CSc 8830: Computer Vision

Homework Assignment 1

PART A: Theory

1. Point the camera to a chessboard pattern or any known set of reference points that lie on the same plane. Capture a series of 10 images by changing the orientation of the camera in each iteration. Select any 1 image, and using the image formation pipeline equation, set up the linear equations in matrix form and solve for intrinsic and extrinsic parameters (extrinsic for that particular orientation). You will need to make measurements of the actual 3D world points, and mark pixel coordinates.

Perspective Projection:

Points:

```
#Calcluation of perspective projection
# considered real world obejct points (20,20),(50,50),(70,70),(90,90),(100,100),(120,120)
# pointed values (833.5,273.5),(861.5,339.5),(881.5,383.5),(905.5,427.5),(913.5,451.5),(933.5,501.5)
```

Matrix:

```
A = [[20, 20, 0, 1, 0, 0, 0, 0,-833.5*20,-833.5*20,0,-833.5],
     [0, 0, 0, 0, 20, 20, 0, 1,-271.5*20,-271.5*20,0,-271.5],
     [50, 50, 0, 1, 0, 0, 0, 0,-861.5*50,-861.5*50,0,-861.5],
     [0, 0, 0, 0, 50, 50, 0, 1,-339.5*50,-339.5*50,0,-339.5],
     [70, 70, 0, 1, 0, 0, 0, 0,-881.5*70,-881.5*70,0,-881.5],
     [0, 0, 0, 0, 70, 70, 0, 1,-385.5*70,-385.5*70,0,-385.5],
      [90, 90, 0, 1, 0, 0, 0, 0,-903.5*90,-903.5*90,0,-903.5],
      [0, 0, 0, 0, 90, 90, 0, 1,-429.5*90,-429.5*90,0,-429.5],
      [100, 100, 0, 1, 0, 0, 0, 0,-913.5*100,-913.5*100,0,-913.5],
      [0, 0, 0, 0, 100, 100, 0, 1,-451.5*100,-451.5*100,0,-451.5],
      [120, 120, 0, 1, 0, 0, 0, 0,-931.5*120,-931.5*120,0,-931.5],
      [0, 0, 0, 0, 120, 120, 0, 1,-503.5*120,-503.5*120,0,-503.5]]
```

Transpose matrix:

```
# 12*12 matrix
Y = [[20,  0,  50,  0,  70,  0,  90,  0,  100,  0,  120,  0],
     [0,  0,  50,  0,  70,  0,  90,  0,  100,  0,  120,  0  ],
     [50, 50,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
     [0,  0,  0,  0,  1,  0,  1,  0,  1,  0,  1,  0],
     [70,  70,  0,  1,  0,  70,  0,  90,  0,  100,  0,  120],
     [0,  0,  0,  0,  70,  70,  0,  90,  0,  100,  0,  120],
     [90,  90,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0],
     [0,  0,  0,  0,  90,  90,  0,  1,  0,  1,  0,  1],
     [100,  100,  0,  1,  0,  0,  0,  0,  -91350.0,  -45150.0, -111780.0,  -60420.0],
     [0,  0,  0,  0,  100,  100,  0,  1,  -45150.0,  -45150.0,  -111780.0,  -60420.0],
     [120,  120,  0,  1,  0,  0,  0,  0,  -111780.0,  -111780.0,  0,  0],
    [0,  0,  0,  0,  120,  120,  0,  1,  -60420.0,  -60420.0,  0,  -503.5]]
```

