**APPENDIX A: ANIMATION CODES OF STEWART (MATLAB)**

Matlab code for animation of Stewart Platform on a horizontal/ vertical circular trajectory (center of top plate).

clear all; close all;clc;

d=0.020;c=0.148;d1=0.020;c1=0.154;h=d1;

A=[];

B=[];

C=[];

for j=0:4:360

% for horizontal circle

r=0.04;

xCircle= [r\*cos(j\*3.14159/180)];

yCircle= [r\*sin(j\*3.14159/180)];

% for vertical circle

r=0.02;

xCircle=r\*cos((270+j)\*3.14159/180);

yCircle=0;

zCircle=r\*sin((270+j)\*3.14159/180)+0.04;

%top center co-ordinate in the frame of base center

xCTop=xCircle;

yCTop=yCircle;

zCTop=[h]; %height

%co-ordinates of the base plate in its own frame

xBase=[-c1/2, -c1/2-d/2, -d/2, d/2, c1/2+d/2, c1/2];

yBase=[-sqrt(3)\*(c1+2\*d)/6, -sqrt(3)\*(c1-d)/6, sqrt(3)\*(2\*c1+d)/6, sqrt(3)\*(2\*c1+d)/6, -sqrt(3)\*(c1-d)/6, -sqrt(3)\*(c1+2\*d)/6];

zBase=[0,0,0,0,0,0];

plate1=[xBase;yBase;zBase];

%co-ordinates of the top plate in its own frame

xTopFrame=[-d/2, -c/2-d/2, -c/2, +c/2,(c+d)/2, d/2];

yTopFrame=[-sqrt(3)\*(2\*c+d)/6, sqrt(3)\*(c-d)/6, sqrt(3)\*(c+2\*d)/6, sqrt(3)\*(c+2\*d)/6, sqrt(3)\*(c-d)/6, -sqrt(3)\*(2\*c+d)/6];

zTopFrame=[0,0,0,0,0,0];

%co-ordinates of the top plate in frame of the bottom plates center

xTop=xTopFrame+xCTop;

yTop=yTopFrame+yCTop;

zTop=zTopFrame+zCTop;

plate2=[xTop;yTop;zTop];

fill3(xBase,yBase,zBase,[1 0.6 0.78]); hold on;

grid on; axis([-.15 .15 -.2 .2 0 .025]);

fill3(xTop,yTop,zTop,'c'); hold on;

for i=1:6

plot3([xBase(i),xTop(i)],[yBase(i),yTop(i)],[zBase(i),zTop(i)],...

'MarkerFaceColor',[1 0.694117665290833 0.39215686917305],...

'MarkerEdgeColor',[0.749019622802734 0 0.749019622802734],...

'Marker','o',...

'LineWidth',3,...

'Color',[0 1 0]);

hold on;

xlabel('X-axis');

ylabel('Y-axis');

zlabel('Z-axis');

end

A=[A;xCTop];

B=[B;yCTop];

C=[C;zCTop];

plot3(A,B,C,'r');

hold off

pause(0.025);

end

Matlab code for animation of Stewart Platform on a Eight shaped trajectory (center of top plate).

clear all; close all;clc;

d=0.020;c=0.148;d1=0.020;c1=0.154;h=d1;

A=[];

B=[];

C=[];

% this loop makes the upper circle

for j=0:4:180

r=0.04;

xCircle=r\*cos(j\*3.14159/180)\*sin(j\*3.14159/180);

yCircle=r\*sin(j\*3.14159/180)\*sin(j\*3.14159/180);

%top center co-ordinate in the frame of base center

xCTop=xCircle;

yCTop=yCircle;

zCTop=[h]; %height

%co-ordinates of the base plate in its own frame

xBase=[-c1/2, -c1/2-d/2, -d/2, d/2, c1/2+d/2, c1/2];

yBase=[-sqrt(3)\*(c1+2\*d)/6, -sqrt(3)\*(c1-d)/6, sqrt(3)\*(2\*c1+d)/6, sqrt(3)\*(2\*c1+d)/6, -sqrt(3)\*(c1-d)/6, -sqrt(3)\*(c1+2\*d)/6];

zBase=[0,0,0,0,0,0];

