DEPARTMENT OF MECHATRONICS ENGINEERING UNIVERSITY OF ENGINEERING & TECHNOLGY,

PESHAWAR

"MtE-5144: Computer Application in Robotics"

MUHAMMAD ARSALAN
12PWMEC-3404

“Obtaining Real world coordinates(x,y) from image coordinates(u,v) for twolink Robotic manipulator ”

Abstract:

The main problem in the robotic vision is, to identify the object and extract the position information from the image that where in the real world the object will be. I will be solving the same problem in the MATLAB as well as practically on the worksheet.
The worksheet will be consists of three different colors and different shapes having circles and rectangles. On this worksheet I will perform the experiment after extracting the information in MATLAB and then I will compare my results in the worksheet. I will take the image of this worksheet, import in the MATLAB and applying algorithms, and then I will be able to exactly tell where the red circle will be on the real worksheet.
After getting the real world coordinates for the desired colored shape then I will command twolink robotic arm to move to the exact location through the inverse kinematics.

INTRODUCTION:

In industries robot has to interact with the different shape and colored object to perform industrial task. In industries robot must have capability to perform artificially in their environment.
The main motivation of this project is that to find the real world coordinates from the camera or vision sensor attach in the environment of the robotic system and then giving that world coordinates to the robot to move their end-effectors to that exact location in the real world.
This is the main part of the vision in robotics to map the image coordinates (u,v) to world coordinate(x,y).
The determination of real world coordinates has many applications in computer vision including robot positioning, object reconstruction and measurement [1].
First of all we will try to make simple experiment to check the accuracy of our solution of finding the real world coordinate from the image coordinates.

OBJECTIVES:

The objectives of this project is to finding the real world coordinates from the image coordinate taken from the camera or vision sensors, in real world the robot knows and understand the world coordinates rather than pixel coordinates and there are the huge difference between the pixel and world coordinates.

Camera take image of the robotic system environment and capture this as the image having (u,v) coordinates, so there is a relationship between the image coordinate to the world coordinates and is known as camera matrix, this matrix includes the intrinsic and extrinsic properties of the cameras. Our goal is to find the camera matrix of our laptop webcam by comparing the world coordinates and the image coordinates.

LITERATURE REVIEW

How images are formed?

Image is formed in a camera by projection of 3-dimensional world coordinates to the 2-dimensional image coordinates on the lens. In the image the distance between the camera and object is not known and thus we can't tell how much the object is away from us . This transformation from 3 to 2 dimensions is known as perspective projection [2].

The image formation geometry for a thin convex lens is shown,

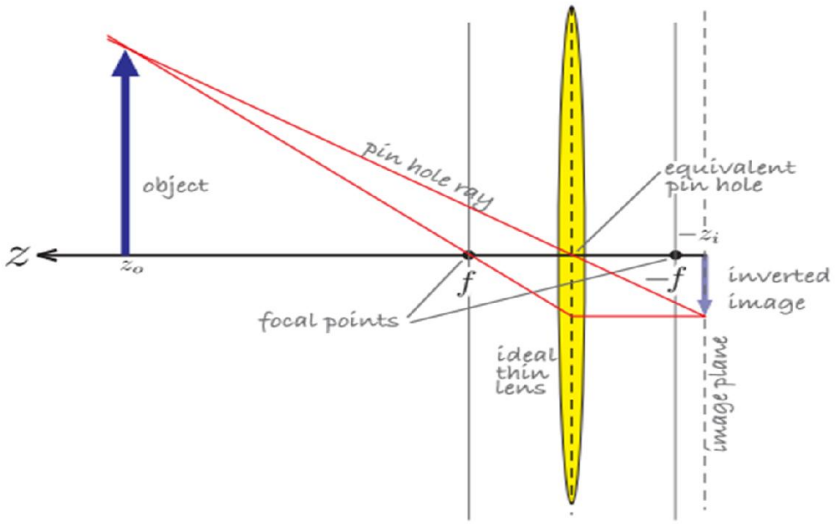


Figure 1: Image formation geometry [2]

A lens has two focal points at a distance of f on each side of the lens. The positive z-axis is the camera's optical axis. The z-coordinate of the object and its image, with respect to the lens center, are related by thin lens equation [2].

Z_0=distance of the object

Z_i=distance of the image

f=focal length of the lens

In computer vision it is common to use the central perspective image mode shown,

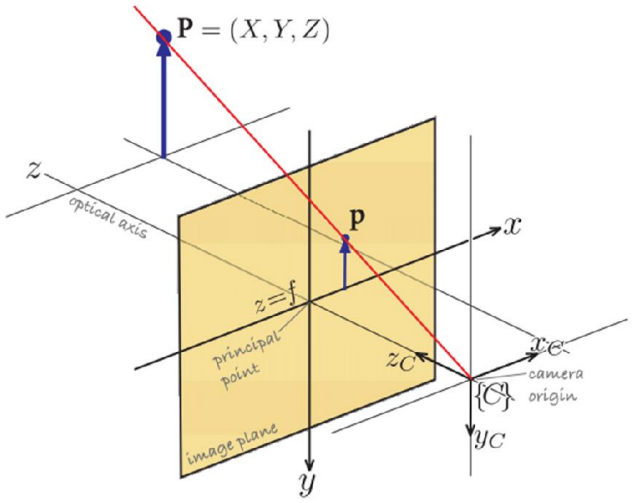


Figure 2: Central camera projection model [2]

The rays converge on the origin of the camera frame {C} and a non-inverted image is projected onto the image plane located at z=f. The z-axis intersects the image plane at the principal point which is the origin of the 2D image coordinate frame [2]. Using similar triangles we can show that point at the world coordinates P=(X, Y, Z) is projected to the image point p=(x,y) by

$X=f\frac{X}{Z}$, $y=f\frac{Y}{Z}$

MODELING A PERSPECTIVE CAMERA:

We can write the image plane point coordinates in homogenous form

$$\tilde{\boldsymbol{p}} = (\tilde{x}, \tilde{y}, \tilde{z})$$

$$\tilde{x} = fX,\ \tilde{y} = fY,\ \tilde{z} = Z$$

We can use in compact matrix form as

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

If we write the world coordinates in homogenous form as well ${}^{C}\tilde{\boldsymbol{P}} = (X, Y, Z, 1)^T$ then,

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}\tilde{\boldsymbol{P}} \quad \textbf{or} \quad \tilde{\boldsymbol{p}} = \boldsymbol{C}\, {}^{C}\tilde{\boldsymbol{P}}$$

Where
tilde=homogenous quantities
C is the 3x4 matrix known as camera matrix

$\tilde{\boldsymbol{p}}$ = pixel cordinates

${}^{C}\tilde{P}$ = world cordinates of the points with respect to camera frame {C}

Expanding the camera matrix now,

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}\tilde{\boldsymbol{P}}$$

here first matrix on the right side of the equation shows the scaling/zooming characteristics of the camera and next to this matrix is used to convert 3-dimensional quantity to 2-dimensional quantity.

