Muhammad Basit Iqbal Awan

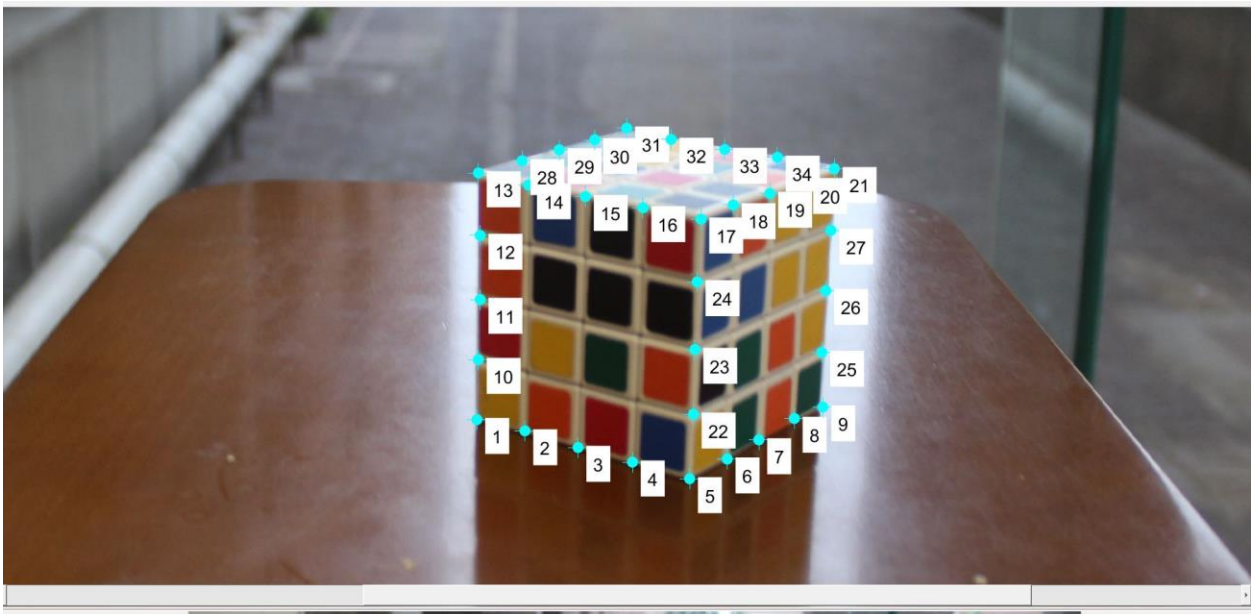
20100153

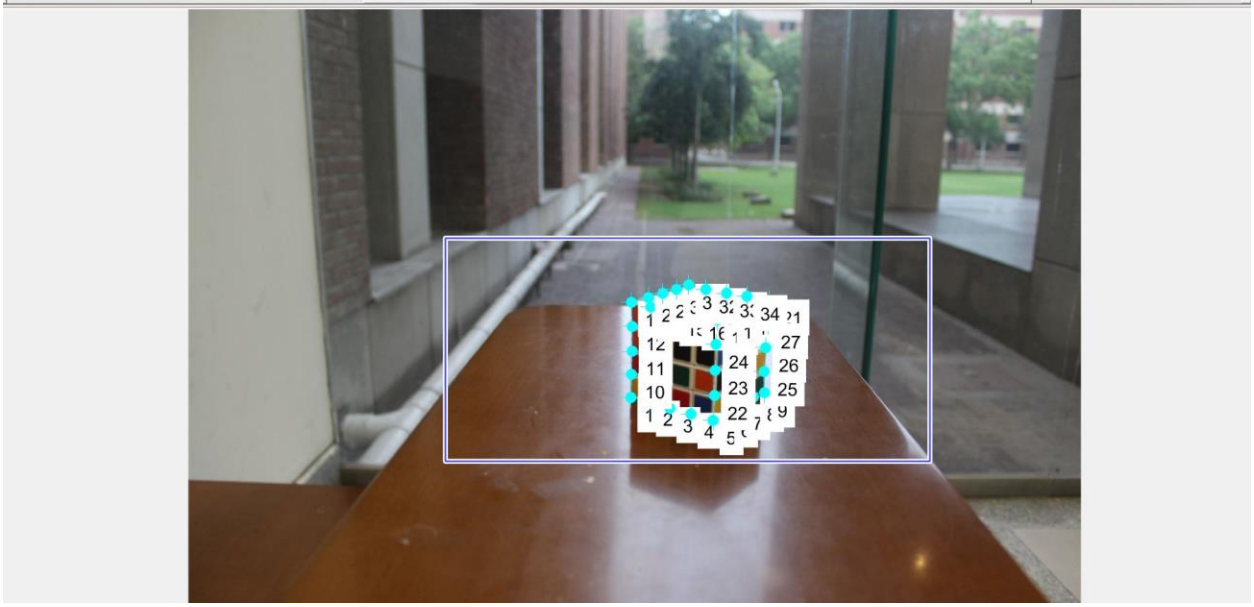
Using 1 late day.

