# HW2

January 24, 2018

## CSE 252B: Computer Vision II, Winter 2018 – Assignment 2

**Instructor: Ben Ochoa**

**Due: Wednesday, February 7, 2018, 11:59 PM**

### Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains both math and programming problems.
- All solutions must be written in this notebook
- Math problems must be done in Markdown/LATEX. Remember to show work and describe your solution.
- Programming aspects of this assignment must be completed using Python in this notebook.
- This notebook contains skeleton code, which should not be modified (This is important for standardization to facilate effeciant grading).
- You may use python packages for basic linear algebra, but you may not use packages that directly solve the problem. Ask the instructor if in doubt.
- You must submit this notebook exported as a pdf. You must also submit this notebook as an .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- It is highly recommended that you begin working on this assignment early.

### Problem 1 (Programing): Linear estimation of the camera projection matrix (15 points)

Download input data from the course website. The file hw2_points3D.txt contains the coordinates of 50 scene points in 3D (each line of the file gives the $\tilde{X}_i$, $\tilde{Y}_i$, and $\tilde{Z}_i$ inhomogeneous coordinates of a point). The file hw2_points2D.txt contains the coordinates of the 50 corresponding image points in 2D (each line of the file gives the $\tilde{x}_i$ and $\tilde{y}_i$ inhomogeneous coordinates of a point). The scene points have been randomly generated and projected to image points under a camera projection matrix (i.e., $\boldsymbol{x}_i = \boldsymbol{P}\boldsymbol{X}_i$), then noise has been added to the image point coordinates.

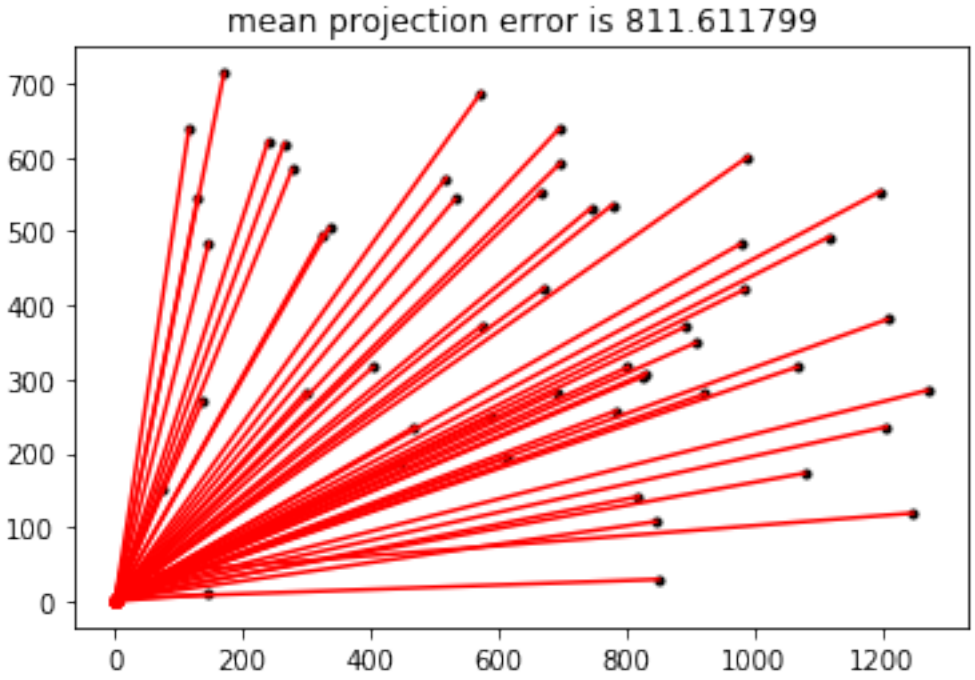
Estimate the camera projection matrix $\boldsymbol{P}_{\text{DLT}}$ using the direct linear transformation (DLT) algorithm (with data normalization). You must express $\boldsymbol{x}_i = \boldsymbol{P}\boldsymbol{X}_i$ as $[\boldsymbol{x}_i]^{\perp}\boldsymbol{P}\boldsymbol{X}_i = \boldsymbol{0}$ (not $\boldsymbol{x}_i \times \boldsymbol{P}\boldsymbol{X}_i = \boldsymbol{0}$), where $[\boldsymbol{x}_i]^{\perp}\boldsymbol{x}_i = \boldsymbol{0}$, when forming the solution. Return $\boldsymbol{P}_{\text{DLT}}$, scaled such that $||\boldsymbol{P}_{\text{DLT}}||_{\text{Fro}} = 1$

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

```
x=np.loadtxt('hw2_points2D.txt').T
X=np.loadtxt('hw2_points3D.txt').T
print('x is', x.shape)
print('X is', X.shape)

def toHomo(x):
    # converts points from inhomogeneous to homogeneous coordinates
    return np.vstack((x,np.ones((1,x.shape[1]))))
def fromHomo(x):
    # converts points from homogeneous to inhomogeneous coordinates
    return x[:-1,:]/x[-1,:]

def computeP_DLT(x,X):
    # inputs:
    # x 2D points
    # X 3D points
    # output:
    # P_DLT the (3x4) DLT estimate of the camera projection matrix

    """your code here"""
    P = np.eye(3,4)+np.random.randn(3,4)/10
    return P/np.sqrt(np.sum(P**2))

def proj(P,X):
    # projects 3d points X to 2d using projection matrix P
    return fromHomo(np.matmul(P,toHomo(X)))

def rmse(x,y):
    # calculates the root mean square error (RMSE)
    # used to measure reprojection error
    return np.mean(np.sqrt(np.sum((x-y)**2,0)))

def displayResults(P, x, X, title):
    print (title+' =')
    print (P)
    print ('||%s||=%f'%(title, np.sqrt(np.sum(P**2)) ))

    x_proj = proj(P,X)
    plt.plot(x[0,:], x[1,:],'.k')
    plt.plot(x_proj[0,:], x_proj[1,:],'.r')
    for i in range(x.shape[1]):
        plt.plot([x[0,i], x_proj[0,i]], [x[1,i], x_proj[1,i]], '-r')

    plt.title('mean projection error is %f'%rmse(x,x_proj))
    plt.show()

P_DLT = computeP_DLT(x,X)
```

```
        displayResults(P_DLT, x, X, 'P_DLT')
```

```
x is (2, 50)
X is (3, 50)
P_DLT =
[[ 0.59344377 -0.08964572  0.0116238  -0.0919074 ]
 [ 0.03159558  0.53853317  0.02790091 -0.01749567]
 [-0.01372009  0.12111904  0.56553239 -0.06648533]]
||P_DLT||=1.000000
```



