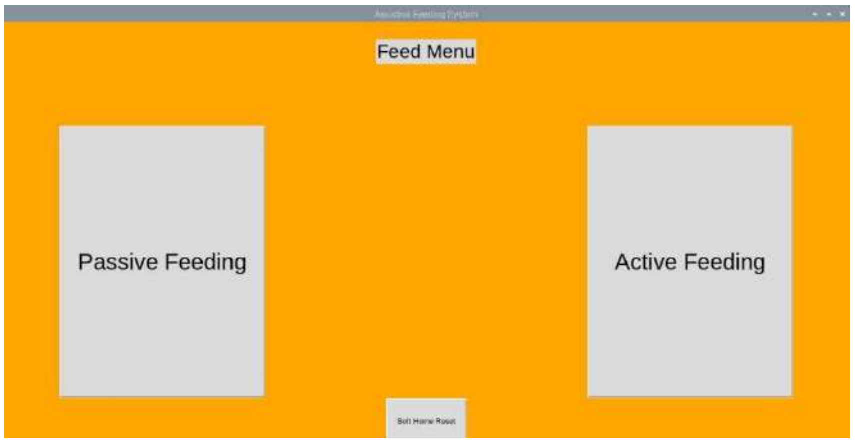


Figure 68: GUI Feeding menu

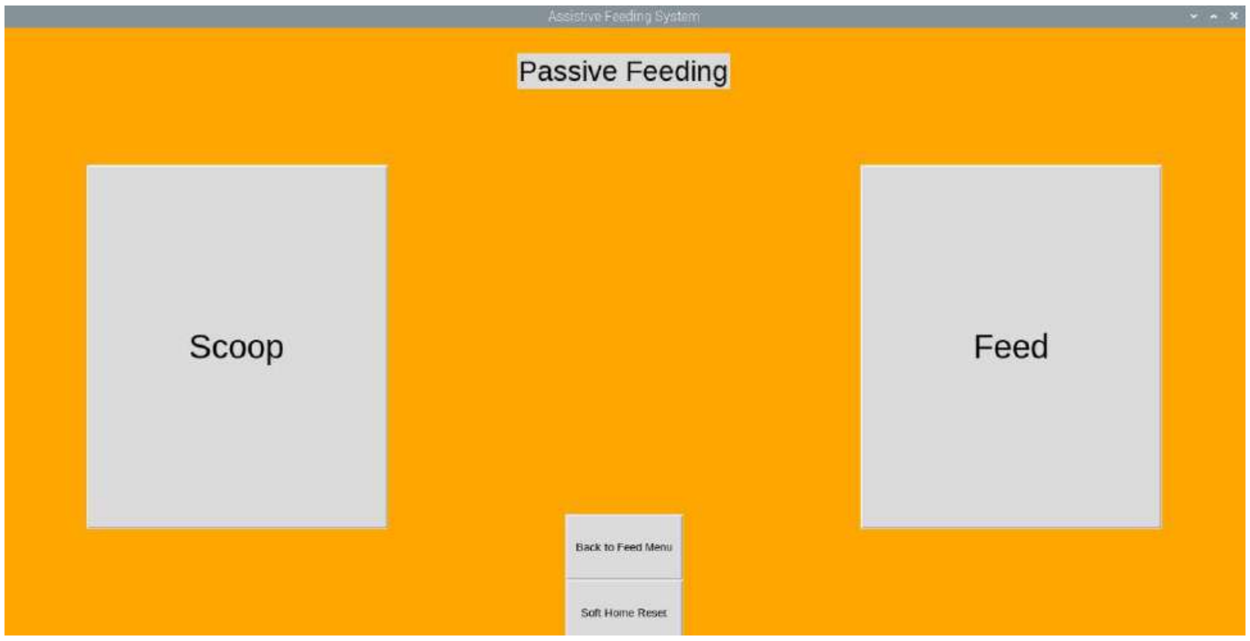


Figure 69: GUI Passive Feeding

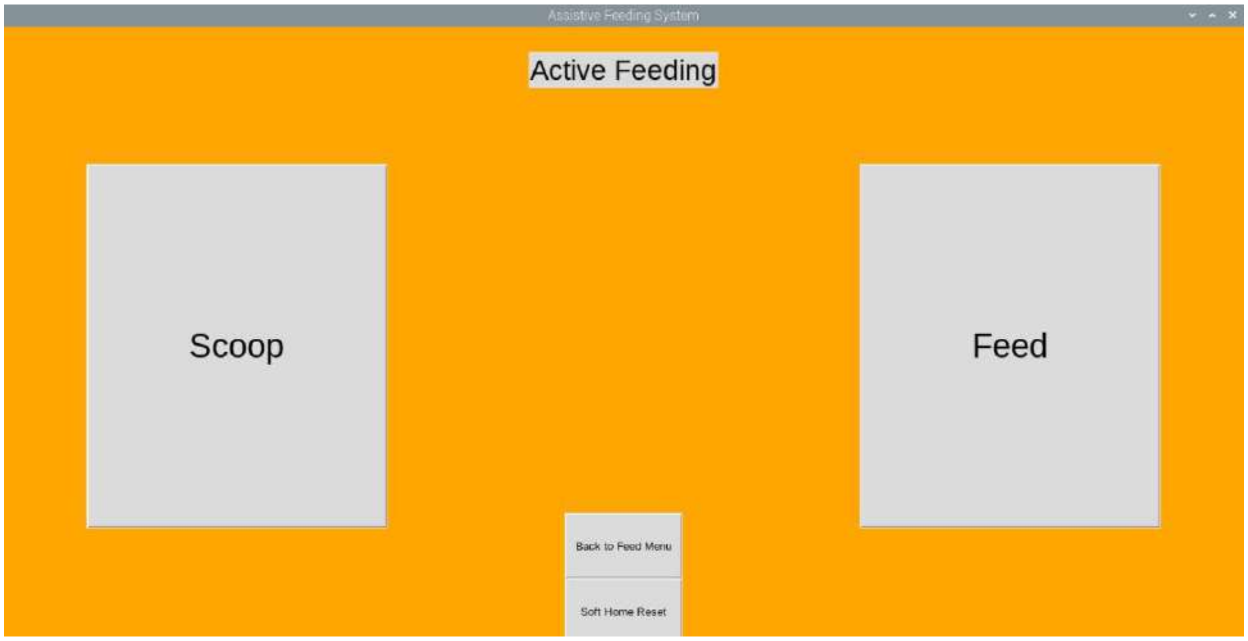


Figure 70: GUI Active Feeding

4.6. Counterweight mechanism

The   final counter weight is shown below:

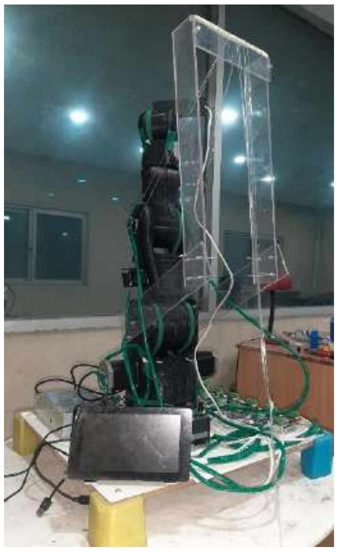


Figure 71: Counterweight assembly with robot

After using counterweight the maximum angle limit shifted from 20 degrees to approximately 60 degrees for link 2.