Problem 1:

a)

I started the assignment by taking an image of a 4x4 cube with a nice background. I then used cpselect function to mark 34 points on the cube, as shown below





I then saved these points into a variable **imagePoint** and loaded them into the script for use. I also swapped the first and second column because Matlab doesn't follow our convention of coordinate system. I then manually put in all corresponding world points in coordinate form and then multiplied them with the dimension of each cube which was 16.5mm.

I then formed the matrix A by using a for loop which in every iteration adds the elements of each point into the A matrix. This is the code:

```
%Converting the points into A, as discuessed in class
A = [worldPoints(1,1), worldPoints(1,2),worldPoints(1,3), 1, 0,0,0,0,(-imagePoints(1,1))*worldPoints(1,1),
    0,0,0,0,worldPoints(1,1), worldPoints(1,2),worldPoints(1,3), 1, (-imagePoints(1,2))*worldPoints(1,1),(-
for i=  2:34
    A = [A;worldPoints(i,1), worldPoints(i,2),worldPoints(i,3), 1, 0,0,0,0,(-imagePoints(i,1))*worldPoints
        0,0,0,0,worldPoints(i,1), worldPoints(i,2),worldPoints(i,3), 1, (-imagePoints(i,2))*worldPoints(i,
end
```

I am sorry I couldn't get to display the whole code because I couldn't enable text warping.

After finding A I found it's Null Vector by taking it's SVD and getting the last column of V. This is how I found P, I then reshaped P. After which I converted worldPoints into P3 space and then applied P to all of them. Then I normalized the new image points which are in the variable imagePoint1. I computed the Error by taking the norm of the difference and then dividing it by the number of points as shown below:

```
%Found the NULL vector through SVD, by finding the last column of V
[U,S,V] = svd(A);
P= V(:,end);
P= [P(1:4,:)';P(5:8,:)';P(9:12,:)'];
worldPoints = worldPoints'; %Converting the worldPoints into column vectors form.
worldPoints = [worldPoints;ones(1,34)]; %converting world points into P3 space
imagePoint1 = P*worldPoints;
c = imagePoint1;
c(1,:) = c(1,:)./c(3,:);
c(2,:) = c(2,:)./c(3,:);
c(3,:) = c(3,:)./c(3,:);
imagePoint1 = c;
imagePoints = imagePoints';%Converting the imagePoints into column vectors form.
imagePoints = [imagePoints;ones(1,34)];%converting imagepoints into P3 space
Error=norm(imagePoint1-imagePoints,2)/34
```

The P vector came out as:

