Project Report

Modeling and Simulation of a 3 DOF Delta Robot

By:
Aditya Khopkar | 116911627

Project Partner:
Sukoon Sarin | 116955522

Instructor:
Prof. Chad Kessens

Index

1. Introduction

The complexity of industrial robot manipulators has grown manifold over the last few years, from Cartesian manipulators to Articulated manipulators to Parallel Manipulators. Each type of manipulator functions accurately for the required task and payload requirements. While Serial Manipulators are most commonly used for pick & place operations of heavy objects, Parallel manipulators are majorly used for lighter objects. While serial manipulators constitute of links configured serially through the joints responsible for actuation, parallel manipulators have two parallel frames - a fixed frame and an end effector frames which are connected through parallel links and actuated by joints connecting the fixed frame and these links. In a broader sense, parallel manipulators have less or equal number of joints compared to serial manipulators, which facilitate enough degrees of freedom for appropriate functioning. There are two main types of parallel manipulators- a. Stewart Platform and b. Delta Robot.

In this project, we intend to perform the modeling of a 'Delta Robot', and subsequently show how it can be used to perform a pick and place application. Delta Robot is highly used in industrial applications and tasks, such as pick-and-place, high precision operations, etc. The Delta Robot consists of a fixed frame and end effector frame with parallel links joining the two triangular frames as shown in figure 1. The robot can be actuated to move along all three axes *XYZ*. Thus, conforming to three degrees of freedom. The Delta Robot is preferred in industrial applications which require fast action service majorly because, unlike the joint complexity in a serial manipulator, the parallel manipulator has just three driving joints, enabling faster computations. Also, unlike in serial manipulators, where the actuators are placed at all the joint positions, the delta robot has all the actuation units, often referred to as the driving joints located at a fixed location, generally the fixed frame [fig. 1]. This enables the end-effector to be almost massless and facilitate faster service.

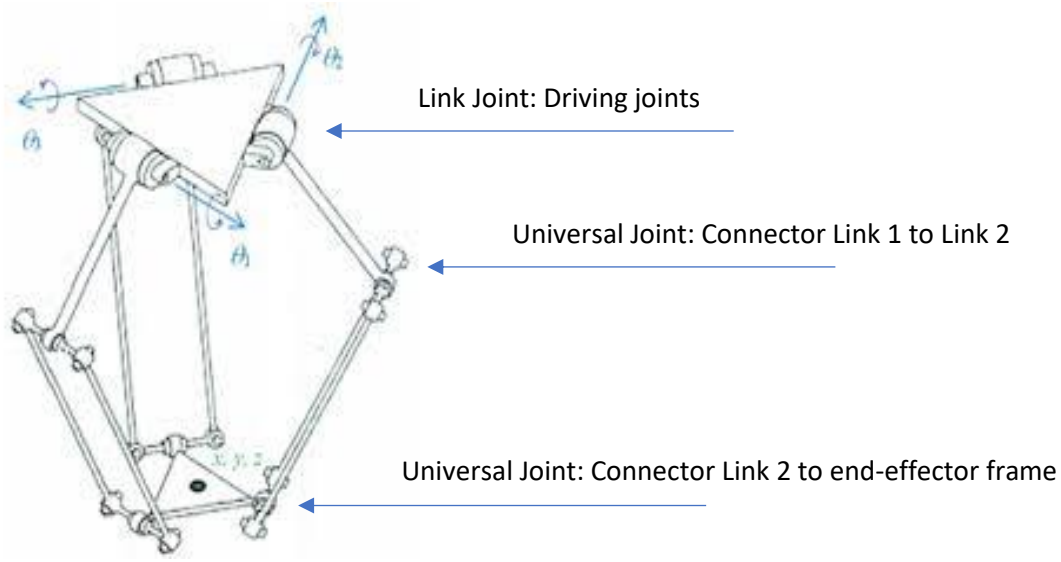

Figure 1:Delta Robot Diagram

The universal joint is responsible in establishing a free movement between the two links it connects [10]. Due to this structure, the two frames - the fixed frame and the end effector frame is always kept parallel to each other.

1.1 Aim & Objective

Through this project report, we learn to realize the models of the Delta Robot and subsequently understand how a Delta Robot can be used in Industrial space for providing a fast action service such as sorting the objects and/or packages in the assembly line. The proposed task of the robot is to pick the elements/packages coming on the conveyor belt and place them over another conveyor belt which would carry these elements/packages to a different destination. Thus, through this project, we wish to establish an intuitive understanding of modeling and subsequent simulation of a Delta robot using Forward Kinematics, Inverse Kinematics, Trajectory Planning. The project also discusses the theoretical model of grasping which can be used for further analysis of the task.

1.2 Organization

The project report first describes the robot specifications which is a necessary guide towards understanding the robot parameters while modeling the robot along with some assumptions taken into consideration. A detailed explanation of the mathematical model of FK, IK and Grasping will be provided in section 5. An analytical approach towards modeling the robot is discussed in subsequent sections (Section 6), which aims at providing a holistic understanding of implementing the robot model and thus includes, implementation of FK, IK, Trajectory Planning, Workspace definition. The report discusses a method to validate some of the models effectively and simulate the models on MATLAB and later simulate on a simulation platform such as RoboDK. The simulation thus, is an analytical validation of the mathematical models.

2. Motivation

The advantage of a Delta Robot manipulator is its high speed and thus finds its application in the fields of packaging, medical industry, 3D printing, and even surgery. The Delta Robot is adept to sustain a high-on-delivery requirement of the industry and succeeds in providing easy and fast performance for the task requirement. A multitude of industries are replacing the standard articulated robotic manipulators with the delta manipulators. A Delta Robot exhibits versatility which ranges from performing tasks such as stacking the pancakes in a packaging industry, to playing board games as a demonstrative activity. The robot proves itself in performing each task with utmost efficiency and grace and is nothing short of ‘a piece of art’. This instance was enough to instill that fervor and curiosity in us to explore this type of manipulator which is not much explored or focused on through the scope of this course. Whenever we interacted with peers and people about this manipulator, almost all of them advised me to avoid this topic, eventually, fueling the curiosity about the manipulator even more. This drive to prove them wrong, motivated us to opt for ‘Delta Robot Modeling’. The robot model implemented in this project derives its motivation from ABB IRB 360, which is a 3-DOF Delta Robot by ABB, which is known to have Shortest cycle times, precision accuracy, and high payload. Details about the same will be briefly explained in the following section.

3. Model Specifications

3.1 Defining the Model

A Delta Robot with three sets of parallel arms will be modelled (fig. 2a). In the figure 2a, the green triangle represents the fixed frame of the robot and the pink triangle represents the end effector frame. Robot simulation is done on the model of ABB IRB 360 on RoboDK. The robot is known for having high accuracy

and precision in the operations of pick and place. The mathematical modeling and supporting simulation of the robot will be done in MATLAB.

3.2 Degrees of freedom

Degrees of freedom define the number of possible motion or states that the robot can freely achieve. In the scope of this project, the Delta Robot is capable of translational motion along all three axes. Thus, the degree of freedom of a Delta Robot is 3 (fig. 2b). Some manipulators have an additional actuating unit located at the fixed frame which is responsible in the rotation of the end effector about the z-axis.

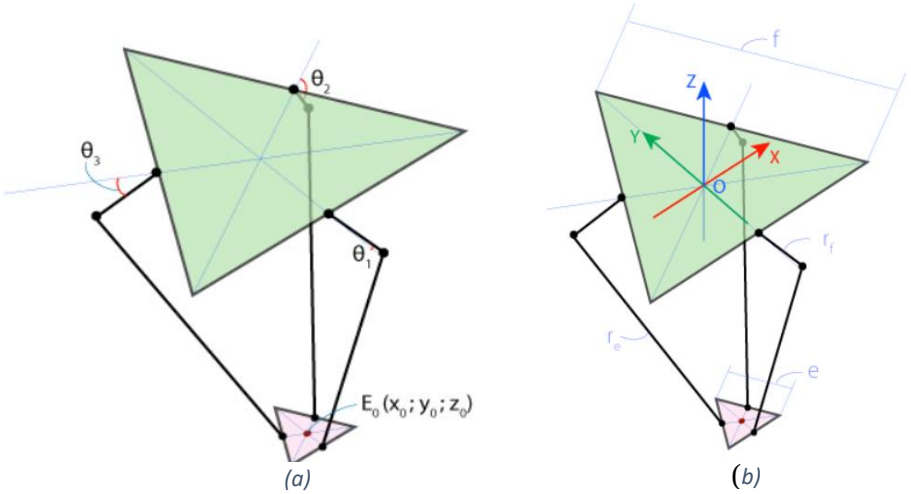

Figure 2: (a) Delta Robot geometric diagram, (b) Delta Robot frames

3.3 ABB IRB 360 Specifications

The Delta Robot Manipulator in inefficient for higher payloads and thus, throughout the scope of this project, the payload capacity is restricted to less than 5kg. ABB IRB 360 has a payload capacity ranging from 1kg, 2kg, 5kg to 8kg and working ranges for IRB 360/1kg and IRB 360/3kg is: Diameter: 1130 mm (967 mm), Height: 250 mm (300 mm). Maximum operating speed of the robot is almost 10 m/s.