Chapter 5: CONCLUSION AND FUTURE WORK

5.1. Conclusion

In this project, we developed serial robotic manipulator from scratch using multiple manufacturing techniques (3D Printing and manual machining). We developed all the sub-system from scratch which increased our knowledge and experience in the respective domain including product design, innovation, computer vision, control, kinematics of serial robotic manipulator and many more.

Overall in this project we were able to develop understanding for complex system of the like and to find innovative solution to these complex problems.

Assistive Feeding System is quite complex system involving not only the general requirements like precision, accuracy and repeatability but in as it has to interact with humans it has to be collaborative and has maximum safety features and fail safes. In case of active feeding approach the system needs to be even more robust and incorporate anomaly detections.

5.2. Future Work

There are many opportunities and area to work on in this project which includes but not limited to:

- ➔ Shifting embedded system to better controller like jetson nano etc.
- ➔ Using encoder to get accurate hardware based feedback
- ➔ Improving the current computer vision algorithm
- ➔ Introducing anomaly detection in the system
- ➔ Further modifying the end-effector for multi-tasking
- ➔ Solving kinematic problem using Quaternions or screw theory for complete close form solution

References:

[1] The ebrary website, [Online]. Available: https://ebrary.net/64566/computer_science/spoon

[2] The Kerry medical website, [Online]. Available: https://www.kerrymedical.com/product/bestic/

[3] ProVAR Assistive Robot System Architecture, [Online]
Available at :https://web.stanford.edu/group/rrd/2ndVA/vdl.html

[4] Chi, M., Liu, Y., Yao, Y. *et al*. Development and evaluation of demonstration information recording approach for wheelchair mounted robotic arm. *Complex Intell. Syst.* **8**, 2843–2857 (2022). https://doi.org/10.1007/s40747-021-00350-9

[5] D. -J. Kim *et al*., "How Autonomy Impacts Performance and Satisfaction: Results From a Study With Spinal Cord Injured Subjects Using an Assistive Robot," in *IEEE Transactions on Systems, Man,and Cybernetics - Part A: Systems and Humans*, vol. 42, no. 1, pp. 2-14, Jan. 2018, doi: 10.1109/TSMCA.2011.2159589.

[6] Frank, S. S. I. K. B., Ball, M. V. L. F. T., & Burgard, W. An Autonomous Robotic Assistant for Drinking.

[7] Assistive Mobile Manipulation [Online]
www.researchgate.net/publication/293180411_Assistive_mobile_manipulation_for_self-care_tasks_around_the_head

[8] Chen, Q., Zhu, S., & Zhang, X. (2015). Improved inverse kinematics algorithm using screw theory for a six-DOF robot manipulator. *International Journal of Advanced Robotic Systems*, *12*(10), 140.

[9] The Intech website, [Online]. https://www.intechopen.com/chapters/57605

[10] A. Csiszar, J. Eilers and A. Verl, "On solving the inverse kinematics problem using neural networks," *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Auckland, New Zealand, 2017, pp. 1-6, doi: 10.1109/M2VIP.2017.8211457.

[11] Selective laser sintering (SLS) 3D printing of medicines, [Online] Available at:
https://www.sciencedirect.com/science/article/pii/S0378517317305902?casa_token=0PkS1IbHQqQA
A
AAA:YSYsCM7DjXxJB8IAZQfd2Kszuh7De9iZHAXWJ6i8CFrU8KeR7hO_du32du00zXeVhi8Ftdk
3 1w

[12] The Fritz Ai website, [Online]. Available at: https://www.fritz.ai/object-detection/

[13] Quora website, [Online]. Available at: https://www.quora.com/What-is-the-localization-in-computer-vision

[14] Object Localization in Autonomous Driving, [Online] Available: www.medium.com

[15] Ankita Jaisingh Sheron, Harish Kumar, Fused deposition modeling process parameters and effect on mechanical properties and part quality [Online] Available at: https://link.springer.com/article/10.1007/s40747-021-00350-9

[16] Selective laser sintering (SLS) 3D printing of medicines website , [Online] Available at: https://www.sciencedirect.com/science/article/pii/S0378517317305902?casa_token=2oPDe15P50gAA AA:X4gF-xsBKd34Do5VNW6dEfffVonBjk6JdlhIUiQAx8R9w_J_ZXAboeRSGH4fJ5qCDkVbsQcIl

[17] The Manufacturing guide website, [Online]. Available: https://www.manufacturingguide.com/en/selective-laser-sintering-sls

[18] The motion website [Online]. Available: http://motion.cs.illinois.edu/RoboticSystems/Kinematics.html

[19] D. Chand, S. Gupta and I. Kavati, "Computer Vision based Accident Detection for Autonomous Vehicles," *2020 IEEE 17th India Council International Conference (INDICON)*, New Delhi, India, 2020, pp. 1-6, doi: 10.1109/INDICON49873.2020.9342226.

[20] The website for Twi [Online]. Available: https://www.twi-global.com/technical-knowledge/faqs/what-is-3d-printing/pros-and-cons

[21] G. Luo, L. Zou, Z. Wang, C. Lv, J. Ou, and Y. Huang, “A novel kinematic parameters calibration method for industrial robot based on Levenberg-Marquardt and Differential Evolution hybrid algorithm,” *Robotics and Computer-Integrated Manufacturing*, vol. 71, Article ID 102165, 2021.

[22] D. Sivasamy, M. Dev Anand, and K. Anitha Sheela, “Robot forward and inverse kinematics research using matlab,” *International Journal of Recent Technology and Engineering*, vol. 8, no. 3, pp. 29–35, 2019.

[23] G. Chen, W. G. Zhang, and X. N. Zhang, “Fuzzy neural control for unmanned robot applied to automotive test,” Industrial Robot: International Journal, vol. 40, no. 5, pp. 450–461, 2016.

[24] J. Li, A. Ito, H. Yaguchi, and Y. Maeda, “Simultaneous kinematic calibration, localization, and mapping (SKCLAM) for industrial robot manipulators,” Advanced Robotics, vol. 33, no. 23, pp. 1225–1234, 2019.

[25] Chu Q, Ouyang W, Li H, Wang X, Liu B, Yu N. Online multi-object tracking using CNN-based single object tracker with spatial-temporal attention mechanism. In Proceedings of the IEEE International Conference on Computer Vision 2017 (pp. 4836–4845)

[26] J. Shan, C. K. Toth, *Topographic Laser Ranging and Scanning: Principles and Processing*, CRC Press, Boca Raton, FL 2018

[27] Rostam Affendi Hamzah, Haidi Ibrahim, "Literature Survey on Stereo Vision Disparity Map Algorithms", Journal of Sensors, vol. 2016, Article ID 8742920, 23 pages, 2016. https://doi.org/10.1155/2016/8742920

[28] N. Doulamis and A. Voulodimos, “FAST-MDL: Fast Adaptive Supervised Training of multi-layered deep learning models for consistent object tracking and classification,” in *Proceedings of the 2016 IEEE International Conference on Imaging Systems and Techniques, IST 2016*, pp. 318–323, October 2016.

