# Homework 5

| **Meta information** | |
|---|---|
| Name | Savan Kiran |
| Program | Masters in Computer Science |
| Questions skipped | N/A |
| Questions substituted | N/A |
| Extra credit questions | N/A |

## PART A

In the given image, we note that the X-axis goes left from center, Y-axis goes right from center and Z-axis goes up from center and that the grid lines are 1 inch apart. Using these information, we calculate the world coordinates for 15 points. Table 1 shows the list of points.

Now, we use the Matlab's data cursor tool to identify the image coordinates for these 15 points in Standard Image Coordinate System. The values are shown in Table 2.

| x | y | z |
|---|---|---|
| 2 | 0 | 1 |
| 3 | 0 | 2 |
| 2 | 0 | 2 |
| 1 | 0 | 3 |
| 1 | 0 | 1 |
| 0 | 1 | 2 |
| 0 | 3 | 1 |
| 0 | 2 | 3 |
| 0 | 3 | 4 |
| 0 | 2 | 2 |
| 1 | 4 | 0 |
| 2 | 1 | 0 |
| 1 | 3 | 0 |
| 4 | 4 | 0 |
| 3 | 2 | 0 |

Table 1. 15 points in world coordinates

Using the points information, we calculate the Camera Matrix, `M` and try to replot the 15 points in Standard Image Coordinates.

We first calculate the matrix, `P` which has 2 rows added for each point given by, $P_i$

$$row1 = (P_i', 0000, -u_i P_i')$$
$$row2 = (P_i', 0000, -v_i P_i')$$

| u | v |
|---|---|
| 630 | 468 |
| 456 | 369 |
| 620 | 321 |
| 766 | 129 |
| 772 | 422 |
| 1012 | 315 |
| 1204 | 624 |
| 1119 | 232 |
| 1244 | 137 |
| 1109 | 390 |
| 1155 | 931 |
| 723 | 709 |
| 1050 | 832 |
| 683 | 1155 |
| 659 | 870 |

Table 2. Coordinates of 15 points in Standard Image Coordinate System

From `P`, we calculate the Camera Matrix, `M` using homogeneous least squares as follows,

```
TP=P'*P;
[V,D]=eig(TP);
m=V(:,1);

M=reshape(m,4,3)';
m1=M(1,:)';
m2=M(2,:)';
m3=M(3,:)';
```

The reported Camera Matrix, `M` is,

```
Camera Matrix:
   -0.1459    0.0567   -0.0231    0.8473
    0.0291    0.0580   -0.1383    0.4835
   -0.0000   -0.0000   -0.0000    0.0009
```

Once we have the components of Camera Matrix, `M`, we can calculate the projected points as follows,

$$u_i = \frac{m_1.P_i}{m_3.P_i}$$
$$v_i = \frac{m_2.P_i}{m_3.P_i}$$

We now plot the projected points (shown in blue) alongside the actual image points (shown in red) in the Standard Image Coordinate System which is depicted in Figure 1.

