

(Company No. 101067-P)

الجامعة الإسلامية العالمية ماليزيا  
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA  
يُونِيسَيْتِي إِسْلَامِيَّةٌ أَنْتَارَايَغُسِيَا مَلَيْسِيَا

*Garden of Knowledge and Virtue*

DEPARTMENT OF MECHATRONICS ENGINEERING

## *Project for Robotics*

MCTE 4352

ROBOTICS

SECTION 1

SEM 2, 20/21

LECTURER: DR. TANVEER SALEH

NAME	MATRIC No.
AHMAD FAAIZ BIN AZMAN SHAH	1721421

## TABLE OF CONTENTS

TABLE OF CONTENTS-----	2
INTRODUCTION-----	3
OBJECTIVES-----	3
DENAVIT-HARTENBERG REPRESENTATION-----	4
METHODOLOGY-----	5
EXTRACTING X-Y COORDINATES FROM AN IMAGE-----	9
ROBOTICS TOOLBOX + MATLAB (CODING EXPLANATION)-----	11
INVERSE KINEMATICS COMPARISON BETWEEN EXCEL & RTB TOOLBOX-----	18
FUTURE WORKS-----	20
REFERENCES-----	21
APPENDICES-----	22

## INTRODUCTION

This project is one of the requirements to complete the core course subject MCTE 4352, which is Robotics. The concepts of robotics ranging from Denavit-Hartenberg representation, forward kinematic, inverse kinematic and so much more are included in this robotics project.

Note that some of the attachments, files and works done in [my github repository](#) might not be entirely related to the project.

I have to mention that at first, I mistook the instructions of the project and thought that it would be okay to use SCARA robot for the project but then I realized, it is supposed to be a 3DOF Planar robot instead. I had started working on the SCARA robot, but I had tried my best to change it into a 3DOF planar robot. Some traces of the previous works might still appear in the attachments or at [my github repository](#).

There are also more issues worth mentioning when doing this project which will be covered throughout this report.

## OBJECTIVES

1. To develop a 3DOF planar robot of suitable size to write self last name in '*Brush Script MT*' font (on an A4 paper) using RTB toolbox.
2. To compare results of both methods of inverse kinematics between using the RTB toolbox and inverse kinematics using formula derived.
3. To simulate the robot using RTB toolbox.

## DENAVIT-HARTENBERG REPRESENTATION

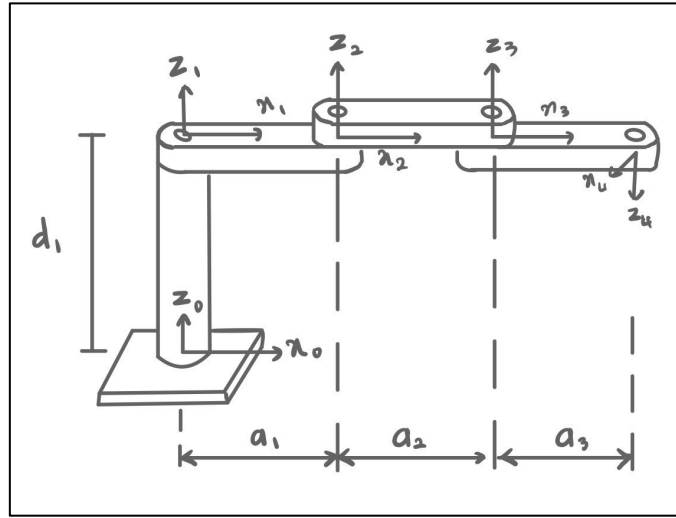


Figure 1: 3DOF Planar Robot at its zero configurations

D-H parameters:

Link, $i$	$a_i$	$\alpha_i$	$d_i$	$\theta_i$
1	$a_1$	0	$d_1$	$\theta_1$
2	$a_2$	0	0	$\theta_2$
3	$a_3$	180°	0	$\theta_3$
4	0	0	0	0

$$A_i = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_1 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & a_1 C\theta_1 \\ S\theta_1 & C\theta_1 & 0 & a_1 S\theta_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & a_2 C\theta_2 \\ S\theta_2 & C\theta_2 & 0 & a_2 S\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_3 = \begin{bmatrix} C\theta_3 & S\theta_3 & 0 & a_3 C\theta_3 \\ S\theta_3 & -C\theta_3 & 0 & a_3 S\theta_3 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_1 A_2 A_3 A_4 = \begin{bmatrix} C\theta_{123} & S\theta_{123} & 0 & a_1 C\theta_1 + a_2 C\theta_{12} + a_3 C\theta_{123} \\ S\theta_{123} & -C\theta_{123} & 0 & a_1 S\theta_1 + a_2 S\theta_{12} + a_3 S\theta_{123} \\ 0 & 0 & -1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

