Parallel Kinematic Robot Controller

by

Amir Muhammad Farhan Bin Mohd Fouzi

15490

Dissertation in partial fulfillment of

the requirements for the

Bachelor of Engineering (Hons)

(Electrical and Electronic)

SEPTEMBER 2014

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

PARALLEL KINEMATIC ROBOT CONTROLLER

By

Amir Muhammad Farhan Bin Mohd Fouzi
15490

A project dissertation submitted to the
Electrical and Electronic Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical and Electronic)

Approved by,

_____

(Dr Ho Tatt Wei)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
September 2014

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

Amir Muhammad Farhan Bin Mohd Fouzi

ABSTRACT

In this report, the main work frame of parallel kinematic robot controller consist of two main components, the hardware and its circuitry, and controlling the end-effector, which is the software counterparts. In the subject of a parallel kinematic robot, or specifically in this project, the Delta robot, it is essential to understand the inverse and forward kinematics as well as the Jacobian matrix. These mathematical model helps to understand and to calculate the end-effector position in relation to all three of the robot's angular motor position which are calculated using the inverse and forward kinematics. Then, the velocity of the end-effector which corresponds to the angular velocity of the motors are calculated using the Jacobian matrix. The use of these three models is to regulate and control the trajectory of the end-effector so that its path can be tightly control. The trajectory path of the end-effector is calculated using cubic spline interpolation. The control points are then extracted using this method and each control points can be individually assigned its end-effector velocity as well as its joint angle velocity. The result is a smooth control of trajectory of the end-effector.

All these are implemented using an Arduino Mega 2560 microcontroller. PID algorithm are also implemented in the control system for error compensation of the DC motors. The design of the Delta robot in CATIA are presented as well as the simulation done in MATLAB. The finished prototype is also shown together with the complete schematics of the system.

ACKNOWLEDGEMENT

I would like to express my special appreciation and thanks to my advisor Dr Ho Tatt Wei, you have been a tremendous mentor for me. I would like to thank you for encouraging my FYP and for allowing me to grow as an Electrical and Electronic Engineering. Your advice on both research as well as on my project have been priceless. Furthermore, I would never have been able to finish my dissertation without the guidance from friends, and support from my family. I would also like to acknowledge with much appreciation the crucial role of the staff of Electrical and Electronic Engineering department as well as the technicians, who gave the permission to use all required equipment and the necessary material to complete my final year project.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

The introduction section describes the background, problem statement, and objective and scope of study of the project

1.1. Background

Since the rise of industrial era, the increasing needs for efficiency are greatly demanded especially in manufacturing industry. To achieve this engineers and scientist have come up with an automation system that speed the process of repetitive task that beat human rotor skills itself. This invention is called robot. Robot can achieved multiple programmed task using electrical and mechanical principle. The science and engineering of robots are called robotics. Robots are practically excellent in executing mundane and repetitive task that requires high accuracy and minute need for feedback for the same routine process [1]. The rise of electronics in this modern age has increase the need for a high precision robot that could not only operated with high degree of accuracy but speed as well. This has led to the birth of industrial robots which are usually consist of serial kinematic robot and parallel kinematic robot. The most generic form of industrial robot are serial manipulators. Although popular, it has several disadvantages. The nature of open kinematic structure characteristic is its low stiffness. The design of serial manipulators are like an arm structure that has link that is connected with another link by a joint which would introduce errors. Errors can be greatly accumulated and increased going from link to link. The end-effector can move a relatively small load due to the large weight possessed by most of the actuators located throughout the arm. Due to this constraint, high precision and speed was not possible. Parallel kinematic robots have this capabilities. It has several key advantages when compared with serial arms. The motors of parallel robots are fixed at the base hence

the mass of the robot are mostly situated at the base therefore reducing a great deal of active mobile mass [2]. This also contributes to its high flexibility [3]Direct-drive actuation are perfect with parallel robots, since the motors are located at the base [2]. Other characteristics such as control on maximum allowable velocities, high rigidity, great in determining exact orientation and good positional repetitivity makes up the specialty of a parallel robot [4]. These characteristics allows for great precision and high speed manipulations. There are various parallel robots known with one of them being the flight simulator with 6 DOF (Degree of freedom) using the Gough-Steward platform [5], the parallel manipulator star [6], the HEXA [7] and Delta robot. In this paper, Delta robot will be used. Delta robot was first introduced by Dr Clavel in 1988 [8]. The difference of this robot compared with other parallel kinematic robot is that it partially solve the limited workspace that other parallel manipulators had. Delta robot unique features lies within the parallelogram. In general, if the number of arms and parallelogram is equal to the number of degree of freedom (DOF) of the end-effector with each motor controlling each limb of the robots and number of joints are equal, then is is said to be symmetrical [9].

1.2. Problem Statement

Parallel kinematic robot is designed mainly for accurate positioning with speed. This creates a problem when the end-effector is moving from one position to another. There are multiple ways or solution for the end-effector to travel. The trajectory merely depends on the speed and position of the end-effector. Jerking may occur if proper planning of the motion is not executed. When this occur, this can lead to inaccuracy and sluggish movements. As such, inverse kinematics, forward kinematics and Jacobian Matrix are used to tightly control the trajectory of the end-effector so that smooth motion can be achieved.

1.3. Objective and Scope of study

- Analyze and improve performance of the system such as the use of velocity, acceleration, joint angles and velocity using measurement feedback to achieve synchronized control and smooth trajectory motion.
- To create a control system that can improve path planning in optimal way so that the end-effector can be tightly control to increase the efficiency when navigating through the workspace.

1.4. Relevancy and feasibility

1.4.1. Relevancy

The relevancy of this project is to control the trajectory of the end-effector using position and velocity control so that tighter and more synchronized movement can be achieve. The use of mathematical models of inverse kinematics, forward kinematics and Jacobian matrix is use to achieve the objectives. This project is also relevant to Electrical and Electronics engineering as it uses knowledge from control system, digital electronics, microprocessor, circuit theory and structured programming.

1.4.2. Feasibility

By proper planning and execution, this project can be completed as per scope before the date submission as the major component of this project is the coding part of the system using the Arduino environment.

CHAPTER 2
LITERATURE REVIEW AND THEORY

2. LITERATURE REVIEW AND THEORY

2.1. Literature Review

The complexity of the joint-variable interactions exist within Delta robot caused researchers to find a way to carried out the ease of calculation of dynamic models in real time processing. Several researches have come up with models such as the Lagrange multipliers or the use of Jacobian matrix of the manipulator. Newton-Eulor method for certain parallel structure has also gone through. Techniques based on the virtual work principle, Langrage formalism, Hamilton's equations and other various methods or techniques. All came with a conclusion that modeling by considering mass and inertia of all the links can present a very complex models that are inefficient to be implemented in a control scheme. Mass and inertia of the arms are ignored in order to create a simple control algorithm.

2.2. Theory

A lot of mathematical models have been sought out in order to find the best representative of the real system. When using the control algorithm, it should be simple enough to be calculated in real time [2]. The approach of simple dynamics are used for the modeling of the Delta robot in this project. Three dynamic models are introduced, forward kinematics, inverse kinematics and Jacobian matrix.

First off, the key physical dimensions of the Delta robot's geometry must be determined. The side of the fixed base triangle are denoted as f, the end-effector side triangle as e, the upper arm side starting from the actuators or motors to the joints as rf and the lower arm as re. As mention earlier in this report, the base triangle would be the reference frame with

its center of symmetry be the center of origin of the reference frame. Therefore, end-effector would always be in the $-Z$ coordinates.

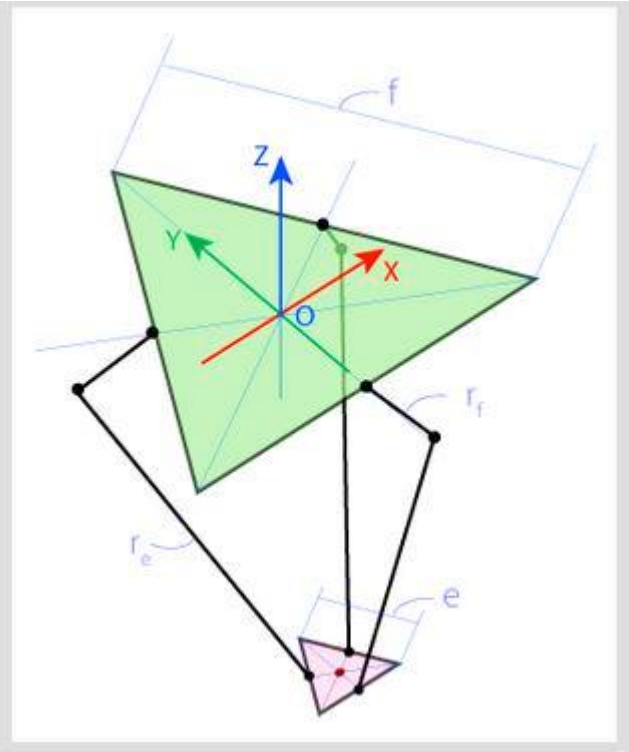


Figure 1 : Parameters definition for Delta robot

2.2.1. Inverse Kinematics

Inverse kinematics is to determine the motor or actuator angles based on the end-effector coordinates. This is important when developing the PID controller of the system. Inverse kinematics is used as feedback to the system in order to correct any error of the end-effector coordinates in regards to the three angles of the DC motors. The coordinates of the joints J_1, J_2 and J_3 are key to finding the angle of the motors. Taking E_1' as the centre of the sphere with radius $E_1'J1$. The intersection of this sphere with another sphere centered at F_1 along the YZ plane would give the coordinates of J_1. The end-effector is a triangle with side's e, connecting to the robot joints J_1, J_2, J_3.

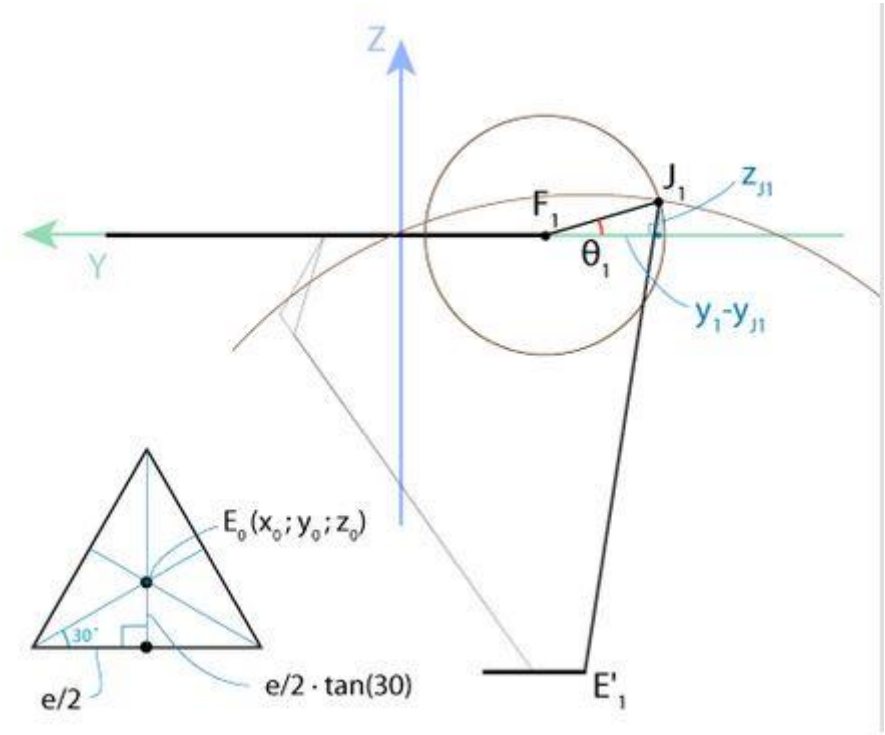


Figure 2 : Intersection of 2 circle for Inverse Kinematics calculation

$$EE_1 = \frac{e}{2\sqrt{3}} \tag{1}$$

$$EE_1 = x_0 \implies E_1'J_1 = \sqrt{r_e^2 - x_0^2} \tag{2}$$

$$F_1\left(0, -\frac{f}{2\sqrt{3}}, 0\right) \tag{3}$$

$$\left(y_{J1} - y_{F1}\right)^2 + \left(z_{J1} - z_{F1}\right)^2 = r_f^2 \tag{4}$$

$$\left(y_{J1} - y_{E_1'1}\right)^2 + \left(z_{J1} - z_{E_1'1}\right)^2 = r_e^2 - x_0^2 \tag{5}$$

$$\theta_1 = \arctan\frac{z_{J1}}{y_{F1} - y_{J1}} \tag{6}$$

To compute for θ_2 and θ_3, we simply rotate the reference frame around the Z-axis 120 degrees clockwise and counterclockwise respectively.

2.2.2. Forward Kinematics

Forward kinematics are used in determining the end-effector position (X, Y, Z) relative to the angles of the motors $(\theta_1, \theta_2, \theta_3)$ by the use of kinematic equations [10]. Although straight forward, the use of trilateration in forward kinematics in finding the end-effector coordinates is not simple as the algebraic approach is complex and tedious. This is the same as inverse kinematics. However, this direct algebrization approach is straight forward.

We represent the end-effector as a single point coordinate at $E_0(x_o, y_o, z_0)$. J_1', J_2', J_3' being the center of three spheres intersecting at E_0. Hence in order to find E_0, three spherical equations of the form $(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = re^2$ where $(x_i, y_i, z_i)$is the coordinates of the center of the spheres with radius re where all the spheres intersects. Using simultaneous equation together with quadratic equation we could solve this.

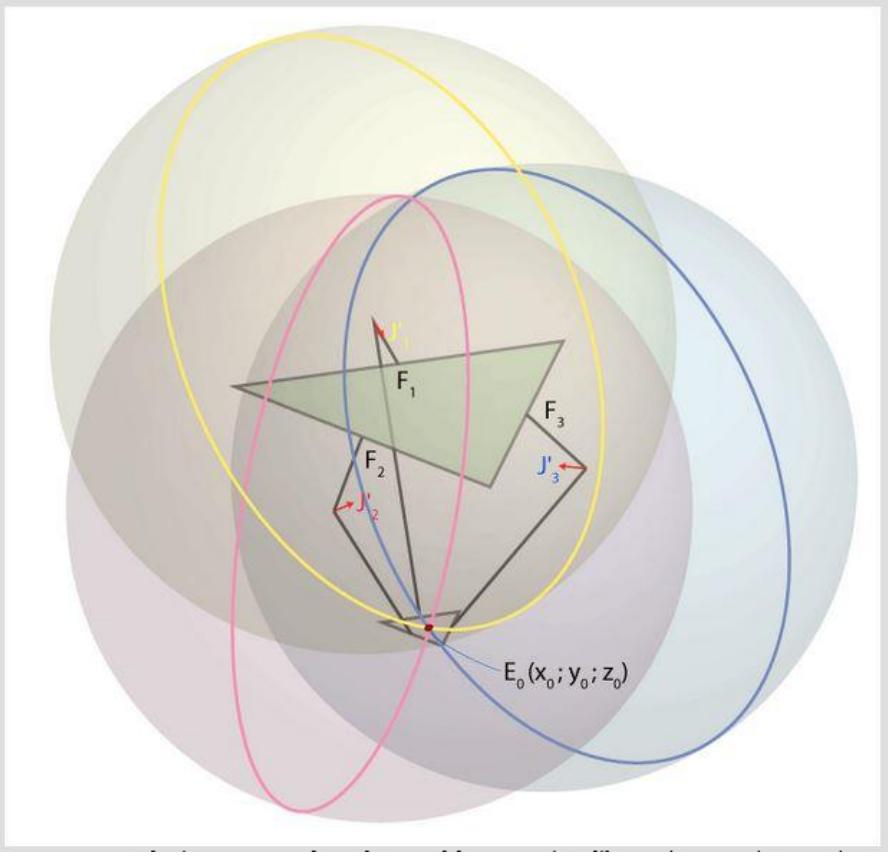


Figure 3 : Trilateration method to solve for Forward Kinematics

2.2.3. Jacobian Matrix

The Jacobian matrix J are derived for Delta robot to express the relation between operational-space velocities or the end-effector velocities and joint velocities or rate of actuator displacement [2] as follows:

$$\dot{X} = J\,\dot{\theta} \tag{8}$$

This Jacobian matrix was based on a numerical computation approach of the partial derivatives of the direct-geometric model with respect to the joint variables[2]. To calculate the Jacobian matrix, constraint equations connecting operational space variables to the joint-space variables are considered[11].

