19AIE201: Introduction to Robotics

Picasso the Drawing Robot: An Application of Inverse Kinematics

ASHWIN V
AM.EN.U4AIE19018

VISHAL MENON
AM.EN.U4AIE19070

Abstract: *With the recent advancement in the field of robotics, Robots have become versatile in size and form and can be programmed to do the most mundane as well as the most sophisticated and specialised tasks. Inverse kinematics is one of the key techniques in automation and working of robots and deep understanding of the same is required for designing algorithms for strenuous tasks. Hence, in this paper we have designed an algorithm which uses the concept of Inverse-Kinematics for achieving the basic task of drawing. Hence our simulated robot (Picasso) can draw basic shapes like: circle, square, pentagon and complex figures like: cat, flower etc.*

I. INTRODUCTION

In recent years we have seen an escalating development in the field of robotics and AI. The aim of industrial revolution 4.0 is to automate systems and tasks which are capable of doing things which humans are not capable of and with ease. But the current development in this field is far from the vision that the futurists had set, yet there is a steady development in this field and everyday engineers and researchers are making the impossible possible by designing robots and automated systems which are capable of doing strenuous tasks. Forward and Inverse kinematics is the crux of robot design and functionality, hence as a part of our project we aim to design a robot capable of drawing sketches from images using inverse kinematics. We have used MATLAB for the development and simulation of the project. The designed algorithm also opens up new fields and opportunity for robot applications. As our proposed system works on the basic idea of developing a path minimizing the movement of manipulators, which can be useful in various industrial, medical, military applications.

II. LITERATURE REVIEW

1. *Inverse kinematics for humanoid robots - Rajesh Kannan (Amrita University)*

   In this manuscript {ref}, Rajesh Kannan and his co-authors design a low-cost drawing bot as an embedded system that works on the basis of Computer Numerical Control. This paper deals with the design, implementation, and analysis of a low-cost drawing robot for educational purposes. The research elaborates on the technology behind building this low-cost bot. The hardware includes a simple embedded system with a low-cost microcontroller board. The bot has a 2 axis control using the stepper motor and the pen is controlled using a servo motor. It uses Inkscape software for drawing images and to create G-code files. Universal G-code sender sends these G-code files to the microcontroller board for controlling the pen/pencil to move in x, y, and z directions to draw the image. The experimental results show how close the images drawn by the bot to that of the original one. In addition, the detailed simple way of building this bot is also provided in this paper.

2. *Development of a self-reliant humanoid robot for a sketch drawing - Avinash Kumar Singh*

   The Research conducted by Avinash Kumar Singh, Neha Baranwal, and G. C. Nandi demonstrates the capability of a humanoid robot in the field of sketch drawing. Sketch drawing is a complex job that requires three basic problems to be solved. The first problem is to extract prominent features (the image point) of the object shown. The second is to define the image points lying on the Humanoid's camera plane with respect to its end-effector position. The third problem is to provide the inverse kinematic solution and control strategy for smooth drawing. A H25 NAO humanoid robot is used as a test-bed in this paper to conduct this experiment and illustrate the whole process. A calibration matrix is defined which transforms image points in the NAO body coordinate system while inverse kinematics has been solved using a gradient descent numerical method. The analytical solution of the inverse kinematics for NAO's hands is not suitable due to its mechanical design which is not following the piper's recommendation. The Denavit-Hartenberg (DH) parameters of the system have been defined in order to measure the working envelope of the right hand as well as to avoid singularities

3. *A Humanoid Robot Drawing Human Portraits - Sylvain Calinon*

   The paper authored by Sylvain Calinon, Julien Epiney, and Aude Billard presents the creation of a robot capable of drawing artistic portraits. The application is purely entertaining and based on existing tools for face detection and image reconstruction, as well as classical tools for trajectory planning of a 4 DOFs robot arm. The innovation of the application lies in the care we took to make the whole process as human-like as possible. The robot's motions and its drawings follow a style characteristic of humans. The portraits conserve the aesthetic features of the original images. The whole process is interactive, using speech recognition and speech synthesis to conduct the scenario.

4. *A Computational Approach to Edge Detection - John Canny*

   John Canny describes a computational approach to edge detection. The success of the approach depends on the definition of a comprehensive set of goals for the computation of edge points. These goals must be precise enough to delimit the desired behavior of the detector while making minimal assumptions about the form of the solution. He defines detection and localization criteria for a class of edges, and presents mathematical forms for these criteria as functionals on the operator impulse response. A third criterion is then added to ensure that the detector has only one response to- a single edge.

III. THEORY

In computer animation and robotics, inverse kinematics is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain, such as a robot manipulator or animation character's skeleton, in a given position and orientation relative to the start of the chain. Given joint parameters, the position and orientation of the chain's end, e.g. the hand of the character or robot, can typically be calculated directly using multiple applications of trigonometric formulas, a process known as forward kinematics. However, the reverse operation is, in general, much more challenging.

