Imperial College London

MEng Robotics Manipulation Project

Imperial College London

Department of Electrical and Electronic Engineering

Robotics Manipulation

Author:
Boon Liang Wong, Pelin Ulusoy

Module Leader:
Prof. Ad Spiers

March 31, 2022

Contents

Chapter 1

Robot Modelling

Forward kinematics:

i	α	a	d	θ
1	0	0	0	θ_1
2	0	0	7.7	0
3	-90	0	0	$\theta_2 - 90 + rad2deg(atan(2.4/12.8))$
4	0	$l_1 = 13.0$	0	$\theta_3 - rad2deg(atan(2.4/12.8))$
5	0	$l_2 = 12.4$	0	θ_4
6	0	$l_3 = 12.6$	0	0

Table 1.1: DH Table for Open Manipulator-X

Base frame of notation z is pointing upwards with x and y forming the horizontal plane. All θs represent angles of their respective actuators. In implementation rather than simulation, θ value of 0 would mean 2048 in encoder values except for the third encoder that has a θ offset which we get rid of by subtracting 90 when converting to encoder values after inverse kinematics. To summarise, in our simulation, 0 for all θ in DH table would mean robot pointing upwards and looking towards the x-axis directly.(figures: A.1, A.2)

Transformation 1: Contains only θ_1 because z-axis of world frame is already pointing out axis of rotation.(figure A.3)
Transformation 2: This transformation is only made for visualisation purposes as combining transformations 1 and 2 would create the same calculations for latter frames. In our case, we wanted to make sure the visualized frames are at same locations as our actuators. Link offset of 7.7cm is the distance to the second frame along z-direction. (figure A.4)
Transformation 3: Transformation 3 is made up of several components. First of all, we wanted to twist the z into axis of rotation and x to the link direction. Angle of -90 is the link twist which puts z into position but y along the link. So we subtract 90 from the joint angle in θ column to create a 90 degree turn around the z axis and put x-axis along the link. The other offset is the atan(2.4/12.8) which is angle between imaginary hypotenuse to the next frame and next link. As there is some offset in y direction of the current frame, we can neither use link length which is along x or link offset which is along z. So we are taking the link as the imaginary line between the two actuators and add the θ offset to make x point in the same direction as the imaginary link. We also have θ_2 which belong to the angle of the second actuator.(figure A.5)
Transformation 4: In transformation 4, we are using previously mentioned imaginary link length as 13.0cm and subtract the θ because when all angles are 0 we want θ_3 to be perpendicular to ground. Not only that, θ_3 itself is not measured from imaginary link but the real one. θ_3 here is the angle of the third actuator.(figure A.6)
Transformation 5: Transformation 5 has next link length of 12.4cm and joint angle, θ_4. We don't need to add/subtract offset as they cancel each other out in frame 3 & frame 4.(figure A.7)
Transformation 6: Transformation 6 only consists of 12.6cm link length which extends until the middle length of gripper. X-direction is the robot direction when encoder 1 is 2048. Y-direction is the other axis forming the base plate together with x. Both are increasing away from the base of the robot. (figure A.7)

Graphical simulation: For our simulation, we are creating the links by using x, y, and z co-ordinates located at the last column of our transformation matrix. Links are created by draw_link function which takes the 2 matrices representing their respective frames as inputs and draws a line between their points using plot3 function.

For **visualising co-ordinate frames of each axis**, we shifted their respective DH matrices along x y and z by 1 to help draw unit vectors along each of the axis. For x shift we did a DH transform with link length. For z shift we did a DH transform with link offset. For y we first did a link twist of 90 to make z in the direction of y. Then we did the link offset again as now we moved along the new z direction. We got the position of each of these new frames and then created lines from original matrices to shifted ones representing each of the axis using plot3, red for x axis, blue for y and green for z.

To **visualise the whole model**, our draw function gets 4 θ inputs and uses forward kinematics to create individual frames. We then call draw_link function on connected pairs of frames and show coordinates on individual ones. Frame 2 is not shown as it does not represent an individual actuator. We assign these plots to variables and delete them when the function is next called. This is why all of the variable names are global and deletion is only done after function is called at least once. There is also a 0.000001 seconds pause to ensure all robot positions can be seen individually. We draw a dot at the tip of the tool which is the only part of the function not get deleted next call to ensure path can be visible at the end.

We have tested our forward kinematics with different combinations to ensure the angles are as intended and the forward kinematics works.(figures: A.1, A.2)

Inverse kinematics: In inverse kinematics we first assigned the tool angle which is the angle it makes with the z axis. This angle is the total angle of θ_2, θ_3 and θ_4 which we named it ϕ. Then we find the θ_1 from x and y using atan2 which allows us to treat motion as 2d. We can assign a temporary variable $\hat{x}$ which is the horizontal distance of the tip to the base. We also create a temporary variable $\hat{z}$ which is z - 7.7 to ensure all the points are shifted down by the distance between actuators 1 and 2. As there is no axis creating a change in z of the tool before actuator 2 we can treat this point as the new origin.

$$\begin{aligned}
\theta_1 &= atan2(y, x) \\
\hat{x} &= \sqrt{(x^2 + y^2)}\text{,horizontal distance of the tip to the base} \\
l_1 &= 13.0 \\
l_2 &= 12.4 \\
l_3 &= 12.6 \\
\hat{z} &= z - 7.7
\end{aligned}$$

Using $\hat{x}$ and $\hat{z}$ we now have a system with three θ and their sum which is the tool angle ϕ. Using the tool angle we can find coordinates of the third actuator by subtracting tool length's $\hat{x}$ and $\hat{z}$ components from the tool position.

$$\begin{aligned}
\hat{z} &= \hat{z} - l_3 \cos\phi \\
\hat{x} &= \hat{x} - l_3 \sin\phi
\end{aligned}$$

Now we have a model similar to the one in the lectures with an end point and two encoder values one for the link1 which is θ_2 the other is for relative one of the link2 which is θ_3.

$$\begin{aligned}
\cos\theta_2 &= \frac{\hat{z}^2 + \hat{x}^2 - l_1^2 l_2^2}{2 l_1 l_2} \\
\sin\theta_2 &= \sqrt{1 - \cos(\theta_2)} \\
\theta_3 &= atan2(\sin\theta_2, \cos\theta_2) \\
k_1 &= l_1 + l_2 \cos\theta_2 \\
k_2 &= l_2 \sin\theta_2
\end{aligned}$$

Finishing the inverse kinematics we need to make sure we get rid of the offset the same way we did in forward kinematics. In this case θ_2 being 0 would mean actuator 2 and 3 along the same line which needs a shift back and then we add the offset to the next θ. Also note we are using elbow up version for our inverse kinematics.