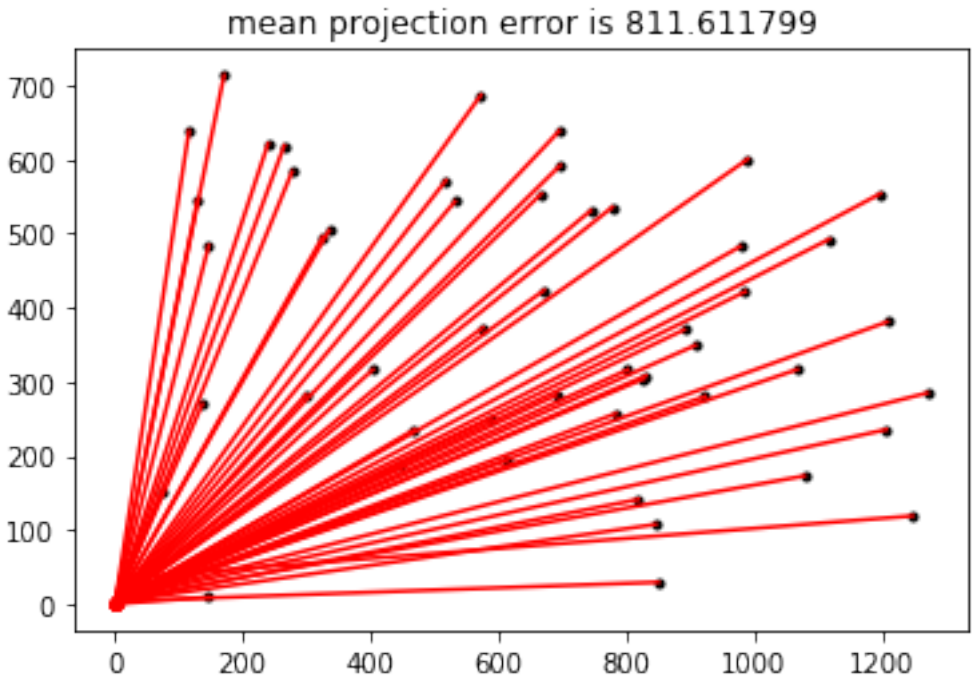

**Problem 2 (Programing): Nonlinear estimation of the camera projection matrix (30 points)**

Use $\boldsymbol{P}_{\text{DLT}}$ as an initial estimate to an iterative estimation method, specifically the Levenberg-Marquardt algorithm, to determine the Maximum Likelihood estimate of the camera projection matrix that minimizes the projection error. You must parameterize the camera projection matrix as a parameterization of the homogeneous vector $\boldsymbol{p} = vec(\boldsymbol{P}^\top)$. It is highly recommended to implement a parameterization of homogeneous vector method where the homogeneous vector is of arbitrary length, as this will be used in following assignments. Return $\boldsymbol{P}_{\text{LM}}$, scaled such that $||\boldsymbol{P}_{\text{LM}}||_{\text{Fro}} = 1$. You may need to change the max itterations or implement another stoping criteria.

```
In [2]: def LMstep(P, x, X, l, v):
            # inputs:
            # P current estimate of P
```

```
            # x 2D points
            # X 3D points
            # l LM lambda parameter
            # v LM change of lambda parameter
            # output:
            # P updated by a single LM step
            # l accepted lambda parameter

            """your code here"""
            return P, l

        # use P_DLT as an initalization for LM
        P_LM = P_DLT.copy()

        # LM hyperparameters
        l=.001
        v=10
        max_iters=10

        # LM optimization loop
        for i in range(max_iters):
            P_LM, l = LMstep(P_LM, x, X, l, v)
            print ('iter %d mean reprojection error %f'%(i+1, rmse(x,proj(P_LM,X))))

        displayResults(P_LM, x, X, 'P_LM')
```

```
iter 1 mean reprojection error 811.611799
iter 2 mean reprojection error 811.611799
iter 3 mean reprojection error 811.611799
iter 4 mean reprojection error 811.611799
iter 5 mean reprojection error 811.611799
iter 6 mean reprojection error 811.611799
iter 7 mean reprojection error 811.611799
iter 8 mean reprojection error 811.611799
iter 9 mean reprojection error 811.611799
iter 10 mean reprojection error 811.611799
P_LM =
[[ 0.59344377 -0.08964572  0.0116238  -0.0919074 ]
 [ 0.03159558  0.53853317  0.02790091 -0.01749567]
 [-0.01372009  0.12111904  0.56553239 -0.06648533]]
||P_LM||=1.000000
```


mean projection error is 811.611799
700
600
500
400
300
200
100
0
0
200
400
600
800
1000
1200