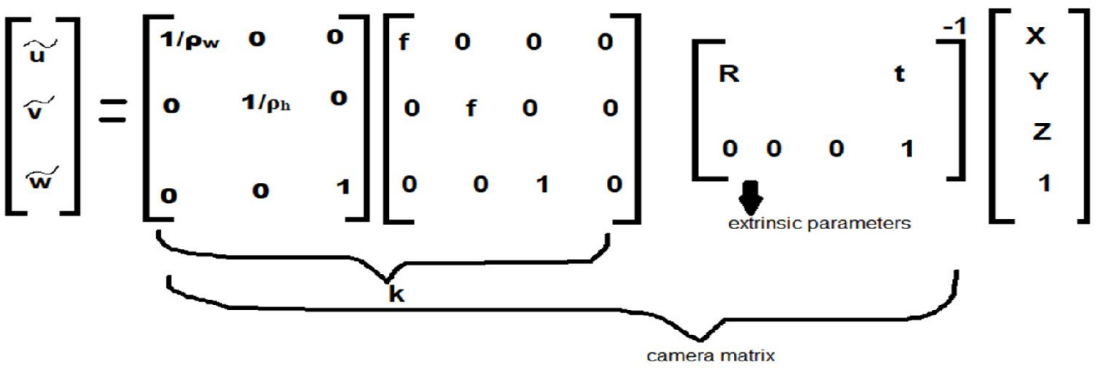
Now, if we project the world camera points to the image plane, the image plane is made up of small grids (u,v) called pixels.
origin of image plane is at top left hand corner by convention. So,

$$u = \frac{x}{\rho_w} + u_0,\ v = \frac{y}{\rho_h} + v_0$$

ρ_w are width of each pixel and ρ_h are the height of each pixel and (u_0, v_0) is the principal point.
we can write starting equation in term of pixel coordiantes as

$$\tilde{\boldsymbol{p}} = \underbrace{\begin{pmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}\tilde{\boldsymbol{P}}$$

we can further expand the camera matrix because there must be the pose of the camera involved in the maping of the points [2].

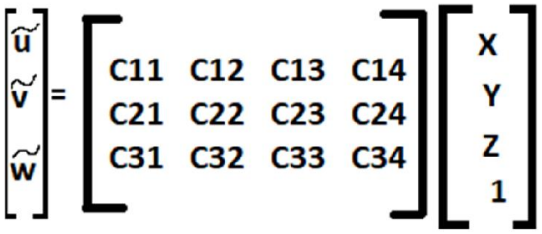


K=it is the internal parameters of the camera like pixel values and the focal length

Extrinsic paramerters contain what is the pose of the camera, it is the external parameters of camera. Pose conatains Rotation and translation of camera.

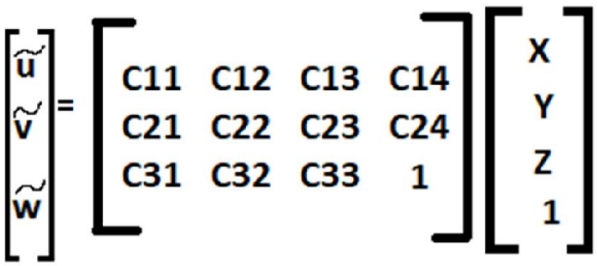
By combining these two parameters we get the camera matrix which maps the real world points to the image coordinates.

We can generally write the above equation As



it is very difficult to find the camera matrix from the basic principles because of the lack of information about the camera intrinsic parameters, thus we can estimate this matrix by doing some calibration process.

This camera matrix is independent of the scaling factor, if we multiply by any number and then after finding the world cordinates to the pixel cordinates this number is cancells out and thus we can write above equation as



POINTS ON THE PLANE/ PLANER HOMOGRAPHY:-

If we are observing the plane or paper in the real world from the camera then we know that all point on the plane have z=0, because of the 2-D plane. Thus equation becomes

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} C11 & C12 & C13 & C14 \\ C21 & C22 & C23 & C24 \\ C31 & C32 & C33 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

When camera matrix multiply with z=0 of world coordinates then the entire 3rd row of camera matrix becomes zero, we don't need 3rd row in this case.

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} C11 & C12 & C14 \\ C21 & C22 & C24 \\ C31 & C32 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} H11 & H12 & H13 \\ H21 & H22 & H23 \\ H31 & H32 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

central matrix is then known as homography matrix and it is denoted by H, which maps the points in the plane to the points in the image.
There are 8 unique numbers in Homography matrix.
we need at least 4 points on the plane and their coresponding point in the image to find the H matrix.
By finding the planer homography matrix we can do the inverse also to find the real world coordinates from the image cordinates [3].

TWOLINK MANIPULATOR:

When we get the world coordinates from the image coordinates, we then command the twolink manipulator to move to that point. For this function to accomplish, we will need the inverse and forward kinematics equations and also some dimensions of the robot, so the following are the robot description for just theoretical movement of the robot,
L1= link 1 length =15cm
L2=link 2 length =15 cm
2R manipulator

INVERSE KINEMATICS:

c2= (xd^2 +yd^2 -L1^2 -L2^2)/(2*L1*L2);
s2=(1-c2^2)^0.5;
K1= L1+(L2*c2);
K2=L2*s2;
des_th1= atan2 (yd,xd)-atan2(K2,K1);
des_th2=atan2 (s2, c2);
des_th1= joint-1 angle
des_th2=joint-2 angle

FORWARD KINEMATICS:
x=L2*cos(des_th1+ des_th2) + L1*cos(des_th1)
y=L2*sin(des_th1+ des_th2) + L1*sin(des_th1)

We will use robotics toolbox in MATLAB to create and move the twolink robot.
DH-TABLE of twolink robot [4].

```
                ([theta,  d,    a,   alpha    offset])
L(1)  =Link([   0       0     15       0         0])
L(2)  =Link([   0       0     15       0         0])
```

METHEDOLOGY

The vision worksheet is shown below,

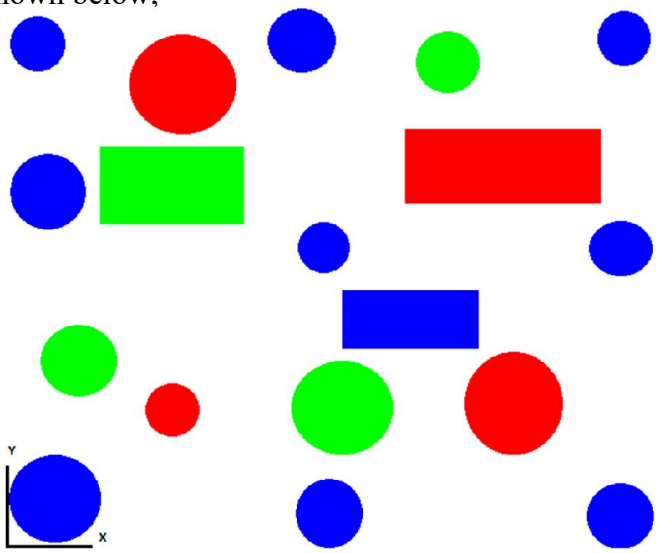


Figure 3: design of Vision worksheet

First of all I took the print of this worksheet and then took the image through my laptop webcam as shown below,

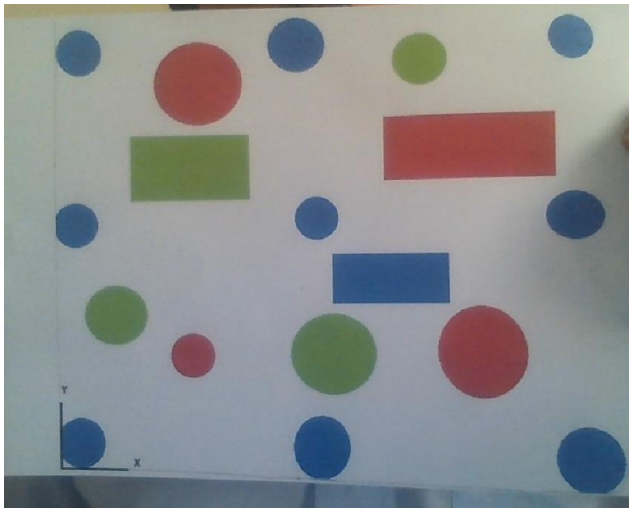


Figure 4: Image taken from webcam of vision worksheet

Now I have the actual vision worksheet and the image of the worksheet. I want to map the vision worksheet image pixels (u, v) to the real vision worksheet coordinates (x, y).