$$\begin{aligned}
\theta_2 &= atan2(\hat{x}, \hat{z}) - atan2(k_2, k_1) - atan(\frac{0.024}{0.128}) \\
\theta_3 &= \theta_3 + atan(\frac{0.024}{0.128}) \\
\theta_4 &= \phi - \theta_3 - \theta_2
\end{aligned}$$

Some important things to point here is that our inverse kinematics does not use the whole directional information of the tool which in this case is reasonable as we can only make combinations formed in the vertical plane formed by the hypotenuse of x and y axes and z axis. This is caused by having only 4 actuators thus having only 4DOF. $\phi = 0$ would mean tool pointing upwards, 90 means away from the robot horizontally and 180 would point downwards. X, y and z inputs are defined the same way as our forward kinematics. In the actual implementation of the robot as 2048 is not the upwards but forward direction for encoder 3 we implement one more line which is subtracting 90 degrees from θ_3. This shifts the angle counterclockwise direction in the previously mentioned vertical frame ensuring any angle we calculate for the encoder 3 doesn't have an offset.

It is also important to show how these values which are currently in degrees are calculated to encoder values. As we said the 0 degrees are equal to 2048 in encoder values we add -180 to the degrees to shift their range from [-180,180] to [0,360]. Afterwards we do the basic conversion required by rationing the degree range to encoder range,

$$enc = ((deg + 180)/360) * 4095$$

Creation of the squares: To create the squares we use the inverse kinematics to feed the desired end positions (and plausible tool directions) to get the θ required for the simulation. We calculate the points which are put into an array to prevent any delays that can be caused by calling inverse kinematics throughout the simulation. (Full Code: A.13)

The other important thing is the pacing of these points. While the simulation itself doesn't have torque thus would be smooth regardless of the placement of the points. To consider how this actually would be implemented in the real robot, we used cubic interpolation for waypoints in Cartesian Space.

$$\theta(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$
$$\text{a}_0 = x_0, a_1 = 0, a_2 = \tfrac{3}{x^2 f}(x_f - x_0), a_3 = \tfrac{-2}{x^3 f}(x_f - x_0)$$

As this function creates the points that are in a line, thing that needs to be added is x_f and x_0 which are the start and end points for each of the lines (x is shorthand notation for 4 variables required: x, y, z and ϕ). Thus any square needs to have 4 corners as 4 points defined and we feed that into a function we name getthosepoints to output an array of points on the line. It is important that each corner point is x_f and x_0 once to get them fully connected. This means that for getthosepoints functions has to have input point pairs, (p1,p2), (p2,p3), (p3,p4), (p4,p1). We also need to specify number of points we want to create along the line and angles at both end points.(figure A.9)

Chapter 2

Pick and Place (Wooden Blocks)

2.1 Setting of Open Manipulator-X

In this task, we decided to use: (Appendix: A.10)

1. Drive Mode(10) is set to Time-based Profile(4).
2. Operating Mode(11) is put into Position Control Mode(3) with Max/Min Position.
3. Profile Velocity(112) for 4 joints excluding Gripper = 300 ms
4. Profile Acceleration(108) for 4 joints excluding Gripper = 30 ms
5. Profile Velocity(112) for Gripper = 200 ms
6. Profile Acceleration(108) for Gripper = 30 ms

Time-based Profile is selected for this task because it is more intuitive and it encourages smooth movement as compared to Velocity-based Profile. By changing Profile Velocity and Profile Acceleration, we are setting the time span to reach the velocity (the total time) as well as acceleration time of the Profile. In Time-based Profile, when DYNAMIXEL receives updated desired position from a new Goal Position(116) while it is moving toward the previous Goal Position(116), velocity smoothly varies for the new desired velocity trajectory.

2.2 Standby Position

To perform any task, Open Manipulator-X is given a predefined position as Standby Position to ensure that the robot is moving to known position before starting any tasks. This allows us to understand the trajectory of the robot when it moves to the next position. (Appendix: A.10)

2.3 Cartesian Space Trajectory and Joint Space Trajectory

For task 2, we manually craft the Waypoints for Cartesian Space to plan its Trajectory where we can decide which path is the fastest for robot arm to move. Along the path, inverse kinematics used in Task 1 is utilised to calculate the 4 joint angles for each waypoint. However, the movement is not as smooth as we thought. Therefore, Joint Space Trajectory is implemented to calculate joint angles between 2 waypoints. In this case, cubic interpolation with 6 time step values is used as below:

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, (Appendix : A.11)$$
$$\mathrm{a}_0 = \theta_0, a_1 = 0, a_2 = \frac{3}{t^2 f}(\theta_f - \theta_0), a_3 = \frac{-2}{t^3 f}(\theta_f - \theta_0)$$

To avoid collision in Cartesian Space, joint space trajectory is not implemented for end effector. When end effector reached the desired location, it is programmed separately to release or grip object with pausing in place before next movement. (Example code for Waypoints + Inverse Kinematics + Joint Space Trajectory: A.12)

2.4 Specific Elements of Task 2

For **Task 2a**, Gripper position is always pointing downward towards the spectral board to grip object and place it in designated location. This ensures the gripper grips at the center of cube which allows easier release/placement of cube in end position.

For **Task 2b**, the general trajectory in Cartesian Space is to move to the location of cube, grip the cube at the center with gripper pointing downwards. The gripper turns 90° degree to become parallel to the board and place it at end position when the red face of cube is at the top. For different orientation of cube, the planned trajectory above can be repeated until red face of cube is facing upwards. Placing of cube when gripper is parallel to the board is challenging because inverse kinematics used assumed that the cube is always gripped at the center of gripper length instead of the tip of gripper. However, we grip using the tip of gripper. This factor is taken into account when deciding waypoints with fine-tuning function in place to ensure generalisation of code.

For **Task 2c**, we start by picking an ending position that is easier for turning of gripper joint. The rest of the trajectory is the combination of task2a with task2b. If the red face is pointing inwards or outward, we generally solve this by moving the cube to stacking position while changing orientation of cube at the same time to smooth out the movement. If red face is facing downwards, repeated motion of task2b is needed before moving to end position.

For more information, please refer to code submission or `https://youtu.be/qZf5Lk_meao?t=24`

Chapter 3

Trajectory Following (Drawing)

There are two main additions needed to be done for task 3. First is the design of the pen holding tools. The second one is creating trajectories for drawing straight lines and arcs.