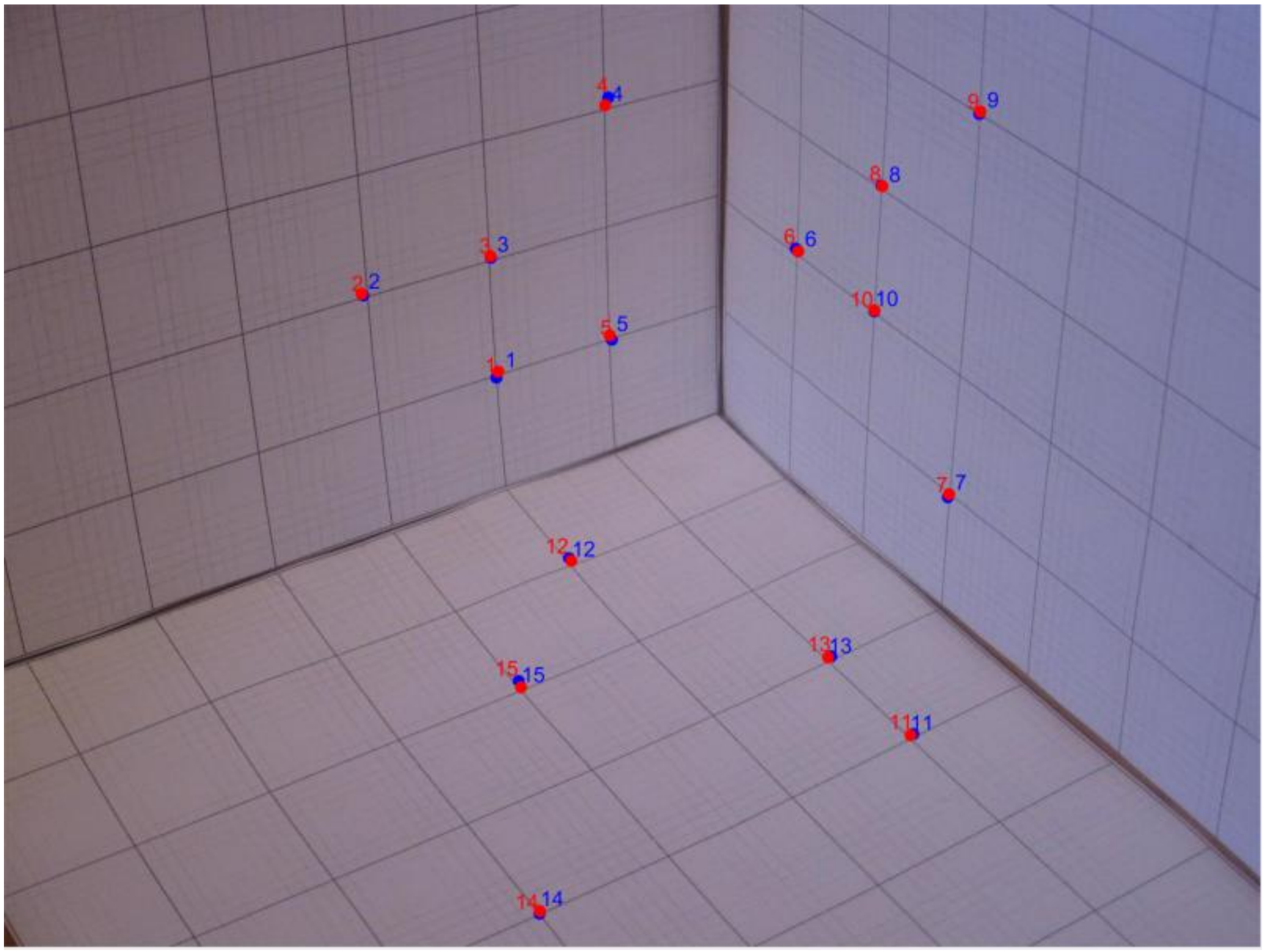

Figure 1. Blue shows projected points and red shows actual image coordinate points plotted in Standard Image Coordinate System

The RMS error for this Camera Matrix, `M` is calculated to be `4.2811`.

We found a better Camera Matrix, `M` than what was provided in last assignment. I think it is because the Matrix, `M` is derived from the exact same points that were plotted and tested against.

The calibration process did not try to minimize the sum of the squared error. Because it tried to calculate M, that would give close to the ideal extrinsic (location, orientation) and intrinsic parameters (focal length, pixel aspect ratio, principal point, skew). It was trying to minimize the error in these parameters and not the RMS error between actual and projected points. This is in fact a good behavior as having close to precise camera calibration parameters will in turn minimize RMS error.

Below is the code snippet that shows how the Camera Matrix, M is calculated and used to project the points (2D) on the image from actual points (3D) in real world.

```
P=[];
z=repelem(0,4);
for i=1:15
    wi=world_coords(i,:);
    ii=image_coords(i,:);

    iu=ii(1);
    r1=[wi, z, -iu*wi];

    iv=ii(2);
    r2=[z, wi, -iv*wi];

    P=[P; r1; r2];
end

TP=P'*P;
[V,D]=eig(TP);
m=V(:,1);

M=reshape(m,4,3)';
m1=M(1,:)';
m2=M(2,:)';
m3=M(3,:)';
disp('Camera Matrix:');
disp(M);

U=[];
V=[];
for i=1:15
    wi=world_coords(i,:);
    ui=dot(m1,wi)/dot(m3,wi);
    vi=dot(m2,wi)/dot(m3,wi);
    U=horzcat(U,ui);
    V=horzcat(V,vi);
end
```

## PART B

The light was located at (33,29,44). We render a sphere with shading confirm if the shading is appropriate considering the light's position. The sphere with radius 0.5 is located at (3,2,3). We use the equations below to calculate sphere points.

$$x = x_0 + \cos(phi) * \cos(theta) * R$$
$$y = y_0 + \cos(phi) * \sin(theta) * R$$
$$z = z_0 + \sin(phi) * R$$

$$where, \qquad phi = \left\{-\frac{pi}{2}, \frac{pi}{2}\right\} and\ theta = \{0,2 * pi\}$$

Below is the code snippet from calculating the sphere points.

```
[phi,theta]=meshgrid(linspace(-pi/2,pi/2,100),linspace(0,2*pi,100));
radius=0.5;
x0=3;y0=2;z0=3;
x=x0+cos(phi).*cos(theta)*radius;
y=y0+cos(phi).*sin(theta)*radius;
z=z0+sin(phi)*radius;
[nx,ny,nz]=surfnorm(x,y,z);
```