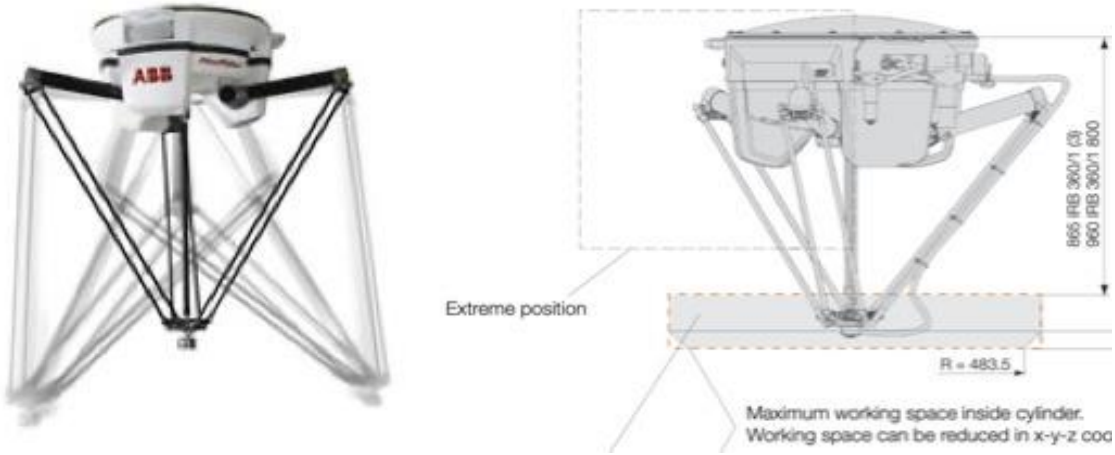

Figure 3:ABB IRB 360 Robot and workspace [6]

4. Model Assumptions

The geometric and reference frame consideration of the delta robot throughout the course of this project is considered as the one described in figure (2b).
In this project, we take few assumptions into consideration:

a) The dimensions of the system are known and all the links are considered massless.
b) All links and the frames taken into consideration are rigid, uniform bodies.
c) The package/element has uniform surface i.e., uniform coefficient of friction.
d) All the forces, torques on the joints are ideal - and are not considered for the modeling.
e) The velocity analysis of the joints is not considered in the scope of this project.
f) Object/Package poses are exactly known.
g) The pick and place positions are assumed to be in the workspace of the robot.
h) The angular deflection of the driving joints estimated, vary as follows: θ_1 = -35 to 95, θ_2 = -45 to 95, θ_3 = -45 to 95 (in deg.), due to geometry of ABB IRB360 and parameters chosen [8].
i) Initial position of the robot is when all the joint angles is 0. i.e., horizontal to the surface. Before the simulation, the robot is always in its initial position.
j) The origin of the frame coincides with the geometrical centers of the two equilateral triangles (fig. 2(b)).
k) The edge of the end-effector is always less than the edge of the fixed frame.

5. Theoretical Approach

This section describes the theoretical and mathematical approach taken towards modeling the robot. The robot has a fixed frame and a moving end effector [fig. 2]. There are three parallel arms connecting the two frames, forming a parallelogram, thus ensuring that the two frames remain parallel to each other always. Each arm makes an angle with the base frame ($\theta_1, \theta_2, \theta_3$). The following nomenclature is maintained throughout the project [fig. 2(b)]

f = The edge of the fixed frame equilateral triangle (green).

e= The edge of the end effector equilateral triangle (pink).

rf= The link of the fixed frame (the short link).

re = The link of the end frame (the long link).

F = Fixed frame nomenclature.

E = End effector frame nomenclature

J = Driving joint nomenclature

In this section, geometrical approach is adapted to mathematically model the robot. The major advantage of the geometrical approach is that, the constraints of the robot is taken care of, as we will see in the section ahead. The position of the fixed frame center is (0,0,0) and subsequently, the end effector position (initial) is (0,0, -z). The negative sign indicates that the end effector is always beneath the fixed frame.

Mathematical Modeling is responsible in driving the project as the implemented modeling techniques give a structured approach towards the latter part of the project. In parallel manipulator, the mathematical techniques for FK is more complex than the techniques used for IK.

5.1 Inverse Kinematics Modeling

Inverse Kinematics is responsible in getting the joint variables of the robot based on the end-effector pose of the robot. Having defined some key nomenclature which will be extensively used to understand the Inverse Kinematics modeling, we define the sphere of rotation of each joint with center at the end-effector frame.

Let's consider joint 1 (fig. 4), E_1 denotes the end-effector offset, which is the universal joint (refer Glossary), connecting the end-effector link (given by, *re*) with the end-effector frame (*E*), and J_1 denotes the driving joint 1. Thus, we get a sphere of rotation of the joint with radius E_1J_1 *(= re)*.
Since, we are considering just one of the joints, the problem can be reduced into a planar problem considering the plane of rotation in the *YZ* plane - resulted by a circle with radius E'_1J_1, where, E'_1 is the projection of E_1 on the *YZ* plane.

However, the joint J_1, is also connected to the fixed frame by the universal joint F_1, which produces a circle in the *YZ* plane of radius F_1J_1 *(= rf)* [fig. 5]. Since, the two circles which define the plane of rotation of the joint now lie in the same plane, the joint can thus be found on the intersection of the two circles (with radius *rf* and E'_1J_1). Thus, the problem is now reduced to planar geometry, and the value of θ_1 can thus be derived. The end effector position is given by $E_0\,(x_0, y_0, z_0)$.

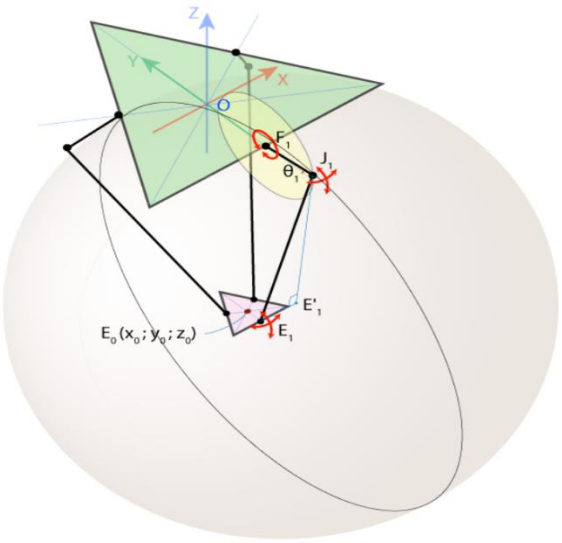

Figure 4: Sphere of Joint

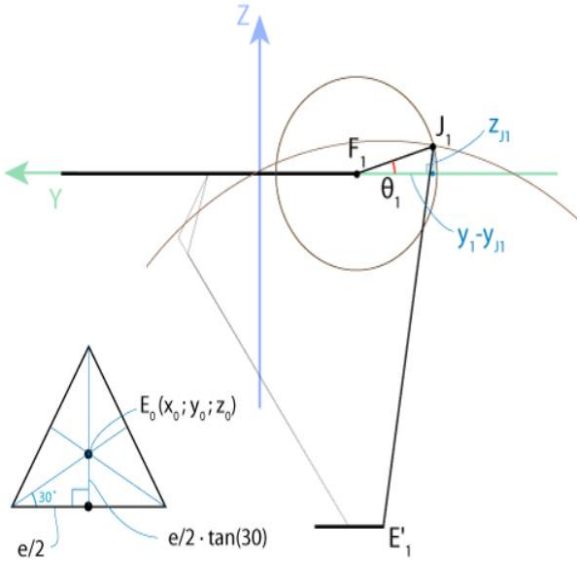

Figure 5: Planar representation of Joint motion

We now have to find out the length E'_1J_1. By geometry of the triangle, E_0E_1= *(e/2) * tan (30).*
J_1 = *(0, y, z)* and F_1 = *(0, y_1, 0)*

$$E_0 = (x_0, y_0, z_0)$$
$$E_1 = (x_0, y_0 - (e/2)\ tan30, z_0)$$
$$E'_1 = (0, y_0 - (e/2)\ tan30, z_0)$$

By applying Pythagoras theorem in triangle $E_1E'_1J_1$ we get,

$$E'_1J_1 = \sqrt{re^2 - x_0^2}$$

$$F_1 = (0, -(f/2)\ tan30, 0)$$

Thus, writing the equations of circle,

$$(y - y_1)^2 + (z)^2 = rf^2 \tag{1}$$

$$(y - y_0 - (e/2)\ tan30)^2 + (z - z_0)^2 = re^2 - {x_0}^2 \tag{2}$$

Eq. (1) refers to the equation of the circle with center as F_1 and Eq. (2) refers to the equation of the circle with center as E_1. Solving these equations, We can now, find the *y* and the *z* coordinate of the joint J_1. We select the minimum *y*. Referring to figure 5, the position of J_1 is on the negative y-axis. By selecting a value which more negative (minimum *y*), we take care of the angular joint constraint of the robot. We can thus obtain the joint variable θ_1 = arctan(z/y_1-y)

Similarly, we repeat these steps for all three joint variables θ, by rotating the base frame by 120 degrees to align with the next joint. A rotation of 120 degrees is needed due to the geometry of the equilateral triangle, so that the axes align along the driving joints.

