Chapter 4: Delta Robot

4.1 Introduction

In this section we area to discuss the different types of manipulators, then it is cleared why delta robot is chose over the other types of both serial and parallel manipulator, after that workspace analysis using forward kinematics technique for the delta robot is done so that we can understand and visualize the region in which the robot can move and hence ensure that the size of the workspace can cover the area on the conveyor belt and reach any point on it in the specified region using **MATLAB™**, after that we mentioned the dynamic model which include kinematic analysis such as inverse kinematics, and kinetic analysis such as torque analysis, which is very helpful in motor selection phase, after that we indicate the motion simulation done in **V-REP™** which gives the user the ability to visualize the workspace and how robot moves and it can be used to ensure workspace analysis as if the point is out of the workspace an error is shown, then the structure of the delta robot is investigated in **ANSYS™** so that we ensure both suitable design and decreasing the price while maintaining the robot stiff enough to lift demanded objects, we also used it for motor selection as we applied torque measurement tool on the robot and deduced the maximum torque needed, lastly we sketched the model on **SOILDWORKS™** so that it can be fabricated based on what is available in the market.so this is considered as a complete thesis to make a desired delta robot from scratch. [65]

4.2 Type of manipulator

After the wastes are recognized by the computer vision algorithm, it is needed to pick the desired waste and sort it based on its type. There are two types of robotic arms that can accomplish this task, both types are analyzed, giving reasons why we chose the delta robot.

Robots can be classified according various criteria, such as degrees of freedom, structure, drive technology, workspace, motion characteristics, and control. Most robot applications require rigidity. The main classification in robotics is based on the kinematic structure which is dived into serial manipulator and parallel manipulators as indicated in **Fig.4.1**.

Serial robots may achieve rigidity by using high-quality rotary joints that permit movement in one axis but are rigid against movement outside this. Any joint permitting movement must also have this movement under deliberate control by an actuator. A movement requiring several axes thus requires a number of such joints. Unwanted flexibility or sloppiness in one joint cause a similar sloppiness in the arm, which may be amplified by the distance between

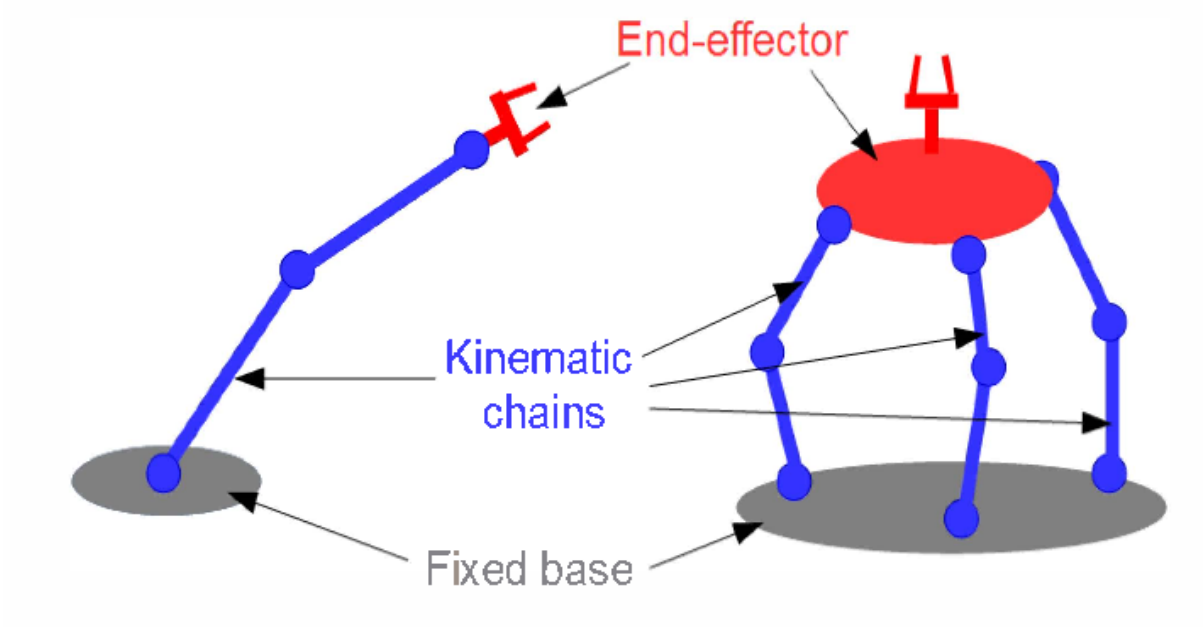


Fig.4.1: Structure of serial and parallel robots

the joint and the end-effector: there is no opportunity to brace one joint's movement against another. Their inevitable hysteresis and off-axis flexibility accumulates along the kinematic chain; a precision serial manipulator is a compromise between precision, complexity, mass of manipulator and payload, and cost. [65]

On the other hand, with parallel manipulators, a high rigidity may be obtained with a relatively small mass of the manipulator. This allows high precision and high speed of movements, and motivates the use of parallel manipulators in flight simulators (high speed with rather large masses) and electrostatic or magnetic lenses in particle accelerators (very high precision in positioning large masses).

A drawback of parallel manipulators, in comparison to serial manipulators, is their limited workspace. As for serial manipulators, the workspace is limited by the geometrical and mechanical limits of the design (collisions between legs). The workspace is also limited by the existence of *singularities*, which are positions where, for some trajectories of the movement, the variation of the lengths of the legs is infinitely smaller than the variation of the position. Conversely, at a singular position, a force (like gravity) applied on the end-effector induce infinitely large constraints on the legs, which may result in a kind of "explosion" of the manipulator so these singular points must be avoided during the operation to ensure the healthy operation of the robot . The determination of the singular positions is difficult in parallel manipulators. This implies that the workspaces of the parallel manipulators are, usually, artificially limited to a small region where one knows that there is no singularity. Another

drawback of parallel manipulators is their nonlinear behavior: the command which is needed for getting a linear or a circular movement of the end-effector depends dramatically on the location in the workspace and does not vary linearly during the movement. [65]

Parallel manipulators are widely popular recently even though conventional serial manipulators possess large workspace and dexterous maneuverability. The basic problems with serial one are their cantilever structure makes them susceptible to bending at high load and vibration at high speed leading to lack of precision and other problems. In this paper, parallel manipulators advantages over the serial one are compared. Hence, in applications demanding high load carrying capacity and precise positioning, the parallel manipulators are the better alternatives and the last two decades points to the potential embedded in this structure that has not yet been fully exploited.

Among the advantages of parallel manipulators are greater load carrying capacities as total load can be shared by number of parallel links connected to fixed base, low inertia, higher structural stiffness, reduced sensitivity to certain errors, easy controlling and built-in redundancy but smaller and less dexterous workspace due to link interference, physical constraints of universal and spherical joints and range of motion of actuators and suffer from platform singularities. The abundant use of multi DOF spherical and universal joints in parallel manipulator not only simplify the kinematics, but they also make sure that the legs in the experience only compressive or tensile loads, but no shear forces or bending and torsion moments. This reduces the deformation of the platform, even under high loads. The fully parallel designs of robots have all actuators in or near the base, which results in a very low inertia of the part of the robot that has actually to be moved. Hence, a higher bandwidth can be achieved with the same actuation power. This is why parallel structures are used for, for example, flight simulators and fast pick-and-place robots. [65]

Parallel architecture is always lucrative for many practical applications to improve robot performance beyond the reach of serial manipulators as apparent from. Errors will not be amplified throughout a parallel structure. For a parallel kinematic mechanism, the kinematic equations will be considerably more complex due to the closed kinematic loops, than for an open (serial) kinematic structure. Parallel manipulators are also termed as closed loop manipulators. For development of high-performance robots, models will be required for simulation and performance prediction, and for model-based compensation in the control system to obtain advanced performance. Any mechanical systems composed of conventional joints, traditional parallel manipulators suffer from errors due to backlash, hysteresis, and

manufacturing errors in the joints. In contrast to serial manipulators, there can be presence of un-actuated or passive joints. The presence of passive joints and multi-DOF joints makes the kinematic analysis very different from kinematic analysis of serial manipulators Parallel robots are intrinsically more accurate than serial robots because their errors are averaged instead of added cumulatively due to many parallel links as well as closed loop architecture. These robots possess many intrinsic characteristics over serial robots. Another advantage is low cost of the parallel manipulators as the joints are much cheaper than the serial joints also easier to construct as it does need any special fabrication technologies unlike the serial robot this is indicated in the following Table **(4.1).** [65]

Table (4.1) Comparison between serial and parallel manipulators

	Parallel robot	Serial robot
Type of manipulator	Closed loop	Open loop
Location of actuators	Near the immovable base	In joint space
End effector	Platform	Gripper
Inertia and stiffness	Less and high respectively	High and less respectively
Preferred property	stiffness	Light weight
Singular points	Static	Kinematic
Preferred applications	Precise positioning	Gross motion

So, our application aims at speeding the waste sorting process so we needed fast, precise with decent payload capability so parallel manipulators were chosen. There are many parallel structures in order to minimize the cost as it is known any additional degree of freedom means additional motor with motor drive and more complexity we chose 3 DOF that will allow us to reach any point in the 3 dimensional space with specific orientations and specially delta structure because it's easy to be hanged above the conveyor belt to pick the object whenever it enters its workspace and also easy to be fabricated and do not need any complicated orientations and installations requirements also it needs rotary actuators only unlike linear delta

robot and transitional parallel robots which needs linear to rotary mechanisms or prismatic joints which are very complex to implement and some of different parallel manipulators are indicated in **Fig.4.2**. [66]

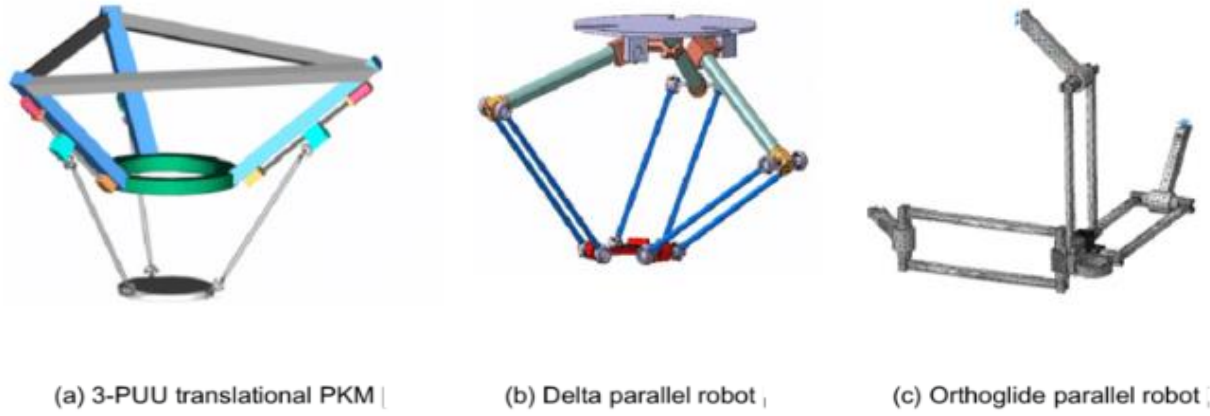


Fig.4. 2: Different types of parallel mainpulators

4.3 Workspace plannig and analysis

The workspace of a robot arm is the set of all positions that it can reach without any damage to the mechancal structure of the robot or motor blocking so workspace planning is the first step in designing any robot because it allows the user to know perivously the region at which the robot can move hence determine whether type of robot is suitable for his application or not. The motion of the robot manipulators within the working space may be restricted by several factors, such as: the constructive limits of the passive kinematical couplings, the limits given by the driving devices of the active kinematical couplings, cohesions given by the constructive elements of the robot as well as by points or areas of singularity that may divide the workspace into various component parts. An important parameter to be considered during the study of the robot workspace is the degrees of freedom that the mobile platform has.

The Delta parallel robot studied in this paper consists of a static platform, three active arms, three parallel quadrilateral driven arms, and other basic parts. Each part is connected by a rotating pair, forming three closed chains **(O-A_i-C_i-B_i-P-O)**. The active arm receives three input motors at the same time, and generates the translational motion of the end platform along **x**, **y** and **z**. The base and the active arm are connected by rotating pairs, the active arm and the driven arm are connected by spherical hinges, and the driven arm and the moving platform are also connected by spherical hinges. The structural sketch of Delta parallel robot is shown in Fig (4.3). $\mathbf{A_1A_2A_3}$ is a static platform, $B_1B_2B_3$ is a moving platform, Ai is a rotating joint, Bi and Ci are spherical joints, $\mathbf{A_iC_i}$ is an active arm and $\mathbf{C_iB_i}$ is a driven arm. Among them, **i = 1, 2, 3**. **O** is the center of the static platform and P is the center of the dynamic platform [67]. R

is the outer circle radius of the static platform and R is the outer circle radius of the moving platform. Both $\Delta A_1A_2A_3$ and $\Delta B_1B_2B_3$ are regular triangles.[67]

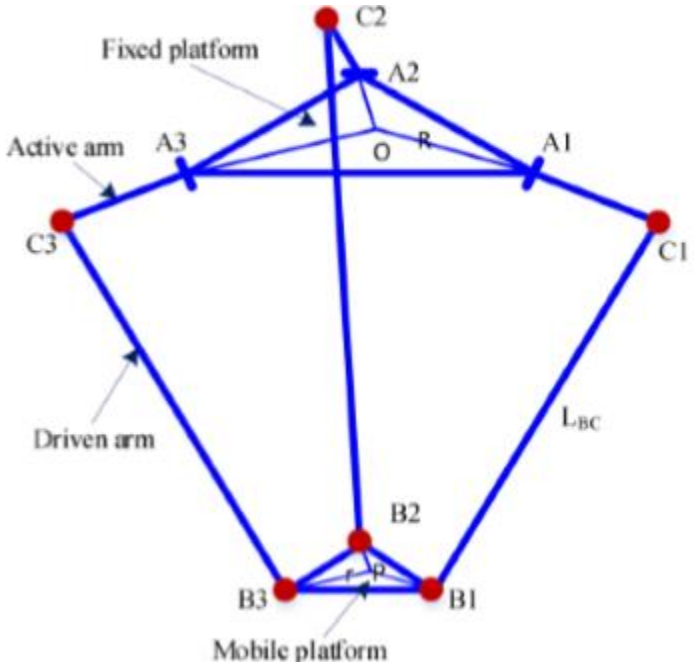


Fig.4.3: simplified Delta robot for the purpose of analysis.

We have used the approach of forward kinematics to figure out the workspace as we solved the kinematics equations for a sequence of angle pairs as we varied the angles from -30°:90° and substitute each pair in the equations and so on till drawing the workspace now the forward kinematics will be previewed.

4.3.1 Establishment of Coordinate System

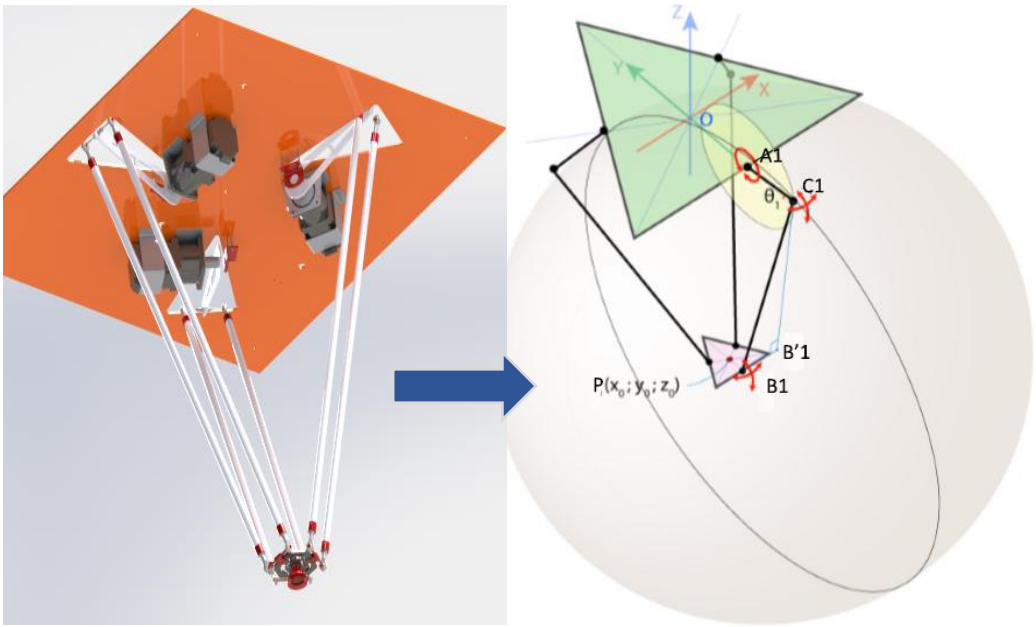


Fig.4. 4: Geometrical model of the Delta robot.