%co-ordinates of the top plate in its own frame

xTopFrame=[-d/2, -c/2-d/2, -c/2, +c/2,(c+d)/2, d/2];

yTopFrame=[-sqrt(3)\*(2\*c+d)/6, sqrt(3)\*(c-d)/6, sqrt(3)\*(c+2\*d)/6, sqrt(3)\*(c+2\*d)/6, sqrt(3)\*(c-d)/6, -sqrt(3)\*(2\*c+d)/6];

zTopFrame=[0,0,0,0,0,0];

%co-ordinates of the top plate in frame of the bottom plates center

xTop=xTopFrame+xCTop;

yTop=yTopFrame+yCTop;

zTop=zTopFrame+zCTop;

fill3(xBase,yBase,zBase,[1 0.6 0.78]); hold on;

grid on; axis([-.15 .15 -.2 .2 0 .025]);

fill3(xTop,yTop,zTop,'c'); hold on;

for i=1:6

plot3([xBase(i),xTop(i)],[yBase(i),yTop(i)],[zBase(i),zTop(i)],...

'MarkerFaceColor',[1 0.694117665290833 0.39215686917305],...

'MarkerEdgeColor',[0.749019622802734 0 0.749019622802734],...

'Marker','o',...

'LineWidth',3,...

'Color',[0 1 0]);

hold on;

xlabel('X-axis');

ylabel('Y-axis');

zlabel('Z-axis');

end

A=[A;xCTop];

B=[B;yCTop];

C=[C;zCTop];

plot3(A,B,C,'y');

hold off;

pause(0.025);

end

% this loop makes the lower circle

for j=0:4:180

r=0.04;

xCircle=-r\*cos(j\*3.14159/180)\*sin(j\*3.14159/180);

yCircle=-r\*sin(j\*3.14159/180)\*sin(j\*3.14159/180);

%top center co-ordinate in the frame of base center

xCTop=xCircle;

yCTop=yCircle;

zCTop=[h]; %height

%co-ordinates of the base plate in its own frame

xBase=[-c1/2, -c1/2-d/2, -d/2, d/2, c1/2+d/2, c1/2];

yBase=[-sqrt(3)\*(c1+2\*d)/6, -sqrt(3)\*(c1-d)/6, sqrt(3)\*(2\*c1+d)/6, sqrt(3)\*(2\*c1+d)/6, -sqrt(3)\*(c1-d)/6, -sqrt(3)\*(c1+2\*d)/6];

zBase=[0,0,0,0,0,0];

%co-ordinates of the top plate in its own frame

xTopFrame=[-d/2, -c/2-d/2, -c/2, +c/2,(c+d)/2, d/2];

yTopFrame=[-sqrt(3)\*(2\*c+d)/6, sqrt(3)\*(c-d)/6, sqrt(3)\*(c+2\*d)/6, sqrt(3)\*(c+2\*d)/6, sqrt(3)\*(c-d)/6, -sqrt(3)\*(2\*c+d)/6];

zTopFrame=[0,0,0,0,0,0];

%co-ordinates of the top plate in frame of the bottom plates center

xTop=xTopFrame+xCTop;

yTop=yTopFrame+yCTop;

zTop=zTopFrame+zCTop;

fill3(xBase,yBase,zBase,[1 0.6 0.78]); hold on;

grid on; axis([-.15 .15 -.2 .2 0 .025]);

fill3(xTop,yTop,zTop,'c'); hold on;

for i=1:6

plot3([xBase(i),xTop(i)],[yBase(i),yTop(i)],[zBase(i),zTop(i)],...

'MarkerFaceColor',[1 0.694117665290833 0.39215686917305],...

'MarkerEdgeColor',[0.749019622802734 0 0.749019622802734],...

'Marker','o',...

'LineWidth',3,...

'Color',[0 1 0]);

hold on;

xlabel('X-axis');

ylabel('Y-axis');

zlabel('Z-axis');

end

A=[A;xCTop];

B=[B;yCTop];

C=[C;zCTop];

plot3(A,B,C,'y');

hold off;

pause(0.025);

end

**APPENDIX B: ARDUINO CODE FOR TRAJECTORIES**

Arduino code for Horizontal circular trajectory.

