**SIBTANU RAHA**

**Final Project-Intelligent robotics**

**DESIGN A HUMAN ROBOT INTERFACE**

**Brief Introduction:**

The project is a simulation of a 3R planar manipulator. The program is designed to teach a non-programmer to interact with the virtual robot using keyboard controls. The aim of the project is to introduce the characteristics of a 3-link planar manipulator to a person with little experience in robotics. The user can control the robot and see whether it can perform the desired actions (for e.g. picking up an object and placing it elsewhere).

**Model and Approach:**

The following points will summarize the approach followed in the project:

* The manipulator consists of 3 links. Lines have been drawn as links in the program.
* For inverse kinematics the algebraic approach has been followed. The user enters the x, y and phi from console and the end effector positions itself at the desired location.
* For animation, the angles in the joint space have been made to move by a certain degree at a time till the robot reaches the desired location.
* For straight line motion a number of knot points have been predicted as the user moves and the manipulator is constrained to follow a straight line path by applying instantaneous inverse kinematics

**How to use the program:**

Once you have the program running you can see the GUI window with the manipulator in it.

You can control the robot only through your keyboard. The GUI contains the instructions but I am repeating them here with more clarity.

**Instructions:**

* **H** :to bring the robot to an upright position
* **Space :** If the robot is moving you can pause/resume motion anytime
* **V:** Each time you press V the location of the end effector is saved. You can save up to 3 locations.
* **1,2,3:** You can execute the paths to the 3 saved locations using 1,2,3 buttons
* **J:** Keep pressing J to move the robot along the end effector axis.(straight line motion)
* **W,A,S,D:** Keep pressing these move the gripper along the base axis.(straight line motion)
* **P:** If you press P you will be prompted in the console to enter the x, y and phi values. The robot will animate to the entered location with an elbow down solution
* **Q:** Performs the same function as P with an elbow up solution.
* **E:** Press E to enter the 3 joint angle values in the console and see the robot move to the desired position
* **X, Y, Z:** Press X to select link 1. Then press the Right/Left Arrow to rotate it. Similarly Press Y to select joint 2 and use the arrows for manual rotation. Joint 3 can be selected by pressing Z and then rotated by pressing the arrows.
* **G:** Press G to open/close gripper. When open the gripper is red. If you move your gripper close to the object and it is open the object is picked up. You can ungrasp the object anytime by pressing G again.
* **Esc:** To exit.

**Points to Noted:**

* The origin of the window is at the bottom left corner.
* The size of the window is 800\*800
* The base of the robot is positioned at coordinates (400,400) i.e. the center of the window.
* Each link is 50 units long.

**Equations:**

I will briefly go over the equations I have followed during the implementation.

Method:

void inverse\_kinematics1(float x, float y, float phi)**//input parameters: End effector location+phi**

{

float the1,the2,the3;

float c2,s2;

float k1,k2;

float xw\_temp=0,yw\_temp=0,xw=0,yw=0;

xw=x-50\*cos((phi\*pi)/180)-400; **//Getting wrist coordinates from end effector location**

yw=y-50\*sin((phi\*pi)/180)-400**;//Getting wrist coordinates from end effector location**

c2=((xw\*xw)+(yw\*yw)-5000)/5000;

if((c2\*c2)>1)

inverse\_error\_flag1=1;//checking if soln exists

s2=sqrt(1-(c2\*c2));

the2=atan2(s2,c2)\*(180/pi); **//Calculating theta2**

k1=50+50\*c2;

k2=50\*s2;

the1=atan2(yw,xw)\*(180/pi)-atan2(k2,k1)\*(180/pi);//Calculating theta1

the3=phi-the1-the2; **//Calculating theta3**

if((c2\*c2)>1){ **//Updating the joint angles**

inverse\_error\_flag1=1;

sol3[0]=theta1;

sol3[1]=theta2;

sol3[2]=theta3;

}

else{ **// Updating the joint angles**

sol3[0]=GetFloatPrecision(the1,1);

sol3[1]=GetFloatPrecision(the2,1);

sol3[2]=GetFloatPrecision(the3,1);

}

}

As you can see above the method follows the algebraic approach to get the joint angles from the end effector location and phi.