Delta robot coordinate system is established as shown in **Fig.4.4**. On the left side is the sketch of three-dimensional coordinate system, and on the right side is the analysis and modeling of a single connecting rod on the **YOZ** plane. The mechanism parameters of the robot system are as follows: the distance between the center point of the active arm and the center point of the coordinate system is **R (200mm)**, the length of the active arm is $\mathbf{L_1}$ **(300mm)**, the length of the driven arm is $\mathbf{L_2}$ **(1000mm)**, the distance between the center point of the terminal mobile platform and the middle point of the end of the driven arm is **r (50mm)**, the angle between the active arm and the **X** axis is theta1,the side length of the upper triangle is **f (692.82mm)**, the side length of the lower triangle is e(173.2mm) using the geometrical relations between the different parts the forward kinematics problem can be solved as follows

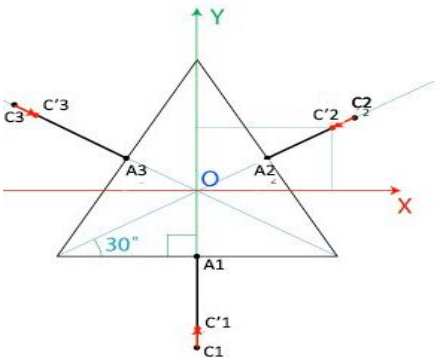


Fig.4. 5: plane view of Delta robot base

$$\mathbf{OA_1 = OA_2 = OA_3 = \frac{f}{2 * \tan(30)} = \frac{f}{2\sqrt{3}}} \quad \textbf{(4.66)}$$

$$\mathbf{C_1C'_1 = C_2C'_2 = C_3C'_3 = \frac{e}{2 * \tan(30)} = \frac{e}{2\sqrt{3}}} \quad \textbf{(4.67)}$$

$$\mathbf{A_1C_1 = Rcos(\theta_1), A_2C_2 = Rcos(\theta_2), A_3C_3 = Rcos(\theta_3)} \quad \textbf{(4.68)}$$

Where $[\boldsymbol{\theta_1, \theta_2, \theta_3}]$ are the motors angles of the three legs of the delta robot that we will input them to get all the possible points for the robot and draw the workspace.

$$\boldsymbol{C'_1 = \left(0; \frac{-(f-e)}{2\sqrt{3}} - Rcos(\theta_1); -Rsin(\theta_1)\right)} \quad \textbf{(69.4)}$$

$$\boldsymbol{C'_2 = \left(\left[\frac{(f-e)}{2\sqrt{3}} + Rcos(\theta_2)\right]\cos(30); \left[\frac{(f-e)}{2\sqrt{3}} + Rcos(\theta_2)\right]\sin(30); -Rsin(\theta_2)\right)} \quad \textbf{(4.70)}$$

$$\boldsymbol{C'_2 = \left(\left[\frac{(f-e)}{2\sqrt{3}} + Rcos(\theta_3)\right]\cos(30); \left[\frac{(f-e)}{2\sqrt{3}} + Rcos(\theta_3)\right]\sin(30); -Rsin(\theta_3)\right)} \quad \textbf{(4.71)}$$

Where the points $[\boldsymbol{C'_1}, \boldsymbol{C'_2}, \boldsymbol{C'_3}]$ are the location of the end of the bicep if the motors rotate $[\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \boldsymbol{\theta_3}]$.

4.3.2 Forward kinematics

In the following equations we will designate for the purpose of simplicity coordinates of points $\mathbf{C_1}$, $\mathbf{C_2}$, $\mathbf{C_3}$ as $\mathbf{(x_1, y_1, z_1)}$, $\mathbf{(x_2, y_2, z_2)}$ and $\mathbf{(x_3, y_3, z_3)}$. Please note that $\mathbf{x_1=0}$.

$$\begin{cases} \boldsymbol{x^2 + (y - y_1)^2 + (z - z_1)^2 = r_e^2} & \textbf{(4.72)} \\ \boldsymbol{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_e^2} & \textbf{(4.8)} \\ \boldsymbol{(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r_e^2} & \textbf{(4.9)} \end{cases}$$

These equations can be rewritten in the following form

$$\begin{cases} \boldsymbol{x^2 + y^2 + z^2 - 2y_1y - 2z_1z = r_e^2 - y_1^2 - z_1^2} & \textbf{(4.9)} \\ \boldsymbol{x^2 + y^2 + z^2 - 2x_2x - 2y_2y - 2z_2z = r_e^2 - x_2^2 - y_2^2 - z_2^2} & \textbf{(4.10)} \\ \boldsymbol{x^2 + y^2 + z^2 - 2x_3x - 2y_3y - 2z_3z = r_e^2 - x_3^2 - y_3^2 - z_3^2} & \textbf{(4.11)} \end{cases}$$

For the purpose of simplicity also we assume that

$$\boldsymbol{w_i = x_i^2 + y_i^2 + z_i^2} \quad \textbf{(4.12)}$$

$$\begin{cases} \boldsymbol{x_2x + (y_1 - y_2)y + (z_1 - z_2)z = (w_1 - w_2)/2} & \textbf{(4.13)} \\ \boldsymbol{x_3x + (y_1 - y_3)y + (z_1 - z_3)z = (w_1 - w_3)/2} & \textbf{(4.14)} \\ \boldsymbol{(x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = (w_2 - w_3)/2} & \textbf{(4.15)} \end{cases}$$

Subtracting (4.13) -(4.14), (4.13) -(4.15), and (4.14) -(4.15) receptively

$$\begin{cases} \boldsymbol{x_2x + (y_1 - y_2)y + (z_1 - z_2)z = (w_1 - w_2)/2} & \textbf{(4.16)} \\ \boldsymbol{x_3x + (y_1 - y_3)y + (z_1 - z_3)z = (w_1 - w_3)/2} & \textbf{(4.17)} \\ \boldsymbol{(x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = (w_2 - w_3)/2} & \textbf{(4.18)} \end{cases}$$

From **(4.16)**, **(4.17)**

$$\mathbf{x = a_1 * z + b_1} \quad \textbf{(4.19)}$$

$$\mathbf{y = a_2 * z - b_2} \quad \textbf{(4.20)}$$

$$\mathbf{a_1 = \frac{1}{d}[(z_2 - z_1) * (y_3 - y_1) - (z_3 - z_1) * (y_2 - y_1)]} \quad \textbf{(4.21)}$$

$$\mathbf{a_2 = -\frac{1}{d}[(z_2 - z_1) * x_3 - (z_3 - z_1) * x_2]} \quad \textbf{(4.22)}$$

$$\boldsymbol{b_1 = -\frac{1}{2d}[(w_2 - w_1) * (y_3 - y_1) - (w_3 - w_1) * (y_2 - y_1)]} \quad \textbf{(4.23)}$$

$$\boldsymbol{b_2 = \frac{1}{2d}[(w_2 - w_1) * x_3 - (w_3 - w_1) * x_2]} \quad \textbf{(4.24)}$$

$$\boldsymbol{d = [(y_2 - y_1) * x_3 - (y_3 - y_1) * x_2]} \quad \textbf{(4.25)}$$

Now we can substitute **(4.19)** and **(4.20)** in **(4.13)**

$$(a_1^2 + a_2^2 + 1)z^2 + 2(a_1 + a_2(b_2 - y_1) - z_1)z + (b_1^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2) = 0 \quad \textbf{(4.26)}$$

Then we use the general formula for second order equation to obtain the value of z we refuse any positive values because the robot moves only in the negative Z direction, after that we substitute z in **(4.19)**, **(4.20)** to get the point and repeat for all the generated pairs of angles. [68] We implemented this algorithm in **MATLAB™** code to get all the possible points and draw the workspace of the delta robot. We created **MATLAB™** function for the forward kinematics which is recalled for each set of angles and we obtain the corresponding set of points. [69]

4.3.3 Forward kinematics MATLAB™ code

```
function Output = FK(u)
% Defining delta robot dimensions
e  =  173.2051;
f  =  692.8203;
l2 =  1000;
l1 =  300 ;
% Some constants like installation angles trig functions for simplicity
sqrt3 = sqrt(3.0);
pi     = 3.141592653;
sin120 = sqrt3 / 2.0;
cos120 = -0.5;
tan60  = sqrt3;
sin30  = 0.5;
tan30  = 1.0 / sqrt3;
%.................. Forward Kinematics.........................%
theta1 =u(1);
theta2 =u(2);
theta3 =u(3);
t = (f-e) * tan30 / 2.0;
dtr = pi / 180.0; % degree to radian conversion.
theta1 =theta1.*dtr;
theta2 =theta2.*dtr;
theta3 =theta3.*dtr;

y1 = -(t + l1.*cos(theta1));
z1 = -l1 .* sin(theta1);

y2 = (t + l1.*cos(theta2)) .* sin30;
x2 = y2 .* tan60;
z2 = -l1 .* sin(theta2);

y3 = (t + l1.*cos(theta3)) .* sin30;
x3 = -y3 .* tan60;
z3 = -l1 .* sin(theta3);9

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 = 2.0 .* (a1.*b1 + a2.*(b2 - y1.*dnm) - z1.*dnm.*dnm);
c = (b2 - y1.*dnm).*(b2 - y1.*dnm) + b1.*b1 + dnm.*dnm.*(z1.*z1 - l2.*l2);

% discriminant
d = b.*b - 4.0.*a.*c;
    if d < 0.0
        Output= [0,0,0]; % non-existing povar.
    else
        z0 = -0.5.*(b + sqrt(d)) ./ a;
        x0 = (a1.*z0 + b1) ./ dnm;
        y0 = (a2.*z0 + b2) ./ dnm;

        Output =[x0,y0,z0];
    end
end
```

Then this function is used in a loop to obtain the total points

```
theta_1 = linspace(-30,90,50);
theta_2 = linspace(-30,90,50);
theta_3 = linspace(-30,90,50);

cell = {theta_1, theta_2, theta_3};
angles_cell = cell;
[angles_cell{:}] = ndgrid(cell{:});
combinations = cell2mat(cellfun(@(m)m(:), angles_cell,'uni',0));
final_points=[];

for i =1:125000
     e=[combinations(i,1),combinations(i,2),combinations(i,3)];
     FK_Solution=FK(e);
     final_points=[final_points;FK_Solution];
end
```

After we obtained the points pairs, we plot the three projections using the following piece of code which is used to demonstrate the 3D shape of the workspace and its three projections.

```
% Plot 3D point cloud
figure(1)
pcshow(final_points)
title('Delta Robot Workspace')
xlabel('X (mm)')
ylabel('Y (mm)')
zlabel('Z (mm)')
%% X_Y projection
figure(2)
pcshow(final_points)
view(2)
title('Projection on X-Y plane')
xlabel('X (mm)')
ylabel('Y (mm)')
```

```
%% Y-Z projection
figure(3)
pcshow(final_points)
view([1 0 0])
title('Projection on Y-Z plane')
ylabel('Y (mm)')
zlabel('Z (mm)')
%% X_Z projection
figure(4)
pcshow(final_points)
view([0 1 0])
title('Projection on X-Z plane')
xlabel('X (mm)')
zlabel('Z (mm)')
```

After we applied this piece of code the result shape of workspace is as shown in **Fig.4.6**.

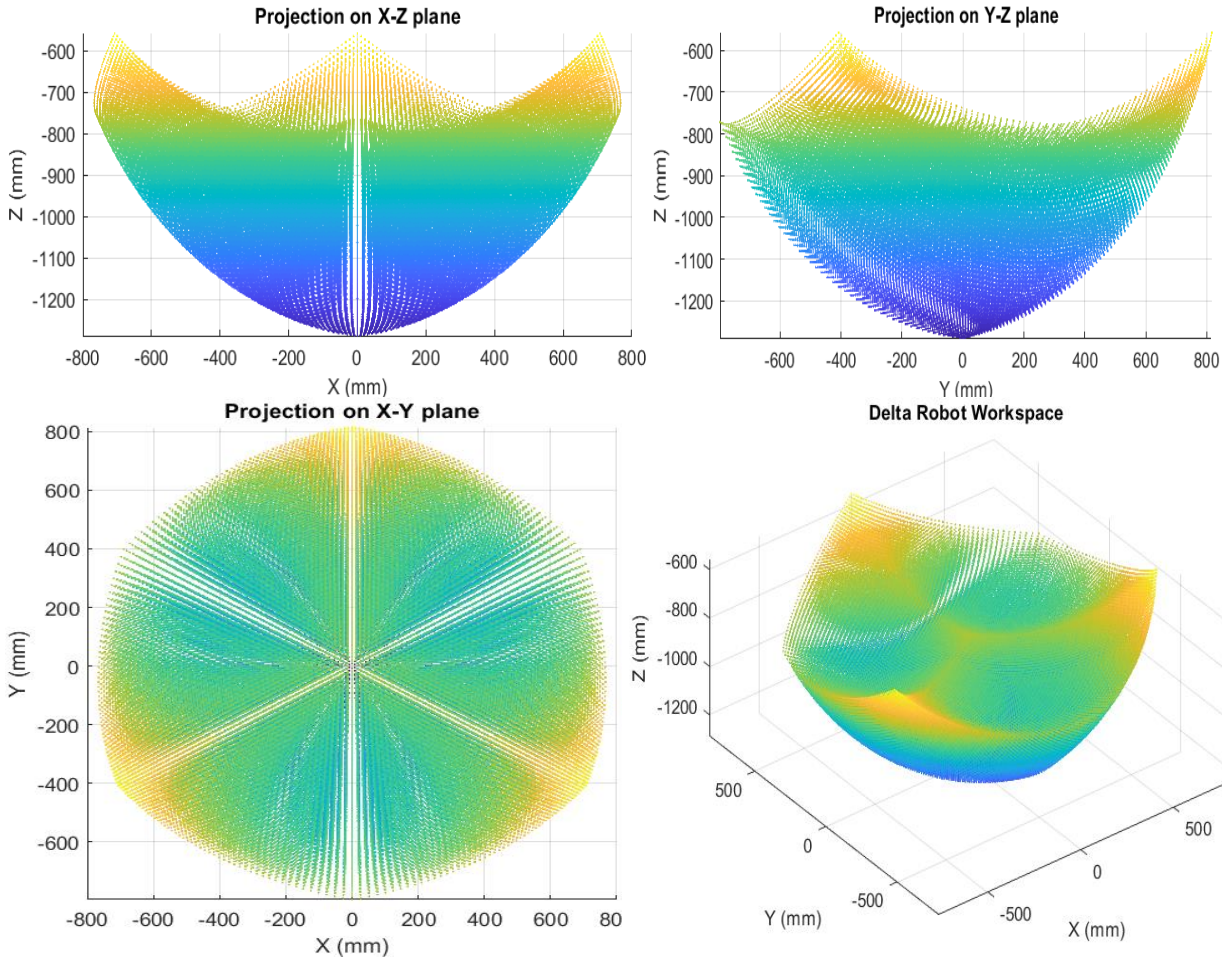


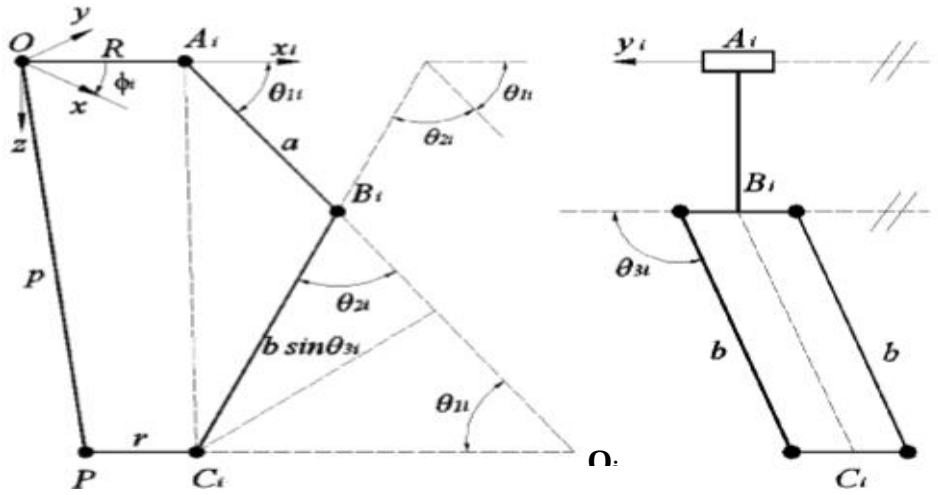
Fig.4.6: Theoretical delta robot workspace with its three projections

4.3.4 Workspace constrains

4.3.4.1 Ball joint constraints

But we have to put some constraints on this work space as the obtained one was based on some assumptions that will not be present when the robot operates such that the ball joint is assumed to have **60°** of freedom along its y axis and during the design it was obvious that the largest angle obtained was **28°** so the obtained wok space went through additional modifications so that we get the physical workspace which is collision free and eliminating the singular points which may cause mechanical damages.