[29] N. Doulamis, “Adaptable deep learning structures for object labeling/tracking under dynamic visual environments,” *Multimedia Tools and Applications*, pp. 1–39, 2017

[30] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, “ChestX-Ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases,” in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3462–3471, Honolulu, HI, May 2017

[31] Thomas, “3D printed jellyfish robots created to monitor fragile coral reefs,” *3D Printer and 3D Printing News*, 2018. [Online]. Available: http://www.3ders.org/articles/20181003-3d-printed-jellyfish-robots-created-to-monitor-fragile-coral-reefs.html, 2019

[32] A Look Inside the Japanese 3D Printing Industry, [Online] Available at: https://3dprint.com/155980/japanese-3d-printing-industry/

[33] M. Lang, “An overview of laser metal deposition,” *A publication of the Fabricators & Manufacturers Association*, 2017. [Online].
Available: https://www.thefabricator.com/article/additive/an-overview-of-laser-metal-deposition. (2019)

[34] GitHub/ BCN3D Moveo [Online] Available at: https://github.com/BCN3D/BCN3D-Moveo

[35] Esorensen [Online] Available at: http://esorensen.com/robotic_arm_1/

[36] Amazon UK [Online] Available at: https://www.amazon.co.uk/FangFang-Stepper-Motor-17HS19-1684S-PG5-Ratio/dp/B09MR17894

[37] All Data Set [Online] Available
https://www.alldatasheet.com/view.jsp?Searchword=Tb6560%20datasheet&gclid=Cj0KCQjwlumhBhClARIsABO6p-xqSkbxHvQiwk4USV1pO92sIrCqC015EEo5GQUq2tY26OCmVprp5lwaAtJTEALw_wcB

[38] Ghael, Hirak, “A Review Paper on Raspberry Pi and its Applications”, October 2020.

[39] Lertsima, C. Chaisomphob, T. and Yamaguchi, E. 2004, Three dimensional FE modelling of a long-span cable bridge for local stress analysis, Structural Engineering and Mechanics. 18(1), 113-124.

[40] Vyavahare, S., Teraiya, S., Panghal, D. and Kumar, S. (2020), "Fused deposition modelling: a review", *Rapid Prototyping Journal*, Vol. 26 No. 1, pp. 176-201

[41] Flash Forge Website [Online] Available at: https://www.flashforge.com/product-detail/flashforge-creator-pro-2-3d-printer

[42] Etron Technologies Inc [Online] Available at: https://etron.com/3d-sensing/

[43] MediaPipe Library [Online] Available at: https://developers.google.com/mediapipe

[44] GitHub FaceMesh Data [Online] Available at:
https://github.com/google/mediapipe/blob/master/docs/solutions/face_mesh.md

[45] Google Developers Face Detector [Online] Available at:
https://developers.google.com/mediapipe/solutions/vision/face_detector%20%20%5B

[46] Google Developers Face Landmarkers [Online] Available at:
https://developers.google.com/mediapipe/solutions/vision/face_landmarker

[47] Electronics Projects Focus [Online] Available at: https://www.elprocus.com/vl53l0x-pin-configuration-circuit-diagram-and-applications/#:~:text=VL53L0X%20is%20a%20laser%20ranging,the%20object%20and%20bounces%20back

[48] Learn Open CV [Online] Available at: https://learnopencv.com/camera-calibration-using-opencv/

[49] Camera Callibration Open CV [Online] Available at:
https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

[50] Calib.io [Online] Available at: https://calib.io/pages/camera-calibration-pattern-generator

[51] Carnegie Mellon University [Online] Available at:
https://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf

[52] Camera Callibration [Online] Available at:
https://www.ics.uci.edu/~majumder/docs/cameracalib.pdf

[53] Robot Academy: Image Formation [Online] Available at:
https://robotacademy.net.au/lesson/image-formation/

[54] Inverse Projection Transformation [Online] Available at:
https://towardsdatascience.com/inverse-projection-transformation-c866ccedef1c

[55] Transforming 2D image point to 3D world Point [Online] Available at:
https://stackoverflow.com/questions/63531312/transforming-2d-image-point-to-3d-world-point-where-z-0

[56] Stanford University [Online] Available at:
https://web.stanford.edu/class/ee364a/lectures/functions.pdf

[57] Edmund Optics [Online] Available at: https://www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/

[58] Princeton Instruments [Online] Available at: https://www.princetoninstruments.com/learn/camera-fundamentals/field-of-view-and-angular-field-of-view

[59] Depth Perception in Computer Vision [Online] Available at: https://birupakshyamahapatra.com/understanding-depth-perception-in-computer-vision/

[60] Monocular Depth Estimtion [Online] Available at: https://paperswithcode.com/task/monocular-depth-estimation

[61] University of Cmabridge [Online] Available at: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-546.pdf

[62] PinHole Camera model [Online] Available at: https://www.dsi.unive.it/~bergamasco/teachingfiles/cvslides/11_pinhole_camera_model.pdf

[63] HediVision [Online] Available at: https://hedivision.github.io/Pinhole.html

[64] Wisama Khalil, Etienne Dombre, Modeling, Identification and Control of Robots, Taylor & Francis, Inc., Bristol, PA, 2002

[65] A. Chennakesava Reddy , B. Kotiveerachari, Different methods of robotic motion planning for assigning and training paralysed person, Journal of Institution of Engineers, 2008:88(2), pp. 37-41.

[66] Latifnavid, M., Azizi, A.: Kinematic modelling and position control of a 3-DoF parallel stabilizing robot manipulator. Intell. Robot. Syst. **107**, 17 (2023)

[67] Robotics Blogspot [Online] Available at: http://malirobotics.blogspot.com/2015/07/pauls-inverse-kinematic-solution-for.html

ANNEXES

MATLAB Codes

Generation of Transformation matrix

```
%% Code to Get Transformation matrics from DH-Parameters
(Modified)
clear all
clc

%% Defining Symbols
syms L1 L2 L3 Q1 Q2 Q3 Q4 Q5
alphaa = [0,90,0,90,-90];    % this is the alpha value for all
the link
a=[0,0,L2,0,0];              % Length of the Link
d=[L1,0,0,L3,0];             %Offset
Q=[Q1,Q2,Q3,Q4,Q5];          % joint angle variation

%% Transformation Matrices
for i = 1:5
switch i
    case 1
       T01= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 2
        T12= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 3
        T23= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 4
        T34= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 5
```

```
        T45= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
end
end

%% Transformation matrics of wrist w.r.t End-effector
T01
T12
T23
T34
T45
T05 = T01*T12*T23*T34*T45;
simplify(T05)
```

