Computer Vision - coursework 1

1. Homogeneous Coordinates

Given a point x with coordinates $x = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix}^T$, compute and apply the following transformations in homogeneous representation:

(i) translation t with translation vector $t = \begin{bmatrix} 1 & 0 & -2 \end{bmatrix}^T$
(ii) rotation $R_z(\varphi)$ with $\varphi = 20°$
(iii) the rigid body transformation resulting from (i) and (ii)
(iv) the transformation given by H_1 followed by H_2 with:

$$H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & -1 \\ 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad H_2 = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Intrinsic and extrinsic parameters, Direct Linear Transform (DLT)

For this exercise you will use a 3D cube with a checkerboard pattern to get the correspondences between the 3D points on the cube and 2D points in the image (see folder checkerboard_cube). Use the python script provided to get you started. The checkerboard pattern makes it possible to compute the 3D points simply by computing the number of boxes along the axes. The origin of the coordinates is at the nearest corner of the cube with respect to the current view as depicted below. The axes are aligned along the edges of the cube with X-axis, Y-axis and Z-axis shown in red, blue and green respectively.

Load the first image of cube `cube0.jpg` from the data folder `checkerboard_cube`. Choose a set of 3D points X on the cube (control points) and find the corresponding points in the image by clicking.

Hint:

- To avoid clicking the points again with every test run, you can save them using numpy `save` and load them later.

```
# load the input image (cube0.jpg)
# Define 3D control points (Minimum of 6 Points)
X = [[0, 0, 0],
     [7, 0, 0],
     [7, 0, 7],
     [0, 0, 7],
     [7, 9, 7],
     [0, 9, 7],
     [0, 9, 0]]
X = np.array(X)
print("3D points: \n", X)

print(" \n Please click the corresponding points in the image following the order of g
# Get observed points from user (use cv2.setMouseCallback() for this)
x = None
print("\n corresponding image coordinates: \n", x)
```

Compute projection matrix using DLT

Implement the function `dlt` which estimates the projection matrix P from the 3D-2D correspodences generated in the previous step. Print the estimated projection matrix P.

```
# perform dlt
P = ex.dlt(x, X)
print("\n The estimated projection matrix: \n", P)
```

Determination of the calibration parameters

Determine the intrinsic matrix K, and the extrinsics, i.e. the rotation matrix R and the projection center X_0 from the estimated projection matrix P. Implement this in the function `decompose_P` and print the results.

```
# decompose projection matrix to get instrinsics and extrinsics
[K, R, X0] = ex.decompose_P(P)

print('Intrinsic matrix: ', K)
print('Extrinsic matrix: ', "\n R: \n", R, "\n X0: \n", X0)
```

Assessment of the reprojection error

Check the estimated projection matrix using the back projection errors. This is obtained by projecting the object points back into the image and comparing them with the given

image points.

Hints

- You should create the projection matrix again from the extracted calibration parameters and project the object points onto the image.
- Show the given and the back-projected points together in the image.
- Determine the distances between given and back-projected pixels.

```
# generate homogeneous 3D points
...

# Compute reprojection error
x_hat = None
print("\n reprojected image coordinates: \n", x_hat)

err_reproj = None
print("\n The reprojection error: ", err_reproj)

# Visualize observed and reprojected points
```

3. Filters and convolutions

Write a program that smooths an image taken by you and then detects and enhances the edges in the image. Only basic opencv commands are to be used. (no cannyEdge, findContours, Sobel, etc.) I specifically want you to implement your own version of convolutions over an image with a kernel.

Feel free to experiment with filters other than those seen in class (Gabor, Scharr, Prewitt, etc) but write your own code for those filters.

Every function in your code should be documented with an explanation in your own words of how it works and what it does.

Input Arguments:

- **Grey-scale Image Filename** Read the input image of arbitrary format into a 1-channel matrix using imread.
- **Filter Width** The width of the noise-filtering filter kernel.
- **Filter Type** The type of noise filtering: Gaussian smoothing, median filtering and mean filter.
- **Standard Deviation** If Gaussian smoothing is chosen, use this additional argument for the standard deviation used while creating the Gaussian filter. How is it related to the filter width?
- **Hysteresis Thresholds** Optionally take two arguments for the hysteresis thresholds t_1 and t_h of the edge detector.

If no input arguments are given or they don't match the expectation, your program should print a brief documentation of how to use it.

OpenCV

Write a program that performs the same tasks as your program, but now using OpenCV methods. Compare both programs in terms of time and results.