5.2 Forward Kinematics Modeling

Forward Kinematics Modeling for the Parallel Manipulator is generally more complex than the IK modeling. FK modeling, like IK modeling is intuitively solved by geometry too. In this, we input the joint variables and expect the final coordinates of the end-effector. As we know the joint angles theta, we can easily find coordinates of points J_1, J_2 and J_3. Considering just a single end-effector joints *E* and driving joints *J*, we can analyze three spheres of motion with the driving joint *J* as the center, similar to the procedure followed in figure 4, just keeping the driving joints as the center. Thus, joints E_1J_1, E_2J_2 and E_3J_3 can freely rotate around points J_1, J_2 and J_3 respectively, forming three spheres with radius *re*, as explained above. Since we are interested in finding the end-effector position, we translate the driving joint positions to J'_1, J'_2 and J'_3, such that, $J_1J'_1 = E_0E_1$, $J_2J'_2 = E_0E_2$, and $J_3J'_3 = E_0E_3$. We come to this point by exploiting the property of a parallelogram [fig. 6]. $J_1J'_1E_0E_1$ forms a parallelogram, and so do other joint geometries.

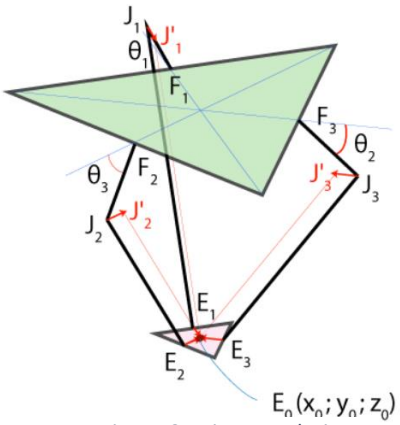

Figure 6: Joint Translation

Thus, now the spheres pass through the end-effector position. And hence, intuitively, solving for end-effector merely becomes solving the equations of spheres to find an intersection point, which is indeed the end-effector position.

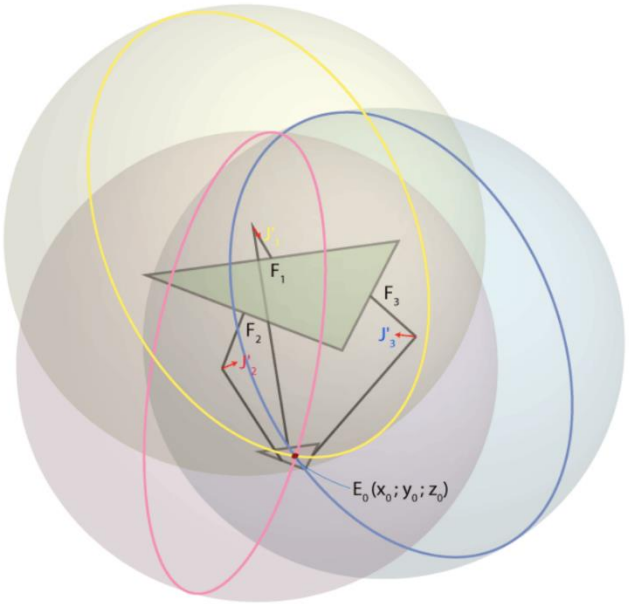

Figure 7: Intersection of spheres

The FK problem is now reduced to algebra and geometry. We now need to find the coordinates of J'_1, J'_2 and J'_3 -

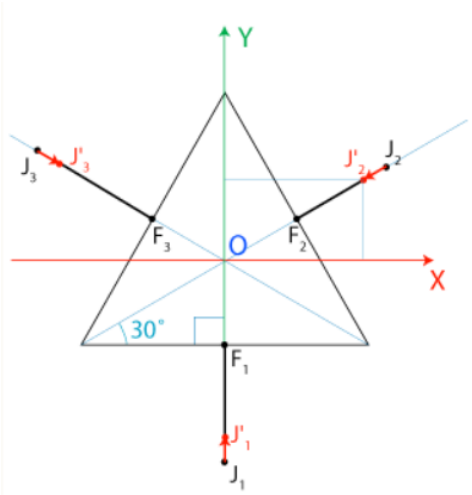

Figure 8: Fixed frame geometry to compute joint positions

F_1, F_2, F_3 are the joints (universal) that link the arm to the fixed frame. O is the center of the frame.

Thus,

$OF_1 = OF_2 = OF_3 = f/2 * \tan(30)$
$J_1 J'_1 = J_2 J'_2 = J_3 J'_3 = e/2 * \tan(30)$
$F_1J_1 = rf\cos(\theta_1) = F_1J_1 = rf\cos(\theta_2) = F_1J_1 = rf\cos(\theta_3)$

Thus, we can derive $OJ'_1 = OF_1 + F_1J_1 - J_1J'_1$. And similarly, we get the coordinates for OJ'_2 and OJ'_3 We can easily equate these lengths because of the parallelogram formed by the arms of the manipulator.
Thus,

$$J'_1 = \left(0, \frac{-(f-e)}{2}\tan 30 - rf\cos\theta_1, -rf\sin\theta_1\right)$$

$$J'_2 = \left(\left(\frac{(f-e)}{2}\tan 30 + rf\cos\theta_2\right)\cos 30, \left(\frac{(f-e)}{2}\tan 30 + rf\cos\theta_2\right)\sin 30, -rf\sin\theta_2\right)$$

$$J'_3 = \left(-\left(\frac{(f-e)}{2}\tan 30 + rf\cos\theta_3\right)\cos 30, \left(\frac{(f-e)}{2}\tan 30 + rf\cos\theta_3\right)\sin 30, -rf\sin\theta_3\right)$$

We can now thus find the end-effector coordinates, by designating the points of J'_1 (x_1, y_1, z_1), J'_2 (x_2, y_2, z_2), J'_3 (x_3, y_3, z_3). Note that, x1 = 0.
We get the equations of the spheres as follows:

$$x^2 + (y - y_1)^2 + (z - z_1)^2 = re^2 \quad (3)$$

$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = re^2 \quad (4)$$

$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = re^2 \quad (5)$$

We follow the steps shown in the following figure to solve the geometry equation, algebraically:

$$w_i = x_i^2 + y_i^2 + z_i^2$$

$$\begin{cases} x_2 x + (y_1 - y_2)y + (z_1 - z_2)z = (w_1 - w_2)/2 & (4) = (1) - (2) \\ x_3 x + (y_1 - y_3)y + (z_1 - z_3)z = (w_1 - w_3)/2 & (5) = (1) - (3) \\ (x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = (w_2 - w_3)/2 & (6) = (2) - (3) \end{cases}$$

From (4)-(5):

$$x = a_1 z + b_1 \quad (7) \qquad y = a_2 z + b_2 \quad (8)$$

$$a_1 = \frac{1}{d}\left[(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)\right] \qquad a_2 = -\frac{1}{d}\left[(z_2 - z_1)x_3 - (z_3 - z_1)x_2\right]$$

$$b_1 = -\frac{1}{2d}\left[(w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1)\right] \qquad b_2 = \frac{1}{2d}\left[(w_2 - w_1)x_3 - (w_3 - w_1)x_2\right]$$

$$d = (y_2 - y_1)x_3 - (y_3 - y_1)x_2$$

Figure of equation: Forward Kinematics solution equation [3]

By plugging (7) and (8) from the figure of equation in (3) derived above, we get a quadratic in z as follows:

$$(a_1^2 + a_1^2 + 1)z^2 + 2(a_1 b_1 + a_2(b_2 - y_1) - z_1)z + (b_1^2 + (b_2 - y_1)^2 + z_1^2 - re^2) = 0$$

We solve these quadratic equations simultaneously to find the values of z0. We must choose the smallest negative equation root, and then calculate x0, y0 through these equations of circle.