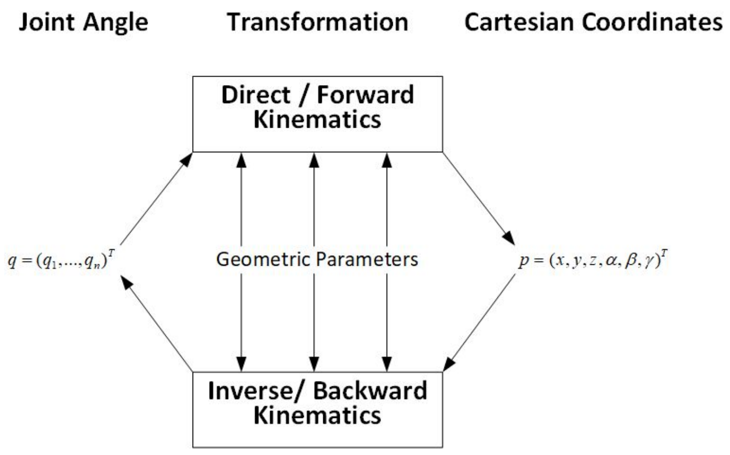

Joint Angle
Transformation
Cartesian Coordinates
Direct / Forward Kinematics
$q = (q_1, ..., q_n)^T$
Geometric Parameters
$p = (x, y, z, \alpha, \beta, \gamma)^T$
Inverse/ Backward Kinematics

Fig.1. Forward vs Inverse Kinematics

Depending on your robot geometry, IK can either be solved analytically or numerically:

A. *Analytical solutions:*
   In some cases, but not all, there exist analytical solutions to inverse kinematic problems. One such example is for a 6-DoF robot (for example, 6 revolute joints) moving in 3D space (with 3 position degrees of freedom, and 3 rotational degrees of freedom). If the degrees of freedom of the robot exceeds the degrees of freedom of the end-effector, for example with a 7 DoF robot with 7 revolute joints, then there exist infinitely many solutions to the IK problem, and an analytical solution does not exist. Further extending this example, it is possible to fix one joint and analytically solve for the other joints, but perhaps a better solution is offered by numerical methods (next section), which can instead optimize a solution given additional preferences (costs in an optimization problem). An analytic solution to an inverse kinematics problem is a closed-form expression that takes the end-effector pose as input and gives joint positions as output,

$$q = f(x)$$

   Analytical inverse kinematics solvers can be significantly faster than numerical solvers and provide more than one solution, but only a finite number of solutions, for a given end-effector pose.

B. *Numerical solutions:*
   The most flexible method for solving IK typically relies on iterative optimization to seek out an approximate solution, due to the difficulty of inverting the forward kinematics equation and the possibility of an empty solution space. The core idea behind several of these methods is to model the forward kinematics equation using a Taylor series expansion, which can be simpler to invert and solve than the original system.

IV. METHODOLOGY

We use the inverse Kinematics System object to create an inverse kinematic (IK) solver to calculate joint configurations for a desired end-effector pose based on a specified rigid body tree model set on default. This model defines all the joint constraints that the solver enforces. If a solution is possible, the joint limits specified in the robot model are obeyed.

To compute joint configurations for a desired end-effector pose:

1. Create the *inverseKinematics* object and set its properties.
2. Call the object with arguments, as if it were a function.

We represent our Robot Structure as *RigidBodyTree* objects. The Rigid Body Tree is a representation of the connectivity of rigid bodies with joints. The *RigidBodyTree* class is used to build robot manipulator models in MATLAB. The import robot function is used to load our robot model from the URDF (Unified Robot Description Format).

The two algorithms used by this Inverse Kinematics Solver are:

A. BFGS (Numerical solution):
The Broyden-Fletcher-Goldfarb-Shanno (BFGS) gradient projection algorithm is a quasi-Newton method that uses the gradients of the cost function from past iterations to generate approximate second-derivative information. The algorithm uses this second-derivative information in determining the step to take in the current iteration. A gradient projection method is used to deal with boundary limits on the cost function that the joint limits of the robot model create. The direction calculated is modified so that the search direction is always valid. This algorithm is effective for configurations near joint limits or when the initial guess is not close to the solution.

B. Levenberg-Marquardt (Numerical solution):
The Levenberg-Marquardt (LM) algorithm variant used in the InverseKinematics class is an error-damped least-squares method. The error-damped factor helps to prevent the algorithm from escaping a local minimum. The LM algorithm is optimized to converge much faster if the initial guess is close to the solution. However the algorithm does not handle arbitrary initial guesses well. Consider using this algorithm for finding IK solutions for a series of poses along a desired trajectory of the end effector. Once a robot configuration is found for one pose, that configuration is often a good initial guess at an IK solution for the next pose in the trajectory. In this situation, the LM algorithm may yield faster results. Otherwise, use the BFGS Gradient Projection instead.