P =

-0.0001 0.0003 -0.0026 0.6585

0.0013 0.0026 -0.0003 0.7526

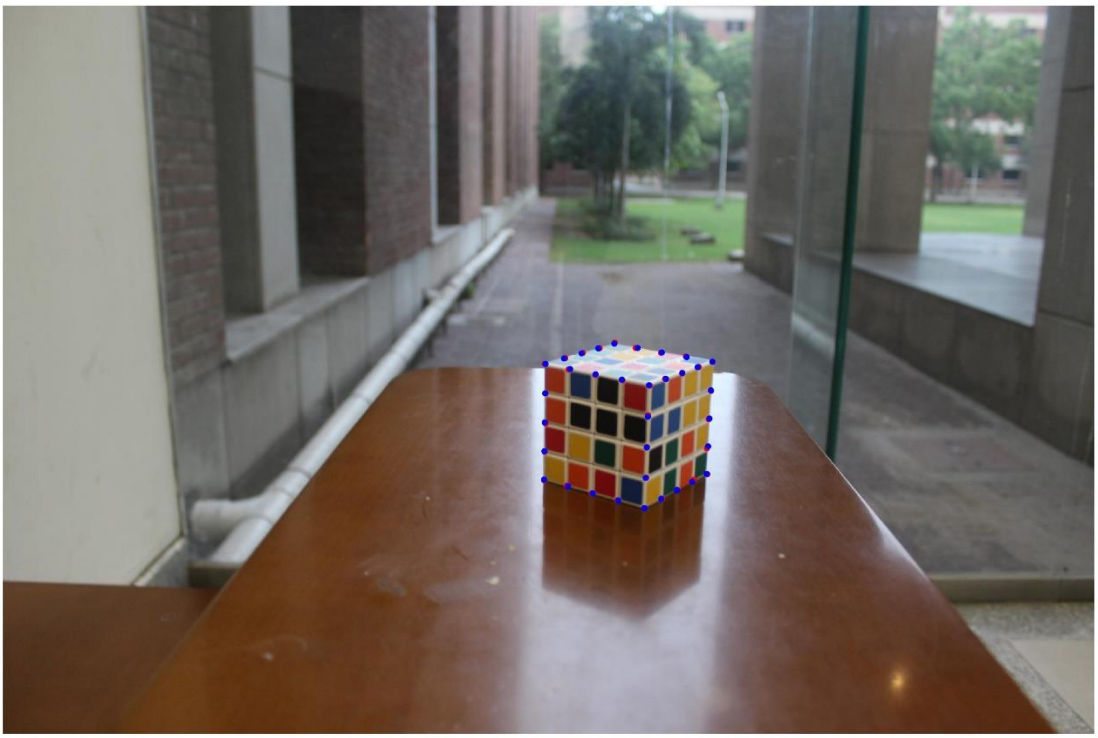
-0.0000 0.0000 -0.0000 0.0003

The error I got was:

Error = 0.7980

Which is not that bad.

I then plotted the points using plot function with red points represent originally marked points and blue points represented the newly project points. The plot came out to look like this:



b) I manually inputted the points again, and the result didn't change by a lot as the error was already too small. In one case I took the points again and the answer was significantly worse then before, but I had overwritten over the previous points. I took more images but at the end this image had more distinctive points as the others did not focus on the cube correctly, so I think this one would have been the better choice.

c) I computed camera center C by taking the null vector of P. I then got K and R from the RQ decomposition of the first 3x3 matrix. Also placed a check to check if R indeed had a determinant +1 (to check if it was truly a rotation) and if it did not fix it by multiplying both R and K by -1.

```
48 -    c = null(P);
49
50 -    c = c/c(4,1); %normalize c
51
52 -    [K, R] = rq(P(1:3,1:3))
53 -    if det(R)<0
54 -        R = -R
55 -        K= -K
56 -    end
```

The values came out as:

c =

329.1836

-425.5706

201.0046

1.0000

```
K =

   1.0e+03 *

   -4.8682   -0.0314    1.6031
         0   -4.8734    2.9234
         0         0    0.0010
```

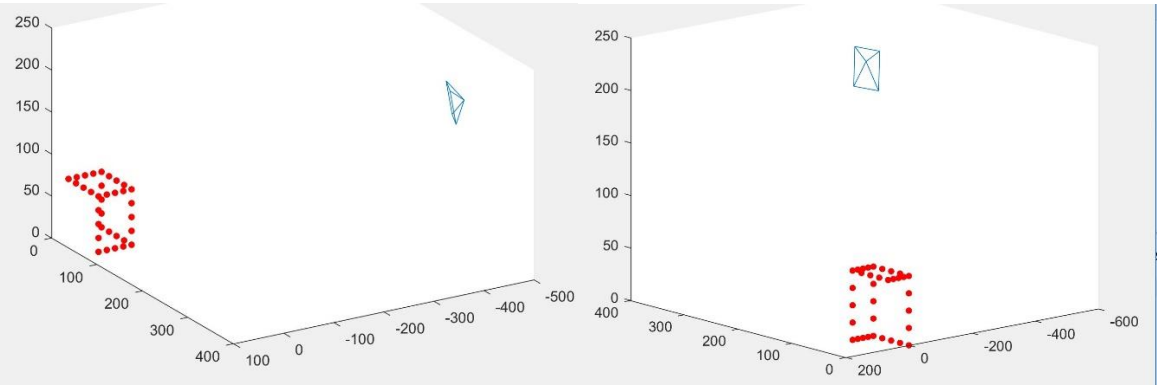
```
R =

    0.1471   -0.1687   -0.9746
    0.8291    0.5584    0.0285
    0.5394   -0.8123    0.2220
```