Multiple of matrix and its transpose:

```
[-1666600.0, -1667000.0, 2000.0, -16670.0, -1764219.0, -1767020.0, 3601.0, -17503.5, 2325819071.0, 1555661070.0, 3726750001.0, 2014822467.25]
[-541600.0, -541600.0, 0.0, -5410.0, -574090.0, -572690.0, 0.0, -2100.5, 757599030.0, 506737031.0, 1213930800.0, 656302701.25]
[-4306500.0, -4307500.0, 5000.0, -43075.0, -4403879.0, -4410880.0, 9001.0, -43936.5, 5931799331.0, 3941724330.0, 9629859001.0, 5205616765.25]
[-1694000.0, -1694000.0, 0.0, -16925.0, -1734650.0, -1731150.0, 0.0, -8313.5, 2337600090.0, 1553365091.0, 3794931000.0, 2051441939.25]
[-6169100.0, -6170500.0, 7000.0, -61705.0, -6266479.0, -6276280.0, 12601.0, -62586.5, 8476006731.0, 5625221730.0, 13794786601.0, 7456876035.25]
[-2693600.0, -2693600.0, 0.0, -26915.0, -2739770.0, -2734870.0, 0.0, -14769.5, 3706744410.0, 2460051411.0, 6032766600.0, 3261078300.25]
[-8129700.0, -8131500.0, 9000.0, -81315.0, -8227319.0, -8239920.0, 16201.0, -82218.5, 11154104971.0, 7397333970.0, 18178803001.0, 9826559512.25]
[-3859200.0, -3859200.0, 0.0, -38565.0, -3910650.0, -3904350.0, 0.0, -22883.5, 5302357890.0, 3516514891.0, 8641711800.0, 4671308054.25]
[-9133000.0, -9135000.0, 10000.0, -91350.0, -9230619.0, -9244620.0, 18001.0, -92263.5, 12524488671.0, 8304098670.0, 20422230001.0, 11039193947.25]
[-4508000.0, -4508000.0, 0.0, -45050.0, -4562090.0, -4555090.0, 0.0, -27600.5, 6190254630.0, 4104344631.0, 10093734000.0, 5456177331.25]
[-11175600.0, -11178000.0, 12000.0, -111780.0, -11272979.0, -11289780.0, 21601.0, -112711.5, 15314275231.0, 10150015230.0, 24989565601.0, 13507964210.25]
[-6033600.0, -6033600.0, 0.0, -60300.0, -6093930.0, -6085530.0, 0.0, -39322.5, 8277751470.0, 5486371471.0, 13507495200.0, 7301435113.25]
```

Its QR factor:

```
# QR Factor
import numpy as np


# Original matrix
matrix1 = np.array([[-1767020.0, -574090.0, -441088.0], [-6276280.0, -2734870.0, -8239920.0],[-9244620.0, -4562090.0, -11289780.0]])
print(matrix1)

# Decomposition of the said matrix
q, r = np.linalg.qr(matrix1)
print('\nQ:\n', q)
print('\nR:\n', r)
```

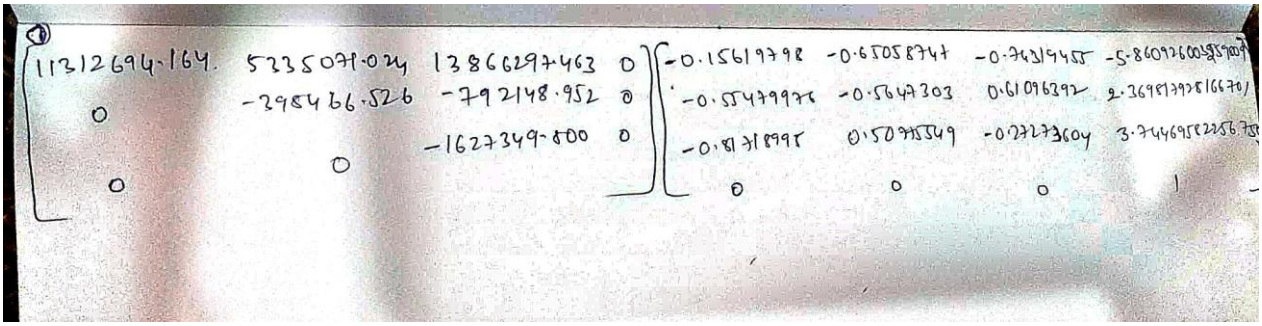
```
[[ -1767020.   -574090.   -441088.]
 [ -6276280.  -2734870.  -8239920.]
 [ -9244620.  -4562090. -11289780.]]

Q:
 [[-0.15619798 -0.65058747 -0.74319455]
 [-0.55479976 -0.5647303   0.61096392]
 [-0.81718995  0.50775549 -0.27273604]]

R:
 [[11312694.16466299  5335071.02487801 13866297.46333579]
 [       0.          -398466.52633207  -792148.95274484]
 [       0.                0.         -1627349.80059501]]
```

Translation vector:

```
[-5.860926003259809]
[2.369817928166701]
[3.74469582256755]
```

Once you compute the Rotation matrix, you also need to compute the angles of rotation along each axis. Choose your order of rotation based on your experimentation setup.