Codourey (1998) use the constraint equations as

$$\|J_i E_i\|^2 - re^2 = 0 \quad i = 1,2,3, \tag{9}$$

stating that the length of the lower arm must be constant. Denoting the vector $J_i E_i = S_i$,

$$S_i = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} - {}^{R}_{i}R \left(\begin{bmatrix} M \\ 0 \\ 0 \end{bmatrix} + \begin{matrix} rf\ cos\ \theta_i \\ 0 \\ -rf\ sin\ \theta_i \end{matrix} \right) \qquad i = 1,2,3, \tag{10}$$

with $\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}$ being the vector matrix of the center of the end-effector and M as the distance from the origin of the reference frame to the center of the motor. Time derivative of S_i from equation (9) with commutativity property results in

$$S_i^T \dot{S}_i = 0 \qquad i = 1,2,3. \tag{11}$$

This then leads to

$$\dot{S}_i = \begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{z}_n \end{bmatrix} - {}^{R}_{i}R \begin{bmatrix} rf\ sin\ \theta_i \\ 0 \\ -rf\ cos\ \theta_i \end{bmatrix} \dot{\theta}_i = \dot{X}_n + b_i \dot{\theta}_i \qquad i = 1,2,3, \tag{12}$$

where

$$b_i = {}^{R}_{i}R \begin{bmatrix} rf\ sin\ \theta_i \\ 0 \\ -rf\ cos\ \theta_i \end{bmatrix} \dot{\theta}_i \qquad i = 1,2,3. \tag{13}$$

Using equations (10) and (13) and rearrange equation (12), the following is obtained:

$$\begin{bmatrix} S_i^T \\ S_i^T \\ S_i^T \end{bmatrix} \dot{X}_n + \begin{bmatrix} S_i^T b_i & 0 & 0 \\ 0 & S_i^T b_i & 0 \\ 0 & 0 & S_i^T b_i \end{bmatrix} \dot{\theta}_i = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad , \tag{14}$$

Comparing this with equation (8), we can deduce that

$$J = -\begin{bmatrix} S_i^T \\ S_i^T \\ S_i^T \end{bmatrix}^{-1} \begin{bmatrix} S_i^T b_i & 0 & 0 \\ 0 & S_i^T b_i & 0 \\ 0 & 0 & S_i^T b_i \end{bmatrix}. \tag{15}$$

It can be noted here that the Jacobian matrix J is a function of the end-effector position and function of actuator angles $\theta_i$with $i = 1,2,3$.

Trajectory planning and optimal motion control can also be achieved using inverse and forward kinematics as well as the Jacobian matrix. The use of spline function could also solve the optimal trajectory planning with case of point to point tasks and continuous path tasks [12].

CHAPTER 3

METHODOLOGY

3. METHODOLOGY

3.1. Flow Chart

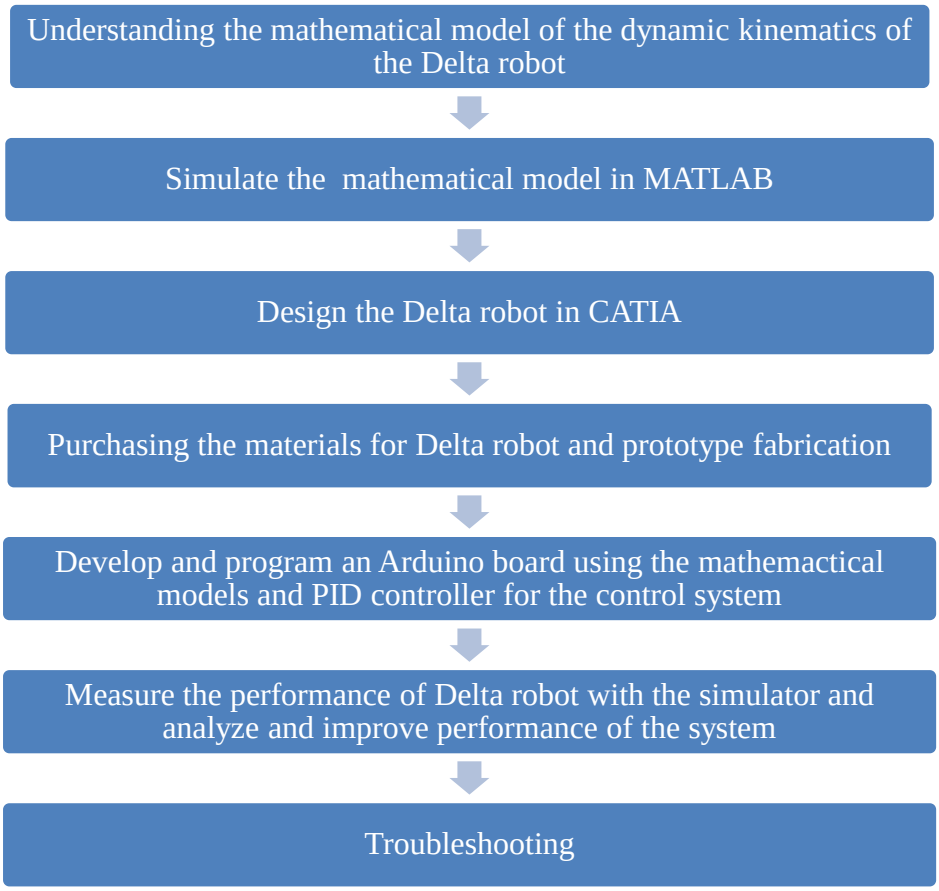


Figure 4 : Flow chart of the methodology

The flow chart above states the several steps of activities that need to be executed to achieve the objectives. First step is to understand the mathematical models, the foundation of the dynamic kinematics of the Delta robot. In particular, the mathematical models are

the forward kinematics, inverse kinematics and Jacobian Matrix in order to manipulate the angle of actuator or DC motors ($\theta_1, \theta_2, \theta_3$), the position of the end-effector (X, Y, Z) and the angular velocity ($\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3$). Inverse kinematics is used to convert the translational motion of the end-effector to the angular position of the DC motors. This is use at the beginning of the control system. Forward kinematics is the opposite of inverse kinematics. It is used to compare the actual angular position of the motors with the desired values for error correction. Jacobian Matrix is used to translate the translational velocity of the end-effector to the angular velocity of the motors. This is important in order to achieve the objectives of the project which is trajectory control. If each control points can be manipulated, velocity can be control hence smooth trajectory. Next step is simulating the mathematical model in MATLAB. In order to fully understand the mathematical models mention above, simulation using MATLAB is a great way to achieve it. MATLAB handles matrices exceptionally well, therefore perfect for simulation Jacobian Matrix calculation. The step after is designing Delta robot. Based on the torques and size of the DC motors, the dimension of the robot is determined. Several redesign have been made in order to choose the best design. Inverse and forward kinematics solely depends on the dimension of the robot. It is important to precisely design the robot. Next, is to purchase the materials needed to fabricate the Delta robot. One of the hardest material to find is the magnetic ball joints. This is needed for connecting the upper and lower arm of the robot at the joint also the lower arm and the end-effector. Arduino board is the microcontroller chosen for this project since it has a wide array of pre-define syntax as well as massive collection of tutorials and libraries that can be easily obtained. After that, the mathematical models as well PID controller for error compensation of the control system is developed and programmed into the Arduino board. Finally, the performance in terms of joint angle velocities, end-effector velocity, angles of the motors, trajectory and motion of the Delta robot is measured and compared with the simulator. Using measurement feedback, all this parameters are analyze and improved during troubleshooting phase.

3.2. Designing and building Delta robot

To design and build the Delta robot, it has to be divided into several sections. First, the mechanical design of the robot in CATIA. The choice of materials used also effect on how the design is made. Second, the selection of the electronic hardware. This includes the selection of three DC motors with encoders, microcontroller, motor drivers and circuitry boards.

3.2.1. Mechanical design

The foundation behind the Delta robot unique features lies on parallelograms usage. The parallelogram allows the end-effector to have fixed orientation, parallel with the base of the robot regardless of the angular motion of the motors. Three parallelogram connecting the end-effector to the upper arm confine the orientation of the end-effector with the intention of three pure translational degree of freedom. The three DC motors (actuators) with encoders are attached 120 degrees far apart from each other at the base of the robot. Refer figure 5 for the schematic of the Delta robot.

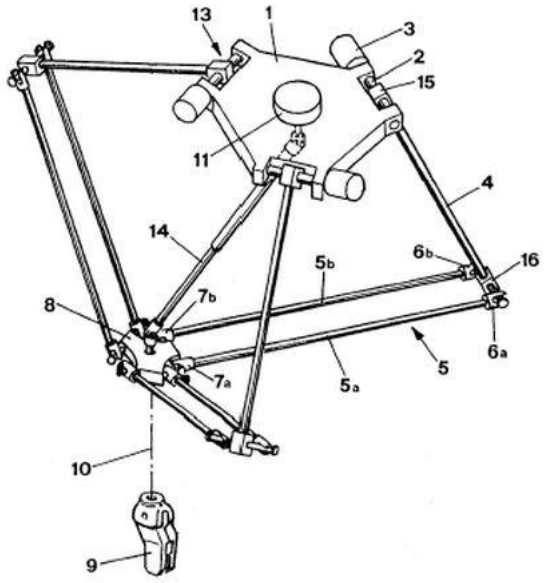


Figure 5: Delta robot parts (extract from US patent 4,976,582)

1) Robot base
2) Shaft (DC motor shaft)
3) Fixed parts
4) Upper arm
5a, 5b) Parallelogram
6a, 6b, 7a, 7b) Revolute joints
8) End-effector
9) Working element (not used in this project)
10) End-effector joint
11) Fixed motor (not used in this project)
12) Control system
13) Actuator (DC motors)
14) Telescopic arm (not used in this project)
15) First extremity
16) Second extremity

The design of the Delta robot has undergone a lot of changes. Initially, the base was design separately with the motor holders. In order to create a better and solid base, the new design has the base and the holders fabricated together using a single aluminum sheet of 1mm thickness. It is created based on the length of the centre of the base to the middle point of thickness of the upper arm. This is important because based on the mathematical models, the position of the end-effector is determine by the distance of the middle of the upper arm to the center of the base. Refer figure 8 for the aluminum base. The end-effector also has undergone several design modification. Each end of the end-effector has an 11mm hole to fit 3 neodymium magnets of 11mm in diameter, and 15 mm in total length. Refer figure 11 and 12 for the end-effector. These magnets will hold the upper arm at the magnetic ball joints. Refer figure 13 and 14 for the upper arm. The lower arm or parallelogram was design simply using an aluminum rod of 9.5mm in diameter with length of 90 mm. Each end of the rod is threaded using a 3 mm thread to connect the magnetic ball joints. Refer figure 15 for the parallelogram. The finished model is then assembled with ease since the joints can be connected magnetically. See appendix C for the complete dimensions of the models made in CATIA.

3.2.1.1. Planetary DC Gear Motor (IG32E-35K)

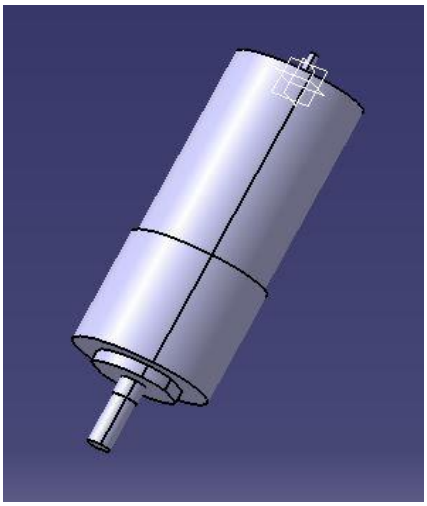


Figure 6 : Planetary DC motor in 3D view

3.2.1.2. Base of Delta Robot

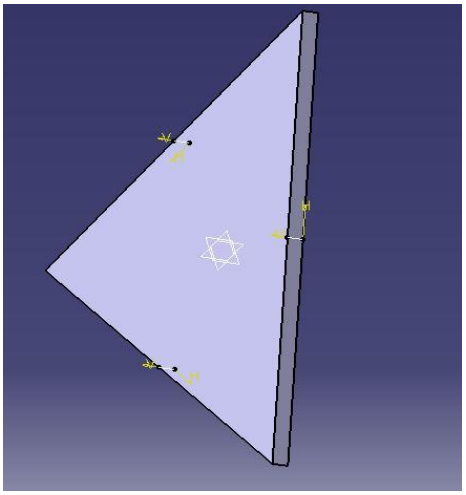


Figure 7 : Old base design

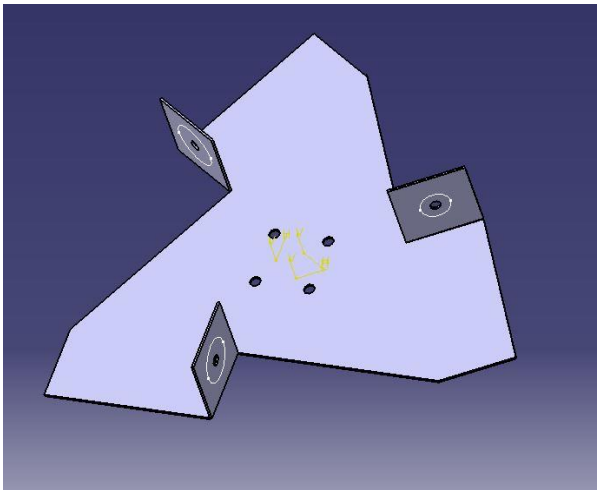


Figure 8 : New base design with motor holder

3.2.1.3. DC motor holder

The new design has the DC motor holders attached together with the base to form one single compound. This design makes the structure more solid and rigid compared if attaching the motor and the base independently. Refer figure 8 for the new design.

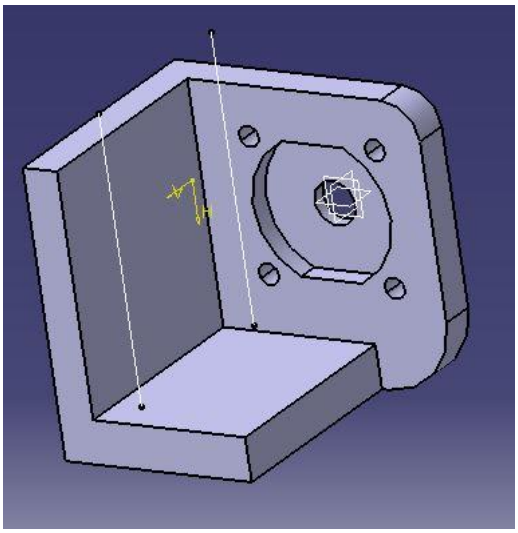


Figure 9 : Old design DC motor holder

3.2.1.4. End-effector

The old design of the end-effector was very basic since it does not include holes to slot in the Neodymium magnets and the thickness was not sufficient. All this concern has been addressed fixed in the new design. The 3 mm holes near the centre was designed to insert M3 screws tighten with nuts so that the magnets do not slipped out.