**SOURCE CODE(Language used: C++, Library: Opengl)**

/\*

Planar 3R control(IR project)

Sibtanu Raha

\*/

#include<cstdlib>

#include <math.h> // For math routines (such as sqrt & trig).

#include <stdio.h>

//#include<windows.h>

#include<iostream>

#include<fstream>

#include<conio.h>

#include <GL/glut.h>

#include <GL/gl.h>

#include <GL/glui.h>

//#include<unistd.h>

#include<string>

#include<limits>

#include <GL/glu.h>

#include <time.h>

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*GLOBAL VARIABLES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int vert[8];

int objx=330,objy=400;

int tempx,tempy;

float i\_phi;

int pts[100];

int ptsu[100];

int ptsd[100];

int ptsr[100];

int ptsl[100];

int xstore[3];

int ystore[3];

int pstore[3];

int ctr1=0,ctr2=1,ctru1=0,ctru2=1,ctrd1=0,ctrd2=1,ctrl1=0,ctrl2=1,ctrr1=0,ctrr2=1,sind=0;

char string1[1000];

float b1,b2,b3;//angle each link makes with the base

float theta1=0, theta2=0, theta3=0;// angle each link makes with the previous link

float pi=3.14;

float link\_length=50;

float sol1[3];

float sol2[3];

float sol3[3];

float sol4[3];

float xx,yy,pp;

int inverse\_error\_flag1=0,inverse\_error\_flag2=0;

bool straight\_line\_motion=false, set\_ctr=true,jagged\_motion=true;

bool set\_down=true,set\_up=true,set\_left=true,set\_right=true, set=true;

bool store\_flag=false, runpath=false, one=false,two=false,three=false;

bool attachobject=false;

int goff=1;

float dist;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION DECLARATIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void drawBitmapText(char \*string,float x,float y,float z); //To render text on opengl window

void renderBitmapString(float x, float y, void \*font, char \*string);//To render text plus changing variables on opengl window

float GetFloatPrecision(float value, float precision);//to round off floating decimals to the required precision

void setPrecision();//to set precision for different variables

void reshapenew(int width, int height);

void inverse\_kinematics(int x, int , float phi);

void reset();

void setverts\_down();

void setverts\_up();

void setverts\_right();

void setverts\_left();

void setverts();

void run();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int s=0;

int i1,i2,i3;

int moving=1, elbow\_down=0,elbow\_up=0,inverse\_elbow\_up,inverse\_elbow\_down;

int A=0,B=0,C=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION TO ROUND OFF FLOATING DECIMALS TO PRESICION 1\*\*\*\*\*\*\*\*\*\*\*

float GetFloatPrecision(float value, float precision)

{

return (floor((value \* pow(10, precision) + 0.5)) / pow(10, precision));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESETS ALL FLAGS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void reset()

{

A=B=C=0;

s=0;

inverse\_error\_flag1=0;

inverse\_error\_flag2=0;

elbow\_up=0;

elbow\_down=0;

inverse\_elbow\_up=0;

inverse\_elbow\_down=0;

set\_down=true;

set\_up=true;

set\_right=true;

set\_left=true;

set=true;

straight\_line\_motion=false;

store\_flag=false;

runpath=false;

one=false,two=false,three=false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setverts\_down()

{

if(set\_down){

tempx=vert[6];

tempy=vert[7];

for(int i=0;i<=64;i+=2)

{

ptsd[i]=tempx;

}

for(int i=1;i<=65;i+=2)

{

ptsd[i]=tempy;

tempy=tempy-5;

}

set\_down=false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setverts\_up()

{

if(set\_up){

tempx=vert[6];

tempy=vert[7];

for(int i=0;i<=64;i+=2)

{

ptsu[i]=tempx;

}

for(int i=1;i<=65;i+=2)

{

ptsu[i]=tempy;

tempy=tempy+5;

}

set\_up=false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setverts\_left()