The basic steps for mapping image coordinates to real world coordinates are

1. Find all mark the sequence of the points so that points must be consistent between the real worksheet and in the image.

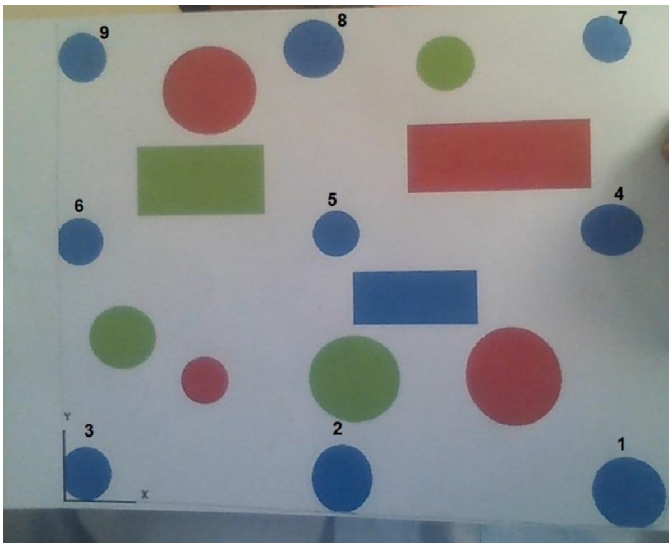


Figure 5: numbering the blue circle for sequence

2. Find the center points (X, Y) by ruler of all the nine blue circles in printed worksheet. As I have measured all the points so the points are given below, we need at least 4 points but for better estimation we will compare 9 points (cm)
   x1=25.2 , **x2**=13.2, **x3**= 1, **x4**= 25.2, **x5**= 12.8, **x6**= 1.3, **x7**= 25.2, **x8**= 12, **x9**= 1.4
   y1=1.3, **y2**= 1.3, **y3**=1.3, **y4**=12, **y5**=12.4, **y6**=12, **y7**=20.5, **y8**=20.7, **y9**=20.5
   In matrix form
   x=[x1 x2 x3 x4 x5 x6 x7 x8 x9 ; y1 y2 y3 y4 y5 y6 y7 y8 y9]
3. Find the centre points (u,v) of all nine blue circles in the image through MATLAB ,I have computed the pixels also, the values are shown below,
   p= [uc1 uc2 uc3 uc4 uc5 uc6 uc7 uc8 uc9; vc1 vc2 vc3 vc4 vc5 vc6 vc7 vc8 vc9]
   p= [812.58 440.79 109.58 790.08 432.46 100.71 783.76 403.78 104.155; 632.44 614.77 609.01 291.088 296.78 307.13 45.65 57.11 69.46]

4. We have now the pixels coordinates and the world coordinates, we can now find the Homography matrix, the matrix which can map the pixel coordinates to the world coordinates, this can be done easily in the MATLAB robotics toolbox by peter corke [5].
5. Once we have the Homography matrix we can now find the inverse problem, i.e. we can now find the world coordinates from the image coordinates.

6. I have find and compared three objects world coordinates (red rectangle, green rectangle and top left corner red circle) and test on the real worksheet by drawing the twolink robot.

7. Once we have (x,y) values then through inverse kinematics we will find the joint angles for twolink manipulator and then command the robot to move to the point by forward kinematics in MATLAB as well as on the vision worksheet.

8. You can do this process to any colored and any shape objects on the vision worksheet it will exactly map world coordinates.

RESULTS:

a) Twolink robot without system dynamics and PD-controller

1. Pixels of each blue circle is computed in MATLAB

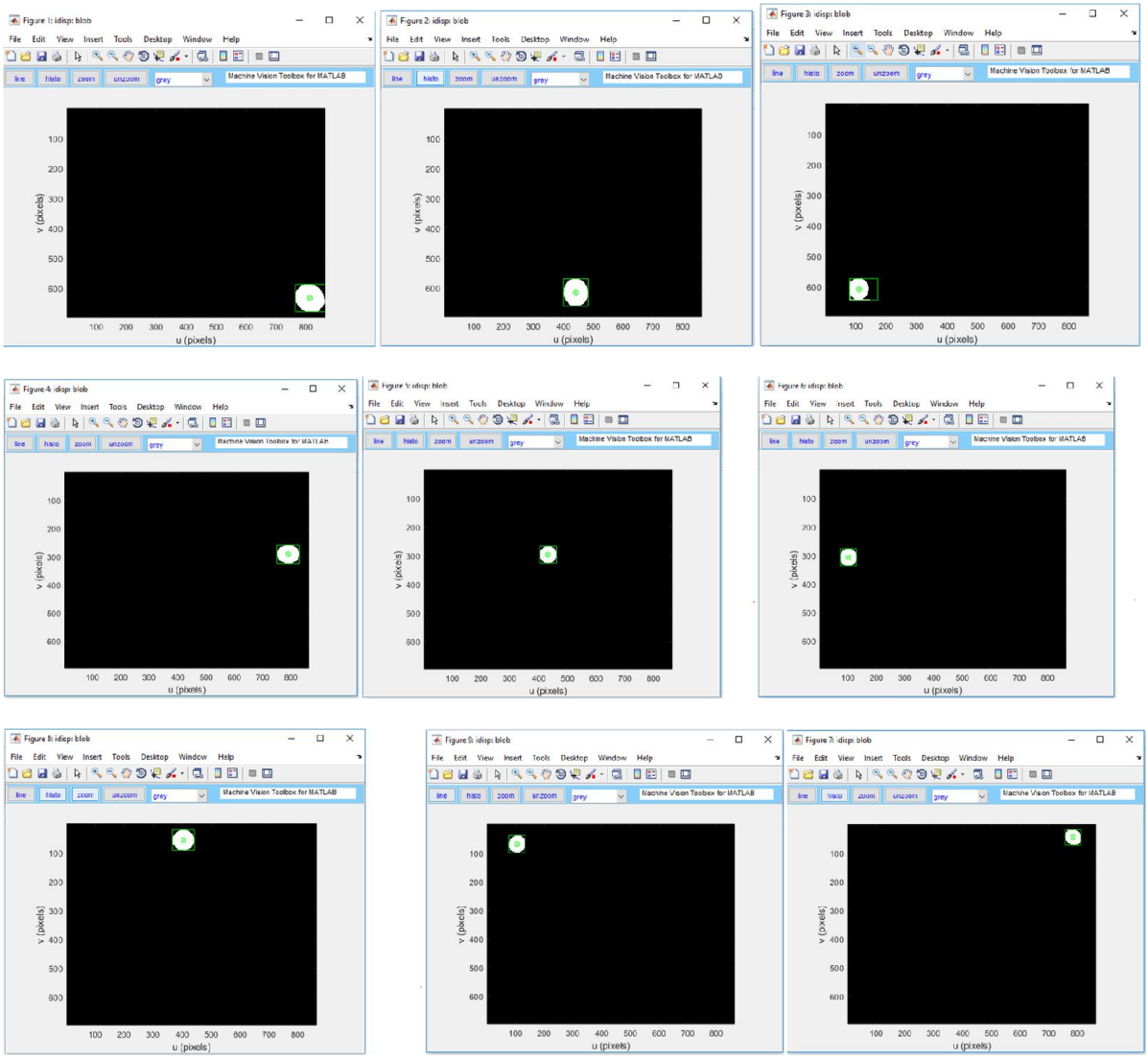


Figure 6: finding the blue circle centers

2. World coordinates of nine blue circle are ,
   x1=25.2 , **x2**=13.2, **x3**= 1, **x4**= 25.2, **x5**= 12.8, **x6**= 1.3, **x7**= 25.2, **x8**= 12, **x9**= 1.4
   y1=1.3, **y2**= 1.3, **y3**=1.3, **y4**=12, **y5**=12.4, **y6**=12, **y7**=20.5, **y8**=20.7, **y9**=20.5
   In matrix form
   x=[x1 x2 x3 x4 x5 x6 x7 x8 x9 ; y1 y2 y3 y4 y5 y6 y7 y8 y9]
3. In MATLAB we can find the Homography matrix
   H =

$$\begin{bmatrix} 0.0384 & -0.0006 & -2.6168 \\ 0.0009 & -0.0362 & 23.3421 \\ 0.0001 & 0.0000 & 1.0053 \end{bmatrix}$$

