Develop and Control an Environment Friendly Robot for Industrial Tasks

(CRS Catalyst-5 robotic Manipulator)

Submitted By:

- Mennatullah Abdalazeem
- Muhammed Bahgat
- Mohannad Mohammad Atta
- Mahmoud Mohamed Alaa
- Ahmed Fatouh
- Seif eldin Hany
- Momen Mohsen
- Ahmed Essam

ROBOTICS & CONTROL LAB

Faculty of Engineering - Ain Shams University
1839

Mechatronics Graduation Project Report

Supervisors:

- DR. Shady Maged

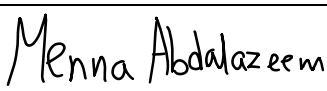
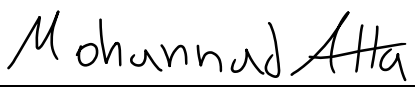
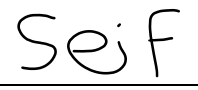
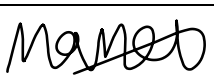
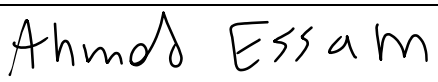
- DR. Mohamed Ibrahim

Date:
2021/2022

DECLARATION

We hereby certify that this Project submitted as part of our partial fulfilment of BSc in ***(Graduation Project-2)*** is entirely our own work, that we have exercised reasonable care to ensure its originality, and does not to the best of our knowledge breach any copyrighted materials, and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our/my work.

Signed: by all students

Mennatullah Abdalazeem	
Muhammed Bahgat	
Mohannad Mohammad Atta	
Mahmoud Mohamed Alaa	
Ahmed Fatouh	
Seif eldin Hany	
Momen Mohsen	
Ahmed Essam	

Date: June 2022.

ACKNOWLEDGMENT

We would like to acknowledge our gratitude to Dr. Mohamed Ibrahim, Faculty of Engineering, Ain-Shams University for helping us throughout our graduation project. And it is such a great pleasure working with him.

We would also like to express our sincere gratitude to Dr. Shady Maged, Faculty of Engineering, Ain-Shams University, without which we wouldn't have been able to get so far in our graduation project, it is a great opportunity for helping us plan & work

We would like also to thank Michel Levis, technical support, CRS Catalyst Robot for his support during our progress in the graduation project, and Roberto Chan, technical support, CRS Catalyst Robot for supporting us in the technical problems while controlling the robotic arm.

Finally, it was a great pleasure to contribute with alexander Eckert, technical support, me-systeme during the configuration of force sensor.

Mennatullah Abdalazeem
Muhamed Bahgat
Mohannad Mohammad Atta
Mahmoud Mohamed Alaa Eldin
Ahmed Fatouh
Seif Eldin Hany
Momen Mohsen
Ahmed Essam

June 2022

ABSTRACT

Conventional robots are made of rigid materials and use rigid motion control approaches that limit their ability to adapt and interact to external forces, constraints and obstacles. Rigid structure and rigid motion control of the robot cannot interact with unplanned change/uncertainty in environment. However, as the field of robotics continues to expand beyond the current control approaches in industrial robots must become increasingly less rigidly controlled and instead capable of safely interacting and navigating through tightly constrained environments. This increases the interest in the interaction between robotics and environment to provide safe work of the robot in addition to adapting to the uncertain dynamic environment.

In this report, we will talk also about our progress in the CRS Catalyst Robotic Arm, about the force configuration required, about the trajectory planning that has been done on it, on the different algorithms done on controlling the robotic arm through MATLAB, our workspace, vision module systems, robot's communication, robot's features & function, challenges we faced and robot's operation method.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

Nowadays, robots have become a huge part in our daily life whether at home, factories, hospitals or any other place. In addition, Robotics is the branch of engineering that deals with design, operation, conception, and manufacturing of robots.
However, dealing with robots isn't an easy task to do as you need to be familiar with robot's behaviors, languages and many other robot's specifications, Also, when we deal with a robot, we should take into consideration the safety of the person dealing with it, so every robot should work in an area to perform its tasks safely and every person should be restricted from entering its workspace while operation, so we need a human-robot interaction model to follow.

Robots aren't limited to those industrial jobs where robots are directly replacing workers, but also, they are able to do many other applications such as:

- Defusing of explosive devices.
- Mine detection.
- Space exploration.
- In biomedical industry such as artificial limbs.
- Working in harsh environments which risks human's health and life such as radioactive environments, extreme pressure and temperature environments and/or lack of oxygen.

Robots are like machines; they can perform many different tough jobs easily but the difference and the advancement are that they can be automated (do it by their own). Once robots are programmed, they can perform the required tasks repeatedly in exactly the same way.

However, Robots have some limitations:

- They need a power supply to keep going.
- People in factories may lose their jobs as robots can replace them.
- They need high maintenance to keep working all day. Unfortunately, the cost of maintaining the robots can be expensive.
- They can store a massive amount of data, but comparing them to our human brains, they are not that efficient for sure.
- Robots can't do anything different than working on the program that has been installed in them.
- Robots can cause a huge amount of destruction if its program comes in wrong hands.

At last, the robot should follow Issac's three laws of robotics:

- A robot may not injure or harm a human being.
- A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
- A robot must protect its own existence except when such protection does not conflict with the First or Second Laws.

Isaac Asimov (1920 – 1992)

CHAPTER TWO: LITERATURE REVIEW

2.1. LITERATURE REVIEW

2.1.1. MATHEMATICAL MODELING OF ROBOTS

While robots are mechanical systems, the focus of this article will be on constructing and using mathematical models for robotics. We'll work on techniques for representing fundamental geometric features of robotic manipulation, dynamic aspects of manipulation, and the numerous sensors available in current robotic systems, in particular. We will be able to build ways for planning and regulating robot motions to fulfil specific tasks using these mathematical models. Here, we'll go over some of the fundamental concepts used in constructing mathematical models for robot manipulators.

2.1.1.1. Symbolic Representation of Robots

The kinematic chain of robot manipulators is made up of links connected by joints. The most common types of joints are rotary (revolute) and linear (prismatic). Like a hinge, a revolute joint permits relative rotation between two links. Between two links, a prismatic joint allows for linear relative motion. R stands for revolute joints, while P stands for prismatic joints.

Each connection between two links is represented by a joint. If the joint is the interconnection of links (I) and (I + 1), we designate the axis of rotation of a revolute joint, or the axis along which a prismatic joint translates, by z i.

2.1.1.2. The Configuration Space

A manipulator's configuration is a detailed description of where each point on the manipulator is located. The configuration space is the collection of all conceivable configurations. Because the individual links of the manipulator are assumed to be rigid, and the base of the manipulator is assumed to be fixed, it is simple to infer the position of any point on the manipulator if we know the values for the joint variables (i.e., the joint angle for revolute joints, or the joint offset for prismatic joints). As a result, throughout this article, a configuration will be represented by a collection of values for the joint variables. We will denote this vector of values by q, and say that the robot is in

configuration q when the joint variables take on the values q 1 · · · q n, with q i = θ i for a revolute joint and q i = d 1 for a prismatic joint.

If an object's configuration can be given with only n parameters, it is said to have n degrees of freedom (DOF). As a result, the number of degrees of freedom is equal to the configuration space's dimension. The number of joints in a robot manipulator influences the number of degrees of freedom. In three-dimensional space, a rigid object has six degrees of freedom (DOF): three for positioning and three for orientation (e.g., roll, pitch and yaw angles). As a result, a manipulator should normally have at least six independent degrees of freedom. With fewer than six degrees of freedom, the arm cannot reach every point in its working environment in any direction. Depending on the application, more than six degrees of freedom may be required, such as reaching around or behind barriers.

2.1.1.3. The State Space

A configuration describes the geometry of a manipulator in real time, but it tells nothing about its dynamic responsiveness. The state of the manipulator, on the other hand, is a set of variables that, when combined with a description of the manipulator's dynamics and input, can predict any future state of the manipulator. The set of all conceivable states is referred to as the state space. The dynamics of a manipulator arm are Newtonian, and they can be defined by generalizing the well-known equation F = ma. Thus, the values for the joint variables q and joint velocities q can be used to specify the manipulator's state (acceleration is related to the derivative of joint velocities). We typically represent the state as a vector $x = (q, \dot{q})^T$. The dimension of the state space is thus 2n if the system has n DOF.

2.1.1.4. The Workspace

The total volume swept out by the end effector while the manipulator executes all possible motions is the manipulator's workspace. The manipulator's shape, as well as mechanical constraints on the joints, limit the workspace. A revolute joint, for example, may be limited to fewer than a full 360 degrees of motion. A accessible workspace and a dexterous workspace are frequently separated in the workplace. The accessible workspace is the whole set of points that the manipulator can reach, whereas the dexterous workspace is the set of points that the manipulator can reach with the end-effector in any orientation. The dexterous workspace is obviously a subset of the reachable workspace. Later in this chapter, the workspaces of various robots are presented

2.1.2. ROBOTS AS MECHANICAL DEVICES

When creating our mathematical models, we will not necessarily consider a lot of physical elements of robotic manipulators. Mechanical elements (such as how the joints are really constructed), accuracy and repeatability, and the tools attached to the end effector are all examples of these. We'll go through a few of them in this section.

2.1.2.1. Classification of Robotic Manipulators

The power source, or how the joints are actuated, the geometry, or kinematic structure, the intended application area, or the technique of control are all factors that can be used to classify robot manipulators. This type of classification is important for determining which robot is best suited for a certain task. A hydraulic robot, for example, would not be appropriate for food handling or clean room applications. We go over this in greater depth below.

Power Source. Robots are typically powered by electricity, hydraulics, or pneumatics. Hydraulic actuators are unequalled in terms of response time and torque generation. As a result, hydraulic robots are mostly utilized to lift hefty weights. Hydraulic robots have the disadvantages of leaking hydraulic fluid, requiring significantly more peripheral equipment (such as pumps, which require more maintenance), and being noisy. Robots powered by DC or AC servo motors are becoming more common as they are less expensive, cleaner, and quieter. Pneumatic robots are affordable and simple to use, but they are difficult to operate accurately. As a result, the range of applications for pneumatic robots is limited, and so is their popularity.

Application Area. Assembly and nonassembly robots are two common classifications for robots. Assembly robots are typically small, electrically powered, and designed in either the revolute or SCARA (explained below) design. Welding, spray painting, material handling, and machine loading and unloading have been the principal nonassembly application areas to date.

Method of Control. Servo and non-servo robots are distinguished by their control methods. Non-servo robots were the first robots. These robots are essentially open-loop machines with movement limited to specified mechanical stops and are mostly used for material transfer. Fixed stop robots, in reality, hardly qualify as robots according to the previously stated criteria. Servo robots can be really multipurpose and

reprogrammable since they use closed-loop computer control to determine their mobility.

Servo controlled robots are further categorized based on the controller's mechanism for guiding the end-effector. The point-to-point robot is the most basic sort of robot in this category. A discrete collection of points can be taught to a point-to-point robot, but there is no control over the path of the end-effector between the taught locations. A teach pendant is normally used to teach such robots a series of points. After then, the points are saved and replayed. The variety of applications for point-to-point robots is severely constrained. Continuous route robots, on the other hand, may control the end-whole effector's path.

Geometry. Currently, most industrial manipulators have less than 6 degrees of freedom. These manipulators are usually kinematically classified based on the first three joints of the arm, and the wrists are described individually. Most of these manipulators fall into one of five geometric types: articulated (RRR), spherical (RRP), scalar (RRP), cylindrical (RPP), or Cartesian (PPP).

Each of these five manipulator arms is a serial link robot. The sixth special class of manipulators consists of so-called parallel robots. In the parallel manipulator, the links are placed in the closed kinematic chain instead of the open kinematic chain. Although this chapter contains a brief description of parallel robots, their kinematics and dynamics are usually only explained in more advanced text because they are more difficult to derive than serial-linked robots.

2.1.2.2. Robotic Systems

A robotic arm should be considered more than just a set of mechanical connections. The mechanical arm is just one component of the overall robotic system consisting of the arm, external power supply, end-of-arm tools, external and internal sensors, computer interface and control computer. Even the programmed software should be considered an integral part of the overall system, as the way a robot is programmed and controlled can have a significant impact on the robot's performance and the scope of subsequent applications.

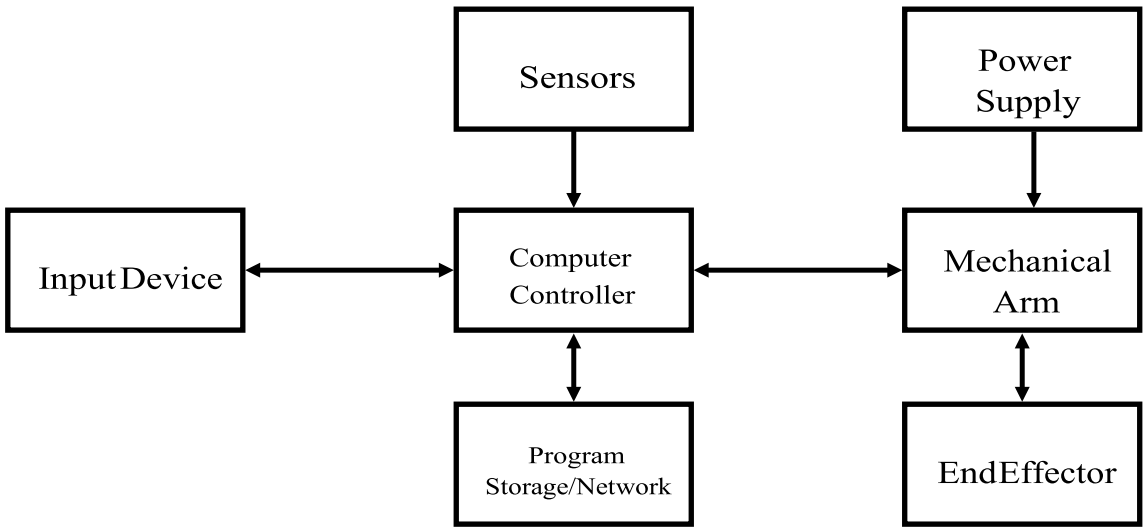


Figure 1. Components of a robotic system

2.1.2.3. Accuracy and Repeatability

The accuracy of a manipulator is a measure of how close the manipulator can come to a given point within its workspace. Repeatability is a measure of how close a manipulator can return to a previously determined point. The primary method of sensing positioning errors in most cases is with position encoders located at the joints, either on the shaft of the motor that actuates the joint or on the joint itself. There is typically no direct measurement of the end effector position and orientation. One must rely on the assumed geometry of the manipulator and its rigidity to infer (i.e., to calculate) the end effector position from the measured joint positions. Therefore, accuracy is affected by calculation errors, processing accuracy of the manipulator structure, flexibility effects such as bending of links under the action of gravity and other loads, gear backlash and many other static and dynamic effects. For this reason, the robot was designed from the start to have very high rigidity. Without high rigidity, accuracy can only be improved by, for

example, determining the position of the end-effector directly with the naked eye.

However, after the operator has programmed a point, for example with a teach pendant, the above effects are taken into account and the control computer stores the exact encoder values needed to return to the set point. Therefore, the reproducibility is mainly affected by the resolution of the controller. The controller resolution represents the smallest increment of motion the controller can detect.

The resolution is calculated as the total distance traveled by the tip divided by 2n. where n is the number of bits of encoder precision. In this regard, a linear axis, i.e., a prismatic joint, usually has a higher resolution than a rotary joint because the linear distance traveled by the end of the linear axis between two points is less than the length of the corresponding arc drawn at the end of the rotary. joint. link.

2.1.2.4. Wrists and End-Effectors

The joint of the kinematic chain between the arm and the end effector is called the wrist. The wrist almost always surrounds everything. It is becoming more common to design manipulators with spherical wrists. This means a wrist where the three joint axes intersect at a common point. The spherical wrist is symbolically shown in the following figure.

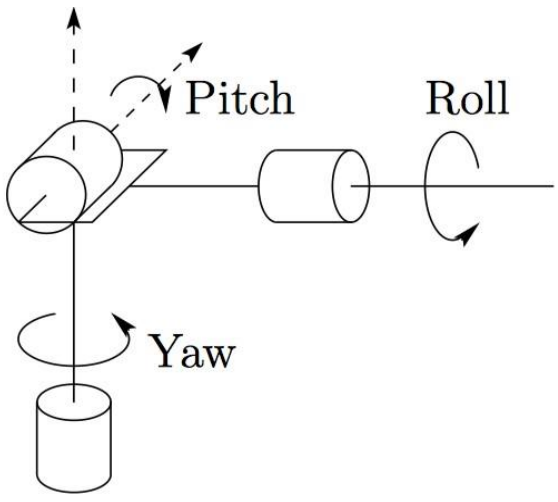


Figure 2. Spherical wrist

2.1.3. FOUNDATIONAL FORCE CONTROL METHODS

In the field of robotic manipulator position and control, many articles show how they can be considered basic, as they are regularly used throughout modern force control research. In, the author proposes a procedure that has become known as accommodation management. As a means of controlling tremors based on closed-loop force feedback, he presents a formal presentation of vector force feedback strategies and needs a sensor that can detect the applied force vector accurately enough to guide the assembly process. This section explains. Adaptive control strategies send speed commands to the manipulator, multiplying the detected force (and torque) by a matrix of damping coefficients that reduce or cancel movement along a particular axis of the end effector frame. Consists of limiting. It has low contact force and high contact force in the direction where contact is not expected, so that the target position can be tracked properly. The stiffness of the end effector frame is controlled using a stiffness matrix that defines the appropriate values for the task. The stiffness required for each joint in relation to these values is determined using the Jacobi mechanism. In this method, by controlling the position of the joint, the position of the end effector is controlled at the same time, and the force applied to the restraint surface is limited.

The main contribution of is a standardized way to describe manipulator task geometry for constrained motion. The idea of dividing restricted movement into natural and artificial limits is formalized in this publication. The official model of the manipulator is represented using an ideal effector. An ideal effector consists of a point in the position space and a point in the force space that represents the position of the end effector and the applied force.

The proposed model of task geometry is represented by an ideal surface composed of smooth hypersurfaces at possible positions of the ideal effector. Model the desired behavior of an ideal effector as a function, using a target trajectory along the ideal surface. This paper also discusses both passive and active compliance control of robot manipulators, but focuses on what the author calls force control. Since the compliance of the mechanism is controllable, "force control" is advantageous because it can be adjusted for different tasks. In one of the most cited papers in the field, the author proposes a method of using hybrid position / force control to simultaneously control the position and force of an end effector on the environment. Hybrid control uses the natural and artificial constraints described above as a means of dividing the problem of position and force control into subtasks, and the axis of the end effector is the axis that needs to be moved and the axis to which the force is applied. Divide between. When a task is split between axes, two separate control loops are shared and the appropriate input from each loop is selected based on the task

definition. These selected outputs are then summed together to work cooperatively to control each joint in the manipulator, where each joint contributes to both position and force. Decomposing the task into purely motion-controlled and purely force-controlled directions is based on the assumption of an ideal constraint consisting of rigid, frictionless contact with a fully known shape. The need for such detailed knowledge of tasks and contact environments, as this method was originally proposed, is a significant flaw that has been the subject of considerable investigation.

Impedance Control, the most cited paper in the field of force control, is published in three parts and controls the dynamic movement of the manipulator that interacts with the environment while controlling the position and velocity at the same time. I am proposing that. The goal, based on physical system theory, is to impose task-appropriate dynamic movements on the manipulator by controlling the gain to give the robot arm the desired impedance. Since the manipulator can experience both free and constrained movement, the behavior must be customizable and the controller can modulate the impedance based on the phase of the task. The impedance imposed should be based on the manipulator's dominant dynamic operation and the task the manipulator is performing. For example, the inertial effect of a manipulator operating underwater can be ignored, but the impedance selection of a robot operating in space should only consider the effect of inertia. Impedance selection should minimize deviations from the desired movement while reducing / controlling the interaction force.

The manipulator must support a robust environment with low impedance (low admittance). Alternatively, the manipulator must have high impedance for contact with compliant surfaces (high admittance) to which motion can be applied. The most serious drawback of the impedance control originally proposed is that it is not possible to place a specific force on the contact surface using this technique.

Several variations of both hybrid and impedance control methods have been proposed since the first announcement. Three of these variations are regularly cited in this area of study and can be considered as basic force control methods in their own right. The hybrid position / force control operating space formulation has been proposed and explained in both the end effector task description and the operating space manipulator dynamics. The task description in the working space is advantageous because the constrained movement in terms of the end effector reference frame only needs to complete two intuitive elements. Vectors of force and moment required to maintain restraint, and end effector degrees of freedom and direction of motion specifications.

A dynamic model of the operating space that describes how movements along the end effector axis interact and how inertia and mass change with manipulator configuration modifies the joint-based dynamic model. Is realized by. The joint-based dynamic model is modified to determine the relationship between the position, velocity, and acceleration of the end effector associated with the virtual force acting along its axis. In this paper, the author also discusses the extension of his formulation to a redundant manipulator and proposes a new approach to address kinematic singularities by treating the manipulator as redundant with respect to end effector movement.

The proposed design improves on the original hybrid control scheme by rigorously considering the dynamics of the manipulator in contact with the environment when developing the position and force control mechanism. This is achieved by using constrained hypersurfaces in the end effector coordinate system for the purpose of creating a complete dynamic model. Parallel position / force control schemes that seek to combine the robustness of impedance control with respect to environmental uncertainty and the hybrid function of controlling both the position of the end effector and the applied force are two main basic methods. The purpose is to combine and fix the shortcomings. This method provides control without the use of a selection matrix that defines force or position control. This means that you can always consider all sensor information when controlling the manipulator. Conflicting control directions between the position loop and the force loop are managed by increasing the priority of the force loop. This limits the deviation from the required applied force.

2.1.4. FORCE CONTROL CATEGORIZATION

Many published articles attempted to summarize the field of manipulator position and force control, classifying the proposed multiple control theorems. We fall into the categories of basic power control and advanced power control.

The basic methods of the force control category are:

- Relationship between position and applied force (both position and force-based stiffness control),
- Uses the relationship between velocity and applied force (impedance and admittance control)
- Apply direct position and applied force (hybrid and hybrid impedance),
- Apply direct force feedback (explicit force control).

The advanced force control category is based on the integration or application of adaptive control, robust control, and / or learning techniques into the basic

techniques described above. Learning methods include neural networks and fuzzy control techniques applied for position and force control. It is not a paper summarizing the field of manipulator force control, but suggests splitting the various methods used to regulate contact force into two basic strategies. Direct force control and internal / external force control. The former includes methods of directly converting the measured force error to the appropriate actuator force / torque, including joint and operating space formulation for hybrid control and parallel position / force control. Internal position / external force control methods include closed-loop force control, provide input to the internal position control loop, and are generally designed for use with existing industrial manipulators with only position-based servos. increase.

A very thorough overview of the subject, including dynamic modeling of the manipulator and its interaction with the environment, a summary of basic approaches to force control, stability concerns, multi-arm system considerations, and task formulas. Great insights can be gained by providing notes on modeling and programming. For those unfamiliar with this field of study. This paper proposes three different basic ideas for controlling position and applied force at the same time as a robot manipulator. Hybrid position / force control, impedance control, linear optimal control. The latter is not explained in enough detail (or using an example) to be considered relevant. Both force control abstracts agree with the idea of classifying control strategies as either hybrid or impedance-based. The above control schemes, which can be considered variations of hybrid control, include: Dynamic hybrid control, parallel force control, and task space hybrid control. Impedance control is identified as a generalization of both stiffness and admittance control, and others identify these basic methods as a subclass of impedance control.

Although hybrid impedance control is classified as an impedance control method, it can be argued that it can also be regarded as a hybrid control method. Due to the more intuitive application of the two core control methods, additional research has been done on the advances made in the field of hybrid position / force control.

CHAPTER THREE: PROJECT MANAGEMENT (1)

Project management is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements.
The five Process groups in project management are:

A. Initiating
B. Planning
C. Executing
D. Monitoring and Controlling
E. Closing

A) Initiating

The Project Initiation **Phase** is the 1st phase in the Project Management Life Cycle, as it involves starting up a new project.
First of all, we should get a team to work in the project in order to reach the project's objectives.

Project objectives:

- Choose suitable place to work in.
- Installing the proper software to interact with the robot.
- Lubrication of robot's internal gears and belts.
- Test the control unit of the robotic arm.
- Configure the robot to do trajectory planning.
- Research about the Force sensor.
- Test the Force sensor.
- Research about the compensator.
- Choose the suitable Compensator.

Project purpose:

Prepare an old robotic platform CRS Catalyst-5 and allowing it to work and perform tasks safely.

Deliverables:

An automated robot to work in a safely environment.