As the basic objective of our project is to create a manipulator capable of drawing input images it is necessary to have a function capable of converting images to coordinates, as inverse kinematics works based on waypoints (trajectory points). Hence, we have implemented a function for converting images to coordinates (pic2point). The details of the function are available in the following sections. Figure.2. shows the conversion of image into coordinate.

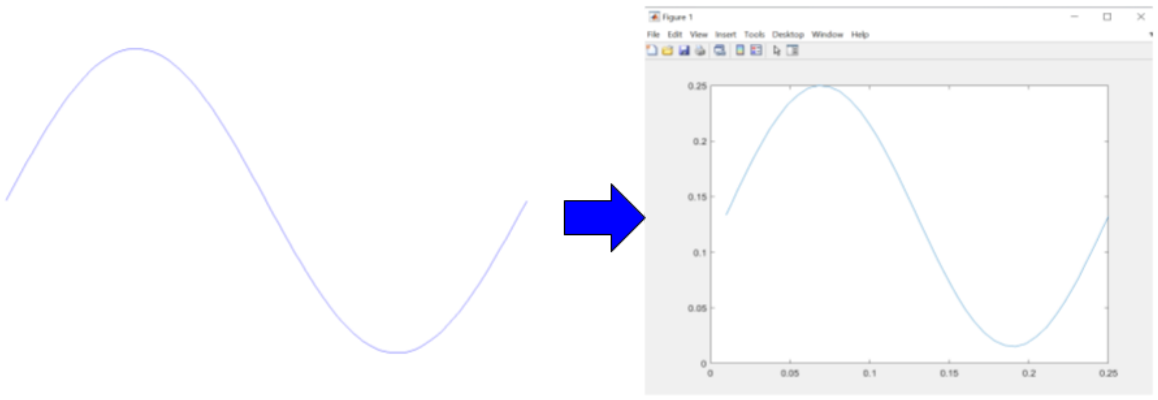


Fig.2. Conversion of image to coordinates using pic2point function in MATLAB

A flow chart of detailed steps involved in the implementation of our project: (Figure.3.)

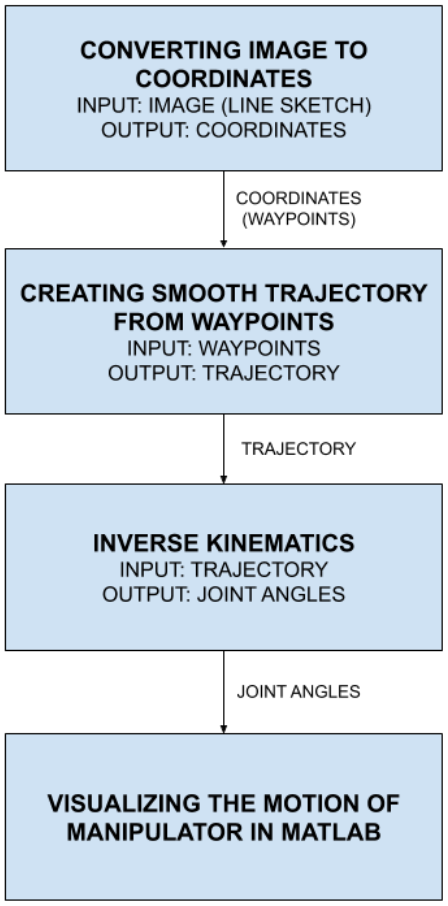
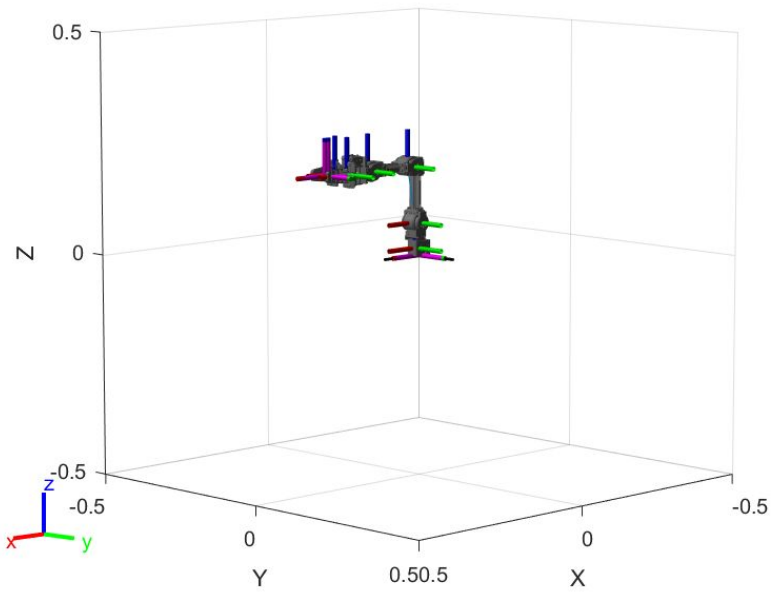