4. Now I want to find the world coordinates of red rectangle,
   world coordinates of the red circle computed in MATLAB is
   x= 20.3402
   y= 15.5460
   And the actual coordinate measured from the ruler is
   x=20.1
   y=15.6
   By inverse kinematic for twolink robot we got the joint1 and joint2 angles
   theta1= 5.9695
   theta2= 62.8423
   thus we command then robot to move to the desired object coordinates and you can see in the picture the robot has exactly moved to that position in MATLAB & on worksheet also.

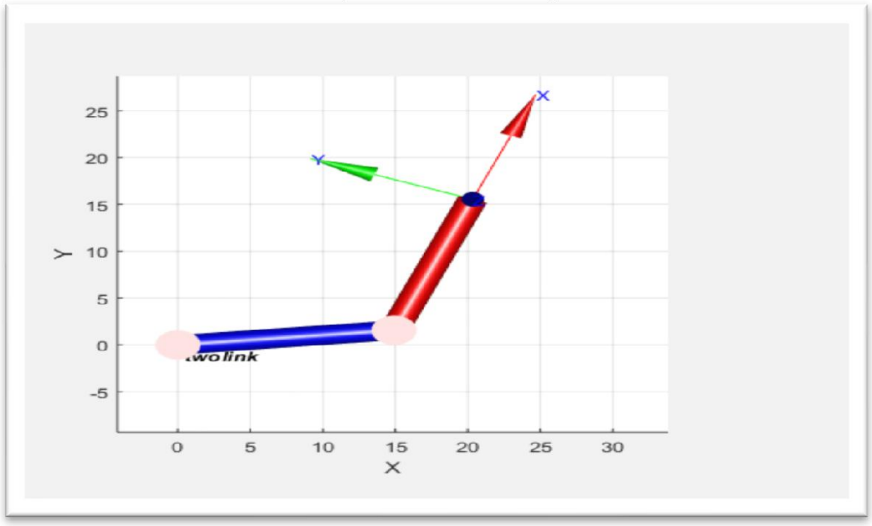


Figure 7: twolink moved to red rectangle in MATLAB

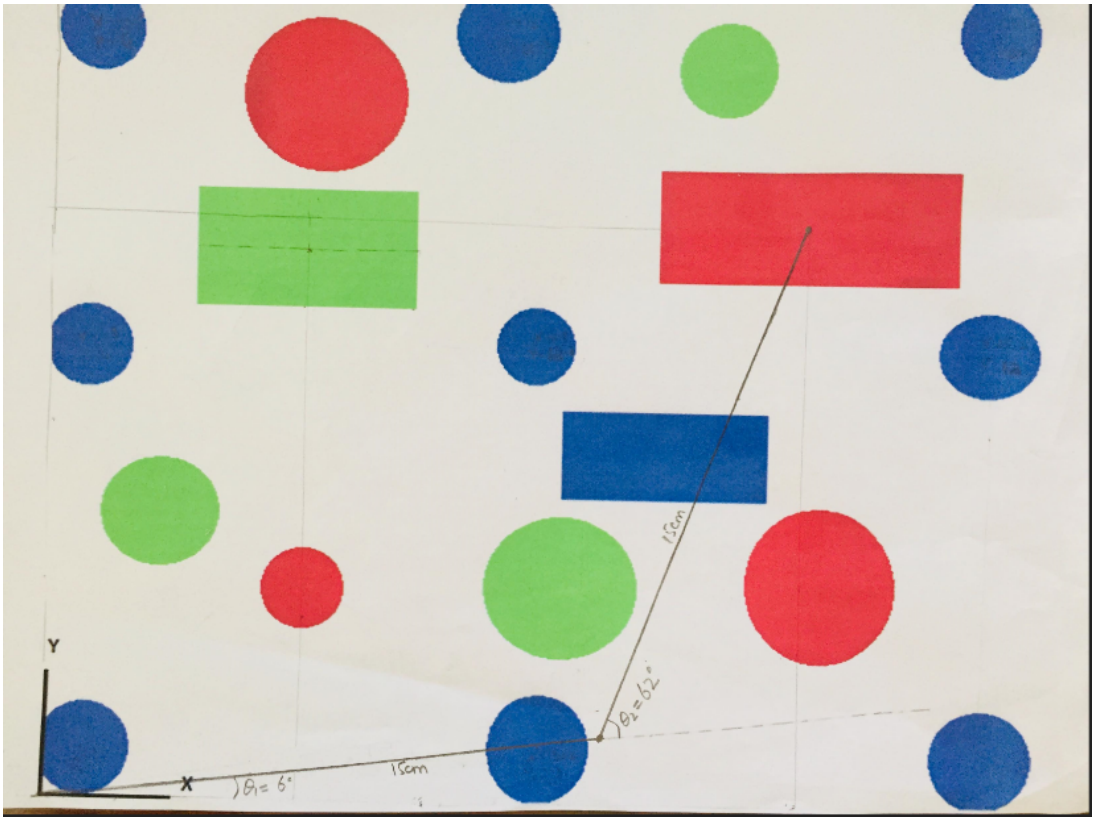


Figure 8: drawing twolink on worksheet for accuracy

5. Now I want to find the world coordinates of green rectangle,
   world coordinates of the red circle computed in MATLAB is
   x= 6.8226
   y= 14.7407
   And the actual coordinate measured from the ruler is
   x=6.8
   y=14.6
   By inverse kinematic for twolink robot we got the joint1 and joint2 angles
   theta1= 7.9448
   theta2= 114.4372

