Xiangyi Cheng (xxc283)

Concept and Background

We already built a 3D virtual world and set two cameras in different locations in Exercise 1. In Exercise 2, we are required to generate the image pairs based on these two cameras.

To obtain the image pair, there are three critical matrices that will be calculated in the latter work, called extrinsic matrix, intrinsic matrix and camera matrix. Extrinsic matrix describes the Euclidean transformation from the world coordinates to the camera coordinates. Intrinsic matrix which also called calibration matrix shows the projection from the camera coordinates to the pixel space. With the extrinsic matrix and intrinsic matrix, one 3D object can be represented into the pixel space which means the location of the 3D objects is converted into the 2D image. To have this conversion, camera matrix is applied which is the dot product of the extrinsic matrix and the intrinsic matrix to the 3D location vector.

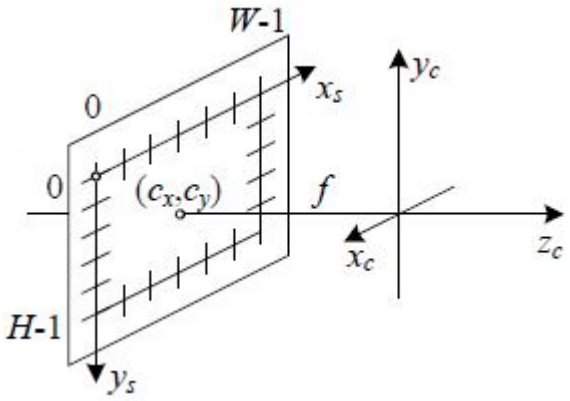
The extrinsic matrix usually is expressed as:

$$M = \begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ Origin * R \end{pmatrix}$$

where X_{cam}, Y_{cam} and Z_{cam} are the camera coordinate frame and Origin is the position vector of the 3D object, R is the rotation matrix as shown below.

$$R = \begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{pmatrix}$$

The intrinsic matrix accomplishes the transformation from the camera coordinate frame into the image pixel space. The image below is showing the simplified camera intrinsics where f is the focal length and (c_x, c_y) is the optical center. The image width and height are W and H.



The mathmetical way to represent the intrinsic matrix K is:

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & af_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

where s encodes any possible skew between the sensor axes and a is the aspect ratio. In practice, the matrix can be simplified by setting a=1 and s=0,

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

The camera matrix describes the transformation between 3D objects into the images. Based on the previous matrices, we can tell that camera matrix P is the multiplication of extrinsic matrix and intrinsic matrix.

$$P = M * K$$

The application below illustrates how to get the image pair using these matrices in python.

Implement and Exploration

Import Exercise 1 into the code so that we can use it explicitly.

In [31]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection


def define_3Dpoints():
    x=[-0.5, 0, 0.5]
    y=[-0.5, 0, 0.5]
    z=[-0.5, 0, 0.5]
    X, Z, Y = np.meshgrid(x, y, z)

    dimension=X.shape[0]*Y.shape[0]*Z.shape[0]

    colors=np.zeros((dimension,3))
    for i in range (0, dimension):
        colors[i,:]=np.random.rand(3)

    return X,Y,Z,colors


def camera_specification():
    # position parameters
    r= 5
    alpha1= np.pi/6
    beta= np.pi/6

    # camera 1 parameters
    cam1_pos= [r*np.cos(beta)*np.cos(alpha1),
               r*np.cos(beta)*np.sin(alpha1), r*np.sin(beta)]
    target= np.array([0,0,0])
    up= np.array([0,0,1])
    focal_length= 0.06
    film_width= 0.035
    film_height= 0.035
    width= 256
    height= 256

    # camera 2 parameters, others are the same as the camera 1
    alpha2= np.pi/3
    cam2_pos= [r*np.cos(beta)*np.cos(alpha2),
               r*np.cos(beta)*np.sin(alpha2), r*np.sin(beta)]
    return cam1_pos,cam2_pos,target,up,focal_length,
film_height,film_width,width,height


def camera_view_unit(target,cam_pos,up):
    zcam= target-cam_pos
    xcam= np.cross(zcam,up)
    ycam= np.cross(zcam,xcam)