1. **Pen holding** (Video: `https://youtu.be/qZf5Lk_meao?t=109`)

Our design went through 2 main adjustments. Cubic design for pen holder was the first idea. In order for pen to not tilt while drawing, planar extensions perpendicular to the length of the pen are added at two opposing sides of the cube (figure A.10). This way the cube would restrict tilting of pen during gripping thus the pen will be stable. Yet these two parallel planes are incompatible for the original gripper because the original gripper of the robot has a slope to it from base to tip. To solve this problem, we decided to design new set of grippers (figure A.11) with straight arm length extensions to ensure fitting in between the two planar extensions of the cube. Another version with form closure (figure A.12) is also designed where one more extension at tip is added, creating a barrier in front of the cube.

Both of these designs did not work because we had to be very precise about the gripping (the planar extensions sometimes obstruct the gripping). Not only that, we did not realize the importance of rubber texture within the original gripper that enhances friction during gripping. We believe that the friction provided by the rubber is a better solution. So we changed our design to not include any sort of extensions but rather a simple cube (figure A.13). It has the same dimensions as the cubes from task 1. The hole within it is designed to have the same slope as the pen to prevent sliding.

The next thing was the pen stand design (figure A.14). To grip the pen holding cube at the center of gripper length, it is easier to grip it if the pen holder(cube) is standing. Although it is easy to change the inverse kinematics for tool length, it is not secure to grip at the tip as the tip is not covered by the rubber texture of the gripper. So we decided to use a pen stand with slightly bigger cut of radius of the pen and a little extrusion at the bottom to fit into the hole of spectral board.

2. **Drawing Straight Lines and Arcs**

For diagonal, horizontal and vertical lines (straight lines), we get the starting point and end point as inputs, then convert it into equidistant interval points (Appendix: A.9, lingen function). Depending on how many points are needed, we can modify the function accordingly. The generated interval points between starting and end positions will be used as waypoints. Whereas for arc drawing, radius and center of arc will be used as inputs and processed by cosine and sine operators to generate interval points for the arc. In this function, arc circumference and drawing direction (clockwise/anticlockwise) can be modified by setting rotation angles (Appendix: A.9, arcgen function). Once these waypoints are generated, we only need to decide drawing sequences of the lines and move joint angles through joint space scheme to smooth out motion.

Chapter 4

Interactive Bartendering

The idea of our task is to enable robotic arm to act as part of futuristic pub/bar that allows customer to interact with robotic arm by giving specific token of different shapes to the robot arm to instruct the robot to perform different tasks such as handing food and pouring alcohol shot. We believe that by giving customers the opportunity to interact with robotic arm, more customers will be attracted by the interactive elements. Moreover, the delicate pouring motion of the robot can also be part of marketing plan. Therefore, more business inflow to the industry.

To be more specific, customer will be given token of different shapes based on their needs. Then, customer will hand the token to the robot by closing its gripper, the encoder in gripper will interpret the token based on the size. It will then perform designated task associated with the size of token. For more information, please take a look at the demonstration video: `https://youtu.be/qZf5Lk_meao?t=148`.

4.1 Specific Implementation

1. **How to accept and interpret token?**

First, we 3D printed tokens that are rectangular in shape but differ in length. To allow the robot arm to accept the token, we disable the torque in the robot gripper when it is in standby/stable position with the rest of joints locked. By disabling torque in gripper, customer can hand the token to the gripper and close it. At the same time, the gripper encoder will read its position continuously until it is less than a threshold value, the gripper will understand that it has been handed a token. After that, another reading of gripper position is recorded, different gripper positions correspond to different tokens (size of token is measured beforehand).

2. How to generate delicate pouring motion?

The approach taken is by having thorough understanding of both Cartesian Space and Joint Space. Since a lot of functionalities have been built through previous task, pouring motion is generated using Joint Space Trajectory to ensure smooth movement. The gripper position is adjusted accordingly based on the waypoints. Essentially, a lot of experiments were done to make sure the gripper grips the alcohol container at the right place so that it will not hit joint link during pouring. After that, gripper adjustment was done multiple times to allow fluid flow out of container in a delicate way. Once pouring is done, we also need to make sure the movement is smooth enough for fluid not to spurt out of container when it is moving back and forth.

3. How to perform different tasks with repetitive motion?

In terms of code, the whole process from accepting, interpreting token to performing tasks are placed into a while loop. Once the task has been performed, the robot arm will return to standby position to accept another token. Each task is programmed with known coordinates beforehand. Therefore, after token is interpreted, it will just run as normal program.