We have proposed an approach to compute the workspace of the "Delta" parallel robot which based on forward kinematic model and for avoiding singular configurations we limited the angles $\boldsymbol{\Theta_{1i}, \Theta_{2i}, \Theta_{3i}}$ which are indicated in Fig **(4.7).** [70], [71]



(a) (b)

Fig.4.7: (a) The x-z projection of the Delta robot. (b) The end-on view

Now we aim to calculate the angles using the geometrical model of the robot

$$\boldsymbol{\Theta_{3i} = \cos^{-1}\left(\frac{x\cos(\varphi_i) + y\sin(\varphi_i)}{L_2}\right)} \qquad \textbf{(4.27)}$$

Where $\boldsymbol{\varphi_i}$ is the installation angle of the three motors = [**0,120,240**]

$\boldsymbol{\Theta_{3i}}$ represents the angle of freedom of the ball joint which is 30°:150° but during the design process we tried to implement more than a physical design to maximize the value of this angle but the maximum value we got was **62°:118°** as indicated below using **SOILDWORKS™** software in Fig **(4.8)** which contains a precious drawing of ball joint to imitate the real operation. From the triangle $\mathbf{C_iB_iQ_i}$ which contains both $\boldsymbol{\Theta_{1i}}$ and $\boldsymbol{\Theta_{2i}}$ then

$$0° < \Theta_{1i} + \Theta_{2i} < 180° \tag{4.28}$$

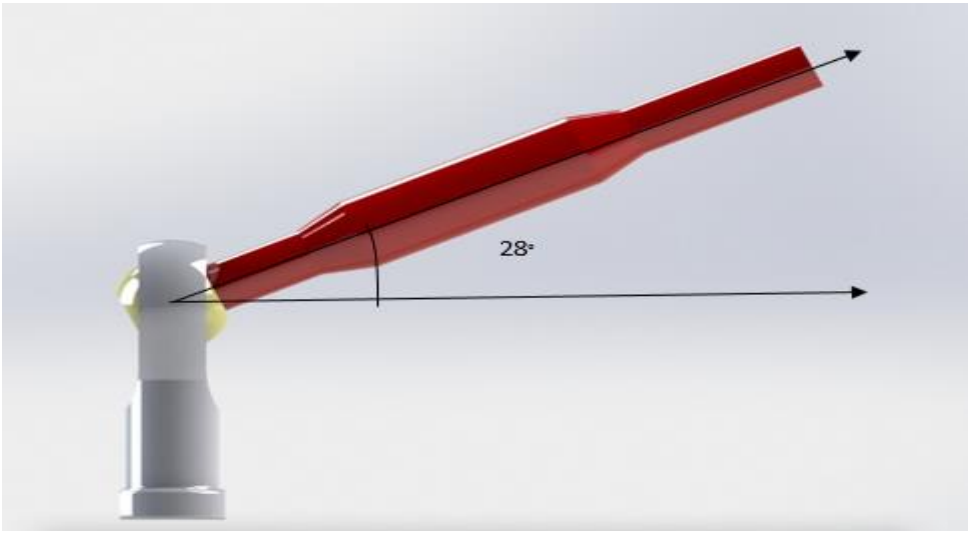


Fig.4.8: shows the maximum angle before a collision occurs between link and ball joint body

Knowing that the values of Θ_{1i} are already known as they were the input of the forward kinematics solution so we need only to calculate Θ_{2i}

$$\Theta_{2i} = \cos^{-1}\left(\frac{z^2 + (x\cos(\Theta_{1i}) + y\sin(\Theta_{1i}) + (R-r))^2 + (x\cos(\varphi_i) + y\sin(\varphi_i))^2 - {L_1}^2 - {L_2}^2)}{2L_1L_2\sin(\Theta_{3i})}\right) \tag{4.29}$$

4.3.4.2 Ball joint constraints MATLAB™ code

Then we applied these constrains on the obtained workspace using the following code

```
%%% Calculating passive joint angles
% Defining delta robot parameters in mm.
b  =  50;   % radius of moving platform
a  =  200;  % radius of fixed platform
L1 =  300 ; % bicep (L1)
L2 =  1000; % forearm (L2)
phai_i   = [0, 2*pi/3, -2*pi/3];
x        =final_points(:,1);
y        =final_points(:,2);
z        =final_points(:,3);
% Variables used to get theta_3i
b_2i     = zeros(3,1);
theta_3i = zeros(3,1);
% Variables used to get theta_2i
b_1i     = zeros(3,1);
k        = zeros(3,1);
theta_2i = zeros(3,1);
% Variables used to store test_stuff and angle1 could be directly taken
% from combinations which were genrated in the previous code
angle2 = [];
angle3 = [];
```

```
for j = 1:1:length(x)
    for i = 1:1:3
        b_2i(i)     = -1*sin(phai_i(i)) *x(j) + cos(phai_i(i)) *y(j);
        theta_3i(i) =  acos(b_2i(i)/L2);
        b_3i        =  z(j);
        b_1i(i)     = cos(phai_i(i)) *x(j) + sin(phai_i(i)) *y(j) + (b-a);
        k(i)        = (b_1i(i)^2 + b_2i(i)^2 + b_3i^2 - L1^2 - L2^2) / (
2*L1*L2* sin(theta_3i(i)) );
        theta_2i(i) = acos(k(i));
    end
    angle3 = [angle3,theta_3i];
    angle2 = [angle2,theta_2i];
end
%%%%%%% we found that all the values of theta2 verify the constrains so we
%%%%%%% care only about theta3
% we get the min & max of each pair of the angles so that we compare it
% with the physical constrains
xl=[];
for ii=1:length(angle3)
    maxx = max (angle3(:,ii))*180/pi;
    minn = min (angle3(:,ii))*180/pi;
    if minn >62 && maxx<118
      xl=[xl,ii];
    end
end
x=x(xl);
y=y(xl);
z=z(xl);
final_points=final_points(xl);
```

Then we plot the obtained points with its three projections as shown in **Fig.4.9**

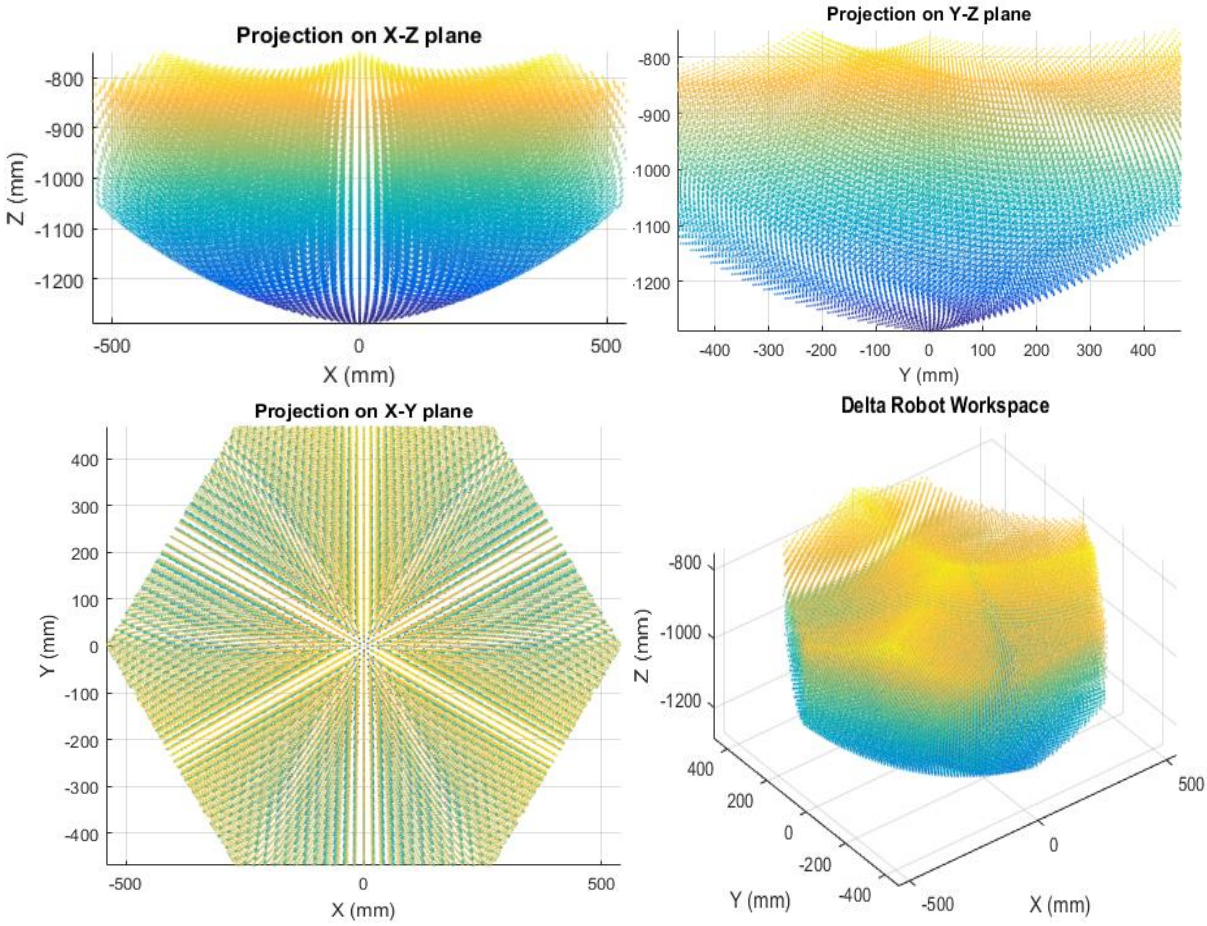


Fig.4.9: Constrained workspace with its three projections

4.3.4.3 Z - Direction Constraints

This was the first type of constrains with limit the delta robot workspace in both x and y directions the second type is to limit the movement of the robot in the direction and its optional as in our application the highest object to be collected is about 150 mm tall so we only concern about 150mm in the workspace z direction the rest will not be reached as it will be under the conveyor belt which is not permissible for sure in the following section we will indicate how we will extract the desires portion of the work space which maximizes the reached points and still 150 mm in z direction. We divided the workspace into slices of 1 mm thickness then we investigate the number of feasible points that the robot can reach in each successive 150 slices then we take the region with largest reachable workspace.

4.3.4.4 Z-Direction Constraints MATLAB™ code

We used the following piece of code to get the desired sets of points

```
xX=final_points(:,1);
xY=final_points(:,2);
xZ=final_points(:,3);
z_min=min(xZ);
z_max=max(xZ);
%we divided the workspace in z direction into slices of 1 mm thickness then
%we are to investigate which 150 slice gives us the largest possible reach
index=round(abs(z_max-z_min));
Lengthx=zeros(index,1);
for i=0:1:index
    i
    Num_points=[];
    z_interval=z_max-150-1;

    z_max=z_max-1;

    for j =1:length(xZ)
        if xZ(j)> z_interval && xZ(j)< z_max
           Num_points=[Num_points,xZ(j)];
        end
    end
    Lengthx(i+1)=length(Num_points);
end
Zmax_final=max(xZ)- find(Lengthx==max(Lengthx));
Zmin_final=max(xZ)- find(Lengthx==max(Lengthx))-150;
X_finalx=[];
for k =1:length(xZ)
    if (xZ(k)<Zmax_final && xZ(k)>Zmin_final )
        X_finalx=[X_finalx,k];
    end
end
xX=xX(X_finalx);
xY=xY(X_finalx);
xZ=xZ(X_finalx);
final points=[xX,xY,xZ];
```

After we got the final points, we use the plotting piece of code to visualize the actual works space as demonstrated in **Fig.4.10.**

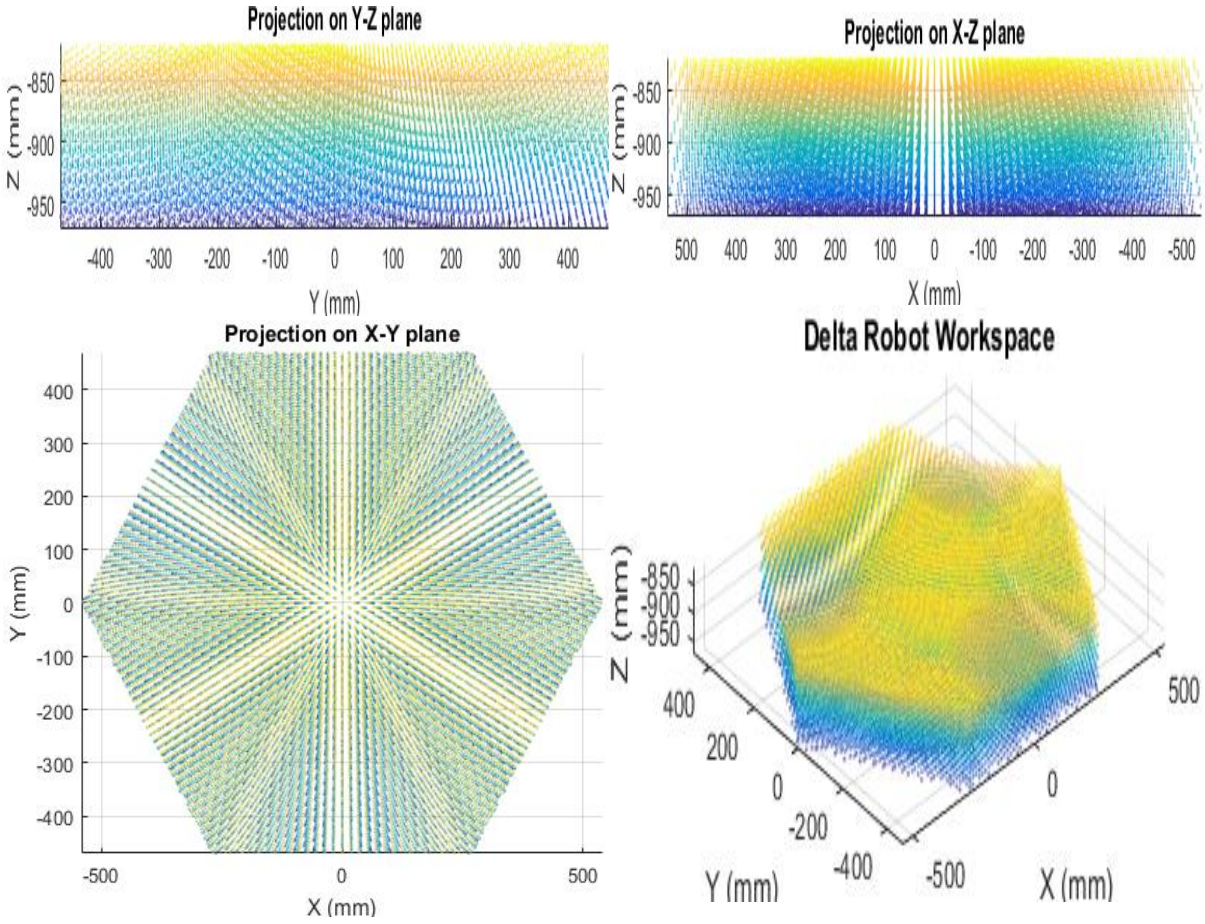


Fig.4.10: Actual constrained workspace with its three projections

As seen from the projections especially X-Y Plane as shown in **Fig.4.11** the area between the two red lines is the reachable area on the conveyor belt which the arm can collect objects from it while the two black triangles represent the areas where the collecting pans would be located so that the robot sorts the objects and throw them into the different pans based on their predefined types.

To conclude from workspace analysis it is clear that the dimensions that are assumed would be suffiecint for our application and would give us satisfying reach to collect as many objects as possible as well as from Z – Direction constraint we were able to identify on which height the motors should be mounted which we will discuss indetail in the following sections.