B) Planning

Project planning is a discipline addressing how to complete a project in a certain timeframe.
We should make a plan for the project to reach the project's goals, so we have made:

- The activities we need to follow:

Activity	Description
A	(Choose suitable Place to work)
B	(Transfer robotic arm to the new place)
C	(Test the old PC "with the robotic arm data")
D	(Transfer data to the new PC)
E	(Lubrication of the internal gears)
F	(Test the control unit of the robotic arm)
G	(Run robotic arm with old configuration)
H	(Choose suitable compensator)
I	(Buy the selected compensator)
J	(connect compensator with the robotic arm)
K	(Research about the force sensor)
L	(test the force sensor)
M	(connect force sensor with the robotic arm)
N	(integrate between force sensor and robotic arm)
O	(Select suitable camera)
P	(Connect camera to the robotic arm)
Q	(Run the complete Robotic arm)

Table 1. Shows the activities of the project

- Every activity within its timeframe:

Activity	From (day)	Duration (day)	No. of days per Activity
A (Choose suitable place to work)	0	8	8 days
B (Transfer robotic arm to the new place)	8	12	20 days
C (Test the old PC " with the robotic arm data ")	20	6	26 days
D (Transfer data to the new PC)	26	12	38 days
E (Lubrication of the internal gears)	38	4	42 days
F (Test the control unit of the robotic arm)	38	10	48 days
G (Run robotic arm with old configuration)	48	5	53 days
H (Choose suitable compensator)	53	13	66 days
I (Buy the selected compensator)	66	10	76 days
J (Connect compensator with the robotic arm)	78	7	85 days
K (Research about the force sensor)	53	10	63 days
L (test the force sensor)	63	13	76 days
M (connect the force sensor with the robotic arm)	76	11	87 days
N (Integrate between force sensor and robotic arm)	87	6	93 days
O (Select suitable camera)	53	4	57 days
P (Connect camera to the robotic arm)	57	11	68 days
Q (Run the complete robotic arm)	93	5	**98 days**

Table 2. Shows the activities of the project within its timeframe

- Gantt chart shows what has to be done (the activities) and when (the schedule).

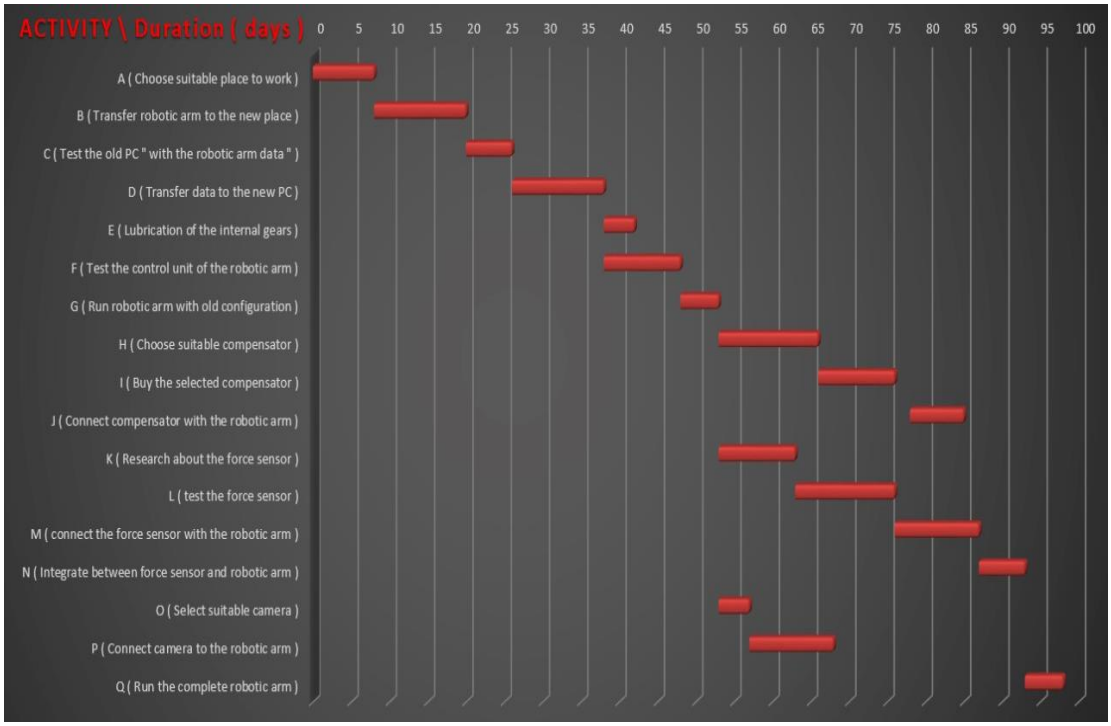


Figure 3. Gantt chart of the project

C) Executing

At first, we had to set the workspace for our robotic arm in order to proceed in our progress to reach our project's goal. Second step, we tried to test the data of the old PC

on the robotic arm but the motherboard was corrupted, so we needed to transfer the data from the old PC to the new one through the Hard disk drive.
Also, we have made many other steps:

- Choose the suitable compensator.
- Research about the force sensor.
- Test the force sensor.
- Test the trajectory planning to move the End-effector to a specific point through a path.

D) Monitoring and controlling

We were able to control the robotic arm via MATLAB by firstly homing it through PUTTY Program.

C) Closing

At the end of this semester, we were able to finish half of the activities for the project according to our schedule [see Table 3].

CHAPTER FOUR: APPROACHING METHODOLOGY

In our graduation project, we used a several tools in order to achieve our target and accomplish our graduation project purpose, which was to prepare an old robotic platform CRS Catalyst 5 and develop a collision protection system to allow the robot to work and perform tasks safely. This can be done either by designing Remote Center Compliance in addition to computer vision.

4.1. End-Effector

In robotics, an end-effector is the device at the end of a robotic arm, designed to interact with the environment. The exact nature of this devise depends on the application of the robot.

In our CRS catalyst manipulator, the end-effector is classified as gripper, in order to do assembly & disassembly process and other operations.

4.1.1. Robotic Arm Appendix

```
%%
% SETUP_LAB_CRS
% This Matlab script sets the control gains, filter parameters, and encoder
% calibration variables in the supplied Simulink models. With WinCon,
% the Simulink models are used to control the CRS CAT-5 Robot.
%
clear;
%
%% Encoders
      % Encoder sensitivity gain (rad/count)
      K_counts = 4000;
      gearA = 72;
      gearB = 19.6;
      gearC = 9.8;
      gear_track = 1000/(1.694/0.0254);
      K_ENC = 2*pi/K_counts*[1/gearA -1/gearA -1/gearA -1/gearB -1/gearC gear_track];
      % track count, about 0 to -67272
      %K_ENC = 360/1000/4/72 * [1 -1 -1 -4.5 -9 15*72/360]*pi/180;
      % Undo the degree to radian change for track sensitivity gain.
      K_ENC(6) = K_ENC(6)/pi/2;
      %
      %%  Control and Filtering
      % Proportional control gain for CRS joints (A/rad)
      Kp = [ 3 3 3 3 3 0.01]*180/pi
      % Derivative control gain for CRS joints(A.s/rad)
      Kd = [.1 .1 .1 .01 .012 .0005]*180/pi
      % Filter cutoff frequency (rad/s)
      wc = 2 * pi * 200;
      % Filter damping ratio
```

```
30.      zeta = 1;
31.      % Gripper proportional control gain (A/mm)
32.      kp_g = 0.08;
33.      % Gripper derivative control gain (A/mm)
34.      kd_g = 1e-4;
35.      % Filter cutoff frequency (rad/s)
36.      wc_g = 2 * pi * 20;
37.      %
38.      %% World and Tool Offsets
39.      % World/task space: [X, Y, Z, pitch, roll]
   (mm,mm,mm,rad,rad)
40.      % World offset to get absolute coordinate
   (mm,mm,mm,rad,rad)
41.      world_offset = [304.8 0 508 pi/2 0];
42.      %
43.      % Tool offset: choose based on end-effector
44.      tool_offset = [0 0 0]; % no attachment
45.      %tool_offset = [50 0 84]; % with force-torque sensor and
   pen wo/ cover
46.      %tool_offset = [64 0 84]; % with force-torque sensor and
   pen
47.      %tool_offset = [0 0 60]; % with force-torque sensor and
   servo gripper
48.      %
49.      %% Force-Torque Sensor: Serial="FT8956"
50.      % BodyStyle="Gamma" Family="DAQ" NumGages="6"
   CalFileVersion="1.0"
51.      C(1,:) = [-0.08970   0.11529   0.65336 -33.13152
   0.24369  33.47858 ]/9.20618775974152;
52.      C(2,:) = [-0.56853  38.75074   0.04872 -19.17145
   0.16300 -19.40194 ]/9.20618775974152;
53.      C(3,:) = [18.59008  -1.27775  18.72747  -1.21139
   19.27238  -1.42622 ]/2.84914887085409;
54.      C(4,:) = [0.02847  -1.01396 -32.87914   2.59443
   33.37469  -1.91408 ]/171.953441268;
55.      C(5,:) = [37.37954  -2.58532 -18.82993   0.37893 -
   19.25579   2.34576 ]/171.953441268;
56.      C(6,:) = [0.31861 -18.47702   0.35463 -18.50179  -
   0.16547 -18.70781 ]/163.930178440371;
57.      %
58.      %% Limits
59.      % Joint 1 Velocity Limit (deg/s)
60.      j1_vel = 100/2;
61.      % Joint 1 Acceleration Limit (deg/s^2)
62.      %j1_acc = 100/2;
63.      j1_acc = 100/4;
64.      %
65.      % Joint 2 Velocity Limit (deg/s)
66.      j2_vel = 100/4;
67.      % Joint 2 Acceleration Limit (deg/s^2)
68.      %j2_acc = 200/4;
69.      j2_acc = 200/8;
70.      %
71.      % Joint 3 Velocity Limit (deg/s)
72.      j3_vel = 100/4;
73.      % Joint 3 Acceleration Limit (deg/s^2)
74.      %j3_acc = 200/4;
75.      j3_acc = 200/8;
```

```
%
% Joint 4 Velocity Limit (deg/s)
j4_vel = 500;
% Joint 4 Acceleration Limit (deg/s^2)
%j4_acc = 500;
j4_acc = 500/2;
%
% Joint 5 Velocity Limit (deg/s)
j5_vel = 250;
% Joint 5 Acceleration Limit (deg/s^2)
%j5_acc = 250;
j5_acc = 250/2;
%
% Track Velocity Limit (mm/s)
track_vel = 1000;
% Track Acceleration Limit (mm/s^2)
track_acc = 250;
%
% Cartesian Velocity Limit: X, Y, and Z (mm/s)
xyz_vel = 100;
% Cartesian Acceleration Limit: X, Y, and Z (mm/s^2)
xyz_acc = 200;
%
% Pitch and Roll Velocity Limit (rad/s)
rot_vel = 250.0 * pi / 180;
% Pitch and Roll Acceleration Limit (rad/s^2)
rot_acc = 250.0 * pi / 180;
```

- Firstly, we tried to search for the com port suitable for our code algorithm, and it was com 1.
- Then we finished setting Encoder sensitivity gain (rad/count) K_counts = 4000.
- Then we define gripper proportional gain and gripper derivative gain and filter cutoff freq. and damping ratio.
- Then we set values for the joints velocity to control its speed, and set a velocity & acceleration limit.
- Finally in order to control the robotic arm either by Joints frame or world frame, we open the Simulink and enter our required values to move the robotic arm to required target

4.1.2. Robotic Arm Homing

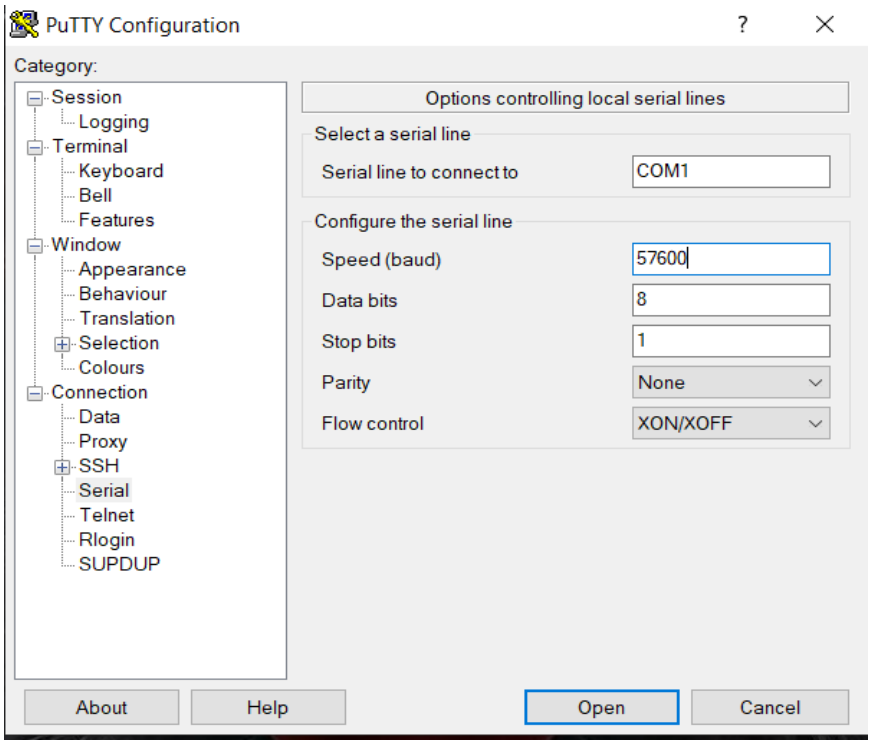


Figure 4. PUTTY Configuration

Due to facing some problems while returning our robotic arm to its home position (homing), finally we had to do the following steps in order to home the robotic arm properly as shown:

a) We had to install PUTTY then open serial and set the speed to (57600) then start session.
b) After starting the session, we press ESC button on the pendant.
c) We write limp on PUTTY software then starting manually adjusting the robotic arm links to be near the home position.
d) After adjusting it manually, we write nolimp on PUTTY software.
e) Next step, we write on PUTTY software (home) then (ready) then (exit).
f) Finally, we can now open MATLAB and start controlling our robotic arm.

4.2. Force Sensor

The force sensor is a transducer that converts an input mechanical load, weight, tension, compression or pressure in an electrical output signal.

The type that is used in our robotic arm is KD24s, where it can measure up to 50N.

The force sensor KD24S is the smallest s-shaped force sensor. It is excellently suited for testing tasks in quality assurance as well as in material testing. Inward and outward force transmission are arranged centrically. Under loading the force transmission brackets are moved parallel.

Figure 5. Force Sensor

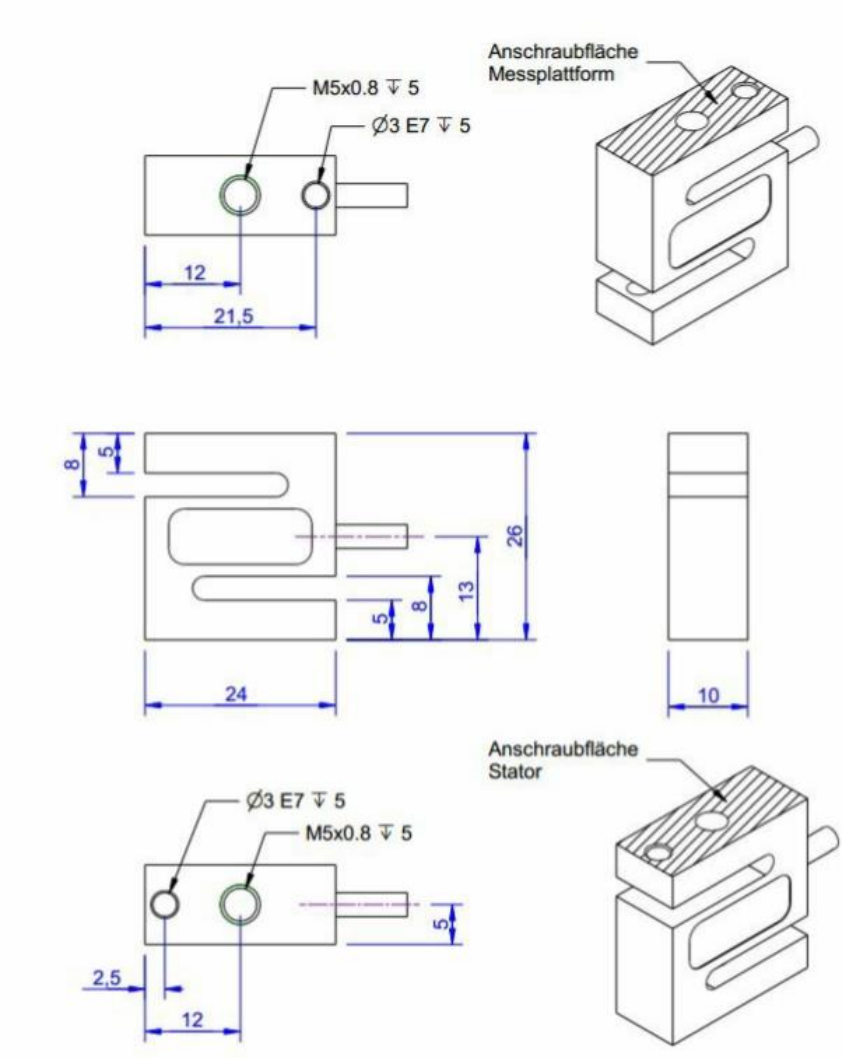


Figure 6. Force Sensor Dimensions

4.2.1. Force Sensor Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matlab example for communication with a GSV-3 %
% (the function requires MinGW-W64 C/C++ compiler package)                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% define the COM Port
com = 4;
% load MEGSV Dynamic Link Library   MEGSV.dll & MEGSV.h for 64bit
if ~libisloaded('MEGSV')
   loadlibrary('.\MEGSV.dll','.\MEGSV.h')
end
% activate channel
[extendet] = calllib('MEGSV','GSVactivate',com,18000);
calllib('MEGSV','GSVstartTransmit',com);
% setting the sampling rate
calllib('MEGSV','GSVsetBaudRate',com, 18000);
%% definition of the variables
x=zeros(1,1); % A Vector filled with Zeros (100x1)
z=int32(0); % A Variable Type Integer32
adx = libpointer('doublePtr',x);
adx2 = libpointer('doublePtr',x);
valsread = libpointer('int32Ptr',z);
errorread = libpointer('int32Ptr',z);
h1 = animatedline;
h2 = animatedline;
h1.Color = 'black';
h2.Color = 'b';
fig = gcf;
fig.Color = 'w';
ax = gca;
ax.Color = [1 1 1];
ax.YGrid = 'on';
ax.GridColor = [0 0 0];
ax.YLimMode = 'auto';
stop = false;
startTime = datetime('now');
while ~stop

    calllib('MEGSV','GSVclearBuffer',com);
     pause(0.01);
[error,data]=calllib('MEGSV','GSVread',com,adx);
[error2,data2]=calllib('MEGSV','GSVread',com,adx2);

    t =  datetime('now') - startTime;
     addpoints(h1,datenum(t),data)
     addpoints(h2,datenum(t),data2)