Appendix A

First Appendix

A.1 Different simulations with thetas

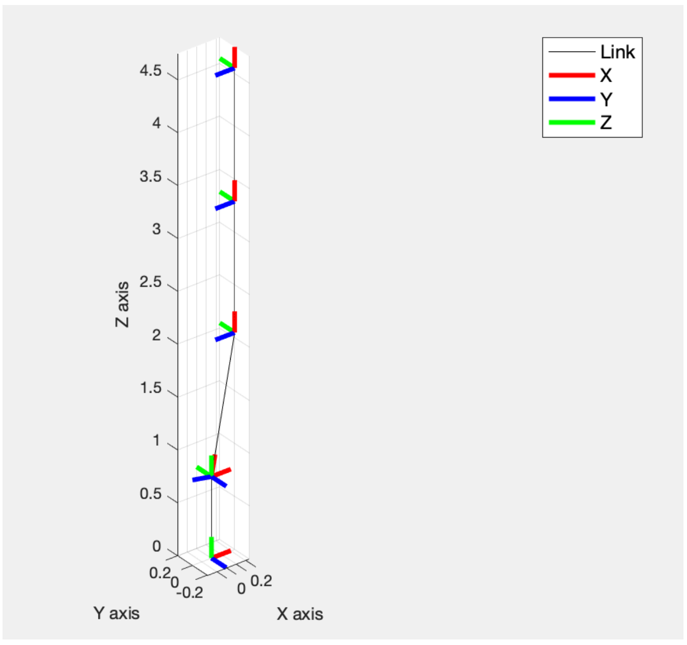

Figure A.1: All angles equal to 0

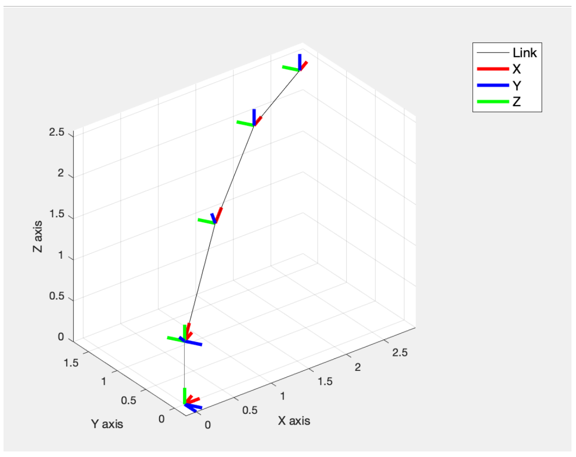

Figure A.2: All angles equal to 30

A.2 DH transformations

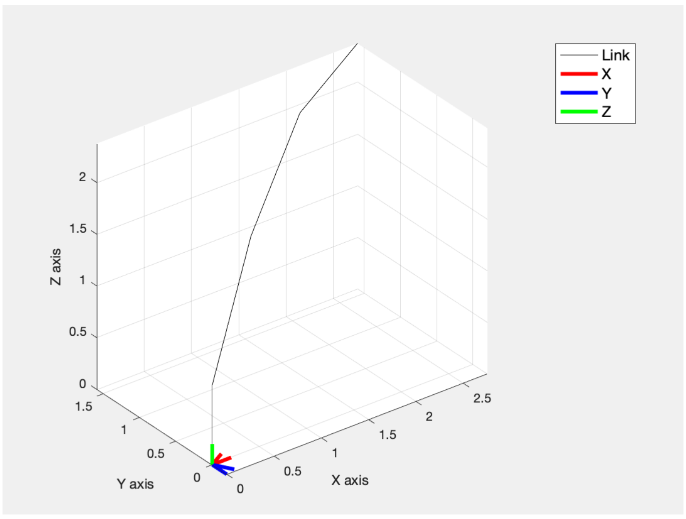

Figure A.3: First transformation on the schema with all angles equal to 30

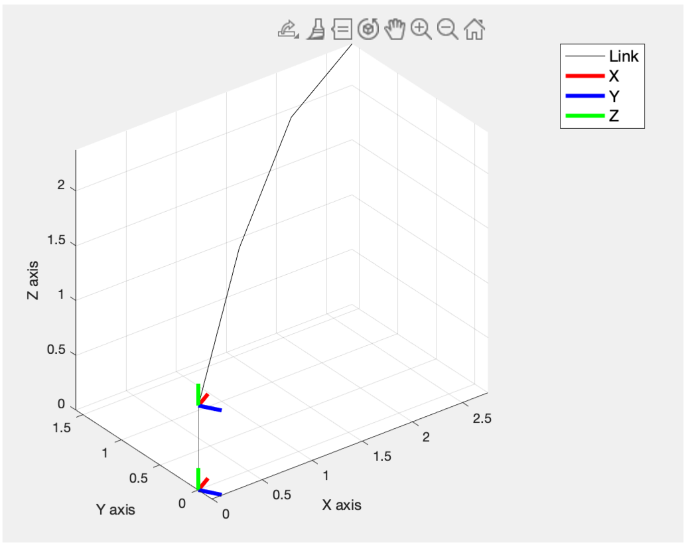

Figure A.4: Second transformation on the schema with all angles equal to 30

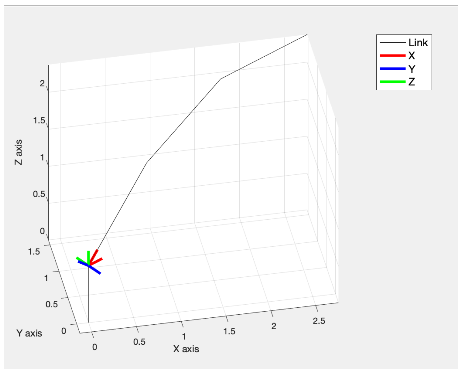

Figure A.5: Third transformation on the schema with all angles equal to 30

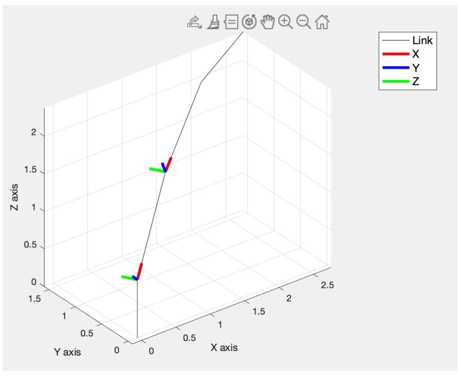

Figure A.6: Fourth transformation on the schema with all angles equal to 30

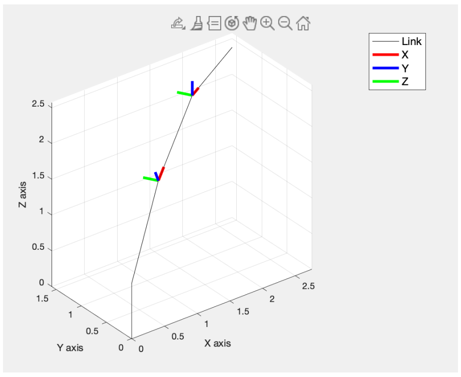

Figure A.7: Fifth transformation on the schema with all angles equal to 30