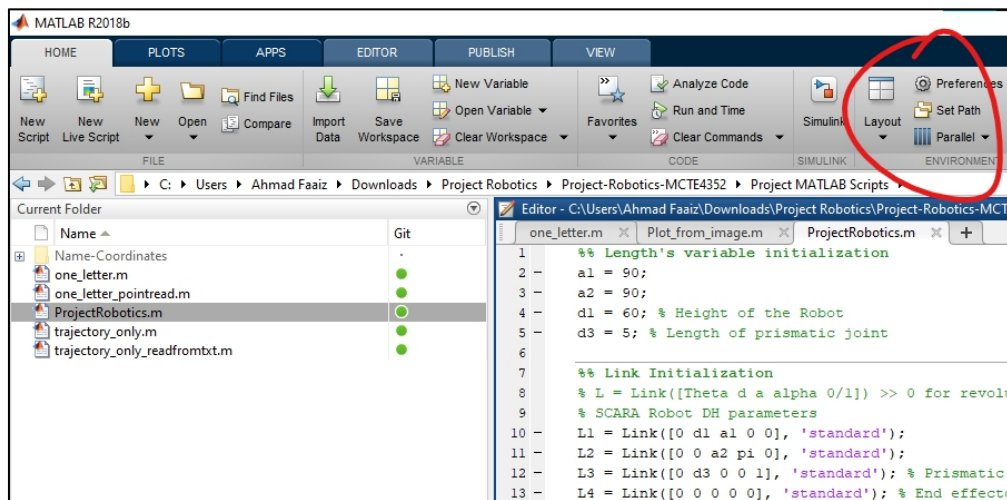
## METHODOLOGY

### Installation of the robotics toolbox (RVC)

To install the toolbox, download the required files/folder from either of this websites:

- <https://petercorke.com/toolboxes/robotics-toolbox/>
- <https://github.com/petercorke/robotics-toolbox-matlab>

From my experience, I downloaded the 'rvctools' folder from the [shared MATLAB Drive folder](#) and install it inside the MATLAB software through the 'Set Path' button.



### Understanding RTB toolbox and its functions

I have looked up some video tutorials and by referring to the [robotics toolbox manual](#), I slowly understand what the code does. Not only that, I also tried to check for each variable stored such as the L1, Rob, P1, q1 and so on to know what value it holds.

### Failed attempts and error in simulation

```
%% To calculate inverse kinematics
q0 = [0 0 0 0];
q1 = Rob.ikine(P1,[0 0 0 0],[1,1,1,0,0,0]);
q2 = Rob.ikine(P2,[0 0 0 0],[1,1,1,0,0,0]);
q3 = Rob.ikine(P3,[0 0 0 0],[1,1,1,0,0,0]);
q4 = Rob.ikine(P4,[0 0 0 0],[1,1,1,0,0,0]);
```

To know more about what the above code does, go to [section 6](#) of the code. The problem with the above code is that it references q0 as the second parameter inside ikine function for all q1, q2, q3 and q4.

By using four of those ‘q’s, the simulation will look like in ‘nonoptimal ikine.mp4’,

```
% To calculate inverse kinematics
q0 = [0 0 0 0];
q1 = Rob.ikine(P1,q0,[1,1,1,0,0,0]);
q2 = Rob.ikine(P2,q1,[1,1,1,0,0,0]);
q3 = Rob.ikine(P3,q2,[1,1,1,0,0,0]);
q4 = Rob.ikine(P4,q3,[1,1,1,0,0,0]);
```

To solve this, I found an inspiration by understanding more about the ‘ikine’ function and figured that I should reference the  $q(n-1)$  for the second parameter of the ‘ikine’ function. The resulting simulation will look like ‘smooth ikine.mp4’.

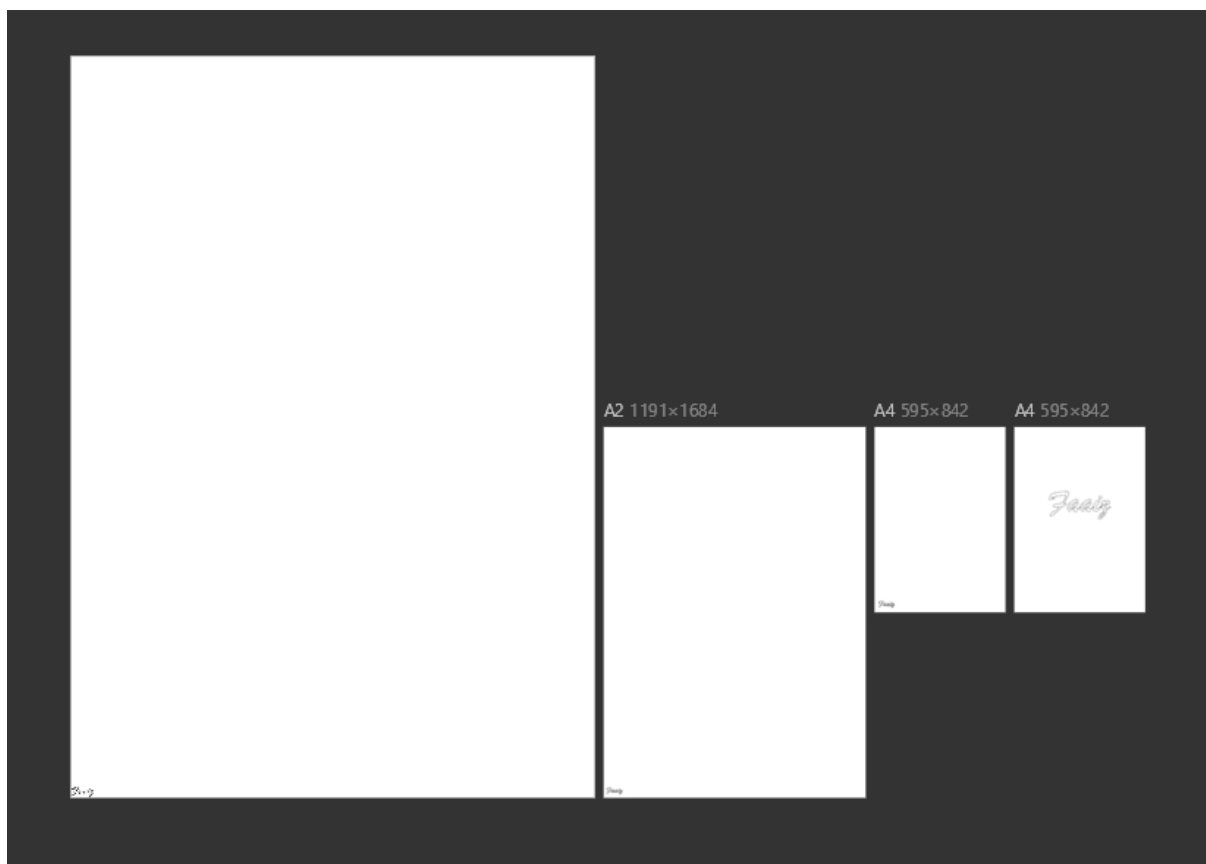
Although both robot simulations managed to get to the points of the transformation matrix, the configuration of the thetas of the robot is different based on their ‘q’. With that said, it is better to put  $q(n-1)$  as the second parameter in the ikine function.