Fig.3. Flowchart

I. IMPLEMENTATION

A. Load and display robot:

```
clear, clc
addpath(genpath(strcat(pwd,'\Dependencies')))
robot = createRigidBodyTree;

axes = show(robot);
```



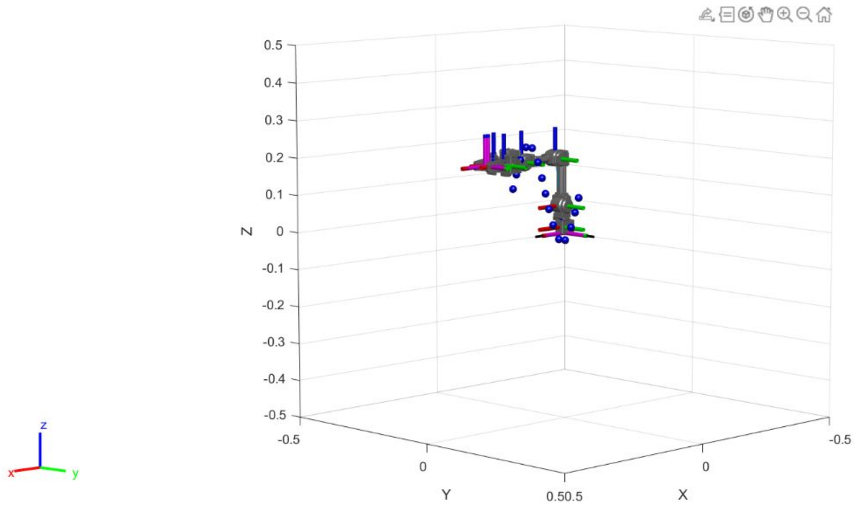
B. Robot Details:

```
axes.CameraPositionMode = 'auto';
showdetails(robot)
```

```
--------------------
Robot: (9 bodies)

 Idx            Body Name            Joint Name            Joint Type
 ---            ---------            ----------            ----------
   1                link1           world_fixed                 fixed
   2                link2                joint1              revolute
   3                link3                joint2              revolute
   4                link4                joint3              revolute
   5                link5                joint4              revolute
   6    end_effector_link    end_effector_joint                 fixed
   7         gripper_link               gripper             prismatic
   8     gripper_link_sub           gripper_sub             prismatic
   9         end_effector      end_effector_jnt                 fixed
--------------------
```

C. *Create a set of desired waypoints from Images:*

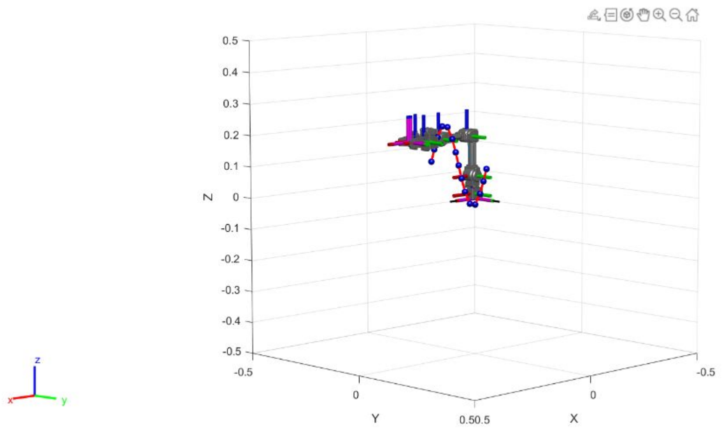
```
Im = imread('C:\Users\HP\Desktop\pic2points\Images\Plot.png');

T = graythresh(Im);
ImPlot = 'true';
maxNum = 15;
M = pic2point(Im,T,ImPlot,maxNum);  % Function pic2point defined below
M = sortrows(M,'descend');

M = scale(M);            % Function scale defined below
M = nearestSort(M);      % Function nearestSort defined below


wayPoints = [zeros(size(M,1),1), M];
wayPoints(wayPoints(:,1)==0) = .2;
wayPointVels = zeros(size(M,1),3);
exampleHelperPlotWaypoints(wayPoints);
```



D. *Create a smooth trajectory from the waypoints:*

```
numTotalPoints = size(wayPoints,1)*10;
waypointTime = 0.001;
trajType = 'cubic'; % or 'trapezoidal'
switch trajType
    case 'trapezoidal'
        trajectory =
trapveltraj(wayPoints',numTotalPoints,'EndTime',waypointTime);
```

```
    case 'cubic'
        wpTimes = (0:size(wayPoints,1)-1)*waypointTime;
        trajTimes = linspace(0,wpTimes(end),numTotalPoints);
        trajectory = cubicpolytraj(wayPoints',wpTimes,trajTimes, ...
                     'VelocityBoundaryCondition',wayPointVels');
end
% Plot trajectory spline and waypoints
hold on
plot3(trajectory(1,:),trajectory(2,:),trajectory(3,:),'r-','LineWidth',2);
```