    if (t<seconds(60)) % defines the window time
```

```
48.             ax.XLim = datenum([0 t]);
49.          else
50.             ax.XLim = datenum([t-seconds(60) t]);
51.          end
52.          datetick('x','keeplimits')
53.          drawnow limitrate
54.      end
55.      calllib('MEGSV','GSVrelease',com)      % release
   Channel
56.      %clear adx valsread
57.      unloadlibrary('MEGSV')             % should be
   done, but crash Matlab
```

- Firstly, we tried to search for the com port suitable for our code algorithm, and it was com 4.
- Then, we loaded the libraries that contains the required functions which is (MEGSV.h).
- Then we finished initialization our variables and set the suitable colors.
- Then we activated our channel and finished setting the sampling rate.
- Then we will use the (calllib()) function the activate a new values in the buffer or clear it.
- Then we used error function that measures the distance between the reference point and the actual point, to correct the output graph with values in mV.
- Finally we used time function that is required to define a new window time when it reaches 60 sec.

4.2.2. Force Sensor Readings

$$Force = Reading \cdot \frac{Max.load}{Max.volt}$$

Where our robotic arm maximum load is (50N) and maximum volt is (0.5mV)

To see the readings click here

4.3. Camera Vision Researchs

It refers to the capability of robot to visually perceive the environment and use this information for execution of different tasks.

Figure 7. (FL3-FW-03S3C-C) Camera

4.3.1. SPECIFICATIONS

Resolution	648 x 488
Frame Rate	76 FPS
Megapixels	0.3 MP
Chroma	Color
Sensor Name	Sony ICX414
Readout Method	Global shutter
Pixel Size	9.9 µm
ADC	12-bit
Gain Range	0 dB to 24 dB
Exposure Range	0.03 ms to >25 seconds
Trigger Modes	Standard, bulb, skip frames, multi-exposure preset, multi-exposure pulse width, overlapped, multi-shot
Image Processing	Gamma, lookup table, hue, saturation, and sharpness
Image Buffer	32 MB
User Sets	2 memory channels for custom camera settings
Flash Memory	1 MB non-volatile memory
Opto-isolated I/O Ports	1 input, 1 output
Non-isolated I/O Ports	2 bi-directional
Power Requirements	8 to 30 V
Power Consumption (Maximum)	<2.5 W
Dimensions	29 mm x 29 mm x 30 mm
Mass	58 rams

Table 3. Camera Specifications

4.3.2. VISION APPLICATION (ARUCO MARKER):

In our robotic arm vision system we will utilize our camera that will be required for object's detection in assembly/disassembly and other specific operations in the ArUco marker process. But it is not accurate.

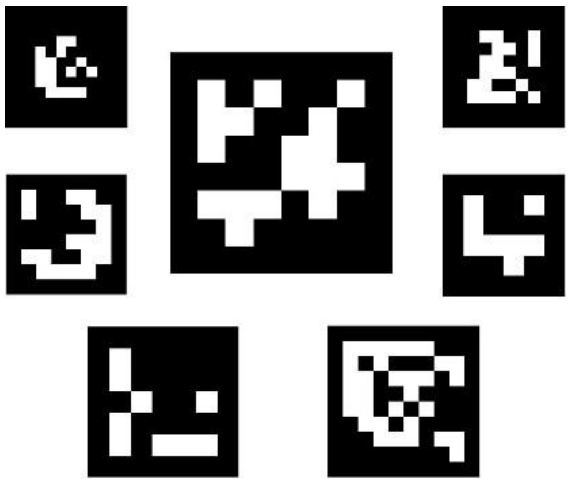


Figure 8. ArUco marker examples

4.3.3. VISION APPLICATION (CNN):

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.Any technology user today has benefitted from machine learning.

CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images. Since CNNs eliminate the need for manual feature extraction, one doesn't need to select features required to classify the images. How CNN work is by extracting features directly from images and the key features are not pretrained; they are learned while the network trains on a collection of images, the post notes. It is the automated feature extraction that makes CNNs highly suited for and accurate for computer vision tasks such as object/image classification.

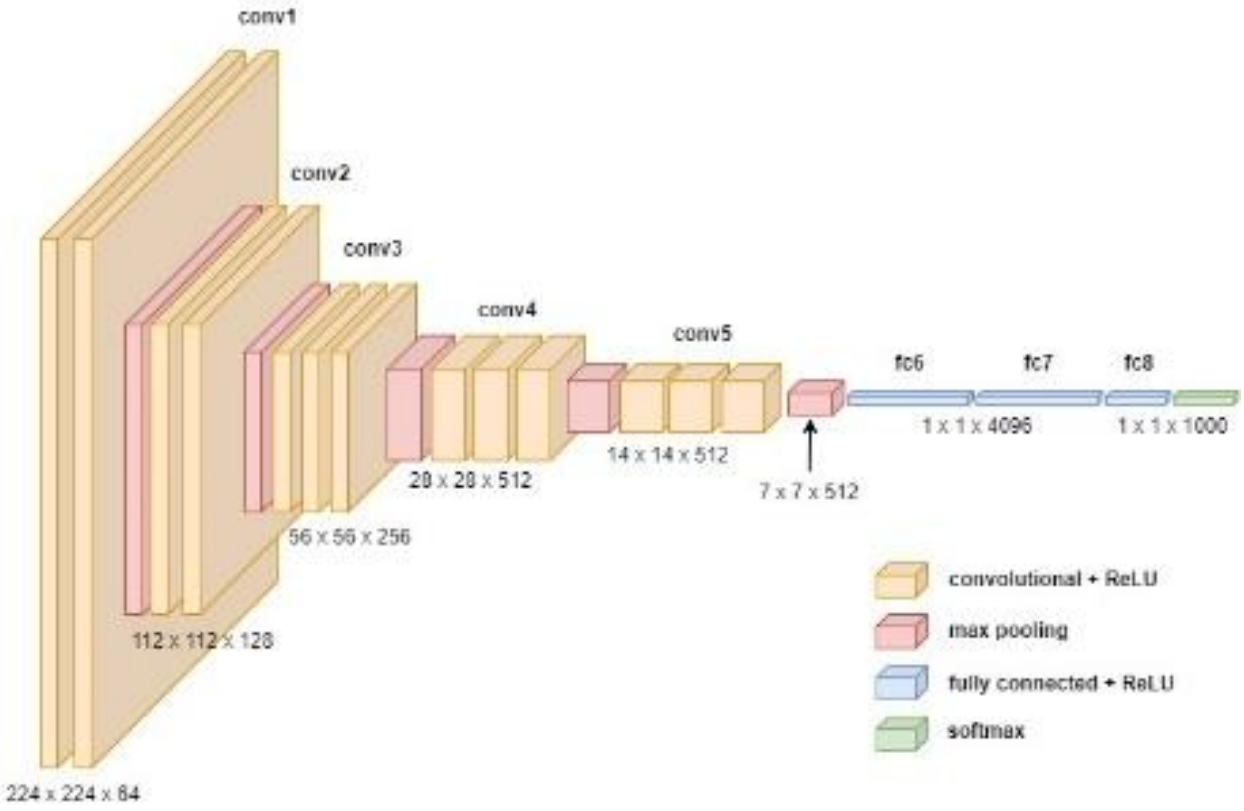


Figure 9. CNN Diagram

4.4. Compensator

It is a device that automatically compensates for part positioning errors, fixture misalignments and variances in part tolerances.

The type that should be used in our robotic arm (RCC) Remote Compliance Compensator.

The Compensator is designed to be utilized in a vertical configuration in "peg-in-hole" operations. The "peg-in-hole" application is a scenario in which one part is inserted into another. Dowel pin insertion, mould alignment, washer insertion, bearing insertion into housings, and shaft insertion into bearings are just a few examples of "peg-in-hole" applications. If the Compensator is utilized in a horizontal position, the shear pads will droop with time. Memorization is a property of rubber and most rubber-like materials. The rubber material memorizes the repeated position over time and will return to it. The shear pads have sagged when this happens. To avoid shear pad sag, the lock-up option should be used.

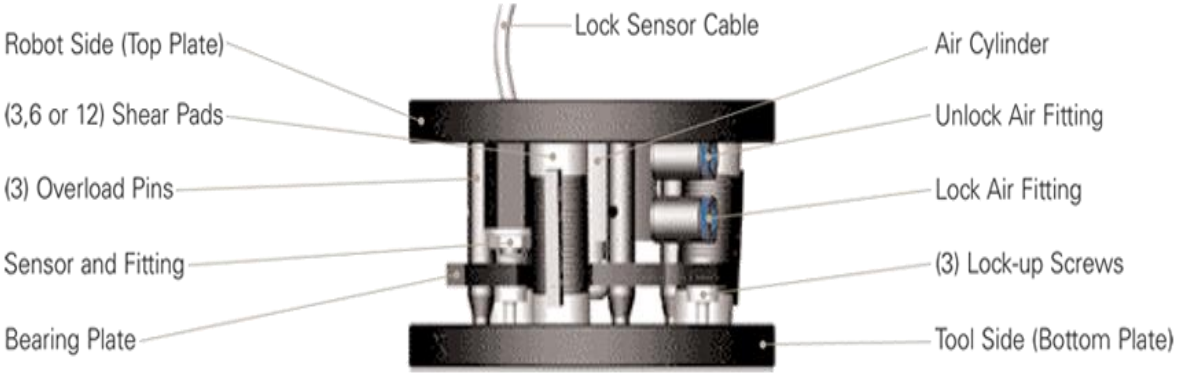


Figure 10. RCC Components

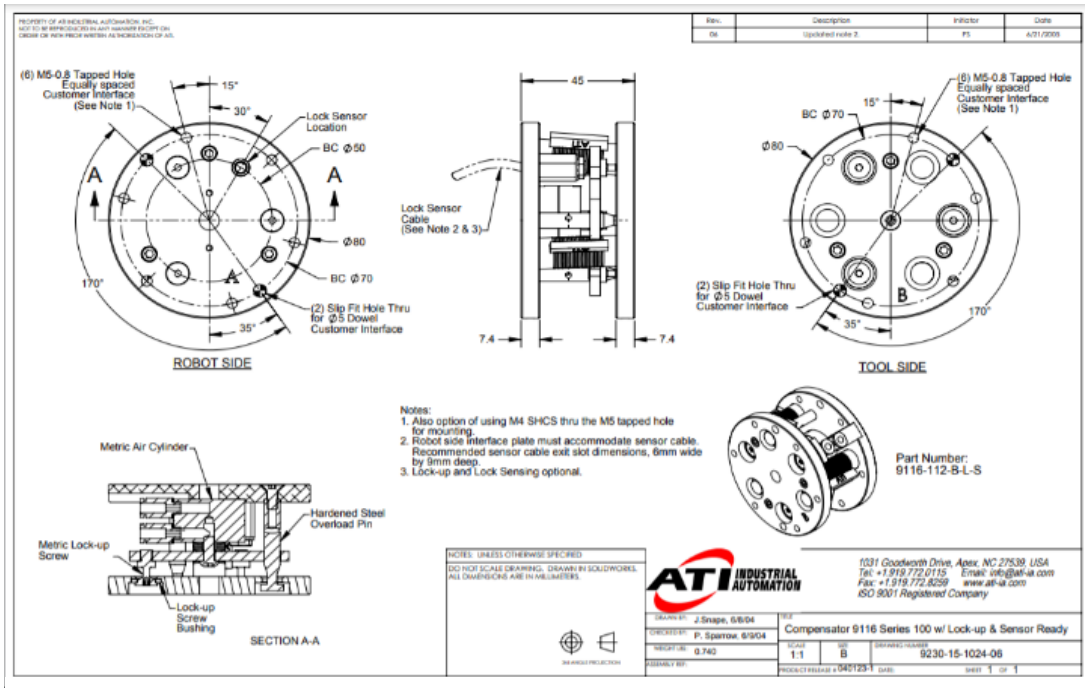


Figure 11. RCC Diagram

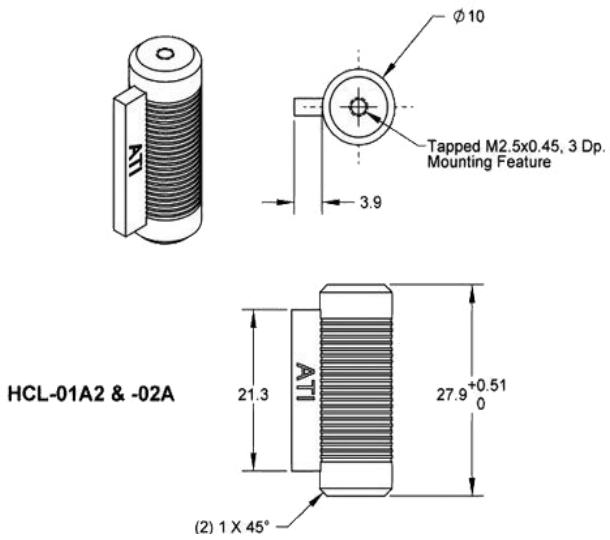


Figure 12. Shear Pad Dimensions

4.4.1. CONTACT FORCE

In many assembly applications, excessive contact force is the fundamental issue. Galling, jamming, and damaged components are all caused by excessive contact force. There are three basic contact pressures in normal assembly processes: single-point, sliding, and two-point (see next figure). The employment of a compliance device with low lateral stiffness is essential for lowering single-point or sliding contact force. A compliance device with a low cocking stiffness reduces two-point contact force.

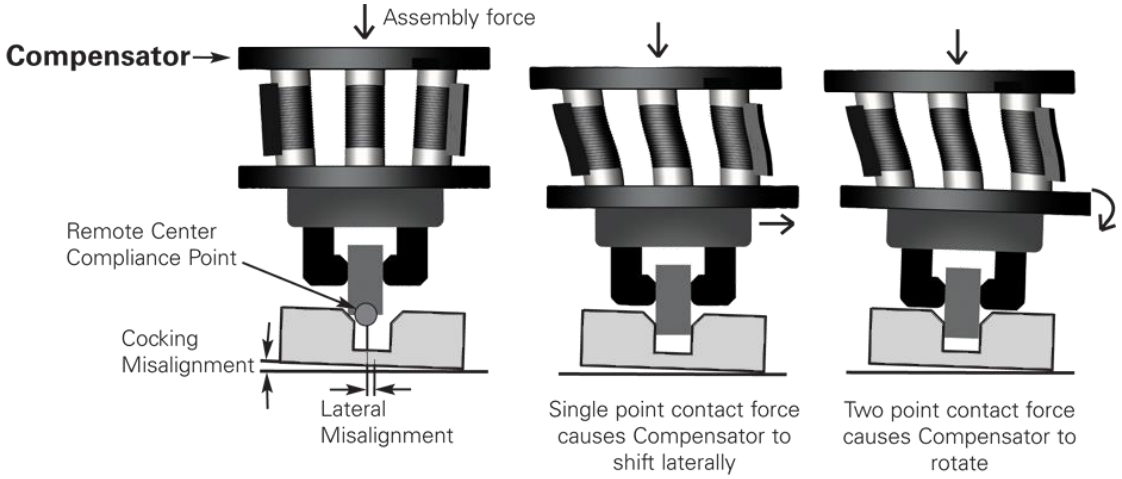


Figure 13. Contact Force on RCC

4.4.2. KEY FEATURES AND BENEFITS

- Designed to provide compliance in the lateral, cocking, axial, and torsional directions.

- Projected (remote) compliance centre: The Centre-of-Compliance (C-of-C) is the point in space at which a contact force will cause a translation with no rotation and a torque will cause rotation with no translation. When the C-of-C is near the insertion contact point, the insertion part axis will align with the location axis during assembly.

- The Compensator consists of a single device with all components contained within the unit's outside diameter.

- The 9116 Series is available in various sizes and configurations

- Units with lock-up include an air cylinder, bearing plate, lock-up screws, and lock-up screw bushings.

- Units with lock-sensing also require a sensor fitting and a cabled proximity sensor.

- 9116 Series 000, 100, 200 and 400 size Compensators have two options for interfacing to a robot assembly machine. Units can be mounted by using the tapped holes on the robot side (top) plate or by bolting through the robot side (top) plate to the robot or assembly machine. The tool side (bottom) plate uses the same two methods for mounting tooling to the unit as the robot side (top) plate.

- All units have (2) dowel pins for location on the robot (top) side plate and (2) dowel pins for location on the tool (bottom) side plate.

- Compliance is limited by three overload pins. When the unit has reached maximum compliance, the overload pins support the load to prevent damage to shear pads.

4.5. Trajectory Planning

Robot trajectory planning usually refers to track points given several expectations and target pose, and timely adjust the rotation angle of each joint of the robot to the end effector at a prescribed trajectory followed by each point to eventually reach the target point.

- For smoothing commands control we inserted a sigmoid block.

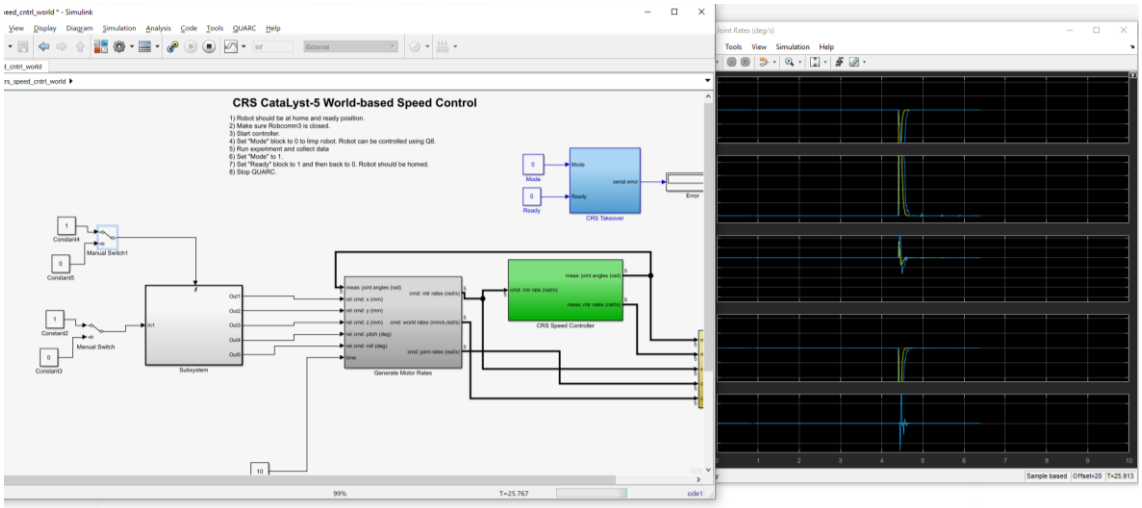


Figure 14. Trajectory Planning without sigmoid block

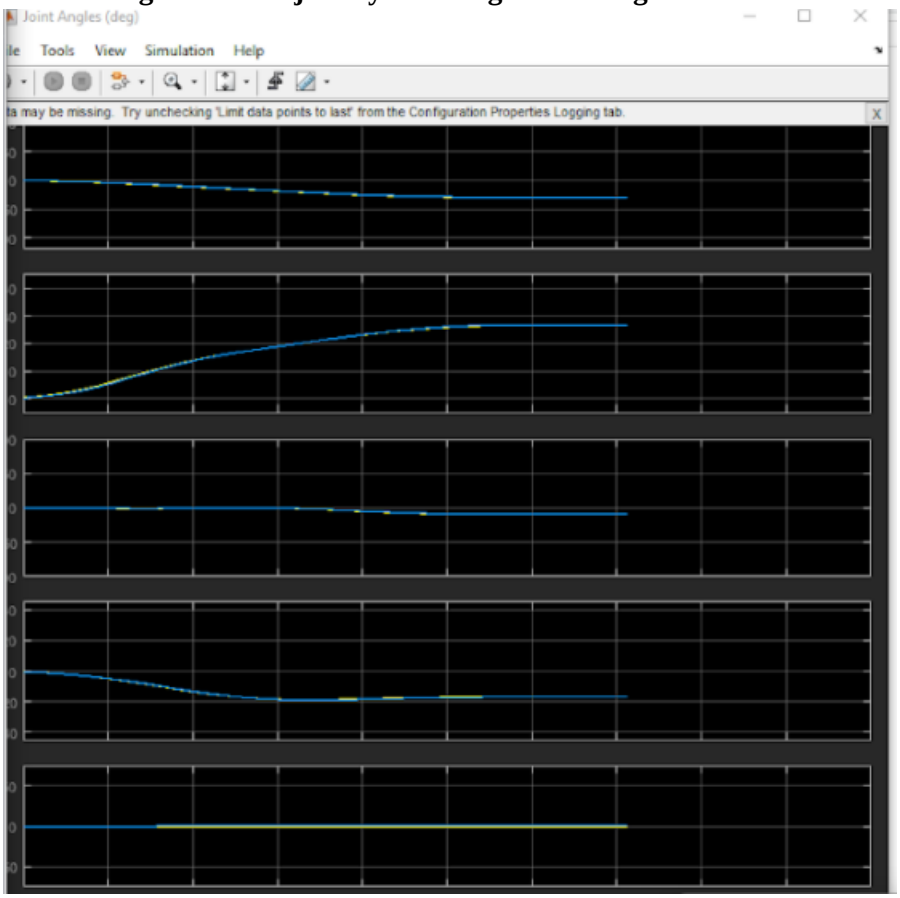
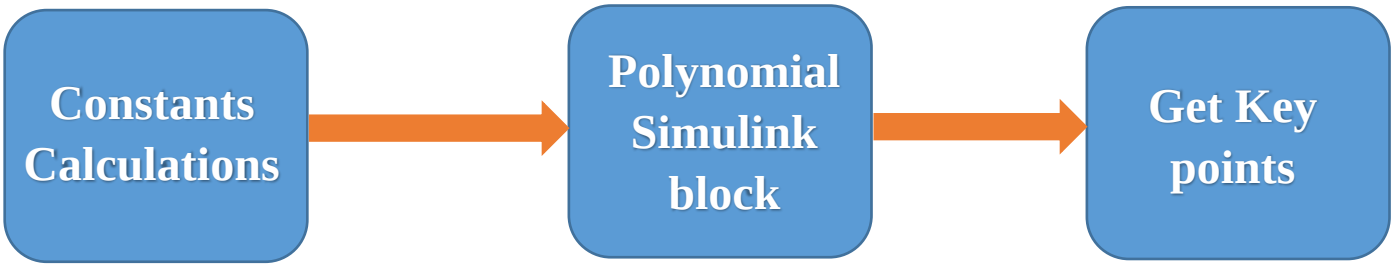


Figure 15. Trajectory Planning with sigmoid block

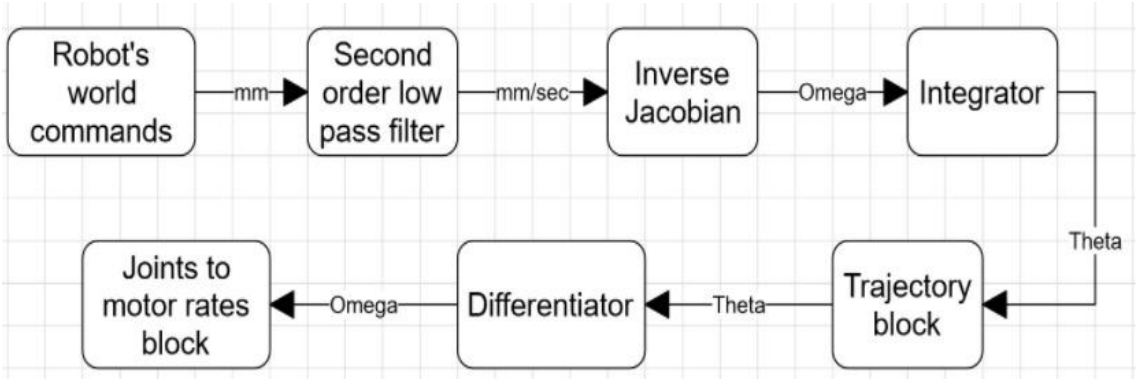


We create a quintic Cartesian space trajectory block by first calculating the constants given the targeted robot's position and time.

Then passing these constants to the polynomial Simulink block.

Finally, obtaining the key points to send them to the robot.

Figure 16. Trajectory Planning block Diagram



The robot's "World commands" are first passed through the "Second order low pass filter" Simulink block, which outputs world rates.

Which are then passed to the "Inverse jacobian block" to get the joint rates.

Which are then integrated to get the final thetas.

Which are then pass to the "Trajectory block" to get the key points.

Then passed over a "Differentiator" to get the new joint rates.

Finally, passed to the "Joints to motor rates block"

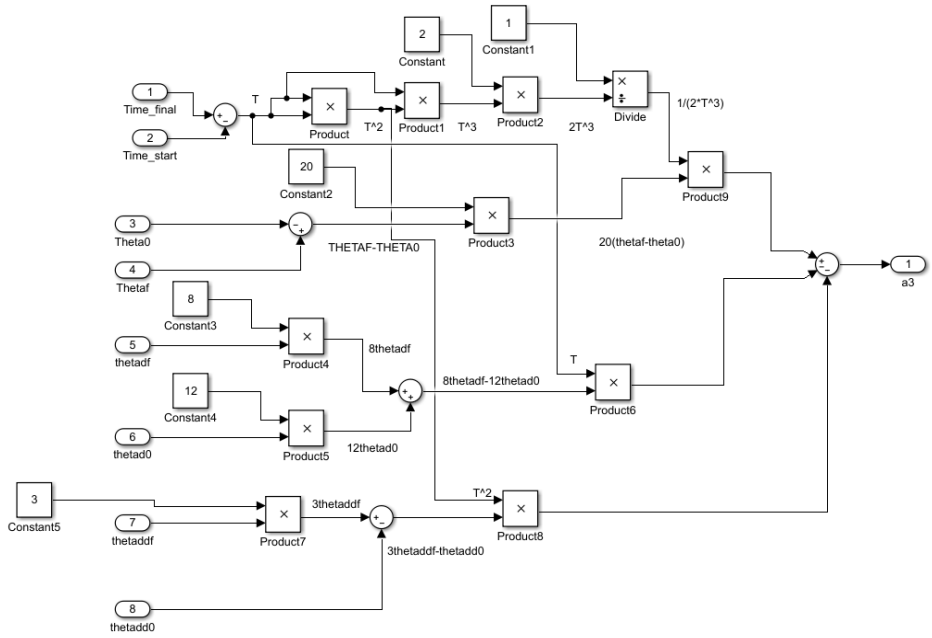


Figure 17. Constant 5 block Diagram

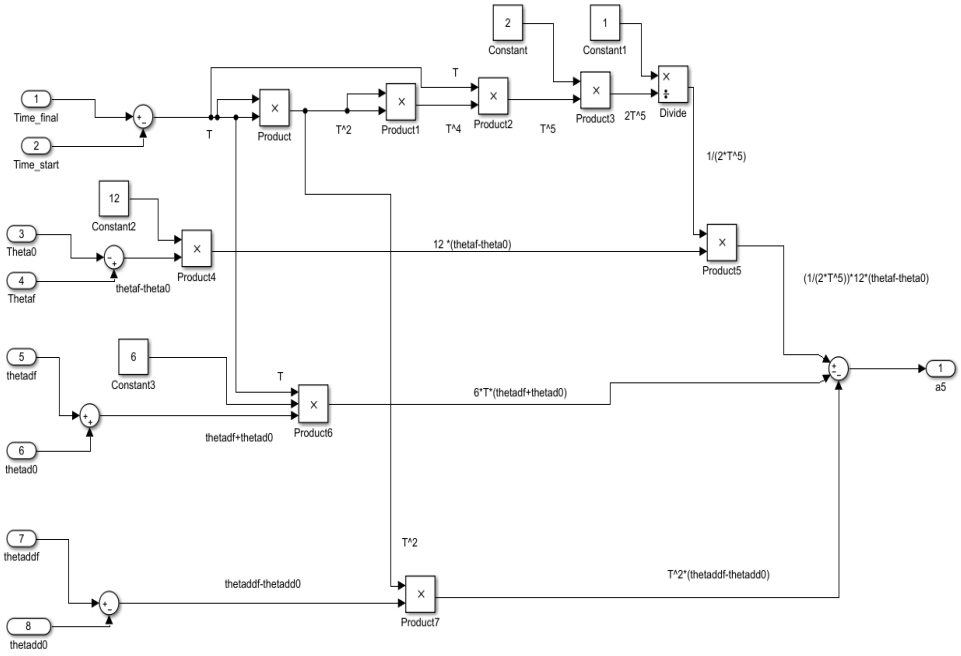


Figure 18. Constant 6 block Diagram

The Signal Builder block allows you to create interchangeable groups of piecewise linear signal sources and use them in a model. You can quickly switch the signal groups into and out of a model to facilitate testing.

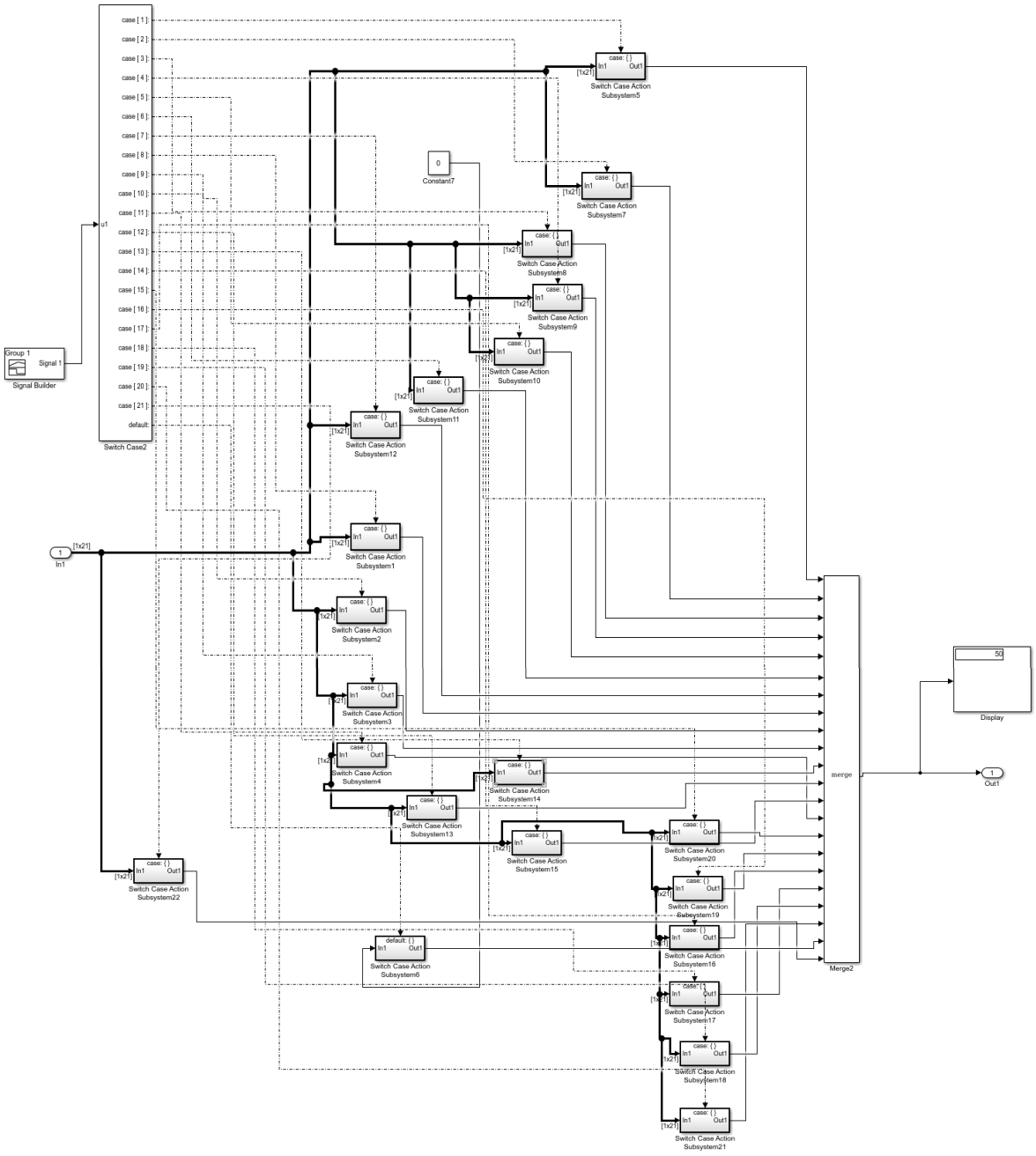


Figure 19. Signal Builder block diagram

CHAPTER FIVE: PROJECT'S CHALLENGES

In every project, you may face problems that can be solved easily and others that are difficult and take a long time to be solved, in this chapter we will discuss the problems that we faced and how we managed to solve these problems.

- **Challenges (1):**

5.1. Transferring the robotic arm

Finding a suitable workspace to work at was very necessary to help us to control the robotic arm safely and avoid any undesirable collision, so we searched for a suitable place to work at, and it took us somedays to get the approval from the university headmasters to transfer the robot to the chosen place, and after getting the approval we contacted a truck driver to transfer the robotic arm safely without any damage.

5.2. Testing the old pc with the robotic arm

To make a human-robot interaction the first thing we should do is testing whether the robot responds to the pc orders or not, so we tried to do this test but the pc wasn't working and we tried to repair it by taking it to a computer services shop and they responded that the pc's motherboard isn't working, then we tried to buy a new motherboard but we weren't able to find it as it was very old version, so we requested another pc from the university and we were provided by it, after getting the new pc we transferred all the data related to the robot into the new pc.

5.3. Transferring the data from the old pc to the new one

After transferring the data related to the robot into the new pc, we wanted to test whether the robot responds to the pc orders or not, so we opened the softwares that are responsible for controlling the robot and we found that the versions of these softwares were old, we downloaded the new versions and we worked with given QUARC software that is compatible with MATLAB software, as the MATLAB software is the main software to control the robotic arm, then we started to test whether the robot responds to the pc orders or not and we found that it was working.

5.4. Robotic arm homing

Before performing a certain task, we should test whether the robot returns to its home position or not, to do this we installed ROBCOMM3 software and we found that it doesn't work with our installed windows (WINDOWS 10), so we downloaded a virtual machine then we installed WINDOWS XP on it to be able to run ROBCOMM3, but unfortunately it didn't work again, so we decided to find another way to adjust robot's homing which was downloading PUTTY software and then connecting the

communication cable from the control unit of the robotic arm to the pc, after testing this method, the robot's homing worked successfully.

5.5. Controlling problem

After we were able to control the robot, The robot was performing well until suddenly it didn't respond to any order, so we tried to find the problem and we thought that control card (Q8) was damaged, so we contacted the manufacturing company (Quanser) to see if the card is damaged or not, and they told us to check the main fuse of the control card and after checking we found that the fuse is damaged so we replaced it with a new one and after testing again the robot worked well.

5.6. Robot's noise

While the robot was operating it was producing a high noisy sound also the robot's temperature was increasing during operation, so to overcome this problem we lubricated all the robot's internal gears and belts.