5.3 Grasping Modeling

Robot manipulators use contact forces to grasp and manipulate objects in their environment. Fixtures rely on contacts to immobilize workpieces. Modeling of the contact interface, therefore, is fundamental to analysis, design, planning, and control of many robotic tasks. Different kinds of forces are available for any given contact.

Here we plan to use the coulomb friction model which is a type of friction point contact model [5].

$$f^t \leq \mu f^n$$

f^n: is set of forces in normal direction

f^t: is forces in tangential direction

μ: is the coefficient of friction between the contact surfaces (the hand gripper and the package surface)

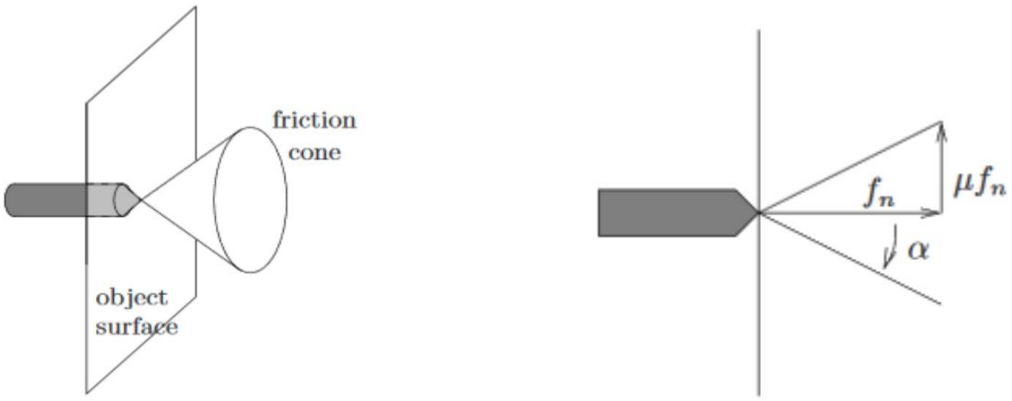

Figure 9: Cone of Friction [5]

The normal force indicates the contact force applied on the object and constitutes the internal force, which is responsible in gripping the object, whereas the tangential force is responsible in moving the object. The relation between the two can be intuitively realized better from the concept of a friction cone (fig. 8), where, α is the angle of repose. (Refer Glossary). The wrench is 6x1 matrix defining the spatial forces and torques applied to the object at a point of contact can be defined as, F_{ci} (i = 1,2: Contact points)

$F_{ci} = B_{ci} * f_{ci}$

We consider the contact basis, B_{ci} to be [1 0 0; 0 1 0; 0 0 1; 0 0 0; 0 0 0; 0 0 0] as we aren't considering torques and only normal and tangential forces (Hard Finger model).

Therefore, $F_{ci} = B_{ci} * f_{ci}$ = [1 0 0; 0 1 0; 0 0 1; 0 0 0; 0 0 0; 0 0 0] * f_{ci}

Here, we consider $f_{ci} \in FC_{ci}$ which is the Force Constraint

$$FC_{ci} = \{f_{ci} \in R^3 : \sqrt{{f_{xci}}^2 + {f_{yci}}^2} \leq \mu * f_{zci}, f_{zci} \geq 0 \}$$

[1 0 0; 0 1 0; 0 0 1; 0 0 0; 0 0 0; 0 0 0] * [f_{xci} f_{yci} f_{zci} 0 0 0]

$$\sqrt{{f_{xci}}^2 + {f_{yci}}^2} \leq \mu * f_{zci}$$

And the Friction Cone is given by: $FC = FC_{c1} * FC_{c2}$

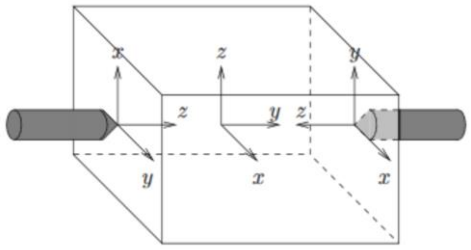

Figure 10: Orientation of the frames [5]

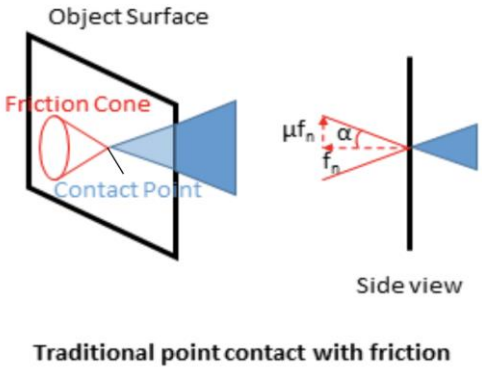

Figure 11: Realizing Friction Cone [5]

After we calculate the contact forces on the two contact points, we calculate the map between the contact forces and total object force which is the Grasp map, for which we will require the position and orientation of the contact frame from the perspective of the object frame (COM) (refer fig. 9):

Let, Position matrix of the point of contact be p_{oci}

Rotation matrix be R_{oci}

The contact map needed to define the Grasp map will require us to define the force exerted by a single contact on the surface of our object as:

$F_0 = \sum_{i=1}^{n} Ad^T_{goci^{-1}} * B_{ci} * f_{ci} = [R_{oci}\ 0;\ \widehat{p_{oci}} * R_{oci}\ 0] * (B_{ci} * f_{ci})$

Where, $Ad^T_{goci^{-1}}$ is the wrench transformation matrix that maps the contact wrenches (forces) to the object (package) wrenches

So finally, the Contact Map will be defined as:

$G_i \equiv Ad^T_{goci^{-1}} * B_{ci}$

From this we can add, all of the contact forces (expressed in the object frame), to create the total object wrench:

$F_0 = (G_1 * f_{c1}) + (G_2 * f_{c2})$

Finally, we'll have with us the Grasp map, defined as:

G = [$(Ad^{T}_{goc1\ -1} * B_{c1})$ $(Ad^{T}_{goc2\ -1} * B_{c2})$]

6. Analytical Approach

This section describes the analytical approach to implement the mathematical models discussed in the previous section and achieve simulation of the robot. Coding of the project is done mainly in MATLAB. Coding is just a tool to realize the theoretical concepts and subsequently realize the analysis of the models. Implementation of both IK and FK follows the theoretical approach. The advantage of a geometrical solution towards solving for FK and IK is that the constraints of the robots are addressed. When we find the intersection of two shapes (circles or spheres), we are, as a matter of fact, subjecting the robot to geometric constraints.

Inverse kinematics is majorly used to actuate the robot to attain the desired end-effector goal position. The Inverse Kinematics code function takes the position vector as the input along with the robot parameters and defines the joint variables. Whereas, the Forward Kinematics can be used to realize the holistic workspace of the end-effector, and the code function inputs the joint variables vector and the robot parameter and returns the end-effector position vector. These two functions also return a flag to realize the status of the program (refer Appendix). The parameters taken into consideration in this project are: *re = 1.24, rf = 0.52, e = 0.07, f =0.57 (in m),* These are the dimensions of an ABB IRB360 robot [8]**.** The user may input the parameters of his/her choice.

6.1 Defining the workspace

Defining the workspace of the robot is a crucial part of realizing the robot model. The workspace intuitively suggests the positions that the end-effector can achieve in space. Similar to when dealing with a serial manipulator, we geometrically find all the possible points the end-effector pose can achieve in the given plan and space, a similar approach is used while defining the workspace of the parallel manipulator.

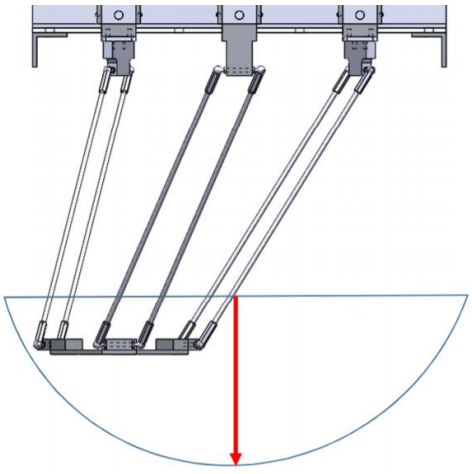

Figure 12: Defining the workspace of Delta Robot [7]

We need to realize the condition of singularity to define the limits of the workspace. Each joint motion is constrained by the other joint motion. For example, if all the joints assume an angular pose of -90° (refer Fig. 2), due to the property of parallelograms, the end-effector edge must be equal to the fixed frame edge, which is not possible and forms a singularity. Similarly, if all the joints assume an angular pose of 90°, same problem of singularity arise. Thus, the joints cannot assume the same angular pose. This becomes instrumental in understanding the extent of z-direction deflection of the end-effector (shown by a red-line in fig. 11).