I then made a camera like structure which opened up in the -Z direction(if I didn't do that the structure was pointing to the other direction) and then rotated it with the inverse of R and then translated it by the c vector. I then plotted the 3D coordinates and the camera like structure using this code :

```
figure()
 plot3(worldPoints(1,:), worldPoints(2,:),worldPoints(3,:), 'r.', 'LineWidth', 2, 'MarkerSize', 15);
 hold on;
 cam = [0,0,0;1,1,1;1,-1,1; 0,0,0;-1,1,1;1,1,1;0,0,0;-1,-1,1;-1,1,1;-1,-1,1;1,-1,1];
 cam = -cam*20; %Made the opening in the opposite direction so the opening faces the cube.
 Cam = (R'*cam')+c(1:3,:);
 plot3(Cam(1,:),Cam(2,:),Cam(3,:));
```

These are some screenshots with different angles of the plot, I have also made this into a different figure so when the script runs this opens up as well.



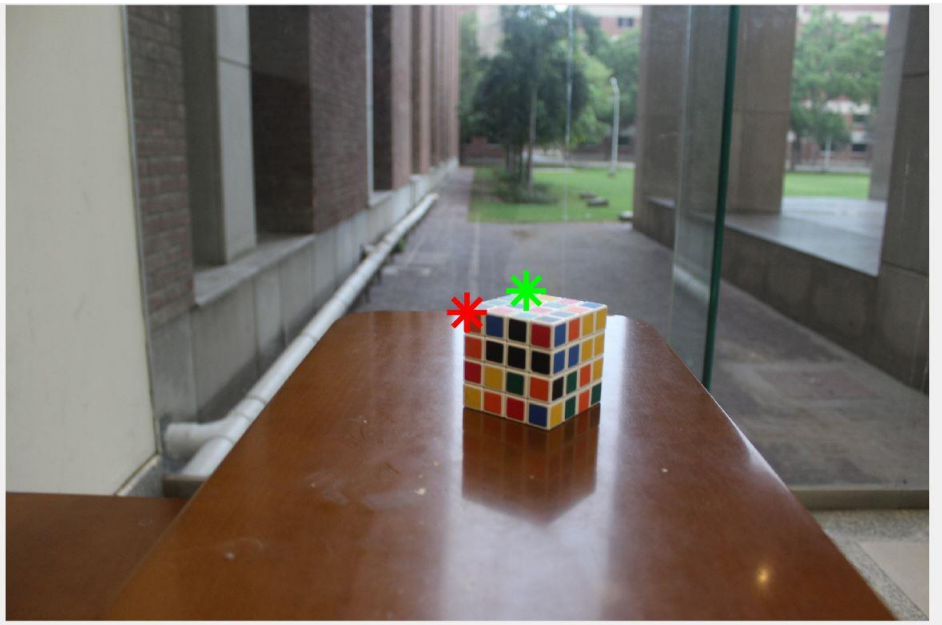
d) I looked up the sensor data, the sensor size was 22.3 x 14.9 mm and resolution 5184 x 3456. So I first solved from mx and my and from there solved for focal length in both direction and then averaged them out. I also took the angle between the x and y axis. Code as shown below:

```
66    %part d
67 -  K= K/K(3,3);
68 -  m_x = 5184/22.3;
69 -  m_y = 3456/14.9;
70
71 -  f1 = abs(K(1,1))/m_x;
72 -  f2 = abs(K(2,2))/m_y;
73 -  f= (f1+f2)/2
74
75 -  u = [K(:,1)];
76 -  v = [K(:,2)];
77 -  CosTheta = dot(u,v)/(norm(u)*norm(v));
78 -  AngleBetweenAxis = acosd(CosTheta); %calculating the angle between x axis and y
79
```

The focal length came out to be 20.9762, which is very close to the EXIF data which shows it to be 18mm.

The angle came out to be 89.6312, which is very close to 90.

e) I got the last column of K and normalized it to get the principal point. The principal point is in green and the center of the image is in red.