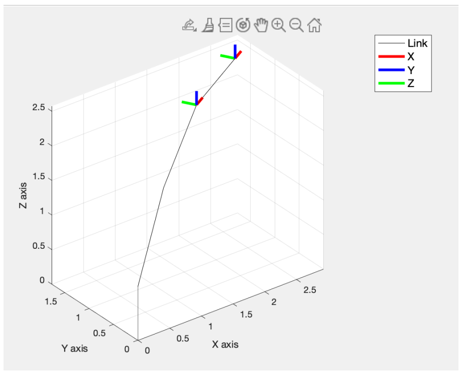

Figure A.8: Sixth transformation on the schema with all angles equal to 30

A.3 Squares

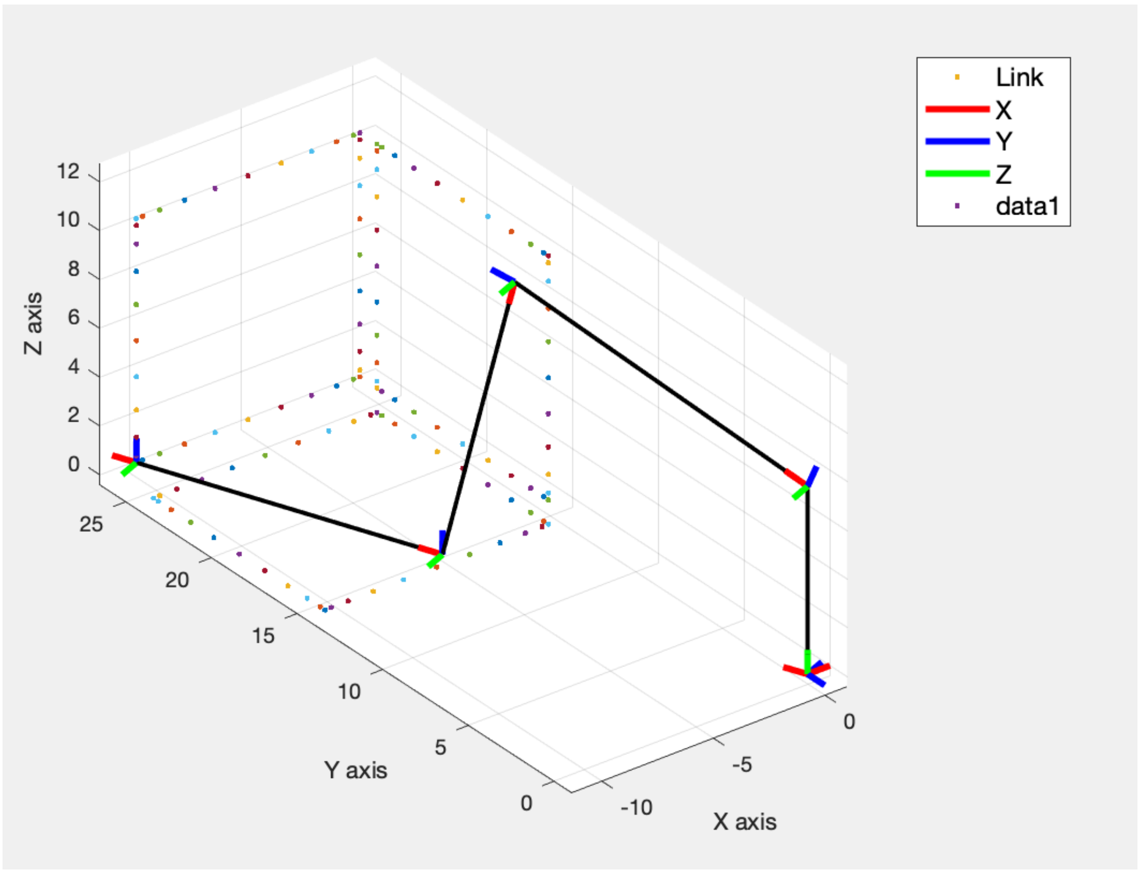

Figure A.9: Squares drawn for task 1.d

A.4 Planar extension of Cubic Design for Pen Holder

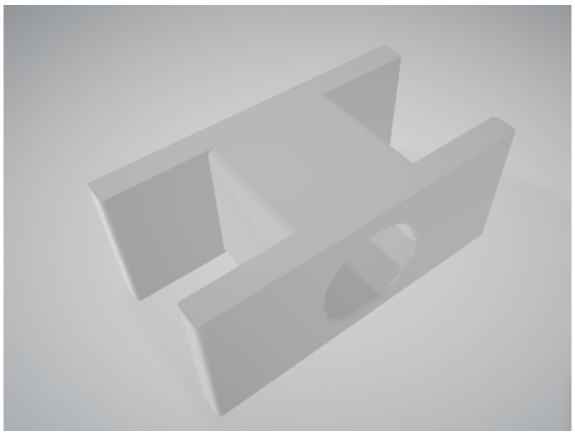

Figure A.10: Planar Extension of Cube Pen Holder

A.5 Force Closure Gripper with Straight Extension of Link Length

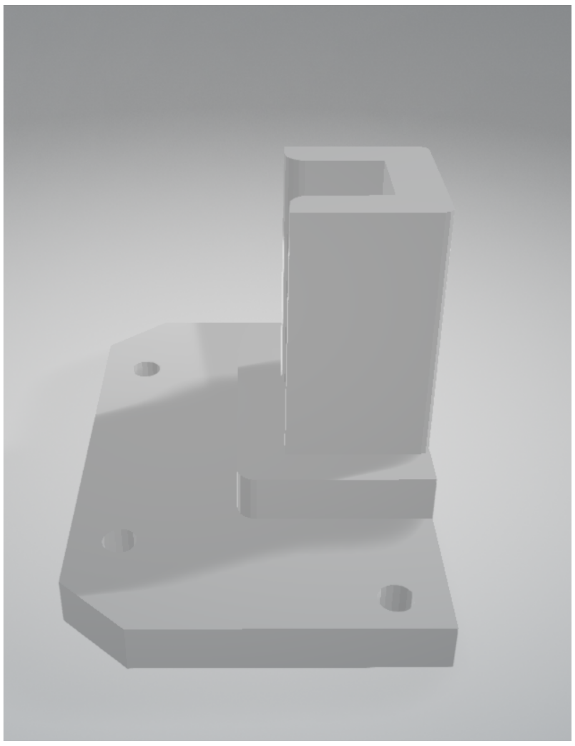

Figure A.11: Gripper Design with Straight Link Length

A.6 Form Closure Gripper with Barrier

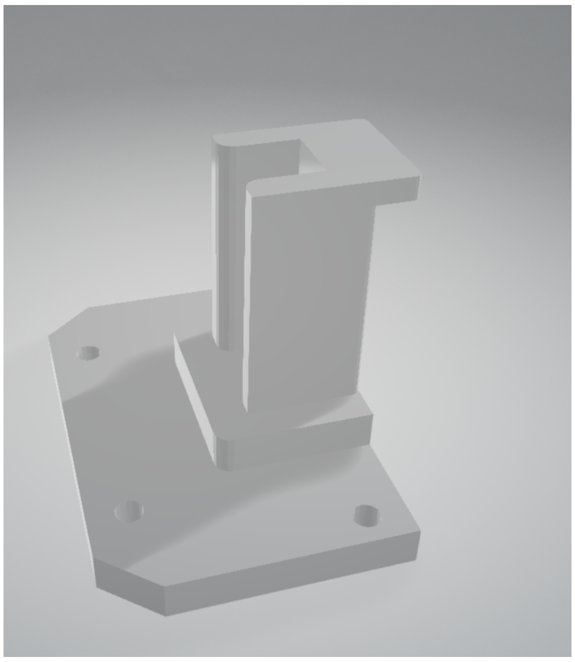

Figure A.12: Another design of gripper (form closure)

A.7 Pen Holder Version 2

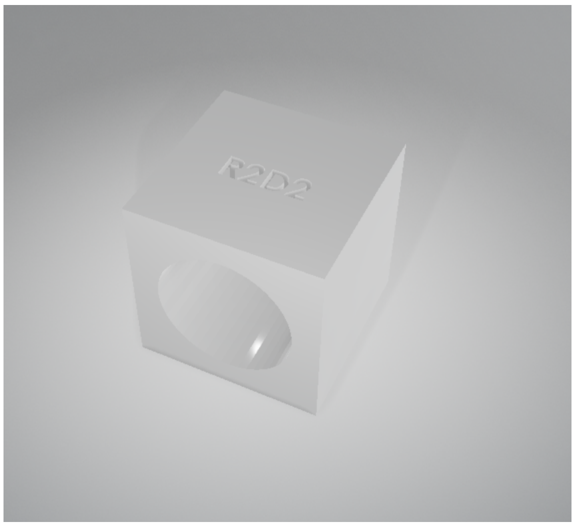

Figure A.13: Pen Holder Without Planar Extension (Final Version)

