HW2

February 6, 2018

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

1.0.1 Instructor: Ben Ochoa

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

1.1 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.

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

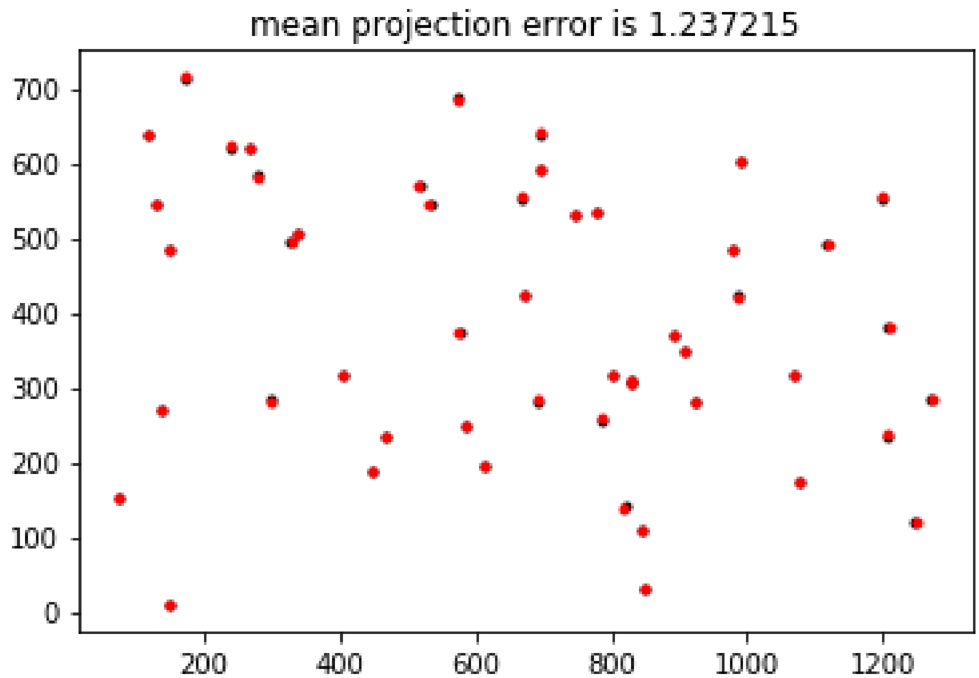
Download input data from the course website. The file hw2_points3D.txt contains the coordinates of 50 scene points in 3D (each line of the file gives the $\tilde{X}_i$, $\tilde{Y}_i$, and $\tilde{Z}_i$ inhomogeneous coordinates of a point). The file hw2_points2D.txt contains the coordinates of the 50 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 [11]: 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"""
             #normalization
             x_homo=toHomo(x)
             X_homo=toHomo(X)

             x_nl=x_homo
             X_nl=X_homo

             v=np.zeros((3,50))
             A=np.zeros((2*50,12))
             for i in range(50):
                 #calculate v vector
                 v[:,i]=(x_nl[:,i].reshape(3,1)+np.sign(x_nl[0,i])*np.sqrt(np.dot(x_nl[:,i]\
                 ,x_nl[:,i]))*np.array([[1],[0],[0]])).ravel()
                 #calculate H_v matrix
                 H_v=np.eye(3,3)-2*v[:,i].reshape(3,1)*v[:,i].reshape(3,1).T/np.dot(v[:,i],v[:
                 A[2*i:2*i+2,:]=np.kron(H_v[1:3,:],X_nl[:,i].reshape(1,4))
             #svd and find the smallest eigenvalue (which is supposed to be 0)
             u,sigma,vv=np.linalg.svd(A)
             index=np.where(sigma==np.min(sigma))
             Pcol=vv[index[0][0],:]
             P=Pcol.reshape(3,4)

             norm=np.sqrt(np.sum(P*P))

             return P/norm
```

```
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 =
[[-6.46262236e-03  2.83275252e-03 -8.10457168e-03 -8.58034821e-01]
 [-8.17628011e-03  1.46133403e-03  4.93686655e-03 -5.13386709e-01]
 [-5.75770103e-06 -5.19232400e-06 -2.92889885e-06 -1.22444378e-03]]
||P_DLT||=1.000000
```



1.3 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 iterations or implement another stoping criteria.

```
In [8]: def sinc(x):
            if x==0:
                return 1
            else:
                return np.sin(x)/x
        def frompavec(v):
            norm=np.sqrt(np.dot(v.T,v))
            a=np.cos(norm/2)
            b=sinc(norm/2)/2*v
            return np.vstack((a,b))
        def topavec(P):
            a=P[0]
            b=P[1:]
            v=2/sinc(np.arccos(a))*b
```

```
    norm=np.sqrt(np.dot(v.T,v))
    if norm>np.pi:
        v=(1-2*np.pi/norm*np.ceil((norm-np.pi)/(2*np.pi)))*v
    return v
def Jacobian(X_nl,v):
    #inputs
    #X 3D inhomogeneous coordinates
    #v parameterizated P