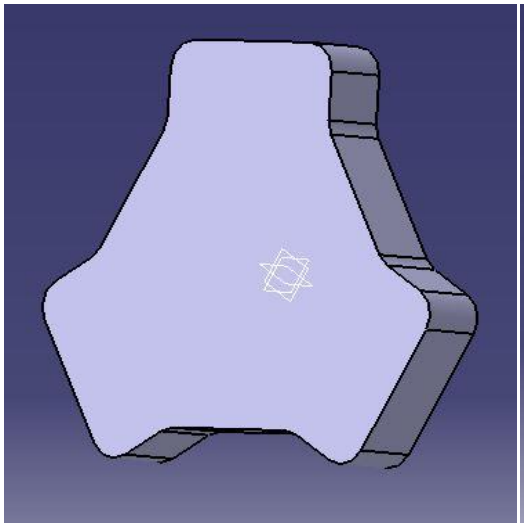


Figure 10 : Old design End-effector

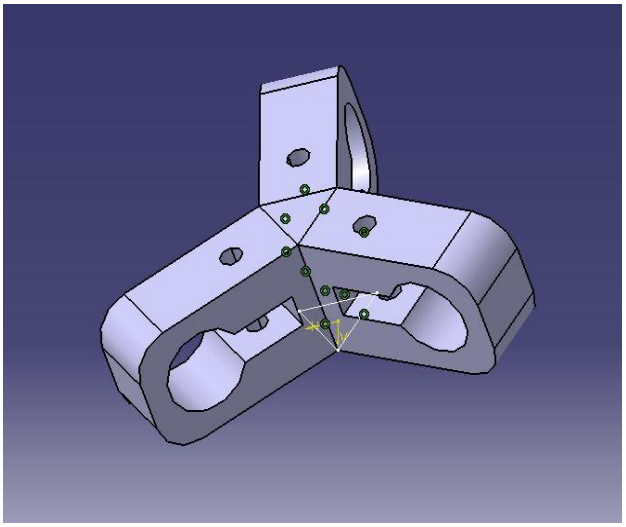


Figure 11 : New design end-effector

Figure 12 : End-effector fabricated using acrylic

3.2.1.5. Upper arm

The upper arm has a length of 50 mm from each centre of the holes. This also acts as a coupler to the shaft, hence the M3 screw.

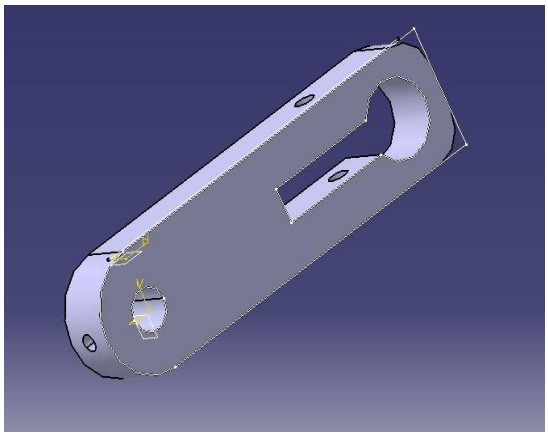


Figure 13 : Upper arm design

Figure 14 : Upper arm fabricated using Acrylic

3.2.1.6. Lower arm with magnetic ball joints

The lower arm or parallelogram was fabricated using an aluminum rod of 9.5mm diameter, and length of 90mm. Each end of the rod was drilled and thread with diameter of 3mm. This threaded hole is to connect each end of the rod with the magnetic ball joints.

Figure 15 : Lower arm with magnetic ball joints.

3.2.2. Selection of electronic hardware

The electronic hardware chosen for the microcontroller is Arduino Mega 2560. Arduino is an open-source, single board microcontroller built for making interactive objects with the environment easier. It is designed based on 8-bit Atmel AVR or 32-bit Atmel ARM. The open source nature of this board makes it a great selection for student projects even at undergraduate level. It has a huge community doing various projects across a lot of cluster. Variety of libraries and references, including PID library, exist making it more convenient and suitable to use. Because of this, it is a great microcontroller to use for this project.

Actuators chosen for this project are based on the required torque. Even though stepper and servo motor are great for control angular position, it is deemed as not fast and smooth enough for Delta robot application. Hence, DC motor is chosen. DC motor used in this project to control the end-effector would be Planetary DC Gear Motor (model: IG32E-35K). For the position feedback and angular velocity measurement, encoders are used. IG32E-35K motor is a great choice since it has encoder attached to the end shaft of the motor, enough torque and has sufficient speed for Delta robot application. The motor driver to accomodate for this project is Flexibot Driver, FD04A.

3.2.2.1. Arduino Mega 2560

Arduino Mega is based on ATmega2560. This board features 54 digital input/output pins where 15 of them can be used as Pulse Width Modulation (PWM) outputs. For analog counterparts, it has 16 inputs. The board also has 3 SPI pins that support SPI communication using the SPI library, 2 Two Wire Interface (TWI) communication pins, USB port, power jack and a automatic (software) reset. This is where the main program of the robot will be implemented in as well as the PID control algorithm. See appendix A for its schematics.

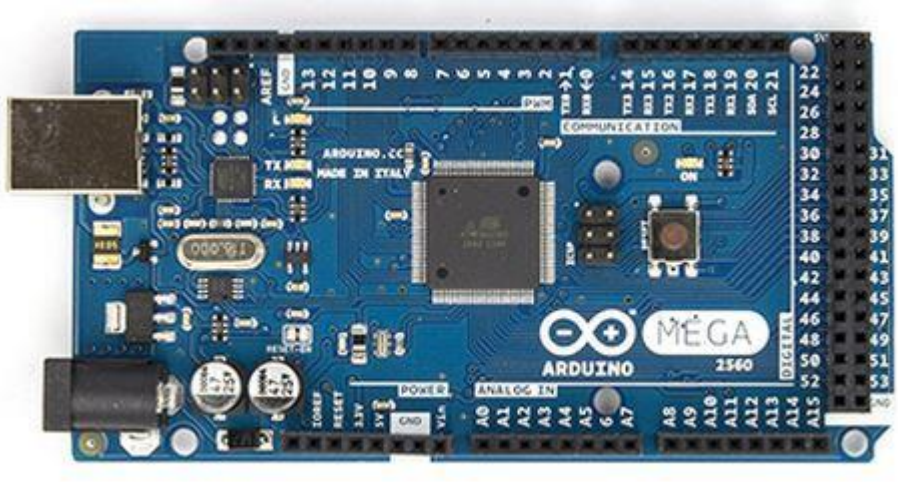


Figure 16 : Arduino MEGA 2560

3.2.2.2. Flexibot Driver, FD04A

This driver has 4 channel specially dedicated for 4 DC brush motors at 3A. It is able to drive DC motors in 2 direction (CW or CCW) and speed control with maximum 40KHz PWM frequency via a 14 way IDE cable connector. This is where the driver connects with the Arduino Due. It has pluggable connector for 4 DC motors and 1 for power supply of maximum 26V.

Figure 17 : Flexibot Driver, FD04A

3.2.2.3. Planetary DC Gear Motor (IG32E-35K)

This DC motor has a gear ratio of 35, rated torque at 270mN.m, rated speed of 170 RPM, 12V input and a rated power of 7W. The encoder (IG32E) is a 2 channel Hall Effect encoder with its output that provides a 245 pulses per rotation. This is equivalent of 0.68 pulses per degree or 1.47 degree per pulse. This degree of resolution although enough for the Delta robot, it can be increase using either 2X, 3X or 4X encoding technique. See appendix A for the encoder specification.

Figure 18 : Planetary DC Gear Motor (IG32E-35K)

3.3. Controlling the robot

In this section of the report, simulation in MATLAB is first presented. Then, it will be on coding the encoder, angular position control of the DC motors, PID control and tuning. Lastly will be on the implementation of the mathematical models of the system into the Arduino. A part-by-part approach of the coding is to minimize error during compilation of the codes.

3.3.1. MATLAB simulation of the mathematical models

To fully understand the mathematical reasoning presented in the theory section of this report, all the required mathematical models are simulated using MATLAB. The reason why MATLAB is chosen is because it can handle matrix very efficiently which in this case, the models are presented a lot in matrix form. To check whether the inverse and forward kinematics are correct, the value gotten from the inverse kinematics are put as the input to the forward kinematics equation. If both values appears the same, the codes and the mathematics are correct. For the Jacobian Matrix, the end-effector velocity is only considered in one axis at a time (X, Y, Z). See Appendix B for the MATLAB codes for the forward kinematics, inverse kinematics and Jacobian Matrix.

3.3.2. Coding in Arduino IDE

For this report, some of the coding has been made in parallel with the fabrication and designing work of the Delta robot. The programming is made using Arduino integrated development environment (IDE). The programs are written in Arduino IDE which uses a language based on C. The codes are done in several stages. The first stage is the coding to read the encoder values from the DC motors. Next stage is to do the code for controlling the angles of the motors using feedback from the encoders. Then, PID controller is used for error compensation of the motors. The later part would be implementing the inverse kinematics, forward kinematics and Jacobian matrix compatible with Arduino IDE.

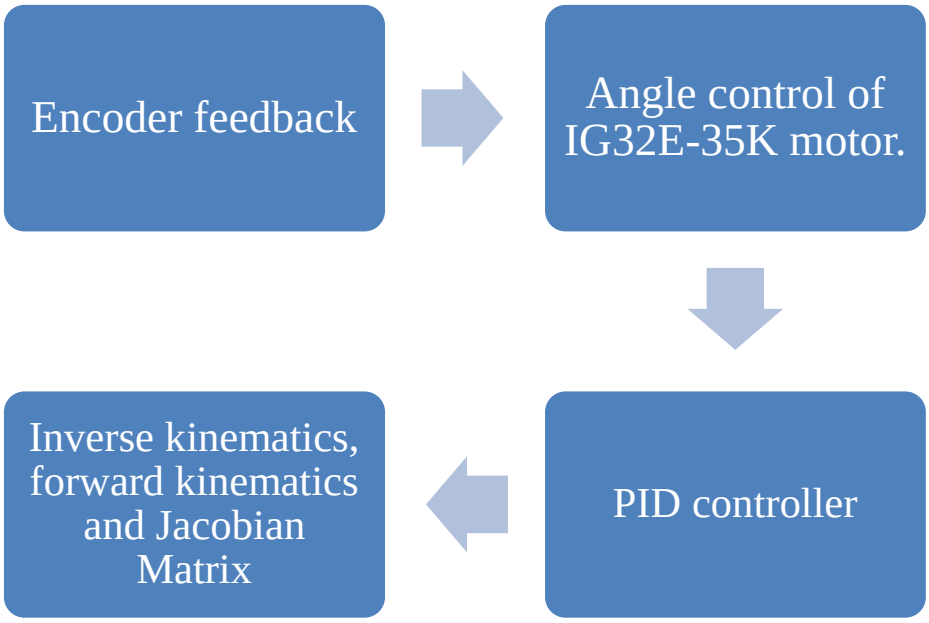
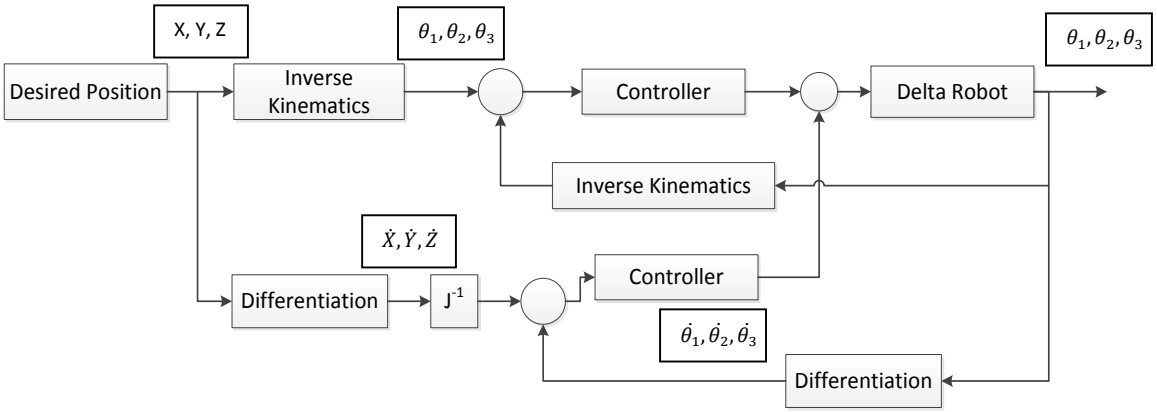


Figure 19 : Coding stages

Below is the control loop of the system, together with the inverse kinematics, forward kinematics and Jacobian Matrix.



3.3.3. Encoder

The encoder used in this project is a 2-channel encoder with resolution up to 245 pulse per revolution (ppr). By dividing with 360 only gives the resolution up to 0.68 pulse per degree or 1.469 degree per pulse. Quadrature encoder has 2 channel with each channel

producing a set of pulses 90 degree out of phase with the other channel. In order to make encoder measurements, the number of edges (low to high or high to low transitions) must be counted. External interrupts pins can be used to count this. Arduino Due has powerful interrupt abilities that allows the user to attach ISR (Interrupt Service Routine) on all available pins. From the number of edges counted, this information can be used to convert to angular position, speed, and acceleration. The pulse resolution can further be increase using either 1X, 2X or 4X encoding. 1X encoding is by measuring the low to high transition edges of channel A. By looking at the state of channel A and B during this transition edge, we can determine the CW direction or CCW direction. The downside of 1X encoding is that it uses the same frequency of channel A. Therefore that is the result with 245 ppr.

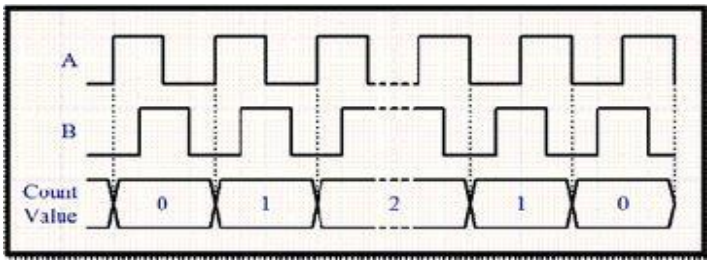


Figure 20 : 1X encoding

This can be improve by using both edges of channel A. The same principle works just as well only this time for falling edges. Thus we can get twice the transition, increasing the resolution from 245 ppr to 490 ppr.

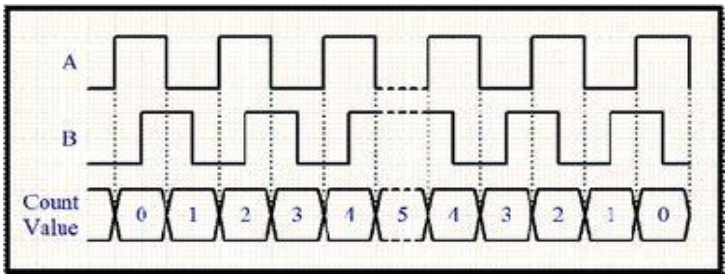


Figure 21 : 2X encoding

With 4X encoding, we check the failing and rising edges of both channel A and channel B. Therefore getting four times the resolution from the original ppr, which 980 ppr.

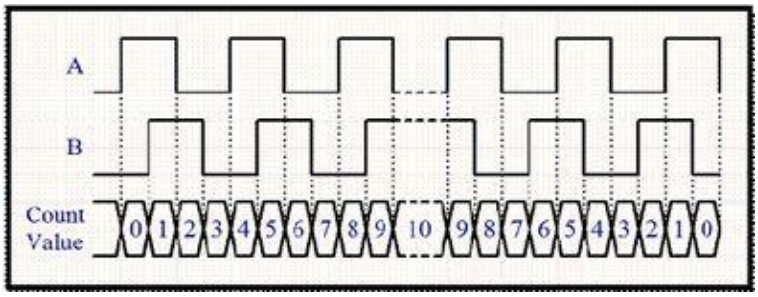


Figure 22: 4X encoding

For this project, I use 4X encoding to maximize the resolution of the motor angle. In Arduino IDE, in order to detect the transition edges, attachInterrupt() is used. AttachIntterupt() calls a specified Interrupt Service Routine (ISR) when interrupt happens. Interrupts are a good way to read rotary encoder as reading rotary encoder requires high timing precision and it need to be done as quickly as possible. The encoder wires for channel A and B are connected to the digital I/O pins of the Arduino Due. Inside the attachInterrupt() function is where the pin number, ISR to call, and the mode is declared. Mode defines when the interrupt should be triggered. Following are the modes:

Mode	**Description**
LOW	When pin is low, interrupt is triggered
CHANGE	When pin changes value, interrupt is triggered
RISING	When pin goes from low to high, interrupt is triggered
FALLING	When pin goes from high to low, interrupt is triggered
HIGH	When pin is high, interrupt is triggered

Table 1 : attachInterrupt() modes.