For center of image I just got the size of the image and divided it by two. I plotted the points as shown below:

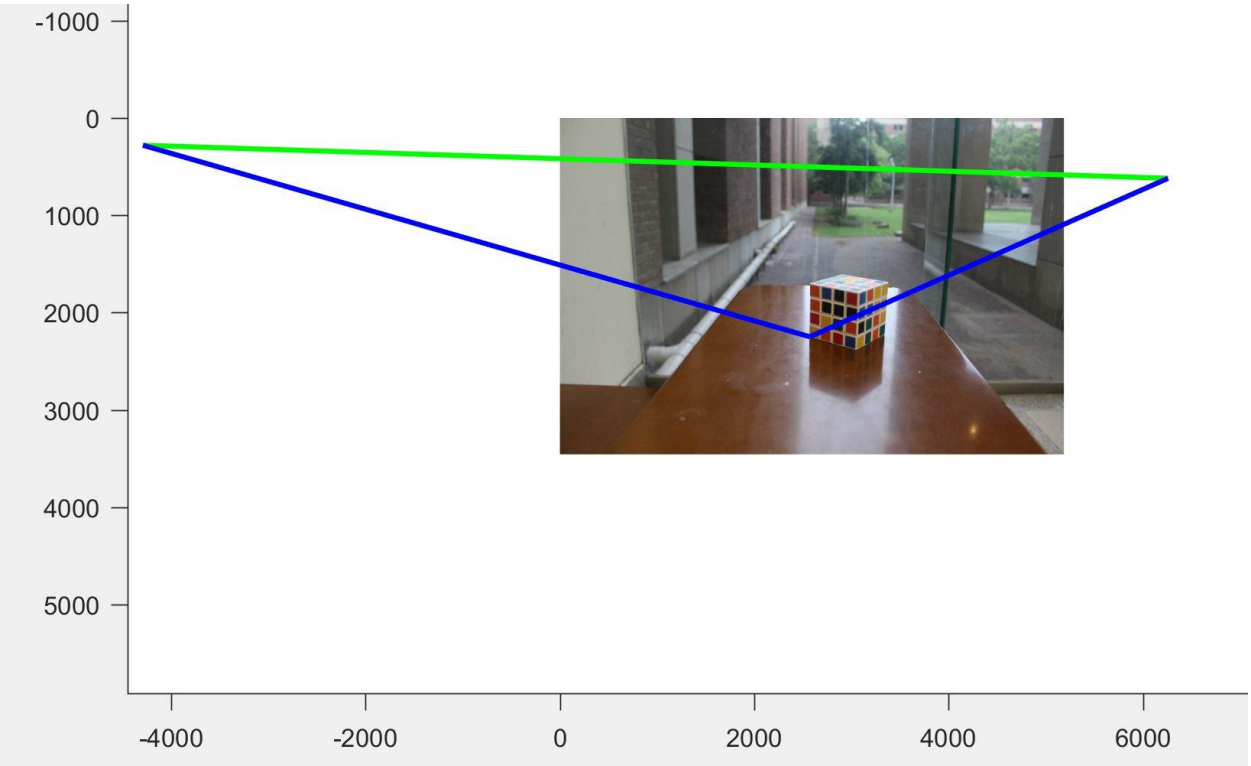
```
%part e
figure()
myImage = imread('cube.jpg');
imshow(myImage);
hold on;
plot(K(2,3),K(1,3),'g*' ,'LineWidth', 2, 'MarkerSize', 15);
[y1,x1,~] = size(myImage)
plot(x1/2,y1/2,'r*' ,'LineWidth', 2, 'MarkerSize', 15);
```

f) I found the point at infinity along the x-axis, y axis, and also the origin of the world from P matrix. The first two column being the points at infinity and the last being the origin. I normalized the points and

drew lines between them by the code shown below :

```
figure()%generated new figure to show horizon line
myImage = imread('cube.jpg');
imshow(myImage);
hold on;
%made the horizon line first
plot([X_inf(2),Y_inf(2)],[X_inf(1),Y_inf(1)],'g' ,'LineWidth', 2, 'MarkerSize', 15);
%made a line between origin and point at infinity along yaxis
plot([origin(2),Y_inf(2)],[origin(1),Y_inf(1)],'b' ,'LineWidth', 2, 'MarkerSize', 15);
%made a line between origin and point at infinity along yaxis
plot([X_inf(2),origin(2)],[X_inf(1),origin(1)],'b' ,'LineWidth', 2, 'MarkerSize', 15);
```