Forward Kinematics

```
%% Forward Kinematics
clc
%%%% Enter Values of Thetas
Q1 = input('Enter Value of Theta 1: ');
Q2 = input('Enter Value of Theta 2: ');
Q3 = input('Enter Value of Theta 3: ');
Q4 = input('Enter Value of Theta 4: ');
Q5 = input('Enter Value of Theta 5: ');

%%%% Converting Theta from degree to Radians
Q1 = (Q1*pi)/180;
Q2 = (Q2*pi)/180;
Q3 = (Q3*pi)/180;
Q4 = (Q4*pi)/180;
Q5 = (Q5*pi)/180;

%%%% Link Lengths for BCN3D Moveo
L1 = 0.1536;
L2 = 0.221;
L3 = 0.223;

%%%% Calculations for 11 equations
R1 = cos(Q5)*(cos(Q4)*(cos(Q1)*cos(Q2)*cos(Q3) -
cos(Q1)*sin(Q2)*sin(Q3)) + sin(Q1)*sin(Q4)) -
sin(Q5)*(cos(Q1)*cos(Q2)*sin(Q3) + cos(Q1)*cos(Q3)*sin(Q2));
R2 = -
cos(Q5)*(cos(Q1)*cos(Q2)*sin(Q3)+cos(Q1)*cos(Q3)*sin(Q2))-
sin(Q5)*(cos(Q4)*(cos(Q1)*cos(Q2)*cos(Q3)-
cos(Q1)*sin(Q2)*sin(Q3))+sin(Q1)*sin(Q4));
```

```
R3 = cos(Q4)*sin(Q1) - sin(Q4)*(cos(Q1)*cos(Q2)*cos(Q3) -
cos(Q1)*sin(Q2)*sin(Q3));
R4 = cos(Q1)*(L3*sin(Q2 + Q3) + L2*cos(Q2));

R5 = cos(Q5)*(cos(Q4)*(cos(Q2)*cos(Q3)*sin(Q1) -
sin(Q1)*sin(Q2)*sin(Q3)) - cos(Q1)*sin(Q4)) -
sin(Q5)*(cos(Q2)*sin(Q1)*sin(Q3) + cos(Q3)*sin(Q1)*sin(Q2));
R6 = -sin(Q5)*(cos(Q4)*(cos(Q2)*cos(Q3)*sin(Q1) -
sin(Q1)*sin(Q2)*sin(Q3)) - cos(Q1)*sin(Q4)) -
cos(Q5)*(cos(Q2)*sin(Q1)*sin(Q3) + cos(Q3)*sin(Q1)*sin(Q2));
R7 = -sin(Q4)*(cos(Q2)*cos(Q3)*sin(Q1) -
sin(Q1)*sin(Q2)*sin(Q3)) - cos(Q1)*cos(Q4);
R8 = sin(Q1)*(L3*sin(Q2 + Q3) + L2*cos(Q2));

R9  =  cos(Q2 + Q3)*sin(Q5) + sin(Q2 + Q3)*cos(Q4)*cos(Q5);
R10 =  cos(Q2 + Q3)*cos(Q5) - sin(Q2 + Q3)*cos(Q4)*sin(Q5);
R11 =  -sin(Q2 + Q3)*sin(Q4);
R12 =  L1 - L3*cos(Q2 + Q3) + L2*sin(Q2);

%%%% Making Transformation matrics T04
T05 = [R1, R2, R3, R4
       R5, R6, R7, R8
       R9, R10, R11, R12
       0, 0, 0, 1];
%%%% Display T04 Matrics
disp("The Final Transformation matrics T05 for given thetas is:
");
disp(T05);
disp("");
disp("For Inverse Kinematics use following: ");
disp(R1 + " " + R2 + " " + R3 + " " + R4 + " " + R5 + " " + R6
+ " " + R7 + " " + R8 + " " + R9 + " " + R10 + " " + R11 + " "
+ R12 );
```

<u>Inverse Kinematics (Algebraic and Numerical method)</u>

```
%% Inverse Kinematics
clear all
clc

%%%% Get desired transformation matrics TBW
TBW = inputdlg('Enter TBW seprate with space:');
TBW_mat = str2num(TBW{1});

%%%% performing inverse kinematics
% Solution for Theta 1
Th_1 = atan2(TBW_mat(8),TBW_mat(4));

% Solution for Theta 2 & 3
syms Q2 Q3
```

```
eqns = [sin(Th_1)*(0.223*sin(Q2+Q3)+0.221*cos(Q2)) ==
TBW_mat(8), 0.1536-0.223*cos(Q2+Q3)+0.221*sin(Q2) ==
TBW_mat(12)];
vars = [Q2 Q3];
[Th_2, Th_3] = solve(eqns,vars);
Th_1 = (double(Th_1))*180/pi;
Th_2 = (double(Th_2))*180/pi;
Th_3 = (double(Th_3))*180/pi;

% Solution for Theta 4
Th_4 = asind(TBW_mat(11)/(-1*(sind(Th_2+Th_3))));

% Solution for Theta 5
%Th_5 =
acosd((cosd(Th_2+Th_3)*TBW_mat(10)+sind(Th_2+Th_3)*cosd(Th_4)*(
TBW_mat(9)))/((cosd(Th_2+Th_3)*cosd(Th_2+Th_3))+(sind(Th_2+Th_3
)*cosd(Th_4)*sind(Th_2+Th_3)*cosd(Th_4))));

%%%% Print Angles
disp("Joint Angles (Degrees) will be: ");
disp("Theta 1 = " + Th_1);
disp("Theta 2 = " + Th_2);
disp("Theta 3 = " + Th_3);
disp("Theta 4 = " + Th_4);
disp("Theta 5 = " + Th_5);
```

<u>Inverse Kinematics (Paul's Method)</u>

```
%% Code to Get Transformation matrics from DH-Parameters
(Modified)
clear all
clc

%% Defining Symbols
syms Q2 Q3 Q4 Q5 R1 R2 R3 R4 R5 R6 R7 R8 R9 P1 P2 P3
L1 = 0.1536;
L2 = 0.221;
L3 = 0.223;

Q1 = 10*pi/180;
alphaa = [0,90,0,90,-90];     % this is the alpha value for all
the link
a=[0,0,L2,0,0];               % Length of the Link
d=[L1,0,0,L3,0];              %Offset
Q=[Q1,Q2,Q3,Q4,Q5];           % joint angle variation

%% Transformation Matrices
for i = 1:5
switch i
    case 1
```

```
       T01= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 2
        T12= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 3
        T23= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 4
        T34= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
    case 5
        T45= [cos(Q(1,i)),-
sin(Q(1,i)),0,a(1,i);sin(Q(1,i)).*cosd(alphaa(1,i)),cos(Q(1,i))
.*cosd(alphaa(1,i)),-sind(alphaa(1,i)),-sind(alphaa(1,i))*d(1,
i); sin(Q(1,i)).*sind(alphaa(1,i)),
cos(Q(1,i)).*sind(alphaa(1,i)), cosd(alphaa(1,i)),
cosd(alphaa(1,i))*d(1, i);0,0,0,1];
end
end

TBW = [R1, R2, R3, P1; R4, R5, R6, P2; R7, R8, R9, P3; 0, 0, 0,
1];

%% Transformation matrics of wrist w.r.t End-effector
T25 = T23*T34*T45;
simplify(T25)
T02 = inv(T01*T12*TBW);
display(T02)
simplify(T02)

% for Q1 = 10 Q2 = 20

Q2  = solve (((T02(6)*T02(4))/(T02(2)*T02(3)))==L2, Q2);
```

Work Space

```
%d
d1 = 153.6;
d2 = 0;
d3 = 0;
d4 = 223;
d5 = 0;

%a
a1 = 0;
a2 = 0;
a3 = 221;
a4 = 0;
a5 = 0;

%Alpha
al1 = 0;
al2 = pi/2;
al3 = 0;
al4 = pi/2;
al5 = -pi/2;

%DH Parameters

L1=Link([0,d1,a1,al1,0,0], 'modified');  %% theta, d, a, alpha,
0 for revolute, variable(revolute and ) ofset
L1.qlim = [0 pi/2];
L2=Link([0,d2,a2,al2,0,0], 'modified');
L2.qlim = [0 pi/2];
L3=Link([0,d3,a3,al3,0,0], 'modified');
L3.qlim = [0 pi/2];
L4=Link([0,d4,a4,al4,0,0], 'modified');
L4.qlim = [0 pi/2];
L5=Link([0,d5,a5,al5,0,0], 'modified');
L5.qlim = [0 pi/2];

System = SerialLink([L1 L2 L3 L4 L5], 'name', 'Assistive
Feeding System');
System.plot([0,0,0,0,0],'workspace', [-100 400 -100 400 -50
400]); %,'workspace', [-20 40 -20 20 -10 40]
System.teach;

%forward kinematics
q_test = [0.0000,   -0.1604,   -0.9539 ,   0.0000 ,        0];
T=System.fkine (q_test)

%inverse kinematics
T_test = transl(18, 0, 20)* trotx(pi/2)
```

```
q0 = [0,0, 0, 0, 0]; % Initial guess for joint angles
q = System.ikine(T_test, q0,'mask', [1 1 1 1 1 0], 'ilimit',
1000)
```