Below is the flow chart of coding the interrupt routine of the encoder and to determine the direction and pulses of each encoder motors.

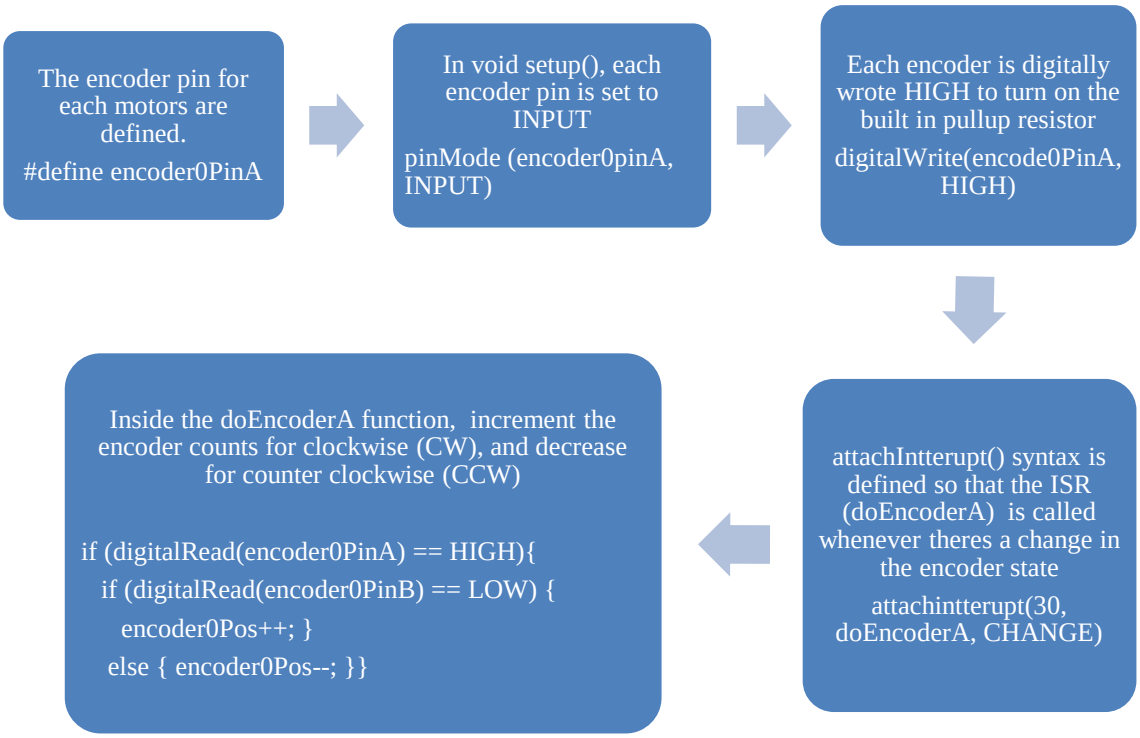


Figure 23 : Process flow of programming the encoder

3.3.4. Angle control of IG32E-35K motor

As Arduino IDE does not have any library for controlling angle for DC motors, this has to be code from scratch. Below is the flow chart of the coding for angle control for DC motors.

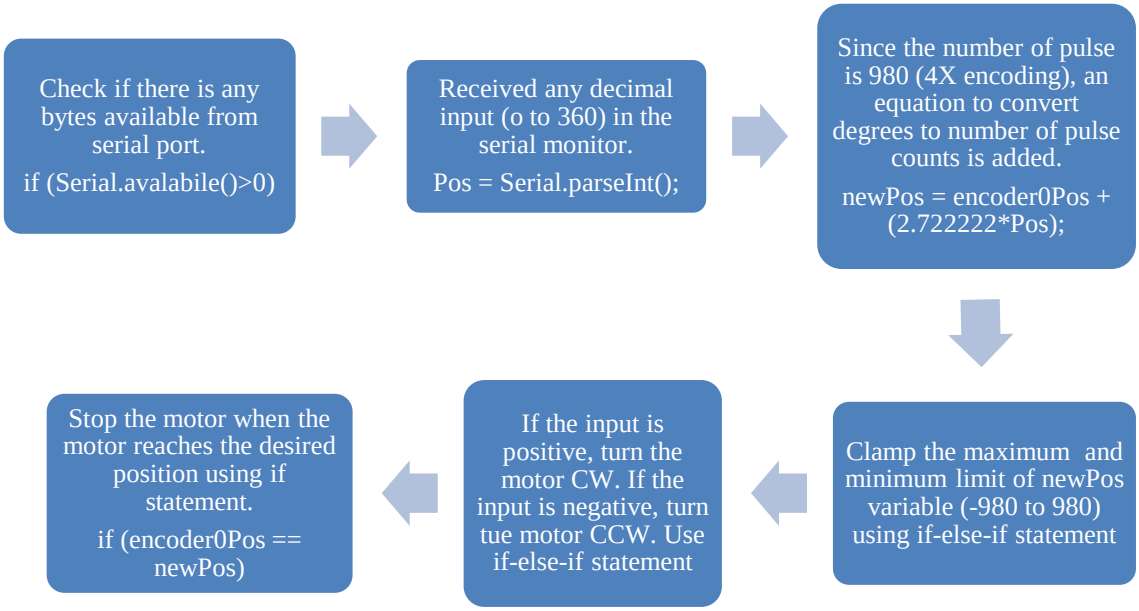


Figure 24 : Flow chart of coding for motor angle control

3.3.5. PID controller

The measured value (input) subtract it with desired value (set-point) would equals to an error. This error is then minimize by the PID controller by adjusting the output. This is the definition of PID control. In a control system, we would want the system's input to be close to the set-point as possible. This is achieved by PID controller. The PID controller would then adjust the output accordingly. In any PID control, tuning the parameters is one of the important aspects. There are three tuning parameters, Kp (Proportional), Ki (Intergral) and Kd (Derivative).

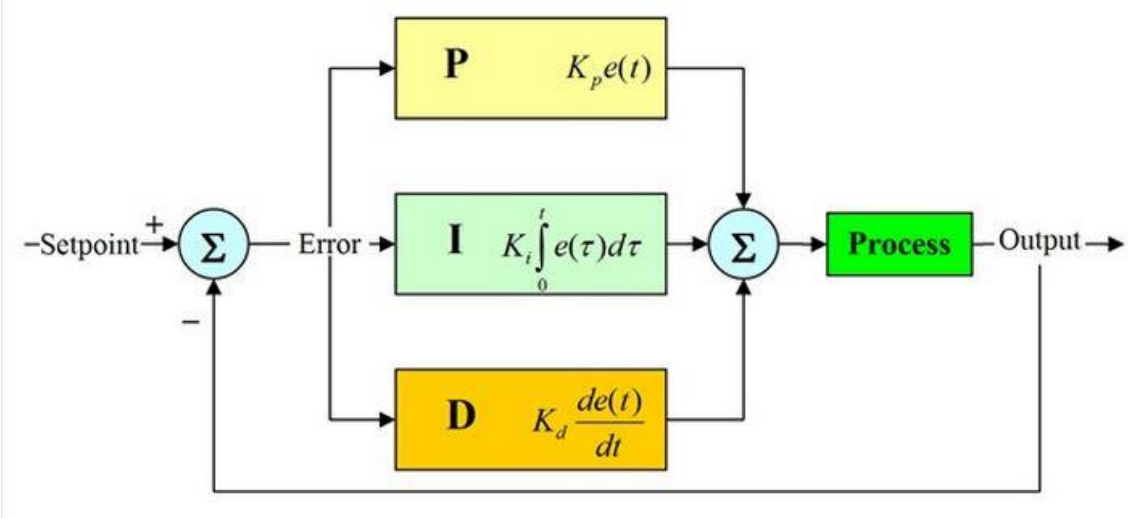


Figure 25 : Block diagram of PID controller

From figure 24, we can see that, Error is equivalent of Set Point minus Process Variable. Output in this system would be the PWM. Each error goes to each P, I and D before it is summed and feed to the process block. Below are the equation governing this PID controller.

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \quad (16)$$

Proportional controller is used to compensate the error present in the system. For example if the desired position is still far away then the current position of the shaft, more power is needed, so that it can reach there faster. If there is an overshoot, it must revere its direction to achieve the desired position. This is why we use Kp. The power of the motor is proportional to the error.

However, Kp will not get the motor to reach its required position as the closer the shaft to the desired position, the smaller the area it would be. As this becomes smaller. PWM also becomes smaller hence not sufficient enough to drive the motor because of friction and gravity. This small error that exist when the system has settled down is known as steady state error. Integral controller is introduced to overcome this as it accumulates the error signals since the start of the system. Only then would the motor reached its desired position.

The last controller would be Derivative controller. Derivative controller speeds up the process to prevent overshoot from happening. This is useful when the process takes a long time to reach its set point. It calculates the rate of change of the error.

All this controllers are rarely used alone. When combined, it takes the advantages of each controller to make a one robust controller. Each of this parameters can be adjust and the output will change accordingly. This is all simplified using Arduino PID library. The tuning is done manually by inputting a set point and observing the current position of the motor shaft u. This is done until the optimum parameter values of the controller is reached.

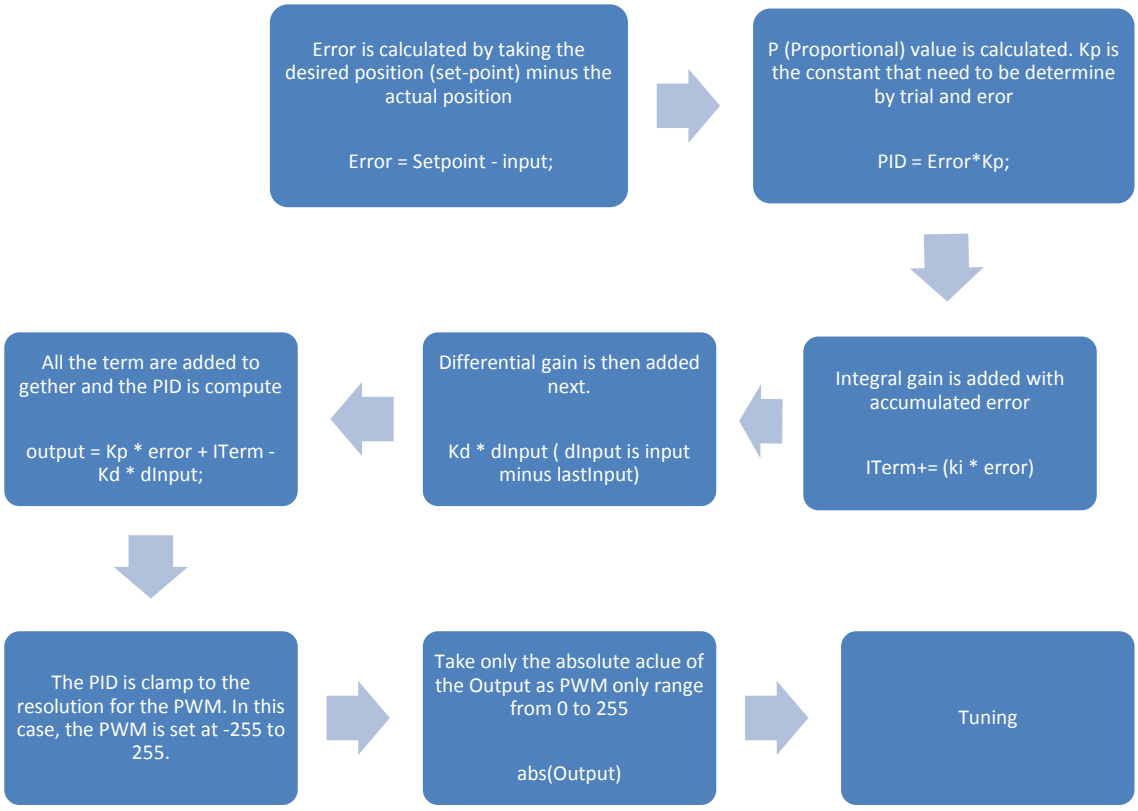


Figure 26 : Flow chart of coding the PID algorithm

3.4. Gantt Chart

Table 2 shows the Gantt chart for FYP I

Table 2 : Gantt Chart for FYP I

Activities	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
First meeting with coordinator and supervisor														
Problem Statement and analysis of Forward kinematics, inverse kinematics and Jacobian Matrix														
Preliminary research work to find the most optimal and easiest way to compute forward and inverse kinematics and literature review.														
Submission of extended proposal defense														
Finding the materials for fabrication of Delta Robot														
Oral proposal defense presentation														
Fabrication of Delta Robot together with the Arduino board.														
Preparation of Interim Report														
Submission of Interim Draft Report														
Submission of Interim Final Report														

Table 3 shows the Gantt chart for FYP II

Table 3 : Gantt Chart FYP II

Activities	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15
Designing and fabricating the Delta robot															
Coding and Debugging the Arduino															
Submission of Progress Report															
Pre-EDX															
Draft Report															
Final Report															
VIVA															

CHAPTER 4
RESULTS AND DISCUSSION

4. RESULTS AND DISCUSSION

This section explains about the finalized design and fabrication of Delta robot along with the circuit schematics of the system. For control system, results of the mathematical model simulation using MATLAB, PID control and tuning, and the mathematical models implementation in Arduino are shown.

4.1. Mechanical Design and Circuit Schematics

4.1.1. Circuit Schematics of the system

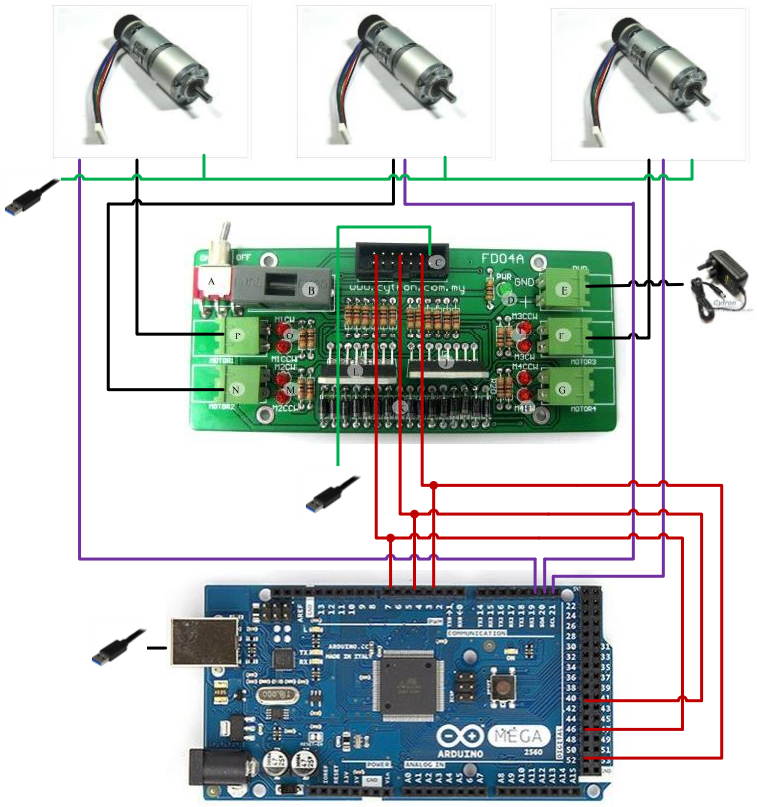


Figure 27 : Circuit Schematic of the system

Figure 19 shows the connection of the Arduino Mega 2560, the motor driver, DC motors and the power supply. Black line from the motor going to the motor driver indicates the Vcc and GND pin of the motor. The purple line connecting the motors to the Arduino are the encoder pin A and pin B. The motor driver connects to the Arduino via the 7 x 2 IDE socket. There are 3 wires for each channel of the motor with wire 1 indicating ON/OFF for Vcc pin, wire 2 indicating ON/OFF for GND pin , and wire 3 indicating speed control (PWM) of the motor. Power supply of 12V and 2A is used to supply the power to the motors via the motor driver. The Arduino's power is supplied by the USB cable of 5V. The details of the schematic wiring is shown in the table 4 below. Refer to figure 26 for the numbering.