    # normalization
    if np.linalg.norm(xcam)!=0:
        xcam= xcam/np.linalg.norm(xcam)

    if np.linalg.norm(ycam)!=0:
```

```
        ycam= ycam/np.linalg.norm(ycam)

    if np.linalg.norm(zcam)!=0:
        zcam= zcam/np.linalg.norm(zcam)

    return xcam,ycam,zcam
```

Define the extrinsic matrix by knowing X_{cam}, Y_{cam}, Z_{cam} which are obtained from Exercise 1. What we need to do is to reshape the matrix then add another row, $\left(Origin * R \right)$. Origin is the camera position which was defined before and R is the rotation matrix as shown below.

$$R = \begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{pmatrix}$$

In [32]:
```
def extrinsic_matrix(camera):
    cam1_pos,cam2_pos,target,up,_,_,_,_,_=camera_specification()

    if camera==1:
        cam_pos=cam1_pos
    elif camera==2:
        cam_pos=cam2_pos
    else:
        print 'camera is not defined.'

    xcam,ycam,zcam=camera_view_unit(target,cam_pos,up)

    rotation_matrix=np.column_stack((xcam,ycam,zcam))
    add=np.dot(np.dot(-1,cam_pos),rotation_matrix)

    extrinsic_matrix=np.vstack([rotation_matrix,add])

    return extrinsic_matrix
```

Next step is intrinsic matrix specification. As discussed before, the intrinsic matrix K is

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

where (c_x, c_y) are the optical center and f_x, f_y are the independent focal lengths which can be compulated by similar triangle rule.

In [33]:
```
def intrinsic_matrix(camera):
    _,_,_,_,focal_length,film_height,film_width,
    width,height=camera_specification()
    cx= 0.5* (width +1)
    cy= 0.5* (height+1)

    fx= focal_length* width /film_width
    fy= focal_length* height/film_height

    # K matrix
    intrinsic_matrix= [[fx,0,0],[0,fy,0],[cx,cy,1]]

    return intrinsic_matrix
```

With extrinsic matrix and intrinsic matrix, we are able to compute the camera matrix by simply multiplying them two.

In [34]:
```
def camera_matrix(camera,extrinsic_matrix,intrinsic_matrix):
    extrinsic_matrix=extrinsic_matrix(camera)
    intrinsic_matrix=intrinsic_matrix(camera)
    camera_matrix= np.dot(extrinsic_matrix, intrinsic_matrix)

    # P matrix
    return camera_matrix
```

Multiply the 3D object position to the camera matrix to get the 2D coordinates of the object in the image space. All the x coordinate values and y coordinate values are returned into two seperate vectors. As well, the colors of the points are also returned.

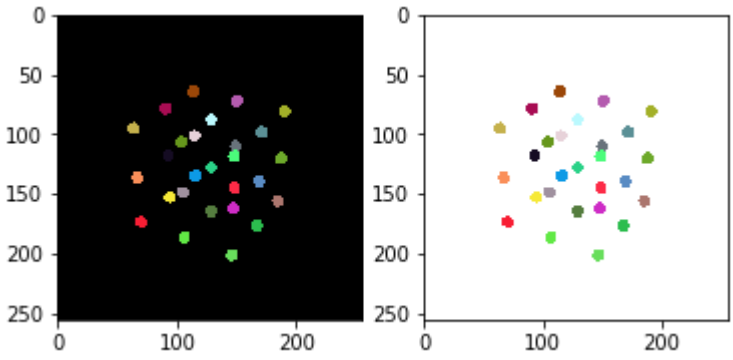
In [35]:
```
def conv_2Dimage(camera,camera_matrix):
    X,Y,Z,colors=define_3Dpoints()
    X_reshape=X.reshape(1,-1)
    Y_reshape=Y.reshape(1,-1)
    Z_reshape=Z.reshape(1,-1)
    _,dimension=X_reshape.shape