/\*

This program controls the motion of the STEWART PLATFORM and contain its inverse kinematics

\*/

// includes the library that Provides an option to use servo motors

#include <Servo.h>

//ref\_servo: defining sero's and the final name we will use to refer to our servos in physical structure

Servo ref\_servo[6];

//servo\_pin\_num: an array storing pin number that we are going to assign to the servo mototrs

const int servo\_pin\_num[] = {3,5,6,9,10,11};

//servo\_zero: not Extended position of each servo

const int servo\_zero[6]={1000,1000,1000,1000,1000,1000};

/\*

low\_mpoints: co-ordinate of lower mounting points of ref\_servo [x,y,z][ref\_servo]

top\_mpoints: co-ordinate of upper mounting points of ref\_servo [x,y,z][ref\_servo]

diff\_low\_up: diffrence in the co-ordinates of the lower and upper mounting point of ref\_servo [dx,dy,dz][ref\_servo]

dist\_low\_up: distance between upper and lower mounting points of the ref\_servo [ref\_servo distance]

\*/

float low\_mpoints[3][6], top\_mpoints[3][6], diff\_low\_up[3][6], dist\_low\_up[6];

/\*

s\_width\_top: width of the shorter side on the upper base in meters

l\_width\_top: width of the longer side on the upper base in meters

s\_width\_bottom: width of the shorter side on the lower base in meters

l\_width\_bottom: width of the longer side on the lower base in meters

i\_sep: initial sepration between the centers of the lower base and the upper base

\*/

float s\_width\_top=0.020 , l\_width\_top=0.148 , i\_sep=0.020, s\_width\_bottom=0.020, l\_width\_bottom=0.154;

//array storing the co-ordinates of the center points of the top base [x,y,z]

float top\_center[3]={0.0,0.0,0.13};

// I don't know a shit about what this is :P

float a1[3]={radians(0),radians(0),radians(0)};

// contains the length of the legs returned by in inverseKinematics function

float servo\_pos[6];

void setup(){

Serial.begin(9600);

/\*

Assigning co-ordinates of the lower/upper mounting points of servos

origin at the Geometrical center of the base plate

\*/

low\_mpoints[0][0] = -l\_width\_bottom/2;

low\_mpoints[1][0] = -sqrt(3)\*(l\_width\_bottom+2\*s\_width\_bottom)/6;

low\_mpoints[2][0] = 0;

low\_mpoints[0][1] = -l\_width\_bottom/2-s\_width\_bottom/2;

low\_mpoints[1][1] = -sqrt(3)\*(l\_width\_bottom-s\_width\_bottom)/6;

low\_mpoints[2][1] = 0;

low\_mpoints[0][2] = -s\_width\_bottom/2;

low\_mpoints[1][2] = sqrt(3)\*(2\*l\_width\_bottom+s\_width\_bottom)/6;

low\_mpoints[2][2] = 0;

low\_mpoints[0][3] = s\_width\_bottom/2;

low\_mpoints[1][3] = sqrt(3)\*(2\*l\_width\_bottom+s\_width\_bottom)/6;

low\_mpoints[2][3] = 0;

low\_mpoints[0][4] = l\_width\_bottom/2+s\_width\_bottom/2;

low\_mpoints[1][4] = -sqrt(3)\*(l\_width\_bottom-s\_width\_bottom)/6;

low\_mpoints[2][4] = 0;

low\_mpoints[0][5] = l\_width\_bottom/2;

low\_mpoints[1][5] = -sqrt(3)\*(l\_width\_bottom+2\*s\_width\_bottom)/6;

low\_mpoints[2][5] = 0;

top\_mpoints[0][0] = -s\_width\_top/2;

top\_mpoints[1][0] = -sqrt(3)\*(2\*l\_width\_top+s\_width\_top)/6;

top\_mpoints[2][0] = 0;

top\_mpoints[0][1] = -l\_width\_top/2-s\_width\_top/2;

top\_mpoints[1][1] = sqrt(3)\*(l\_width\_top-s\_width\_top)/6;

top\_mpoints[2][1] = 0;

top\_mpoints[0][2] = -l\_width\_top/2;

top\_mpoints[1][2] = sqrt(3)\*(l\_width\_top+2\*s\_width\_top)/6;

top\_mpoints[2][2] = 0;

top\_mpoints[0][3] = l\_width\_top/2;

top\_mpoints[1][3] = sqrt(3)\*(l\_width\_top+2\*s\_width\_top)/6;

top\_mpoints[2][3] = 0;

top\_mpoints[0][4] = l\_width\_top/2+s\_width\_top/2;

top\_mpoints[1][4] = sqrt(3)\*(l\_width\_top-s\_width\_top)/6;

top\_mpoints[2][4] = 0;

top\_mpoints[0][5] = s\_width\_top/2;

top\_mpoints[1][5] = -sqrt(3)\*(2\*l\_width\_top+s\_width\_top)/6;

top\_mpoints[2][5] = 0;