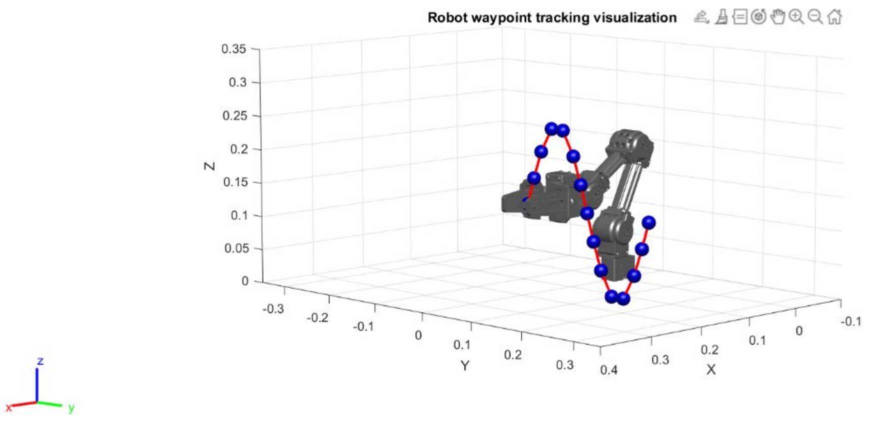


E. *Perform Inverse Kinematics:*

```
ik = robotics.InverseKinematics('RigidBodyTree',robot);
weights = [0.1 0.1 0 1 1 1];
initialguess = robot.homeConfiguration;

% Call inverse kinematics solver for every end-effector position using the
% previous configuration as initial guess
for idx = 1:size(trajectory,2)
    tform = trvec2tform(trajectory(:,idx)');
    configSoln(idx,:) = ik('end_effector',tform,weights,initialguess);
    initialguess = configSoln(idx,:);
end
```

F. *Visualize robot configurations:*

```
title('Robot waypoint tracking visualization')
axis([-0.1 0.4 -0.35 0.35 0 0.35]);
for idx = 1:size(trajectory,2)
    show(robot,configSoln(idx,:), 'PreservePlot', false,'Frames','off');
    pause(0.1)
end
hold off
```



G. *Functions:*

scale: Used for scaling the coordinates into the work envelope of the manipulator.

Parameter: M - x,y coordinates (waypoints in this case)

Output: M - x,y coordinates after scaling

pic2point: Used for converting images to coordinate points.

Parameters:

- Im - Input image, it can be gray, color or binary image. The image can be Double = [0,1] or Unsign integer = [0,255]. The image foreground must be black on white background.
- TshV - Denotes the threshold value for image binarization. It is between 0~1 and by default is calculated by graythresh(Im).
- ImPlot - If the image is a plot, ImPlot must be ‘1’ or ‘true’. The image is thinned to 1-pixel width line when ImPlot is true therefore finding the coordinate of drawn line in plot would be possible.
- maxNum - Denotes maximum number of points that should be extracted from the image. If all points in the image are requested, leave the maxNum empty; otherwise enter the arbitrary number to select randomly from all points. Actually, it is a random resampling from all point for output.

Output:

- MResult - Matrix of pixels/points coordinates in the image. MResult has two columns, first column denotes x-coordinates and second column is for y-coordinates. Number of rows equals to number of points in the image.

nearestSort: Sorts the values according to order of connectivity such that total distance is reduced (uses travelling salesman method).