## Plotting x-y coordinates from an image

Refer to [page 9](#) to learn more about the code on how I extract the points information from an A4-sized-paper image. To get the image of the 'lastname' inside an A4 paper, we can either use Microsoft Word or any document creation software and make sure the size is A4 paper. The issue I faced here is that there are too many points to extract the x-y coordinates. Two methods that I can think of, to reduce the points are:

1. Use a small image of my lastname inside a huge white canvas
2. Manually picked and estimated the points that we want to use

I did the first method as I wasn't aware of the requirements "[on A4 paper](#)" and ended up with fewer points but it does not meet the first objective of this project. The points that I used in this project is an image of my lastname inside an A2 paper which I exported the size two times(first from left). As you can see, there are a small word consisting my lastname at the bottom left corner of the canvas. The image that I should use to achieve the objective of this project is ones on A4-sized paper(first from right).



### Experimenting simulation with only one letter

After I had fully understand the code and confident that the code could finally simulate the robot, I tried to input the points for one letter of my lastname and see the results. During this experiment, I encountered some problems as stated before in 'Failed attempts'.

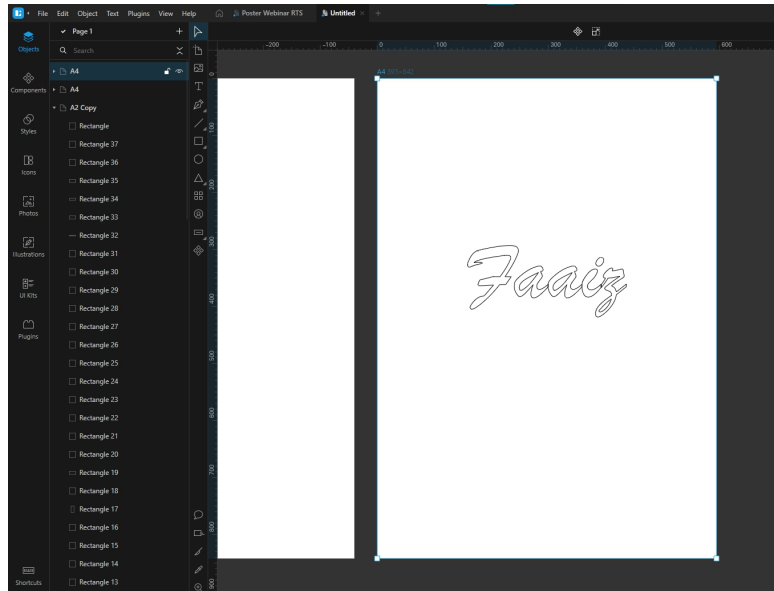
### Simulating last name by adding more coordinates

Lastly, I just put more x-y coordinates info inside the points matrix to get the full simulation of the robot writing my lastname in the *Brush Script MT* font. I have also tried reducing the lines inside the 'ProjectRobotics.m' by implementing 'dlmread' function that can read and write matrix from a text file into a MATLAB variable.

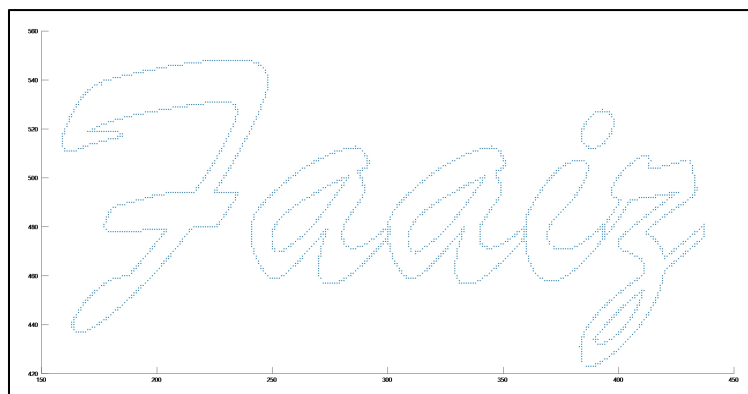


## EXTRACTING X-Y COORDINATES FROM AN IMAGE

As I mentioned before, we can create our image in any kind of document creation software. I created the image of my lastname inside “Lunacy” software:



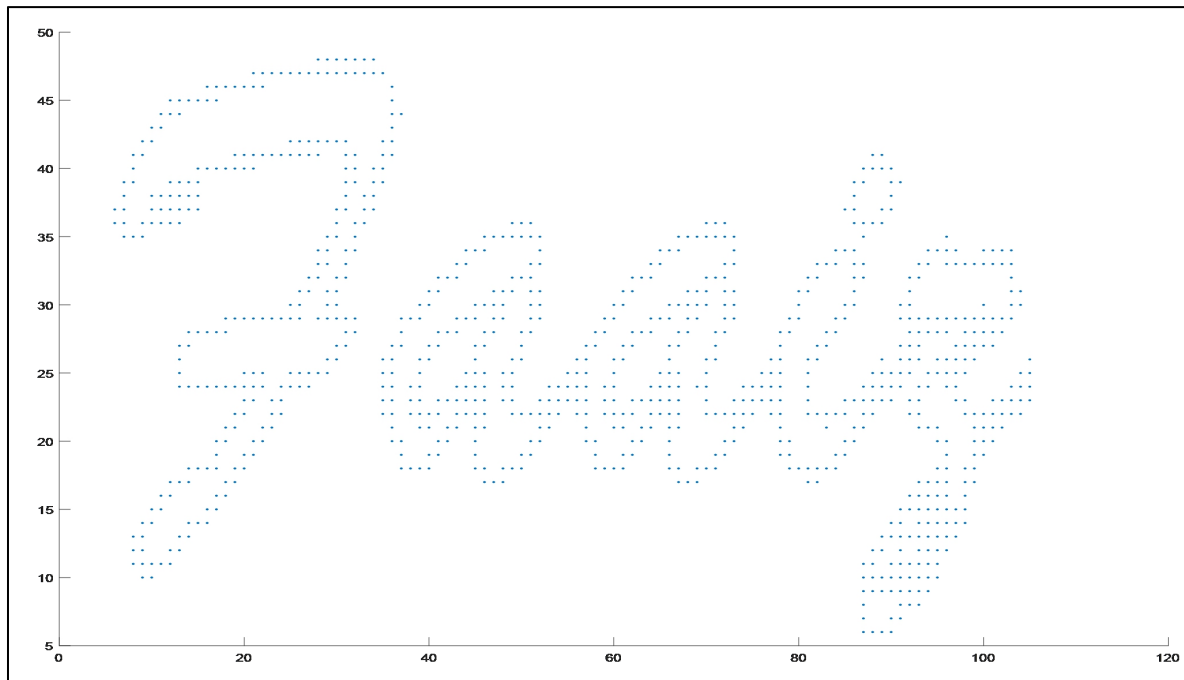
Inside the ‘pic2points’ folder in [my github repository](https://github.com/yourname/pic2points), use “Plot\_from\_image.m” to get the below plot. The MATLAB script uses ‘pic2points’ function downloaded from <https://www.mathworks.com/matlabcentral/fileexchange/54799-convert-image-pixels-to-xy-coordinates>.



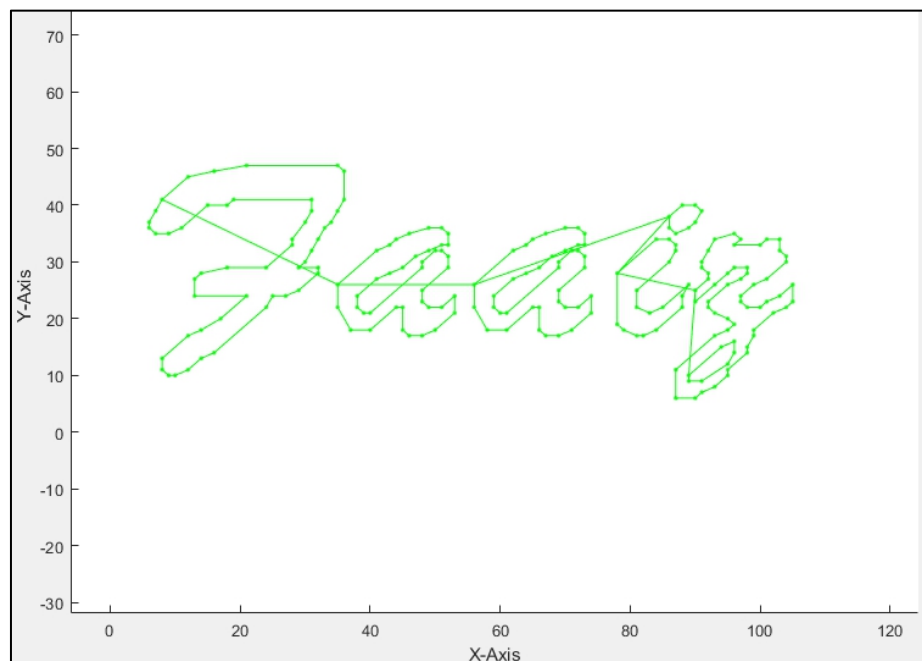
```
% Plotting points from image
Im = imread('./Images/A4_big.png'); % For testing
CoordinateMatrix = pic2points(Im,0.5);
scatter(CoordinateMatrix(:,1), CoordinateMatrix(:,2), '.');
```

The points I got using 'A4\_big' PNG image is a lot and I had experimented with different images inside 'Project-Robotics-MCTE4352/pic2points/Images/' folder to get fewer points as possible.