Table 4 : List of wiring connections

No	**Motor Driver Pins**	**Arduino Mega 2560 pins**	**DC Motor IG32E-35K**
1	Pin 1	Pin 52	-
2	Pin 2	Pin 5	-
3	Pin 3	Pin 50	-
4	Pin 4	Pin 46	-
5	Pin 5	Pin 6	-
6	Pin 6	Pin 44	-
7	Pin 7	Pin38	-
8	Pin 8	Pin 7	-
9	Pin 9	Pin 40	-
10	Pin 13	-	USB, +
11	Pin 14	-	USB, -
12	-	Pin 21	Motor 2, Encoder A
13	-	Pin 20	Motor 2, Encoder B
14	-	Pin 19	Motor 3, Encoder A
15	-	Pin 18	Motor 3, Encoder B
16	-	Pin 2	Motor 1, Encoder A
17	-	Pin 3	Motor 1, Encoder B
18	Channel 1, CCW	-	Motor 1, Black

19	Channel 1, CW	-	Motor 1, Red
20	Channel 2, CCW	-	Motor 2, Black
21	Channel 2, CW	-	Motor 2, Red
22	Channel 3, CCW	-	Motor 3, Black
23	Channel 3, CW	-	Motor 3, Red
24	Power Supply, GND	-	Power Adapter 12V, GND
25	Power Supply, +	-	Power Adapter 12V, +
26	Motor 1, Encoder Vcc	-	USB, +
27	Motor 1, Encoder GND	-	USB, -
28	Motor 2, Encoder Vcc	-	USB, +
29	Motor 2, Encoder GND	-	USB, -
30	Motor 3, Encoder Vcc	-	USB, +
31	Motor 3, Encoder GND	-	USB, -

4.1.2. Delta Robot design

Below shows the old and new Delta robot design drawn together as a product file in CATIA.

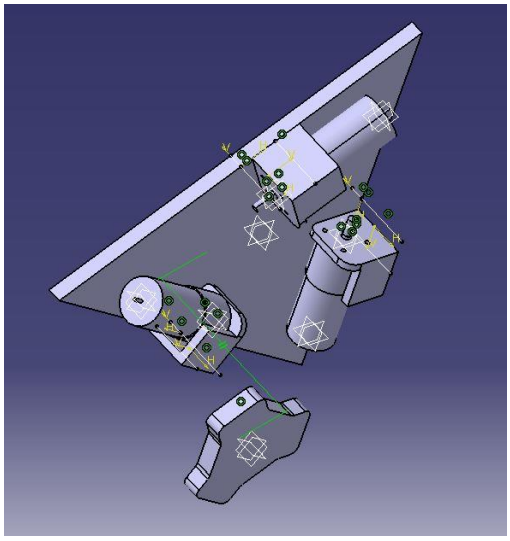


Figure 28 : Old Design of Delta robot.

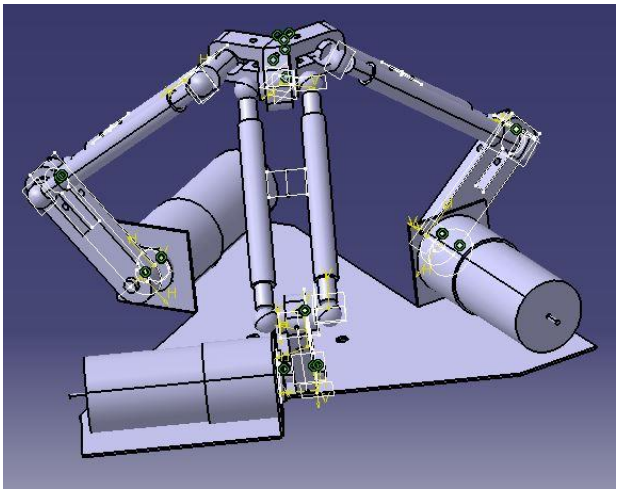


Figure 29 : New Design of Delta robot

Below is the fabricated final prototype of the Delta robot with all of the components attached.

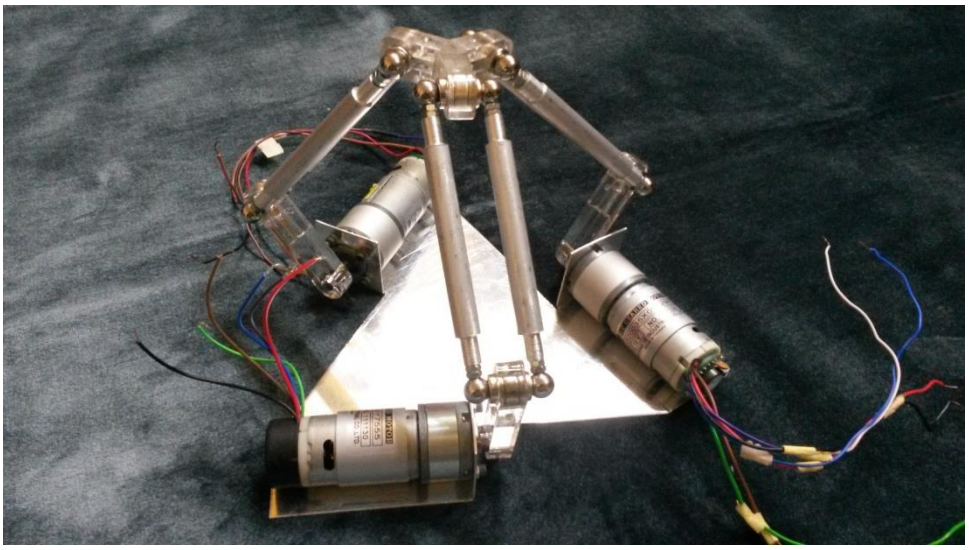


Figure 30 : Delta robot

4.2. Control system of the robot

4.2.1. Inverse and Forward Kinematics in MATLAB

For the software part, most of the time was done for the research and the building of the code for inverse kinematics, forward kinematics, and Jacobian matrix using MATLAB environment. The same value are compared when using forward and inverse kinematics in order to check for the accuracy of the code [see table 5and table 6 below].

Table 5 : Converting motor angles to X, Y, Z coordinates using Forward Kinematics

Joint Angle of Motor 1 (radian)	**Joint Angle of Motor 2 (radian)**	**Joint Angle of Motor 3 (radian)**	**X-coordinate**	**Y-coordinate**	**Z-coordinate**
0	0	0	0	0	-1.2759
0	0	0.1	0.0478	-0.0276	-1.2917
0	0.1	0	-0.0478	-0.0276	-1.2917
0.1	0	0	0	0.0552	-1.2917
0.1	0.1	0.1	0	0	-1.3274
0.2	0.2	0.2	0	0	-1.3814
0.1	0.2	0.3	0.0529	-0.0901	-1.3776

Table 6 : Converting X, Y, Z coordinates to motor angles using Inverse Kinematics

X-coordinate	**Y-coordinate**	**Z-coordinate**	**Joint Angle of Motor 1 (radian)**	**Joint Angle of Motor 2 (radian)**	**Joint Angle of Motor 3 (radian)**
0	0	-1.2759	0	0	0
0.0478	-0.0276	-1.2917	0	0	0.1000
-0.0478	-0.0276	-1.2917	0	0.1000	0
0	0.0552	-1.2917	0.1000	0	0
0	0	-1.3274	0.1000	0.1000	0.1000
0	0	-1.3814	0.2001	0.2001	0.2001
0.0529	-0.0901	-1.3776	0.1000	0.2001	0.3000

Based on table 5 and table 6, we can conclude that the solution of forward kinematics when inputting the radian value will give the coordinates of the end-effector, will is confirmed by the inverse kinematics equation.

4.2.2. Jacobian Matrix

Table 7 below listed the resulting end-effector velocity in terms of X, Y and Z coordinates using Jacobian matrix with $\dot{\theta}_1, \dot{\theta}_2 \; and \; \dot{\theta}_3$ being the angular velocity for the first, second and the third motor respectively. $\theta_1, \theta_2 \; and \; \theta_3$ represent the joint angle for the first, second and third motor respectively. The result would be the end-effector velocity which are denoted as $\bar{X}, \bar{Y} \; and \; \bar{Z}$. Below are the table for the resulting jacobian matrix with three case.

The first case is considering the direction of end-effector moving only in the X-axis, starting from (0, 0, -1) coordinates to (1, 0, -1) coordinates with a constant end-effector velocity of 0.1 m/ s in the direction of increasing X-axis. Table 7 below shows the resulting angular velocity $\dot{\theta}_1, \; \dot{\theta}_2 \; and \; \dot{\theta}_3$.

Table 7: Moving only in X direction with constant velocity.

θ_1	θ_2	θ_3	X	Y	Z	$\bar{X}$	$\bar{Y}$	$\bar{Z}$	$\dot{\theta}_1$	$\dot{\theta}_2$	$\dot{\theta}_3$
0.79	0.79	0.79	0	0	-1	0.1	0	0	-0.0044	0	-0.0027
0.72	1.21	0.35	0.2	0	-1	0.1	0	0	0.0043	0.0004	0.0037
0.53	1.32	-0.076	0.4	0	-1	0.1	0	0	0.016	0.0046	0.017
0.28	1.21	-0.47	0.6	0	-1	0.1	0	0	0.027	0.014	0.032
-0.012	1.06	-0.86	0.8	0	-1	0.1	0	0	0.037	0.028	0.045
-0.36	0.89	1.24	1	0	-1	0.1	0	0	0.019	-0.0021	0.047

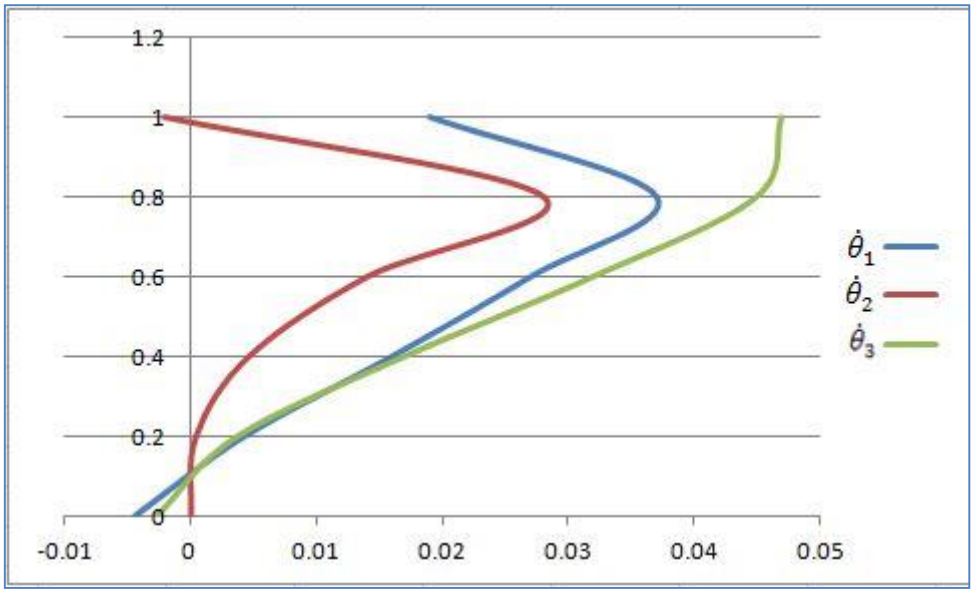


Figure 31 : X coordinates VS $\dot{\theta}_1, \dot{\theta}_2$ and $\dot{\theta}_3$.

The second case is considering the direction of end-effector moving only in the Y-axis, starting from (0, 0, -1) coordinates to (0, 1, -1) coordinates with a constant end-effector velocity of 0.1 m/ s in the direction of increasing Y-axis. Table 8 below shows the resulting angular velocity $\dot{\theta}_1, \dot{\theta}_2$ and $\dot{\theta}_3$.

Table 8: Moving only in Y direction with constant velocity.

θ_1	θ_2	θ_3	X	Y	Z	$\bar{X}$	$\bar{Y}$	$\bar{Z}$	$\dot{\theta}_1$	$\dot{\theta}_2$	$\dot{\theta}_3$
0.79	0.79	0.79	0	0	-1	0	0.1	0	0.0022	-0.0038	-0.0027
0.29	0.97	0.97	0	0.2	-1	0	0.1	0	0.0032	-0.0047	-0.0045
-0.15	0.96	0.96	0	0.4	-1	0	0.1	0	-0.0012	0.0013	0.0015
-0.56	0.82	0.82	0	0.6	-1	0	0.1	0	-0.0109	0.0092	0.011
-0.95	0.63	0.63	0	0.8	-1	0	0.1	0	-0.024	0.016	0.021
-1.32	0.41	0.41	0	1	-1	0	0.1	0	-0.038	0.023	0.0303

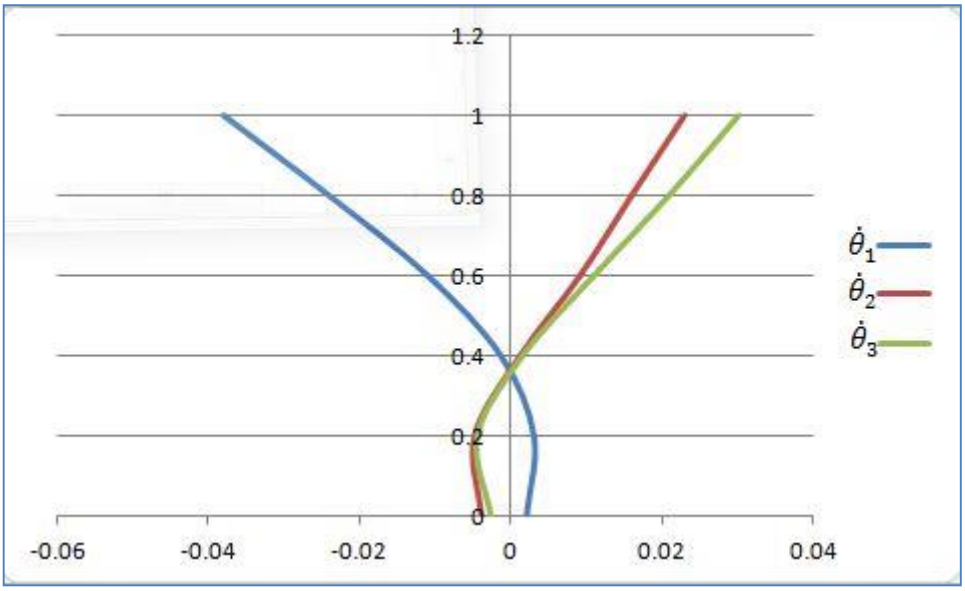


Figure 32: Y coordinates VS $\dot{\theta}_1, \dot{\theta}_2$ and $\dot{\theta}_3$.

The third case is considering the direction of end-effector moving only in the Z-axis, starting from (0, 0, -1) coordinates to (0, 0, -2) coordinates with a constant end-effector velocity of 0.1 m/ s in the direction of increasing Z-axis in the negative direction. Table 9 below shows the resulting angular velocity $\dot{\theta}_1, \dot{\theta}_2$ and $\dot{\theta}_3$.