Parameters: M- x,y coordinates

Output: M- x,y coordinates

```
function [M] = scale(M)
x = M(:,1);
y = M(:,2);
x = x./(max(x)*4);
y = y./(max(y)*4);
M = [x y];
end

function [MResult] = pic2point(Im,TshV,ImPlot,maxNum)


switch nargin
    case 1
        Pmode = false; % Not a plot
        randomPick = false;
        TshV = graythresh(Im);
    case 2
        Pmode = false; % Not a plot
        randomPick = false;
    case 3
        if strcmpi(ImPlot,'plot')
            Pmode = true; % It is a plot
        else
            Pmode = false; % It is an image
        end
        randomPick = false;
    case 4
        if strcmpi(ImPlot,'plot')
            Pmode = true; % It is a plot
        else
            Pmode = false; % It is an image
        end
        randomPick = true;
end
ImBW=im2bw(Im,TshV);
ImBW=1-ImBW; % When the plot drew by black ink on the white background
if Pmode
    ImBW = bwmorph(ImBW,'thin',Inf);
end
% figure(); imshow(ImBW);
[r,c,v] = find(ImBW==1);
r = size(ImBW,1) - r; % Correct the coordinates from the image
TMresult = [c r];
if randomPick
    Sran = linspace(1,sum(v),maxNum);
    Sran = uint32(Sran);
    %Sran = randperm(sum(v),maxNum);
    MResult = TMresult(Sran',:);
else
    MResult = TMresult;
end
end

%% Nearest Sort

function [Mnew] = nearestSort(M)

x = M(:,1);
y = M(:,2);
numPoints = size(M,1);

newx = [];
newy =[];

set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.08, 1, 0.92]);

% Make a list of which points have been visited
beenVisited = false(1, numPoints);
% Make an array to store the order in which we visit the points.
visitationOrder = ones(1, numPoints);
% Define a filasafe
maxIterations = numPoints + 1;
iterationCount = 1;
% Visit each point, finding which unvisited point is closest.
% Define a current index.  currentIndex will be 1 to start and then will vary.
currentIndex = 1;
while sum(beenVisited) < numPoints && iterationCount < maxIterations
  % Indicate current point has been visited.
  visitationOrder(iterationCount) = currentIndex;
  beenVisited(currentIndex) = true;
  % Get the x and y of the current point.
  thisX = x(currentIndex);
  newx = [newx thisX];

  thisY = y(currentIndex);
  newy = [newy thisY];

  % Compute distances to all other points
  distances = sqrt((thisX - x) .^ 2 + (thisY - y) .^ 2);
  % Don't consider visited points by setting their distance to infinity.
  distances(beenVisited) = inf;
  % Also don't want to consider the distance of a point to itself, which is 0
and would also ways be the minimum distances of course.
  distances(currentIndex) = inf;
  % Find the closest point.  this will be our next point.
  [minDistance, indexOfClosest] = min(distances);
  % Save this index
  iterationCount = iterationCount + 1;
  % Set the current index equal to the index of the closest point.
  currentIndex = indexOfClosest;
end
Mnew(:,1) = newx;
Mnew(:,2) = newy;
end
```

V. RESULTS

Please click the following link for seeing the Robot Manipulator in Action: https://drive.google.com/file/d/1fvEC4ViD_82dIvvPTGznCzg7yydY8zCZ/view?usp=sharing