Python Codes

Main File (AFS.py)

```
# Main code for 6DOF Serial Robotic Manipulator as "Assistive Feeding System"
import GUI

def main():
    print("Starting Assistive Feeding System..")
    GUI.User_interface()
    return

if _name_ == "_main_":
    main()
```

GUI Code (GUI.py)

```
from tkinter import *
from tkinter import messagebox
import Motor_Control

def User_Interface():

    # Define Stepper Motor pins and create an object
    M1 = Motor_Control.Stepper_Motor(DIR =  7, PUL =  1, limitS = 12, r_angle =
350)
    M2 = Motor_Control.Stepper_Motor(DIR = 20, PUL = 21, limitS = 12, r_angle =
3.3)
    M3 = Motor_Control.Stepper_Motor(DIR =  0, PUL =  5, limitS = 12, r_angle = 10)
    M4 = Motor_Control.Stepper_Motor(DIR =  6, PUL = 13, limitS = 12, r_angle =
2.5)
    M5 = Motor_Control.Stepper_Motor(DIR = 19, PUL = 26, limitS = 12, r_angle =
18)



    def GotoMenu():
        B1.pack_forget()
        L1.pack_forget()
        Main_Menu()
        return

    def GotoFeedMenu():
        B2.pack_forget()
        L2.pack_forget()
        Feed_Menu()
        return
```

```
def GotoPFeed():
    B3.pack_forget()
    B4.pack_forget()
    L3.pack_forget()
    PFeed()
    return

def GotoAFeed():
    B3.pack_forget()
    B4.pack_forget()
    L3.pack_forget()
    AFeed()
    return

def GoBackP():
    B5.pack_forget()
    B6.pack_forget()
    B7.pack_forget()
    L4.pack_forget()
    Feed_Menu()
    return

def GoBackA():
    B8.pack_forget()
    B9.pack_forget()
    B10.pack_forget()
    L5.pack_forget()
    Feed_Menu()
    return

# initialise main window
def init(win):
    win.title("Assistive Feeding System")
    win.configure(bg = 'orange')
    Title()
    return

# 1st Window
def Title():
    L1.pack(side = "top", pady = 30)
    B1.pack(side = "top", pady = 60)
    return

# 2nd window
def Main_Menu():
    L2.pack(side = "top", pady = 30)
    B2.pack(side = "top", pady = 60)
    return

# 3rd Window
```

```
def Feed_Menu():
    L3.pack(side = "top", pady = 30)
    B3.pack(side = "left", padx = 100, pady = 60)
    B4.pack(side = "right", padx = 100, pady = 60)
    return

# 4th Window
def PFeed():
    L4.pack(side = "top", pady = 30)
    B5.pack(side = "left", padx = 100, pady = 60)
    B6.pack(side = "right", padx = 100, pady = 60)
    B7.pack(side = "bottom")
    return

# 5th Window
def AFeed():
    L5.pack(side = "top", pady = 30)
    B8.pack(side = "left", padx = 100, pady = 60)
    B9.pack(side = "right", padx = 100, pady = 60)
    B10.pack(side = "bottom")
    return

# Creating top level window
win = Tk()
# gets the win height and width
windowWidth = win.winfo_screenwidth()
windowHeight = win.winfo_screenheight()
# Positions the window in the center of the page.
win.geometry(f"{windowWidth}x{windowHeight}")

# Title Page
L1 = Label(win,  text = "Assistive Feeding System", font=("Arial", 25))
B1 = Button(win, text = "Get Started..", command=GotoMenu, width = 30, height = 
15, font=("Arial", 30))
# Main Menu Page
L2 = Label(win,  text = "Main Menu", font=("Arial", 25))
B2 = Button(win, text = "Start Feeding", command=GotoFeedMenu, width = 30, 
height = 15, font=("Arial", 30))
# Feed Menu Page
L3 = Label(win,  text = "Feed Menu", font=("Arial", 25))
B3 = Button(win, text = "Passive Feeding", command=GotoPFeed, width = 23, 
height = 13, font=("Arial", 30))
B4 = Button(win, text = "Active Feeding", command=GotoAFeed, width = 23, 
height = 13, font=("Arial", 30))
# Passive Feeding Menu Page
L4 = Label(win,  text = "Passive Feeding", font=("Arial", 25))
B5 = Button(win, text = "Scoop", command=Scoop, width = 23, height = 13, 
font=("Arial", 30))
B6 = Button(win, text = "Feed", command=PFeed, width = 23, height = 13, 
font=("Arial", 30))
B7 = Button(win, text = "Back to Feed Menu", command=GoBackP, width = 23,
height = 13, font=("Arial", 30))
# Active Feeding Menu Page
L5 = Label(win,  text = "Active Feeding", font=("Arial", 25))
B8 = Button(win, text = "Scoop", command=Scoop, width = 23, height = 13, 
font=("Arial", 30))
B9 = Button(win, text = "Feed", command=AFeed, width = 23, height = 13, 
font=("Arial", 30))
B10 = Button(win, text = "Back to Feed Menu", command=GoBackA, width = 23, 
height = 13, font=("Arial", 30))


# initialise and start main loop
init(win)
mainloop()
return

User_Interface()
```

Scooping Code (Scoop.py)

```
# Code for scooping food from a pre-defined position
def SCOOP(M1, M2, M3, M4, M5):
    # Motor Speeds while scooping food
    SM5 = 0.0005
    SM4 = 0.004
    SM3 = 0.004
    SM2 = 0.008
    SM1 = 0.000005

    # Go to safe Position above plate
    MAs = [10, 10, 10, 10, 10]
    M1.Move(MAs(1), SM1)
    M2.Move(MAs(2), SM2)
    M3.Move(MAs(3), SM3)
    M4.Move(MAs(4), SM4)
    M5.Move(MAs(5), SM5)

    # Start Scooping
    SAs = [0, 0, 10, 0, 10]
    M5.Move(SAs(5), SM5)
    M3.Move(SAs(3), SM3)
    M5.Move(SAs(5), SM5)

    Return
```