    #deparameterization
    P0=frompavec(v)
    p3=P0[8:12]
    norm=np.sqrt(np.dot(v.T,v))
    a=P0[0]
    b=P0[1:]
    #calculate projection
    x=proj(P0.reshape(3,4),fromHomo(X_nl))
    #calculate partialpv
    if norm==0:
        partialav=np.zeros((11,1))
        partialbv=1/2*np.eye(11,11)
    else:
        partialav=-1/2*b.T
        partialbv=sinc(norm/2)/2*np.eye(11,11)+1/4/norm*(np.cos(norm/2)/norm*2\
        -np.sin(norm/2)/norm/norm*4)*np.dot(v,v.T)
    partialpv=np.vstack((partialav,partialbv))
    #calculate partialxp
    partialxp=np.zeros((100,12))
    for i in range(50):
        XT=X_nl[:,i].reshape(1,4)
        w=XT.dot(p3)
        Ai=np.zeros((2,12))
        Ai[0,:4]=XT
        Ai[1,4:8]=XT
        Ai[0,8:12]=-x[0,i]*XT
        Ai[1,8:12]=-x[1,i]*XT
        partialxp[2*i:2*i+2,:]=1/w*Ai
    J=partialxp.dot(partialpv)
    return J

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"""
#normalization

x_homo=toHomo(x)
X_homo=toHomo(X)
x0mean=np.mean(x_homo[0,:])
x1mean=np.mean(x_homo[1,:])
x0var=np.var(x[0,:])
x1var=np.var(x[1,:])
sigma=np.sqrt(x0var**2+x1var**2)
s2d=np.sqrt(2/(sigma**2))
H2d=np.array([[s2d,0,-s2d*x0mean],[0,s2d,-s2d*x1mean],[0,0,1]])

X0mean=np.mean(X_homo[0,:])
X1mean=np.mean(X_homo[1,:])
X2mean=np.mean(X_homo[2,:])
X0var=np.var(X[0,:])
X1var=np.var(X[1,:])
X2var=np.var(X[2,:])
sigma=np.sqrt(X0var**2+X1var**2+X2var**2)
s3d=np.sqrt(2/(sigma**2))
H3d=np.array([[s3d,0,0,-s3d*x0mean],[0,s3d,0,-s3d*x1mean],[0,0,s3d,-s3d*x1mean],[0

x_nl=H2d.dot(x_homo)
X_nl=H3d.dot(X_homo)

P_nl=H2d.dot(P).dot(np.linalg.inv(H3d))
P_nl=P_nl/np.sqrt(np.sum(P_nl*P_nl))

#calculate Jacobian Matrix
J=Jacobian(X_nl,topavec(P_nl.reshape(12,1)))

#calculate current error
xproj=proj(P_nl,fromHomo(X_nl))
epsilon=(fromHomo(x_nl)-xproj).reshape((100,1),order='F')
Sigmax=s2d**2*np.eye(100,100)
error=epsilon.T.dot(np.linalg.inv(Sigmax)).dot(epsilon)
flag=True
while flag:
    delta=np.linalg.inv(J.T.dot(np.linalg.inv(Sigmax)).dot(J)+l*np.eye(11,11))\
    .dot(J.T.dot(np.linalg.inv(Sigmax)).dot(epsilon))
    P_candidate=topavec(P_nl.reshape(12,1))+delta
    xproj1=proj(frompavec(P_candidate).reshape(3,4),fromHomo(X_nl))
    epsilon1=(fromHomo(x_nl)-xproj1).reshape((100,1),order='F')
    error_1=epsilon1.T.dot(np.linalg.inv(Sigmax)).dot(epsilon1)
```

```
                if error_1<error:
                    print('error = '+ str(error_1[0][0]))
                    flag=False
                    l=0.1*l
                else:
                    l=10*l
            P=frompavec(P_candidate).reshape(3,4)
            P=np.linalg.inv(H2d).dot(P).dot(H3d)
            norm=np.sqrt(np.sum(P*P))
            return P/norm, 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')
```

```
error = 96.73772007616172
iter 1 mean reprojection error 1.227809
error = 91.4591392821938
iter 2 mean reprojection error 1.180109
error = 89.95878219801654
iter 3 mean reprojection error 1.165503
error = 89.24503483386754
iter 4 mean reprojection error 1.158954
error = 88.77968855752593
iter 5 mean reprojection error 1.155246
error = 88.42450058676607
iter 6 mean reprojection error 1.152843
error = 88.13044759412563
iter 7 mean reprojection error 1.151152
error = 87.87548695962558
iter 8 mean reprojection error 1.149896
error = 87.64798405941926
iter 9 mean reprojection error 1.148924
error = 87.44111319371865
iter 10 mean reprojection error 1.148151
P_LM =
[[ 6.34516774e-03 -3.31830422e-03  8.25560374e-03  8.51267863e-01]
```

```
 [ 8.43140474e-03 -1.61868453e-03 -5.19754857e-03  5.24520088e-01]
 [ 5.73872948e-06  5.13557252e-06  2.91076731e-06  1.24570450e-03]]
||P_LM||=1.000000
```