{

if(set\_left){

tempx=vert[6];

tempy=vert[7];

for(int i=0;i<=64;i+=2)

{

ptsl[i]=tempx;

tempx=tempx-5;

}

for(int i=1;i<=65;i+=2)

{

ptsl[i]=tempy;

}

set\_left=false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setverts\_right()

{

if(set\_right){

tempx=vert[6];

tempy=vert[7];

for(int i=0;i<=64;i+=2)

{

ptsr[i]=tempx;

tempx=tempx+5;

}

for(int i=1;i<=65;i+=2)

{

ptsr[i]=tempy;

}

set\_right=false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setverts()

{

if(set){

tempx=vert[6];

tempy=vert[7];

for(int i=0;i<=64;i+=2)

{

pts[i]=tempx;

tempx=tempx+50\*cos((b3\*pi)/180)\*0.25;

}

for(int i=1;i<=65;i+=2)

{

pts[i]=tempy;

tempy=tempy+50\*sin((b3\*pi)/180)\*0.25;

}

set=false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void init(void)

{

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION TO ROUND OFF JOINT ANGLES\*\*\*\*\*\*\*\*

void setPrecision()

{

theta1=GetFloatPrecision(theta1,1);

theta2=GetFloatPrecision(theta2,1);

theta3=GetFloatPrecision(theta3,1);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void screen\_to\_ndc (int x, int y, float \*ndc\_x, float \*ndc\_y) {

// assume the default ndc (-1 to 1 in X and Y)

\*ndc\_x = 2.\*x/800 - 1.;

\*ndc\_y = -2.\*y/800 +1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void handleMouseEvent(int button, int state, int x, int y)

{

switch(state) {

case GLUT\_DOWN:

if (button == GLUT\_LEFT\_BUTTON)

{

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void mouseEvent(int button, int state, int x, int y) {

// all interaction via this function

handleMouseEvent(button, state, x, y);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void inverse\_kinematics1(float x, float y, float phi)//WRIST COORDINATES

{

float the1,the2,the3;

float c2,s2;

float k1,k2;

float xw\_temp=0,yw\_temp=0,xw=0,yw=0;

xw=x-50\*cos((phi\*pi)/180)-400;

yw=y-50\*sin((phi\*pi)/180)-400;

c2=((xw\*xw)+(yw\*yw)-5000)/5000;

if((c2\*c2)>1)

inverse\_error\_flag1=1;

s2=sqrt(1-(c2\*c2));

the2=atan2(s2,c2)\*(180/pi);

k1=50+50\*c2;

k2=50\*s2;

the1=atan2(yw,xw)\*(180/pi)-atan2(k2,k1)\*(180/pi);

the3=phi-the1-the2;

if((c2\*c2)>1){

inverse\_error\_flag1=1;

sol3[0]=theta1;

sol3[1]=theta2;

sol3[2]=theta3;

}

else{

sol3[0]=GetFloatPrecision(the1,1);

sol3[1]=GetFloatPrecision(the2,1);

sol3[2]=GetFloatPrecision(the3,1);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void inverse\_kinematics2(float x, float y, float phi)//WRIST COORDINATES

{

float the1,the2,the3;

float c2,s2;

float k1,k2;

float xw\_temp=0,yw\_temp=0,xw=0,yw=0;

xw=x-50\*cos((phi\*pi)/180)-400;

yw=y-50\*sin((phi\*pi)/180)-400;

c2=((xw\*xw)+(yw\*yw)-5000)/5000;

if((c2\*c2)>1)

inverse\_error\_flag2=1;

s2=-sqrt(1-(c2\*c2));

the2=atan2(s2,c2)\*(180/pi);

k1=50+50\*c2;

k2=50\*s2;

the1=atan2(yw,xw)\*(180/pi)-atan2(k2,k1)\*(180/pi);

the3=phi-the1-the2;

if((c2\*c2)>1){

inverse\_error\_flag2=1;

sol4[0]=theta1;

sol4[1]=theta2;

sol4[2]=theta3;