This is what the plot looked like:



g) I first found the line of the horizon in P3, by cross multiplying both points at infinity. I then found the normal of the plane formed from the back projection of this line by multiplying K transpose with the l. When I got n, I found the angle between it's normal and the camera's normal, which is the tilt between the plane and the camera. I then got the equation of the plane in canonical view by taking the transpose of [K|**0]** (the camera is in the canonical view hence this is it's P) and multiplying that by the line. The code is as shown below :

```
%part g
l = cross(X_inf, Y_inf); %got line from cross product along horizon
d = l; %normalized it
d(1,:) = d(1,:)./d(3,:);d(2,:) = d(2,:)./d(3,:);d(3,:) = d(3,:)./d(3,:);l = d;
n = K'*(l); %got normal with respect to the plane
u = n;
v = [1; 0; 0] ; %normal with respect to the camera
tilt = atan2d(norm(cross(u,v)),dot(u,v)); %found angle from both normals

PlaneEquation = [K(1,:),0;K(2,:),0;K(3,:),0]'*l;
```

The tilt was : **12.934 degrees**

and the equation of the plane was

PlaneEquation =

12.4869

-0.3655

-2.8443

0

h) The way I found the principal point in 3D was that I first placed the principal point in the canonical view is f coordinates away from camera center at the Z axis. I then rotated then point by the inverse of R and then translated it by C. To get where the coordinates would be in the 3D space.

```
Principal3d = [0;0;f]; %principal point is 1
Principal3d = (R'*Principal3d) + c(1:3,:) %
```

The coordinates were this:

Principal3d =

340.4976

-442.6091

205.6614

I) For this part I found the indexes of the right most and left most point by using max and min function. Then I used those point to get me a back-projected ray in the word. I used inverse of K as I assumed it to

be canonical view. I then got the angle of these two rays.

```
%part i
[M,I] = max(imagePoints');
maximum = I(2);
[M,I] = min(imagePoints');
minimum = I(2);
maximumRay = inv(K)*imagePoints(:,maximum);
minimumRay = inv(K)*imagePoints(:,minimum);
u = maximumRay;
v = minimumRay;
CosTheta = dot(u,v)/(norm(u)*norm(v));
AngleBetweenRays = asind(CosTheta);
```

,

The angle between the rays came out to be:

78.5316 degrees

Challenge Problem)

I first looked over the Blue Mosque to find similar structures near it. I found the Obelisk_of_Theodosius and the Walled Obelisk. I made the base of the Obelisk_of_Theodosius as my origin as shown below and calculated the world points by using Google Earth and Wikipedia for accurate height. I made sure that my coordinate system was right handed :

I then calculated the world coordinates and got the values below.

Obelisk_of_Theodosius 0,0,0 = ground point, 0,0,25.6 = at peak

Walled Obelisk 73.76,0,0= ground point, 73.76,0,0, 32 = at peak

Right most minar at peak = 0, 64.8, 64

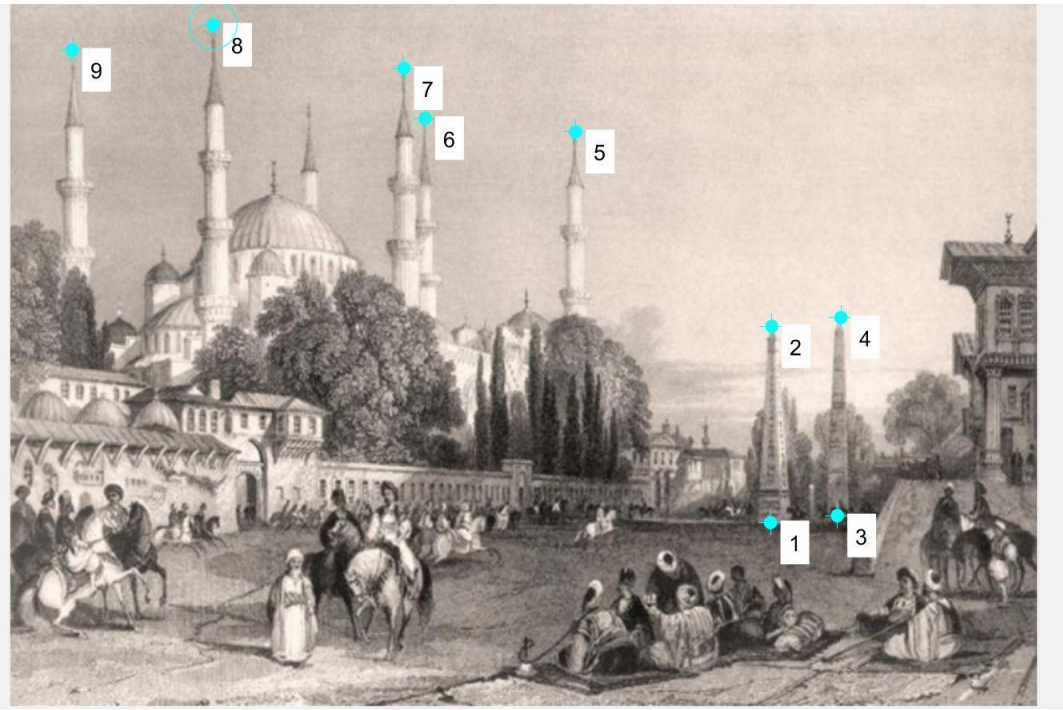
Second from the right at peak = 0,119.7,64