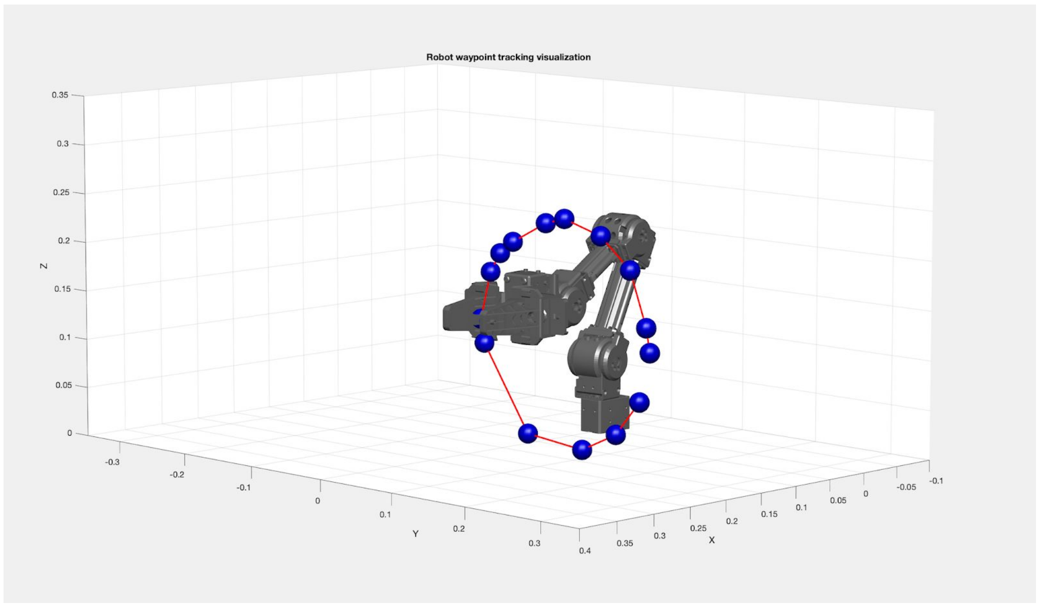


Fig.4. Manipulator drawing a circle

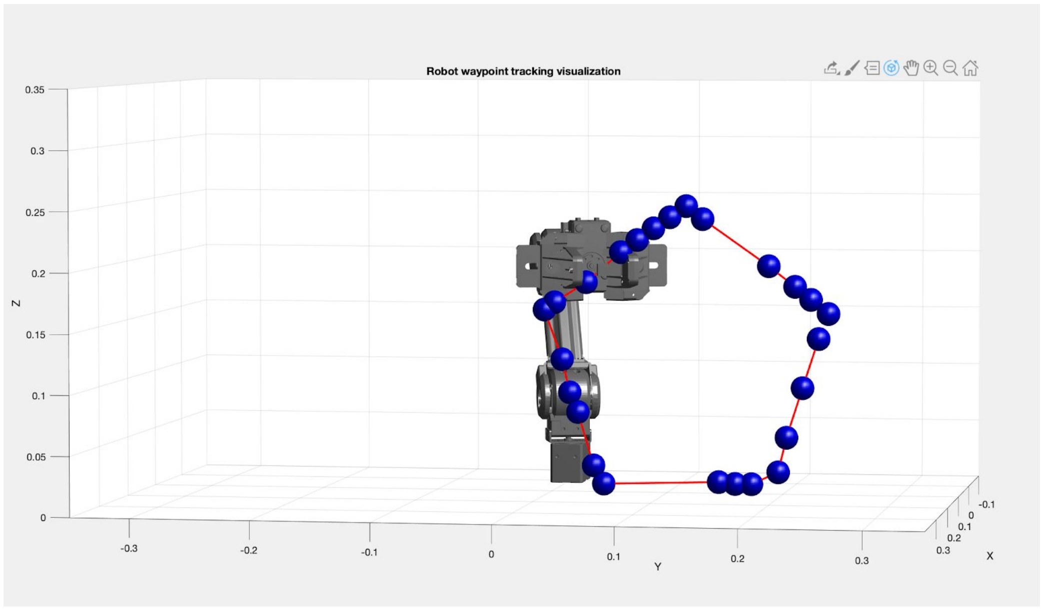


Fig.5. Manipulator drawing a pentagon

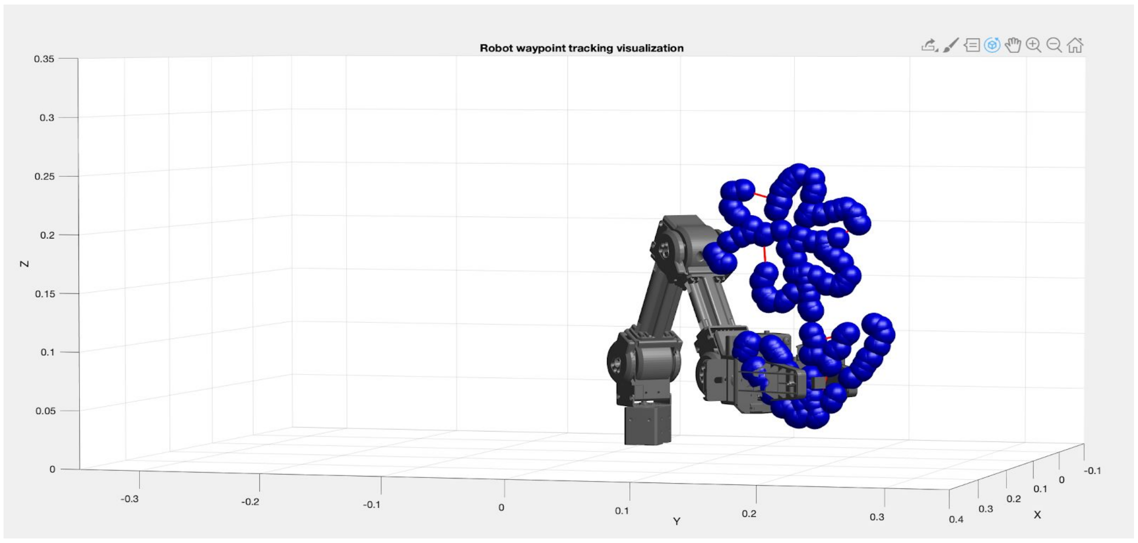


Fig.6. Manipulator drawing a flower

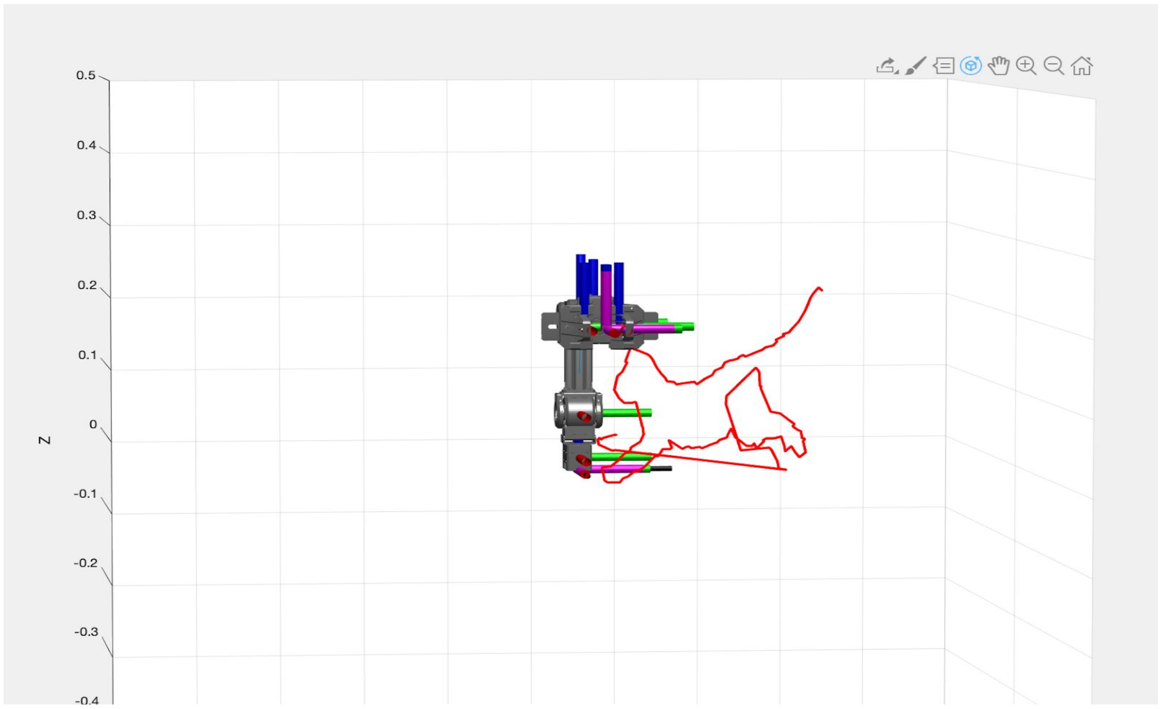


Fig.7. Manipulator drawing a cat

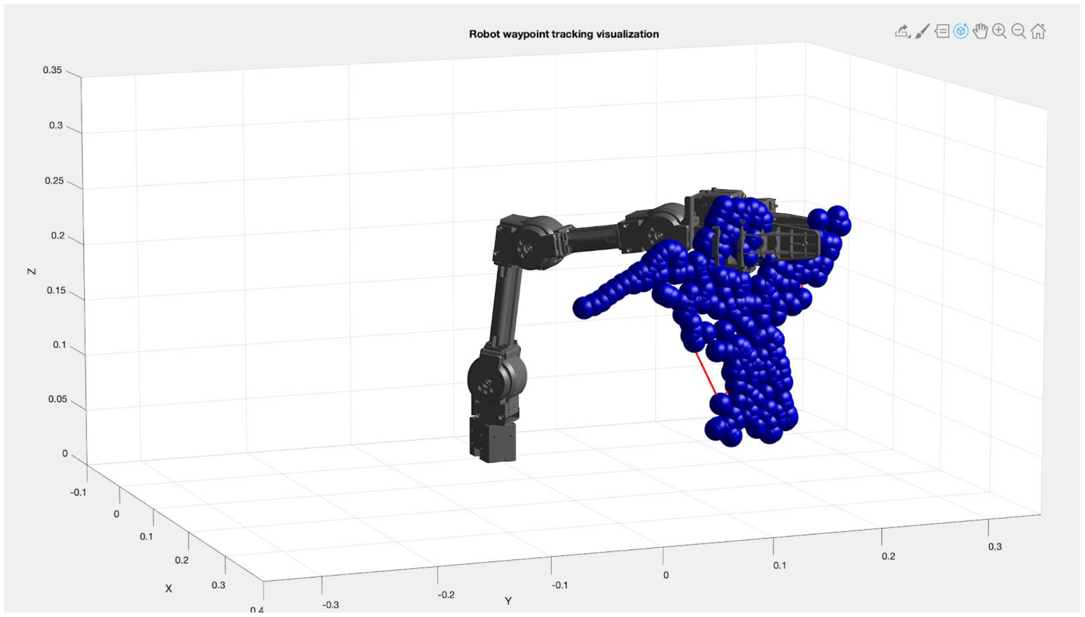


Fig.8. Manipulator drawing a Humming bird

VI. REFERENCES

[1] IMegalingam, Rajesh Kannan & Raagul, Shri Rajesh & Dileep, Sonu & Sathi, Sarveswara & Pula, Bhanu & Vishnu, Souraj & Sasikumar, Vishnu & Sai, Uppala & Gupta, Chaitanya. (2018). Design, Implementation and Analysis of a Low Cost Drawing Bot for Educational Purpose.

[2] Flash, Tamar & Meirovitch, Yaron & Barliya, Avi. (2013). Models of human movement: Trajectory planning and inverse kinematics studies. Robotics and Autonomous Systems. 61. 330–339. 10.1016/j.robot.2012.09.020.

[3] J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[4] Singh, A.K., Baranwal, N. & Nandi, G.C. Development of a self reliant humanoid robot for sketch drawing. *Multimed Tools Appl* 76, 18847–18870 (2017). https://doi.org/10.1007/s11042-017-4358-x