Motor Control Code (Motor_Control.py)

```
# Stepper Motor Control
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
```

```
class Stepper_Motor:
    old_angle = 0.0
    dir_flag = 1     # 1 Old direction
    dir = 1          # 1 for Clockwise

    def __init__(self, DIR, PUL, limitS, r_angle):
        self.DIR = DIR
        self.PUL = PUL
        self.limitS = limitS
        self.r_angle = r_angle
        GPIO.setup(self.DIR, GPIO.OUT)
        GPIO.setup(self.PUL, GPIO.OUT)
        GPIO.setup(self.limitS, GPIO.IN)

    def Move(self, angle, tspeed):
        if angle > self.old_angle and self.dir_flag == 1:
            net_angle = angle - self.old_angle
            self.old_angle = angle
            # Calculate the required no of steps
            steps = int(net_angle * self.r_angle)
            # Move that motor to that angle using steps
            self.Rotate(steps, tspeed)

        elif angle < self.old_angle and self.dir_flag == 1:
            self.dir_flag = 0
            if self.dir == 0: self.dir = 1
            else: self.dir = 0
            net_angle = self.old_angle - angle
            self.old_angle = angle
            steps = int(net_angle * self.r_angle)
            self.Rotate(steps, tspeed)

        elif angle > self.old_angle and self.dir_flag == 0:
            # Change the direction
            if self.dir == 0: self.dir = 1
            else: self.dir = 0
            self.dir_flag = 1
            net_angle = angle - self.old_angle
            self.old_angle = angle
            steps = int(net_angle * self.r_angle)
            self.Rotate(steps, tspeed)

        elif angle < self.old_angle and self.dir_flag == 0:
            net_angle = self.old_angle - angle
            self.old_angle = angle
            steps = int(net_angle * self.r_angle)
            self.Rotate(steps, tspeed)
        return

    def Rotate(self, steps, tspeed):
        if self.dir==1:
            GPIO.output(self.DIR, GPIO.HIGH)
        else:
            GPIO.output(self.DIR, GPIO.LOW)
        for i in range(steps):
            GPIO.output(self.PUL, GPIO.HIGH)
            sleep(tspeed)
            GPIO.output(self.PUL, GPIO.LOW)
            sleep(tspeed)
        return
```

Generation of Target pose matrix code (GKin.py)

```
#General Kinematics of Robotic Arm including Transformation matrix calculations and forward kinematics
import numpy as np

### General Algorithm
# First find the TBC (Camera to Base) Transformation matrix
# Second we are assuming that the person is upright and its yaw pitch and roll are fixed and alligned with the camera hence only
#   using the 3 cartesian coordinates driving the TWC (World to Camera) matric

#   Then the final equation will be TBT = TWC^(-1)*TBW then result will be used as TOE


def Target():
    TBT = np.linalg.inv(TWC()) * Trans_C_B()
    return

def TWC(X, Y, dZ):
    # The target is in the robot workspace
    if (dZ < 40):
        print('Inside Workspace..')
    else: print('Target is Outside Workspace....')
    return

# Function to get Camera to wrist transformation matrix
def Trans_C_H():
    # Camera position relative to wrist (translation vector)
    x = 1
    y = 1
    z = 1
    T_c = np.array([x, y, z])  # Specify the camera's x, y, z coordinates

    # Transformation matrix from camera to wrist
    TCH = np.eye(4)
    TCH[:3, 3] = T_c

    print("Transformation matrix from camera to wrist:")
    print(TCH)
    return TCH

# Function to get Camera to Base transformation matrix
def Trans_C_B(TOE):
    TBC = TOE * Trans_C_H()
    return TBC
```

Forward Kinematics Code (FKin.py)

```
import numpy as np

def FKin(Q1, Q2, Q3, Q4, Q5):
    # Converting theta from degree to radians
    Q1 = (Q1*np.pi)/180
    Q2 = (Q2*np.pi)/180
    Q3 = (Q3*np.pi)/180
    Q4 = (Q4*np.pi)/180
    Q5 = (Q5*np.pi)/180

    # Link lengths for BCN3D Moveo
    L1 = 0.1536
    L2 = 0.221
    L3 = 0.223

    # Calculations for 11 equations
    R1 = np.cos(Q5)*(np.cos(Q4)*(np.cos(Q1)*np.cos(Q2)*np.cos(Q3) -
np.cos(Q1)*np.sin(Q2)*np.sin(Q3)) + np.sin(Q1)*np.sin(Q4)) -
np.sin(Q5)*(np.cos(Q1)*np.cos(Q2)*np.sin(Q3) + np.cos(Q1)*np.cos(Q3)*np.sin(Q2))
    R2 = -np.cos(Q5)*(np.cos(Q1)*np.cos(Q2)*np.sin(Q3) +
np.cos(Q1)*np.cos(Q3)*np.sin(Q2)) -
np.sin(Q5)*(np.cos(Q4)*(np.cos(Q1)*np.cos(Q2)*np.cos(Q3) -
np.cos(Q1)*np.sin(Q2)*np.sin(Q3)) + np.sin(Q1)*np.sin(Q4))
    R3 = np.cos(Q4)*np.sin(Q1) - np.sin(Q4)*(np.cos(Q1)*np.cos(Q2)*np.cos(Q3) -
np.cos(Q1)*np.sin(Q2)*np.sin(Q3))
    P1 = np.cos(Q1)*(L3*np.sin(Q2 + Q3) + L2*np.cos(Q2))

    R4 = np.cos(Q5)*(np.cos(Q4)*(np.cos(Q2)*np.cos(Q3)*np.sin(Q1) -
np.sin(Q1)*np.sin(Q2)*np.sin(Q3)) - np.cos(Q1)*np.sin(Q4)) -
np.sin(Q5)*(np.cos(Q2)*np.sin(Q1)*np.sin(Q3) + np.cos(Q3)*np.sin(Q1)*np.sin(Q2))
    R5 = -np.sin(Q5)*(np.cos(Q4)*(np.cos(Q2)*np.cos(Q3)*np.sin(Q1) -
np.sin(Q1)*np.sin(Q2)*np.sin(Q3)) - np.cos(Q1)*np.sin(Q4)) -
np.cos(Q5)*(np.cos(Q2)*np.sin(Q1)*np.sin(Q3) + np.cos(Q3)*np.sin(Q1)*np.sin(Q2))
    R6 = -np.sin(Q4)*(np.cos(Q2)*np.cos(Q3)*np.sin(Q1) -
np.sin(Q1)*np.sin(Q2)*np.sin(Q3)) - np.cos(Q1)*np.cos(Q4)
    P2 = np.sin(Q1)*(L3*np.sin(Q2 + Q3) + L2*np.cos(Q2))

    R7 = np.cos(Q2+Q3)*np.sin(Q5) + np.sin(Q2+Q3)*np.cos(Q4)*np.cos(Q5)
    R8 = np.cos(Q2+Q3)*np.cos(Q5) - np.sin(Q2+Q3)*np.cos(Q4)*np.sin(Q5)
    R9 = -np.sin(Q2+Q3)*np.sin(Q4)
    P3 = L1 - L3*np.cos(Q2+Q3) + L2*np.sin(Q2)
    T0E = np.array([[R1, R2, R3, P1],[R4, R5, R6, P2],[ R7, R8, R9, P3],[0, 0, 0, 1]])
    print(T0E)

    return [R1, R2, R3, P1, R4, R5, R6, P2, R7, R8, R9, P3]
```