These are the points that I managed to extract from an A2-sized paper using the code.



These are the points I used for the trajectory of my robot, the points are from 'lastname only.txt',



## ROBOTICS TOOLBOX + MATLAB (CODING EXPLANATION)

There are **9 sections** inside the code “ProjectRobotics.m”

1. Length's variable initialization
2. Link Initialization
3. Robot settings
4. Transformation points
5. Transformation matrix
6. Calculation of Inverse Kinematics
7. Calculation of Forward Kinematics
8. Trajectory path line
9. Robot animation/simulation

### SECTION 1

```
%% Setting of the lengths of each links  
a1 = 60;  
a2 = 60;  
a3 = 60;  
d1 = 30; % Height of the Robot
```

This section lets you change the variables used for D-H parameters inside the Link settings in [section 2](#).

## SECTION 2

```
% Link Initialization
% L = Link([Theta d a alpha 0/1]) >> 0 for revolute and 1
for prismatic
% SCARA Robot DH parameters
L1 = Link([0 d1 a1 0 0], 'standard');
L2 = Link([0 0 a2 0 0], 'standard');
L3 = Link([0 0 a3 pi 0], 'standard');
L4 = Link([0 0 0 0 0], 'standard');
```

This section is to initialize the links of the robot specified with its D-H parameters.

To initialize the link, we use the `Link`([theta d a alpha sigma]) function. The first four parameters for the `Link` function is to specify the robot's D-H parameters while the fifth parameter, sigma is either '0' for revolute or '1' for prismatic.

All four links, L1, L2, L3 and L3 are specified as a revolute joint.

Variable a1, a2 and a3 represents the robot's length of link 1, link 2 and link 3 while the variable 'd1' represents the height of the robot.

For the third link, its alpha has a value of pi, so that the 'z' of the frame will point downward.

[Refer to `Denavit-Hartenberg Representation`]

### SECTION 3

```
%% Robot settings
Rob = SerialLink([L1 L2 L3 L4], 'name', '3DOF Planar Robot');
Rob.base = [1 0 0 -25; 0 1 0 -60; 0 0 1 0; 0 0 0 1];
```

In this section of the code, we set up our robot using `SerialLink` function and named our robot as 3DOF Planar Robot. The name of the robot will appear when plotting our robot.

Also, we set up the robot's base to be at the location of  $x = -25$ ,  $y = -60$  and  $z = 0$ .

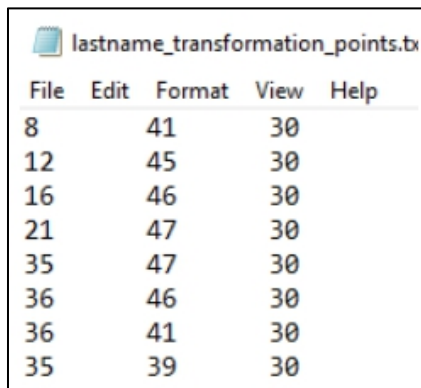
$$Rob.base = \begin{bmatrix} 1 & 0 & 0 & -25 \\ 0 & 1 & 0 & -60 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### SECTION 4

```
%% Transformation points (x,y,z)
matrix = dlmread('./Name-Coordinates/lastname_transformation_points.txt');
```

All the points that the robot need to get to is put inside a text file specifying its x,y and z coordinates.

#### **Preview of the text file 'lastname\_transformation\_points.txt'**



File	Edit	Format	View	Help
8	41	30		
12	45	30		
16	46	30		
21	47	30		
35	47	30		
36	46	30		
36	41	30		
35	39	30		

First column represents the x-coordinates, second column represents the y-coordinates and the third column represents the z-coordinates. The 'dlmread' function will read the file and store the contents inside 'matrix'.

## SECTION 5

```
%% Transformation matrix
P1 = transl(matrix(1,1),matrix(1,2),matrix(1,3));
P2 = transl(matrix(2,1),matrix(2,2),matrix(2,3));
P3 = transl(matrix(3,1),matrix(3,2),matrix(3,3));
.
.
.
P232 = transl(matrix(232,1),matrix(232,2),matrix(232,3));
```

This part of the code will store all the transformation matrix of each points from section 4 in terms of translation matrix. The transformation matrix will look like this:

$$P_i = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{For } P1 = \begin{bmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 41 \\ 0 & 0 & 1 & 30 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## SECTION 6

```
%% Calculation of inverse kinematics
q0 = [0 0 0 0];
q1 = Rob.ikine(P1,q0,[1,1,1,0,0,0]);
q2 = Rob.ikine(P2,q1,[1,1,1,0,0,0]);
q3 = Rob.ikine(P3,q2,[1,1,1,0,0,0]);
.
.
.
q232 = Rob.ikine(P232,q231,[1,1,1,0,0,0]);
```

Using the ‘[ikine](#)’ functions from the RTB toolbox, we calculate each inverse kinematics of each points (P1,P2,..Pn) by getting each configuration of the thetas through (q1,q2,...qn). The ‘[ikine](#)’ function also references the previous configuration of thetas, q(n-1) from the second parameter so that it could a smooth transformation from each adjacent ‘q’.