}

else{

sol4[0]=GetFloatPrecision(the1,1);

sol4[1]=GetFloatPrecision(the2,1);

sol4[2]=GetFloatPrecision(the3,1);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void renderBitmapString(float x, float y, void \*font, char \*string){ // Function to render Score on screen

char \*c;

glRasterPos2f(x,y);

for (c=string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void run()

{

if(theta1!=sol3[0]){

if(theta1<sol3[0] )

theta1=theta1+0.1;

else

theta1=theta1-0.1;

}

if(theta2!=sol3[1] ){

if(theta2<sol3[1])

theta2=theta2+0.1;

else

theta2=theta2-0.1;

}

if(theta3!=sol3[2] ){

if(theta3<sol3[2] )

theta3=theta3+0.1;

else

theta3=theta3-0.1;

}

//cout<<"run"<<endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void resizeWindow(int w, int h)

{

double aspectRatio;

glViewport( 0, 0, w, h );

w = (w==0) ? 1 : w;

h = (h==0) ? 1 : h;

aspectRatio = (double)w / (double)h;

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

gluPerspective( 15.0, aspectRatio, 25.0, 45.0 );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void drawscene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

//glMatrixMode (GL\_PROJECTION); // Tell opengl that we are doing project matrix work

glLoadIdentity(); // Clear the matrix

glMatrixMode(GL\_MODELVIEW); // Tell opengl that we are doing model matrix work. (drawing)

glLoadIdentity(); // Clear the model matrix

glOrtho(0.0, 800.0, 0.0, 800.0, 0.0, 1.0);

setPrecision();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LINK 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glLineWidth(40);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

//glVertex3f(350,700,0);

glVertex3f(400,400,0);

glVertex3f(vert[2],vert[3],0);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LINK 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glLineWidth(40);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINES);

glVertex3f(vert[2],vert[3],0);

//glVertex3f(350,550,0);

glVertex3f(vert[4],vert[5],0);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LINK 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glLineWidth(40);

glColor3f(0.0, 1.0, .0);

glBegin(GL\_LINES);

glVertex3f(vert[4],vert[5],0);

//glVertex3f(350,550,0);

glVertex3f(vert[6],vert[7],0);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BASE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glPointSize(10);

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_POINTS);

glVertex2f(400,400);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OBJECT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glPointSize(25);

glColor3f(0.5, 0.2, 0.1);

glBegin(GL\_POINTS);

glVertex2f(objx,objy);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Joint 1 hinge display\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glPointSize(10);

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_POINTS);

glVertex2f( vert[2], vert[3]);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Joint 2 hinge display\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glPointSize(10);

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_POINTS);

glVertex2f( vert[4], vert[5]);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glPointSize(10);

if(goff==1)

glColor3f(0.0, 0.0, 0.0);

else

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POINTS);

glVertex2f( vert[6], vert[7]);

glEnd();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glColor3f(1,1,1);

drawBitmapText("1. Press Space for Pause/Restart",30,770,0);

drawBitmapText("2. Press H to bring robot to initial position",30,750,0);

drawBitmapText("3. Press E to enter joint angles (JMOVE)",30,730,0);

drawBitmapText("4. Press X,Y,Z to select joints 1,2,3 respectively",30,710,0);

drawBitmapText(" and use Right Arrow/Left Arrow to rotate the joint(JJOG)",30,690,0);

drawBitmapText("5. Press P to enter x,y and phi(elbow down solution)(MOVETO)",30,670,0);

drawBitmapText("6. Press Q to enter x,y and phi(elbow up solution)(MOVETO)",30,650,0);

drawBitmapText("7. Press W,S,A,D for traight line motion along base axis(JOG)",30,630,0);

drawBitmapText("8. Press V to store current gripper location",30,610,0);

drawBitmapText("9. Press 1,2,3 to execute the path to 3 saved locations ",30,590,0);

drawBitmapText("10.Press J for straight line motion along hand axis(JOG)",30,570,0);

drawBitmapText("11.Press G to Open/Close Gripper and grasp/ungrasp object",30,550,0);

glColor3f(1,0,0);