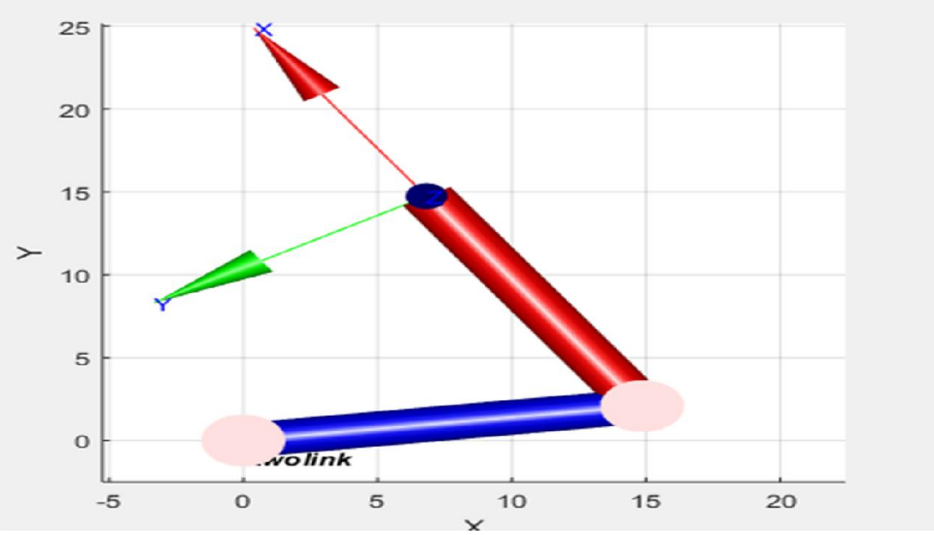


Figure 9: twolink moved to green rectangle in MATLAB

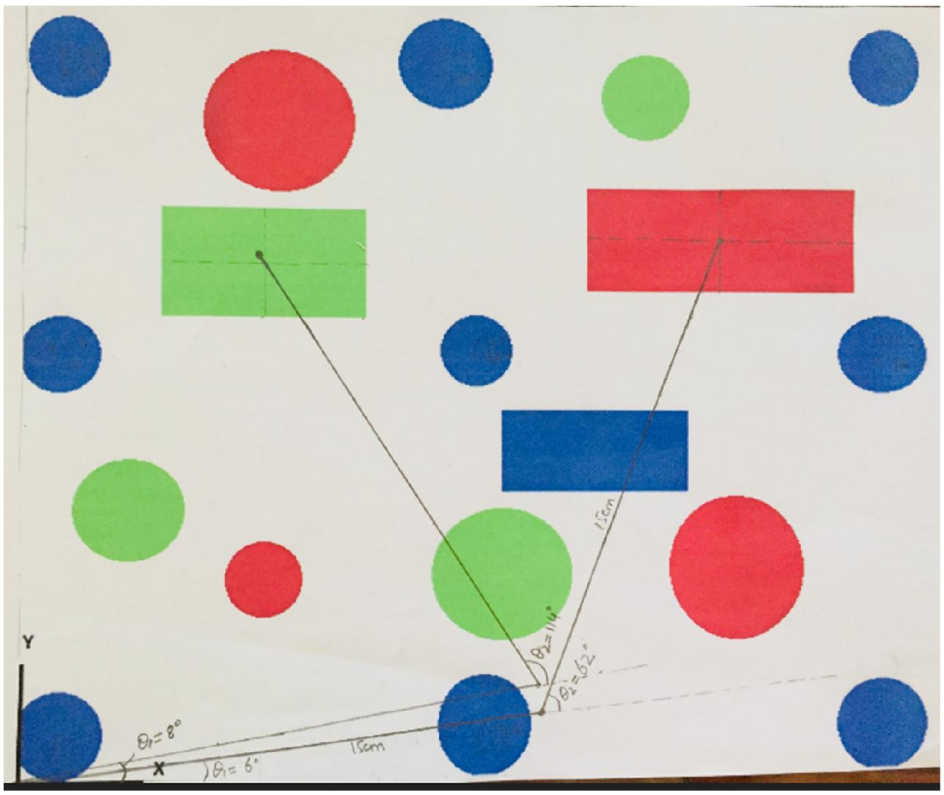


Figure 10: drawing twolink for green rectangle on vision worksheet

6. Now I want to find the world coordinates of red circle (top left corner),
   world coordinates of the desired red circle is
   x= 7.3568
   y= 18.8297
   And the actual coordinate measured from the ruler is
   x=7.4
   y=19
   theta1= 21.0252
   theta2= 95.2682