## SECTION 7

```
%% Calculation of forward kinematics (if needed)
% q1_P1 = Rob.fkine(q1);
```

The forward kinematics can also be calculated using the RTB toolbox via ‘[fkine](#)’ function and we will get the configuration of the x,y and z coordinates of the frame.

It is not being used to simulate the robot but it is included in the code for reference and to know that the toolbox can also calculate the forward kinematics.

## SECTION 8

```
% Trajectory path line of the robot
trajectorypath = dlmread('./Name-
Coordinates/lastname_traj.txt');
[nx,ny] = size(trajectorypath);

figure
hold on

for i = 1:nx-1
    v=[trajectorypath(i,:);trajectorypath(i+1,:)];
    plot3(v(:,1),v(:,2),v(:,3),'b');
    plot3(v(:,1),v(:,2),v(:,3),'bx')
end

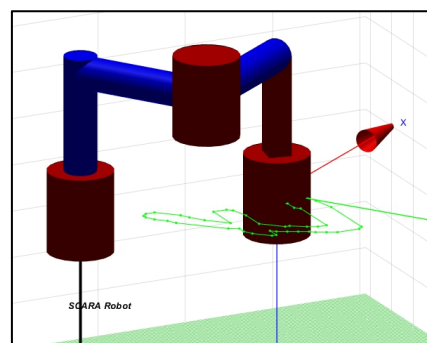
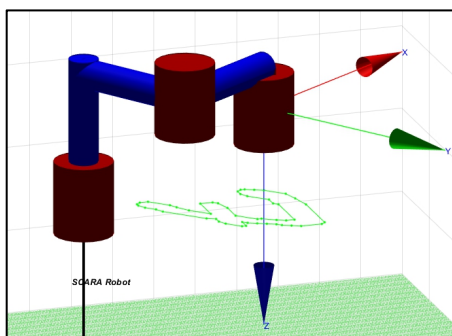
% Labelling certain points
text(8, 41, 30, 'START');
text(20, 80, 40, 'END');

% view settings of 3D plot
axis_matrix = [-40 170 -80 100 -100 100]; % [-x x -y y -z z]

axis(axis_matrix);
xlabel('X-Axis');
ylabel('Y-Axis');
zlabel('Z-Axis');

view(0,90);
%view(3);
```

The purpose of this code section is to plot the path line of the end effector of the robot follows. Just like in [section 4](#), 'trajectorypath' read from a text file to get its matrix data. The matrix data stored for 'trajectorypath' is different from the data stored in 'matrix' in [section 4](#). The reason behind this is to not obscure our view between the trajectory line and the robot. The rest of the code after the for loop is quite self-explanatory.



## SECTION 9

```
% Robot animation/simulation
% Time settings to either speed up/slow down the animation
t = (0: .1: 0.2)';

% All transformation animation

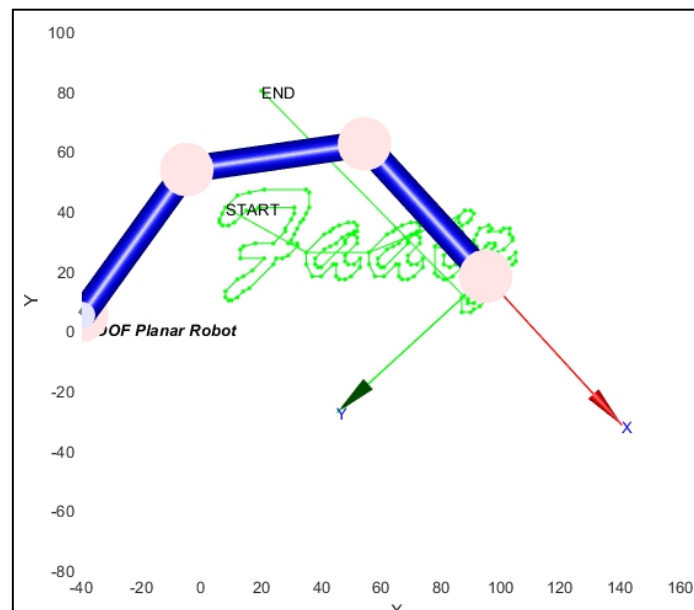
anim1 = jtraj(q0,q1,t); Rob.plot(anim1, 'workspace',
axis_matrix);
anim2 = jtraj(q1,q2,t); Rob.plot(anim2, 'workspace',
axis_matrix);
anim3 = jtraj(q2,q3,t); Rob.plot(anim3, 'workspace',
axis_matrix);
.
.
.
anim232 = jtraj(q231,q232,t); Rob.plot(anim232, 'workspace',
axis_matrix);
```

The last section of the code is where we specify the robot animations using the configuration of thetas we got from the inverse kinematics calculation using the RTB toolbox in [section 6](#). Using the 'jtraj' function, we get the animation from one point to another point by implementing  $q(n-1)$ ,  $q_n$  and a time vector. The time vector can be configured to speed up or slow down the animations. For this code, we use the same time vector for all our animations.