Third from the right minar at at peak = -61.16,70.4,64

Second minar from the left at peak -61.16, 125.31,64

Left most minar peak = -61.16, 175.21,64

I then used cpselect to mark correspondences in the image as showed below and saved those points in a variable bluePoints.



Obelisk_of_Theodosius 0,0,0 ground point 0,0,25.6

Walled Obelisk 73.76,0,0, peak 73.76,0,0, 32

Right most minar at peak 0, 64.8, 64

Second from the right at peak 0,119.7,64

Third from the right minar at ground at peak = -61.16,70.4,64

Second dome from the left at peak -61.16, 125.31,64

Left peak = -61.16, 175.21,64

I then made the A matrix and from it got the P matrix. Then I took it's null and got the center of the camera coordinates.

Center =

-244.0376

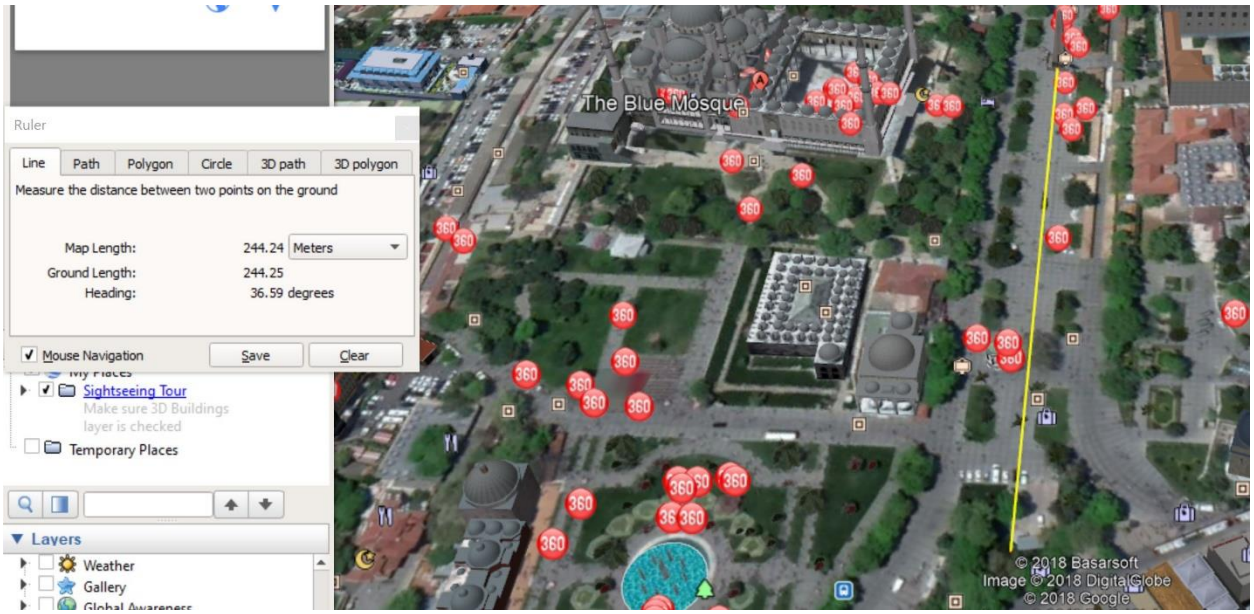
-100.7609

16.2635

1.0000

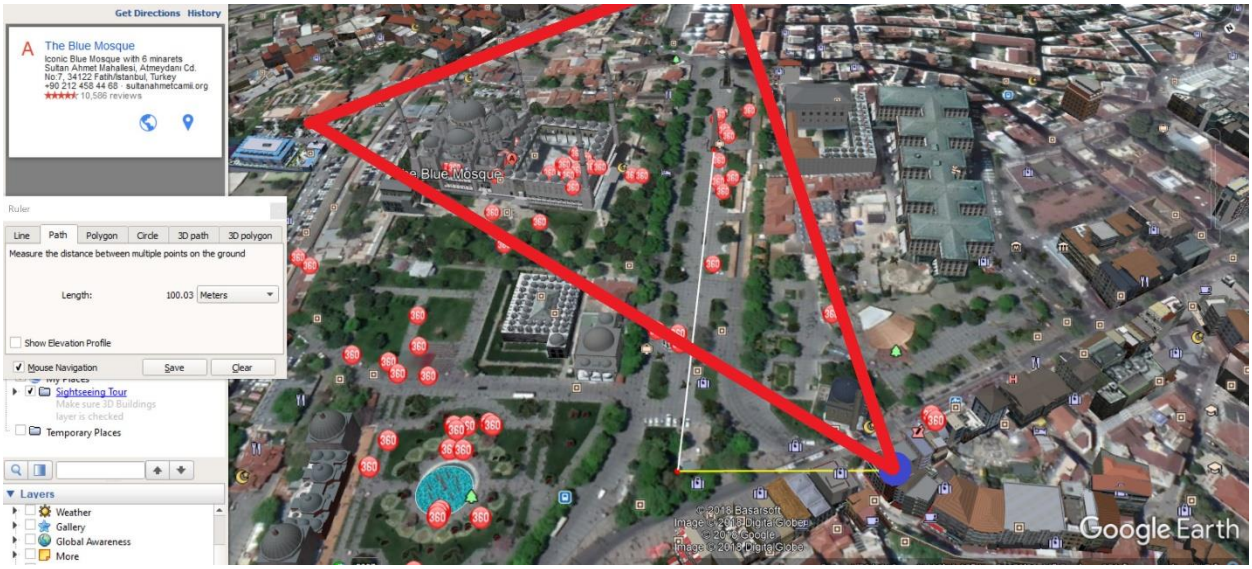
I then went back to google earth and used the scale feature again to get to the point.

This is going back in the -x direction.



This going back into the -y direction:

Then I finally found the point and drew the triangle to represent the viewpoint of the artist.



I also then went into 3D view to see the perspective form the point and looked something like this:

This is zoomed, but it is very close to the original point from where the artist drew the sketch.

Oh this is the code I used :

```
load BluePoints %Image points loaded from set of data I already marked
A = BluePoints;
t1 = A(:,1);t2 = A(:,2);A(:,1) = t2;A(:,2) = t1;
BluePoints = A; %Swapping y axis and x axis as matlab does not follow image convention
worldPoints = [0,0,0;0,0,25.6;73.76,0,0;73.76,0,32;0,64.8,64;0,119.7,64;-61.16,70.4,64;
               -61.16,125.31,64;-61.16,175.21,64]
imagePoints=BluePoints
A = [worldPoints(1,1), worldPoints(1,2),worldPoints(1,3), 1, 0,0,0,0,(-imagePoints(1,1))*worldPoints(1,1),(-imag
    0,0,0,0,worldPoints(1,1), worldPoints(1,2),worldPoints(1,3), 1, (-imagePoints(1,2))*worldPoints(1,1),(-image
for i=  2:9
    A = [A;worldPoints(i,1), worldPoints(i,2),worldPoints(i,3), 1, 0,0,0,0,(-imagePoints(i,1))*worldPoints(i,1),
        0,0,0,0,worldPoints(i,1), worldPoints(i,2),worldPoints(i,3), 1, (-imagePoints(i,2))*worldPoints(i,1),(-
end
[U,S,V] = svd(A);
P= V(:,end);
P= [P(1:4,:)';P(5:8,:)';P(9:12,:)'];
Center = null(P);
Center = Center/Center(4)
```

Which is very similar to the one in the last question.