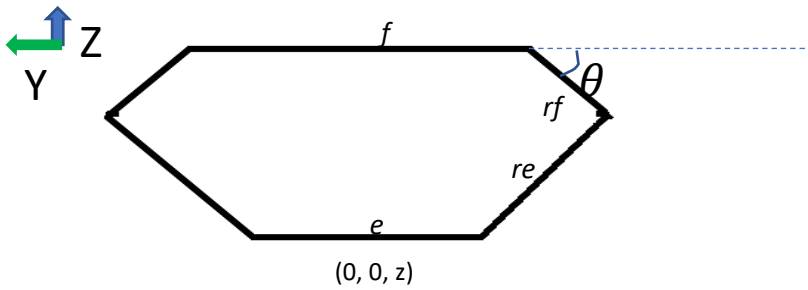

Figure 13: Cross section about YZ for realizing z deflection

By using the properties of trapezoids and parallelograms, the geometry of figure 12 can be solved to obtain the z deflection of the end-effector (The position of the end-effector indicates the initial frame position). The z-deflection is given as:

$$z = -\left[rf\sin(\theta) + \sqrt{re^2 - \left[rf\cos(\theta) + \frac{(f-e)}{2}\right]^2}\right]$$

And as discussed above, θ cannot take the value of -90/90. Similarly, we can find deflections for all the planes (*XY, XZ*). However, this approach is tedious and may not result in appropriate values.

Since, our mathematical models deal with geometrical approaches, the mathematical model of FK can be used to govern the end-effector pose workspace. In this approach, we vary each joint angle according to assumption (h). One can also set the angle variation from -180 to 180, however, this results in a much heavier computation. Thus, it is advisable to stick by the delta robot model specification. When using the FK code, we return a flag which is set to 0 if the solution for the given angle exists and returns 1 if the

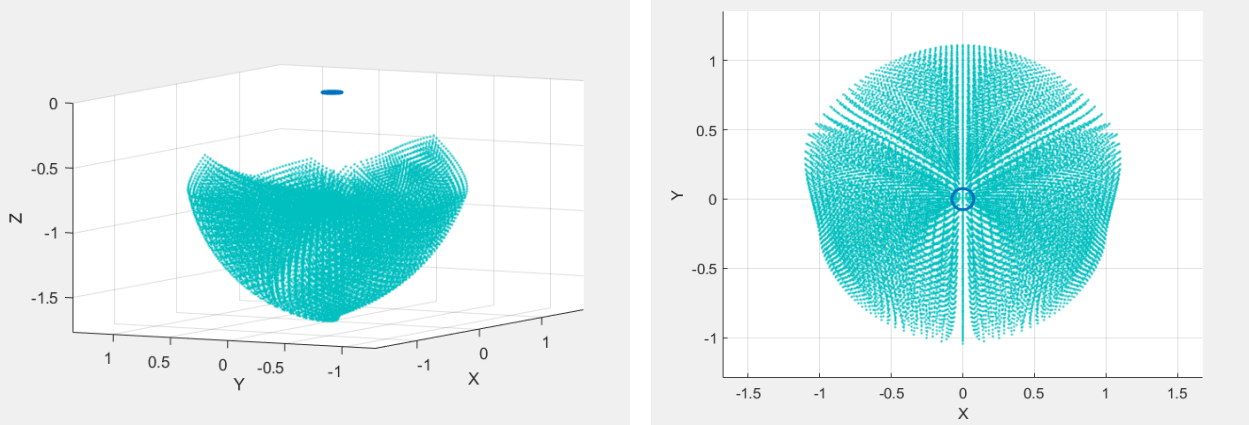

Figure 14: Workspace in MATLAB:(a): 3D workspace (b) XY workspace

solution doesn't exist. This logic is used effectively to return just those points for which the solution exists and results in the workspace of the robot (refer Appendix A).

In figure 13, the blue circular line denotes the fixed frame location figuratively. The workspace may also be used to validate the IK and FK model. The end-effector pose must always lie within the workspace.

6.2 Trajectory Planning

The Delta Robot majorly consists of three driving joints which facilitate a 3DOF robot. The trajectory planning for a delta robot is thus quite easy. The model uses geometry in plane for planning the motion of the end-effector. One may effectively allow the robot to follow the trajectory they wish to, but since the Delta robot taps into the feature of high speed and precision-based performance, the optimum trajectory for the end-effector is linear. In this project, the robot follows a linear path to reach from one position to another. A code to calculate trajectory angles for linear variance of the end-effector position through the workspace achieves this task quite intuitively (refer Appendix for code). Trajectory planning is an important aspect in understanding the motion of the robot based on the positions we wish the robot to achieve. Trajectory planning also enables us in understanding how the model of IK mentioned previously works in the simulation to achieve the task.

The workspace analysis and trajectory planning both thus help us realize the mathematical models analytically (in simulation) to understand the functioning of a delta robot, which will be explored in the following section.

7. Simulation and Validation

This section gives an overview of how the simulation in MATLAB was achieved and subsequently in RoboDK simulation platform. Before we get into the simulation, we need to validate the models we thus described above.

7.1 Validation

There are two ways of validating an FK and IK model:

a) Plugging method: In this method, we take a test case i.e., a random set of position vector and pass them through the IK model. The IK function returns a vector of joint variable ($\theta = [\theta_1, \theta_2, \theta_3]$). This vector is passed through the FK function. If the FK function returns the same position vector we chose, the models work absolutely fine.

b) Workspace method: Sometimes, the plugging method may not be the optimum way to understand the valid positions. We can also check the validity of FK by defining the workspace of the end-effector as shown in figure 13. The points present within the workspace can be used to validate IK and subsequently, FK.

```
%TEST
param;
pos = [-1,1,-8];
T =  inverseKinematics(pos,param);
pos_out = forwardKinematics(T,param);
disp(pos_out);
```

```
pos_ =

    -1     1    -8


T =

   -5.8679   12.9515   -0.8514


Pos_out =

[ -1.059934032055576419266682130074, 1.040585349261484542932957638962, -8.348682794091876343386366111698]
```

Figure 15: Model validity test in MATLAB

7.2 Simulation and Results

After validating the models implemented, we move to validating its operation in simulation. In this stage, the simulation will be done on MATLAB. The MATLAB code will consist of all the functions described above and the simulation will be done for known vectors, taken into consideration from the workspace of the robot i.e., The goal position will be feed into the MATLAB code. The goal position vectors are as follows:
Goal (1, :)= [0, 0, -1.116]: = Initial position of the end-effector
Goal (2, :)= [-0.06558, -0.05222, -1.339]: = First goal (object pick-up)
Goal (3, :)= [0, 0, -1.116]: = Initial position
Goal (4, :)= [0.2315, -0.4401, -1.258]: = Second goal (object drop)
Goal (5, :)= [0, 0, -1.116]: = Initial position.

The robot is thus tasked to pick up the object traverse through the initial position all the way to drop point. All these points are within the workspace and can be validated by verifying the presence of these points in the workspace graph (fig. 13).

In order to completely realize simulation, we first define the workflow of the project. The codes for the same could be referred in the Appendix section.