if(goff==1)

drawBitmapText("Gripper Closed",640,110,0);

else

drawBitmapText("Gripper Open",640,110,0);

if(moving!=1)

drawBitmapText(" Movement Paused",640,90,0);

if(store\_flag)

drawBitmapText(" Location Saved",640,70,0);

if(inverse\_error\_flag1==1 || inverse\_error\_flag2==1 && straight\_line\_motion==false)

drawBitmapText(" Solution Unreachable",610,50,0);

if(one)

drawBitmapText(" Executing Path 1",620,30,0);

if(two)

drawBitmapText(" Executing Path 2",620,30,0);

if(three)

drawBitmapText(" Executing Path 3",620,30,0);

drawBitmapText("(0,0)",0,0,0);

glColor3f(0,1,1);

sprintf(string1, "Theta 1: %f ",theta1); //%d is for integers

renderBitmapString(620, 760, GLUT\_BITMAP\_HELVETICA\_12 , string1);

sprintf(string1, "Theta 2: %f ",theta2);

renderBitmapString(620, 740, GLUT\_BITMAP\_HELVETICA\_12 , string1);

sprintf(string1, "Theta 3: %f ",theta3);

renderBitmapString(620, 720, GLUT\_BITMAP\_HELVETICA\_12 , string1);

sprintf(string1, "Phi : %f ",b3);

renderBitmapString(620, 700, GLUT\_BITMAP\_HELVETICA\_12 , string1);

sprintf(string1, "End Effector: %d, %d",vert[6],vert[7]);

renderBitmapString(620, 670, GLUT\_BITMAP\_HELVETICA\_12 , string1);

//sprintf(string1, ", %d ",vert[7]);

//renderBitmapString(715, 670, GLUT\_BITMAP\_HELVETICA\_12 , string1);

sprintf(string1, "Wrist: %d, %d",vert[4],vert[5]);

renderBitmapString(620, 650, GLUT\_BITMAP\_HELVETICA\_12 , string1);

sprintf(string1, "Object: %d, %d",objx,objy);

renderBitmapString(620, 630, GLUT\_BITMAP\_HELVETICA\_12 , string1);

//sprintf(string1, ", %d ",vert[5]);

//renderBitmapString(715, 630, GLUT\_BITMAP\_HELVETICA\_12 , string1);

glFlush();

b1=theta1; b2=theta1+theta2; b3=theta1+theta2+theta3;

dist= (vert[6]-objx)\*(vert[6]-objx)+(vert[7]-objy)\*(vert[7]-objy);

if(ctr1>64)

ctr1=0;

if(ctr2>65)

ctr2=1;

if(ctru1>64)

ctru1=0;

if(ctru2>65)

ctru2=1;

if(ctrd1>64)

ctrd1=0;

if(ctrd2>65)

ctrd2=1;

if(ctrl1>64)

ctrl1=0;

if(ctrl2>65)

ctrl2=1;

if(ctrr1>64)

ctrr1=0;

if(ctrr2>65)

ctrr2=1;

if(sind>2)

sind=0;

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LINK\_ANGLE 1 UPDATE\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vert[2]=400+link\_length\*cos((b1\*pi)/180);// New point a+rcos(theta) DEGREES TO RADIANS

vert[3]=400+link\_length\*sin((b1\*pi)/180);// b+rsin(theta)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LINK\_ANGLE 2 UPDATE\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vert[4]=vert[2]+link\_length\*cos((b2\*pi)/180);// New point a+rcos(theta)

vert[5]=vert[3]+link\_length\*sin((b2\*pi)/180);// b+rsin(theta)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LINK\_ANGLE 3 UPDATE\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vert[6]=vert[4]+link\_length\*cos((b3\*pi)/180);// New point a+rcos(theta)

vert[7]=vert[5]+link\_length\*sin((b3\*pi)/180);// b+rsin(theta)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(theta1>360 || theta1<-360)

theta1=0;

if(theta2>360 || theta2<-360)

theta2=0;

if(theta3>360 || theta3<-360)

theta3=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if((s==1) && (moving==1))