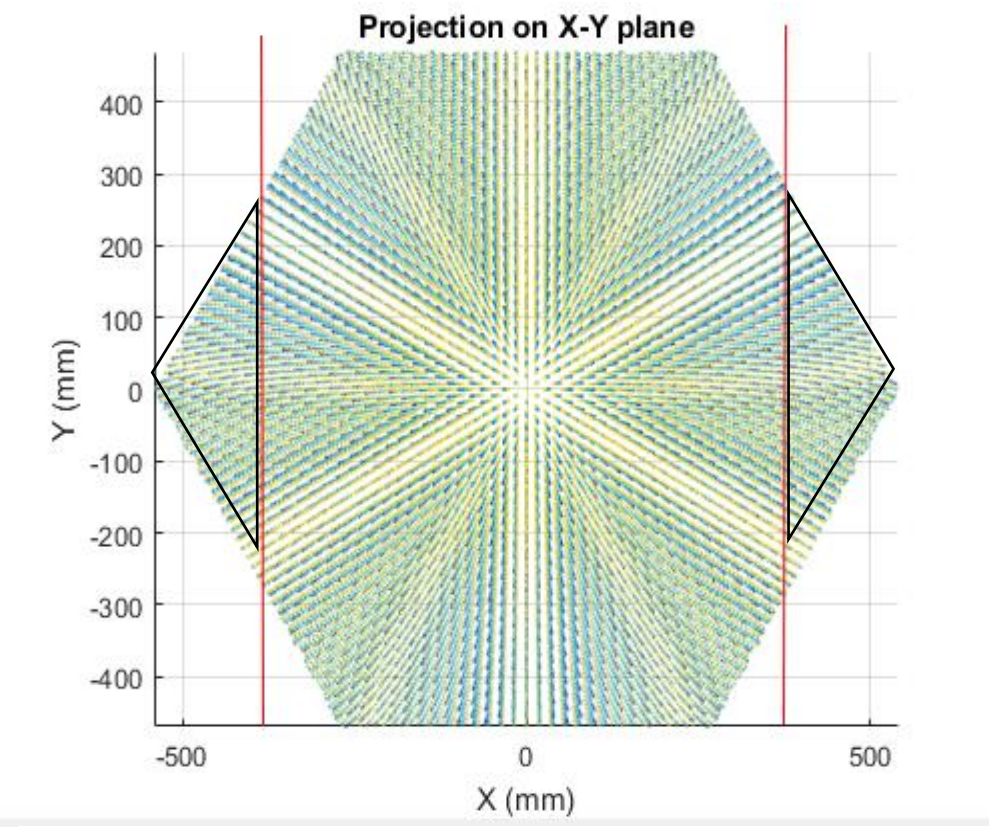


Fig.4.11: Actual work divisions into conveyor belt and collection pans

4.4 Dynamic Model

Accuracy and fast calculation of dynamics are very essential typically in computed torque control of a Delta-type parallel manipulator for high speed applications. In this paper, a simplified dynamic modelling of a Delta-type parallel manipulator is presented. First, the position, velocity, and acceleration analyses are performed. Then, the dynamics modelling is derived by using the Lagrangian equation of the first type due to the complex kinematics.

4.4.1 Kinematic analysis

As shown in **Fig.4.12**, the delta parallel robot consists of three R-Pa (Revolute-Spatial parallelogram) legs connecting in parallel from the base platform to the moving platform, which allows three translational motions to the moving platforms. Since two S-S (Spherical-Spherical) chains consisting of each Pa chain are subject to only axial load, Delta parallel robot has small moving inertia. For modeling the delta robot mathematically, we need to convert the 3D view of the delta robot into vector representation. The dynamic model could be used in torque control, motor selection, and inverse kinematics control. [72], [73]

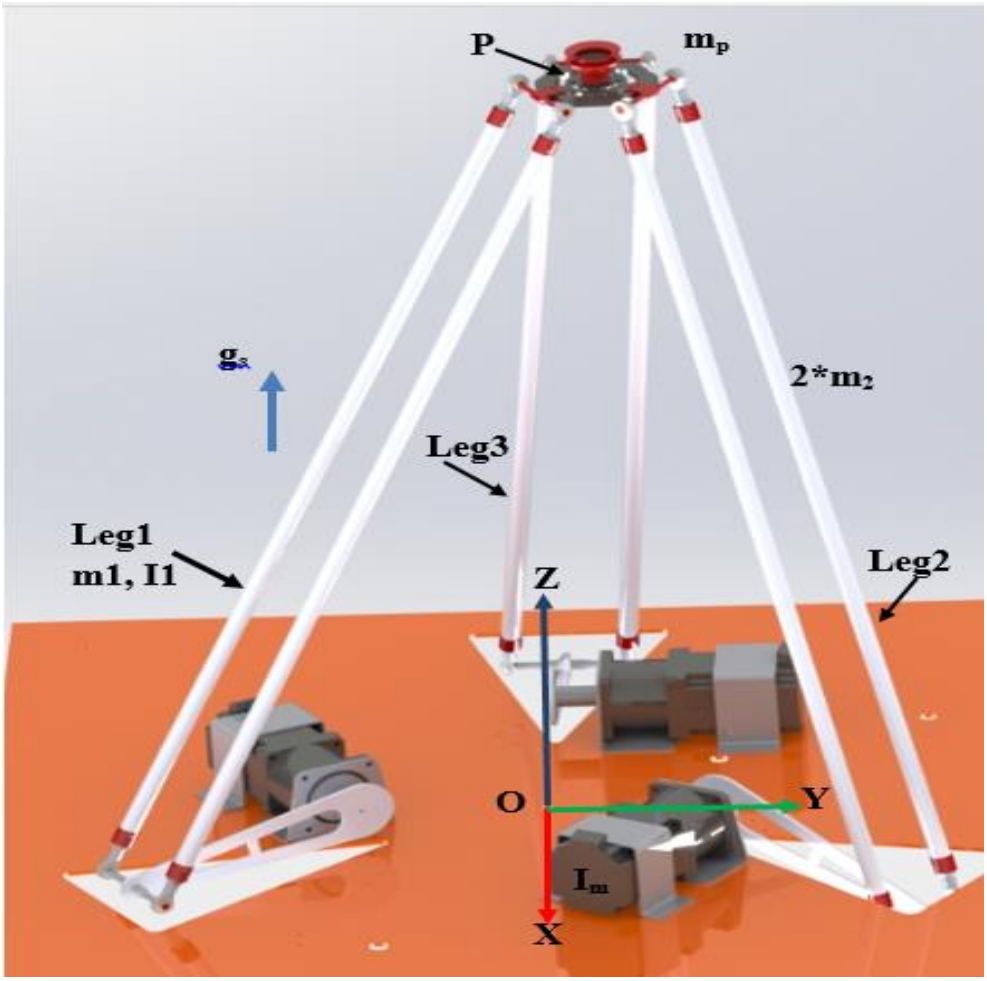


Fig.4.12: Configuration of delta robot

Fig.4.12 shows the vector-loop diagram with the kinematic parameters; ***R*** and ***r*** denote the radii of the fixed and moving platforms, and ***a*** and ***b*** are the actuating and connecting link

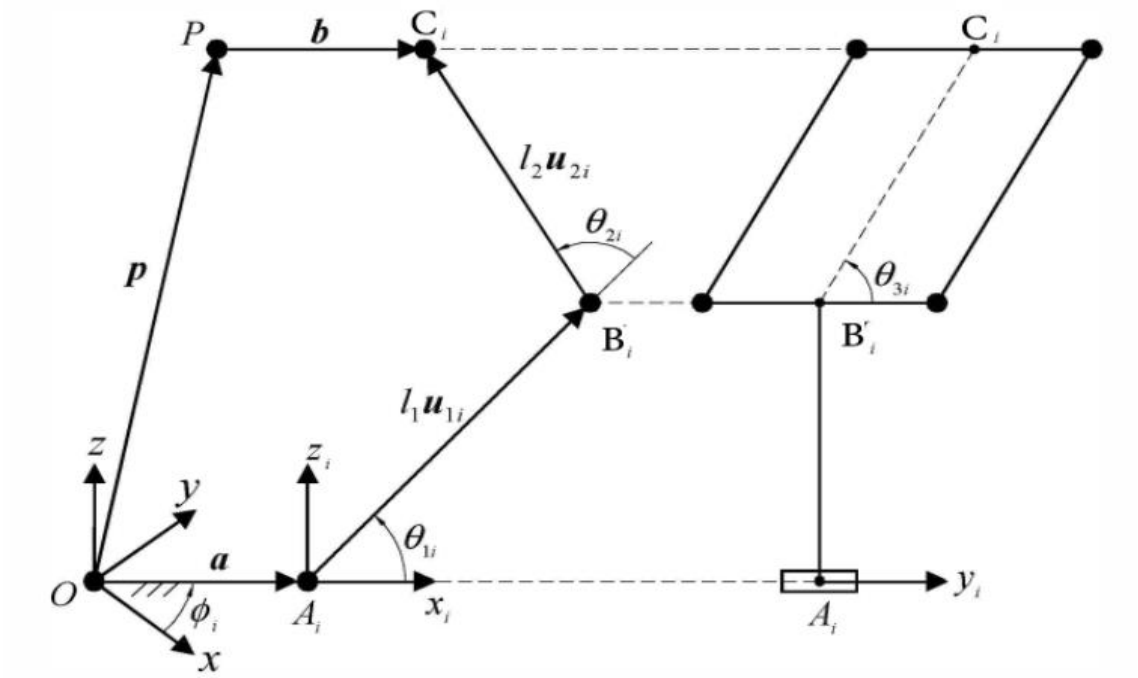


Fig.4.13: Vector representation of delta robot

lengths. The actuated joint angles are denoted by $\boldsymbol{\theta_{1i}}$ and passive joint angles are denoted by $\boldsymbol{\theta_{2i}}$ and $\boldsymbol{\theta_{3i}}$ **.** The vector-loop equation of each leg is given by

$$\overrightarrow{\mathbf{A_iB_i}} + \overrightarrow{\mathbf{B_iC_i}} = \overrightarrow{\mathbf{OP}} + \overrightarrow{\mathbf{OC_i}} - \overrightarrow{\mathbf{OA_i}} \tag{4.30}$$

For simplicity of expression, the local frame for the $\boldsymbol{i^{th}}$ leg $\boldsymbol{(A_i - x_i y_i z_i)}$ is introduced and the superscript denotes the frame in which a vector is expressed. The vector-loop equation is expressed in the $\boldsymbol{i^{th}}$ local frame by

$$\mathbf{a^i u_{1i}} + \mathbf{b^i u_{2i}} = (\mathbf{P} + \mathbf{R} - \mathbf{r})^{\mathbf{i}} \tag{4.31}$$

Where,

$$\boldsymbol{u_{1i}^i} = \begin{bmatrix} \boldsymbol{\cos(\theta_{1i})} \\ \mathbf{0} \\ \boldsymbol{\sin(\theta_{1i})} \end{bmatrix}, \quad \boldsymbol{u_{2i}^i} = \begin{bmatrix} \boldsymbol{\sin(\theta_{3i})\cos(\theta_{1i}+\theta_{2i})} \\ \mathbf{0} \\ \boldsymbol{\sin(\theta_{1i})} \end{bmatrix} \tag{4.32}$$

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p1} \\ \boldsymbol{p2} \\ \boldsymbol{p3} \end{bmatrix}, \qquad \boldsymbol{R^i} = \begin{bmatrix} \boldsymbol{R} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \qquad \boldsymbol{r^i} = \begin{bmatrix} \boldsymbol{r} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{4.33}$$

$$\mathbf{P^i} = (\mathbf{R_x})^{\mathbf{T}}\mathbf{P} \tag{4.34}$$

$$\mathbf{Rx} = \begin{bmatrix} \boldsymbol{\cos(\Phi_i)} & \boldsymbol{-\sin(\Phi_i)} & \mathbf{0} \\ \boldsymbol{\sin(\Phi_i)} & \boldsymbol{\cos(\Phi_i)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}, \tag{4.35}$$

Where $\boldsymbol{\Phi_1 = 0}$, $\boldsymbol{\Phi_2 = +2\pi/3}$, $\boldsymbol{\Phi_1 = -2\pi/3}$ are defined as shown in **Fig.4.14**,

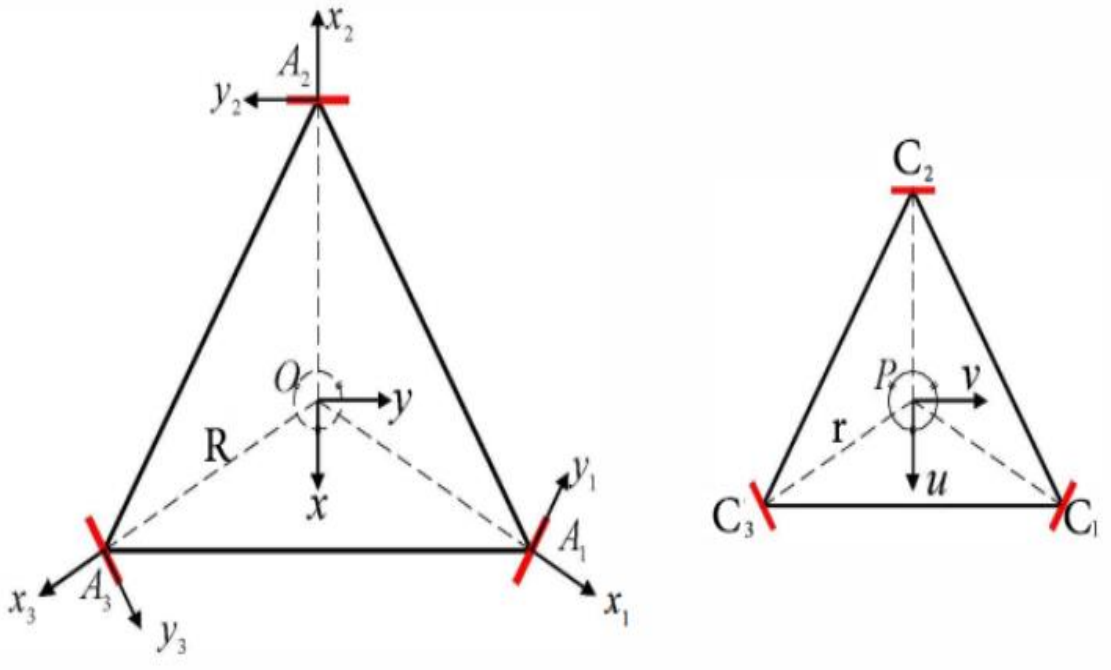


Fig.4.14: (a) Fixed frame and ith local frame, (b) Moving frame

Solving Eq. **(4.31)** for given **P,** two passive joint angles($\boldsymbol{\theta_{2i}}$,$\boldsymbol{\theta_{3i}}$) and one joint angle($\boldsymbol{\theta_{1i}}$) of the $\boldsymbol{i^{th}}$ leg are obtained by

$$\boldsymbol{\theta_{3i} = cos^{-1}(\frac{R_{2i}}{b})} \quad \textbf{(4.36)}$$

$$\boldsymbol{\theta_{2i} = cos^{-1}(K)} \quad \textbf{(4.37)}$$

Where,

$$\boldsymbol{K = \frac{R_{1i}^2 + R_{2i}^2 + R_{3i}^2 - a_1^2 - b_2^2}{2 * a * b * \sin(\theta_{3i})}} \quad \textbf{(4.38)}$$

$$\boldsymbol{R_{3i} = p_3} \quad \textbf{(4.39)}$$

$$\boldsymbol{b_{1i} = \cos(\Phi_i)\, p_1 + \sin(\Phi_i)\; p_2 + (b - a)} \quad \textbf{(4.40)}$$

$$\boldsymbol{b_{2i} = -\sin(\Phi_i)\, p_1 + \cos(\Phi_i) p_2} \quad \textbf{(4.41)}$$

The linear velocity of the moving platform is obtained by taking derivatives of Eqn **(4.31)** with respect to time (in the local frame of the $\boldsymbol{i^{th}}$ leg)

$$\boldsymbol{P^o = a * (w_{1i} * u_{1i}) + b * (w_{2i} * u_{2i}) + b * (w_{3i} * u_{2i})} \quad \textbf{(4.42)}$$

Where,

$$\boldsymbol{w_{1i} = \theta_{1i}^o [0 \quad -1 \quad 0]^T} \quad \textbf{(4.43)}$$

$$\boldsymbol{w_{2i} = (\theta_{1i}^o + \theta_{2i}^o)[0 \quad -1 \quad 0]^T} \quad \textbf{(4.44)}$$

$$\boldsymbol{w_{3i} = \theta_{3i}^o [\sin(\theta_{1i} + \theta_{2i}) \quad 0 \quad -\cos(\theta_{1i} + \theta_{2i})]^T} \quad \textbf{(4.45)}$$

In order to eliminate passive joint rates w_{2i} and w_{3i} , taking dot-product at the both sides of Eqn **(4.31)** yields,

$$\boldsymbol{u_{2i} * (R_x)^T * P^o = a * w_{1i} * (u_{1i} * u_{2i})} \quad \textbf{(4.46)}$$

Writing Eqn **(4.46)** for i = 1, 2, 3 gives the velocity relation by

$$\boldsymbol{P^o = J * \theta^o} \quad \textbf{(4.47)}$$

Where,

$$\boldsymbol{\theta^o = [\theta_1^o \quad \theta_2^o \quad \theta_3^o]^T \equiv [\,\theta_{11}^o \quad \theta_{12}^o \quad \theta_{13}^o]^{\mathrm{T}}} \quad \textbf{(4.48)}$$

$$\boldsymbol{J = J_q^{-1} * J_x} \quad \textbf{(4.49)}$$

$$J_q = \begin{bmatrix} \boldsymbol{Sin(\theta_{21})Sin(\theta_{31})} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Sin(\theta_{21})Sin(\theta_{31})} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{Sin(\theta_{21})Sin(\theta_{31})} \end{bmatrix} \quad \textbf{(4.50)}$$

$$J_x = \begin{bmatrix} \boldsymbol{J_{1X}} & \boldsymbol{J_{1Y}} & \boldsymbol{J_{1Z}} \\ \boldsymbol{J_{2X}} & \boldsymbol{J_{2Y}} & \boldsymbol{J_{2Z}} \\ \boldsymbol{J_{3X}} & \boldsymbol{J_{3Y}} & \boldsymbol{J_{3Z}} \end{bmatrix} \quad \textbf{(4.51)}$$

$$\boldsymbol{J_{ix} = Cos(\theta_{1i} + \theta_{2i})Sin(\theta_{3i})Cos(\emptyset_i) - Cos(\theta_{3i})Sin(\emptyset_i)} \quad \textbf{(4.52)}$$

$$\boldsymbol{J_{iy} = Cos(\theta_{1i} + \theta_{2i})Sin(\theta_{3i})Sin(\emptyset_i) + Cos(\theta_{3i})Cos(\emptyset_i)} \quad \textbf{(4.53)}$$

$$\boldsymbol{J_{iz} = Sin(\theta_{1i} + \theta_{2i})Sin(\theta_{31})} \quad \textbf{(4.54)}$$

Once $\boldsymbol{P^o}$ is solved, the two passive joint rates are obtained from Eqn **(4.42)**

$$\boldsymbol{\theta_{3i}^o = \frac{P_2^o}{b * Sin(\theta_{3i})}} \quad \textbf{(4.55)}$$

$$\boldsymbol{\theta_{2i}^o = \frac{-i_{P_1^o} Cos(\theta_{1i}) - i_{P_1^o} Sin(\theta_{1i}) + b * Cos(\theta_{2i})Cos(\theta_{3i}\theta_{3i}^o)}{b * Sin(\theta_{2i})Sin(\theta_{3i}) - \theta_i}} \quad \textbf{(4.56)}$$

The acceleration relation can be derived by taking derivatives of Eq.(9) with respect to time

$$\boldsymbol{J_x^o * P^o + J_x * P^{oo} = J_q^o * \theta^o + J_q * \theta^{oo}} \quad \textbf{(4.57)}$$

The derivation of the time derivatives of J_x and J_q is omitted.