/\* this loop attaches our physical servos with the specific digital pins in arduino uno board

attach ref\_servo[i] to servo\_pin\_num[i] servo 1 -> pin 6

servo 1 -> pin 6

servo 2 -> pin 7

servo 3 -> pin 8

servo 4 -> pin 9

servo 5 -> pin 10

servo 6 -> pin 11

\*/

for(int i=0;i<6;i++){

ref\_servo[i].attach(servo\_pin\_num[i]);

}

setBack(); //calling function setBack to get it to the no extended state

}

void loop(){

/\*

this code is for circular trajectory of center

\*/

/\*

float r= 0.04 ; //radius of the circle

for(int i=0;i<20;i++){ //takes the center at (r,0,0.12)

top\_center[0]=r\*i/20;

inverseKinematics();

for(int i = 0; i < 6; i++){

ref\_servo[i].writeMicroseconds(servo\_pos[i]);

}

delay(200);

}

\*/

for(int i=0;i<360;i+=4){

top\_center[0]=0;//r\*cos(i\*3.14159/180); //x-coordinate of the center

top\_center[1]=0;//r\*sin(i\*3.14159/180); //y-coordinate of the center

top\_center[2]=0.13; //z-coordinate of the center

a1[0]=0;

a1[1]=0;

a1[2]=0;

inverseKinematics();

for(int i = 0; i < 6; i++){

ref\_servo[i].writeMicroseconds(servo\_pos[i]);

}

delay(200);

}

setBack();

}

/\*

This function takes all the servo's back to the initial stage

i.e a state in which none of the arms is extended

\*/

void setBack(){

for(int i=0;i<6;i++){ // This loop set's servo's back to not extended position

ref\_servo[i].writeMicroseconds(servo\_zero[i]);

}

delay(4000); //Gives servos time to get back to initial stage

}

// Not written by me, just modified it to work with my code

void inverseKinematics()

{

float rx1[3][3], ry1[3][3],rz1[3][3];

//Rotation Matrices

rx1[0][0]=1;

rx1[0][1]=0;

rx1[0][2]=0;

rx1[1][0]=0;

rx1[1][1]= cos(a1[0]);

rx1[1][2]= -sin(a1[0]);

rx1[2][0]=0;

rx1[2][1]= sin(a1[0]);

rx1[2][2]= cos(a1[0]);

ry1[0][0]=cos(a1[1]);

ry1[0][1]= 0;

ry1[0][2]=sin(a1[1]);

ry1[1][0]=0;

ry1[1][1]=1;

ry1[1][2]=0;

ry1[2][0]= -sin(a1[1]);

ry1[2][1]=0;

ry1[2][2]= cos(a1[1]);

rz1[0][0]=cos(a1[2]);

rz1[0][1]=-sin(a1[2]);

rz1[0][2]=0;

rz1[1][0]=sin(a1[2]);

rz1[1][1]= cos(a1[2]);

rz1[1][2]=0;

rz1[2][0]=0;

rz1[2][1]=0;

rz1[2][2]=1;

float r1[3][3],ri1[3][3];

int i=0;

int j=0;

int k=0;

for(i=0;i<3;i++)

{for (j=0;j<3;j++)

ri1[i][j]=0;

}

for(i=0;i<3;i++)

{for (j=0;j<3;j++)

{for(k=0;k<3;k++)

{ ri1[i][j]+=ry1[i][k]\*rx1[k][j];

}}}

i=0;

j=0;

k=0;

for(i=0;i<3;i++)

{for (j=0;j<3;j++)

r1[i][j]=0;

}

for(i=0;i<3;i++)

{for (j=0;j<3;j++)

{for(k=0;k<3;k++)

{r1[i][j]+=rz1[i][k]\*ri1[k][j];

}}}

/\*

Serial.print(r1[0][0]);

Serial.print(r1[0][1]);

Serial.print(r1[0][2]);

Serial.print(r1[1][0]);

Serial.print(r1[1][1]);

Serial.print(r1[1][2]);

Serial.print(r1[2][0]);