{

if(theta1!=i1 )

{

if(theta1<i1 )

theta1=theta1+0.1;

else

theta1=theta1-0.1;

}

if(theta2!=i2 )

{

if(theta2<i2)

theta2=theta2+0.1;

else

theta2=theta2-0.1;

}

if(theta3!=i3 )

{

if(theta3<i3 )

theta3=theta3+0.1;

else

theta3=theta3-0.1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if((inverse\_elbow\_down==1) && (moving==1) && (inverse\_error\_flag1!=1)){

if(theta1!=sol3[0]){

if(theta1<sol3[0] )

theta1=theta1+0.1;

else

theta1=theta1-0.1;

}

if(theta2!=sol3[1] ){

if(theta2<sol3[1])

theta2=theta2+0.1;

else

theta2=theta2-0.1;

}

if(theta3!=sol3[2] ){

if(theta3<sol3[2] )

theta3=theta3+0.1;

else

theta3=theta3-0.1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if((inverse\_elbow\_up==1) && (moving==1) && (inverse\_error\_flag2!=1)){

if(theta1!=sol4[0]){

if(theta1<sol4[0] )

theta1=theta1+0.1;

else

theta1=theta1-0.1;

}

if(theta2!=sol4[1] ){

if(theta2<sol4[1])

theta2=theta2+0.1;

else

theta2=theta2-0.1;

}

if(theta3!=sol4[2] ){

if(theta3<sol4[2] )

theta3=theta3+0.1;

else

theta3=theta3-0.1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if((moving==1)){

if(one)

run();

if(two)

run();

if(three)

run();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if((goff!=1) && (dist<150))

attachobject=true;

if(goff==1)

attachobject=false;

if(attachobject){

objx=vert[6];

objy=vert[7];

}

glutPostRedisplay();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void drawBitmapText(char \*string,float x,float y,float z)

{

char \*c;

glRasterPos3f(x, y,z);

for (c=string; \*c != '\0'; c++)

{

glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13 , \*c);

//glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15 , \*c);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void mySpecialKeyFunc( int key, int x, int y )

{

switch ( key ) {

case GLUT\_KEY\_RIGHT:

if(A==1)

theta1=theta1-2;

else if(B==1)

theta2=theta2-2;

else if(C==1)

theta3=theta3-2;

break;

case GLUT\_KEY\_LEFT:

if(A==1)

theta1=theta1+2;

if(B==1)

theta2=theta2+2;

if(C==1)

theta3=theta3+2;

break;

}

glutPostRedisplay();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void myKeyboardFunc( unsigned char key, int x, int y )

{

switch ( key ) {

case 32:

//reset();

moving=-moving;

glutPostRedisplay();

break;

case 'h':

reset();

theta1=90.0f;

theta2=0.0f;

theta3=0.0f;

glutPostRedisplay();

break;

case 'e':

reset();

s=1;

while ((std::cout << "Enter Theta1: "<<endl) && (!(std::cin >> i1) || i1 <-360 || i1 > 360))

{

std::cout << "Invalid input! Please enter a value between -360 and 360"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

while ((std::cout << "Enter Theta2: "<<endl) && (!(std::cin >> i2) || i2 <-360 || i2 > 360))

{

std::cout << "Invalid input! Please enter a value between -360 and 360"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

while ((std::cout << "Enter Theta3: "<<endl) && (!(std::cin >> i3) || i3 <-360 || i3 > 360))

{

std::cout << "Invalid input! Please enter a value between -360 and 360"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

glutPostRedisplay();

break;

case 'x':

reset();

A=1;

glutPostRedisplay();

break;

case 'y':

reset();

B=1;

glutPostRedisplay();

break;

case 'z':

reset();

C=1;

glutPostRedisplay();

break;

case 'p':

reset();

inverse\_elbow\_down=1;

while ((std::cout << "Enter End Effector Coordinate x: "<<endl) && (!(std::cin >> xx) || xx <0 || xx > 800))