In Eqn **(4.57)**, we have the values of $\boldsymbol{\theta^o}$ and $\boldsymbol{\theta^{oo}}$ from the speed and acceleration profiles of the bipolar stepper motors. We also have $\boldsymbol{P^o}$ from solving Eqn **(4.47)**, Hence we solve Eqn **(4.57)** to get $\boldsymbol{P^{oo}}$.

4.4.2 Kinetic analysis

One important step in design of process of a robot is to understand the behavior of device as it moves around its workspace or doing a specific task. This behavior is determined through of the dynamics of the mechanism, where the force acting on the elements and torques required by the material to be used in the manufacturing process. In this section, the dynamics of Delta-type parallel robot is described based on Lagrangian formulation, which is based on calculus variations, states that a dynamic system can be expressed in terms of its kinetic and potential energy leading in an easy way the solution to the problem. In addition, it is considered a good opinion to be used for real-time control for parallel manipulators. [74], [75]

The Lagrangian equations can be derived by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial q_j}\right) - \frac{\partial L}{\partial q_j} = Q_j + \sum_{i=1}^{k} \lambda_i + \frac{\partial \Gamma_i}{\partial q_j} \qquad \textbf{For j = 1, 2, 3} \tag{4.58}$$

Where L is the Lagrangian function $= K - U$,

K: Total kinetic energy of the body, **U**: Total potential energy of the body,

q: $\mathbf{k^{th}}$ generalized coordinate, **Q**: Generalized external force

λ_i: Lagrangian multiplier, Γ_i: Constraint equation.

By employing the formula above, it is possible to determine external forces of a body. However, friction forces are not constraints even though they play an important role in dynamics analysis so they can be treated separately.

The Lagrangian multipliers are derived as:

$$2\sum_{i=1}^{3} \lambda_i \left(P_1 + r\cos(\phi_i) - R\cos(\phi_i) - a\cos(\phi_i)\cos(\theta_i)\right) = \left(m_p + 3m_2\right)\ddot{P}_1 \tag{4.59}$$

$$2\sum_{i=1}^{3} \lambda_i + \left(P_2 + r\sin(\phi_i) - R\sin(\phi_i) - a\sin(\phi_i)\sin(\theta_i)\right) = \left(m_p + 3m_2\right)\ddot{P}_2 \tag{4.60}$$

$$2\sum_{i=1}^{3} \lambda_i \left(P_3 - a * \sin(\theta_i)\right) = \left(m_p + 3m_b\right)\left(\ddot{P}_3 + g_c\right) \tag{4.61}$$

When the Lagrangian multipliers are found we can find the actuator torques as follows

$$T_1 = \left(\frac{1}{3}m_a + m_b\right) * L_1^2\theta_{11} + \left(\frac{1}{2}m_a + m_b\right) * 9.81L_1\cos(\theta_{11}) - 2L_1\lambda_1[(p_1\cos(\Phi_1) + p_2\sin(\Phi_1) + b - a)\sin(\theta_{11}) - p_3\cos(\theta_{11})] \tag{4.62}$$

$$T_2 = \left(\frac{1}{3}m_a + m_b\right) * L_1^2\theta_{12} + \left(\frac{1}{2}m_a + m_b\right) * 9.8L_1\cos(\theta_{12}) - 2L_1\lambda_2[(p_1\cos(\Phi_2) + p_2\sin(\Phi_2) + b - a)\sin(\theta_{12}) - p_3\cos(\theta_{12})] \tag{4.63}$$

$$T_3 = \left(\frac{1}{3}m_a + m_b\right) * L_1^2\theta_{13} + \left(\frac{1}{2}m_a + m_b\right) * 9.81L_1\cos(\theta_{13}) - 2L_1\lambda_3[(p_1\cos(\Phi_3) + p_2\sin(\Phi_3) + b - a)\sin(\theta_{13}) - p_3\cos(\theta_{13})] \tag{4.64}$$

4.4.3 Actual Parameters

To valid the analysis of dynamics, an experimental setup was built to perform the control of Delta-type parallel robot. Some specifications of delta parallel robot are shown in Table **(4.2)**. This experimental implementation is built on **MATLAB™** using the kinematics and dynamics analyses from above solutions to control the platform. The program is used to control the moving platform with predefined trajectory. We will apply the kinematics, Jacobi, and dynamics to control suitable trajectory of parallel robot based on positions and velocities.

Table (4.2) Specifications of Delta parallel Robot

Parameters	**Value**
Upper robot arm (bicep) [Kg]	**0.180**
Parallelogram (Forearm) [Kg]	**0.2334**
Moving platform (End effector) [Kg]	**0.01**
Radius of fixed platform (R) [mm]	**200**
Radius of moving platform (r) [mm]	**50**
Upper robot arm length (a) [mm]	**300**
Parallelogram length (b) [mm]	**1000**

4.4.4 Dynamic model MATLAB™ code

```
%% Calculating the angular speed and acceleration.
j = 10;
load('final_points.mat')
x1 = final_points(:,1);
y1 = final_points(:,2);
z1 = final_points(:,3);

load('angles_combinations.mat')
combinations = combinations *pi/180;
% both of angles combinations and final points are obtained from workspace
% analysis

%we use these to test points to test the invrse kinematics
p_current = [-51.3222238758787,401.976899058375,-782.201921296407];
p_desired = [-38.0861508684435,499.485223570934,-751.581036157831];
%theta_1i = combinations(j,:);

t = 0.8; % time in seconds
% this the average assumed time that the robot needs to pick the object
% then place it and it is predicted to be much faster
```

```
% Appling Inverse kinematics on both input coordinates
% to get delta_theta ( in degrees)
[theta_11c, theta_12c, theta_13c] = IK(p_current(1),
p_current(2),p_current(3));
[theta_11d, theta_12d, theta_13d] = IK(p_desired(1),
p_desired(2),p_desired(3));

delta_theta = [theta_11d-theta_11c, theta_12d-theta_12c, theta_13d-
theta_13c];
% 360 deg     ................. 16000 step
% delta_theta .................  n_steps
n_steps =  16000.*delta_theta / 360;

accelerating_steps  = 0.2 .* n_steps;
deccelerating_steps = -1*accelerating_steps;
uniform_speed_steps = 3.*accelerating_steps;
% Getting the max. speed
max_speed    = uniform_speed_steps ./ 0.6*t;
speed        = max_speed .* 0.0225 .* pi/180; % rad/s
acceleration = speed ./ 0.2*t; % rad/s^2
%% Calculating active and passive joint angles
% Defining delta robot parameters in mm.
b  =  50;   % radius of moving platform
a  =  200;  % radius of fixed platform
L1 =  300 ; % bicep (L1)
L2 =  1000; % forearm (L2)

phai_i   = [0, 2*pi/3, -2*pi/3];
% Variables used to get theta_3i
b_2i     = zeros(3,1);
theta_3i = zeros(3,1);
% Variables used to get theta_2i
b_3i     = p_desired(3);
b_1i     = zeros(3,1);
k        = zeros(3,1);
theta_2i = zeros(3,1);

Jx = zeros(3,3); % dimensionless
% Variables to get the torque
ma = 0.180;
mb = 0.2334;
mp = 0.02;

for i = 1:3

    b_2i(i)     = -1*sin(phai_i(i)) *p_desired(1) + cos(phai_i(i))
*p_desired(2);
    theta_3i(i) =  acos(b_2i(i)/L2);

    b_1i(i)     = cos(phai_i(i)) *p_desired(1) + sin(phai_i(i)) *p_desired(2)
+ (b-a);
    k(i)        = (b_1i(i)^2 + b_2i(i)^2 + b_3i^2 - L1^2 - L2^2) ./ (
2*L1*L2* sin(theta_3i(i)) );
    theta_2i(i) = acos(k(i));
```

```
% Now that we have the active and passive joint angles values:
    place_holder  = [sin(theta_2i(1)) *sin(theta_3i(1)), sin(theta_2i(2))
*sin(theta_3i(2)), sin(theta_2i(3)) *sin(theta_3i(3))];
    Jq = L1* diag(place_holder); % in mm.
    Jx(i,1) = cos( theta_1i(i)+ theta_2i(i) ) *sin(theta_3i(i)) *
cos(phai_i(i)) - ...
    cos(theta_3i(i))* sin(phai_i(i));
    Jx(i,2) = cos( theta_1i(i)+ theta_2i(i) ) *sin(theta_3i(i)) *
sin(phai_i(i)) + ...
        cos(theta_3i(i))* sin(phai_i(i));
    Jx(i,3) = sin( theta_1i(i)+ theta_2i(i) ) * sin(theta_3i(i));
    % Getting The Linear velocity of desired point.
    J = Jx\Jq; % in mm
end
%% Calculating actuator torques
% lamda1 coeffients
m = p_desired(1)+ (b-a)*cos(phai_i(1)) - L1*cos(phai_i(1))*cos(theta_1i(1));
p = p_desired(2)+ (b-a)*sin(phai_i(1)) - L1*sin(phai_i(1))*cos(theta_1i(1));
s = p_desired(3)- L1*sin(theta_1i(1));
% lamda2 coeffients
n = p_desired(1)+ (b-a)*cos(phai_i(2)) - L1*cos(phai_i(2))*cos(theta_1i(2));
q = p_desired(2)+ (b-a)*sin(phai_i(2)) - L1*sin(phai_i(2))*cos(theta_1i(2));
t = p_desired(3)- L1*sin(theta_1i(2));
% lamda3 coeffients
o = p_desired(1)+ (b-a)*cos(phai_i(3)) - L1*cos(phai_i(3))*cos(theta_1i(3));
r = p_desired(2)+ (b-a)*sin(phai_i(3)) - L1*sin(phai_i(3))*cos(theta_1i(3));
u = p_desired(3)- L1*sin(theta_1i(3));
A = [m,n,o; p,q,r; s,t,u];
% p_dot = [theta_11_dot, theta_12_dot, theta_13_dot]
p_dot = J *speed'; % in mm/s.
p_ddot = J* acceleration'; % mm/s

b1 = 0.5*(mp+3*mb) *p_ddot(1);
b2 = 0.5*(mp+3*mb) *p_ddot(2);
b3 = 0.5*(mp+3*mb) *p_ddot(3) + 0.5*(mp+3*mb)*9.81;
B = [b1;b2;b3];

lamdas = A\B; % = inv(A)*B
T11 = (1/3*ma+mb)*L1^2*theta_1i(1) + (1/2*ma+mb)*9.81*L1*cos(theta_1i(1));
T12 = 2*L1*lamdas(1)*( (p_desired(1)*cos(phai_i(1))+
p_desired(2)*sin(phai_i(1))+ b-a)*sin(theta_1i(1))-
p_desired(3)*cos(theta_1i(1)) );
T1  = T11 - T12;

T21 = (1/3*ma+mb)*L1^2*theta_1i(2) + (1/2*ma+mb)*9.81*L1*cos(theta_1i(2));
T22 = 2*L1*lamdas(2)*( (p_desired(1)*cos(phai_i(2))+
p_desired(2)*sin(phai_i(2))+ b-a)*sin(theta_1i(2))-
p_desired(3)*cos(theta_1i(2)) );
T2  = T21 - T22;

T31 = (1/3*ma+mb)*L1^2*theta_1i(3) + (1/2*ma+mb)*9.81*L1*cos(theta_1i(3));
T32 = 2*L1*lamdas(3)*( (p_desired(1)*cos(phai_i(3))+
p_desired(2)*sin(phai_i(3))+ b-a)*sin(theta_1i(3))-
p_desired(3)*cos(theta_1i(3)) );
T3 = T31 - T32;
T = [T1;T2;T3]*10e-6 % in N.m
```

4.5 Motion simulation

4.5.1 V-Rep IDE

V-REP provides users with rigid bodies, joints, constraints and drives, sensors and actuators. It can model and simulate multi-rigid body mechanisms connected by various motion pairs, and realize the analysis of dynamic performance of mechanism system. We used V-REP to do kinematics simulation. Without deducing kinematics model of transmission mechanism, they can construct graphical simulation model directly by using relevant modules, which is more effective for complex mechanism. V-REP can also connect the control system model to provide an effective environment for the modeling and Simulation of mechatronic mechanical mechanisms, and it brings great convenience to the construction and further research of Delta parallel robot model. In this study, we mainly concentrate on inverse kinematics because it is our main concern in the control field and also to visualize the motion of the robot and establish even stronger knowledge about delta robot.

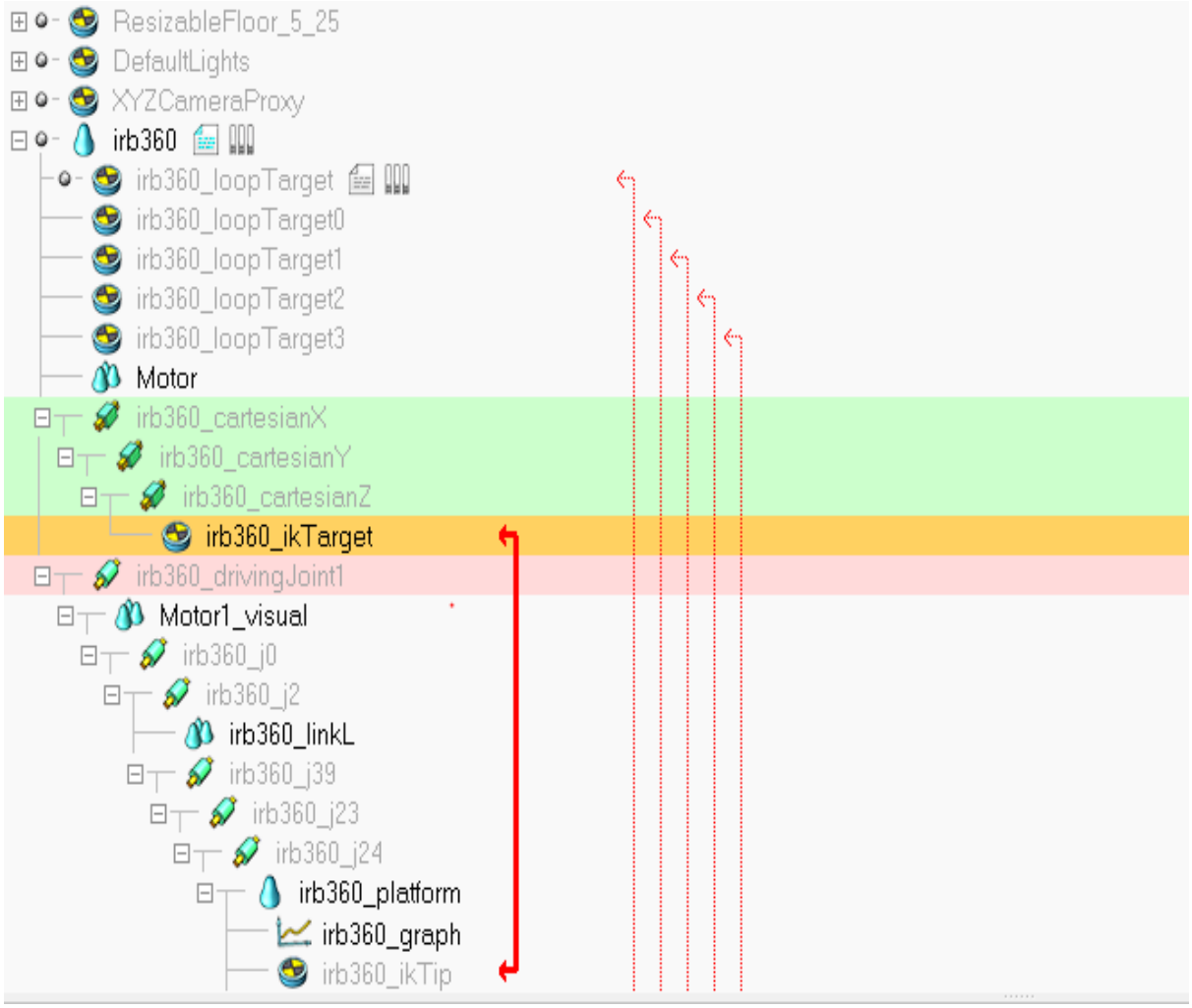


Fig.4.15: Scene hierarchy of the delta robot

4.5.2 URDF from CAD

The first step in this process is to convert the 3D SOLIDWORKS model into Unified Robot Description Format (URDF) [76] that V-REP can work with so we used a module called "sw2urdf" which is added to the SOILDWORKS™ environment to convert the 3D model into which is a format of any robotic structure that indicate joints, bodies, constraints, and actuators in hierarchy format that indicate actuators, sensors, joints, and rigid bodies as shown in **Fig.4.15**.