Inverse Kinematics Code (IKin.py)

```
import numpy as np
from scipy.optimize import fsolve
import FKin

L1 = 0.1536
L2 = 0.221
L3 = 0.223

def equations(q, Th_1, TOE):
    Q2, Q3 = q

    eq1 = np.sin(Th_1) * (0.223 * np.sin(Q2 + Q3) + 0.221 * np.cos(Q2)) - TOE[7]
    eq2 = 0.1536 - 0.223 * np.cos(Q2 + Q3) + 0.221 * np.sin(Q2) - TOE[11]

    return [eq1, eq2]

def InKin(TOE):
    # Calculate Q1
    Q1 = np.arctan(TOE[7]/TOE[3])
    #Q1_2 = Q1_1 + np.deg2rad(180)

    # Solve for Q2 + Q3
    # Initial guess for Q2 and Q3
    q_guess = [0, 0]

    # Solve the equations numerically using fsolve
    result = fsolve(equations, q_guess, args=(Q1, TOE))

    # Extract the solutions
    Q2 = result[0]
    Q3 = result[1]

    # Calculate Q3
    #Q3 = np.arccos((L1+L2*np.sin(Q2)-TOE[11])/(L3)) - Q2

    # Calculate Q4
    Q4 = np.arcsin(TOE[10]/(-np.sin(Q2+Q3)))

    # Calculate Q5
    Q5 = np.arcsin((np.cos(Q2+Q3)*TOE[8]-
np.sin(Q2+Q3)*np.cos(Q4)*TOE[9])/(np.square(np.cos(Q2+Q3))+np.square(np.sin(Q
2+Q3)*np.cos(Q4))))
```

```
    # make list
    Qs = [Q1, Q2, Q3, Q4, Q5]
    return np.rad2deg(Qs)

TOE = FKin.FKin(45, 45, 2, 22, 22)
Qs = InKin(TOE)
print("\nUsing InKin we get following Joint angles: \n")
print(Qs)
```

Computer Vision main Code (CV.py)

```
import cv2
from time import time
import mediapipe as mp
import matplotlib.pyplot as plt
import math
import RPi.GPIO as GPIO
from time import sleep
import Motor_Control

def CV_Demo():

    M4 = Motor_Control.Stepper_Motor(DIR =  6, PUL = 13, limitS = 12, r_angle =
2.5)
    M5 = Motor_Control.Stepper_Motor(DIR = 19, PUL = 26, limitS = 12, r_angle =
20)
    M3 = Motor_Control.Stepper_Motor(DIR =  0, PUL =  5, limitS = 12, r_angle = 10)

    SM5 = 0.0005
    SM4 = 0.004
    SM3 = 0.004
    Detected = 0

    M3.Move(20, SM3)
    M4.Move(-20, SM4)
    M5.Move(70, SM5)


    # Load a model stored in Caffe framework's format using the architecture and the
layers weights file stored in the disk.

    opencv_dnn_model = cv2.dnn.readNetFromCaffe(prototxt="deploy.prototxt",

caffeModel="res10_300x300_ssd_iter_140000.caffemodel")
    opencv_dnn_model

    # Initialize the mediapipe drawing class.
    mp_drawing = mp.solutions.drawing_utils

    # Initialize the mediapipe face detection class.
```

```
   mp_face_detection = mp.solutions.face_detection

   # Set up the face detection function by selecting the full-range model.
   mp_face_detector = 
mp_face_detection.FaceDetection(min_detection_confidence=0.4)
   mp_face_detector


   def mpDnnDetectFaces(image, mp_face_detector, display = True):
      S = 0

      # Get the height and width of the input image.
      image_height, image_width, _ = image.shape

      # Create a copy of the input image to draw bounding box and key points.
      output_image = image.copy()

      # Convert the image from BGR into RGB format.
      imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

      # Get the current time before performing face detection.
      start = time()

      # Perform the face detection on the image.
      results = mp_face_detector.process(imgRGB)

      # Get the current time after performing face detection.
      end = time()

      # Check if the face(s) in the image are found.
      if results.detections:

         # Iterate over the found faces.
         for face_no, face in enumerate(results.detections):

            # Draw the face bounding box and key points on the copy of the input image.
            mp_drawing.draw_detection(image=output_image, detection=face,

keypoint_drawing_spec=mp_drawing.DrawingSpec(color=(0,255,0),
                                                     thickness=-1,

circle_radius=image_width//115),

bbox_drawing_spec=mp_drawing.DrawingSpec(color=(0,255,0),thickness=image_wid
th//180))

            # Retrieve the bounding box of the face.
            face_bbox = face.location_data.relative_bounding_box

            # Retrieve the required bounding box coordinates and scale them according
to the size of original input image.
```

```
        x1 = int(face_bbox.xmin*image_width)
        y1 = int(face_bbox.ymin*image_height)


        # Draw a filled rectangle near the bounding box of the face.
        # We are doing it to change the background of the confidence score to make
it easily visible
        cv2.rectangle(output_image, pt1=(x1, y1-image_height//16),
pt2=(x1+image_width//16, y1) ,
                color=(0, 255, 0), thickness=-1)

        pt1=(x1, y1-image_height//16)
        pt2=(x1+image_width//16, y1)
        # Write the confidence score of the face near the bounding box and on the
filled rectangle.
        cv2.putText(output_image, text=str(round(face.score[0], 1)), org=(x1, y1-
25),
               fontFace=cv2.FONT_HERSHEY_COMPLEX,
fontScale=image_width//700, color=(255,255,255),
               thickness=image_width//200)

        X1 = x1
        Y1 = y1-image_height//16
        X2 = x1+image_width//16
        Y2 = y1


        ######  LANDMARKS and FACIAL KEYPOINTS

        landmarks = face.location_data.relative_keypoints

        right_eye = (int(landmarks[0].x *  image.shape[1]), int(landmarks[0].y *
image.shape[0]))
        left_eye = (int(landmarks[1].x *  image.shape[1]), int(landmarks[1].y *
image.shape[0]))
        nose = (int(landmarks[2].x *  image.shape[1]), int(landmarks[2].y *
image.shape[0]))
        mouth = (int(landmarks[3].x *  image.shape[1]), int(landmarks[3].y *
image.shape[0]))

         ## coordinates of eyes
        right_Eye_X = int(landmarks[0].x *  image.shape[1])
        right_Eye_Y = int(landmarks[0].y *  image.shape[0])
        Left_Eye_X  = int(landmarks[1].x *  image.shape[1])
        Left_Eye_Y  = int(landmarks[1].y *  image.shape[0])
        Nose_X      = int(landmarks[2].x *  image.shape[1])
        Nose_Y      = int(landmarks[2].y *  image.shape[0])
        Mouth_X     = int(landmarks[3].x *  image.shape[1])
        Mouth_Y     = int(landmarks[3].y *  image.shape[0])
```

```
            # Check if the original input image and the output image are specified to be
displayed.
        if display:

            # Write the time take by face detection process on the output image.
            cv2.putText(output_image, text='Time taken: '+str(round(end - start, 2))+'
Seconds.', org=(10, 65),
                        fontFace=cv2.FONT_HERSHEY_COMPLEX,
fontScale=image_width//700, color=(0,0,255),
                        thickness=image_width//500)

            # Display the original input image and the output image.
            plt.figure(figsize=[15,15])
            plt.subplot(121);plt.imshow(image[:,:,::-1]);plt.title("Original
Image");plt.axis('off');
            plt.subplot(122);plt.imshow(output_image[:,:,::-
1]);plt.title("Output");plt.axis('off');


        # Otherwise
        else:
            if results.detections:
                # Return the output image and results of face detection.
                return
output_image,results,X1,Y1,X2,Y2,right_eye,left_eye,nose,mouth,right_Eye_Y,right_
Eye_X,Left_Eye_X,Left_Eye_Y,Nose_X,Nose_Y,Mouth_X, Mouth_Y
            else:
                #M4.Move(S, SM4)
                #S = S+10

                return output_image, results,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;




    # Initialize the VideoCapture object to read from the webcam.
    camera_video = cv2.VideoCapture(0)
    camera_video.set(3,640)
    camera_video.set(4,480)

    #camera_video.set(3,1280)
    #camera_video.set(4,720)

    # Create named window for resizing purposes.
    cv2.namedWindow('Face Detection', cv2.WINDOW_NORMAL)

    algoirthms = ['Mediapipe']
    algo_index = 0
    # Initialize a variable to store the time of the previous frame.
    time1 = 0
    depth = 0
    Z_Coordinate = 0
    X_Coordinate = 0
    Y_Coordinate = 0

    # focal length of the camera in the x and y direction
    focal_camera_X = 995.338634
    focal_camera_Y = 1002.31638

    # x and y coordinate of the principal point of the camera
    c_X = 396.338386
    c_Y = 265.164652


    # Iterate until the webcam is accessed successfully.
    while camera_video.isOpened():


        # Read a frame.
        ok, frame = camera_video.read()

        # Check if frame is not read properly then continue to the next iteration to read the
next frame.
        if not ok:
            continue

        # Flip the frame horizontally for natural (selfie-view) visualization.
        frame = cv2.flip(frame, 1)

        # Get the height and width of the frame.
        frame_height, frame_width, _ = frame.shape


            # Perform face detection using the Mediapipe algorithm.
        frame, _
,X1,Y1,X2,Y2,right_eye,left_eye,nose,mouth,right_Eye_Y,right_Eye_X,Left_Eye_X,
Left_Eye_Y,Nose_X,Nose_Y,Mouth_X, Mouth_Y = mpDnnDetectFaces(frame,
mp_face_detector, display=False)

        ## convert pixel values of mouth coordinates to float

        mouth_x = float(Mouth_X)
        mouth_y = float(Mouth_Y)
        mouth_x = round(mouth_x, 2)
        mouth_y = round(mouth_y, 2)


        if Left_Eye_X - right_Eye_X != 0:
            slope = (Left_Eye_Y - right_Eye_Y)/(Left_Eye_X - right_Eye_X)

            # distance between eyes
        Eyes_Distance =  Left_Eye_X - right_Eye_X

        ## focal legth and depth
        focal_length = 930.98
        depth = (6.985*focal_length)/Eyes_Distance
        depth = round(depth, 2)
        #Z_Coordinate = depth + (0.0284*930.98)
        #Z_Coordinate = round(Z_Coordinate,2)
        X_Coordinate = ( (mouth_x - c_X)*depth )/focal_camera_X
        X_Coordinate = round(X_Coordinate,2)
        Y_Coordinate = ( (mouth_y - c_Y)*depth)/focal_camera_Y
        Y_Coordinate = round(Y_Coordinate ,2)



    else:
        0


    cv2.putText(frame, "depth = "+str(depth)+"cm", (frame_width//30,
frame_height//4), cv2.FONT_HERSHEY_PLAIN, 2, (0,0, 255), 3)
    #cv2.putText(frame, "depth = "+str(Z_Coordinate)+"cm", (frame_width//30,
frame_height//4), cv2.FONT_HERSHEY_PLAIN, 4, (0,0, 255), 3)
    cv2.putText(frame, "X_cord = "+str(X_Coordinate)+"cm", (frame_width//30,
frame_height//10), cv2.FONT_HERSHEY_PLAIN, 2, (0,0, 255), 3)
    cv2.putText(frame, "Y_cord = "+str(Y_Coordinate )+"cm", (frame_width//30,
frame_height//6), cv2.FONT_HERSHEY_PLAIN, 2, (0,0, 255), 3)
    Detected = 1

     # Set the time for this frame to the current time.
    time2 = time()

    # Check if the difference between the previous and this frame time &gt; 0 to avoid
division by zero.
    if (time2 - time1) > 0:

        # Calculate the number of frames per second.
        frames_per_second = 1.0 / (time2 - time1)

        # Write the calculated number of frames per second on the frame.
        cv2.putText(frame, 'FPS: {}'.format(int(frames_per_second)), (10,
30),cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 0), 3)

    # Update the previous frame time to this frame time.
    # As this frame will become previous frame in next iteration.
    time1 = time2

    # Display the frame.
    cv2.imshow('Face Detection', frame)

    # Wait for 1ms. If a a key is pressed, retreive the ASCII code of the key.
```

```
        k = cv2.waitKey(1)

        # Check if 'ESC' is pressed and breaSk the loop.
        if(k == 27):
            break

        # Check if 's' is pressed then increment the algorithm index.
        elif (k == ord('q')):
            break


    # Release the VideoCapture Object and close the windows.
    camera_video.release()
    cv2.destroyAllWindows()
    return 1
```