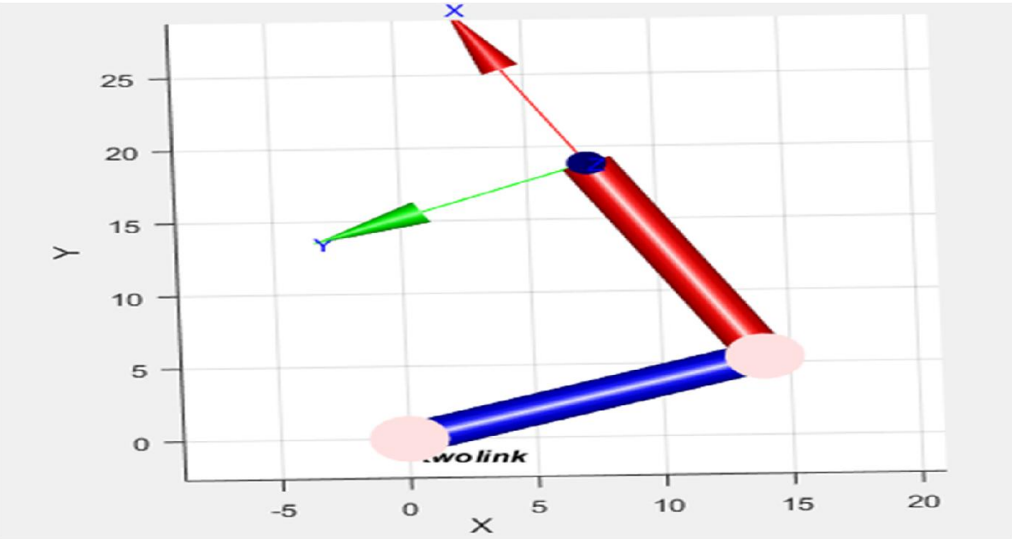


Figure 11: twolink moved to red circle in MATLAB

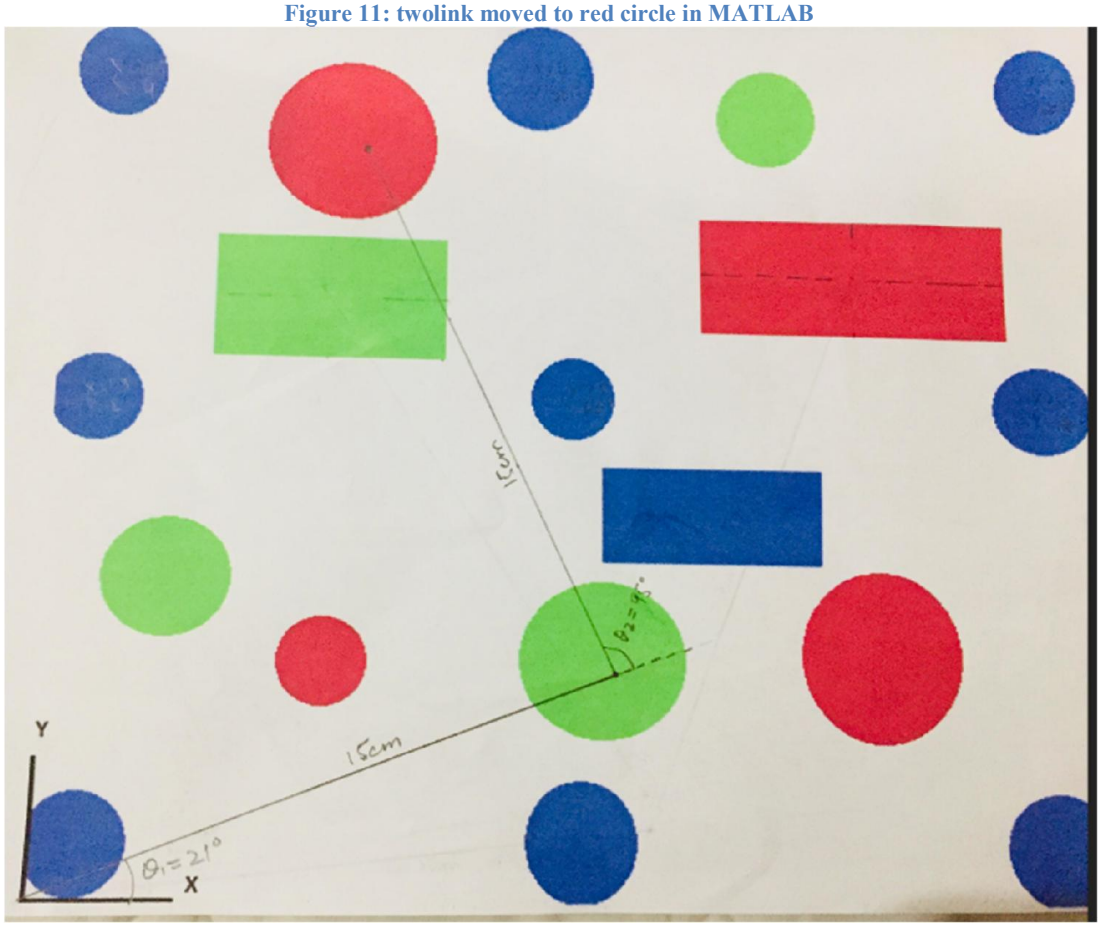


Figure 12: Drawing twolink for red circle on vision worksheet

7. **Twolink robot with system dynamics and PD-controller:**

Once we get the world coordinates(x,y) then we command the robot to move to that point, in the previous part we command twolink manipulator which had no system dynamics and no controller we assumed the idea case .but here I have computed the system dynamics and also there is independent joint controller in the robot [6].

I have created this whole system in MATLAB SIMULINK

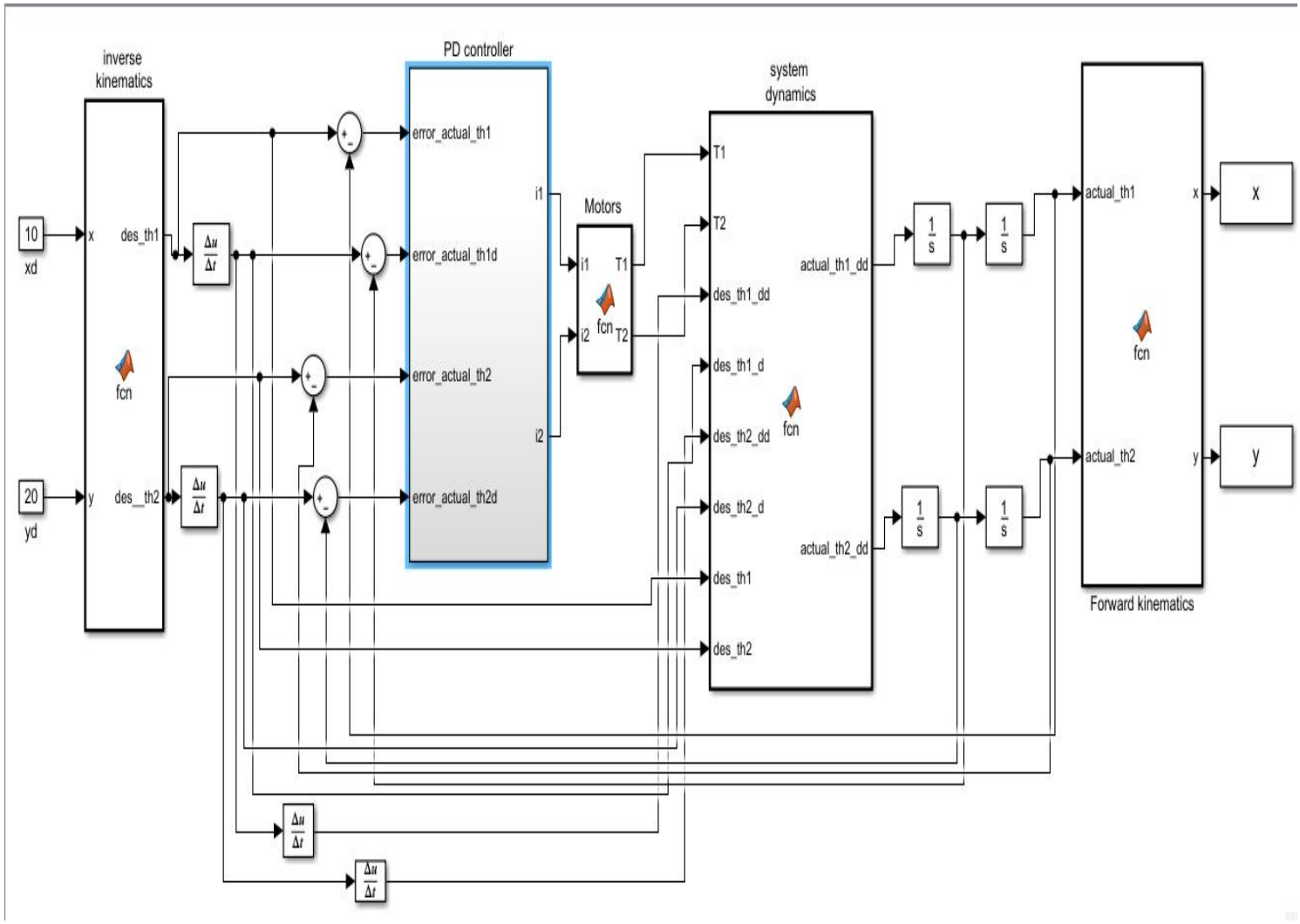


Figure 13: Twolink SIMULINK diagram

Let us suppose we give the input xd=10, yd=20 to check the performance of twolink robot and PD-controller. Our robot's initial position is (joint1=0, joint2=0), the end-effector coordinates will be then at this position is x=30cm, y=0cm

When kp1=80, kd1= 5, kp2= 80, kd2=5

before tuning PD-controller, there is overshoot and oscillation in our system and the robot didn't reached its desired input, so we need to tune our controller