Serial.print(r1[2][1]);

Serial.print(r1[2][2]);

\*/

//reference frame

float ppf[3][6];

i=0;

j=0;

k=0;

for(i=0;i<3;i++)

{for (j=0;j<6;j++)

ppf[i][j]=0;

}

for(i=0;i<3;i++)

{for (j=0;j<6;j++)

{for(k=0;k<3;k++)

{ppf[i][j]+=r1[i][k]\*top\_mpoints[k][j];

}}

}

for(i=0;i<3;i++)

{for (j=0;j<6;j++)

ppf[i][j]=ppf[i][j]+top\_center[i];

// Serial.print(top\_center[i]);

// Serial.print(" \* ");

}

for(int i = 0; i < 6; i++)

{

diff\_low\_up[0][i] = low\_mpoints[0][i] - ppf[0][i];

diff\_low\_up[1][i] = low\_mpoints[1][i] - ppf[1][i];

diff\_low\_up[2][i] = low\_mpoints[2][i] - ppf[2][i];

dist\_low\_up[i] = sqrt(diff\_low\_up[0][i]\*diff\_low\_up[0][i] + diff\_low\_up[1][i]\*diff\_low\_up[1][i] + diff\_low\_up[2][i]\*diff\_low\_up[2][i]) ;

servo\_pos[i] = 1000 + ((dist\_low\_up[i]-0.130)/(0.184-0.130)) \* ( 2000 - 1000 ) ;

Serial.print(servo\_pos[i]);

Serial.print(" \* ");

}

Serial.print("\n");

}

Void loop’s code for eight shaped trajectory:

void loop(){

/\*

this code is to make a "8" type of trajectory of center

\*/

float r1= 0.025 ; //radius of the upper circle

float r2= 0.025; //radius of the lower circle

for(int th=0;th<360;th+=4){ //makes the upper circle in 90 steps

top\_center[0]=r1/2\*sin(th\*3.14159\*2/180);

top\_center[1]=r1\*sin(th\*3.14159/180)\*sin(th\*3.14159/180);

inverseKinematics();

for(int i = 0; i < 6; i++){

ref\_servo[i].writeMicroseconds(servo\_pos[i]);

}

delay(200);

}

for(int th=0;th<360;th+=4){ //makes the upper circle in 90 steps

top\_center[0]=-r1/2\*sin(th\*3.14159\*2/180);

top\_center[1]=-r1\*sin(th\*3.14159/180)\*sin(th\*3.14159/180);

inverseKinematics();

for(int i = 0; i < 6; i++){

ref\_servo[i].writeMicroseconds(servo\_pos[i]);

}

delay(200);

}

setBack(); }

Void loop’s code for vertical circular trajectory:

void loop(){

/\*

this code is for circular trajectory of center

\*/

float r= 0.02 ; //radius of the circle

for(int i=0;i<360;i+=4){

top\_center[0]= r\*cos((270+i)\*3.14159/180); //x-coordinate of the center

top\_center[1]= 0; //y-coordinate of the center

top\_center[2]= r\*sin((270+i)\*3.14159/180)+0.04; //z-coordinate of the center

a1[0]=0;

a1[1]=0;

a1[2]=0;

inverseKinematics();

for(int i = 0; i < 6; i++){

ref\_servo[i].writeMicroseconds(servo\_pos[i]);

}

delay(200);

}

setBack();

}

**APPENDIX C: FOUR BAR ANIMATION CODE**

For this part we have some function files that you need to create and save them all in one directory and then run (FB\_run\_file). It will ask you for inputs and omega provide that and animation will begin.

getMin.m

function [ minIndex , minNum ] = getMin( A )

minNum=A(1); minIndex=0;

for i=1:4

if le(A(i),minNum)

minNum=A(i);

minIndex=i;

end

end

end

getMax.m

function [ maxIndex , maxNum ] = getMax( A )

maxNum=A(1); maxIndex=0;

for i=1:4

if ge( A(i),maxNum)

maxNum=A(i);

maxIndex=i;

end

end

end

checkType.m

function [Type] = checkType( minIndex , maxIndex , A )

Atotal=A(1)+A(2)+A(3)+A(4);

if 2\*(A(minIndex) + A(maxIndex)) < Atotal

disp('shortest+longest < sum of other two')

if minIndex==1

disp('Crank-Crank')