Camera image acquisition Code (Camera_pic.py)

```
import cv2
cam = cv2.VideoCapture(0)
cv2.namedWindow("test")

img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("test", frame)

    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img_name = "opencv_frame_{}.png".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

cam.release()

cv2.destroyAllWindows()
```

Camera calibration Code (Camera_calibration.py)

```
import numpy as np
import cv2 as cv
import glob


# FIND CHESSBOARD CORNERS
# OBJECT POINTS AND IMAGE POINTS

chessboardSize = (7,10)
frameSize = (640,480)

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

size_of_chessboard_squares_mm = 15
objp = objp* size_of_chessboard_squares_mm


# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.


images = glob.glob("*.png")

for image in images:

    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)

    # If found, add object points, image points (after refining them)
    if ret == True:

        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)

        # Draw and display the corners
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(1000)


cv.destroyAllWindows()
```

```
# CALIBRATION

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, frameSize,
None, None)

print("Camera Calibrated: ",ret)
print("\nCamera Matrix:\n",cameraMatrix)
print("\nDistortion Parameters:\n",dist)
print("\nRotation Vector:\n",rvecs)
print("\nTranslation Vector:\n",tvecs)


# Perform camera calibration

# Loop over all images and compute camera matrices
camera_matrices_3x4 = []
for i in range(len(images)):
    # Extract rotation and translation vectors for current image
    rvec = rvecs[i]
    tvec = tvecs[i]

    # Convert rotation vector to rotation matrix
    rotation_matrix, _ = cv.Rodrigues(rvec)

    # Concatenate rotation and translation matrices
    extrinsics_matrix = np.hstack((rotation_matrix, tvec))

    # Compute 3x4 camera matrix
    camera_matrix_3x4 = np.dot(cameraMatrix, extrinsics_matrix)

    # Store camera matrix in list
    camera_matrices_3x4.append(camera_matrix_3x4)

# Print all camera matrices
for i, camera_matrix in enumerate(camera_matrices_3x4):
    print("Camera matrix {}:\n{}".format(i+1, camera_matrix))

# UNDISTORTION

img = cv.imread("new/Etron_cali5.png")
h,  w = img.shape[:2]
newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,h), 1, (w,h))

# Undistort
dst = cv.undistort(img, cameraMatrix, dist, None, newCameraMatrix)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('caliResult1.jpg', dst)
```

```
# Undistort with Remapping
mapx, mapy = cv.initUndistortRectifyMap(cameraMatrix, dist, None, newCameraMatrix,
(w,h), 5)
dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('caliResult2.png', dst)

# Reprojection Error
mean_error = 0

for i in range(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], cameraMatrix, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error

print( "total error: {}".format(mean_error/len(objpoints)) )
```