7.2.1 Workflow

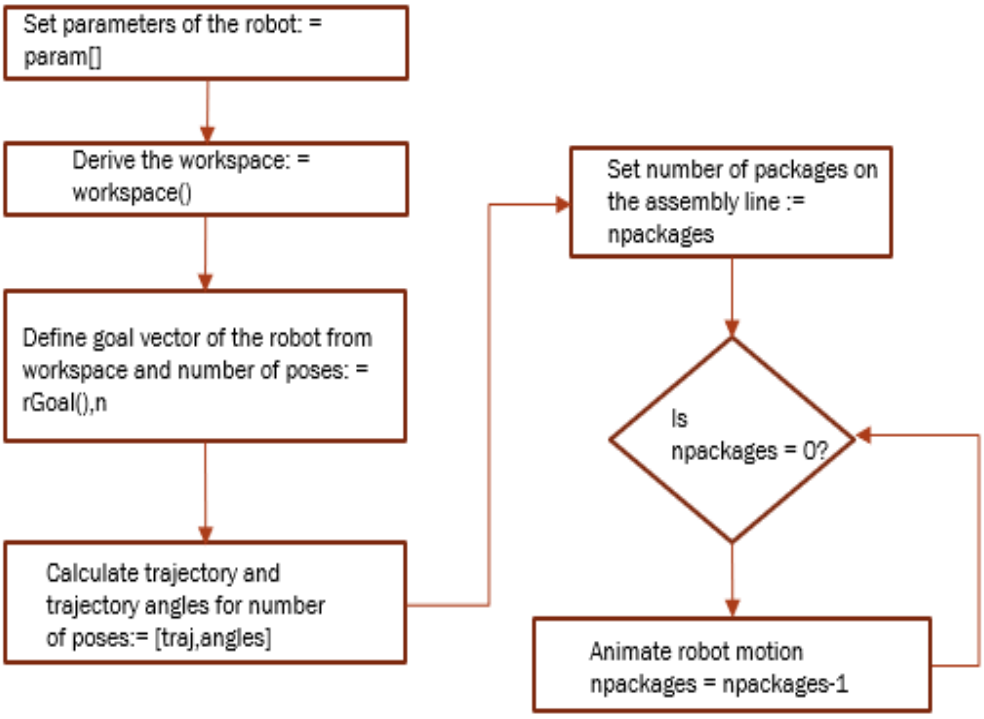

Figure 16: Workflow of the project

The FK and IK implementation are as described in section 5 (Appendix A). The workspace of the robot is derived by invoking the FK function call as described in section 6.1 (Appendix B). Trajectory of the robot is calculated by invoking the trajectory function. The trajectory is tracked linearly, i.e., the trajectory of the end effector is along a straight line (section 6.2) and for all the positions from the start position to the goal position lying on the straight line, returns the trajectory angles by invoking IK function call (Appendix B). To animate the robot in MATLAB, we plot the robot positions iteratively till the goal position is reached (Appendix C).

7.2.2 Results

Referring to the code in the Appendix of this report, the desired results of the simulation could be observed by running the live script of the 'main.m' file in MATLAB (Appendix D).

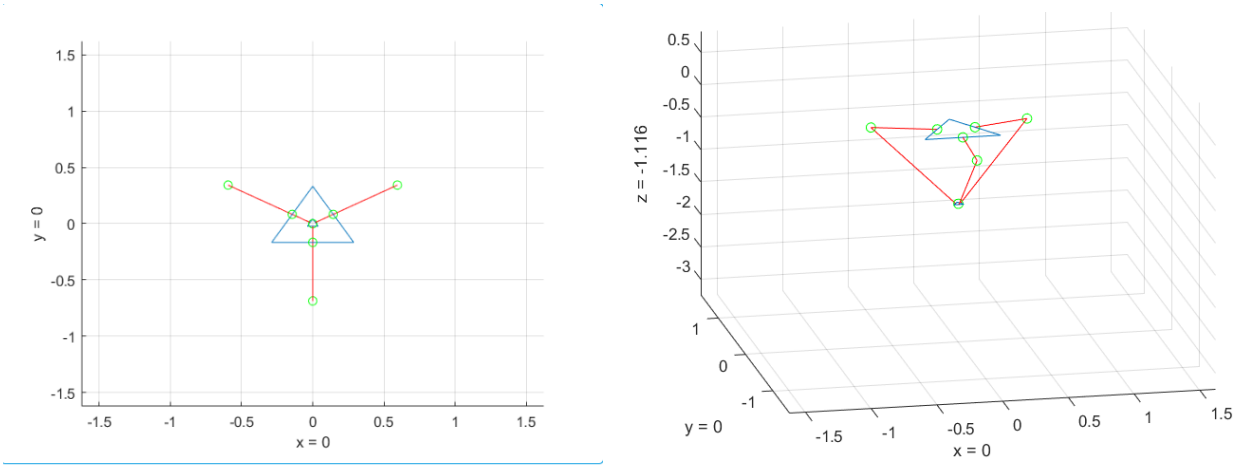

Figure 17: Loading robot in the workspace at Initial Condition

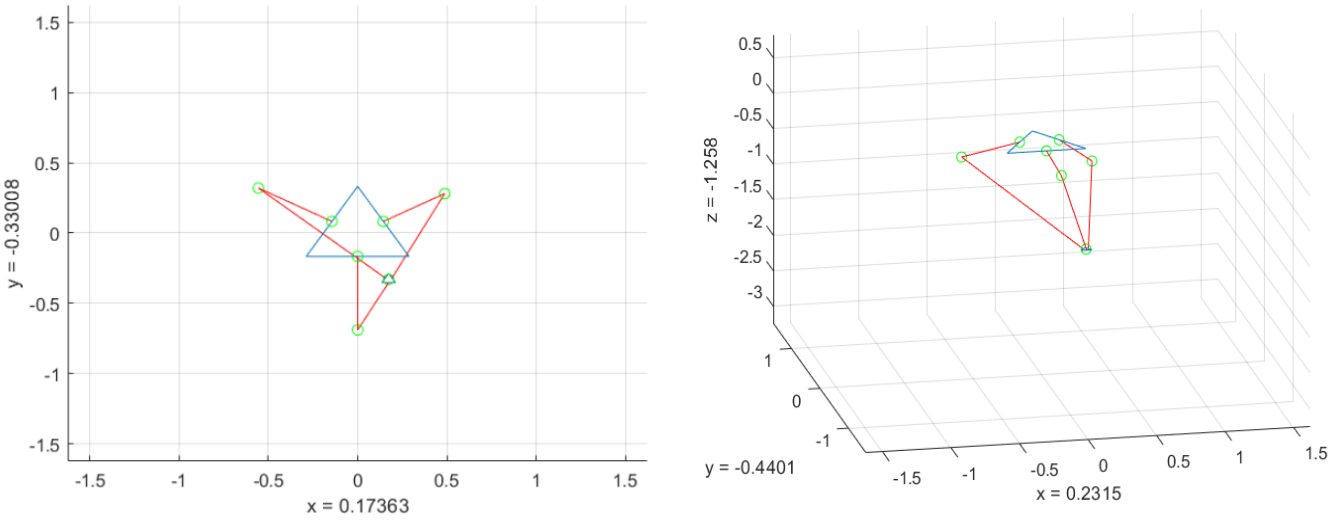

Figure 18: Graphical simulation in MATLAB suggesting motion in Robot

This simulation thus helps in understanding and realizing the working of the delta robot and thus, the delta robot is modelled. The simulation is also carried out in the simulation software such as, RoboDK, which shows the implementation of the pick and place-sort operation of the robot in an environment consisting of a conveyor belt, carrying packages to be sorted. The model of ABB IRB360 is available in the RoboDK library along with other elements of the environment such as the conveyor belts, the package, etc. This platform enacts the grasping mechanism (demonstrated in section 5) with some inbuilt capabilities of the platform (fig. 18).

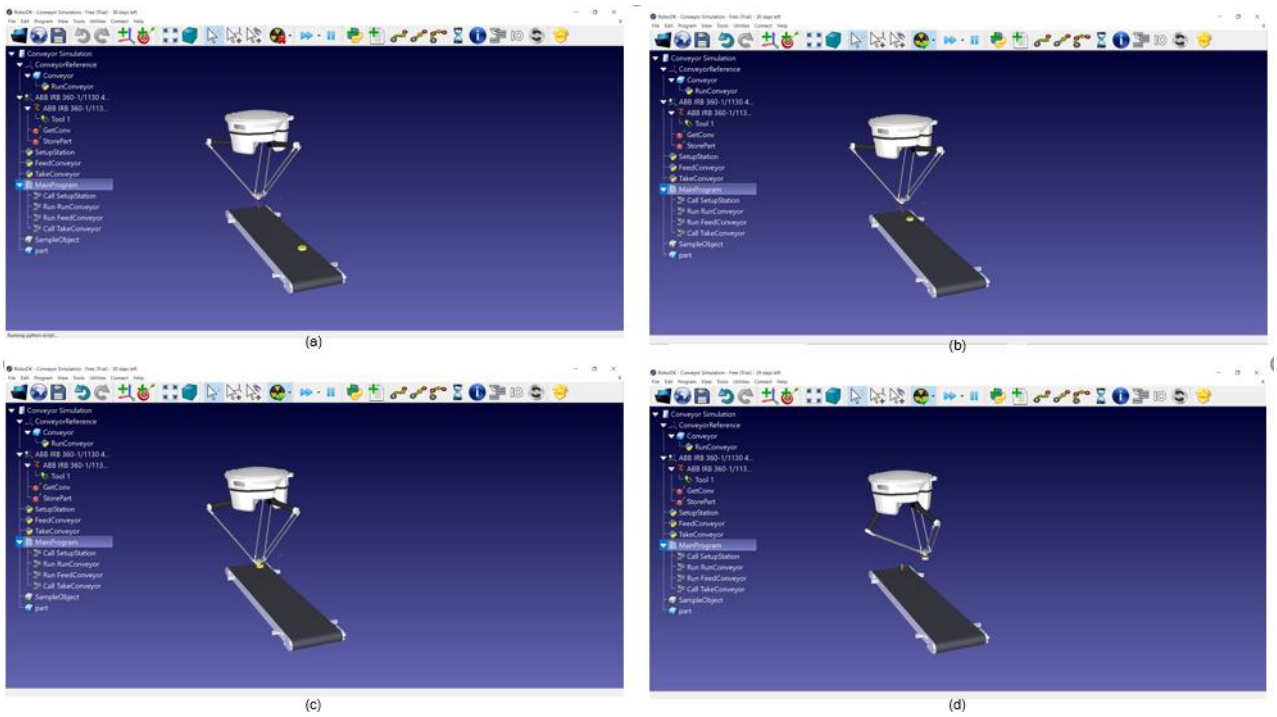

Figure 19: Screenshots of simulation in RoboDK. (a) Object loaded on conveyor, (b) Conveyor runs, (c) Robot Grasps object, (d) Moves the object to be sorted