    point=np.column_stack((X_reshape,Y_reshape,
                           Z_reshape,np.ones((1,dimension))))
    point_reshape=np.transpose(point.reshape(4,-1))
    pt=np.dot(point_reshape,camera_matrix)
    object_x=np.transpose(pt[:,0]/pt[:,2])
    object_y=np.transpose(pt[:,1]/pt[:,2])

    return object_x,object_y,colors
```

The final step is to plot the images taken by the two cameras. Instead of plotting the single pixel, expanding the pixel into a visible dot is a wise idea. Firstly, create a matrix representing the background color. I used two colors, black and white, to show the points. If the region is in the circle pattern near the located pixel, fill the color in the region, otherwise keep the background color.

In [36]:
```
def plot_2Dimage(object_x,object_y,colors):
    _,_,_,_,_,_,_,width,height=camera_specification()

    color_matrix_white= np.ones((height,width,3)) # white background
    color_matrix_black= np.zeros((height,width,3)) # black background

    object_number=object_x.shape[0]
    show_region=5

    for i in range (0,object_number):
        r1= int(max(0,np.floor(object_y[i]-show_region)))
        r2= int(min(height-1,np.ceil(object_y[i]+show_region)))
        c1= int(max(0,np.floor(object_x[i]-show_region)))
        c2= int(min(width-1,np.ceil(object_x[i]+show_region)))

        for r in range (r1,r2+1):
            for c in range (c1,c2+1):
                if (r-object_y[i])**2+(c-object_x[i])**2< show_region**2
                    color_matrix_black[r,c,:]=colors[i,:]
                    color_matrix_white[r,c,:]=colors[i,:]

    plt.subplot(1,2,1)
    plt.imshow(color_matrix_black)
    plt.subplot(1,2,2)
    plt.imshow(color_matrix_white)
    plt.show()
```

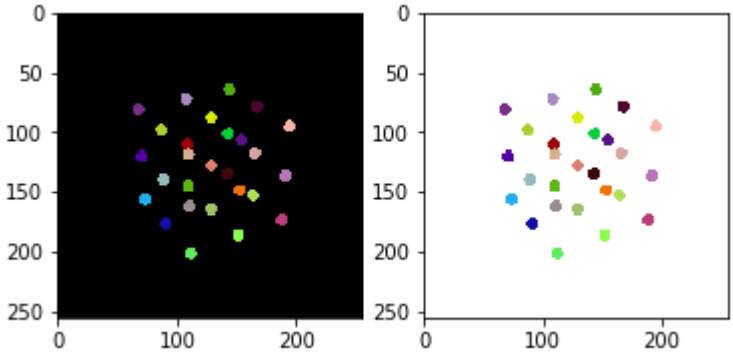
The images taken by the camera 1 in different background colors are shown below.

In [29]:
```
camera=1
camera_matrix=camera_matrix(camera,extrinsic_matrix,intrinsic_matrix)
object_x,object_y,colors=conv_2Dimage(camera,camera_matrix)
plot_2Dimage(object_x,object_y,colors)
```



The images taken by the camera 2 in different background colors are shown below.

In [37]:
```
camera=2
camera_matrix=camera_matrix(camera,extrinsic_matrix,intrinsic_matrix)
object_x,object_y,colors=conv_2Dimage(camera,camera_matrix)
plot_2Dimage(object_x,object_y,colors)
```



Conclusion and Discussion

The image pair is obtained based on the transformation from 3D object into 2D image by applying extrinsic matrix and intrinsic matrix. In the application, instead of converting 3D into 2D, we usually reconstruct the 3D object using several 2D images. This exercise can be a preparation for the reconstruction.

The exploration of my work is that I used python instead so that some representations were developed by myself. Also because of this, there is a limit of my code which will be discussed below.

One thing can be improved is that I have to run seperately to show the images taken by camera 1 and 2. And the color of each point is given randomly. Therefore, the colors of points are slightly different from image 1 and 2. I believe it would be better if I modify the approach to plot.

Reference

Richard Szeliski, "Computer Vision Algorithms and Applications", 2011