Table 9 : Moving only in Z direction with constant velocity

θ_1	θ_2	θ_3	X	Y	Z	$\bar{X}$	$\bar{Y}$	$\bar{Z}$	$\dot{\theta}_1$	$\dot{\theta}_2$	$\dot{\theta}_3$
0.79	0.79	0.79	0	0	-1	0	0	0.1	0.0022	0.0038	-0.0027
0.16	0.16	0.16	0	0	-1.2	0	0	0.1	-0.017	-0.029	0.014
-0.23	-0.23	-0.23	0	0	-1.4	0	0	0.1	-0.0305	-0.052	0.018
-0.58	-0.58	-0.58	0	0	-1.6	0	0	0.1	-0.039	-0.069	0.018
-0.97	-0.97	-0.97	0	0	-1.8	0	0	0.1	-0.042	-0.074	0.013

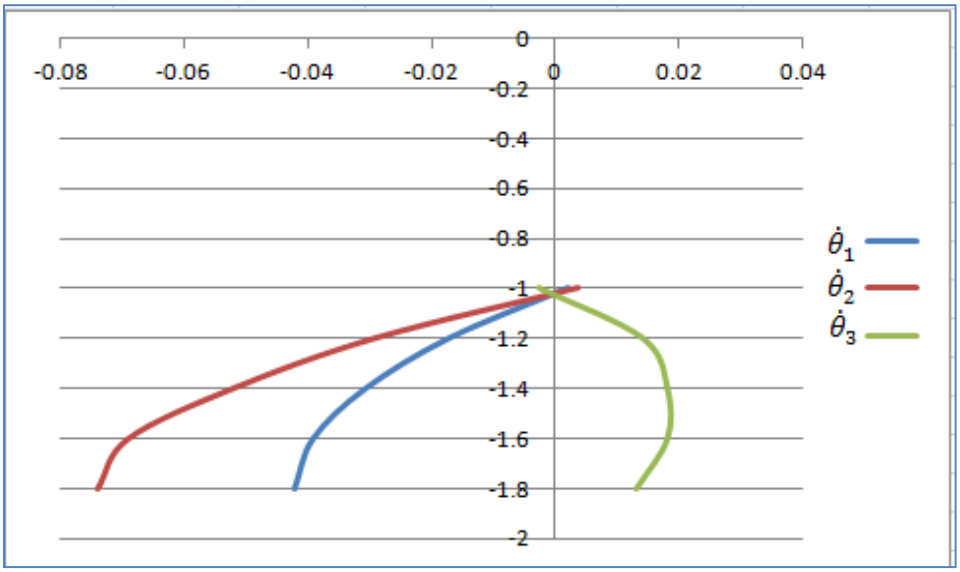


Table 10 : Z coordinates VS $\dot{\theta}_1, \dot{\theta}_2$ and $\dot{\theta}_3$

4.2.3. PID Control and tuning

When PID controller is implemented, the arms would rotate about the desired position for some period of time before it stop oscillates. This is expected from a PID controller. The amount of overshoots (oscillations) highly depends on the Proportional value (Kp). To make sure the arm move towards the set point (desired angular position), Integral value (Ki) needs to be determined. Derivative term (Kd) is only use when we want the motor to move to the set point as quickly as possible. Commonly, Kd is not needed since the motor is already fast enough with Kp and Ki term. But in this project, all P, I and D is use. Each tuning constant need to be define by using trial and error for every motor separately. After the most suitable tuning parameters are known, then the implementation of the mathematical models can proceed. Table 11 below shows the tuning parameters for each motor.

Table 11: P, I and D tuning parameters

	Proportional (Kp)	**Integral (Ki)**	**Derivative (Kd)**
Motor 1	0.4	0.6	0.05
Motor 2	0.6	0.8	0.05
Motor 3	0.4	0.9	0.05

4.2.4. Implementation of Inverse Kinematics and Forward Kinematics in Arduino

The inverse kinematics and forward kinematics functions just as planned. By inputting the coordinates into the serial monitor of the Arduino IDE, the motor moves accordingly. By using a mouse to control the end-effector, it can smoothly move by the actuators with each coordinates of the end-effector at each control points of the movement converted by the inverse kinematics. Forward kinematics can then be use to compare the actual and desired angular position of the DC motors for error correction by the PID controller.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5. CONCLUSION AND RECOMMENDATION

5.1. Conclusion

To achieve a tight and synchronized control of the trajectory of the kinematic parallel robot or Delta robot, the mathematical model of forward and inverse kinematics as well as the Jacobian matrix must be fully understood. Forward kinematics is useful to determine the exact coordinates of the end-effector in 3 dimensional space given only the angle of the actuators. Inverse kinematics are then used in the opposite way of forward kinematics. This is extremely useful when designing the PID controller based on the position of the end-effector to correct any error conduct by the actuators. The Jacobian matrix can be used for speed control using the given joint angle velocities. Smooth trajectory could also be achieved. With all these mathematical method in consideration, the Delta robot can be designed with position and velocity control as well as optimal trajectory planning.

5.2. Recommendation

Recommendation for this project would be to increase the FYP budget. With only RM500, the hardware range is very limited. A better motor and encoder could have been chosen so that better resolution and accuracy can be reach. With small motors, limits the dimension of the robot, therefore smaller workspace. To enable a larger workspace of the Delta robot, longer upper arm and parallelogram could have been considered. Time management also play a large role in managing project. If fabrication of the robot have been done in FYP I, more time can be spent of the software part, making the robot more finely tune.

Proposed future works are to calculate the cubic spline function of the trajectory of the end-effector with GUI for ease of changing the parameters. With all the mathematical models implemented, the trajectory can be further defined by manipulating each control points.

REFERENCES

[1] M. Mustafa, R. Misuari, and H. Daniyal, "Forward Kinematics of 3 Degree of Freedom Delta Robot," in *Research and Development, 2007. SCOReD 2007. 5th Student Conference on*, 2007, pp. 1-4.

[2] A. Codourey, "Dynamic modeling of parallel robots for computed-torque control implementation," *The International Journal of Robotics Research,* vol. 17, pp. 1325-1336, 1998.

[3] L. Hui-Hung, W. Chih-Chin, L. Shi-Wei, T. Yuan-Hung, and L. Chao-Shu, "Robust control for a delta robot," in *SICE Annual Conference (SICE), 2012 Proceedings of*, 2012, pp. 880-885.

[4] S. Staicu and D. C. Carp-Ciocardia, "Dynamic analysis of Clavel's Delta parallel robot," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, 2003, pp. 4116-4121 vol.3.

[5] S. D, *A Platform with Six Degrees of Freedom* vol. 15. Proceedings of the Inst. Mech. Engrs, 1965.

[6] H. J. M and S. F, *Star. A New Concept in Robotics*, 1992.

[7] F. Pierrot, P. Dauchez, and A. Fournier, "HEXA: a fast six-DOF fully-parallel robot," in *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, 1991, pp. 1158-1163 vol.2.

[8] R. Clavel, "Delta A fast robot with parallel geometry.," *Proc. Int. symposium on Industrial Robots,* pp. 91-100, 1988.

[9] L.-W. Tsai, *Robot analysis: the mechanics of serial and parallel manipulators*: John Wiley & Sons, 1999.

[10] R. Paul, "Robot manipulators: mathematics, programming, and control : the computer control of robot manipulators.," 1981.

[11] P. Guglielmetti, "Model-based control of fast parallel robots - a global approach in operational space," EPFL, 1994.

[12] M. Afroun, T. Chettibi, and S. Hanchi, "Planning Optimal Motions for a DELTA Parallel Robot," in *Control and Automation, 2006. MED '06. 14th Mediterranean Conference on*, 2006, pp. 1-6.

APPENDICES

Appendix A : IG32E encoder specification

Appendix B : MATLAB codes for forward kinematics, inverse kinematics and Jacobian matrix.

Appendix C : Delta Robot dimensions

Appendix D: Encoder Coding in Arduino IDE

Appendix E : Full coding with PID controller and Inverse and Forward Kinematics

Appendix A

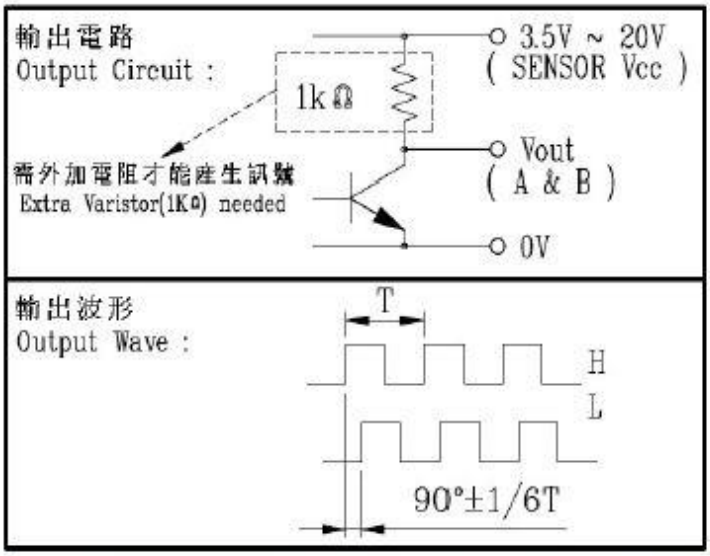
IG32E – Two Channel Hall Effect Encoder

Two Channel Encoder
Connections :

1. Black : -MOTOR
2. Red : +MOTOR
3. Brown : HALL SENSOR Vcc
4. Green : HALL SENSOR GND
5. Blue : HALL SENSOR A Vout
6. Purple : HALL SENSOR B Vout

ELECTRICAL CHARACTERISTICS

規格特性 CHARACTERISTICS	代號 SYMBOL	測試條件 TEST CONDITIONS	最小 MIN.	基準 REF.	最大 MAX.	單位 UNITS
輸入電壓 Supply Voltage	Vcc	---	3.5	–	20	V
輸出飽和電壓 Output Saturation Voltage	Vce(sat)	Vcc=14V ; Ic=20mA	–	300	700	mV
輸出漏電流 Output Leakage Current	Icex	Vce=14V ; Vcc=14V	–	< 0.1	10	μV
輸入電流 Supply Current	Icc	Vcc=20V Output open	–	5	10	mA
輸出上升時間 Output Rise Time	t_r	Vcc=14V ; R_L=820Ω; C_L=20pF	–	0.3	1.5	μS
輸出下降時間 Output Fall Time	t_f	Vcc=14V ; R_L=820Ω; C_L=20pF	–	0.3	1.5	μS



Appendix B

Forward kinematics

```
function [x,y,z] = forward(t1,t2,t3)
    theta1 = t1;
    theta2 = t2;
    theta3 = t3;
% robot geometry

 e = 0.5;     %// end effector - length of 1 side of the
triangle
 f = 1.5;    % // base - length of 1 side of the triangle

 rf = 0.5;
 re = 1.50;

% // trigonometric constants
 sqrt3 = sqrt(3.0);
 pi = 3.141592653;   % // PI
 sin120 = sqrt3/2.0;
 cos120 = -0.5;
 tan60 = sqrt3;
 sin30 = 0.5;
 tan30 = 1/sqrt3;

% // forward kinematics: (theta1, theta2, theta3) -> (x0, y0,
z0)
% // returned status: 0=OK, -1=non-existing position


     t = (f-e)*tan30/2;
     dtr = pi/180.0;

     y1 = -(t + rf*cos(theta1));
     z1 = -rf*sin(theta1);

     y2 = (t + rf*cos(theta2))*sin30;
     x2 = y2*tan60;
     z2 = -rf*sin(theta2);

     y3 = (t + rf*cos(theta3))*sin30;
     x3 = -y3*tan60;
     z3 = -rf*sin(theta3);

     dnm = (y2-y1)*x3-(y3-y1)*x2;

     w1 = y1*y1 + z1*z1;
     w2 = x2*x2 + y2*y2 + z2*z2;
     w3 = x3*x3 + y3*y3 + z3*z3;
```

```
      %// x = (a1*z + b1)/dnm
      a1 = (z2-z1)*(y3-y1)-(z3-z1)*(y2-y1);
      b1 = -((w2-w1)*(y3-y1)-(w3-w1)*(y2-y1))/2.0;

      %// y = (a2*z + b2)/dnm;
      a2 = -(z2-z1)*x3+(z3-z1)*x2;
      b2 = ((w2-w1)*x3 - (w3-w1)*x2)/2.0;

      % // a*z^2 + b*z + c = 0
      a = a1*a1 + a2*a2 + dnm*dnm;
      b = 2*(a1*b1 + a2*(b2-y1*dnm) - z1*dnm*dnm);
      c = (b2-y1*dnm)*(b2-y1*dnm) + b1*b1 + dnm*dnm*(z1*z1 -
re*re);

     % // discriminant
      d = b*b - 4.0*a*c;
      if (d < 0)
      d=-1;% // non-existing point
      end

      z0 = -0.5*(b+sqrt(d))/a;
      x0 = (a1*z0 + b1)/dnm;
      y0 = (a2*z0 + b2)/dnm;

      x=x0;
      y=y0;
      z=z0;


end
```

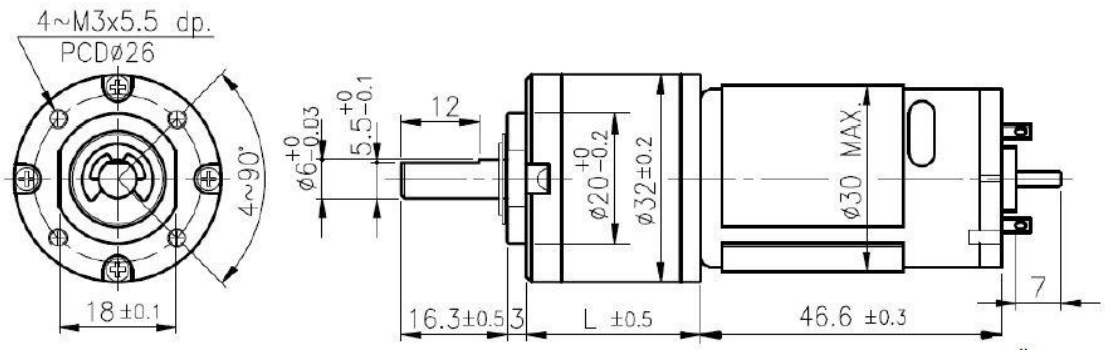
Inverse kinematics

```
function [t1,t2,t3] = inverse (x,y,z)
for n=0:2
if n==0
  x0=x;
  y0=y;
  z0=z;
elseif n==1
  x0=x*cos(2.0944) + y*sin(2.0944); %% +120 degree rotation
  y0=y*cos(2.0944) - x*sin(2.0944);
  z0=z;
else
  x0=x*cos(2.0944) - y*sin(2.0944); %% -120 degree rotation
  y0=y*cos(2.0944) + x*sin(2.0944);
  z0=z;

end
 % robot geometry
     e = 0.5;     %// end effector - length of 1 side of the triangle
     f = 1.5;    % // base - length of 1 side of the triangle

     rf = 0.5; % arm near motor
     re = 1.50; % parallelogram

     y1 = -0.5*0.57735*f; %// f/2 * tg 30
     y01 = y0-(0.5*0.57735*e);   % // shift center to edge
     %// z = a + b*y
     a = (x0*x0 + y01*y01 + z0*z0 +rf*rf - re*re - y1*y1)/(2*z0);
     b = (y1-y01)/z0;
     %// discriminant
     d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
     if (d < 0)
       d= -1;
     end %// non-existing point
     yj = (y1 - a*b - sqrt(d))/(b*b + 1); %// choosing outer point
     zj = a + b*yj;
     theta = atand(zj/(y1 - yj));
     theta=theta/180*pi;
     if n==0
       t1=theta;
     elseif n==1
       t2=theta;
     else
       t3=theta;
     end
end


end
```