## ANIMATION/SIMULATION

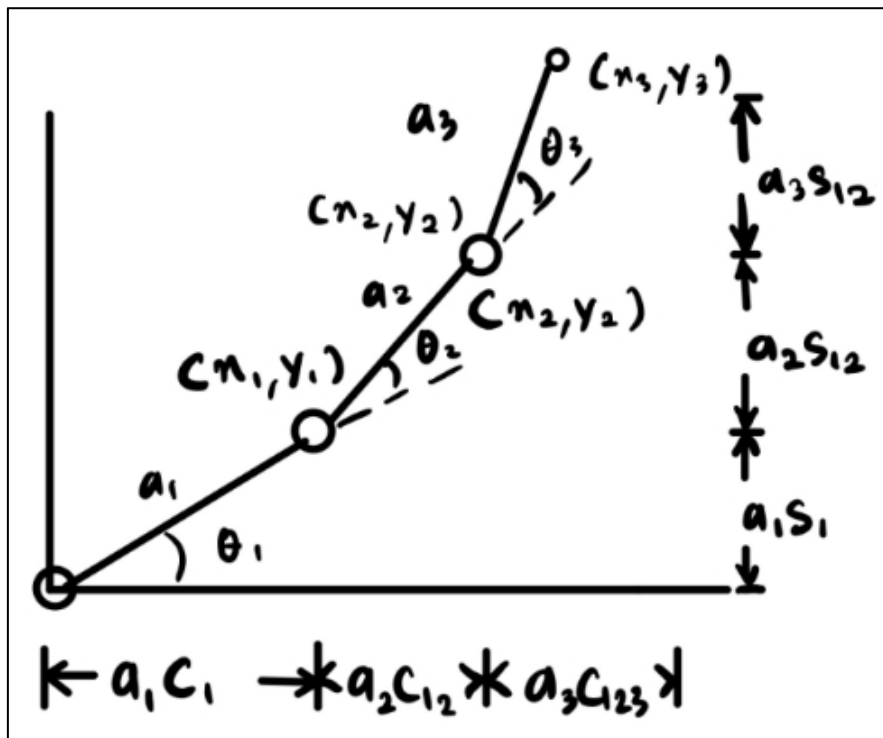
Refer to the video '[lastname\\_simulation.mp4](#)'.



For the trajectory pathline, I used the text file '[lastname\\_traj.txt](#)' that shows the path including the up and down movement when finishing a letter. The up and down movement was meant for the SCARA robot that I had previously used, but for the 3DOF Planar robot, the robot will not have any z-axis movement.

## INVERSE KINEMATICS COMPARISON BETWEEN EXCEL & RTB TOOLBOX

### Derivation of Equation of $\Theta_1$ , $\Theta_2$ and $\Theta_3$



$$A_1 A_2 A_3 A_4 = \begin{bmatrix} C\theta_{123} & S\theta_{123} & 0 & a_1 C\theta_1 + a_2 C\theta_{12} + a_3 C\theta_{123} \\ S\theta_{123} & -C\theta_{123} & 0 & a_1 S\theta_1 + a_2 S\theta_{12} + a_3 S\theta_{123} \\ 0 & 0 & -1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_1 A_2 A_3 A_4 = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forward Kinematics ( $\Theta_1, \Theta_2, \Theta_3 \rightarrow P_x, P_y, P_z$ ):

$$P_x = a_1 C\theta_1 + a_2 C\theta_{12} + a_3 C\theta_{123}$$

$$P_y = a_1 S\theta_1 + a_2 S\theta_{12} + a_3 S\theta_{123}$$

$$P_z = d_1$$

Inverse Kinematics ( $P_x, P_y, P_z > \Theta_1, \Theta_2, \Theta_3$ ):

By referring to the diagram above, we get:

$$x_2 = x_3 - a_3 C\theta_{123}$$

$$y_2 = y_3 - a_3 S\theta_{123}$$

$$\cos \theta_2 = \frac{x_2^2 + y_2^2 - a_1^2 - a_2^2}{2a_1 a_2}$$

$$x_2 = a_1 C\theta_1 + a_2 C\theta_{12}$$

$$x_2 = a_1 S\theta_1 + a_2 S\theta_{12}$$

$$x_2 = C\theta_1(a_1 + a_2 C\theta_2) - S\theta_1(a_2 S\theta_2)$$

$$y_2 = C\theta_1(a_2 S\theta_2) + S\theta_1(a_1 + a_2 C\theta_2)$$

$$C\theta_1 = \frac{(a_1 + a_2 C\theta_2)x_2 + a_2 S\theta_2 y_2}{x_2^2 + y_2^2}$$

$$S\theta_1 = \frac{(a_1 + a_2 C\theta_2)y_2 - a_2 S\theta_2 x_2}{x_2^2 + y_2^2}$$

$$\tan \theta_1 = S\theta_1 / C\theta_1$$

$$\phi = \theta_{123} = \theta_1 + \theta_2 + \theta_3$$

$$\theta_3 = \phi - \theta_1 - \theta_2$$

## Preview of excel file, 'Finding inverse kinematics.xlsx'

Note that this is for the SCARA robot that I had previously done.