5.7. Uncontrolled speed problem

One of the problems that faced us while working was the high speed of the robot's joints which causes the robot to turn off due to safety conditions, to solve this problem we used trajectory planning to control the speed of the robot and the path that the robot will move on.

5.8. Force sensor configuration

When we received the force sensor from the university (KD24S-50N) it didn't contain any data sheet or any M files (MATLAB) to work with, so we searched on the internet to find the data sheet of this force sensor and we opened the manufacturing company website and we were able to find the data sheet, now we needed to search for M files to control the force sensor so we tried to contact the main manufacturing company to help us with the force sensor's configuration but unfortunately they weren't helpful, but we contacted one of their branches and they responded that to control the force sensor we have to control the amplifier of the force sensor first, and they don't have the M files of our amplifier (GSV-3USB) and they advised us to start building M files for our amplifier from scratch, so we were able to find M files for another amplifier (GSV-8USB) and we started to understand the files and edit these files to be compatible with our amplifier and after implementing M files for our amplifier(GSV-3USB) it worked properly.

5.9. MATLAB problem

We were forced to use MATLAB version 2017a because the installed QUARC wasn't compatible with any newer version more than 2017a, this caused some problems which were:

- We were forced to download an old version of Mingw (compiler application) and this compiler application wasn't available in MATLAB libraries, so we contacted QUANSER to send us a compiling software and thanks for them they send us two compiling softwares (Mingw, Windows SDK compiler), after installing these two softwares, the MATLAB compiled without errors. (We used Mingw for compiling force sensor's M files).
- We couldn't use trajectory planning block diagram as this block diagram is only available with MATLAB 2019a and newer versions only, so to overcome this problem we built a trajectory planning model from scratch, so we were able to solve the problem, but it took us a lot of time and efforts.

❖ Challenges (2):

5.10. Robot's Limitations

The first problem was related to the robot itself as the robot can move only 9 cm along the x-axis direction, So the space of the assembly stage where the gears and their base are located was narrow, but we were able to handle this problem by reducing the gears and the base's dimensions.

5.11. Force Sensor & Compiler

Another problem faced us and it was related to the force sensor, the compiler that is compatible with the force sensor is Mingw, while the project's compiler is windows SDK, this problem was the main reason why the force sensor didn't work, but after many modifications on the code we managed to solve this problem.

5.12. Vision Module

When implementing the vision module, we passed through many problems:

- one of these problems appeared when we tried to integrate both the vision module which implemented using MATLAB and Kinect camera with the robot in order to control it, we found that the Kinect camera is not a windows version, so integrating this with the robot would take a lot of time, so we decided to build our vision module using python and a web camera.

- Then another problem appeared after finishing the vision module and integrating it to the robot in order to control it, the problem was that the camera detects the robot's frame curves as a circle so this caused a problem as we wanted the camera to detect only the gears circles, so we decided to find another way to detect the gears centers which was the detection of the gears contour and their centers, and after implementing this method it worked very well.

5.13. Python & Quarc Communication

Finally, the last problem was related to the communication between the python file and Quarc, as the robot's file didn't accept the MATLAB function that calls the python vision module file, so after too many trials to do this connection, we were able to overcome this problem by making a TCP/IP connection between 2 MATLABs, one for the robot's files and the other that contains the function that calls the python file, and after testing it was successful.

CHAPTER SIX: THEORETICAL APPROACH

The robot used (Catalyst-5t) is a 5DoF robot, that consists of 5 revolute joints presented in figure below.

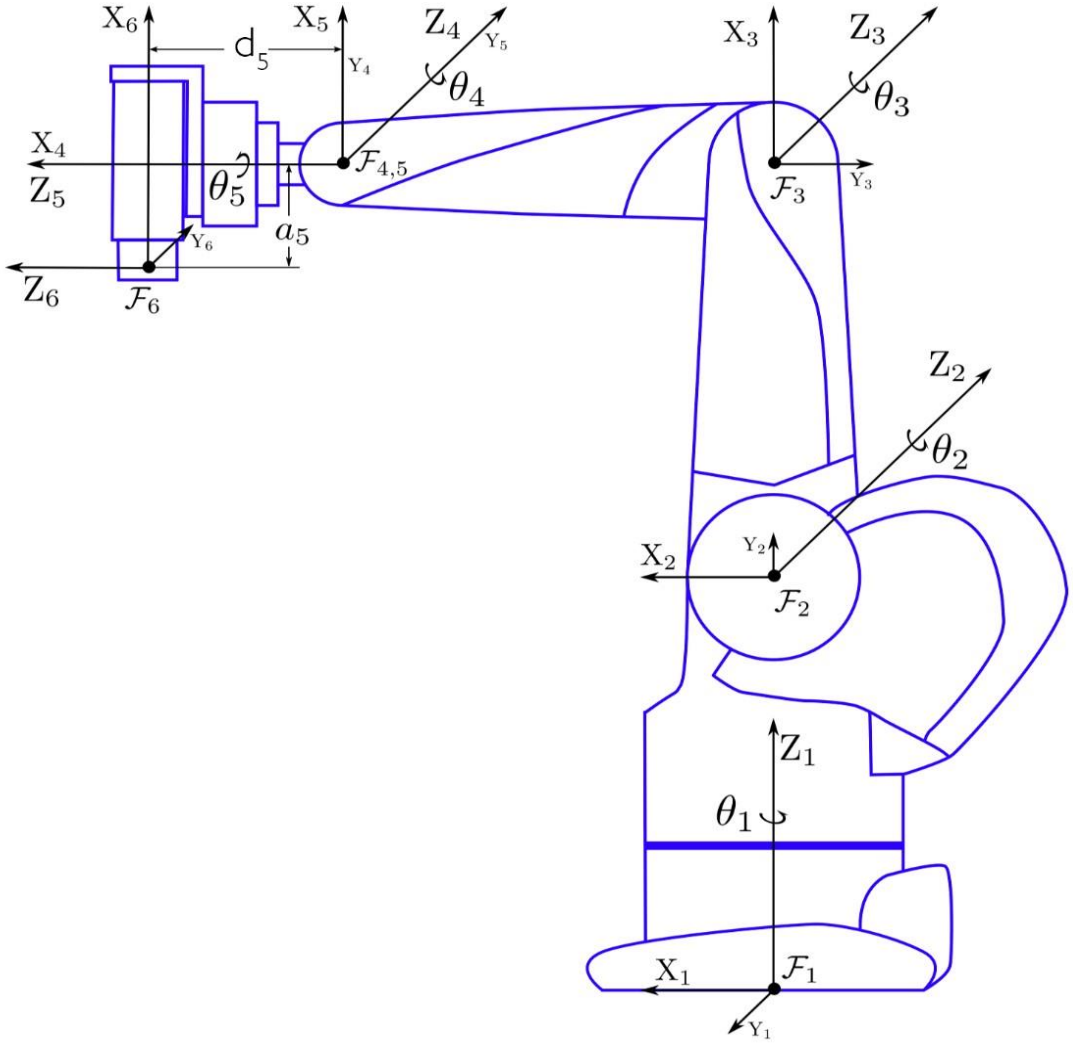


Figure 20. DH Axes - Ready Position

- a i is link length which is defined as the distance between Z i and Z i+1 along X i+1
- α i is the link twist which is defined as the angle to bring Z i parallel with Z i+1 rotated about the positive X i+1 axis
- d i is the joint offset which is defined as the distance along Z i where X i+1 intersects Z i
- θ i is the joint angle and is defined as the angle to bring X i parallel with X i+1 rotated about the positive Z i axis.

6.1. KINEMATIC MODELING

The dynamic modelling of a manipulator robot with n degree of freedom (DOF.) can be divided in four steps: Direct Kinematics, Inverse Kinematics, Differential Kinematics and Dynamics.

6.1.1. Direct Kinematics

The objective of the direct kinematics is to determine the accumulative effect that comes from the set of variables of each link, that is, to determine the position and orientation of the end-effector. The analysis of the Direct Kinematics was made using the Denavit-Hartembrerg convention as follows:

$$A_i = [Rot]_{(zi,\ \theta i)}\,[Tras]_{(zi,\ di)}\,[Tras]_{(\mathrm{xi},\ ai)}\,[Rot]_{(\mathrm{xi},\ \alpha i)}$$

$$A_i = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha i)\sin(\theta_i) & a_i\cos(\theta i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Joint	θ_i	d_i (mm)	a_i (mm)	α_i	Range	Home
1	θ1	254	0	90	±180	0
2	θ2	0	254	0	0 : 110	90
3	θ3	0	254	0	-125 : 0	-90
4	θ4	0	0	90	±110	90
5	θ5	116	58.7	0	±180	0

Table 4. DH parameters of the Catalyst-5

After some calculations, the end-effector position vector is defined by:

$$X= \frac{1}{2} d_5 \sin(\theta 1 + \theta 3 + \theta 4 + \theta 2) + \frac{1}{2} d_5 \sin(-\theta 1 + \theta 3 + \theta 4 + \theta 2) + \frac{1}{2} a3 \cos(-\theta 1 + \theta 2 + \theta 3) + \frac{1}{2} a3 \cos(\theta 1 + \theta 2 + \theta 3) + \frac{1}{2} a2 \cos(-\theta 1 + \theta 2) + \frac{1}{2} a2 \cos(\theta 1 + \theta 2)$$

$$Y= \frac{1}{2} d_5 \cos(-\theta 1 + \theta 3 + \theta 4 + \theta 2) - \frac{1}{2} d_5 \cos(\theta 1 + \theta 3 + \theta 4 + \theta 2) + \frac{1}{2} a3 \sin(\theta 1 + \theta 2 + \theta 3) - \frac{1}{2} a3 \sin(-\theta 1 + \theta 2 + \theta 3) + \frac{1}{2} a2 \sin(\theta 1 + \theta 2) - \frac{1}{2} a2 \sin(-\theta 1 + \theta 2)$$

$$Z=-d_5 \cos(\theta 3 + \theta 4 + \theta 2) + a3 \sin(\theta 2 + \theta 3) + a2 \sin(\theta 2) + d_1$$

6.1.2. INVERSE KINEMATICS

The inverse kinematics main problem can be reduced to both the calculus of the position and orientation inverse kinematics, where a geometrical approach to the problem solution is based in the Method proposed by Sponge.

6.1.3. DIFFERENTIAL KINEMATICS

In the previous steps, the dynamics related with the position of the end-effector as well as the location of the joints and its position with respect to the reference frame, therefore, the position problem will be changed to a speed analysis problem of a serial manipulator.

$$\dot{X} = J\dot{q}$$

The differential kinematics is defined by the next equation: where J is the Jacobian matrix. This matrix depends on the robot configuration and robot degrees of freedom (DOF).

6.1.4. DYNAMICS

The dynamic equation of an n-DOF in the manipulator can be defined as:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

6.2. MODELING AND SIMULATION

6.2.1. SOLIDWORKS MODEL

To begin the simulation, first there must be a 3D model to be the base for the simulation. First, there was a file at ASU, but it was a non-editable file, and there were not any online models for the Catalyst-5. After several unsuccessful trials to recreate the model, we were able to contact someone who worked on the same robot in Mexico and he provided us with the solidworks model.

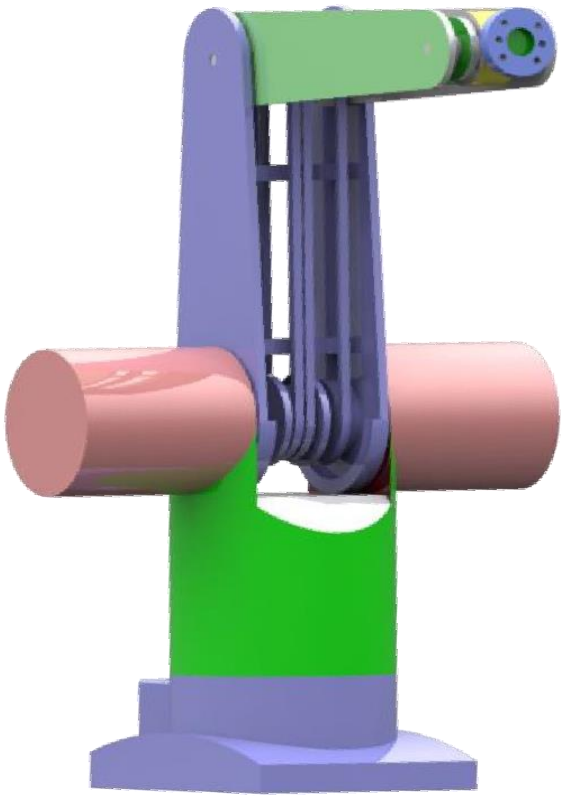


Figure 21. Solidworks Model

6.2.2. MATLAB / SIMSCAPE MULTIBODY

6.2.2.1. Simscape base model

After getting the solidworks model, it needed to be imported to the MATLAB/simscape multibody, so using a method created by MATLAB, it automatically generates a simscape model using the solidworks model.

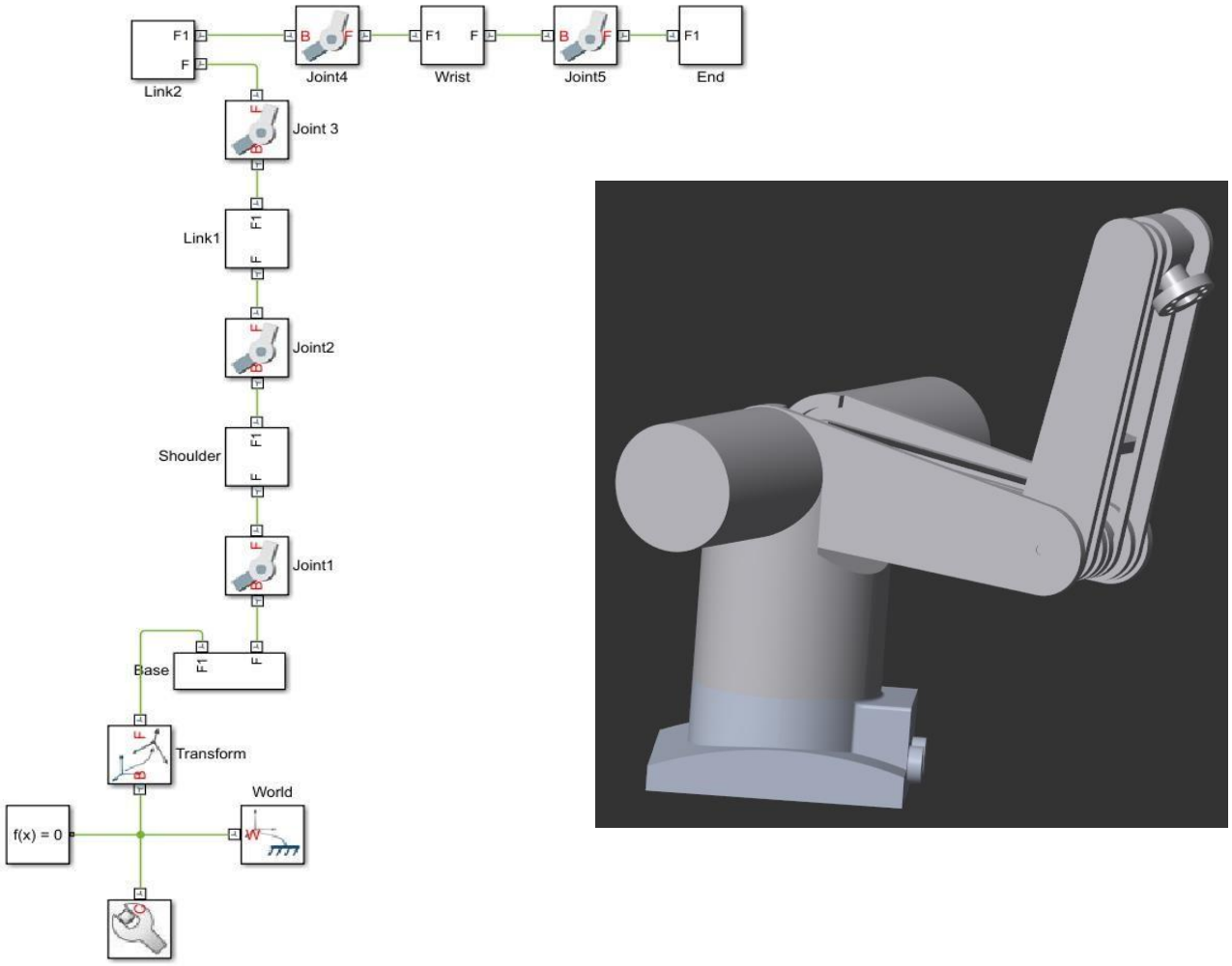


Figure 22. Simscape model and render

6.2.2.2. Simple angular control

After getting the simulink model, we added angle inputs to all of the joints to move the body to the desired angles as shown in the figure below.

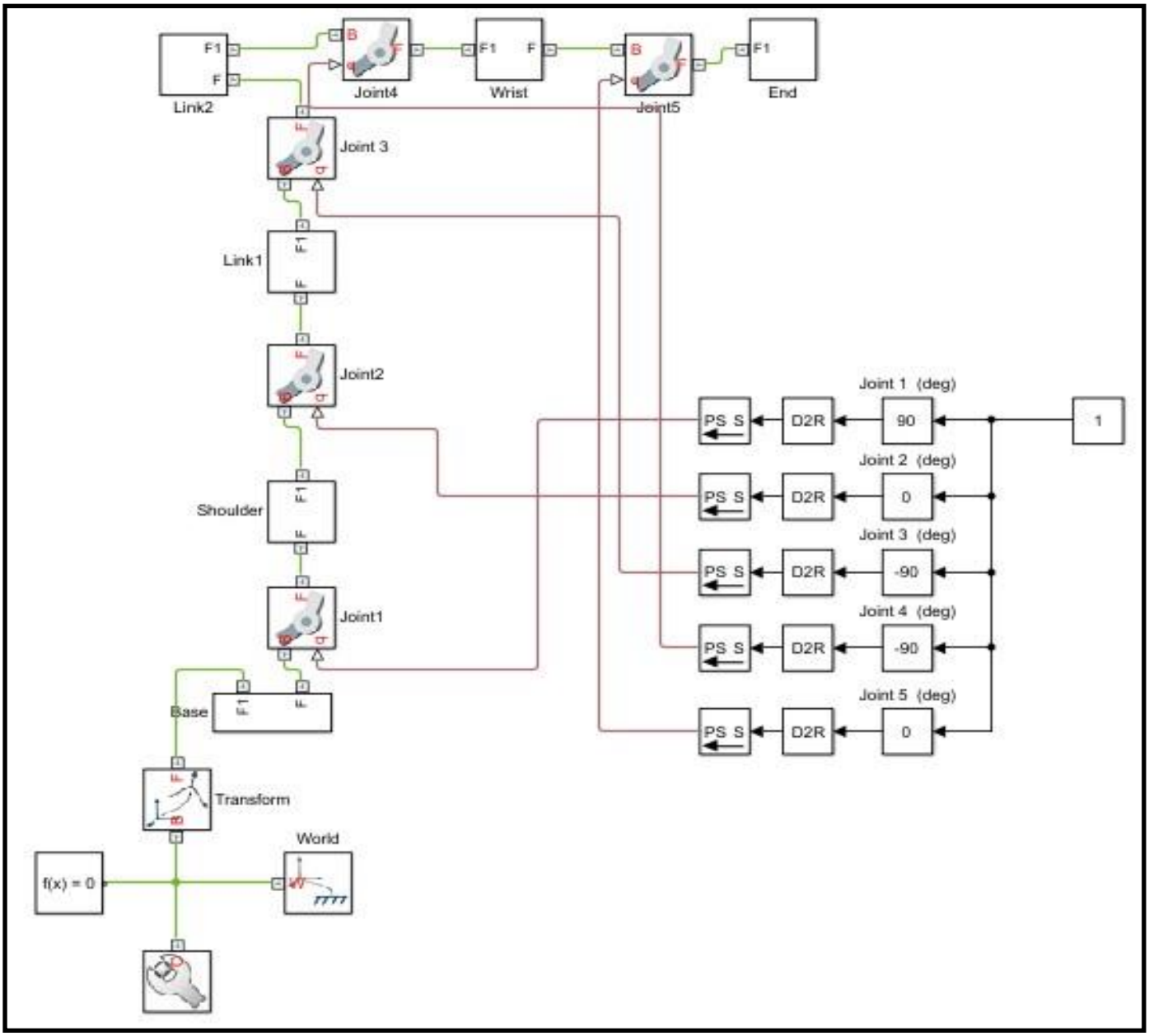


Figure 23. Angular control model

CHAPTER SEVEN: PROJECT MANAGEMENT (2)

7.1. CATALYST ROBOTIC ARM TASKS:

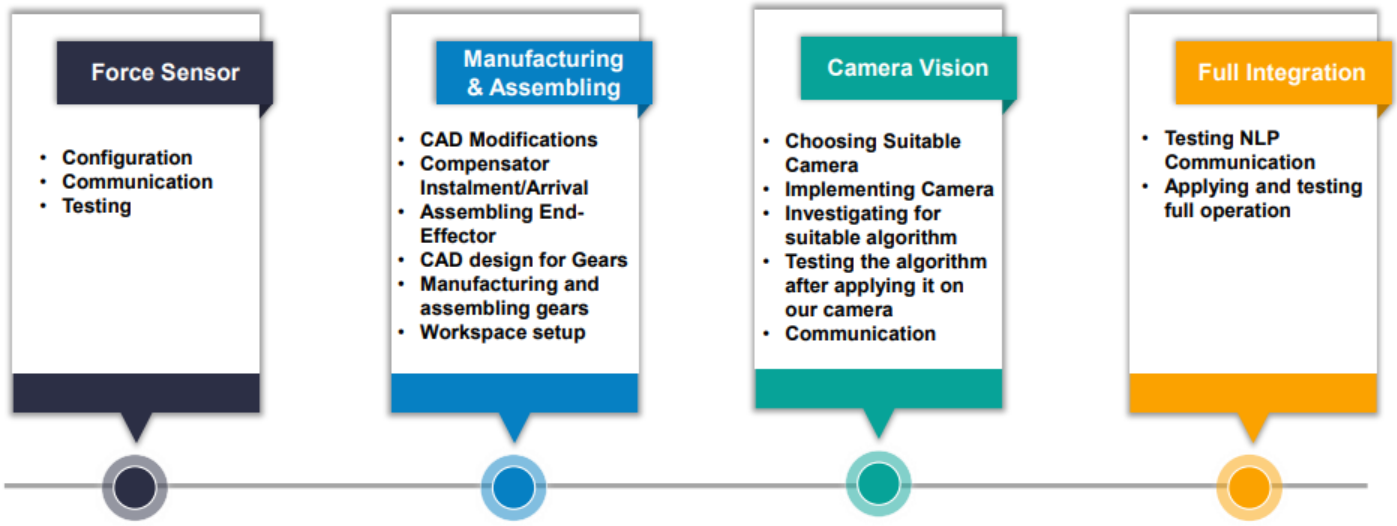


Figure 24: Main Tasks

7.2. GANTT CHART (TIME PLAN):

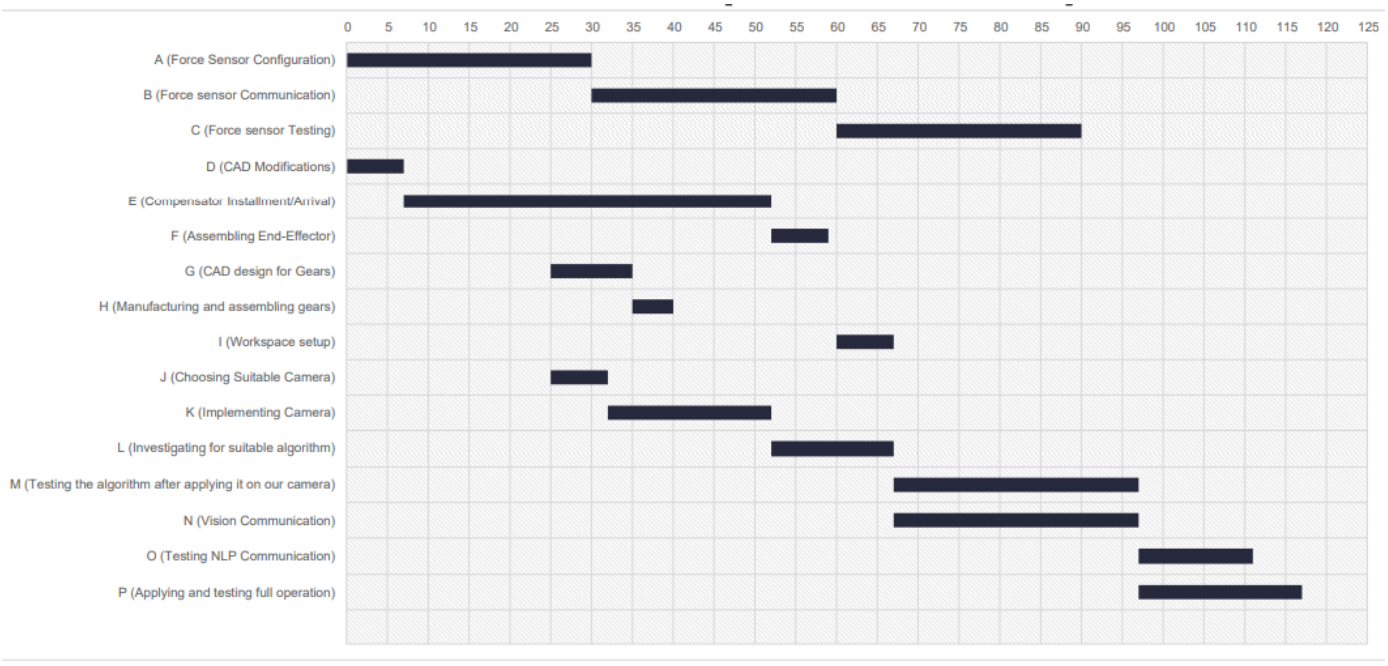


Figure 25: Gantt Chart

7.3. ACTIVITIES TIME PLAN TABLE:

Activity	From (day)	Duration (day)	To (day)
A (Force Sensor Configuration)	0	30	30
B (Force sensor Communication)	30	30	60
C (Force sensor Testing)	60	30	90
D (CAD Modifications)	0	7	7
E (Compensator Instalment/Arrival)	7	45	52
F (Assembling End-Effector)	52	7	59
G (CAD design for Gears)	25	10	35
H (Manufacturing and assembling gears)	35	5	40
I (Workspace setup)	60	7	67
J (Choosing Suitable Camera)	25	7	32
K (Implementing Camera)	32	20	52
L (Investigating for suitable algorithm)	52	15	67
M (Testing the algorithm after applying it on our camera)	67	30	97
N (Vision Communication)	67	30	97
O (Testing NLP Communication)	97	14	111
P (Applying and testing full operation)	97	20	117

Table 5: Activities Time

CHAPTER EIGHT: CAMERA VISION

8.1. INTRODUCTION:

In this chapter we are required to design a camera vision system, in order to control our robotic arm to do its specific function, by detecting the center location of the gears to send it to its base.

This could be done by many methods, so as we did, we will illustrate our three stages that we had made till we finally reached our best module system to control our robotic arm.

So, in this section we will illustrate each method we tried to localize our gears until we reached the optimum one.

8.2. MATLAB WITH KINECT CAMERA:

Firstly, we tried to implement our vision module by MATLAB and Kinect camera and we were able to do this by m-file code but the problem appeared when we tried to integrate this vision module with our robot, as the robot is controlled by Simulink blocks which is not compaitable to our Kinect camera, as the camera was Xbox Kinect not the windows one, so it would take a lot of time and effotrt to configure it by the Simulink blocks as we had to build these blocks from scratch , so we decided to use web cam instead of the Kinect camera as the web camera is easier in configuration and fixation.

Now we will show the vision module that we were able to do using Matlab and Kinect camera, including gears circles detection and depth.

The following images illustrate the circle detection and depth

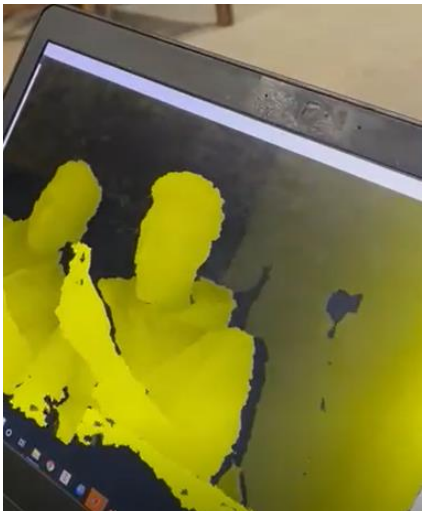


Figure 26: this image shows the depth illustration

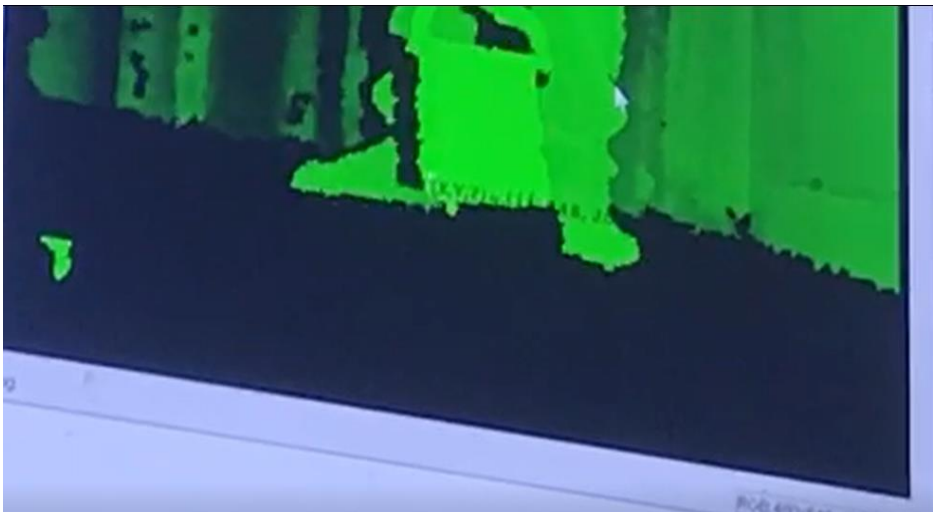


Figure 27: another image to show the depth including the x, y, z coordinates of the object

Figure 28: Kinect camera captures the gears

Figure 29: circle detection using MATLAB and Kinect camera

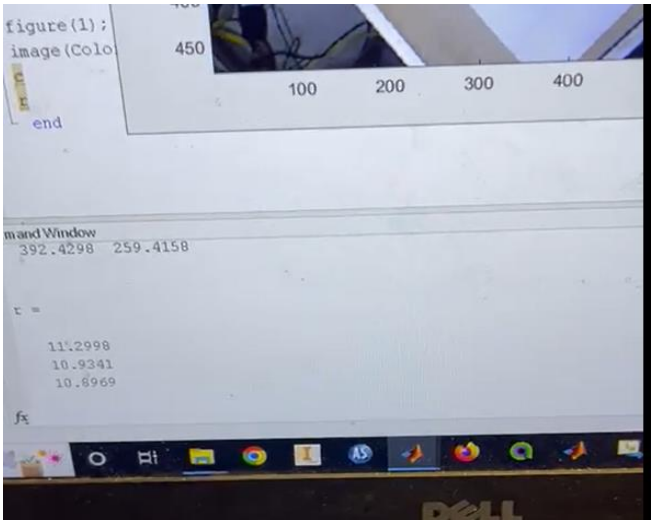


Figure 30: circles radii

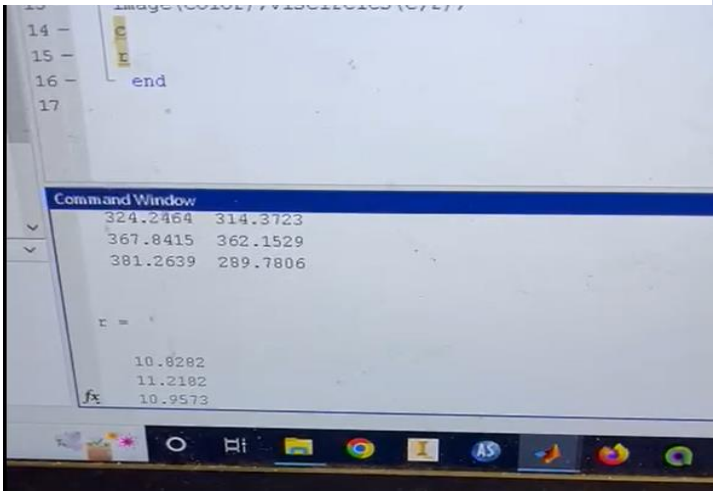


Figure 31: x, y coordinates of each circle

❖ CODE:

```
1. Colorvid = videoinput('kinect',1,'RGB_640x480');
2.
3.  while(1)
4. Color = getsnapshot(Colorvid);
5.
6.
7. %call imfindcircles
8. [c,r] = imfindcircles(Color,[10,30]);
9. %display detected circles
10.
11.      figure(1);
12.      image(Color);viscircles(c,r);
13.      c
14.      r
15.       end
```

8.3. PYTHON WITH WEB CAMERA (CIRCLE DETECTION MODULE):

In this method we were able to to detect the gears centers and detecting their x and y coordinates and integrating these informations with the robot to be able to do the required task which is assembly of these gears to their base, but the only problem was that when we fixed the camera it detects the gears circles centers and the robot circles centers and we want the camera to detect only the gears centers, which we were able to do it in a later stage.

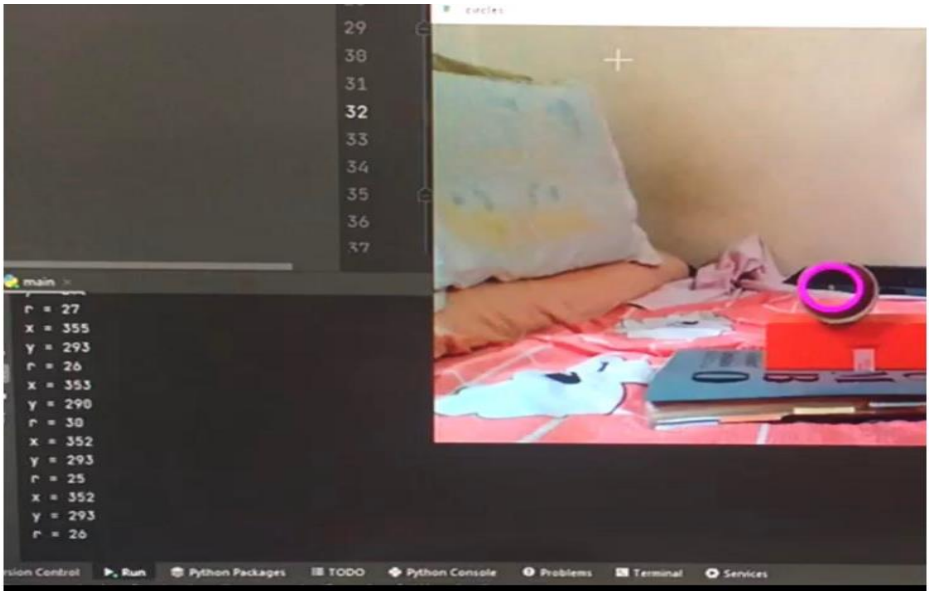


Figure 32: This image shows the circle detection using web camera and python

Figure 33: The two captured gears

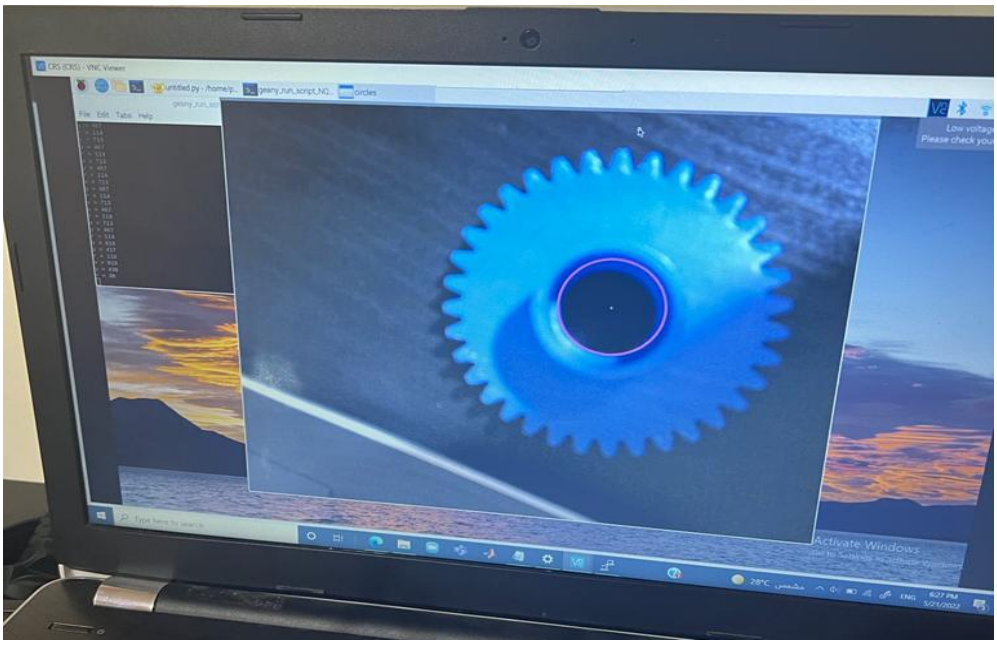


Figure 34: Another implementation of web camera and python vision module detecting the circle's radius and its x, y coordinates

❖ CODE:

```
16.      import cv2 as cv
17.      import numpy as np
18.
19.      # define a video capture object
20.      videoCapture = cv.VideoCapture(0)
21.      prevCircle = None
22.      dist = lambda x1, y1, x2, y2: (x1 - x2) ** 2 + (y1 - y2)
   ** 2
23.
24.      while True:
25.          # Capture the video frame by frame
26.          ret, frame = videoCapture.read()
27.          if not ret: break
28.
29.          # Convert to grayscale.
30.          grayFrame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
31.          # Gaussian Blur using 17 * 17 kernel.
32.          blurFrame = cv.GaussianBlur(grayFrame, (17, 17), 0)
33.
34.          # Apply Hough transform on the blurred image.
35.          circles = cv.HoughCircles(blurFrame,
   cv.HOUGH_GRADIENT, 1.2, 100,
36.                                    param1=100, param2=30,
   minRadius=0, maxRadius=650)
37.
38.          # Draw circles that are detected.
39.          if circles is not None:
40.              # Convert the circle parameters a, b and r to
   integers.
41.              circles = np.uint16(np.around(circles))
42.              chosen = None
43.              for i in circles[0, :]:
44.                  if chosen is None: chosen = i
45.                  if prevCircle is not None:
46.                      if dist(chosen[0], chosen[1],
   prevCircle[0], prevCircle[1]) <= dist(i[0], i[1],
   prevCircle[0],
47.
   prevCircle[1]):
48.                          chosen = i
49.              cv.circle(frame, (chosen[0], chosen[1]), 1, (0,
   100, 100), 3)
50.              cv.circle(frame, (chosen[0], chosen[1]),
   chosen[2], (255, 0, 255), 3)
51.              prevCircle = chosen
52.
53.          # Display the resulting frame
54.          cv.imshow("circles", frame)
55.          # Print detected circles coordinates
56.          print("x =",chosen[0])
57.          print("y =",chosen[1])
58.          print("r =",chosen[2])
59.
60.          # the 'q' button is set as the quitting button you
   may use any desired button of your choice
61.          if cv.waitKey(1) & 0xFF == ord('q'): break
```

```
62.      videoCapture.release()
63.      cv.destroyAllWindows()
```

8.4. PYTHON WITH WEB CAMERA (GEARS CONTOUR DETECTION):

In this stage, we tried to depend on the boundary of the object to detect its center and area, instead of the circle function. So, in this code we have created a contour detection around the gears to detect its outer boundries and in order to increase the accuracy and limit the error, we put an upper and lower limits for the contour based on the color of the gears, in order to be able to detect the center (x,y) of the robot and its area.

So, to sum it up, contours detection is a process can be explained simply as a curve joining all the continuous points (along with the boundary), having same colour or intensity. The contours are a useful tool for shape analysis and object detection and recognition, and so we decided to integrate it with our system to control the robotic arm to do its assembly function

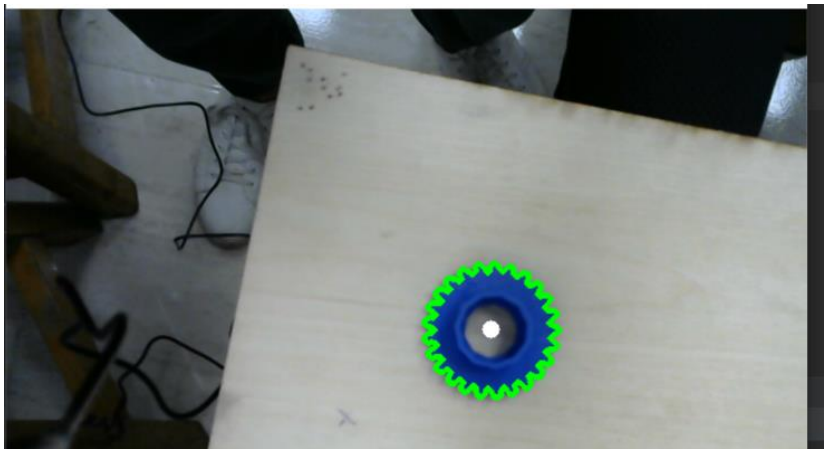


Figure35: In this image, the code detects the object boundaries

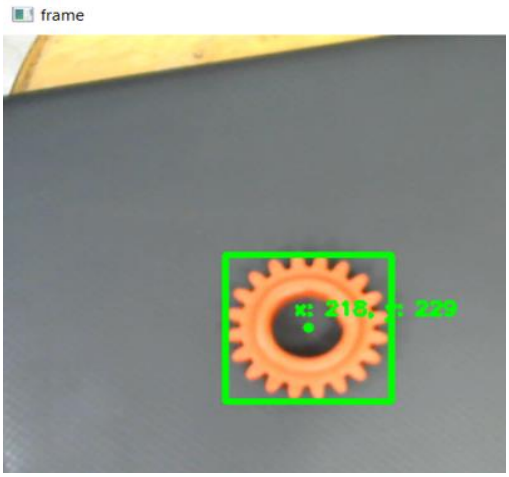


Figure 36: This image shows another type shape of object detection, showing its x, y coordinates

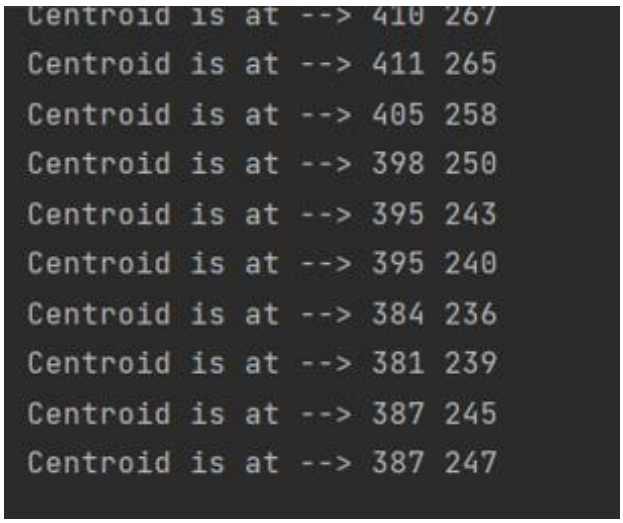


Figure 37: This image shows the centre that is detected by the code after detection object's contour.

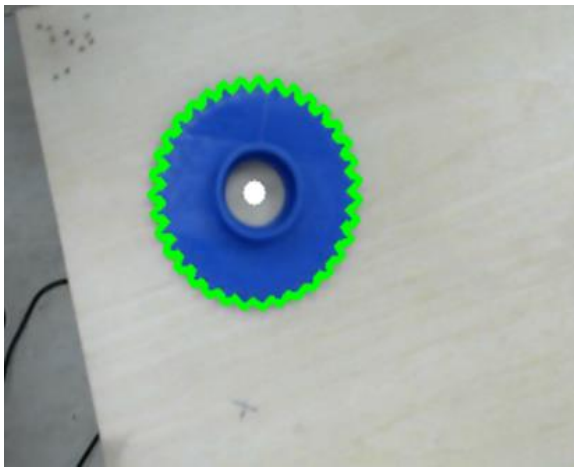


Figure 38: Another image showing the object detection of a different size gear.

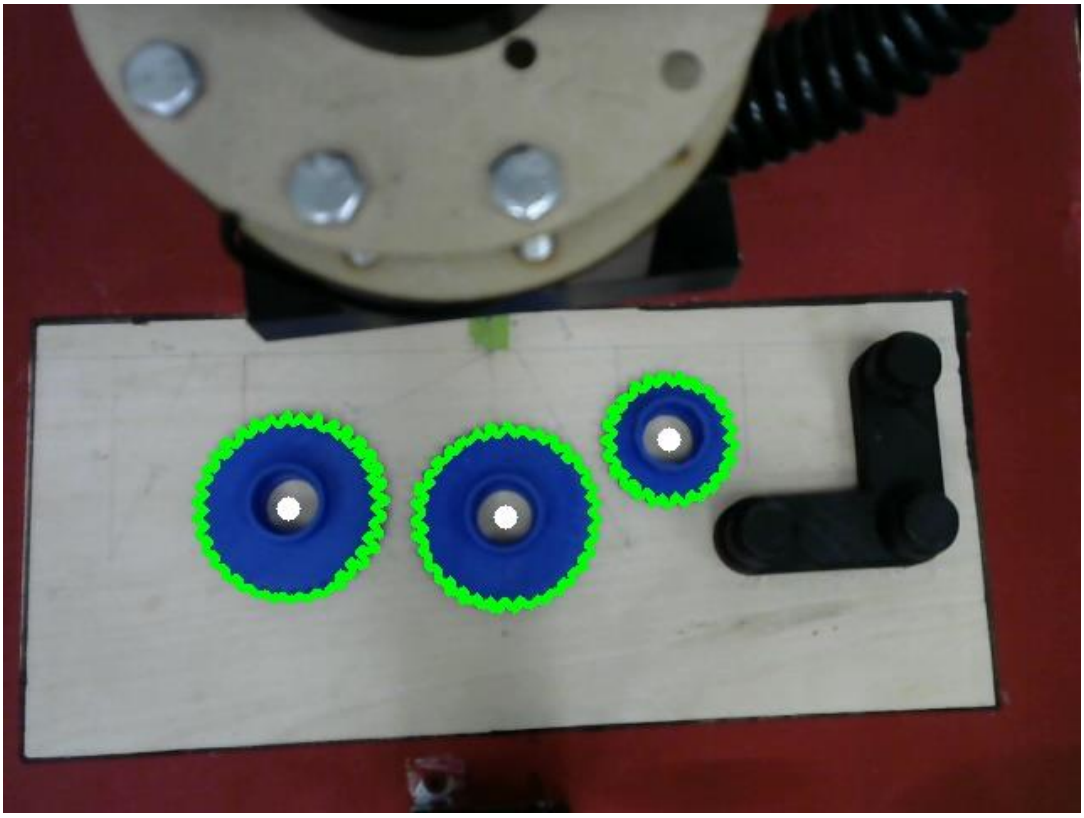


Figure 39: Camera's Vision in the python contour code

❖ CODE:

```
64.      import cv2
65.      import numpy as np
66.      import imutils
67.      import array as arr
68.
69.
70.
71.      def camera():
72.          cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
73.          cap.set(3, 640)
74.          cap.set(4, 480)
75.
76.          ret, frame = cap.read()
77.          # a = arr.array()
78.          hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
79.
80.          lower_blue = np.array([90, 60, 120])
81.          upper_blue = np.array([121, 255, 255])
82.
83.          mask = cv2.inRange(hsv, lower_blue, upper_blue)
84.
85.          cnts = cv2.findContours(mask, cv2.RETR_TREE,
   cv2.CHAIN_APPROX_SIMPLE)
86.          cnts = imutils.grab_contours(cnts)
87.          coordinates=list()
88.          for c in cnts:
```

```
89.              area = cv2.contourArea(c)
90.              if area > 3000:
91.                  cv2.drawContours(frame, [c], -1, (0, 255,
   0), 3)
92.
93.                  M = cv2.moments(c)
94.
95.                  cx = int(M["m10"] / M["m00"])
96.                  cy = int(M["m01"] / M["m00"])
97.                  #print("Centroid is at -->", cx, cy)
98.                  coordinates.append([cx, cy,area])
99.                  cv2.circle(frame, (cx, cy), 7, (255, 255,
   255), -1)
100.         cv2.imwrite("mask.jpg",frame)
101.         #cv2.imshow("result", frame)
102.
103.         return coordinates
104.
105.     #  print("Area is -->", area)
106.
107.
108.     print(camera())
```

CHAPTER NINE: ROBOT'S FUNCTION & FEATURES

9.1. PURPOSE OF ROBOT:

The Thermo Scientific CRS CataLyst-5 Express Robot enables to automate the transfer of gear kit between table and their compatible base to be meshed together. The CRS CataLyst manipulator provides the reliability, precision, and dexterity of Thermo's CRS CataLyst robot in an installed workspace that makes it easy for handling the loading and unloading of several, gear sets with the help of a vision system to perceive the environment for object's detection.

Figure 40: CRS Catalyst-5 Express integrated with workspace and a web cam

9.2. BENEFITS OF THE CRS CATALYST:

The Thermo Scientific CRS CataLyst-5 Express Robot is ideal for applications that require complex movements, such as dispensing or machine loading and unloading. For applications requiring flexible movement without sacrificing speed or reliability, the CRS CataLyst-5 provides those movements and offers a linear track option for tending multiple machines. Features increased throughput and efficiency, designed to run 24/7; automatic homing: lets you start moving payloads on power up. Easy to integrate: advanced software reduces programming time Linear Track Features.

Users of our robotic platform CRS CataLyst 5 enjoy the following benefits:

- Continuous, error-free operation.
- Precise, repeatable motion.
- The ability to handle a vast range of wide base with integrated hollow-shaft objects.
- The ability to automate loading and unloading of different size gear sets.
- Safety features that meet OSHA requirements.
- Easy to use, yet powerful method development and scheduling.

9.3. CRS CATALYST FEATURES AND FUNCTIONALITY:

Figure 41: Main Hardware Features of the Thermo CRS Catalyst 5

- The **robotic arm** gets gears from table, and puts gears into their specified base unit.
- The **robot base** supports the arm, allows for movement of whole robot, and houses the electronics that power and control the system.
- The **gear-shaft servo gripper** holds the gear shafts as the arm moves them. Its features enable the CRS CataLyst to control the strength of the grip and the distance between the fingers, allowing the robot to apply the optimum gripping force for each type of gear and to confirm that it has a gear in its grasp.
- The **gear base** supports the gears that loaded on it to maintain gears meshing together.
- The **gears** placed on the table waiting for the CRS catalyst in order to pick them.
- The **RCC compensator** compensates for part positioning errors, fixture misalignments and variances in part tolerances.
- The **table** provides workspace for the gears and their base to be placed as well as the camera support for observing the environment.
- The **web camera** perceives the workspace and detects the gear's coordinates in relation to the robot EE coordinates.
- The **parking bracket** provides a parking location for the arm, a known physical position that enables the CataLyst 5 Express to establish an origin in space, or home, for the arm.

  Note: The arm must be homed before it can move the gears.

- The **E-Stop control** provides an emergency stop button that, when pressed, immediately cuts power to the arm. The control also provides arm status indicators, and a brake release that enables operators to manually move the arm into its parking bracket.