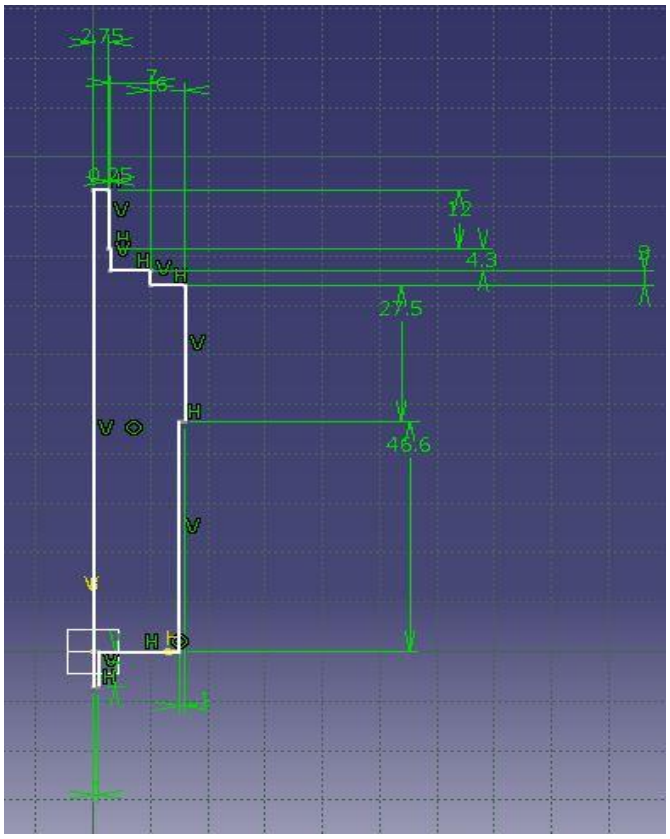
Jacobian Matrix

```
function v = jacobm(t1,t2,t3,qb1,qb2,qb3,x,y,z)
%e = 0.5;     %// end effector - length of 1 side of the triangle
f = 1.5;    % // base - length of 1 side of the triangle

rf = 0.5; % arm near motor
re = 1.50; % parallelogram

LA=rf;
f1=f/(2*sqrt(3));
OA=[f1; 0; 0];
OB=transpose([x y z]); %position of end-effector
qbar=transpose([qb1,qb2,qb3]);
for n=0:2
if n==0
  zeta=0;
  theta=t1;
elseif n==1
  zeta=2.0944;
  theta=t2;
else
  zeta=-2.0944;
  theta=t3;
end
R=[cos(zeta) -sin(zeta) 0; sin(zeta) cos(zeta) 0; 0 0 1];
AC=[LA*cos(theta); 0; -LA*sin(theta)];
S=OB-R*(OA+AC);
ACder=[LA*sin(theta); 0; -LA*cos(theta)];
B=R*ACder;
S_T = transpose(S);
if n==0
    B1 = S_T*B;
    S1_T=S_T;
elseif n==1
    B2 = S_T*B;
    S2_T=S_T;
else
    B3 = S_T*B;
    S3_T=S_T;
    B = [B1 0 0; 0 B2 0; 0 0 B3];
    ST=[S1_T; S2_T; S3_T];
    J = -(inv(ST))*B;

    v=J*qbar;

end




end
```

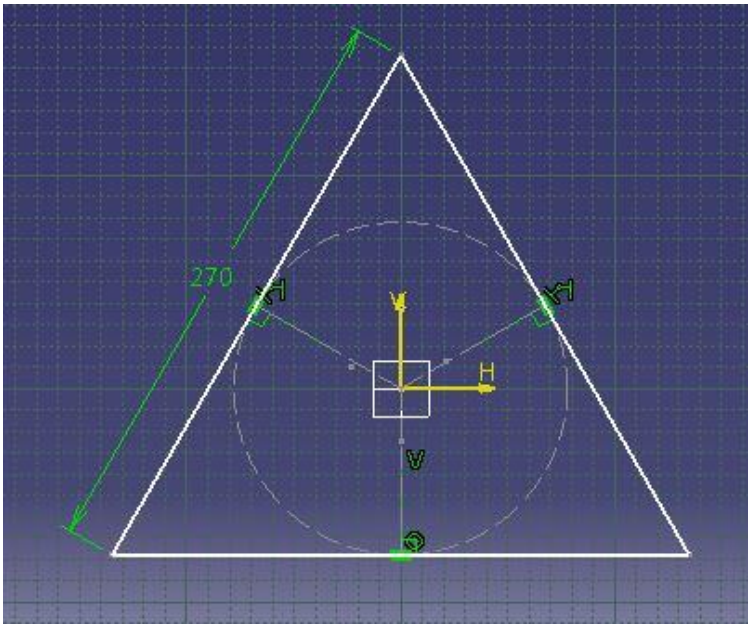
Appendix C



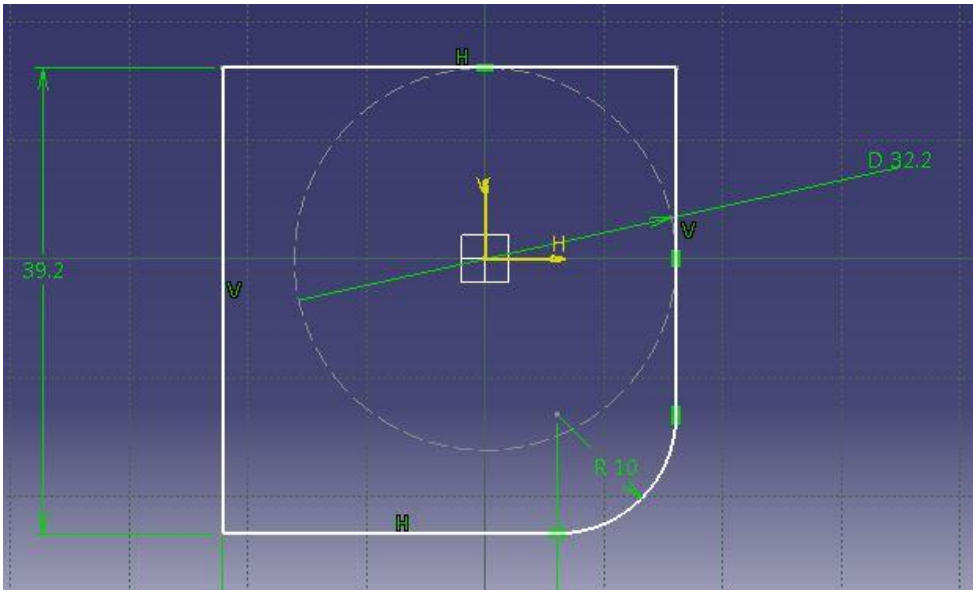
Front and side view of Planetery DC Gear Motor with dimensions



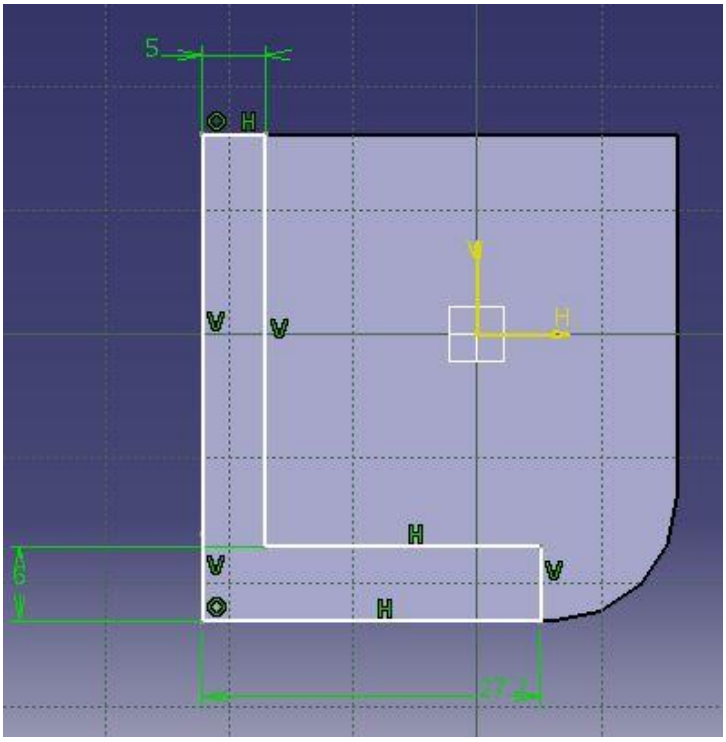
Planetary DC motor dimensions before applying shaft technique in CATIA



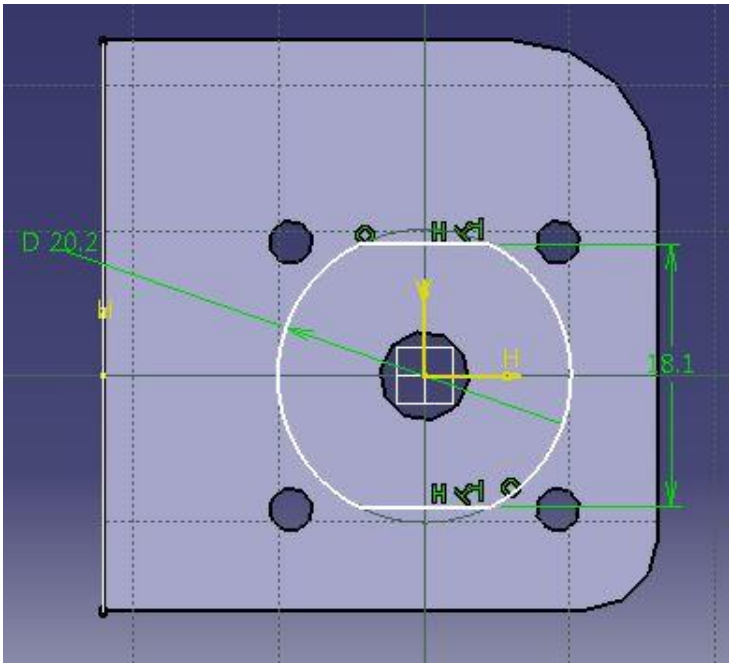
Base of the Delta Robot's dimension



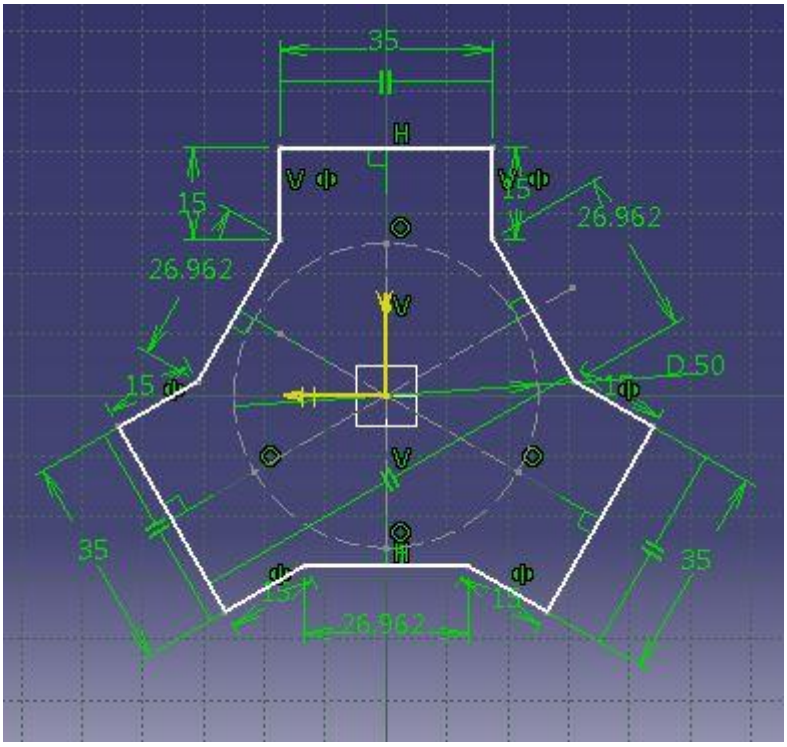
DC motor holder dimensions 1



DC motor holder dimensions 2



DC motor holder dimensions 3



End-effector dimensions

Appendix D : Encoder coding in Arduino IDE

```
void setup() {
  pinMode(encoder0PinA, INPUT);
  digitalWrite(encoder0PinA, HIGH);
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);

  attachInterrupt(30, doEncoderA, CHANGE);
  attachInterrupt(32, doEncoderB, CHANGE);

  Serial.begin (115200);
  Serial.println("start");
  pinMode(mtr2_p1, OUTPUT);
  pinMode(mtr2_p2, OUTPUT);
  pinMode(mtr2_spd, OUTPUT);
}

void loop(){
Serial.println(encoder0Pos,DEC);
digitalWrite(mtr2_p1, LOW);
digitalWrite(mtr2_p2, HIGH);
analogWrite(mtr2_spd, 50);



}
void doEncoderA(){
  if (digitalRead(encoder0PinA) == HIGH) {

    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos++;         // CW
```

```
    }
    else {
      encoder0Pos--;          // CCW
    }

   if (encoder0Pos > 979) {
   encoder0Pos = encoder0Pos - 980;
   } else if (encoder0Pos < -979) {
   encoder0Pos = encoder0Pos + 980;
   }
}
  else
  {

    if (digitalRead(encoder0PinB) == HIGH) {
      encoder0Pos++;           // CW
    }
    else {
      encoder0Pos--;           // CCW
    }
  }

   if (encoder0Pos > 979) {
   encoder0Pos = encoder0Pos - 980;
   } else if (encoder0Pos < -979) {
   encoder0Pos = encoder0Pos + 980;
   }
  }

void doEncoderB(){
```

```
if (digitalRead(encoder0PinB) == HIGH) {

  if (digitalRead(encoder0PinA) == HIGH) {
    encoder0Pos = encoder0Pos + 1;         // CW
  }
  else {
    encoder0Pos = encoder0Pos - 1;         // CCW
  }
 if (encoder0Pos > 979) {
 encoder0Pos = encoder0Pos - 980;
 } else if (encoder0Pos < -979) {
 encoder0Pos = encoder0Pos + 980;
 }
}

else {

  if (digitalRead(encoder0PinA) == LOW) {
    encoder0Pos = encoder0Pos + 1;          // CW
  }
  else {
    encoder0Pos = encoder0Pos - 1;          // CCW
  }
   if (encoder0Pos > 979) {
 encoder0Pos = encoder0Pos - 980;
 } else if (encoder0Pos < -979) {
 encoder0Pos = encoder0Pos + 980;
 }
} }
```