A.8 Pen Stand

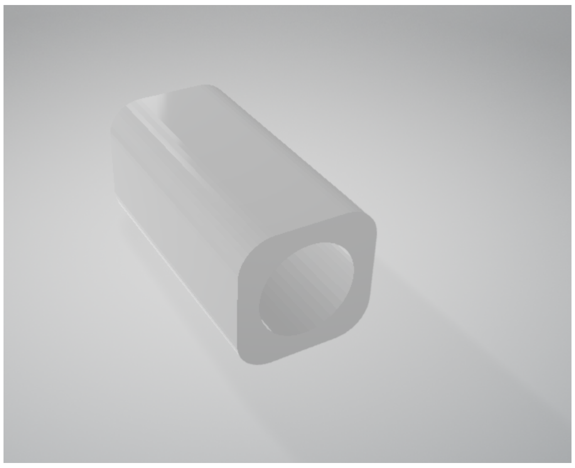

Figure A.14: Pen Stand that fits into spectral board

A.9 Code for Drawing Functions

```
function M = arcgen(x,y,z,R)
    M = [];
    for i =180:10:360 %90:10:270 %270:-10:90
        M = [M;x + R * sin(deg2rad(i)), y+R*cos(deg2rad(i)),z,90,0];
    end
end


function M = lingen(x1,y1,z1,x2,y2,start)
    M = []
    xinc = (x2 - x1)/6;
    yinc = (y2 - y1)/6;
    if start==true
        M = [x1,y1,z1,90,0];
        M = [x1,y1,z1,90,0];
        M = [x1,y1,z1,90,0];
        M = [M;x1,y1,z1,90,0];% created to start at higher spot before moving
    end
    i = 0;
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    for i = 0:6
        M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    end
    i=6;
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
    M = [M;x1 + xinc * i, y1 + yinc * i,z1,90,0];
end
```

A.10 General Setting of Open Manipulator-X & Standby Position

```
% ---- Control Table Addresses ---- %%

ADDR_PRO_TORQUE_ENABLE     = 64;          % Control table address is different in
    Dynamixel model
ADDR_PRO_GOAL_POSITION     = 116;
ADDR_PRO_PRESENT_POSITION = 132;
ADDR_PRO_OPERATING_MODE   = 11;
ADDR_DRIVE_MODE            = 10;
ADDR_PRO_VELOCITY          = 112;
ADDR_PRO_ACCELERATION      = 108;

%% ---- Other Settings ---- %%

% Protocol version
PROTOCOL_VERSION          = 2.0;         % See which protocol version is used in the
    Dynamixel

% Default setting
DXL_ID1                    = 11;          % Dynamixel ID: 1
DXL_ID2                    = 12;
DXL_ID3                    = 13;
DXL_ID4                    = 14;
DXL_ID5                    = 15;

BAUDRATE                  = 115200;
DEVICENAME                = '/dev/tty.usbserial-FT5WJ63F';   % Check which port is being
    used on your controller
                                         % ex) Windows: 'COM1' Linux: '/dev/ttyUSB0' Mac:
                                             '/dev/tty.usbserial-*'

TORQUE_ENABLE             = 1;           % Value for enabling the torque
TORQUE_DISABLE            = 0;           % Value for disabling the torque
POSITION_CONTROL_MODE     = 3;
TIME_BASED_PROFILE        = 4;

DXL_MINIMUM_POSITION_VALUE = -150000;   % Dynamixel will rotate between this value
DXL_MAXIMUM_POSITION_VALUE = 150000;    % and this value (note that the Dynamixel would
    not move when the position value is out of movable range. Check e-manual about the
    range of the Dynamixel you use.)
DXL_MOVING_STATUS_THRESHOLD = 20;       % Dynamixel moving status threshold
GRIPPER_CLOSED = 2500;
GRIPPER_OPEN = 1800;

ESC_CHARACTER             = 'e';         % Key for escaping loop

COMM_SUCCESS              = 0;           % Communication Success result value
COMM_TX_FAIL              = -1001;       % Communication Tx Failed

%% ------------------ %%

% Initialize PortHandler Structs
% Set the port path
% Get methods and members of PortHandlerLinux or PortHandlerWindows
port_num = portHandler(DEVICENAME);

% Initialize PacketHandler Structs
packetHandler();

index = 1;
```

```
dxl_comm_result = COMM_TX_FAIL;       % Communication result
dxl_goal_position = [DXL_MINIMUM_POSITION_VALUE DXL_MAXIMUM_POSITION_VALUE]; % Goal
    position

dxl_error = 0;                         % Dynamixel error
dxl_present_position = 0;              % Present position


% Open port
if (openPort(port_num))
   fprintf('Port Open\n');
else
   unloadlibrary(lib_name);
   fprintf('Failed to open the port\n');
   input('Press any key to terminate...\n');
   return;
end


% Set port baudrate
if (setBaudRate(port_num, BAUDRATE))
   fprintf('Baudrate Set\n');
else
   unloadlibrary(lib_name);
   fprintf('Failed to change the baudrate!\n');
   input('Press any key to terminate...\n');
   return;
end

% ----- SET MOTION LIMITS ----------- %
ADDR_MAX_POS = 48;
ADDR_MIN_POS = 52;
MAX_POS = 3400;
MIN_POS = 600;
% Set max position limit
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_MAX_POS, MAX_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_MAX_POS, MAX_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_MAX_POS, MAX_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_MAX_POS, MAX_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_MAX_POS, MAX_POS);
% Set min position limit
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_MIN_POS, MIN_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_MIN_POS, MIN_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_MIN_POS, MIN_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_MIN_POS, MIN_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_MIN_POS, MIN_POS);
% ---------------------------------- %


% Put actuator into Position Control Mode
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_PRO_OPERATING_MODE,
    POSITION_CONTROL_MODE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_PRO_OPERATING_MODE,
    POSITION_CONTROL_MODE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_PRO_OPERATING_MODE,
    POSITION_CONTROL_MODE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_PRO_OPERATING_MODE,
    POSITION_CONTROL_MODE);
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_PRO_OPERATING_MODE,
    POSITION_CONTROL_MODE);

% Put actuator into Drive Mode
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_DRIVE_MODE, TIME_BASED_PROFILE );
```

```
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_DRIVE_MODE, TIME_BASED_PROFILE );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_DRIVE_MODE, TIME_BASED_PROFILE );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_DRIVE_MODE, TIME_BASED_PROFILE );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_DRIVE_MODE, TIME_BASED_PROFILE );


% ENABLE Dynamixel Torque
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE
    );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE
    );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE
    );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE
    );
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE
    );

%%%% SET STANDBY POSITON
original_position_dxl1 = deg_to_enc(0);
original_position_dxl2 = deg_to_enc(-20);
original_position_dxl3 = deg_to_enc(40);
original_position_dxl4 = deg_to_enc(-20);
original_gripper_position = 1600;

% SET MAX VELOCITY (THE LARGER, THE SLOWER)
velocity = 300;
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_PRO_VELOCITY , velocity);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_PRO_VELOCITY , velocity);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_PRO_VELOCITY , velocity);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_PRO_VELOCITY , velocity);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_PRO_VELOCITY , 200); %Gripper
    need to move faster

% SET MAX ACCELERATION
acceleration = 30;
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_PRO_ACCELERATION , acceleration);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_PRO_ACCELERATION , acceleration);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_PRO_ACCELERATION , acceleration);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_PRO_ACCELERATION , acceleration);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_PRO_ACCELERATION , 30); %Gripper
    need to move faster


% MOVE TO DESIRED POSITION
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_PRO_GOAL_POSITION ,
    original_position_dxl1);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_PRO_GOAL_POSITION ,
    original_position_dxl2);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_PRO_GOAL_POSITION ,
    original_position_dxl3);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_PRO_GOAL_POSITION ,
    original_position_dxl4);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_PRO_GOAL_POSITION ,
    GRIPPER_OPEN);
pause(1)
%%
```