The working demonstration begins with the robotic arm in its parked position. The arm then unparks and does a self check. After the arm is homed it grabs the pinion gear from the loading stage and places it into base position 1. After this the arm moves the gripper to loading stage again. Then the arm moves the second gear to position 2. From here the gripper ensures the gear is dropped and returns to get the third gear. Next the arm moves the last gear into position 3 to be meshed with the assembled gears. Then the gripper goes back to position 1 to turn the pinion gear so as to maintain fine mesh of the gears. The gripper is returned to the loading stage where there is no gears to start the disassembly stage. It goes back to start disassemblying from the last gear placed on base in reverse sequence, starting from gear in position 3, then 2 and finally takes the pinion gear in position 1 back to its initial location. After the test run is completed the robotic arm returns to the park position.

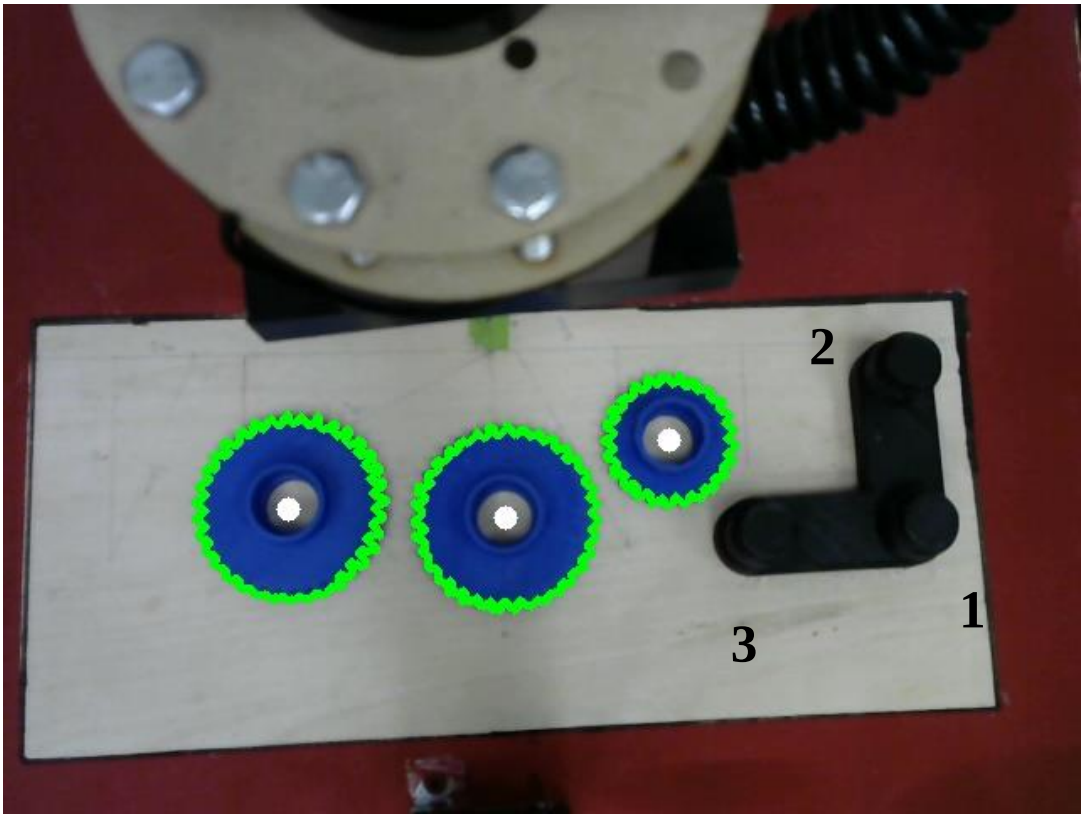


Figure 42: This image shows Gears and positions (1,2 & 3)

CHAPTER TEN: WORKSPACE SETUP

10.1. PREPARING THE TABLE:

The table which CRS CataLyst 5 is placed, should conform to the layout as shown:

Figure 43: Positioning of the Thermo CRS Catalyst 5 and the Table

We select a table of 50 x 50 x 100 cm dimensions to be wide enough to provide clearance between the side of the robotic platform CRS Catalyst-5 and the side edge of the table.

A Wooden support is fixed on the upper side of the table to provide fixation for the camera for a clear, wide vision of the whole workspace.

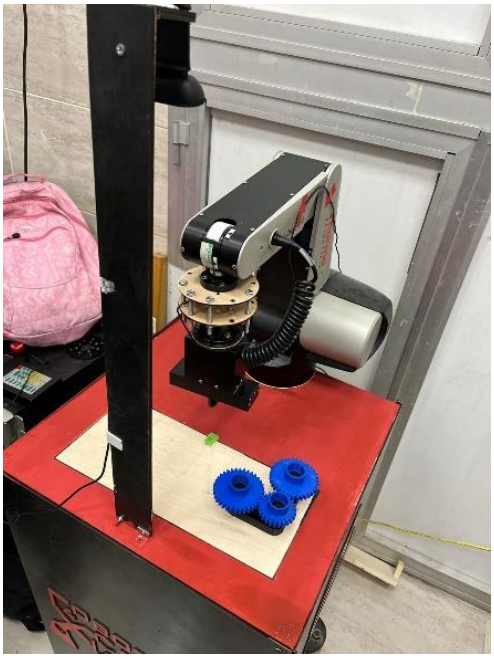


Figure 44: Full view of the table's workspace with the robotic arm

We place the table against a wall to restrict access to the rear side of the CRS CataLyst and made sure the table is level, fixed in a place, and clean.

Figure 45: Front scene of the CRS Catalyst manipulator workstation

CHAPTER ELEVEN: COMMUNICATION SYSTEM

11.1. COMMUNICATION:

- In this section we will talk about the communication method used to collaborate between the camera and the robotic arm to do the task.

- The detection of the objects is done by python code with using OpenCV library implemented on the camera that is positioned on top of the work space.

- The Code is about contouring the objects in the frame and getting its center coordinates of each object and returns it as a list.

- Because of limitations when utilizing QUARC, especially when employing an extrinsic function that calls the python methods in MATLAB, we were unable to run the python code inside the QUARC Simulink model.

- Due to that we used the QUARC TCP/IP stream that allowed us to run the python code in a separate MATLAB window and sends the output to another MATLAB window that is running together.

- So, to Apply the w used the client and server QUARC Simulink blocks.

- By putting the server Simulink model in the Simulink model and build and start it then running the client model in the other MATLAB window to send the coordinates and proceeds the operation.

11.2. CLIENT FILE:

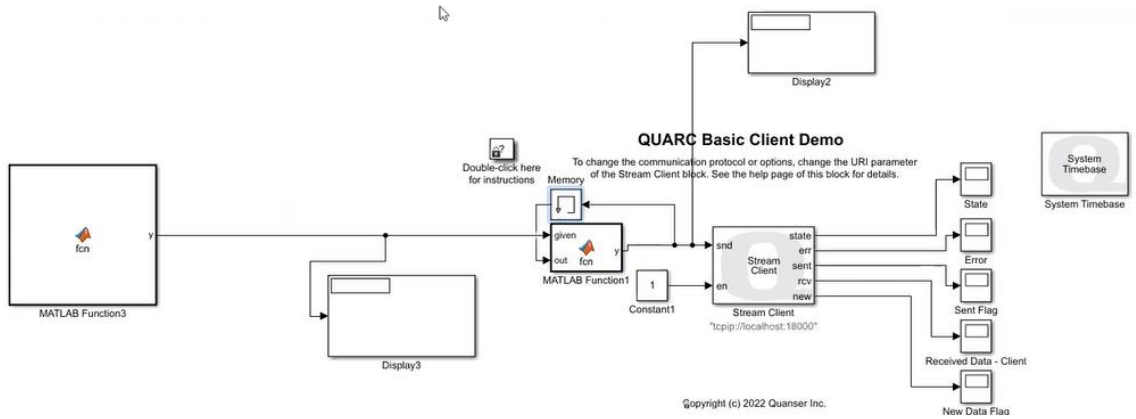


Figure 46: QUARC Basic Client Demo

The client file is the one to be run in a separate MATLAB window.

Inside (MATLAB function 3) block we call the python code and its output is passed to the send in the (Stream client) block.

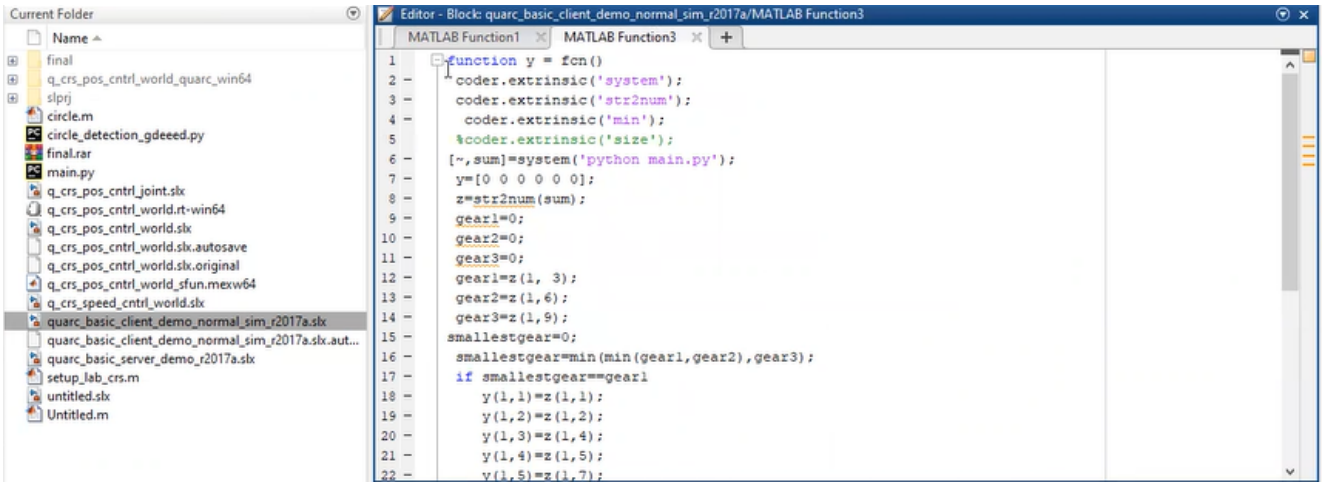


Figure 47: MATLAB Function 3 Block

As shown in the figure inside (MATLAB Function 3) we called some extrinsic functions specially system that calls the python code file inside MATLAB.

One of the important things is to assign the length of the output to be sent.

X`11.3. SERVER FILE:

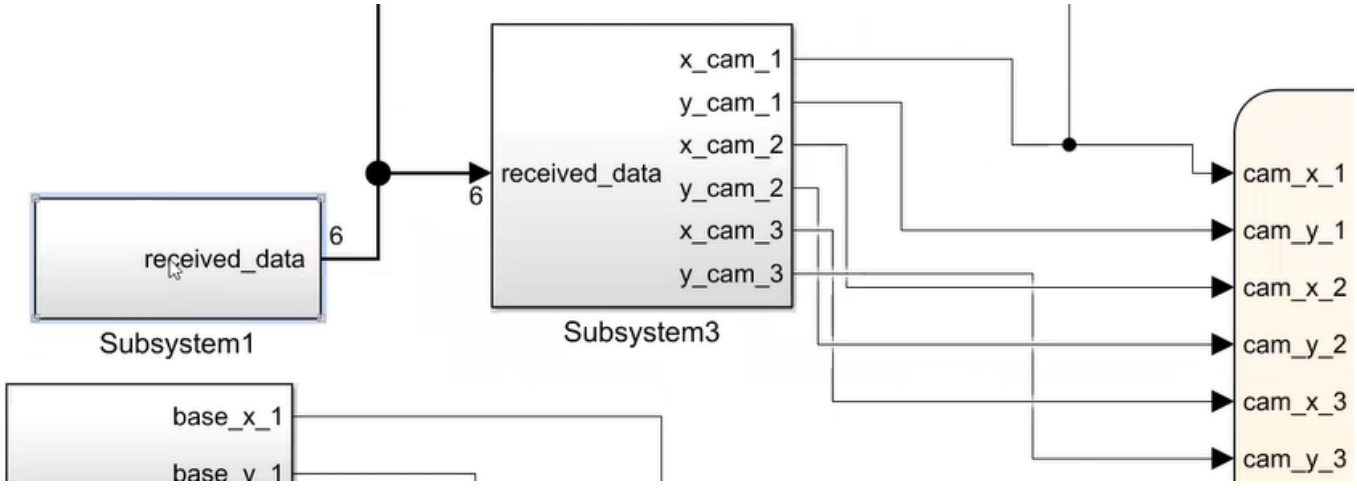


Figure 48: Received Data Subsystem

Inside the (received_data) subsystem is the server Simulink models that outputs the coordinates inside the QUARC Model and input it in the stateflow to do the task.

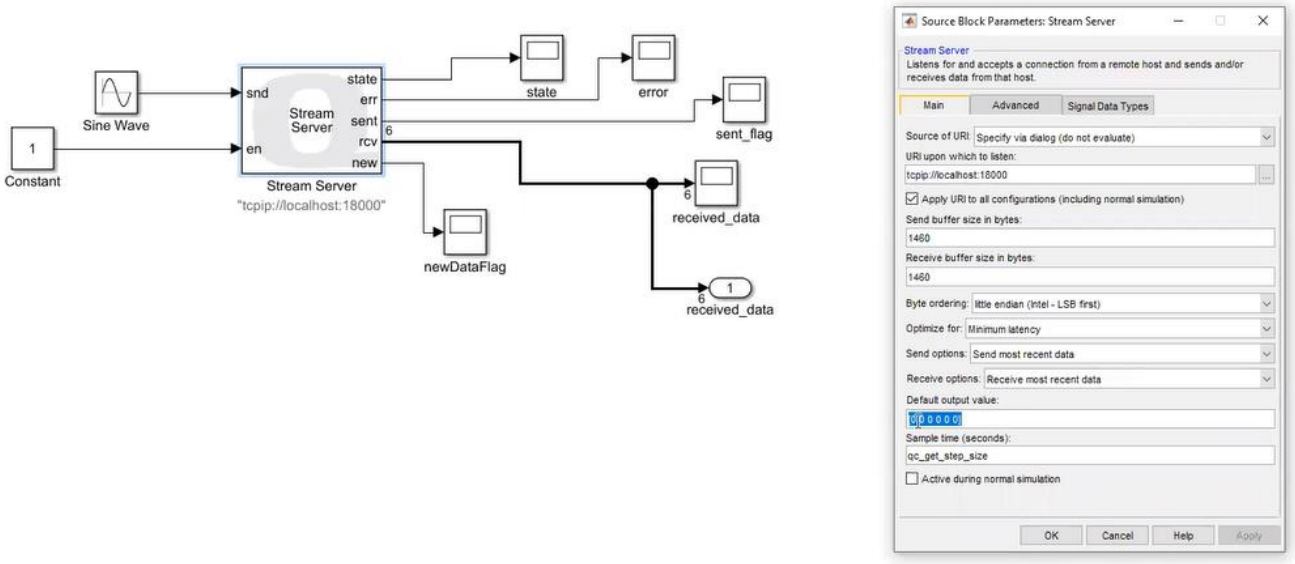


Figure 49: server Simulink model

The server Simulink model is as shown in the figure. Inside the (Stream Server) block we should check on the default output value to match the value sent by the client in the other MATLAB window.

CHAPTER TWELVE: METHOD OF OPERATION

The thermo CRS catalyst robotic arm controlled by the QUARC Q8 card from the MATLAB platform with the Simulink blocks of the QUARC installed library version.

The operation method passes through stages. The first stage is Homing which is so important to identify the start point for the robot as (0,0,0) x, y, z coordinates.

The second stage is running the setup file and the Simulink file to make the robot ready to operate. At the end we should return the robot to its home position to prevent any errors.

12.1. HOMING:

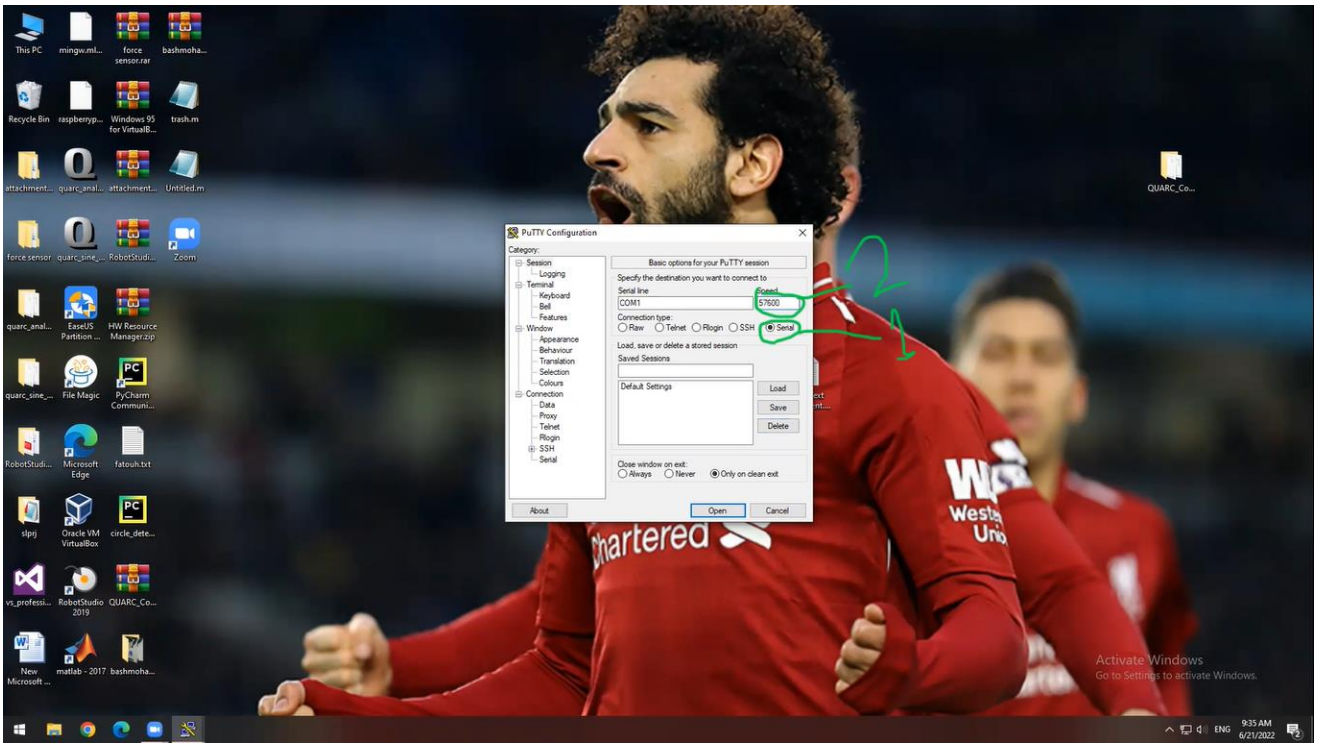


Figure 50: PUTTY Configuration Window

- Open (PUTTY) from start menu
- Choose (Serial) and change the speed to 5700 then press (Open)
- Power On the CRS controller and wait till its fully started
- Disable the pendant by pressing (ESC) button and choose yes
- Press on the (arm Power) button on the CRS controller

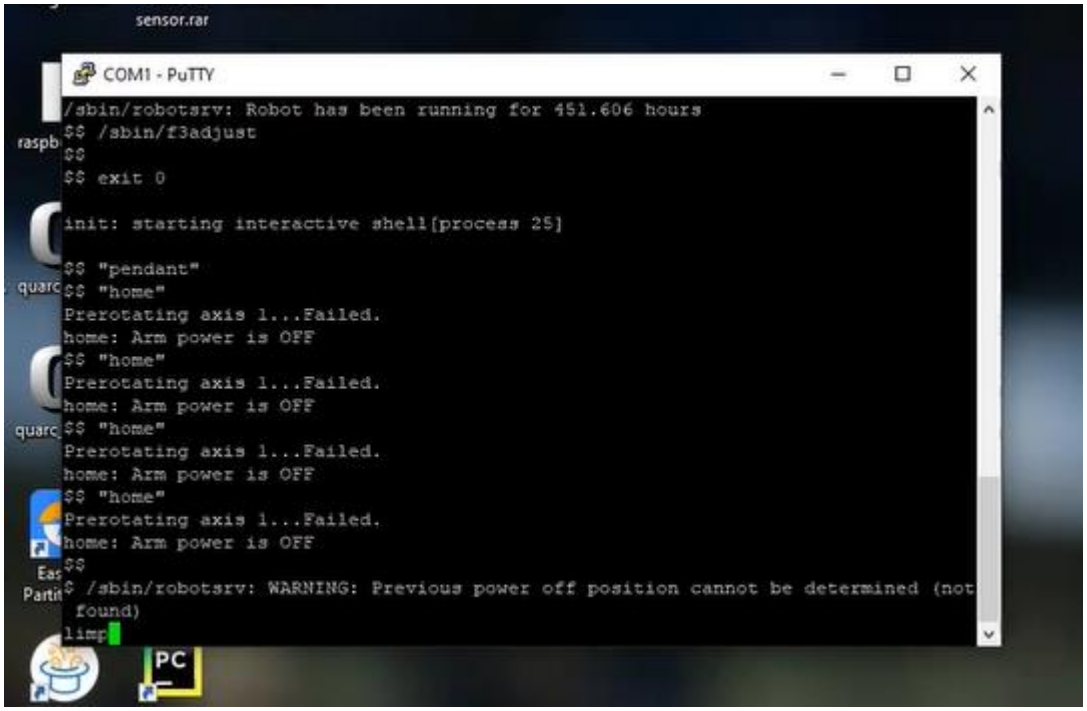


Figure 51: PUTTY Command Window (1)

- Now Type (limp) in the putty terminal

- NOTE: hold the robot cause limp will release all the joints from breaks

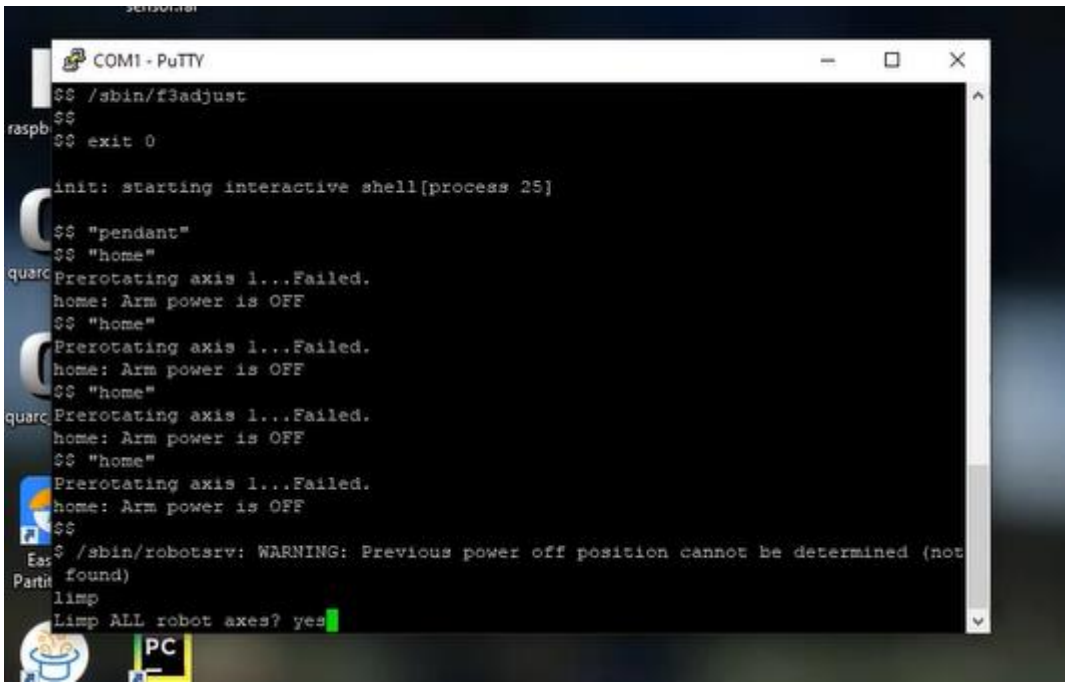


Figure 52: PUTTY Command Window (2)