{

std::cout << "Invalid input! Please enter a value between 0 and 800"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

while ((std::cout << "Enter End Effector Coordinate y: "<<endl) && (!(std::cin >> yy) || yy <0 || yy > 800))

{

std::cout << "Invalid input! Please enter a value between 0 and 800"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

while ((std::cout << "Enter Phi: "<<endl) && (!(std::cin >> pp) || pp <-360 || pp > 360))

{

std::cout << "Invalid input! Please enter a value between -360 and 360"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

inverse\_kinematics1(xx,yy,pp);

glutPostRedisplay();

break;

case 'q':

reset();

inverse\_elbow\_up=1;

while ((std::cout << "Enter End Effector Coordinate x: "<<endl) && (!(std::cin >> xx) || xx <0 || xx > 800))

{

std::cout << "Invalid input! Please enter a value between 0 and 800"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

while ((std::cout << "Enter End Effector Coordinate y: "<<endl) && (!(std::cin >> yy) || yy <0 || yy > 800))

{

std::cout << "Invalid input! Please enter a value between 0 and 800"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

while ((std::cout << "Enter Phi: "<<endl) && (!(std::cin >> pp) || pp <-360 || pp > 360))

{

std::cout << "Invalid input! Please enter a value between -360 and 360"<<endl<<endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

inverse\_kinematics2(xx,yy,pp);

glutPostRedisplay();

break;

case 'j':

//reset();

setverts();

inverse\_kinematics1(pts[(ctr1+=2)],pts[(ctr2+=2)],b3);

theta1=sol3[0];

theta2=sol3[1];

theta3=sol3[2];

if(inverse\_error\_flag1==1){

inverse\_error\_flag1=0;

ctr1=0;

ctr2=1;

set=true;

}

glutPostRedisplay();

break;

case 'r':

reset();

glutPostRedisplay();

break;

case 'w':

reset();

setverts\_up();

inverse\_kinematics1(ptsu[(ctru1+=2)],ptsu[(ctru2+=2)],b3);

theta1=sol3[0];

theta2=sol3[1];

theta3=sol3[2];

if(inverse\_error\_flag1==1){

inverse\_error\_flag1=0;

ctru1=0;

ctru2=1;

set\_up=true;

}

glutPostRedisplay();

break;

case 's':

reset();

setverts\_down();

inverse\_kinematics1(ptsd[(ctrd1+=2)],ptsd[(ctrd2+=2)],b3);

theta1=sol3[0];

theta2=sol3[1];

theta3=sol3[2];

if(inverse\_error\_flag1==1){

inverse\_error\_flag1=0;

ctrd1=0;

ctrd2=1;

set\_down=true;

}

glutPostRedisplay();

break;

case 'a':

reset();

setverts\_left();

inverse\_kinematics1(ptsl[(ctrl1+=2)],ptsl[(ctrl2+=2)],b3);

theta1=sol3[0];

theta2=sol3[1];

theta3=sol3[2];

if(inverse\_error\_flag1==1){

inverse\_error\_flag1=0;

ctrl1=0;

ctrl2=1;

set\_left=true;

}

glutPostRedisplay();

break;

case 'd':

reset();

setverts\_right();

inverse\_kinematics1(ptsr[(ctrr1+=2)],ptsr[(ctrr2+=2)],b3);

theta1=sol3[0];

theta2=sol3[1];

theta3=sol3[2];

if(inverse\_error\_flag1==1){

inverse\_error\_flag1=0;

ctrr1=0;

ctrr2=1;

set\_right=true;

}

glutPostRedisplay();

break;

case 'v':

reset();

store\_flag=true;

xstore[sind]=vert[6];

ystore[sind]=vert[7];

pstore[sind]=b3;

/\*for(int i=0;i<=sind;i++)

cout<<endl<<i<<"-"<<xstore[i]<<endl;\*/

sind++;

break;

case '1':

reset();

one=true;

inverse\_kinematics1(xstore[0],ystore[0],pstore[0]);

break;

case '2':

reset();

two=true;

inverse\_kinematics1(xstore[1],ystore[1],pstore[1]);

break;

case '3':

reset();

three=true;

inverse\_kinematics1(xstore[2],ystore[2],pstore[2]);

break;

case 'g':

goff=-goff;

break;

case 27: // Escape key

exit(1);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main( int argc, char\*\* argv )