Appendix E : Full coding with PID controller and Inverse and Forward Kinematics

```
#include <PID_v1.h> //PID Algorithm library header

#define encoder0PinA 2 //encoder for motor 1
#define encoder0PinB 3 //

#define encoder0PinC 21 //encoder for motor 2
#define encoder0PinD 20 //

#define encoder0PinE 19 //encoder for motor 3
#define encoder0PinF 18 //

volatile long encoder0Pos = 0;
volatile long encoder0Pos2 = 0;
volatile long encoder0Pos3 = 0;

const int mtr1_p1 = 52; // declare motor_1 p1, p2 and speed
const int mtr1_p2 = 50; //
const int mtr1_spd = 5; //

const int mtr2_p1 = 46; // declare motor_2 p1, p2 and speed
const int mtr2_p2 = 44; //
const int mtr2_spd = 6; //

const int mtr3_p1 = 40; // declare motor_3 p1, p2 and speed
const int mtr3_p2 = 38; //
const int mtr3_spd = 7; //
```

```
// robot geometry
const float e = 48.301;      // end effector width in mm
const float f = 173.205;     // base width in mm
const float re = 108.0;      // parallelogram length in mm
const float rf = 50.0;      // upperarm length in mm

// trigonometric constants
const float sqrt3 = sqrt(3.0);
const float pi = 3.141592653;    // PI
const float sin120 = sqrt3 / 2.0;
const float cos120 = -0.5;
const float tan60 = sqrt3;
const float sin30 = 0.5;
const float tan30 = 1 / sqrt3;

float x, y, z, t1, t2, t3, xEE, yEE, zEE;
int newPos, newPos2, newPos3;

//Define Variables we'll be connecting to
double Setpoint, Input, Output;
double Setpoint2, Input2, Output2;
double Setpoint3, Input3, Output3;

//Specify the links and initial tuning parameters
PID myPID(&Input, &Output, &Setpoint, 3, 1.5, 0, DIRECT);
PID myPID2(&Input2, &Output2, &Setpoint2, 1.8, 1.3, 0, DIRECT);
PID myPID3(&Input3, &Output3, &Setpoint3, 2, 1, 0, DIRECT);

void setup() {
```

```
pinMode(encoder0PinA, INPUT);
digitalWrite(encoder0PinA, HIGH);       // turn on pullup resistor
pinMode(encoder0PinB, INPUT);
digitalWrite(encoder0PinB, HIGH);       // turn on pullup resistor

pinMode(encoder0PinC, INPUT);
digitalWrite(encoder0PinC, HIGH);       // turn on pullup resistor
pinMode(encoder0PinD, INPUT);
digitalWrite(encoder0PinD, HIGH);       // turn on pullup resistor

pinMode(encoder0PinE, INPUT);
digitalWrite(encoder0PinE, HIGH);       // turn on pullup resistor
pinMode(encoder0PinF, INPUT);
digitalWrite(encoder0PinF, HIGH);       // turn on pullup resistor

attachInterrupt(0, doEncoderA, CHANGE);
attachInterrupt(1, doEncoderB, CHANGE);

attachInterrupt(2, doEncoderC, CHANGE);
attachInterrupt(3, doEncoderD, CHANGE);

attachInterrupt(4, doEncoderE, CHANGE);
attachInterrupt(5, doEncoderF, CHANGE);

pinMode(mtr1_p1, OUTPUT);
pinMode(mtr1_p2, OUTPUT);
pinMode(mtr1_spd, OUTPUT);

pinMode(mtr2_p1, OUTPUT);
pinMode(mtr2_p2, OUTPUT);
```

```
  pinMode(mtr2_spd, OUTPUT);

  pinMode(mtr3_p1, OUTPUT);
  pinMode(mtr3_p2, OUTPUT);
  pinMode(mtr3_spd, OUTPUT);

  //turn the PID on
    myPID.SetOutputLimits(-255, 255);
    myPID2.SetOutputLimits(-255, 255);
    myPID3.SetOutputLimits(-255, 255);

    myPID.SetMode(AUTOMATIC);
    myPID2.SetMode(AUTOMATIC);
    myPID3.SetMode(AUTOMATIC);

  Serial.begin (115200);
  Serial.println("start");
}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%               ENCODER FOR MOTOR 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\\
void doEncoderA() {

  if (digitalRead(encoder0PinA) == HIGH) {

    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos++;         // CW
    }
    else {
```

```
      encoder0Pos--;         // CCW
    }

    if (encoder0Pos > 979) {
      encoder0Pos = encoder0Pos - 980;
    } else if (encoder0Pos < -979) {
      encoder0Pos = encoder0Pos + 980;
    }
  }
  else
  {

    if (digitalRead(encoder0PinB) == HIGH) {
      encoder0Pos++;          // CW
    }
    else {
      encoder0Pos--;          // CCW
    }
  }

  if (encoder0Pos > 979) {
    encoder0Pos = encoder0Pos - 980;
  } else if (encoder0Pos < -979) {
    encoder0Pos = encoder0Pos + 980;
  }

}

void doEncoderB() {
```

```
if (digitalRead(encoder0PinB) == HIGH) {

  if (digitalRead(encoder0PinA) == HIGH) {
    encoder0Pos = encoder0Pos + 1;         // CW
  }
  else {
    encoder0Pos = encoder0Pos - 1;         // CCW
  }

  if (encoder0Pos > 979) {
    encoder0Pos = encoder0Pos - 980;
  } else if (encoder0Pos < -979) {
    encoder0Pos = encoder0Pos + 980;
  }

}

else {

  if (digitalRead(encoder0PinA) == LOW) {
    encoder0Pos = encoder0Pos + 1;          // CW
  }
  else {
    encoder0Pos = encoder0Pos - 1;          // CCW
  }
  if (encoder0Pos > 979) {
    encoder0Pos = encoder0Pos - 980;
  } else if (encoder0Pos < -979) {
    encoder0Pos = encoder0Pos + 980;
  }
```

```
  }

}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%               ENCODER FOR MOTOR 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\\
void doEncoderC() {

  if (digitalRead(encoder0PinC) == HIGH) {

    if (digitalRead(encoder0PinD) == LOW) {
      encoder0Pos2++;         // CW
    }
    else {
      encoder0Pos2--;         // CCW
    }

    if (encoder0Pos2 > 979) {
      encoder0Pos2 = encoder0Pos2 - 980;
    } else if (encoder0Pos2 < -979) {
      encoder0Pos2 = encoder0Pos2 + 980;
    }
  }
  else
  {

    if (digitalRead(encoder0PinD) == HIGH) {
      encoder0Pos2++;          // CW
    }
    else {
```

```
      encoder0Pos2--;          // CCW
    }
  }

  if (encoder0Pos2 > 979) {
    encoder0Pos2 = encoder0Pos2 - 980;
  } else if (encoder0Pos2 < -979) {
    encoder0Pos2 = encoder0Pos2 + 980;
  }

}

void doEncoderD() {

  if (digitalRead(encoder0PinD) == HIGH) {

    if (digitalRead(encoder0PinC) == HIGH) {
      encoder0Pos2 = encoder0Pos2 + 1;         // CW
    }
    else {
      encoder0Pos2 = encoder0Pos2 - 1;         // CCW
    }

    if (encoder0Pos2 > 979) {
      encoder0Pos2 = encoder0Pos2 - 980;
    } else if (encoder0Pos2 < -979) {
      encoder0Pos2 = encoder0Pos2 + 980;
    }

  }
```

```
  else {

    if (digitalRead(encoder0PinC) == LOW) {
      encoder0Pos2 = encoder0Pos2 + 1;          // CW
    }
    else {
      encoder0Pos2 = encoder0Pos2 - 1;          // CCW
    }
    if (encoder0Pos2 > 979) {
      encoder0Pos2 = encoder0Pos2 - 980;
    } else if (encoder0Pos2 < -979) {
      encoder0Pos2 = encoder0Pos2 + 980;
    }
  }

}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%        ENCODER FOR MOTOR 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\\
void doEncoderE() {

  if (digitalRead(encoder0PinE) == HIGH) {

    if (digitalRead(encoder0PinF) == LOW) {
      encoder0Pos3++;         // CW
    }
    else {
      encoder0Pos3--;         // CCW
    }
```

```
    if (encoder0Pos3 > 979) {
      encoder0Pos3 = encoder0Pos3 - 980;
    } else if (encoder0Pos3 < -979) {
      encoder0Pos3 = encoder0Pos3 + 980;
    }
  }
  else   // must be a high-to-low edge on channel A
  {

    if (digitalRead(encoder0PinF) == HIGH) {
      encoder0Pos3++;          // CW
    }
    else {
      encoder0Pos3--;          // CCW
    }
  }

  if (encoder0Pos3 > 979) {
    encoder0Pos3 = encoder0Pos3 - 980;
  } else if (encoder0Pos3 < -979) {
    encoder0Pos3 = encoder0Pos3 + 980;
  }

}

void doEncoderF() {

  if (digitalRead(encoder0PinF) == HIGH) {
```

```
  if (digitalRead(encoder0PinE) == HIGH) {
    encoder0Pos3 = encoder0Pos3 + 1;         // CW
  }
  else {
    encoder0Pos3 = encoder0Pos3 - 1;         // CCW
  }

  if (encoder0Pos3 > 979) {
    encoder0Pos3 = encoder0Pos3 - 980;
  } else if (encoder0Pos3 < -979) {
    encoder0Pos3 = encoder0Pos3 + 980;
  }

}

else {

  if (digitalRead(encoder0PinE) == LOW) {
    encoder0Pos3 = encoder0Pos3 + 1;          // CW
  }
  else {
    encoder0Pos3 = encoder0Pos3 - 1;          // CCW
  }
  if (encoder0Pos3 > 979) {
    encoder0Pos3 = encoder0Pos3 - 980;
  } else if (encoder0Pos3 < -979) {
    encoder0Pos3 = encoder0Pos3 + 980;
  }
}
```

```
}



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      forward kinematics: (theta1, theta2,
theta3) -> (x0, y0, z0)     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\\


int delta_calcForward(float theta1, float theta2, float theta3, float &x0,
float &y0, float &z0) {


  float t = (f - e) * tan30 / 2;

  float deg_to_rad = pi / (float)180.0;


  theta1 *= deg_to_rad;

  theta2 *= deg_to_rad;

  theta3 *= deg_to_rad;


  float y1 = -(t + rf * cos(theta1));

  float z1 = -rf * sin(theta1);


  float y2 = (t + rf * cos(theta2)) * sin30;

  float x2 = y2 * tan60;

  float z2 = -rf * sin(theta2);


  float y3 = (t + rf * cos(theta3)) * sin30;

  float x3 = -y3 * tan60;

  float z3 = -rf * sin(theta3);


  float dnm = (y2 - y1) * x3 - (y3 - y1) * x2;


  float w1 = y1 * y1 + z1 * z1;

  float w2 = x2 * x2 + y2 * y2 + z2 * z2;
```

```
  float w3 = x3 * x3 + y3 * y3 + z3 * z3;

  // x = (a1*z + b1)/dnm
  float a1 = (z2 - z1) * (y3 - y1) - (z3 - z1) * (y2 - y1);
  float b1 = -((w2 - w1) * (y3 - y1) - (w3 - w1) * (y2 - y1)) / 2.0;

  // y = (a2*z + b2)/dnm;
  float a2 = -(z2 - z1) * x3 + (z3 - z1) * x2;
  float b2 = ((w2 - w1) * x3 - (w3 - w1) * x2) / 2.0;

  // a*z^2 + b*z + c = 0
  float a = a1 * a1 + a2 * a2 + dnm * dnm;
  float b = 2 * (a1 * b1 + a2 * (b2 - y1 * dnm) - z1 * dnm * dnm);
  float c = (b2 - y1 * dnm) * (b2 - y1 * dnm) + b1 * b1 + dnm * dnm * (z1 *
z1 - re * re);

  // discriminant
  float d = b * b - (float)4.0 * a * c;
  if (d < 0) return -1; // non-existing point

  z0 = -(float)0.5 * (b + sqrt(d)) / a;
  x0 = (a1 * z0 + b1) / dnm;
  y0 = (a2 * z0 + b2) / dnm;
  return 0;
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                     inverse kinematics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\\

int delta_calcAngleYZ(float x0, float y0, float z0, float &theta) {
```

```
  float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30

  y0 -= 0.5 * 0.57735    * e;    // shift center to edge

  // z = a + b*y

  float a = (x0 * x0 + y0 * y0 + z0 * z0 + rf * rf - re * re - y1 * y1) / (2
* z0);

  float b = (y1 - y0) / z0;

  // discriminant

  float d = -(a + b * y1) * (a + b * y1) + rf * (b * b * rf + rf);

  if (d < 0) return -1; // non-existing point

  float yj = (y1 - a * b - sqrt(d)) / (b * b + 1); // choosing outer point

  float zj = a + b * yj;

  theta = 180.0 * atan(-zj / (y1 - yj)) / pi + ((yj > y1) ? 180.0 : 0.0);

  return 0;

}


// inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)


int delta_calcInverse(float x0, float y0, float z0, float &theta1, float
&theta2, float &theta3) {

  theta1 = theta2 = theta3 = 0;

  int status_delta = delta_calcAngleYZ(x0, y0, z0, theta1);

  if (status_delta == 0) {

    status_delta = delta_calcAngleYZ(x0 * cos120 + y0 * sin120, y0 * cos120 -
x0 * sin120, z0, theta2); // rotate coords to +120 deg

  }

  if (status_delta == 0) {

    status_delta = delta_calcAngleYZ(x0 * cos120 - y0 * sin120, y0 * cos120 +
x0 * sin120, z0, theta3); // rotate coords to -120 deg

  }

  return status_delta;

}
```

```
void loop() {

  if (Serial.available() > 0) {
    x = Serial.parseInt();
    y = Serial.parseInt();
    z = Serial.parseInt();
    delta_calcInverse(x, y, z, t1, t2, t3);
    newPos = (2.722222 * t1);
    newPos2 = (2.722222 * t2);
    newPos3 = (2.722222 * t3);

    //Clamping the converted angle to encoder values to -980 to 980
    if (newPos > 979) {
      newPos = newPos - 980;
    }
    else if (newPos < -979) {
      newPos = newPos + 980;
    }

    if (newPos2 > 979) {
      newPos2 = newPos2 - 980;
    }
    else if (newPos2 < -979) {
      newPos2 = newPos2 + 980;
    }

    if (newPos3 > 979) {
      newPos3 = newPos3 - 980;
```

```
    }
    else if (newPos3 < -979) {
      newPos3 = newPos3 + 980;
    }

  }
    Setpoint = newPos;
    Input = encoder0Pos;
    myPID.Compute();
    Output=abs(Output);

    Setpoint2 = newPos2;
    Input2 = encoder0Pos2;
    myPID2.Compute();
    Output2=abs(Output2);

    Setpoint3 = newPos3;
    Input3 = encoder0Pos3;
    myPID3.Compute();
    Output3=abs(Output3);

//  int spd = 70;
  do {
    if (newPos > encoder0Pos) {              //%%%%%%%%%%%%%%%%%%%%%%%%%
MOTOR 1  %%%%%%%%%%%%%%%%%%%%%%\\
      digitalWrite(mtr1_p1, LOW);
      digitalWrite(mtr1_p2, HIGH);
      analogWrite(mtr1_spd, Output);
    }
    if (newPos < encoder0Pos) {
```

```
      digitalWrite(mtr1_p1, HIGH);
      digitalWrite(mtr1_p2, LOW);
      analogWrite(mtr1_spd, Output);
    }

    if (newPos2 > encoder0Pos2) {        //%%%%%%%%%%%%%%%%%%%%%%%%  MOTOR 2
%%%%%%%%%%%%%%%%%%%%%%\\
      digitalWrite(mtr2_p1, LOW);
      digitalWrite(mtr2_p2, HIGH);
      analogWrite(mtr2_spd, Output2);
    }
    if (newPos2 < encoder0Pos2) {
      digitalWrite(mtr2_p1, HIGH);
      digitalWrite(mtr2_p2, LOW);
      analogWrite(mtr2_spd, Output2);
    }
    if (newPos3 > encoder0Pos3) {      //%%%%%%%%%%%%%%%%%%%%%%%%  MOTOR 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%\\
      digitalWrite(mtr3_p1, LOW);
      digitalWrite(mtr3_p2, HIGH);
      analogWrite(mtr3_spd, Output3);
    }
    if (newPos3 < encoder0Pos3) {
      digitalWrite(mtr3_p1, HIGH);
      digitalWrite(mtr3_p2, LOW);
      analogWrite(mtr3_spd, Output3);
    }

  } while ( (encoder0Pos != newPos && encoder0Pos2 != newPos2 &&
encoder0Pos3 != newPos3) );
  digitalWrite(mtr1_p1, LOW);
```

```
  digitalWrite(mtr1_p2, LOW);
  digitalWrite(mtr2_p1, LOW);
  digitalWrite(mtr2_p2, LOW);
  digitalWrite(mtr3_p1, LOW);
  digitalWrite(mtr3_p2, LOW);



}
```