- Type (YES) to limp and put the robot in the home position that you want manually

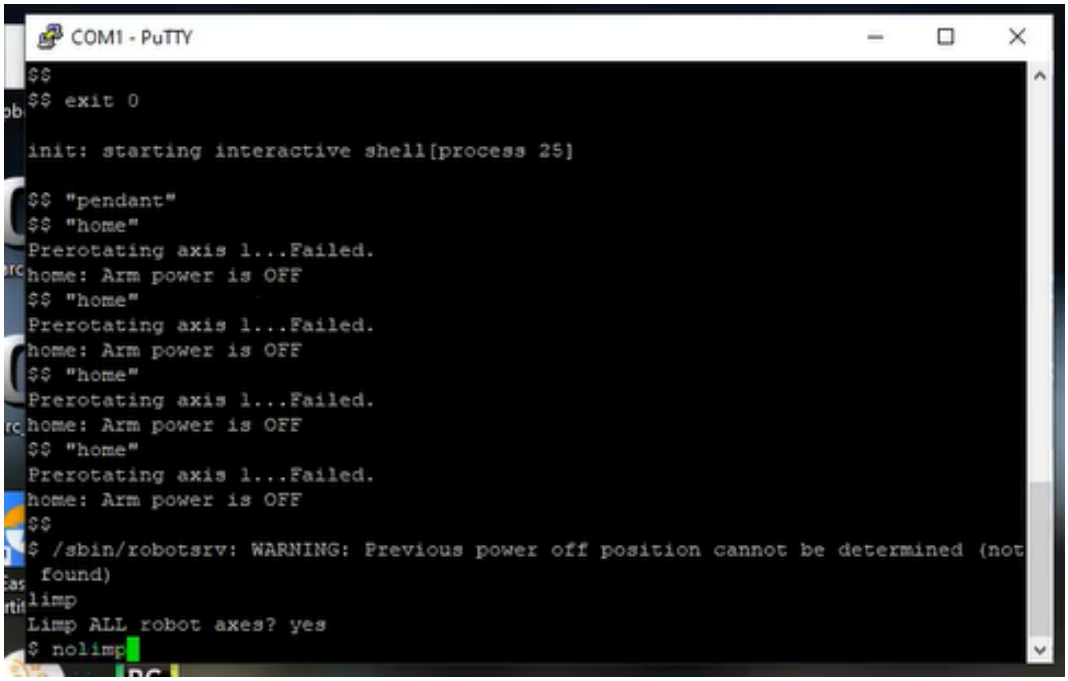


Figure 53: PUTTY Command Window (3)

- Type (nolimp)

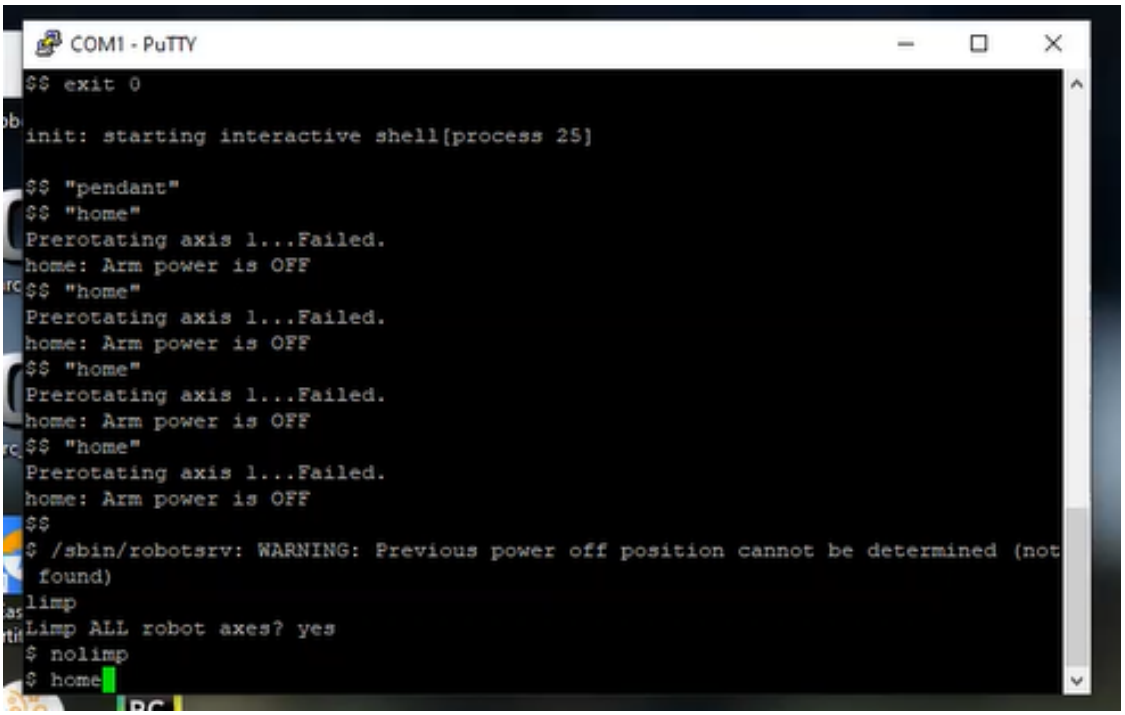


Figure 54: PUTTY Command Window (4)

- Type (home) and wait till the robot process the homing position.

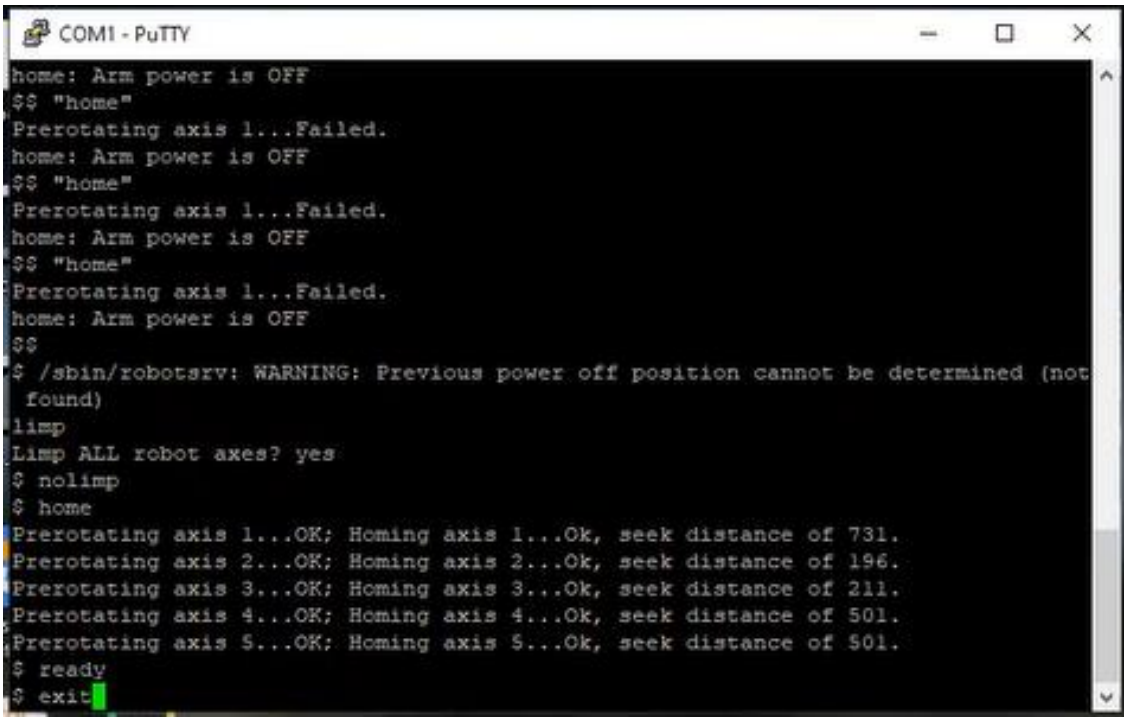


Figure 55: PUTTY Command Window (5)

- Type (ready)
- Type (exit) to end the homing process

12.2. RUNNING AND OPERATING (MATLAB):

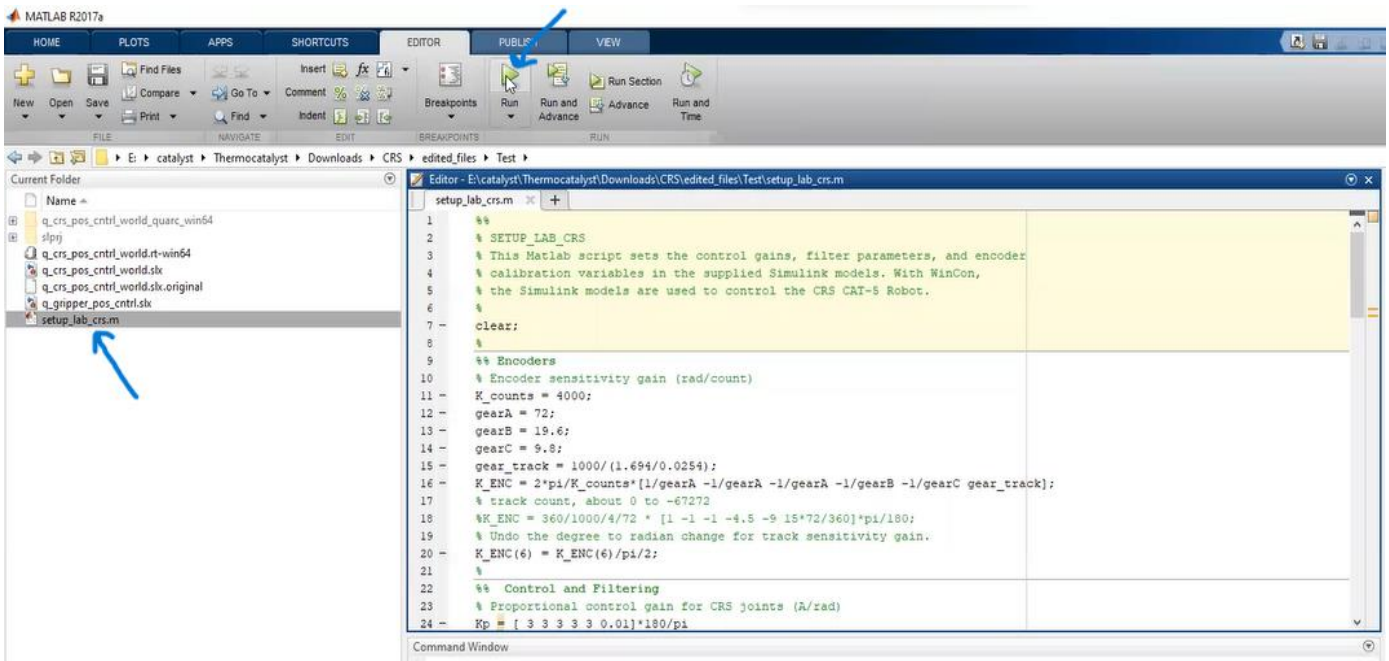


Figure 56: MATLAB Window (1)

- Run (MATLAB R2017a) version cause its compatible with QUARC licensed version that we have
- Choose the directory that has the QUARC files with setup file as shown
- Open (setup_lab_crs.m) and press Run to initialize parameters of the robot

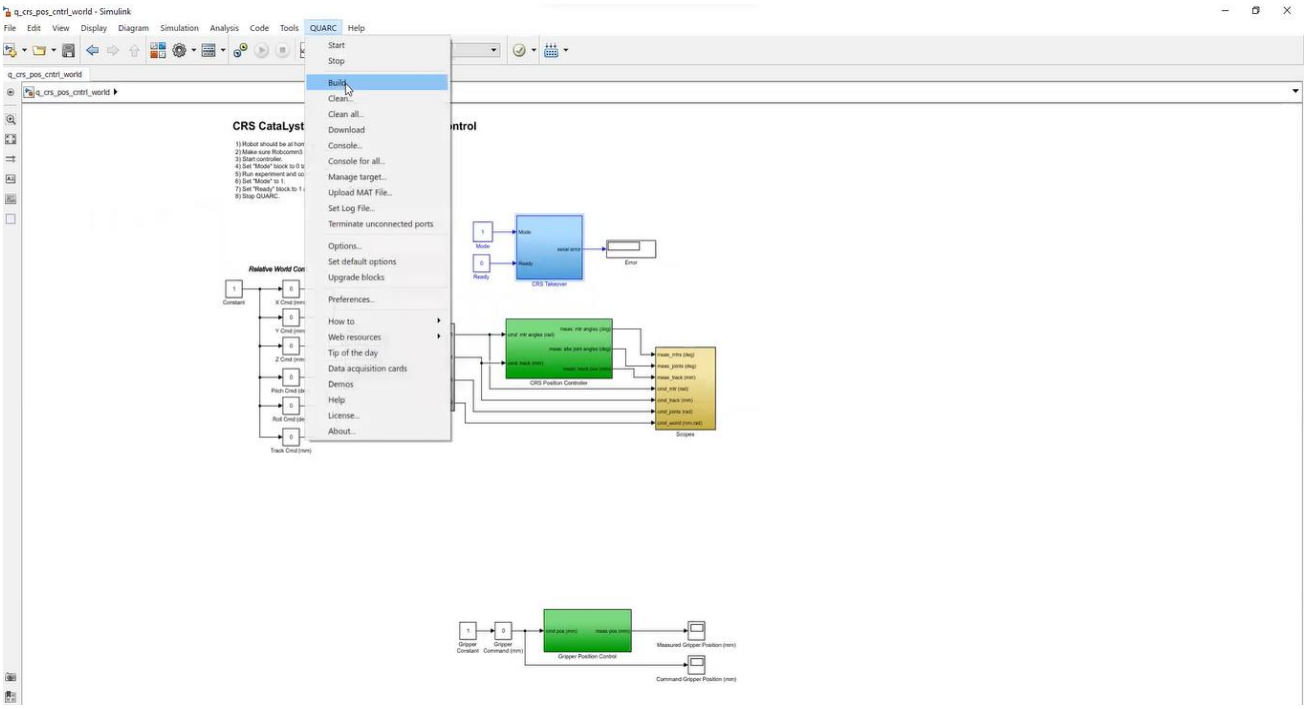


Figure 57: Simulink file(q_crs_pos_cntrl_world.slx)

- Open the Simulink file(q_crs_pos_cntrl_world.slx)
- From QUARC menu choose (Build)

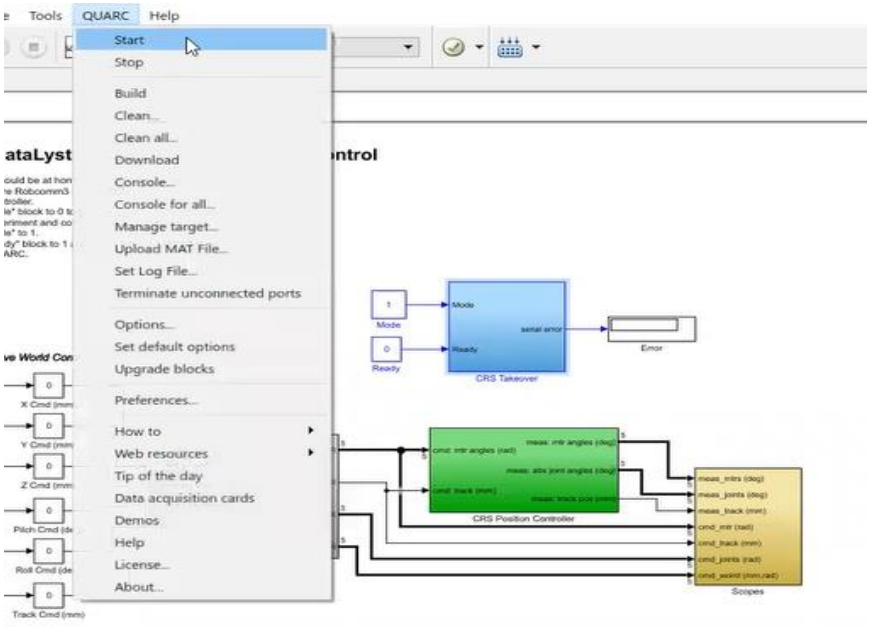


Figure 58: QUARC Submenu

- From QUARC choose (start)

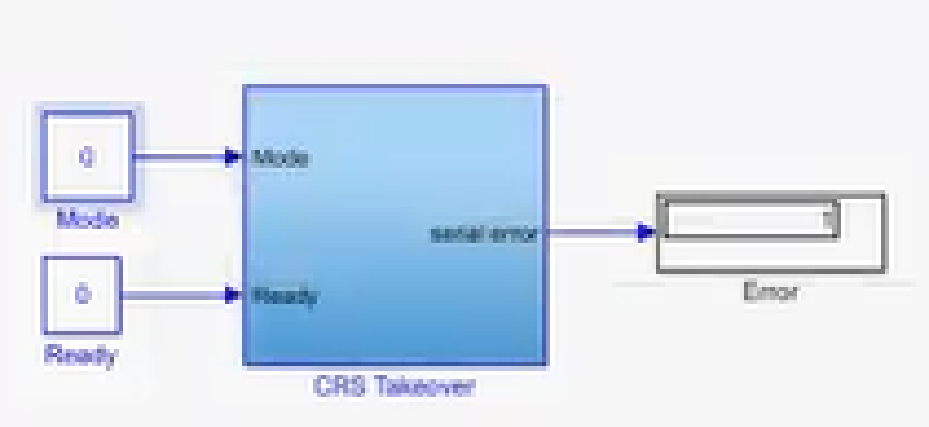


Figure 59: CRS Block

- Change the mode block to be zero to be in open architecture mode

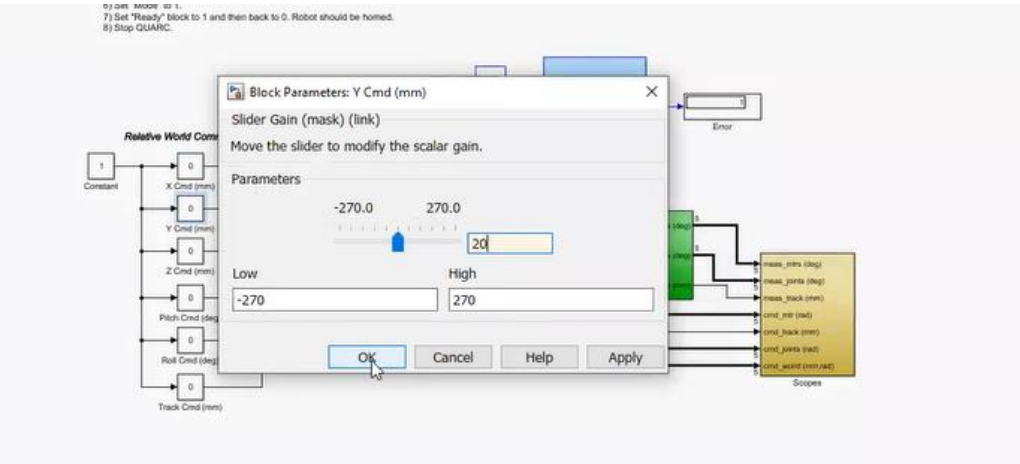


Figure 60: Block Parameters

- Change the values and press ok to make the robot move

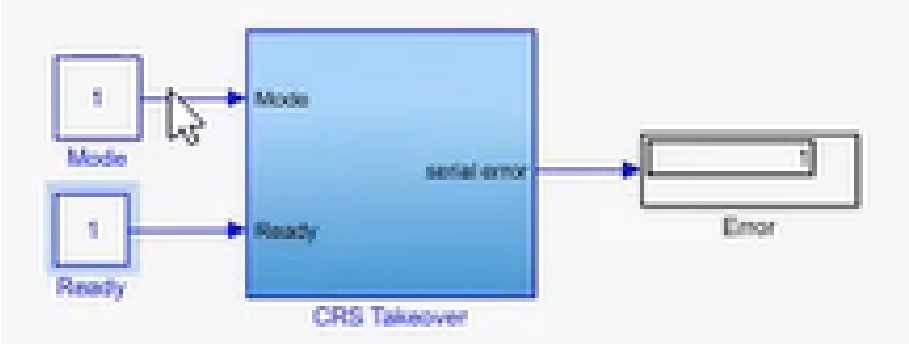


Figure 61: CRS Block

- Change the (mode) and (read) blocks to 1 to be in closed architecture mode and return to the home position.
- From the QUARC menu choose (stop) to end the session.

CHAPTER THIRTEEN: MATLAB MODEL SYSTEM

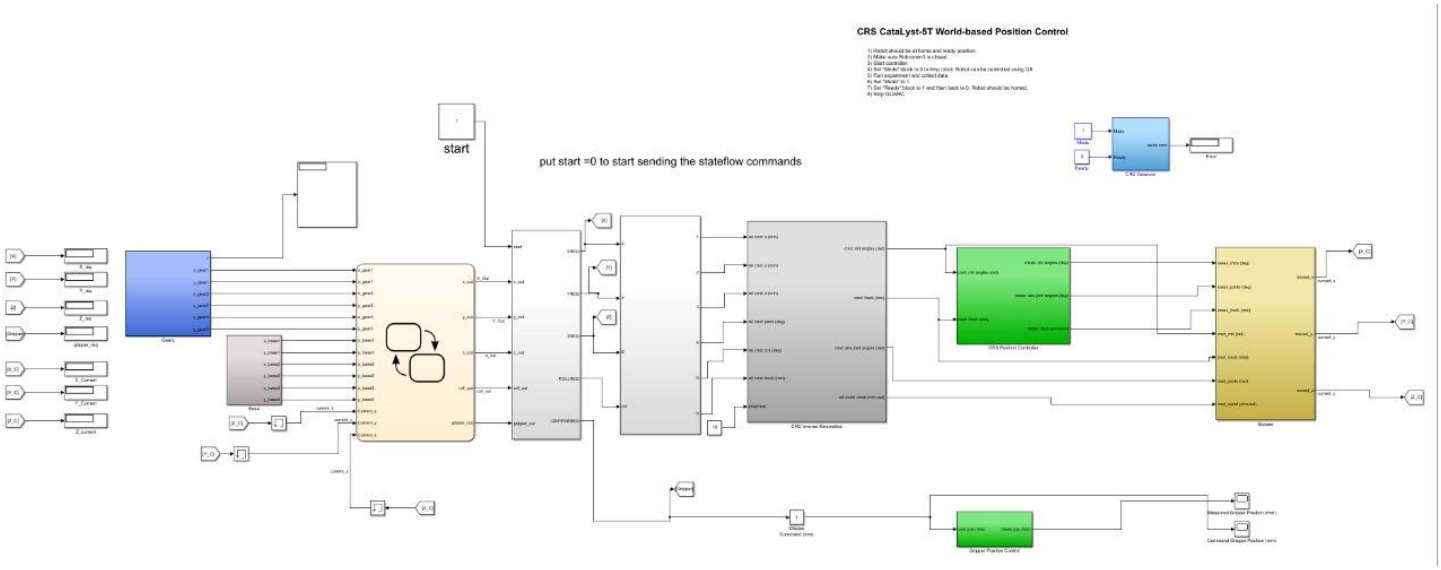


Figure 6082: MATLAB Model

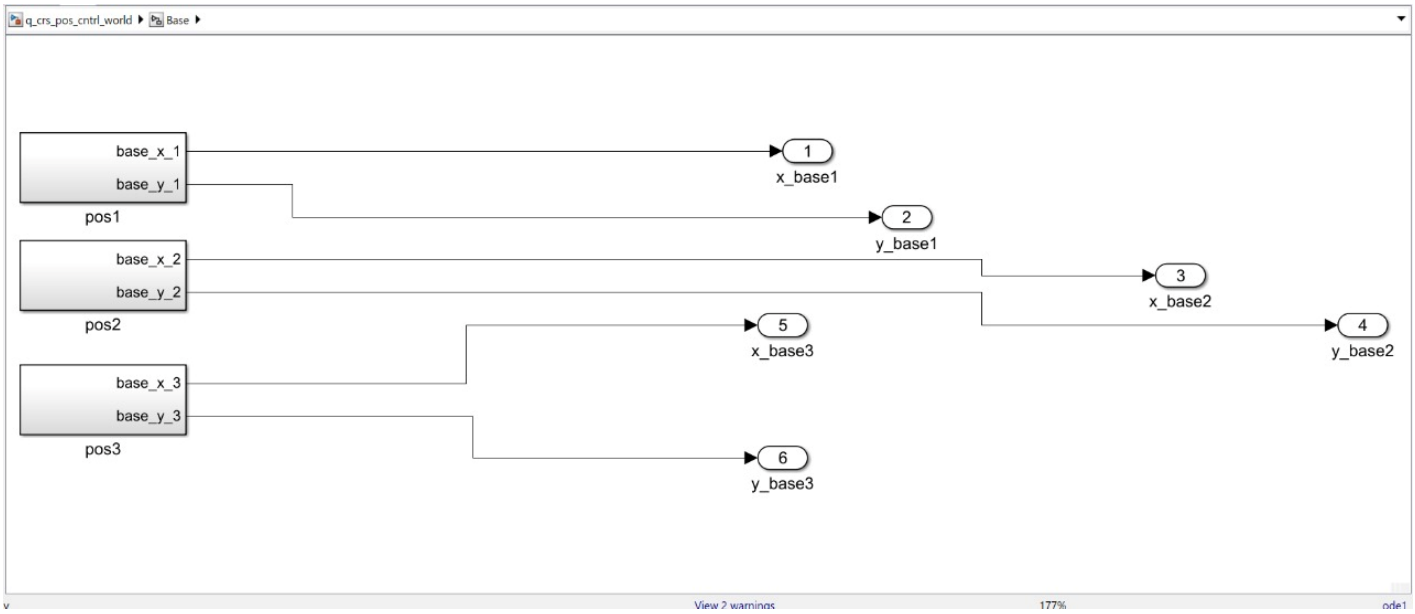


Figure 6073: Base Subsystem

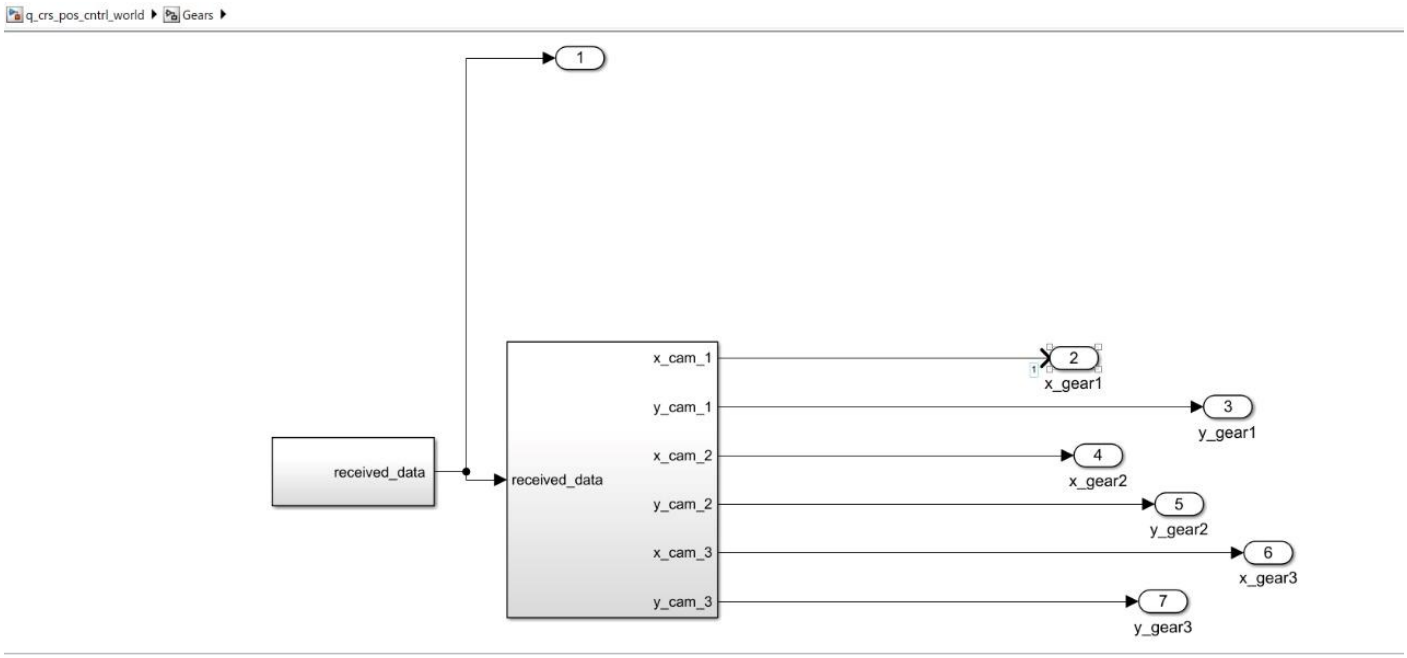


Figure 6104: Gears Subsystem

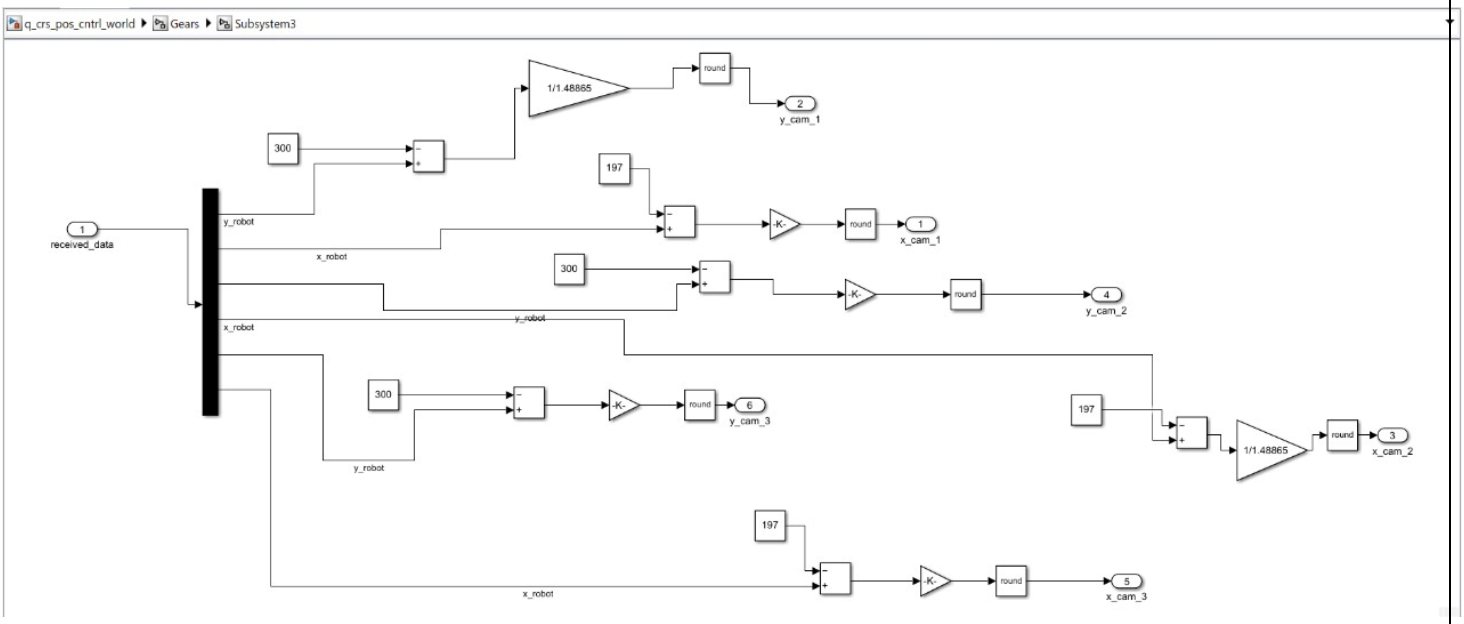


Figure 6095: Subsystem (3)

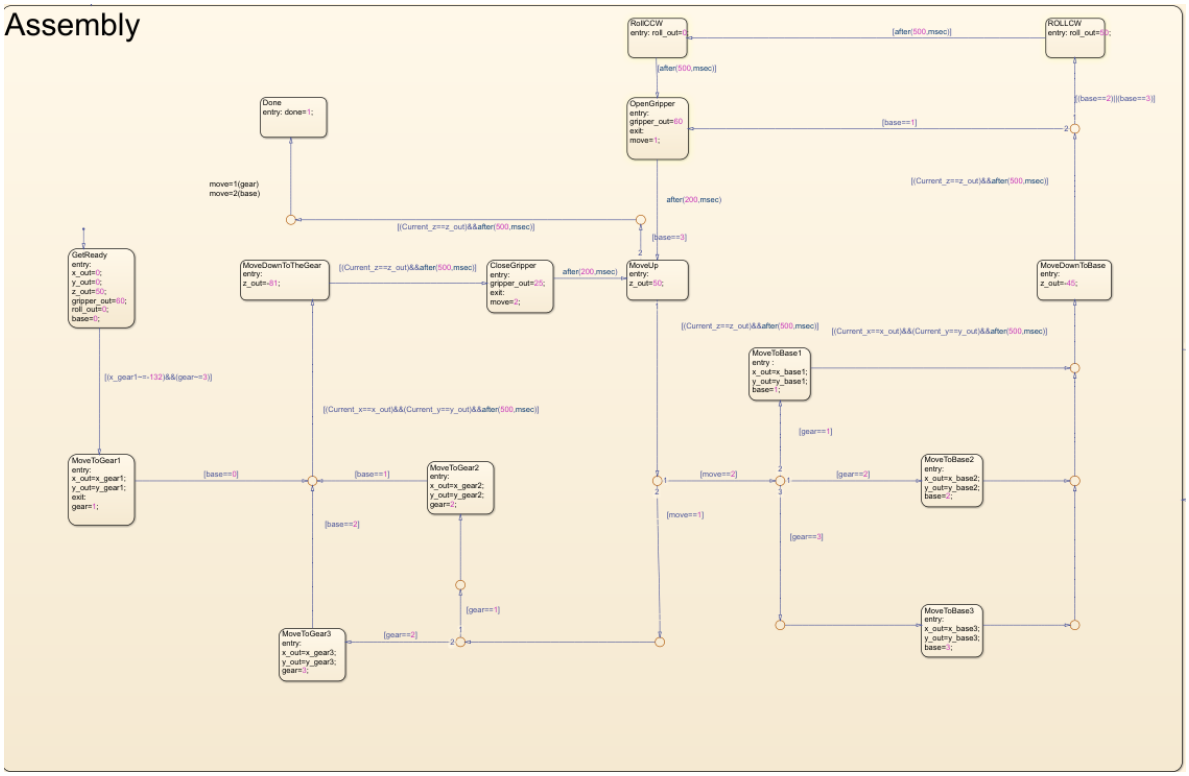


Figure 6126: Assembly State Flow

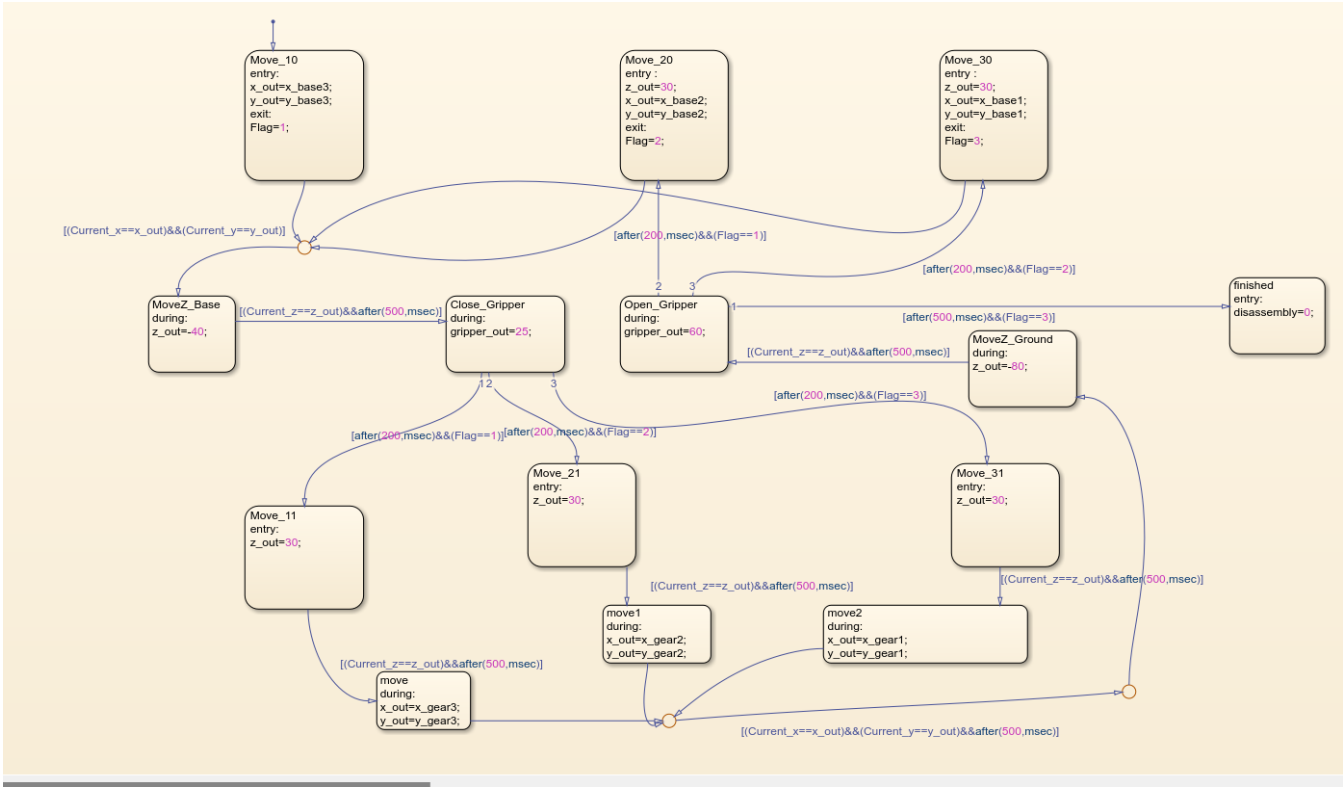


Figure 6117: Disassembly State Flow

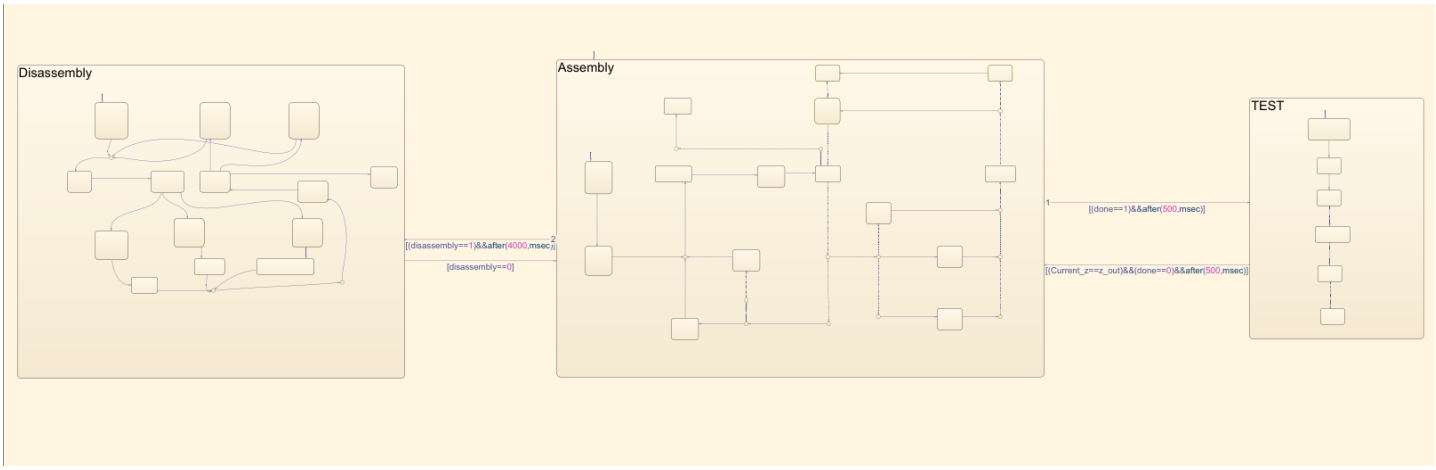


Figure 6138: Integrating Assembly and Disassembly State Flow

CHAPTER FOURTEEN: CONCLUSION

At the end and after all that we have faced in our robotic arm project, it is obvious that setting the robotic arm to be ready to be controlled for assembly/disassembly and other specific operations wasn't easy as we thought. As we faced many problems and challenges as described in chapter (IV), and it took much efforts to be done and to proceed forward in our graduation project process.

The problems with the simulation having unrealistic values, which prevented the design of the impedance controller in time.

The challenges that we face were annoying and weren't easy to solve, but we learnt from it a lot and gave us the courage to reach our goal at the end.

Last but not least, we finally succeeded in controlling our robotic arm via MATLAB software with the aid of vision module system to do its assembly and disassembly function.

To see all the related files, codes and MATLAB blocks scan the QR code below.

REFERENCES

[1] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. Robot Modeling and Control First Edition, Hoboken, N.J. 2005

[2] Daniel E. Whitney. Force feedback control of manipulator fine motions. Journal of Dynamic Systems, Measurement, and Control, 99(2):91–97, June 1977.

[3] J.K. Salisbury. Active stiffness control of a manipulator in cartesian coordinates. In Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on, volume 19, pages 95–100, Dec 1980.

[4] Matthew T. Mason. Compliance and force control for computer-controlled manipulators. Systems, Man and Cybernetics, IEEE Transactions on, 11(6):418–432, June 1981.

[5] M. H. Raibert and J. J. Craig. Hybrid position/force control of manipulators. Journal of Dynamic Systems, Measurement, and Control, 103(2):126–133, June 1981.

[6] Frank Lewis, Darren Dawson, and Chaouki Abdallah. Robot Manipulator Control, Theory and Practice. Marcel Dekker, Inc, second edition edition, 2004.

[7] Joris De Schutter, Herman Bruyninckx, Wen-Hong Zhu, and MarkW. Spong. Force control: A bird's eye view. In Bruno Siciliano and KimonP. Valavanis, editors, Control Problems in Robotics and Automation, volume 230 of Lecture Notes in Control and Information Sciences, pages 1–17. Springer Berlin Heidelberg, 1998.

[8] N. Hogan. Impedance control: An approach to manipulation. In American Control Conference, 1984, pages 304–313, June 1984.

[9] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. Robotics and Automation, IEEE Journal of, 3(1):43–53, February 1987.

[10] T. Yoshikawa. Dynamic hybrid position/force control of robot manipulators– description of hand constraints and calculation of joint driving force. Robotics and Automation, IEEE Journal of, 3(5):386–392, October 1987.

NOMENCLATURE

ACRONYM	Definition of Acronym
RCC	Remote Compliance Compensator
EF	End Effector
RRR	Articulated Geometric Type
RRP	Spherical/Scalar Geometric Type
RPP	Cylindrical Geometric Type
PPP	Cartesian Geometric Type
DOF	Degree Of Freedom
CRS	Catalytic Reaction System