Using "sw2urdf" module [77], the model can be generated automatically or with some manual work, but Generating URDF file may be very tedious especially in parallel manipulators so with the help of predefined examples that V_REP represents it may be accomplished by imitation. Parallel manipulators are represented as three open loop legs connected at the end effector one of them is regarded as the driving leg and the other two follow it to solve the inverse kinematics problem, as one motor moves until it reaches the destination point while other two solve their inverse kinematics equations and find to what extend they should move. On the other hand, all actuators move simultaneously. we also implemented a drawing toll in the end effector so that we draw the path which the robot takes to reach its destination to help us understand the motion of it as shown in **Fig.4.16**.

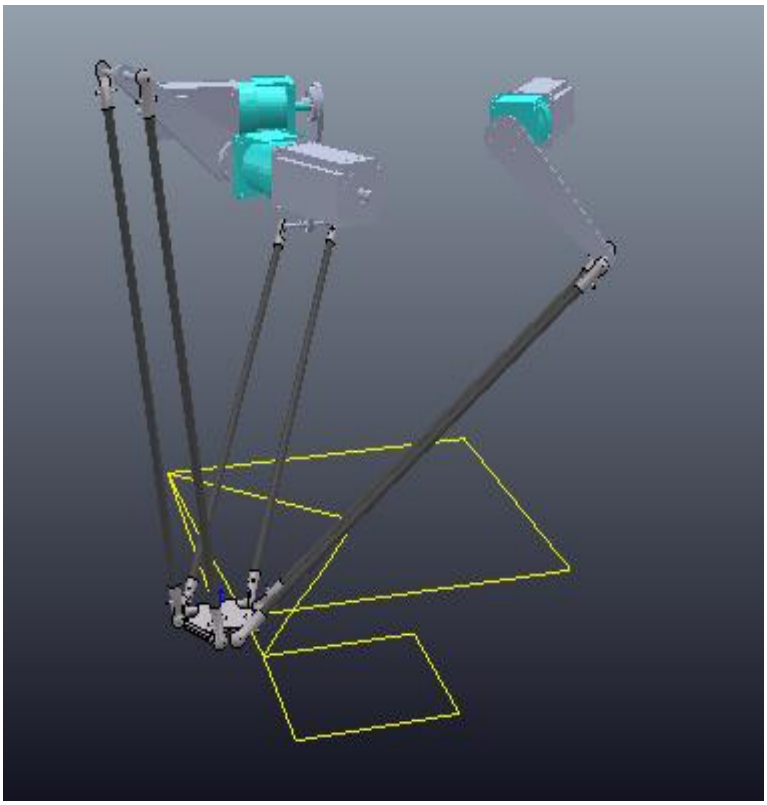


Fig.4.16: Motion path demonstration

V-REP also has main components with which any robot can be modeled and simulated such as actuators joints, rigid bodies, and dummies which is used mainly to retrieve its position and make the system aware where each link is. also, it is used to move the actuators as they are positioned in the destination and the link should follow them to reach it too. This type of connection is called Link-Type [78] which can be ik mode or an overlap constraint in our model which is represented as vertical red arrow connecting two dummies as shown in **Fig.4.16**. It also provides us with scripting tools in LUA language to validate more complex motion simulation the developers of V-REP provide detailed demonstrations with clear examples on their website that we laid on while making this simulation. scripts can be either threaded or non-threaded; the difference is that threaded is used in case it's wanted to repeat executing this script until user stops it, meanwhile non-threaded is executed only in the beginning to check whether everything works fine or there are failures such as destination point is out of the workspace of the robot, or there is no solution for the inverse kinematics problem.

4.5.3 Associated simulation scripts

In threaded script the simulator is to establish both forward and inverse kinematics function that will control the motion of the robot these models have nothing to do with the mathematical modeling it is based on some algorithms inside the simulator after that some parameters must be passed such as angular velocity and angular acceleration as well as maximum angular velocity these can vary from robot to another so they are user defined after that the simulator checks whether the non-threaded script detected an error if not it retrieves the robot to its initial state using forward kinematics so that it starts from the predefined start point, after that the user is choose whether to operate in forward kinematics which is implemented through **'sim.moveToJointPositions'** command which moves the driving joints to its destination, or inverse kinematics mode through **'sim.moveToJointPositions'** command also with the help of some dummies that move to the destination and then the driving joints follow these dummies.

The simulator then follows certain sequence of points that the user has defined and the it checks whether these points are reachable or not if yes then this sequence is executed and the whole process is repeated until the user stops it. The simulator also provide us with suction cup or grippers which give the user more variety and complexity of simulation.

4.5.3.1 Threaded script

```
-- There is another child script in this model. The other one is attached
to object 'irb360_loopTarget' and doesn't run in a thread and is in charge
of explicitly executing the inverse kinematics calculation

setFkMode=function()
    -- disable the platform positional constraints:
    sim.setIkElementProperties(mainIkTask,ikModeTipDummy,0)
    -- Set the driving joints into passive mode (not taken into account
    during IK resolution):
    sim.setJointMode(fkDrivingJoints[1],sim.jointmode_passive,0)
    sim.setJointMode(fkDrivingJoints[2],sim.jointmode_passive,0)
    sim.setJointMode(fkDrivingJoints[3],sim.jointmode_passive,0)
end

setIkMode=function()
    -- Make sure the target position is at the same position at the tip
    position:
    local p=sim.getObjectPosition(ikModeTipDummy,irb360Base)
    sim.setJointPosition(ikDrivingJoints[1],p[1]-initialPosition[1])
    sim.setJointPosition(ikDrivingJoints[2],p[2]-initialPosition[2])
    sim.setJointPosition(ikDrivingJoints[3],p[3]-initialPosition[3])
    -- enable the platform positional constraints:
    sim.setIkElementProperties(mainIkTask,ikModeTipDummy,
    sim.ik_x_constraint+sim.ik_y_constraint+sim.ik_z_constraint)
    -- Set the base joints into ik mode (taken into account during IK
    resolution):
    sim.setJointMode(fkDrivingJoints[1],sim.jointmode_ik,0)
    sim.setJointMode(fkDrivingJoints[2],sim.jointmode_ik,0)
    sim.setJointMode(fkDrivingJoints[3],sim.jointmode_ik,0)
end

function sysCall_threadmain()
    -- Put some initialization code here:
    -- Retrieve some values:
    mainIkTask=sim.getIkGroupHandle('irb360_mainTask')
    ikModeTipDummy=sim.getObjectHandle('irb360_ikTip')
    -- Following are the joints that we control when in FK mode:
    fkDrivingJoints={-1,-1,-1}
    fkDrivingJoints[1]=sim.getObjectHandle('irb360_drivingJoint1')
    fkDrivingJoints[2]=sim.getObjectHandle('irb360_drivingJoint2')
    fkDrivingJoints[3]=sim.getObjectHandle('irb360_drivingJoint3')
    -- Following are the joints that we control when in IK mode (we use
    joints so that we can use the sim.moveToJointPositions command here
    ikDrivingJoints={-1,-1,-1}
    ikDrivingJoints[1]=sim.getObjectHandle('irb360_cartesianX')
    ikDrivingJoints[2]=sim.getObjectHandle('irb360_cartesianY')
    ikDrivingJoints[3]=sim.getObjectHandle('irb360_cartesianZ')

    irb360Base=sim.getObjectAssociatedWithScript(sim.handle_self)
```

```
angVel=180*math.pi/180
angAccel=360*math.pi/180
angJerk=3600*math.pi/180
currentAngVel={0,0,0,0}
currentAngAccel={0,0,0}
maxAngVel={angVel,angVel,angVel}
maxAngAccel={angAccel,angAccel,angAccel}
maxAngJerk={angJerk,angJerk,angJerk}
targetAngVel={0,0,0}

linVel=2
linAccel=3
linJerk=30
currentLinVel={0,0,0}
currentLinAccel={0,0,0}
maxLinVel={linVel,linVel,angVel}
maxLinAccel={linAccel,linAccel,angAccel}
maxLinJerk={linJerk,linJerk,angJerk}
targetAngVel={0,0,0}

-- First, make sure we are in initial position:
setFkMode()
sim.rmlMoveToJointPositions(fkDrivingJoints,-
1,currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
{0,0,0},targetAngVel)
initialPosition=sim.getObjectPosition(ikModeTipDummy,irb360Base)

-- The main routine here:
-- In following loop, we move alternatively in forward, then
inverse kinematic mode.
-- We drive the main joints (the motors) in FK mode with the
'sim.moveToJointPositions'
-- In IK mode, we could directly set the desired tip position for \
the 'irb360_ikTarget' object
-- with the 'simMoveToPositon' command. However, in order to be
able to drive simultaneously the
-- rotational joint, we simply attached the 'irb360_ikTarget' to 3
cartesian joints, that we drive
-- in a similar way as in FK mode. We could also imagine writing
additional threads where each could
-- take care of one motor for instance.
-- The 'sim.moveToJointPositions' command, as used below, will
drive all joints simultaneously (they start
-- and stop at the same time). In order to be able to drive the
central axis much faster than the other
-- joints, we applied a trick: the rotational motor is built on top
of another motor ('irb360_motorAux')
-- that is linearly dependent of the first one (motorAux=3*motor).
So if we drive the axis motor to x degrees,
-- the total rotation will be 4x degrees. So we have to remember to
feed always 1/4 of the desired angular
-- value for the central axis.
```

```
While
sim.getSimulationState()~=sim.simulation_advancing_abouttostop do

    -- First in forward kinematics mode:
    setFkMode()

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {80*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {45*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,80*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,45*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,-80*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,45*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {-80*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {-80*math.pi/180,-80*math.pi/180,0*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,-80*math.pi/180,-80*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,80*math.pi/180},targetAngVel)

    sim.rmlMoveToJointPositions(fkDrivingJoints,1,
    currentAngVel,currentAngAccel,maxAngVel,maxAngAccel,maxAngJerk,
    {0*math.pi/180,0*math.pi/180,0*math.pi/180},targetAngVel)
```

```
-- Now in inverse kinematics mode:
        setIkMode()
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0.3,0.3,0},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {-0.3,0.3,0},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
           {-0.3,-0.3,0},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0.3,-0.3,0},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0.3,0.3,0},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0.15,0.15,-0.3},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {-0.15,0.15,-0.3},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {-0.15,-0.15,-0.3},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0.15,-0.15,-0.3},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0.15,0.15,-0.3},targetLinVel)
        sim.rmlMoveToJointPositions(ikDrivingJoints,1,
        currentLinVel,currentLinAccel,maxLinVel,maxLinAccel,maxLinJerk,
        {0,0,0},targetLinVel)
    end
end
```

4.5.3.2 Non-threaded script

```
function sysCall_init()
    allIkTasks={-1,-1,-1,-1,-1,-1,-1,-1}
    allIkTasks[1]=sim.getIkGroupHandle('irb360_mainTask')
    allIkTasks[2]=sim.getIkGroupHandle('irb360_bridge')
    allIkTasks[3]=sim.getIkGroupHandle('irb360_bridge0')
    allIkTasks[4]=sim.getIkGroupHandle('irb360_bridge1')
    allIkTasks[5]=sim.getIkGroupHandle('irb360_bridge2')
    allIkTasks[6]=sim.getIkGroupHandle('irb360_bridge3')
    allIkTasks[7]=sim.getIkGroupHandle('irb360_bridge4')
    allIkTasks[8]=sim.getIkGroupHandle('irb360_axis')
    baseObjectHandle=sim.getObjectHandle('irb360')
end
-- We handle inverse kinematics explicitly in this script, so that we
can control
-- what happens if an IK task fails

function sysCall_cleanup()

end
```

```
function sysCall_actuation()
    -- Now save the robot joint configuration in case IK fails at
    the next stage.
    -- We have to explore the tree structure of the model:
    toExplore={baseObjectHandle}
    jointsAndTheirPositions={}
    while (#toExplore>0) do
        obj=toExplore[#toExplore]
        table.remove(toExplore,#toExplore)
        if (sim.getObjectType(obj)==sim.object_joint_type) then
        table.insert(jointsAndTheirPositions,obj)
        table.insert(jointsAndTheirPositions,sim.getJointPosition(obj))
        end
        index=0
        child=sim.getObjectChild(obj,index)
        while (child~=-1) do
            table.insert(toExplore,child)
            index=index+1
            child=sim.getObjectChild(obj,index)
        end
    end
    --Execute the IK tasks (order matters!):
    ikFailed=false
    for i=1,#allIkTasks,1 do
        if (sim.handleIkGroup(allIkTasks[i])==sim.ikresult_fail) then
            ikFailed=true
            break
        end
    end
```

4.6 Stress analysis

4.6.1 ANSYS transient structural analysis

Dynamic optimization uses dynamic structural analysis in defining the strains, stresses, and deformations that affect the structure, so as to optimize materials, geometry, and the mass of each mechanical structure component, and to commercially select the robot's actuators. The main objective of this section is dynamic optimization of delta-type parallel robot, based on dynamic structural analysis. This analysis enables optimization of the robot as to materials, geometry, mass, and selection of actuators. The result is more economic and efficient structures that will meet desired operation specifications.

This analysis is done through **ANSYS Workbench 18.0** using transient structural analysis in **Mechanical** library. It analyzes the dynamic response of a structure under the action of any general time-dependent loads. The transient structural analysis is the most versatile dynamic analysis and it can use it to determine the time-varying displacements, strains, stresses, and forces in a structure as it responds to any transient loads. A typical application of the transient structural analysis is to investigate the response of a structure under a transient load. The Transient structural analysis is demanding on computing resource. It should only be used when

the inertia or damping effects are considered to be important under the time scale of loadings. **ANSYS Workbench 18.0** algorithms depend on iterating the calculation until it reaches convergence of results so that it needs huge computing power as it does this process for each mesh in the model. [79]

4.6.2 ANSYS model setup

Before creating an ANSYS project simplification of original 3D model so to remove all the unnecessary detail such a motor and gear box which do not affect the analysis but consumes computing power for rendering and meshing as shown in **Fig.4.18**, then materials are defined

Fig.4.17: Simplified 3D model of the delta robot

in the project which are basically aluminum and air which stands for weightless bodies that are neglected such as ball joints, then the simplified model in Fig (4.18) is imported, afterwards **Mechanical** is started which is library in **ANSYS Workbench** that enables the user to identify joints, actuators, sensors, nature of part either rigid, or flexible.

Setting up the model is done through certain steps which are determining nature of each body and its material, applying joints, meshing, defining initial conditions and simulation parameters such as number of steps and step time, applying acting forces such as acting torques and gravitational force, and finally adding sensors to measure the produced torque, equivalent stress, total strain, and total deformation. As mentioned before we mainly would assign aluminum and air to the bodies in this simulation , then the three biceps (or any other flexible body that is

desired to undergo the analysis), then joints are applied such as revolute joint between the horizontal link and bicep, and spherical joint between link and ball joint. **ANSYS Workbench** automatically defines contacts between any to assembled bodies which could be removed but in our case it is vital to assign all of these contacts to asymmetric behavior and ensure that the flexible body is the contact and the rigid body is the target then all these contacts are set to be frictionless assign any part that we need to examine its total stress to flexible and convert any of its contacts to asymmetric with flexible body as contact and rigid body as target then meshing hexahedral method is applied which ensure more uniform distribution and faster calculations as shown in **Fig.4.19,**

Fig.4.18: Heaxhedral meshing on the three beiceps

Then simulation parameter are assigned such as step time and number of steps in the model afterwards forces are applied as shown in **Fig.4.20** forces can take any from such as constant or time dependent of course to have good simulation forces must be time dependent but this needs huge computing power so we assumed to have constant acting forces, finally sensors are inserted which are basically three moment sensor at the biceps so that we can specify maximum needed torque for motor sizing, strain sensor so that we figure out which position holds the maximum strain and design the flexible parts based on that position and finally total stress to determine whether the body can withstand the load or needs to get thicker or better material.