a1	a2								
60	60								
		Coordinates	x	y		$\Theta_2$ in Radian	$\Theta_2$ in Degrees	$\Theta_1$ in Radian	$\Theta_1$ in Degrees
		1	8	41		1.308780837	74.98761825	0.72370515	41.46525069
		2	12	45		1.247264985	71.46301959	0.686561442	39.33707303
		3	16	46		1.217373491	69.75036312	0.627372744	35.94581041
		4	21	47		1.175889464	67.37350344	0.562654881	32.23775003
		5	35	47		1.054559227	60.42179293	0.403428561	23.11475386
		6	36	46		1.058069152	60.62289683	0.377715588	21.64150905
		7	36	41		1.126149707	64.5236253	0.287167338	16.45347646
		8	35	39		1.161400064	66.54332198	0.258699632	14.82239707
		9	34	37		1.194626177	68.44703805	0.230313477	13.19599019
		10	33	36		1.215448304	69.64005803	0.221124907	12.66952391
		11	32	34		1.245653171	71.37066943	0.192865338	11.05036988
		12	31	32		1.274101432	73.0006347	0.164219131	9.409063098
		13	30	30		1.300863531	74.53399005	0.134966398	7.733004977
		14	29	29		1.317829173	75.50604975	0.126483577	7.246975123
		15	32	29		1.291484257	73.99659722	0.090515301	5.186144707
		16	32	28		1.299710489	74.46792563	0.068974755	3.951962352
		17	29	25		1.348697093	77.27465126	0.037110579	2.12627957
		18	27	24		1.37156424	78.58484226	0.040860221	2.341118208
		19	25	24		1.38627885	79.42792735	0.071853408	4.116896995
		20	24	22		1.406164762	80.56730615	0.038864887	2.226794005

## FUTURE WORKS

Some possible future works that this project can expand to:

1. Using a different robot that can write a word in 3D space.
2. Writing the word with even smoother lines instead of just straight lines or translation motion
3. Instead of outlining the word, let the robot sketch or color the fill of the outlined word.

## REFERENCES

1. <https://www.mathworks.com/matlabcentral/fileexchange/54799-convert-image-pixels-to-xy-coordinates>
2. Corke, P. I. (2011). *Robotics, vision and control: Fundamental algorithms in MATLAB*. Springer. Retrieved from <http://petercorke.com/RTB/robot.pdf>
3. <https://github.com/ahmadfaalz/Project-Robotics-MCTE4352>

## APPENDICES

### ➤ ROBOTICS TOOLBOX FUNCTIONS

#### 1. Link

##### Robot manipulator Link class

A Link object holds all information related to a robot link such as kinematics parameters, rigid-body inertial parameters, motor and transmission parameters.

More info: <https://github.com/petercorke/robotics-toolbox-matlab/blob/master/Link.m>

#### 2. SerialLink

##### Serial-link robot class

A concrete class that represents a serial-link arm-type robot. The mechanism is described using Denavit-Hartenberg parameters, one set per joint.

More info: <https://github.com/petercorke/robotics-toolbox-matlab/blob/master/@SerialLink/SerialLink.m>

#### 3. transl

##### Create a translational transformation matrix

- $T = \text{transl}(x, y, z)$  is an  $SE(3)$  homogeneous transform ( $4 \times 4$ ) representing a pure translation of  $x$ ,  $y$  and  $z$ .
- $T = \text{transl}(p)$  is an  $SE(3)$  homogeneous transform ( $4 \times 4$ ) representing a translation of  $p=[x,y,z]$ . If  $p$  ( $M \times 3$ ) it represents a sequence and  $T$  ( $4 \times 4 \times M$ ) is a sequence of homogeneous transforms such that  $T(:, :, i)$  corresponds to the  $i$ 'th row of  $p$ .

#### 4. ikine

More info: <https://github.com/petercorke/robotics-toolbox-matlab/blob/master/@SerialLink/ikine.m>

## 5. fkine

More info: <https://github.com/petercorke/robotics-toolbox-matlab/blob/master/@SerialLink/fkine.m>

## 6. jtraj

Compute a joint space trajectory between two configurations

- $[q, qd, qdd] = \text{jtraj}(q_0, q_f, m)$  is a joint space trajectory  $q$  ( $m \times N$ ) where the joint coordinates vary from  $q_0$  ( $1 \times N$ ) to  $q_f$  ( $1 \times N$ ). A quintic (5th order) polynomial is used with default zero boundary conditions for velocity and acceleration. Time is assumed to vary from 0 to 1 in  $m$  steps. Joint velocity and acceleration can be optionally returned as  $qd$  ( $m \times N$ ) and  $qdd$  ( $m \times N$ ) respectively. The trajectory  $q$ ,  $qd$  and  $qdd$  are  $m \times N$  matrices, with one row per time step, and one column per joint.
- $[q, qd, qdd] = \text{jtraj}(q_0, q_f, T)$  as above but the trajectory length is defined by the length of the time vector  $T$  ( $m \times 1$ ).

More info: <https://github.com/petercorke/robotics-toolbox-matlab/blob/master/jtraj.m>

## 7. SerialLink.plot

### Graphical display and animation

`R.plot(q, options)` displays a graphical animation of a robot based on the kinematic model. A stick figure polyline joins the origins of the link coordinate frames. The robot is displayed at the joint angle  $q$  ( $1 \times N$ ), or if a matrix ( $M \times N$ ) it is animated as the robot moves along the  $M$ -point trajectory

More info: <https://github.com/petercorke/robotics-toolbox-matlab/blob/master/@SerialLink/plot.m>