Type=1;

%disp('Type-1 Enter "1" in FB\_Animation Function to get desired animation')

end

if minIndex==3

disp('Rocker-Rocker')

Type=2;

%disp('Type-2 Enter "2" in FB\_Animation Function to get desired animation')

end

if minIndex==2 || minIndex==4

disp('Crank-Rocker')

Type=3;

%disp('Type-3 Enter "3" in FB\_Animation Function to get desired animation')

end

end

if 2\*(A(minIndex) + A(maxIndex)) == Atotal

if A(1)==A(3) && A(2)==A(4)

disp('Parallel-Crank Four-Bar Linkage');

Type=4;

%disp('Type-4 Enter "4" in FB\_Animation Function to get desired animation')

end

end

if 2\*(A(minIndex) + A(maxIndex)) > Atotal

disp('Shortest + Longest > Sum of other two')

Type=5;

%disp('Type-5 Enter "5" in FB\_Animation Function to get desired animation')

end

end

FB\_Animation.m

%this function work well for double crank and crank rocker

function FB\_Animation( A,Omega,Type)

% A is a vector of 4th dimension

if Type==1 || Type==3

figure1 = figure('Color',[1 1 1]);

set(gcf,'color',[1 1 1])

axis([-50 50 -30 30]);

axis equal

for t=1:0.05:10

th2=t\*Omega;

a=A(3)\*A(3)-A(1)\*A(1)-A(4)\*A(4)-A(2)\*A(2)+2\*A(1)\*A(2)\*cos(th2)+2\*A(1)\*A(4)-2\*A(4)\*A(2)\*cos(th2);

b=4\*A(2)\*A(4)\*sin(th2);

c=A(3)\*A(3)-A(1)\*A(1)-A(4)\*A(4)-A(2)\*A(2)+2\*A(1)\*A(2)\*cos(th2)-2\*A(1)\*A(4)+2\*A(4)\*A(2)\*cos(th2);

th4=2\*atan((-b+sqrt(b\*b-4\*a\*c))/(2\*a));

x0=10 ; y0=0;

x1=10+A(2)\*cos(th2); y1=A(2)\*sin(th2);

x2=10+A(1)+A(4)\*cos(th4); y2=A(4)\*sin(th4);

x3=10+A(1); y3=0;

cla

hold on

plot([x0 x1],[y0 y1],'LineWidth',2,'Color',[0 0.749019622802734 0.749019622802734]);

plot([x1 x2],[y1 y2], 'MarkerFaceColor',[0 0.749019622802734 0.749019622802734],...

'MarkerEdgeColor',[0.0784313753247261 0.168627455830574 0.549019634723663],...

'MarkerSize',5,...

'Marker','o',...

'LineWidth',2,...

'Color',[0.0588235296308994 0.874509811401367 0.470588237047195]);

plot([x2 x3],[y2 y3],'LineWidth',2,...

'Color',[0.0431372560560703 0.517647087574005 0.780392169952393]);

plot([x0 x3],[y0 y3],...

'MarkerFaceColor',[0.24705882370472 0.24705882370472 0.24705882370472],...

'MarkerEdgeColor',[0.24705882370472 0.24705882370472 0.24705882370472],...

'Marker','v',...

'LineWidth',2,...

'Color',[0.87058824300766 0.490196079015732 0]);

pause(0.05);

hold off

end

end

if Type==4

figure1 = figure('Color',[1 1 1]);

set(gcf,'color',[1 1 1])

axis([-5 15 -10 10]);

axis equal

for t=1:0.05:10

th2=t\*Omega;

th4=th2;

x0=10 ; y0=0;

x1=10+A(2)\*cos(th2); y1=A(2)\*sin(th2);

x2=10+A(1)+A(4)\*cos(th4); y2=A(4)\*sin(th4);

x3=10+A(1); y3=0;

cla

hold on

plot([x0 x1],[y0 y1],'LineWidth',2,'Color',[0 0.749019622802734 0.749019622802734]);

plot([x1 x2],[y1 y2], 'MarkerFaceColor',[0 0.749019622802734 0.749019622802734],...

'MarkerEdgeColor',[0.0784313753247261 0.168627455830574 0.549019634723663],...

'MarkerSize',5,...

'Marker','o',...

'LineWidth',2,...