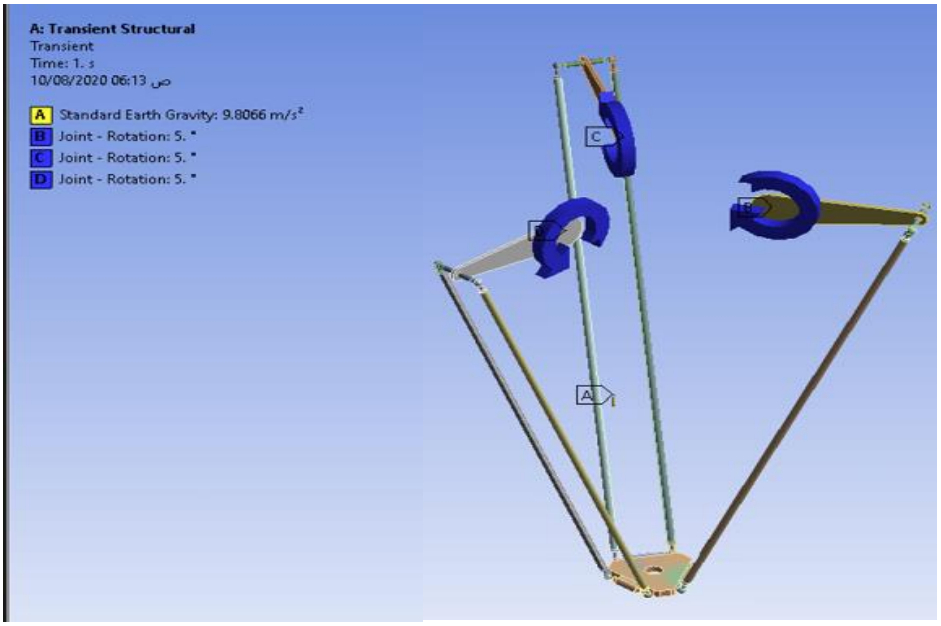


Fig.4.20: Acting forces and torques on delta robot

It should be mentioned that one point simulation is not the correct model as to simulate the worst case that the robot could examine during operation, transferring between the limits of the workspace should be implemented but due to the lack of time and more importantly the lack of computing power as one point only with three angles took about 5 minutes on our average properties to get the results so simulating the analysis of the whole body on the worst case scenario would take ages to be completed, finally solving the model would output the following results which are very acceptable as shown in **Fig.4.20** and **Fig.4.21**

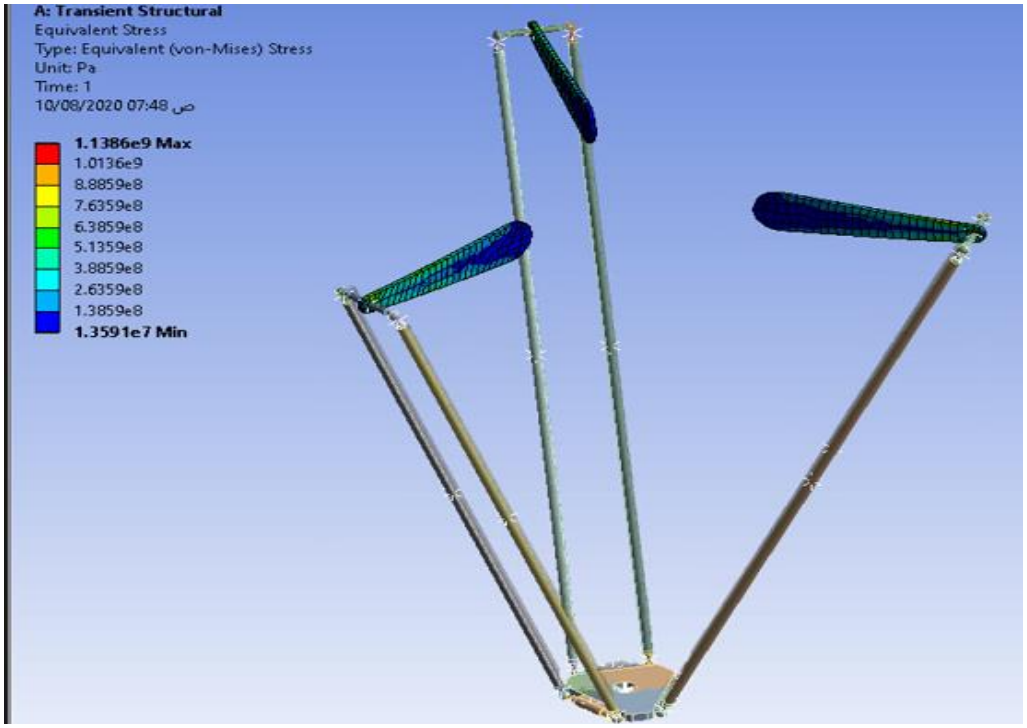


Fig.4. 19: Equivalent stress on the three biceps

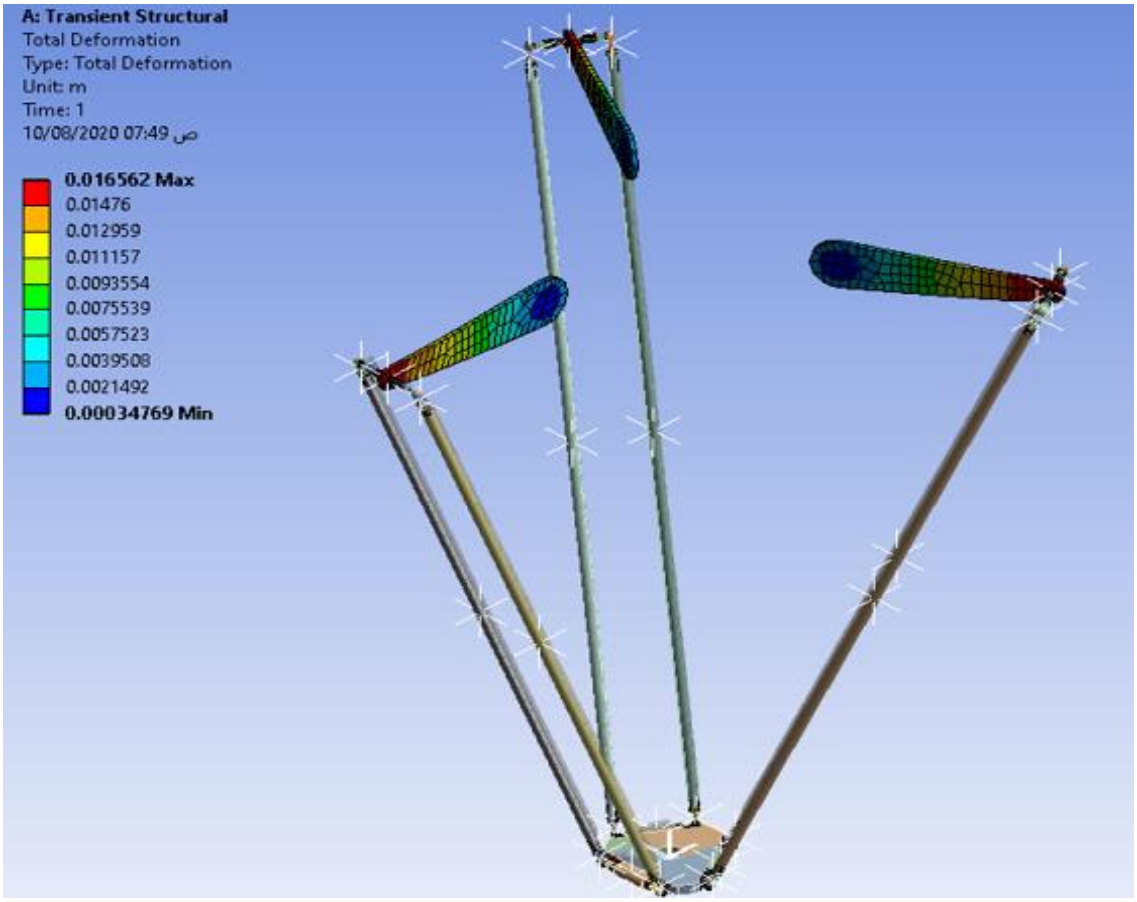


Fig.4.21: Total deformation on the three biceps.

4.6.3 Forearm stress analysis

As mentioned in the introduction one of the main advantages of delta robot that the forearm only suffers normal stress as the ball joint movement cancels any shear or bending stresses which enables the designer to choose low density material to ensue light weight without many concerns about mechanical failure.

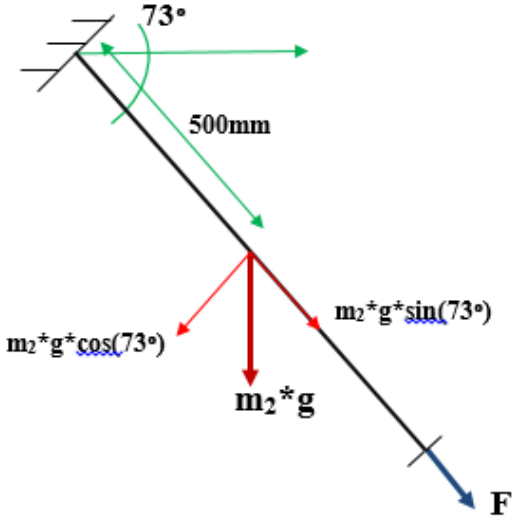


Fig.4.22: Force distribution on the forearm

This arm is studied in its full stretched case as this case maximizes the normal stress on the forearm. This beam only suffer bending moment due to the weight of the beam itself and tension due to the weight of the end effector and the weight of picked object.

First the tension force basically comes from the weight of end effector, picked object, ball joint, and the tangential component of forearm weight force

$$\boldsymbol{F = \frac{1}{6} * 9.8 * (0.02 + 0.5) + (0.35 + 0.07) * 9.8 * sin(73) = 4.3\ N} \quad \textbf{(4.65)}$$

Then tension stress equation is applied

$$\boldsymbol{\sigma_t = \frac{F}{A} = \frac{F}{\frac{\pi}{4} * (D_o^2 - D_i^2)}} \quad \textbf{(4.66)}$$

$$\boldsymbol{\sigma_t = \frac{1.5}{14.7262} = 0.29\ MPa} \quad \textbf{(4.67)}$$

Second is bending stress due to the normal component of the forearm weight force

$$\boldsymbol{\sigma_b = \frac{M * y}{I}} \quad \textbf{(4.68)}$$

Where

$$\boldsymbol{M = F * \frac{l}{2} = 0.02 * 9.8 * cos(73) * 500 = 28.65\ N.mm} \quad \textbf{(4.69)}$$

$$\boldsymbol{\sigma_b = \frac{28.65 * 9.5}{647.26} = 0.42\ MPa} \quad \textbf{(4.70)}$$

$$\boldsymbol{I = \frac{\pi}{64} * (D_o^4 - D_i^4) = 647.26\ mm^4} \quad \textbf{(4.71)}$$

Then we apply design equation

$$\boldsymbol{\sigma_T = \sigma_b + \sigma_t = 0.42 + 0.29 = 0.71\ MPa} \quad \textbf{(4.72)}$$

$$\boldsymbol{\sigma_{design} = \sigma_T * F.O.S = 0.71 * 2 = 1.42\ MPa} \quad \textbf{(4.73)}$$

Which is very small stress and can easily be neglected when compared to aluminum yield stress=276 MPa in Appendix (1). [80]

4.6.4 Horizontal link stress analysis

Horizontal link is classified as the weakest spot in the structure as it suffers from various types of stresses such as bending moment and shear due to torque as well as it must remain light weighted to decrease the total inertia of the structure and ensure high speed performance in addition to that it must not exceeds 8 mm in diameter because it is connected to the ball joint and ball will not move if the link is thicker than 8 mm because of all of that and even more this link is to be studied with great care.

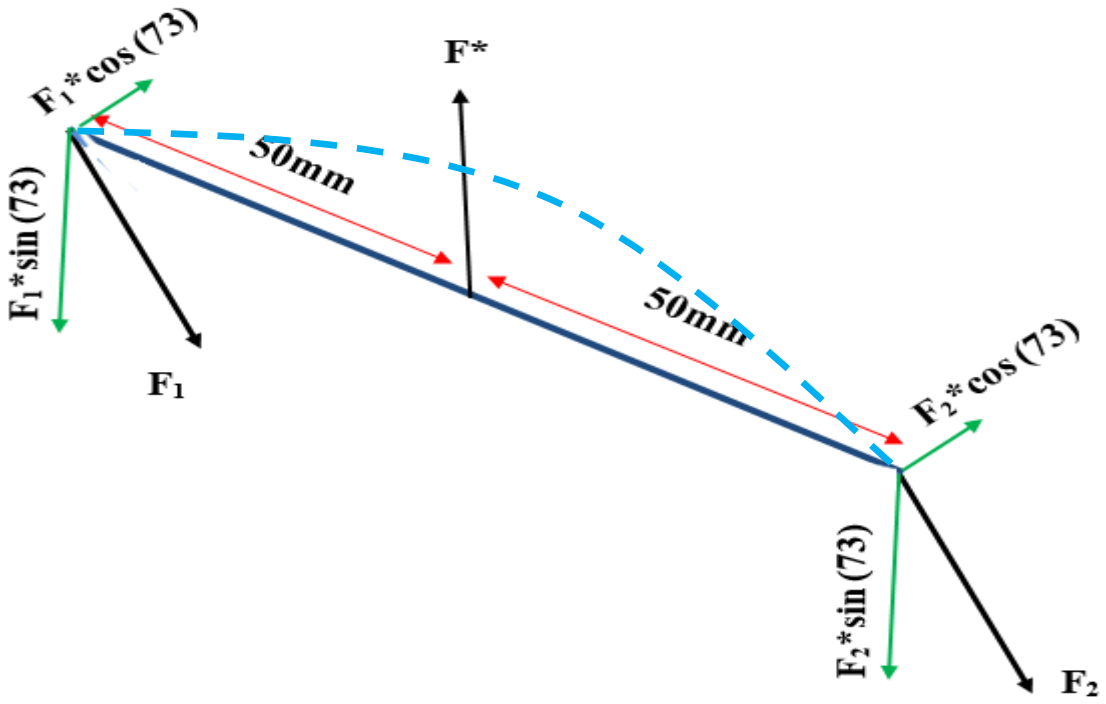


Fig.4. 23: Force distribution on the horizontal link

First we will examine the bending moment because it is the dominant stress in our case it arises basically from weight of forearm, ball joints, end effector, and the picked object each of these forces is divided equally on the six legs of the parallelogram, but translating these forces to the link is also accompanied with moments that would cause torsion except that the existence of ball joint eliminate any torsion on the link. It is also important to mention that the worst case happens when the robot is fully stretched that is where the weight component in the direction of the forearm axis is maximum, also for simplicity we assume that the link is a perfect cylinder as it has some curvatures and fillets.

$$\mathbf{F'_1} = \frac{\mathbf{1}}{\mathbf{6}} * \mathbf{9.8} * (\mathbf{0.5} + \mathbf{0.02}) * \mathbf{sin}(\mathbf{73}) = \mathbf{0.812\ N} \qquad \textbf{(4.74)}$$

$$\mathbf{F'_2 = (2 * 0.07 * 9.8 + 0.35 * 9.8) * \sin(73) = 4.802N} \quad \textbf{(4.75)}$$

$$\mathbf{F_1 = F_2 = F'_1 + F'_2 = 5.614 \approx 5.6\ N} \quad \textbf{(4.76)}$$

Calculating the bending moment and moment of inertia of the link

$$\mathbf{M = 2 * F_1 * \frac{l}{2} = 2 * 5.6 * 50 = 560\ N.mm} \quad \textbf{(4.77)}$$

$$\mathbf{I = \frac{\pi}{64} * (D^4) = 201.06\ mm^4} \quad \textbf{(4.78)}$$

Applying the bending stress equation Eqn **(4.79)** then decomposing it into the **x**, **y** directions

$$\mathbf{\sigma_b = \frac{M * y}{I} = \frac{560 * 4}{201.06} = 12\ MPa} \quad \textbf{(4.79)}$$

$$\mathbf{\sigma_x = \sigma_b * \cos(73) = 3.5\ MPa} \quad \mathbf{, \sigma_y = \sigma_b * \sin(73) = 11.47\ MPa} \quad \textbf{(4.80)}$$