To avoid drawing the invisible points of the sphere, we filter out visible points by using the Camera's position given by (9,14,11) and the surface normal. If the vector from the point to the camera is less than 90 degrees to the surface normal, then the point is visible. Given by,

$$(P - X).N(X) > 0$$
$$where\ P = Camera\ position$$
$$X = (x, y, z)\ point\ on\ sphere$$
$$N(X) = outward\ surface\ normal\ at\ X$$

Below is the code snippet to calculate the visible points on sphere depending on light & camera position.

```
camera_pos=[9,14,11];
points=[x(:),y(:),z(:)];
N=[nx(:),ny(:),nz(:)];
visible_points=[];

for i=1:size(x(:),1)
    a=camera_pos-points(i,:,:);
    if dot(a,N(i,:,:))>0
        visible_points=[visible_points;points(i,:,:)];
    end
end
```

Below is code snippet that maps visible points from 3D real world to the image space (2D) using the Camera Matrix, `M`.

```
U_sphere=[];
V_sphere=[];

for i=1:size(visible_points,1)
    wi=visible_points(i,:);
    ui=dot(m1,wi)/dot(m3,wi);
    vi=dot(m2,wi)/dot(m3,wi);
    U_sphere=horzcat(U_sphere,ui);
    V_sphere=horzcat(V_sphere,vi);
end
```

Below is the code snippet that renders shading on the sphere.

```
light=[33,29,44];
shading=[];
for i=1:size(visible_points,1)
    intensity=dot(N(i,:,:),light);
    if intensity<=0
        point=M*(visible_points(i,:,:))';
        point(1:2)=point(1:2)/point(3);
        shading=[shading;point(1), point(2)];
    end
end
plot(shading(:,1),shading(:,2),'k.');
```

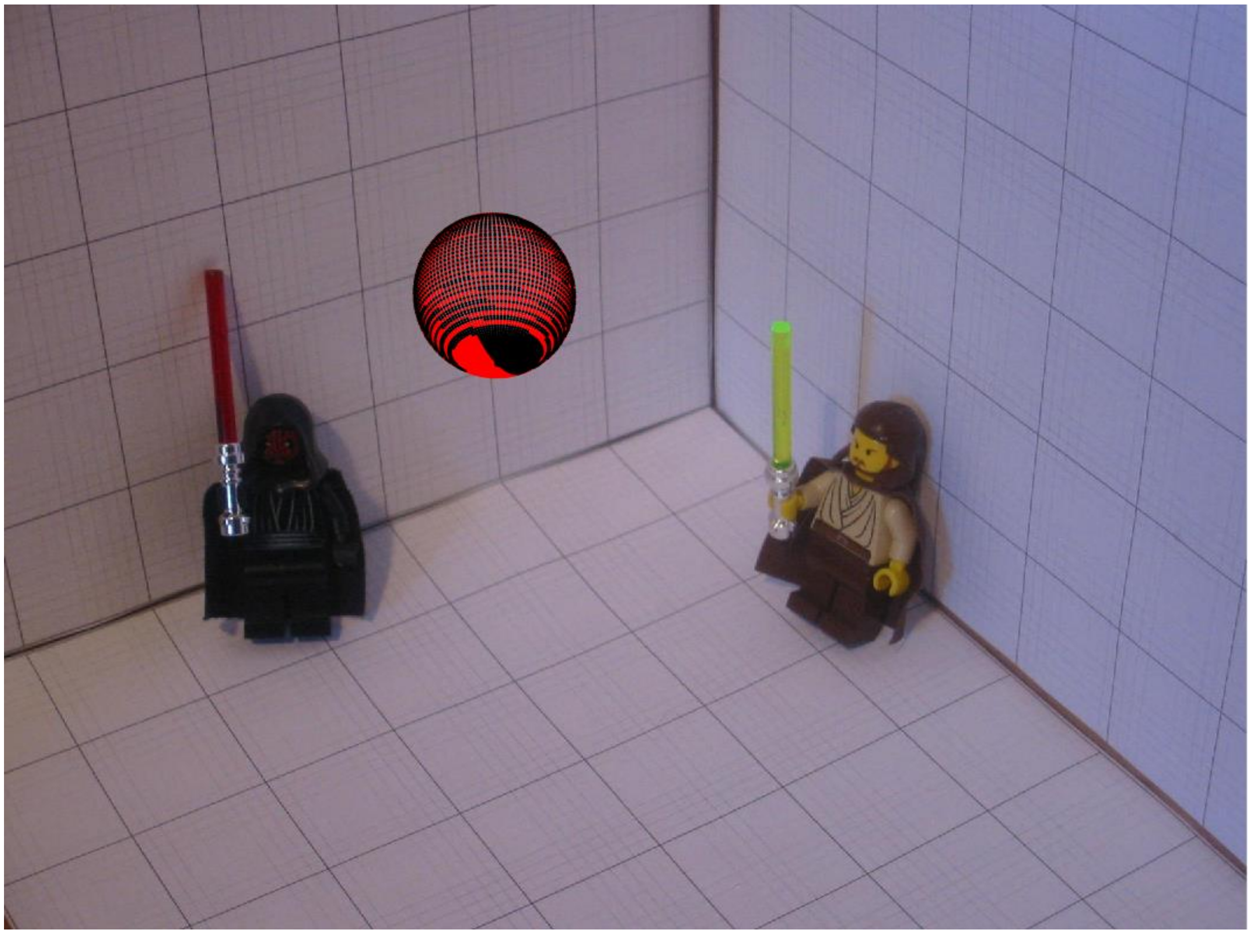
Figure 2. Image showing the sphere with shading