'Color',[0.0588235296308994 0.874509811401367 0.470588237047195]);

plot([x2 x3],[y2 y3],'LineWidth',2,...

'Color',[0.0431372560560703 0.517647087574005 0.780392169952393]);

plot([x0 x3],[y0 y3],...

'MarkerFaceColor',[0.24705882370472 0.24705882370472 0.24705882370472],...

'MarkerEdgeColor',[0.24705882370472 0.24705882370472 0.24705882370472],...

'Marker','v',...

'LineWidth',2,...

'Color',[0.87058824300766 0.490196079015732 0]);

pause(0.05);

hold off

end

end

if Type==5

disp('I have not studied this case')

end

if Type==2

figure1 = figure('Color',[1 1 1]);

set(gcf,'color',[1 1 1])

axis([-5 15 -10 10]);

axis equal

H=[];I=[];

for t=0:0.05:10

th2=t\*Omega;

a=A(3)\*A(3)-A(1)\*A(1)-A(4)\*A(4)-A(2)\*A(2)+2\*A(1)\*A(2)\*cos(th2)+2\*A(1)\*A(4)-2\*A(4)\*A(2)\*cos(th2);

b=4\*A(2)\*A(4)\*sin(th2);

c=A(3)\*A(3)-A(1)\*A(1)-A(4)\*A(4)-A(2)\*A(2)+2\*A(1)\*A(2)\*cos(th2)-2\*A(1)\*A(4)+2\*A(4)\*A(2)\*cos(th2);

th4=2\*atan((-b+sqrt(b\*b-4\*a\*c))/(2\*a));

x1=10+A(2)\*cos(th2); y1=A(2)\*sin(th2);

x2=10+A(1)+A(4)\*cos(th4); y2=A(4)\*sin(th4);

if A(3)== sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1))

H=[H;th2];

I=[I;th4];

end

end

for th2=H(1,1):H(end,1)

a=A(3)\*A(3)-A(1)\*A(1)-A(4)\*A(4)-A(2)\*A(2)+2\*A(1)\*A(2)\*cos(th2)+2\*A(1)\*A(4)-2\*A(4)\*A(2)\*cos(th2);

b=4\*A(2)\*A(4)\*sin(th2);

c=A(3)\*A(3)-A(1)\*A(1)-A(4)\*A(4)-A(2)\*A(2)+2\*A(1)\*A(2)\*cos(th2)-2\*A(1)\*A(4)+2\*A(4)\*A(2)\*cos(th2);

th4=2\*atan((-b+sqrt(b\*b-4\*a\*c))/(2\*a));

x0=10 ; y0=0;

x1=10+A(2)\*cos(th2); y1=A(2)\*sin(th2);

x2=10+A(1)+A(4)\*cos(th4); y2=A(4)\*sin(th4);

x3=10+A(1); y3=0;

cla

hold on

plot([x0 x1],[y0 y1],'LineWidth',2,'Color',[0 0.749019622802734 0.749019622802734]);

plot([x1 x2],[y1 y2], 'MarkerFaceColor',[0 0.749019622802734 0.749019622802734],...

'MarkerEdgeColor',[0.0784313753247261 0.168627455830574 0.549019634723663],...

'MarkerSize',5,...

'Marker','o',...

'LineWidth',2,...

'Color',[0.0588235296308994 0.874509811401367 0.470588237047195]);

plot([x2 x3],[y2 y3],'LineWidth',2,...

'Color',[0.0431372560560703 0.517647087574005 0.780392169952393]);

plot([x0 x3],[y0 y3],...

'MarkerFaceColor',[0.24705882370472 0.24705882370472 0.24705882370472],...

'MarkerEdgeColor',[0.24705882370472 0.24705882370472 0.24705882370472],...

'Marker','v',...

'LineWidth',2,...

'Color',[0.87058824300766 0.490196079015732 0]);

pause(0.005);

hold off

end

disp('Work Going On , not completed')

end

FB\_run\_file.m

clear all;close all;clc;

prompt='Input the four lengths of Four bar Mechanisn as a vector [fixedLink , InputLink , Coupler , EndEffector]= ' ;

A=input(prompt);

prompt='Input Omega= ' ;

Omega=input(prompt);

[ minIndex , minNum ] = getMin( A );

[ maxIndex , maxNum ] = getMax( A );

[Type] = checkType( minIndex , maxIndex , A );

FB\_Animation(A,Omega,Type);