A.11 Joint Space Trajectory Function

```
function [j1,j2,j3,j4,j5]= jointspaceTrajectory(enc1,enc2,enc3,enc4,enc5,
    encN1,encN2,encN3,encN4,encN5,tinc, tf)
    j1 = [];
    j2 = [];
    j3 = [];
    j4 = [];
    j5 = [];
    for t = 0:tinc:tf
        j1 = [j1, theta_t(enc1,encN1,t,tf)];
        j2 = [j2, theta_t(enc2,encN2,t,tf)];
        j3 = [j3, theta_t(enc3,encN3,t,tf)];
        j4 = [j4, theta_t(enc4,encN4,t,tf)];
        j5 = [j5, enc5];
    end
end


function tt = theta_t(thi,thf,t,tf)%theta initial, theta final, time, time final
    if thi == thf
        tt = thi;
    end
    tt = thi - 0.4 + (3 / (tf)^2) * (thf - thi) * t^2 - (2 / (tf)^3) * (thf - thi) * t^3;
end
```

A.12 Example Code for Waypoints + Inverse Kinematics + Joint Space Trajectory

```
Waypoints = ...
[... % column = x, y, z, gripper orientation, gripper open/close
x1, y1, 9, 160, GRIPPER_OPEN; ...
x1, y1, 9, 160, GRIPPER_OPEN; ...
x1, y1, height, 180, GRIPPER_CLOSED;...
x1, y1, 9, 170, GRIPPER_CLOSED;...
x1, y1, 9, 170, GRIPPER_CLOSED;...
x2, y2, 9, 170, GRIPPER_CLOSED,;...
x2, y2+0.1, height, 180, GRIPPER_OPEN;...
x2, y2, 9, 170, GRIPPER_OPEN;...
];

IK = [];

for i = 1:size(Waypoints,1)
    [IK(i,1),IK(i,2),IK(i,3),IK(i,4), IK(i,5)]=
        ik(Waypoints(i,1),Waypoints(i,2),Waypoints(i,3),Waypoints(i,4), Waypoints(i,5));
end

j1= [];
j2= [];
j3= [];
j4= [];
j5= [];

for i = 1:size(IK,1)
    if i==size(IK,1)
        break;
    end
    [e1,e2,e3,e4,e5] = jointspaceTrajectory(IK(i,1),IK(i,2),IK(i,3),IK(i,4), IK(i,5),
        IK(i+1,1),IK(i+1,2),IK(i+1,3),IK(i+1,4), IK(i+1,5),1,6);
    j1=cat(2,j1,e1);
    j2=cat(2,j2,e2);
    j3=cat(2,j3,e3);
    j4=cat(2,j4,e4);
    j5=cat(2,j5,e5);
end


[numRows,numCols] = size(j1);
j5before = GRIPPER_OPEN;

for i = 1:numCols
    write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_PRO_GOAL_POSITION , j1(i));
    write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_PRO_GOAL_POSITION , j2(i));
    write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID3, ADDR_PRO_GOAL_POSITION , j3(i));
    write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID4, ADDR_PRO_GOAL_POSITION , j4(i));
    if j5(i) == 0 || j5(i) == j5before
        pause(0.001) %basically do nothing
    elseif j5(i) ~= j5before
        pause(0.6)
        write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID5, ADDR_PRO_GOAL_POSITION ,
            j5(i));
        pause(0.6)
    end
    j5before = j5(i);
end
```