8. Future scope of the project

While through this project, we have intuitively realized the working and functioning of the Delta Robot, the project can be reproduced in ROS as well. The URDF file of the Delta Robot can be imported in the ROS Gazebo platform. This enables one in developing a software modeling of the Delta Robot, which can be simulated in ROS Gazebo. In the ROS environment, the simulation could be projected to more industrially relevant parameters. Grasping could be analyzed and realized in great detail by using the SynGrasp toolbox [9] and integrate it with the current simulated model to holistically analyze the task of pick and place application, and subsequently, sort the objects.

9. Conclusion

In this project, we aimed at achieving the simulation of a Delta Robot by implementing all the mathematical models thus discussed and realized the analytical approach needed to bring this project to fruition. We apprehended how to define the workspace of the robot, which may also be used to validate the mathematical models such as, FK and IK, and give an intuitive understanding of the robot's reachable positions in space. Moreover, realizing how trajectory of a delta robot could be computed to facilitate the motion of the robot as per the set of goals defined was achieved. The simulation was reproduced on a simulation platform, RoboDK which was responsible in simulating the entire scene of the robot, by implementing the grasping of the object using in-built function in RoboDK.

10. References:

[1] P.J. Zsombor-Murray McGill, University Department of Mechanical Engineering Centre for Intelligent Machines, “Descriptive Geometric Kinematic Analysis of Clavel’s Delta Robot”, April,2004
[2] “The What, Why and How of Delta Robots” retrieved from https://www.engineering.com/AdvancedManufacturing/ArticleID/16651/The-What-Why-and-How-of-Delta-Robots.aspx
[3] “Delta robot kinematics” retrieved from http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/
[4] “Delta Robots” retrieved from en.wikipedia.org/wiki/Delta_robot
[5] Richard M. Murray, S.Shankar Sastry, Li Zexiang, “A Mathematical Introduction to Robotic Manipulation”, 1994
[6] ABB IRB360 FlexPicker® Datasheet retrieved from https://new.abb.com/products/robotics/industrial-robots/irb-360
[7] Michael Louis Pauly “Workspace Analysis of a Linear Delta Robot: Calculating the Inscribed Radius”, 2014
[8] Rodrigo Torres Arrazate “Development of a URDF file for simulation and programming of a delta robot using ROS”, Feb, 2017.
[9] SynGrasp Toolbox by The University of Siena retrieved from http://sirslab.dii.unisi.it/syngrasp/

Glossary

Universal Joint: A universal joint (universal coupling, U-joint, Cardan joint, Spicer or Hardy Spicer joint, or Hooke's joint) is a joint or coupling connecting rigid rods whose axes are inclined to each other, and is commonly used in shafts that transmit rotary motion. It consists of a pair of hinges located close together, oriented at 90° to each other, connected by a cross shaft.

Cone of Friction: The cone of friction is essentially used to describe various friction models. In case of a Coulomb model of friction, the friction cone gives a relation between the tangential forces and the normal forces on the object. By definition, the contact force lies within the cone, whereas, the tangential force lies tangential to the point of contact.

Angle of repose: The angle of repose suggests the deflection or the variance of the contact force within the cone of the friction. Effectively, the angle of repose is the angle made by the contact force and the normal at the point of contact.

Wrench: Defines the spatial forces and torques applied on the object and is a 6x1 matrix.

$$\begin{bmatrix} f_x \\ f_y \\ f_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} ; where, f\ is\ the\ force\ and\ \tau\ is\ torque$$

APPENDIX A

```
%%FK and IK MATLAB Script function calls
% Author: Aditya Khopkar, Sukoon Sarin

function [Pos_out,f] = forwardKinematics(T,param)
e = param(1); %end effector frame
f = param(2); %base frame
rf = param(4); %base arm
re = param(3); %parallelogram
sum = 0;
%defining geometry
t = (f-e)*tand(30)/2;
w = zeros(1,3);
theta1 = T(1);
theta2 = T(2);
theta3 = T(3);
j1 = [0, -(t+rf*cosd(theta1)), -rf*sind(theta1)];
j2 = [(t+rf*cosd(theta2))*cosd(30), (t+rf*cosd(theta2))*sind(30), -
rf*sind(theta2)];
j3 = [-(t+rf*cosd(theta3))*cosd(30), (t+rf*cosd(theta3))*sind(30), -
rf*sind(theta3)];
J1 = round(j1,3);
J2 = round(j2,3);
J3 = round(j3,3);
J = [J1;J2;J3];
for i = 1:3
    for j = 1:3
        sum = sum + J(i,j)^2;
    end
    w(i) = sum;
    sum = 0;
end

d = round(((J2(2)-J1(2))*J3(1)-(J3(2)-J1(2))*J2(1)),3);
a1 = round(((((J2(3)-J1(3))*(J3(2)-J1(2)))-((J3(3)-J1(3))*(J2(2)-
J1(2))))/d),3);
a2 = round((-(((J2(3)-J1(3))*(J3(1)))-((J3(3)-J1(3))*(J2(1))))/d),3);
b1 = round((-(((w(2)-w(1))*(J3(2)-J1(2)))-((w(3)-w(1))*(J2(2)-
J1(2))))/(2*d)),3);
b2 = round((-(((w(2)-w(1))*(J3(1)))-((w(3)-w(1))*(J2(1))))/(2*d)),3);

a = round(((a1^2)+(a2^2)+1),3);
b = round((2*((a1*b1)+a2*(b2-J1(2))-J1(3))),3);
c = round(((b1^2+((b2-J1(2))^2)+(J1(3)^2)-re^2)),3);
%solve geometry - algebraically
disc = (b^2) - 4*a*c;
if disc < 0
    %No solution!
    x0 = nan;
    y0 = nan;
    z0 = nan;
    f = 1;
```

```
else

    z0 = -0.5*(b + sqrt(disc)) / a; %get z
    x0 = -(a1*z0+b1); %find x
    y0 = a2*z0+b2; %find y
    f = 0;
end
    Pos_out = [x0,y0,z0]; %return vector
end
%%Inverse Kinematics function

function [Th, flag] = inverseKinematics(pos_,param)
T = zeros(1,3);
k = [0,120,-120];
for i = 1:3
    T(i) = jointParam(pos_,k(i),param);
end
if (isnan(T(1)) || isnan(T(2)) || isnan(T(3)))
    flag = 1;
    Th = zeros(1,3);
else
    flag = 0;
    Th = [T(1), T(2), T(3)];
end

end
%calculates joint variable
function [theta] = jointParam(pos_,k,param)
rot_Mat = [cosd(k), -sind(k), 0; sind(k), cosd(k), 0; 0, 0, 1];
%coord_in = [x0,y0,z0];
coord_param = rot_Mat*(pos_)';
e = param(1); %end effector
f = param(2); %base
rf = param(4); %base arm
re = param(3); %parallelogram
%Coordinates of the end effector
y = coord_param(2) - (e/(2*sqrt(3))); %shift center to edge
z = coord_param(3);
x = coord_param(1);
%projected point on yz plane
y1 = -0.5 * f / sqrt(3);  % - f/2 * tan(30)
a = (x^2 + y^2 + z^2 + rf^2 - re^2 - y1^2)/(2*z);
b = (y1-y)/(z);
%discriminant
d = -(a+b*y1)^2 + rf*(rf*b^2 + rf);
if d < 0
    theta = nan;
else

    yj = (y1 - a*b - sqrt(d))/(b^2 + 1); % choosing outer point
    zj = a + b*yj;
    theta = 180*atan(-zj/(y1 - yj))/pi;

    if yj>y1
        theta = theta + 180;
    end
end
```

```
end
```