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowPosition(0,0);

glOrtho(0.0, 800.0, 0.0, 800.0, 0.0, 1.0);

glutInitWindowSize( 800, 800 );

glutCreateWindow( "Planar 3R control" );

init();

glutMouseFunc(mouseEvent);

glutKeyboardFunc( myKeyboardFunc );

glutSpecialFunc( mySpecialKeyFunc );

glutReshapeFunc( resizeWindow );

glutDisplayFunc( drawscene );

glutMainLoop( );

return(0); // This line is never reached.

}

**Results:**

The picture below shows the robot with the end effector at (550,400) and phi=0.



Suppose we want to move the end effector to a new location. So we press **P** and enter the new end effector coordinates as we wish.

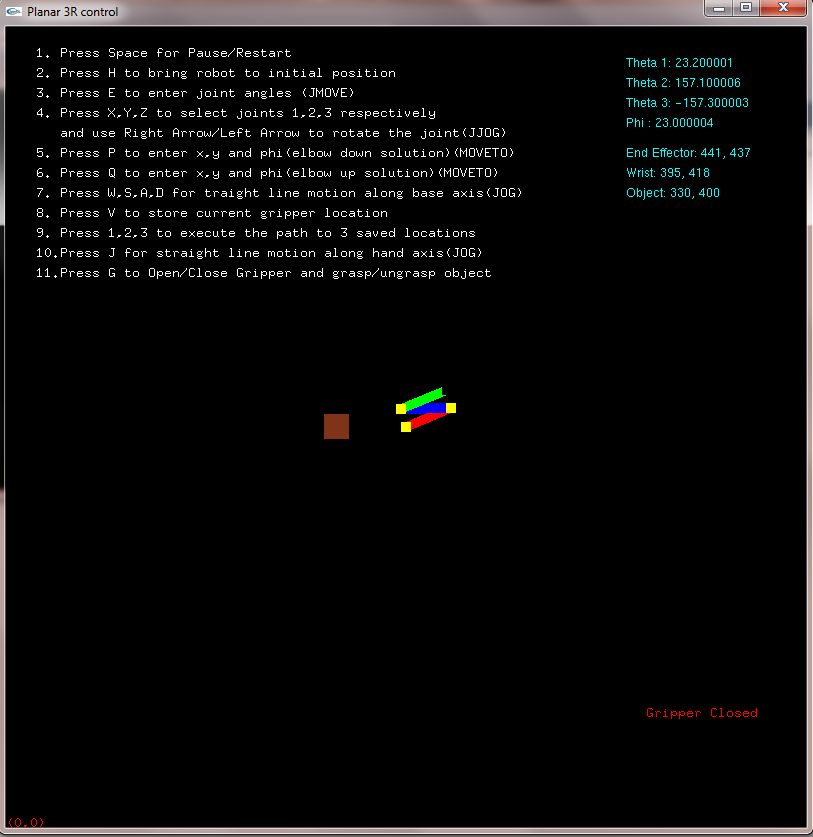
Let the user input be:

New end effector x= 442

New end effector y= 439

New Phi=23 (in degrees)

So the robot animates to this new position as show below.



**Analysis and Discussion of Results:**

The results are pretty much as we expect it to be. Sometimes there is a minor deviation from the exact location (user input) and the actual point where the robot moves to. For example you can from the previous screenshot that the user had input (442,439) as the new end effector position but the robot had moved to (441,437) from its previous position.

These minor differences might be attributed to setting precision to floating point decimals in the program code. Rounding off floating decimals has been to incorporate simple and fast calculations.

**Future Developments:**

* The program can be furthered by making the links non planar.
* It can be interfaced with an actual robot arm manipulator so that the user controls on the virtual environment is synchronized with that for the actual robot’s.
* Adding collision detection between arms of the robot can be another development.