A.13 Code for Open-Manipulation Simulation in MATLAB

```
clear

%getting points of the first square
[points1, inc1] = getthosepoints(25,-10,0,120,15,-10,0,120,1,10)
[points2, inc2] = getthosepoints(15,-10,0,120,15,0,0,120,1,10)
[points3, inc3] = getthosepoints(15,0,0,120,25,0,0,120,1,10)
[points4, inc4] = getthosepoints(25,0,0,120,25,-10,0,120,1,10)

%getting points of the second square
[points5, inc5] = getthosepoints(25,0,11,120,15,0,11,120,1,10)
[points6, inc6] = getthosepoints(15,0,11,120,15,0,1,120,1,10)
[points7, inc7] = getthosepoints(15,0,1,120,25,0,1,120,1,10)
[points8, inc8] = getthosepoints(25,0,1,120,25,0,11,120,1,10)

%getting points of the third square
[points9, inc9] = getthosepoints(26,-10,1,90,26,-10,11,90,1,10)
[points10, inc10] = getthosepoints(26,-10,11,90,26,0,11,90,1,10)
[points11, inc11] = getthosepoints(26,0,11,90,26,0,1,90,1,10)
[points12, inc12] = getthosepoints(26,0,1,90,26,-10,1,90,1,10)

global drawacc

drawacc = 0;%the drawing accumulator that ensures the plotting isnt deleted when null

%drawing the first square
drawthosepoints(points1,inc1);
drawthosepoints(points2,inc2);
drawthosepoints(points3,inc3);
drawthosepoints(points4,inc4);
```

```
%drawing the first square
drawthosepoints(points5,inc5);
drawthosepoints(points6,inc6);
drawthosepoints(points7,inc7);
drawthosepoints(points8,inc8);

%drawing the first square
drawthosepoints(points9,inc9);
drawthosepoints(points10,inc10);
drawthosepoints(points11,inc11);
drawthosepoints(points12,inc12);

%drawing points function which iterates through the points array calling draw
function drawthosepoints(points, inc)
    for t = 1:inc
        draw(points(1,t),points(2,t),points(3,t),points(4,t));
    end
end

%get those points create
function [points,inc] = getthosepoints(xi,yi,zi,thetai,xf,yf,zf,thetaf, tinc,tf)
    inc = 1;
    for t = 0:tinc:tf
        ttx = theta_t(xi,xf,t,tf);
        tty = theta_t(yi,yf,t,tf);
        ttz = theta_t(zi,zf,t,tf);
        tt_theta = theta_t(thetai,thetaf,t,tf);
        allvari = ik(ttx,tty,ttz,tt_theta);
        points(1,inc) = allvari(1);
        points(2,inc) = allvari(2);
        points(3,inc) = allvari(3);
        points(4,inc) = allvari(4);
        inc = inc+1;
    end
    points
    inc = inc - 1;
end

%creating the cubic points along the line based on cubic function
function tt = theta_t(thi,thf,t,tf)%theta initial, theta final, time, time final
    tt = thi - 0.4 + (3 / (tf)^2) * (thf - thi) * t^2 - (2 / (tf)^3) * (thf - thi) * t^3;
end

%drawing the points and deleting the previous ones
function draw(theta1,theta2,theta3,theta4)
    global drawacc
    global sc0
    global sc1
    global sc3
    global sc4
    global sc5
    global sc6
    global dl0
    global dl1
    global dl3
    global dl4
    global dl5
    if(drawacc>0)
        delete(dl0);
        delete(dl1);
        delete(dl3);
        delete(dl4);
        delete(dl5);
        delete(sc0);
        delete(sc1);
        delete(sc3);
        delete(sc4);
        delete(sc5);
        delete(sc6);

    end
    drawacc = drawacc + 1;
    T0 = dhm(0,0,0,0);
    T1 = T0 * dhm(0,0,0,theta1);
    T2 = T1 * dhm(0,0,7.7,0);
    T3 = T2 * dhm(-90,0,0,theta2-90 + rad2deg(atan(0.024/0.128)));
    T4 = T3 * dhm(0,13.0,0,theta3 - rad2deg(atan(0.024/0.128)));
    T5 = T4 * dhm(0,12.4,0,theta4);
    T6 = T5 * dhm(0,12.6,0,0);

    dl0 = draw_link(T0,T1);
    dl1 = draw_link(T1,T3);
    %draw_link(T2,T3);
    dl3 = draw_link(T3,T4);
    dl4 = draw_link(T4,T5);
    dl5 = draw_link(T5,T6);

    sc0 = show_coordinate(T0);
    sc1 = show_coordinate(T1);
    %show_coordinate(T2)
    sc3 = show_coordinate(T3);
    sc4 = show_coordinate(T4);
    sc5 = show_coordinate(T5);
    sc6 = show_coordinate(T6);

    pause(0.001)



    hold on
    plot3(T6(1,4),T6(2,4),T6(3,4),'.');
    grid on
    hold on
end

%dh matrix creation
function tm = dhm(alpha,a,d,theta)

    tm = [cos(theta*pi/180), (-1 * sin(theta*pi/180)),0 ,a;
          sin(theta*pi/180)*cos(alpha*pi/180), cos(theta*pi/180)*cos(alpha*pi/180), (-1 *
              sin(alpha*pi/180)),(-1 * sin(alpha*pi/180))*d;
          sin(theta*pi/180)*sin(alpha*pi/180), cos(theta*pi/180)*sin(alpha*pi/180),
              cos(alpha*pi/180), cos(alpha*pi/180)*d;
          0,0,0,1];

end

%drawing the links
function p = draw_link(a1,a2)
    posx1 = a1(1,4);
    posy1 = a1(2,4);
    posz1 = a1(3,4);
    posx2 = a2(1,4);
    posy2 = a2(2,4);
    posz2 = a2(3,4);
    axis equal
    p = plot3([posx1,posx2],[posy1,posy2],[posz1,posz2], 'black', LineWidth=2)
    zlim([-2 30])
    xlim([-10 30])
    ylim([-20 20])
    hold on
end

%drawing the coordinates
function p = show_coordinate(a)
    a0 = a * dhm(0,1,0,0); %this is for x
    a1 = a * dhm(0,0,1,0); %this is for z
    a2 = a * dhm(90,0,0,0); %this is for y
    a3 = a2 * dhm(0,0,1,0);
    a4 = a3 * dhm(-90,0,0,0);
    posx = a(1,4);
    posy = a(2,4);
    posz = a(3,4);
    %axis equal
    legend('Link', 'FontSize', 12)
    p(1) = plot3([posx,a0(1,4)],[posy,a0(2,4)],[posz,a0(3,4)],'-r',
        'LineWidth',3,'DisplayName','X')
    hold on
    p(2) =
        plot3([posx,a4(1,4)],[posy,a4(2,4)],[posz,a4(3,4)],'-b','LineWidth',3,'DisplayName','Y')
    hold on
    p(3) =
        plot3([posx,a1(1,4)],[posy,a1(2,4)],[posz,a1(3,4)],'-g','LineWidth',3,'DisplayName','Z')
    grid on;
    xlabel('X axis'), ylabel('Y axis'), zlabel('Z axis')
end

%inverse kinematics
function t = ik(x,y,z,phi)
    t(1) = rad2deg(atan2(y,x))
    x = (x^2+y^2)^(1/2);
    l1= 13.0;
    l2 = 12.4;
    l3 = 12.6;
    z = z - 7.7;
    phi = deg2rad(phi);
    z = z - cos(phi) * l3;
    x = x - sin(phi) * l3;
    c2 = (z^2 + x^2 - l1^2 - l2^2) / (2 * l1 * l2);
    s2 = (1 - c2^2)^(1/2);
    t(3) = rad2deg(atan2(s2,c2));
    k1 = l1 + l2 * c2;
    k2 = l2 * s2;
    t(2) = rad2deg(atan2(x,z) - atan2(k2,k1)) - rad2deg(atan(0.024/0.128))
    t(3) = t(3) + rad2deg(atan(0.024/0.128))
    t(4) = rad2deg(phi) - t(3) - t(2)
end
```