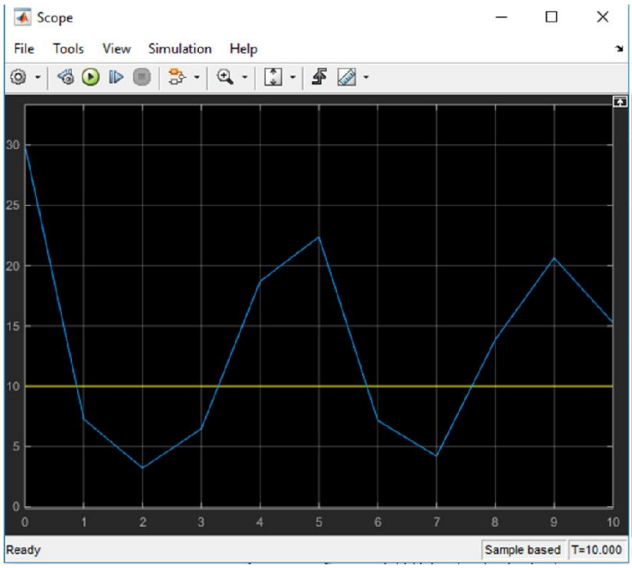


Figure 14: graph b/w desired x and actual x

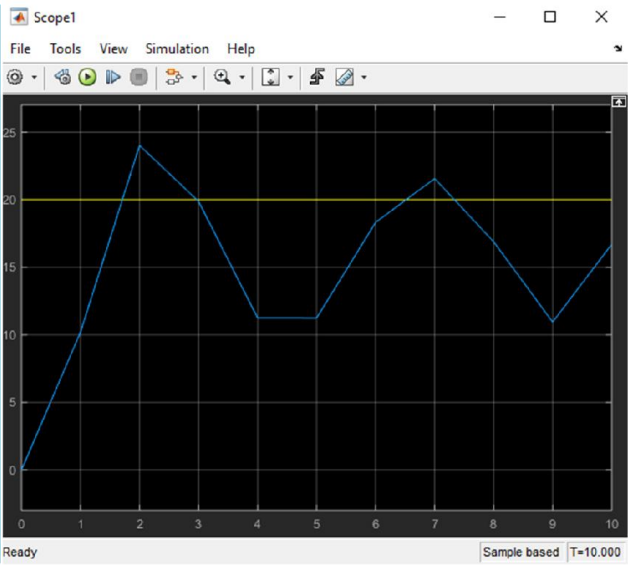


Figure 15: graph b/w desired x and actual x

When kp1=200, kd1= 150, kp2= 200, kd2=150

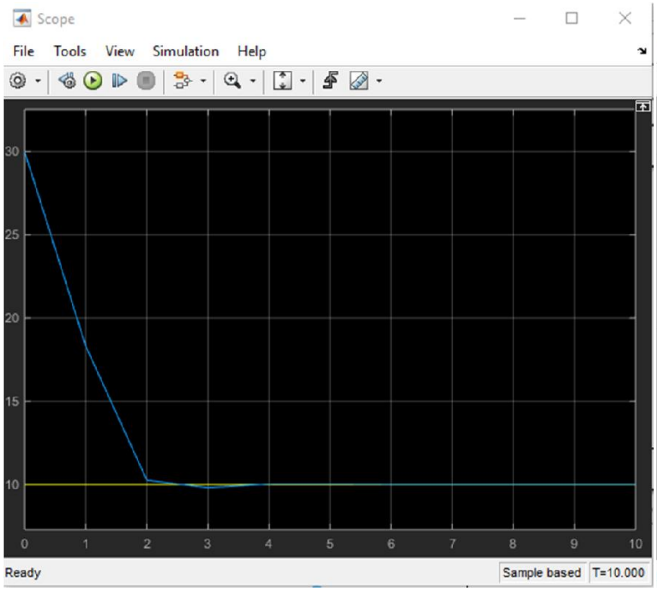


Figure 16: graph b/w desired x and actual x

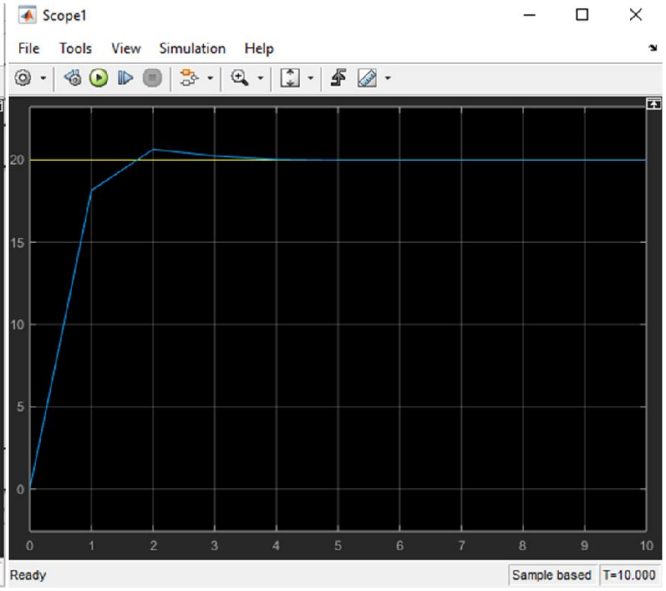


Figure 17: graph b/w desired y and actual y

Hence we have now also computed twolink manipulator having independent joint controller (inverse kinematics) and then we have tuned PD-controller so that overshoot and the oscillation in the system is minimum.

Our robot took 2 seconds to reach it desired x-input and almost 3seconds to reach its y-input after tuned my PD-controller. There is little bit overshoot in the y-axis and no steady state error in both graphs.

DISCUSSION AND CONCLUSION:

Our objective was to map the pixel values to the world coordinates and we have achieved our objectives very well.
We have presented an approach and also algorithm for finding the planar Homography between two planes, the algorithm and strategy can easily maps the pixel coordinates and the plane coordinates.
We can now find any point on the real plane by image pixels and we can obtain any point on the vision worksheet and our robot will go to that point.
This idea can be used in industrial robot where we have specific task to do by observing the environment in the camera, so we have to compute the world coordinates from the images.
There are many uses of these ideas which will help the robot to do their job perfectly.
Industrial Pick and place example is good enough to have concept of the vision and mapping the pixel coordinates to the world coordinates.

REFERENCE:

1. "Real World Coordinate from Image Coordinate Using Single Calibrated Camera Based on Analytic Geometry" by Joko Siswantoro, Anton Satria Prabuwono , and Azizi Abdullah.
2. "Robotics, Vision and Control Fundamental algorithms in MATLAB" by peter corke
3. Introduction to computer vision, Penn state university, instructor:Robert Collins
4. "A Robotics toolbox for MATLAB" by peter corke (http://www.petercorke.com/robot)
5. "A MATLAB toolbox for vision and vision-based control" by peter corke (http://www.petercorke.com/vision)
6. Introduction to Robotics, Mechanics and control by john j.Craig.

APPENDIX

1. I have written "getColor" function to easily detect color in MATLAB.

```
function color = getColor(im, clr, thr)
 %im=image stored, clr= "red/green/blue"  , thr= threshold for image
im=igamm(im,'sRGB');
Y = sum(im,3);
red = im(:,:,1) ./ Y;
green = im(:,:,2) ./Y;
blue = im(:,:,3) ./Y;

redBinary = red > thr;
greenBinary = green > thr;
blueBinary = blue > thr;

switch(clr)
case 'red'
color = redBinary;
case 'blue'
color = blueBinary;
case 'green'
color = greenBinary;

otherwise
fprintf('Invalid color option. Use red, blue or green in single quotes only.');
end
```

2. for finding the center of nine blue circle I have written the .m file and the code is shown below,

```
%findWorldCordinates%

% first of all find the Centroid of the blue circle to compare with the real worksheet points%
im=iread('sheet.jpg' ,'double');
idisp(im);
prompt = 'Enter the color name'; %enter blue here%
q = input(prompt,'s');
b=getColor(im,q,0.45);
idisp(b);
[label, m] = ilabel(b);
idisp(label, 'colormap', jet, 'bar') %shows the the unique color to the unique region
%select the label numbers of the blue circles whose Centroid points are required%
%circle 1= label 13
%circle 2= label 11
%circle 3= label 12
%circle 4= label 4
%circle 5= label 5
%circle 6= label 6
```

```
%circle 7= label 1
%circle 8= label 2
%circle 9= label 3 %

%for circle 1 %
figure
blob = (label == 13);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc1 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc1 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc1, vc1, 'gx', uc1, vc1, 'go');
hold on;

%for circle 2 %
figure
blob = (label == 11);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc2 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc2 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc2, vc2, 'gx', uc2, vc2, 'go');
hold on;

%for circle 3 %
figure
blob = (label == 12);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
```

```
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc3 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc3 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc3, vc3, 'gx', uc3, vc3, 'go');
hold on

%for circle 4 %
figure
blob = (label == 4);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc4 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc4 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc4, vc4, 'gx', uc4, vc4, 'go');
hold on

%for circle 5 %
figure
blob = (label == 5);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc5 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc5 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc5, vc5, 'gx', uc5, vc5, 'go');
hold on

%for circle 6 %
figure
blob = (label == 6);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
```

```
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc6 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc6 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc6, vc6, 'gx', uc6, vc6, 'go');
hold on

%for circle 7 %
figure
blob = (label == 1);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc7 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc7 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc7, vc7, 'gx', uc7, vc7, 'go');
hold on

%for circle 8 %
figure
blob = (label == 2);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc8 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc8 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc8, vc8, 'gx', uc8, vc8, 'go');
hold on

%for circle 9 %
figure
blob = (label == 3);
idisp (blob)
[v,u] = find(blob);
```

```
about (u)
umin = min(u)
umax = max(u)
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc9 = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc9 = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc9, vc9, 'gx', uc9, vc9, 'go');
hold on


%now put the values of all 9 circles in an array%
%note: the sequence of the points must be consistent%

p=[uc1 uc2 uc3 uc4 uc5 uc6 uc7 uc8 uc9 ;
   vc1 vc2 vc3 vc4 vc5 vc6 vc7 vc8 vc9]

%real worksheet points are%

x=[25.2 13.2 1 25.2 12.8 1.3 25.2 12 1.4 ;
    1.3 1.3 1.3 12 12.4 12 20.5 20.7 20.5]


H= homography(p,x); %we get the homogenous matrix that relates the pixel values to the real
coordinates%
```