## PART C

As we know, Camera Matrix, `M` is a product of 3 matrices, one that is known, and two others (correspond to intrinsic and extrinsic parameters) that have 11 parameters between them. We will now try to solve for these parameters.

1. Calculate intrinsic parameter matrix, `K`.

K has 5 parameters, namely, focal length (α), pixel aspect ratio (β), principal point (2) and skew. We assume skew is zero because camera has perpendicular axes.
To calculate α and β, we first start by scaling down M by a scaling constant calculated from M. We then calculate the $(u_{0,}v_0)$ required for the coordinate shift using which we find focal length (α) and pixel aspect ratio (β).

```
scaling_const=sqrt(M(3,1)^2+M(3,2)^2+M(3,3)^2);
M=M/scaling_const;
s=sign(M(3,4));

m1=M(1,1:3)';
m2=M(2,1:3)';
m3=M(3,1:3)';

u0=m1'*m3;
v0=m2'*m3;
alpha=sqrt(m1'*m1-u0^2);
beta=sqrt(m2'*m2-v0^2);

K=[alpha,0,u0;0,beta,v0;0,0,1];
```

The value of K was found to be,

```
Intrinsic parameter matrix, K:
   1.0e+03 *
    3.0592         0    1.4881
         0    3.2090    0.7016
         0         0    0.0010
```

Where,
α = 1.0e+03 *3.0592, β = 3.2090, $(u_{0,}v_0) = (1.4881, 0.7016)$ and skew = 0.0010.

2. Calculate extrinsic parameter matrix, X.
   X has 6 parameters, namely, location (3) and orientation (3).
   We first setup 3 orthonormal rotation vectors and a translation vector which when horizontal concatenation yields X.
   Let us look at the code snippet to understand how exactly it happens.

```
R=zeros(3,3);
R(3,:)=s*M(3,1:3);
R(1,:)=s*(u0*M(3,1:3)-M(1,1:3))/alpha;
R(2,:)=s*(v0*M(3,1:3)-M(2,1:3))/beta;

T(1)=s*(u0*M(3,4)-M(1,4))/alpha;
T(2)=s*(v0*M(3,4)-M(2,4))/beta;
T(3)=s*M(3,4);
T=T';

[U,D,V] = svd(R);
R=U*V';
X=[R,T];
X=[X;0,0,0,1];
disp('Extrinsic parameter matrix, X:');
disp(X);
```

The matrix, X is as follows.

```
Extrinsic parameter matrix, X:
    0.7365   -0.6649   -0.1244    3.7780
   -0.3245   -0.5087    0.7974    1.1356
   -0.5935   -0.5470   -0.5905   20.0040
         0         0         0    1.0000
```

3. We now try to re-estimate Camera Matrix, M_est by multiplying K, projection, X.

```
projection=[1,0,0,0;0,1,0,0;0,0,1,0];
M_est=K*projection*X;
disp('Estimated Camera Matrix, M_est:');
disp(M_est);
```

The estimated M turns out to be,

```
Estimated Camera Matrix, M_est:
    0.1370   -0.2848   -0.1259    4.1326
   -0.1458   -0.2016    0.2145    1.7680
   -0.0001   -0.0001   -0.0001    0.0020
```

4. The $M_{est}$ calculated in last step should now be equal to M found in part B. We now calculate the camera position and orientation using the orthonormal rotation vectors and translation vectors as shown below.

```
Camera_pos=-R'*T;
Orientation=R'*[0;0;1];
```

We will now list down all the 11 parameters we have calculated,

a. Focal length (α) = 1.0e+03 *3.0592
b. pixel aspect ratio (β) = 3.2090
c. principal point $(u_{0,}v_0) = (1.4881, 0.7016)$
d. skew = 0.0010.
e. Camera position = (9.4578,14.0308,11.3759)
f. Orientation = (-0.5935,-0.5470,-0.5905)