APPENDIX B

```
%%Workspace
%Authors: Aditya Khopkar, Sukoon Sarin

function [] = workspace(param)
%Finding Workspace
wkspace = zeros((140/5)^3,3);
n = 1;
for t1 = -38.84:5:94.65
    for t2 = -46.18:5:95.87
        for t3 = -46.18:5:95.87
            T = [t1,t2,t3];
            [pos_out,f] = forwardKinematics(T,param);
            if f == 0
                    wkspace(n,1) = pos_out(1);
                    wkspace(n,2) = pos_out(2);
                    wkspace(n,3) = pos_out(3);
                    n = n+1;
            end

        end
    end
end
%Setting X,Y,Z vectors of workspace
X = wkspace(:,1);
Y = wkspace(:,2);
Z = wkspace(:,3);
disp(wkspace);
hold on
%Initiate time sequence
t=0:pi/180:2*pi;
x=param(1)*cos(t);
y=param(1)*sin(t);
plot(x,y,'Linewidth',2);
%Indicate plotting the point rotated about z-axis (plotting a circle)
plot3(X, Y, Z,'.','color', [0 0.75 0.75], 'MarkerSize',4)
xlabel('X');
ylabel('Y');
zlabel('Z');
grid on
rotate3d on
axis equal
hold off

min(X)
max(Y)
min(Z)
end

%%Calculating linear trajectory

function [ traj ] = CalcTrajectory( r0,rGoal,m )
```

```
traj=[linspace(r0(1),rGoal(1),m);...
        linspace(r0(2),rGoal(2),m);...
        linspace(r0(3),rGoal(3),m)];

end

%%Calculating trajectory angles

function [ angles ] = CalcTrajectoryAngles( traj,param )
%do inverse kinematics for trajectory
m=size(traj,2);
angles=zeros(3,m);

for i=1:m
    angles(:,i)=inverseKinematics([traj(1,i),traj(2,i),traj(3,i)],param);
end

end
```

APPENDIX C

```
%%Plotting functions and animation
%Authors: Aditya Khopkar, Sukoon Sarin

function [] = plotRobot( traj,t, param )
%plot position, given Pose
%parameters contains: [e,f,re,rf]

% initialize
%coords
x=traj(1); y=traj(2); z=traj(3);
t1=t(1); t2=t(2); t3=t(3);
%rod lengths in m:
rf=param(4);
re=param(3);

%triangular side lengths in m:
f=param(2);
e=param(1);

%% plot
%init
grid on
%to adjust the view
l=(f+2*rf);
axis([-l l -l l -l*2 l/2])
cla;

%calc and plot robot f plate
P_f1=[f/2,-f/(2*sqrt(3)),0];
P_f2=[0,f/cos(30*pi/180)/2,0];
P_f3=[-f/2,-f/(2*sqrt(3)),0];

line([P_f1(1) P_f2(1)],[P_f1(2) P_f2(2)],[P_f1(3) P_f2(3)])
line([P_f2(1) P_f3(1)],[P_f2(2) P_f3(2)],[P_f2(3) P_f3(3)])
line([P_f3(1) P_f1(1)],[P_f3(2) P_f1(2)],[P_f3(3) P_f1(3)])

hold on
%calc and plot robot end-effector pose
P_e1=traj'+[e/2,-e/(2*sqrt(3)),0];
P_e2=traj'+[0,e/cos(30*pi/180)/2,0];
P_e3=traj'+[-e/2,-e/(2*sqrt(3)),0];

line([P_e1(1) P_e2(1)],[P_e1(2) P_e2(2)],[P_e1(3) P_e2(3)])
line([P_e2(1) P_e3(1)],[P_e2(2) P_e3(2)],[P_e2(3) P_e3(3)])
line([P_e3(1) P_e1(1)],[P_e3(2) P_e1(2)],[P_e3(3) P_e1(3)])

hold on

plot3(x,y,z,'o','color','green')%plotting the end-effector point

%calc and plot joints f
P_F1=[0,-f/(2*sqrt(3)),0];
```

```
%use rotation matrix to calc other points
deg=120;
R=[cosd(deg),-sind(deg),0;sind(deg),cosd(deg),0;0,0,1];
P_F2=(R*P_F1')';
P_F3=(R*P_F2')';
%Plot F1,F2,F3
hold on
plot3(P_F1(1),P_F1(2),P_F1(3),'o','color','green')
hold on
plot3(P_F2(1),P_F2(2),P_F2(3),'o','color','green')
hold on
plot3(P_F3(1),P_F3(2),P_F3(3),'o','color','green')
hold on

P_J1=[0,-f/(2*sqrt(3))-rf*cos(t1),-rf*sin(t1)];
P_J2=(R*[0,-f/(2*sqrt(3))-rf*cos(t2),-rf*sin(t2)]')';
deg=-120;
R=[cosd(deg),-sind(deg),0;sind(deg),cosd(deg),0;0,0,1];
P_J3=(R*[0,-f/(2*sqrt(3))-rf*cos(t3),-rf*sin(t3)]')';
%rf link plot
line([P_F1(1) P_J1(1)],[P_F1(2) P_J1(2)],[P_F1(3) P_J1(3)],'color','red')
line([P_F2(1) P_J2(1)],[P_F2(2) P_J2(2)],[P_F2(3) P_J2(3)],'color','red')
line([P_F3(1) P_J3(1)],[P_F3(2) P_J3(2)],[P_F3(3) P_J3(3)],'color','red')
%plot J1, J2, J3
hold on
plot3(P_J1(1),P_J1(2),P_J1(3),'o','color','green')
hold on
plot3(P_J2(1),P_J2(2),P_J2(3),'o','color','green')
hold on
plot3(P_J3(1),P_J3(2),P_J3(3),'o','color','green')
hold on

%calc and plot joint e
Transl=[0,-e/(2*sqrt(3)),0];
P_E1=Transl+[x,y,z];
deg=120;
R=[cosd(deg),-sind(deg),0;sind(deg),cosd(deg),0;0,0,1];
P_E2=((R*Transl')+traj)';
deg=-120;
R=[cosd(deg),-sind(deg),0;sind(deg),cosd(deg),0;0,0,1];
P_E3=((R*Transl')+traj)';
%re link plot
line([P_J1(1) P_E1(1)],[P_J1(2) P_E1(2)],[P_J1(3) P_E1(3)],'color','red')
line([P_J2(1) P_E2(1)],[P_J2(2) P_E2(2)],[P_J2(3) P_E2(3)],'color','red')
line([P_J3(1) P_E3(1)],[P_J3(2) P_E3(2)],[P_J3(3) P_E3(3)],'color','red')

hold on
xlabel(['x = ' num2str(x)]);
ylabel(['y = ' num2str(y)]);
zlabel(['z = ' num2str(z)]);

end

%%Animate

function [] = Animation( angles,traj,param )
```

```
%ANIMATION Make Plot animation
%simulation step size
N=size(angles,2);
dt=0.1;

for i=1:N
    tic
    plotRobot(traj(:,i),angles(:,i),param);
    toc
    pause(dt);
end

end
```

APPENDIX D

```
%%Main file
%Authors: Aditya Khopkar, Sukoon Sarin
%Needs to be executed as a live script .mlx
% Simulate a Delta Robot mechanism

%% init
clc;
clear all;

%% Parameters
%rod length in m:
rf = 0.524;  %Base to Joint
re = 1.244;  %Joint to end effector

%triangular side length in m:
f =0.576;  %base
e = 0.076; %endeffector
param=[e,f,re,rf];
workspace(param);
%% kinematics
%actuator angles:
m=5; %number of trajectory points
n=4; %number of Poses

r0=[0, 0, -1.116]; %start Pose
rGoal=zeros(n+1,3);

%Setting goal positions
rGoal(1,:)=[-0.06558, -0.05222, -1.339];
rGoal(2,:)=r0';
rGoal(3,:)=[0.2315, -0.4401, -1.258];
rGoal(4,:)=r0';

%Initialize the matrix
trajectory=zeros(3,m,n);
angles=zeros(3,m,n);

for i=1:n
    trajectory(:,:,i)=CalcTrajectory(rGoal(i,:),rGoal(i+1,:),m);
    angles(:,:,i)=CalcTrajectoryAngles(trajectory(:,:,i),param)*pi/180;
end

%% Plot
npackages = 3; %Define the number of packages the robot wants to pick and
sort
while(npackages~=0)
    for i=1:n
        Animation(angles(:,:,i),trajectory(:,:,i),param);
    end
    npackages =  npackages-1;
end
```