Secondly shear stress die to the motor torque applied on the bicep that lifts and lowers the horizontal link is to be examined

$$\mathbf{F = \frac{T}{r} = \frac{40000}{300} = 133.3\ N} \quad \textbf{(4.81)}$$

$$\mathbf{A = \frac{\pi}{4} * D^2 = 50.26\ mm^2} \quad \textbf{(4.82)}$$

$$\mathbf{\tau = \frac{F}{A} = \frac{133.3}{50.26} = 2.65\ MPa} \quad \textbf{(4.83)}$$

Now we would consider the combined stress Eqn (4.84) because the beam is subjected to both shear and bending moment

$$\mathbf{\sigma_{1,2} = \left(\frac{\sigma_x + \sigma_y}{2}\right) \pm \sqrt{\left(\frac{\sigma_x + \sigma_y}{2}\right)^2 + \tau_{xy}{}^2}} \quad \textbf{(4.84)}$$

$$\mathbf{\sigma_{max} = \left(\frac{3.5 + 11.47}{2}\right) + \sqrt{\left(\frac{3.5 - 11.47}{2}\right)^2 + 2.65^2} = 12.27\ MPa} \quad \textbf{(4.85)}$$

Then design Eqn (4.86) is applied with factor of safety=2

$$\mathbf{\sigma_{design} = \sigma_{max} * F.O.S = 24.54\ \ MPa} \quad \textbf{(4.86)}$$

Using the same combined stress equation to calculate the total shear that the beam is subjected to in Eqn (4.87)

$$\tau_{1,2} = \pm\sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + {\tau_{xy}}^2} \quad (4.87)$$

$$\tau_{max} = \sqrt{\left(\frac{3.5 - 11.47}{2}\right)^2 + 2.65^2} = 4.78 \text{ MPa} \quad (4.88)$$

Then design Eqn (4.86) is applied with factor of safety=2

$$\tau_{design} = \tau_{max} * F.O.S = 9.56 \text{ MPa} \quad (4.86)$$

Comparing these values of stress with the yield stress it is found that

$$\sigma_{design} \ll \sigma_{yield} = 276\ MPa \quad (4.87)$$

$$\tau_{design} \ll \frac{\sigma_{yield}}{2} = 138\ MPa \quad (4.88)$$

The stresses on the beam is too far from the yield stress, then a bar of 8 mm diameter from aluminum can withstand the stresses of our delta robot, but it is always safer to do these calculation in **ANSYS** workbench but as mentioned before duo to the lack in time and computing power it was very difficult to do so.

4.6.5 End Effector

The plate at the end of the delta robot is only subjected to the picked object weight force and since the heaviest targeted object weighs 0.5 kg then all the stress on this part can be neglected but to ensure stiffness this part is to be made from acrylic 8 mm thick in order to be light weighted and quite stiff the properties of acrylic can be found in Appendix (1).

4.6.6 PHS 8

It is the market name of the ball joint which is inner diameter equals 8 mm. ball joints enables the legs of the delta robot to move in 2 direction also universal joint is suitable for this application A ball joint consists of a bearing stud and socket enclosed in a casing it weighs only 70g meanwhile PHS10 weighs 130g so which make the robot about quarter kilogram heavier unnecessarily this joint can hold a normal force of 10kN which is very sufficient to our application where it will be only subjected to small normal forces. [81]

4.7 Vacuum Gripper

4.7.1 Introduction

Although there are many types of gripping tools that can be implement3d with delta robot we chose Vacuum Grippers it can handle a wide range of applications and are ideal for picking up uneven and even work pieces made of different materials, such as cardboard, glass, sheet metal and plastic. Also, easy control makes vacuum gripper the optimal choice for our application. Basically, Vacuum system consists of air with low pressure supply and suction cup there are two means to produce vacuum the first is using vacuum pump as indicated in **Fig.4.24**

Fig.4. 24: Vacuum pump

the second is to use venture principle to generate vacuum which we will demonstrate in this section despite it is much easier to control and design the main disadvantage of vacuum pump that it must operate all the time even if it is not used unlike the system consisting of air compressor, solenoid valve, and vacuum generator which is indicated in **Fig.4.25.** [82]

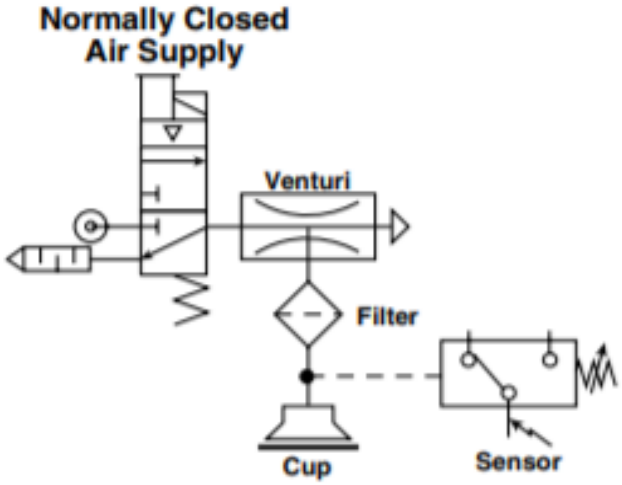


Fig.4. 25: Schematic of vacuum gripper

4.7.2 Suction cup

4.7.2.1 Theory of operation

The working face of the suction cup is made of elastic, flexible material and has a curved surface. When the center of the suction cup is pressed against a flat, non-porous surface, the volume of the space between the suction cup and the surface is reduced, which causes the air between the cup and the surface to be expelled past the rim of the circular cup as shown in Fig **(4.27)**. The cavity which develops between the cup and the surface has little to no air or water in it because most of the fluid has been forced out of the inside of the cup, causing a lack of pressure. The pressure difference between the atmosphere on the outside of the cup and the low-pressure cavity on the inside of the cup keeps the cup adhered to the surface.

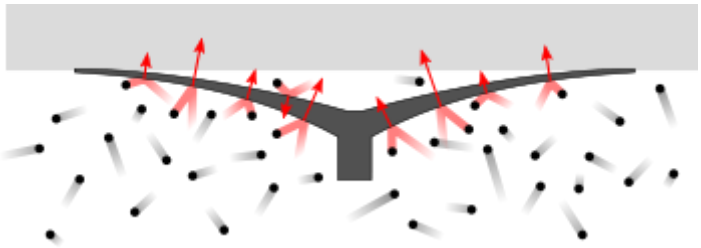


Fig.4.26: One cup suction lifter

When the user ceases to apply physical pressure to the outside of the cup, the elastic substance of which the cup is made tends to resume its original, curved shape. The length of time for which the suction effect can be maintained depends mainly on how long it takes for air or water to leak back into the cavity between the cup and the surface, equalizing the pressure with the surrounding atmosphere. This depends on the porosity and flatness of the surface and the properties of the cup's rim. A small amount of mineral oil or vegetable oil is often employed to help maintain the seal.

4.7.2.2 Types of Suction cups

The suction cup can be categorized into four different types shown in **Fig.4.28**:

1. Universal suction cups
2. Flat suction cups with bars
3. Suction cups with bellow
4. Depth suction Cup

Flat suction cups are suitable for flat or flexible items that need assistance when lifted, suction cups with bellows are usually used for curved surfaces, and depth suction cup can be used for surfaces that are very irregular and curved or when an item needs to be lifted over an edge. [83]

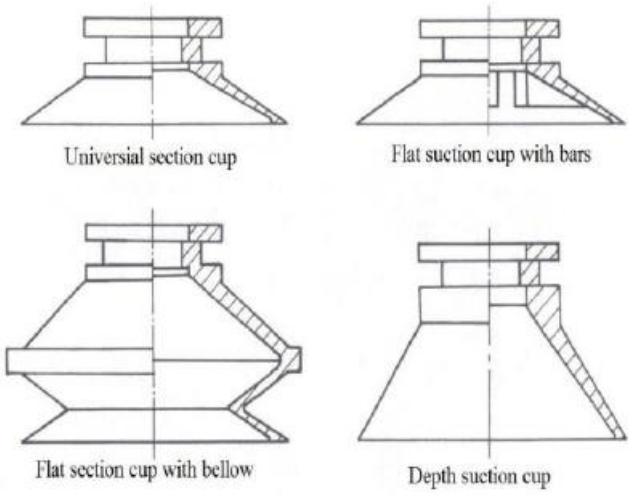


Fig.4.27: Different shapes of suction cup

4.7.3 Compressors

4.7.3.1 Theory of operation

Compressors are mechanical devices used to increase pressure in a variety of compressible fluids, or gases, the most common of these being air. Compressors are used throughout industry to provide shop or instrument air; to power air tools, paint sprayers, and abrasive blast equipment; to phase shift refrigerants for air conditioning and refrigeration; to propel gas through pipelines; etc. As with pumps, compressors are divided into centrifugal (or dynamic or kinetic) and positive-displacement types, compressors are more often of the positive-displacement type. They can range in size from the fits-in-a-glovebox unit that inflates tires to the giant reciprocating or turbo compressor machines found in pipeline service. Positive-displacement compressors can be further broken out into reciprocating types, where the piston style predominates, and rotary types such as the helical screw and rotary vane. [84]

4.7.3.2 Types of Air Compressors

Compressors may be characterized in several different ways, but are commonly divided into types based on the functional method used to generate the compressed air or gas

1. Piston
2. Helical Screw
3. Sliding vane
4. Centrifugal

In selecting air compressors for general shop use, the choice will generally come down to a piston compressor or a helical-screw compressor. Piston compressors tend to be less expensive

than screw compressors, require less sophisticated maintenance, and hold up well under dirty operating conditions, but they are much noisier than screw compressors.

4.7.3.3 Piston Compressor

Piston compressors, or reciprocating compressors as shown in Fig (4.), rely on the reciprocating action of one or more pistons to compress gas within a cylinder (or cylinders) and discharge it through valving into high pressure receiving tanks. In many instances, the tank and compressor are mounted in a common frame or skid as a so-called packaged unit. While the major application of piston compressors is providing compressed air as an energy source. Piston compressors are generally selected on the pressure required (psi) and the flow rate (scfm).

Fig.4.28: Commercial size piston compressor

4.7.4 Solenoid Valve

The definition of a solenoid valve is an electro-mechanical valve that is commonly employed to control the flow of liquid or gas. There are various solenoid valve types, but the main variants are either pilot operated or direct acting. Pilot operated valves, the most widely used, utilize system line pressure to open and close the main orifice in the valve body, while Direct operated solenoid valves directly open or close the main valve orifice, which is the only flow path in the valve. They are used in systems requiring low flow capacities or applications with low pressure differential across the valve orifice.

Fig.4.29: 3/2 Solenoid Valve

4.7.4.1 Theory of operation

It controls the flow of liquids or gases in a positive, fully-closed or fully-open mode. They are often used to replace manual valves or for remote control. Solenoid valve function involves either opening or closing an orifice in a valve body, which either allows or prevents flow through the valve. A plunger opens or closes the orifice by raising or lowering within a sleeve tube by energizing the coil. [85]

Solenoid valves consist of a coil, plunger and sleeve assembly. In normally closed valves, a plunger return spring holds the plunger against the orifice and prevents flow. Once the solenoid coil is energized, the resultant magnetic field raises the plunger, enabling flow. When the solenoid coil is energized in a normally open valve, the plunger seals off the orifice, which in turn prevents flow.

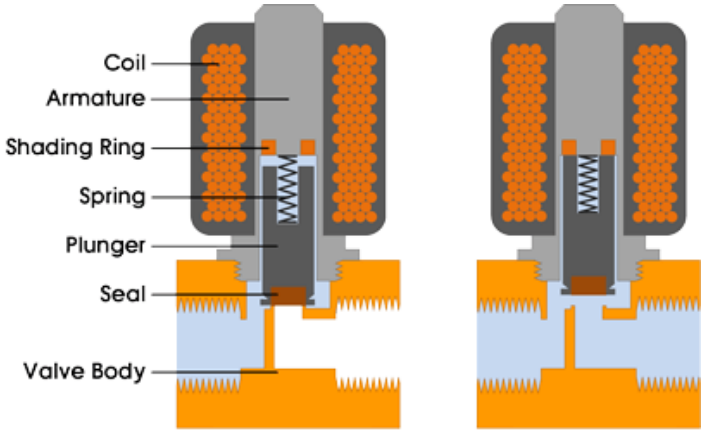


Fig.4.30: Operation of solenoid valve

4.7.4.2 Applications of solenoid valve

By being solenoid actuated, solenoid valves can be positioned in remote locations and may be conveniently controlled by simple electrical switches. Solenoid valves are the most frequently used control elements in fluidics. They are commonly used to shut off, release, dose, distribute or mix fluids. For that reason, they are found in many application areas. Solenoids generally offer fast and safe switching, long service life, high reliability, low control power and compact design. In our application we use it to either let the compressed air flow to the vacuum generator so that the cup picks the object then block that compressed air so that the cup release the object when it places it.

4.7.5 Vacuum Generator

4.7.5.1 principle of VENTURI Vacuum

A vacuum generator is a single stage VENTURI that creates high vacuum with fast response using compressed air. The ability to control this performance renders this technology as an excellent solution for factory automation. In principle, compressed air is throttled as the air exits the nozzle and is discharged into the diffuser [86]. This increased velocity of air lowers the pressure in the diffusion chamber. The volume of air within the closed vacuum system flows into the low-pressure area of the diffusion chamber and is exhausted thru the diffuser. This effect increases the vacuum level and evacuates most of the air within the closed vacuum system at supersonic speeds as shown in **Fig.4.31**.

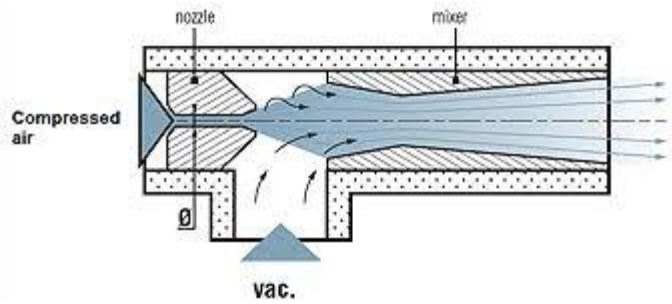


Fig.4. 31: Operation of Vacuum Generator

Vacuum generators have many advantages such as No Moving Components, Low Maintenance, Long Life, Responsive, Physically Small, Cost Effective which makes it the perfect choice for our application rather than vacuum pump which needs regular maintenance and consists of moving parts and physically large.

4.7.6 Vacuum gripper design calculations

Vacuum pressure is recommended to be at the range of 0.5:0.75 kPa for non-porous material such as plastics and metallic objects Coefficient of friction also varies from one surface to the other so we design based on the worst case which is the surface is wet which is likely to happen in residential garbage [87], [83]. As we mentioned before that we attempt to make this machine fully automated, we will take the factor of safety to be 4 to ensure no slipping or faulty grapping. Also, our delta arm is light duty so we can't consider any loads heavier than 500 kg for safety consideration. From the dynamic model we estimate the linear acceleration of the delta robot to be 3:4 times the gravity acceleration so

$$\mu = \mathbf{0.3} \tag{4.89}$$

$$\mathbf{sf} = \mathbf{4} \tag{4.90}$$

$$\mathbf{F} = \frac{\mathbf{m_{max}}}{\mu} * (\mathbf{g} + \mathbf{a}) * \mathbf{sf} \tag{4.91}$$

$$\mathbf{F} = \frac{\mathbf{0.2}}{\mathbf{0.3}} * (\mathbf{9.8} - \mathbf{9.8} * \mathbf{4}) * \mathbf{4} = \mathbf{78.4} \approx \mathbf{80\ N} \tag{7392}$$

4.7.6.1 Suction Cup

Based on these calculations we consider the flat suction cup with bellow as the optimum solution for our application because it is pretty suitable for irregular surfaces such as our application with holding force bagger than 80 N.

From **Festo™** Catalogue we choose Cup with D=55mm, and F=106 N.

4.7.6.2 Vacuum Generator

Vacuum pressure must lay between **0.5** : **0.75** kPa for non-porous material so we Choose JXPC : VEV -15 HS (or any other part that produce vacuum pressure as specified) to operate at this pressure a compressed air supply with pressure about 6 bar and air flow of 100 L/ min must be provided.

4.7.6.3 Air compressor

P_{max} = **8** bar, Tank Capacity >= **15** L, Motor Power = **1** hp

So, any compressor with these specifications is sufficient.

4.7.6.4 Solenoid Valve

Lastly the solenoid valve we need is the most basic 3-way 2 position as we aim to put one position on suction and the other on-air release so that the object fall due to the gravity force and absence of the suction lifting force. So, any 3-way 2 position solenoid valve is accepted.