3. once the Homography matrix is achieve we can now find the real coordinates of any colored and any shape object from the image, the code is below

```
%now to get the real coordinates for any colored object%

im=iread('sheet.jpg' ,'double');
idisp(im);
prompt = 'Enter the color name';
q = input(prompt,'s');
b=getColor(im,q,0.45);
idisp(b);
[label, m] = ilabel(b);
idisp(label, 'colormap', jet, 'bar') %shows the the unique color to the unique region
prompt = 'Enter value of the seperation';
x = input(prompt);
blob = (label == x);
idisp (blob)
[v,u] = find(blob);
about (u)
umin = min(u)
umax = max(u)
```

```
vmin = min(v)
vmax = max(v)
plot_box(umin, vmin, umax, vmax, 'g')
m00 = mpq(blob, 0, 0) %area of th region in units of pixels
uc = mpq(blob, 1, 0) / m00 %centroid in U pixels
vc = mpq(blob, 0, 1) / m00 %cenroid in V pixels
hold on;
plot(uc, vc, 'gx', uc, vc, 'go');
p1=[uc
   vc];
x1=homtrans(H,p1);
xd=x1(1)    %x-coordinate of the desired object
yd=x1(2)    %y-coordinate of the desired object
```

4. Now we have x,y values of the desired object, now I want my twolink manipulator to move to that desired x,y point.

```
L1=15;   %link 1 length (cm)
L2=15;   %link 2 length (cm)
%inverse-kinematics
c2= (xd^2 +yd^2 -L1^2 -L2^2)/(2*L1*L2);
s2=(1-c2^2)^0.5;
K1= L1+(L2*c2);
K2=L2*s2;
des_th1= atan2 (yd,xd)-atan2(K2,K1);
des_th2=atan2 (s2,c2);
q=[des_th1 des_th2]

%% DH-table of twolink
L(1) =Link([ 0 0 15 0 0])
L(2) =Link([0 0 15 0 0])
figure
robot=SerialLink(L)
robot.name='twolink'
robot.plotopt={'workspace', [-30 30 -30 30 -30 30]}
robot.plot(q);
```

5. TWOLINK MANIPULATOR DYNAMICS (SIMULINK)

Explaining the terminology used by me in the SIMULINK model

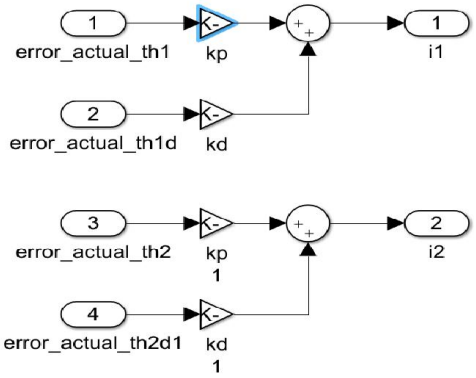
cm1=frictional constant for motor1=0.1
cm2=frictional constant for motor2=0.1
Jm1=inertia constant for motor1=0.01
Jm=inertia constant for motor2=0.01
M1=mass of link1=1kg
M2=mass of link2=0.5kg
g=gravitational acceleration=0.098 cm/s^2
kt=motor constant =10
PD controller is used

xd=desired x-values
yd=desired y-values
des_th1=Desired joint 1 position
des_th2=Desired joint 2 position
des_th1_d=Desired joint 1 velocity
des_th2_d=Desired joint 2 velocity
des_th1_dd=Desired joint 1 acceleration
des_th2_dd=Desired joint 2 acceleration
actual_th1= actual joint 1 position
actual_th2= actual joint 2 position
actual_th1_d= actual joint 1 velocity
actual_th2_d= actual joint 2 velocity
actual_th1_dd= actual joint 1 acceleration
actual_th2_dd= actual joint 2 acceleration
Tt1= motor 1 torque
Tt2=motor 2 torque
T1=total torque joint 1
T2=total torque joint 2
X=Actual X-values
Y=Actual Y-values

INVERSE KINEMATICS:

```
function [des_th1,des__th2]= fcn(x,y)
L1=15;
L2=15;
M1=1;
M2=0.5;
c2= (x^2 +y^2 -L1^2 -L2^2)/(2*L1*L2)
s2=(1-c2^2)^0.5
K1= L1+L2*c2
K2=L2*s2
des_th1= atan2 (y,x)-atan2(K2,K1);
des__th2=atan2 (s2,c2);
```

PD-CONTROLLER:



MOTORS:

```
function [T1,T2]= fcn(i1,i2)
 kt=10;
 T1=kt*i1;
 kt=10;
 T2=kt*i2;
```

SYSTEM DYNAMICS:

```
function [actual_th1_dd,actual_th2_dd]= fcn(T1,
T2,des_th1_dd,des_th1_d,des_th2_dd,des_th2_d,des_th1,des_th2)

N=1;
jm1=0.01;
jm2=0.01;
cm1=0.1;
cm2=0.1;
M1=1;
M2=0.5;
L1=15;
L2=15;
g=9.8;

th1=des_th1
th2=des_th2
th1dd=des_th1_dd
th1d=des_th1_d
th2dd=des_th2_dd
th2d=des_th2_d
M=[((M1+M2)*(L1^2)+ (M2*L2^2)+ (2*M2*L1*L2*cos(th2)))  (M2*L2^2+
M2*L1*L2*cos(th2));
  (M2*L2^2+M2*L1*L2*cos(th2))                (M2*L2^2)];
V=[((-M2*L1*L2)*(2*th1d*th2d+th2d^2)*sin(th2));
```

```
  (M2*L1*L2*th1d^2*sin(th2))];


 G= [((M1+M2)*g*L1*cos(th1)+M2*g*L2*cos(th1+th2));
    M2*g*L2*cos(th1+th2)];

 Tt1= N*T1 -N^2* jm1*des_th1_dd -N^2*cm1 *des_th1_d
 Tt2= N*T2 -N^2* jm2*des_th2_dd -N^2*cm2 *des_th2_d

NJ=[N*jm1 0;
  0 N*jm2]

NT=[N*T1;
N*T2]

NC=[N*cm1 0;
0 N*cm2]

thd=[des_th1_d;
des_th1_d]

actual_th_dd= inv (M+NJ) *(NT -V- G-(NC*thd));
v= actual_th_dd(1)
w=actual_th_dd(2)
actual_th1_dd=v
actual_th2_dd=w
```

FORWARD KINEMATICS:

```
 function [x,y]= fcn(actual_th1,actual_th2)

 L1=15;
 L2=15;

 x=L2*cos(actual_th1 + actual_th2) + L1*cos(actual_th1)
 y=L2*sin(actual_th1 + actual_th2) + L1*sin(actual_th1)
```