Complete matrix:



2. Select any pair of images from the set in problem 1 above. Compute the homography between those two images.

2. Calculation of homography

```
#Calcuation of homography

#points: (833.5,271.5),(863.5, 339.5),(881.5, 387.5),(903.5, 431.5),(915.5, 453.5),(931.5,503.5)
          (735.5,341.5),(789.5,395.5),(823.5,427.5),(861.5,459.5),(879.5,479.5),(915.5,515.5)


#transpose of a matrix
-
# driver code
X  = [[833.5,271.5,1,0,0,0,-735.5*833.5,-735.5*271.5,-735.5],
    [0,0,0,833.5,271.5,1,-341.5*833.5,-341.5*271.5,-341.5],
    [861.5,339.5,1,0,0,0,-789.5*861.5,-789.5*339.5,-789.5],
    [0,0,0,861.5,339.5,1,-395.5*861.5,-395.5*339.5,-395.5],
    [881.5,385.5,1,0,0,0,-823.5*881.5,-823.5*385.5,-823.5],
    [0,0,0,881.5,385.5,1,-427.5*881.5,-427.5*385.5,-427.5],
     [903.5,429.5,1,0,0,0,-861.5*903.5,-861.5*429.5,-861.5],
     [0,0,0,903.5,429.5,1,-459.5*903.5,-459.5*429.5,-459.5],
     [913.5,451.5,1,0,0,0,-879.5*913.5,-879.5*451.5,-879.5],
     [0,0,0,913.5,451.5,1,-479.5*913.5,-479.5*451.5,-479.5],
     [931.5,503.5,1,0,0,0,-915.5*931.5,-915.5*503.5,-915.5],
     [0,0,0,931.5,503.5,1,-515.5*931.5,-515.5*503.5,-515.5]]
```

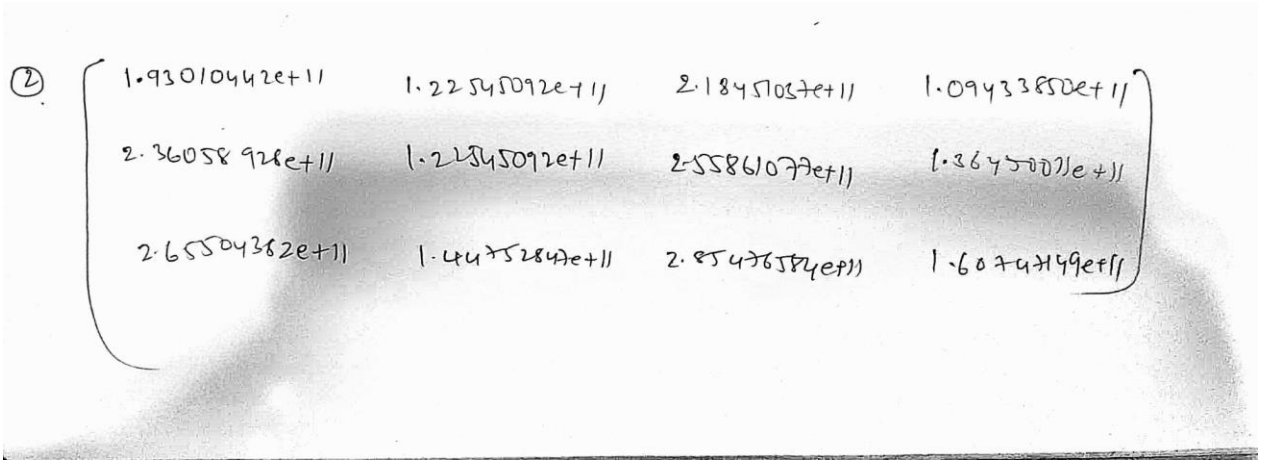
Transpose of matrix:

```
[833.5, 0, 861.5, 0, 881.5, 0, 903.5, 0, 913.5, 0, 931.5, 0]
[271.5, 0, 339.5, 0, 385.5, 0, 429.5, 0, 451.5, 0, 503.5, 0]
[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
[0, 833.5, 0, 861.5, 0, 881.5, 0, 903.5, 0, 913.5, 0, 931.5]
[0, 271.5, 0, 339.5, 0, 385.5, 0, 429.5, 0, 451.5, 0, 503.5]
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
[-613039.25, -284640.25, -680154.25, -340723.25, -725915.25, -376841.25, -778365.25, -415158.25, -803423.25, -438023.25, -852788.25, -480188.25]
[-199688.25, -92717.25, -268035.25, -134272.25, -317459.25, -164801.25, -370014.25, -197355.25, -397094.25, -216494.25, -460954.25, -259554.25]
[-735.5, -341.5, -789.5, -395.5, -823.5, -427.5, -861.5, -459.5, -879.5, -479.5, -915.5, -515.5]
```

Multiplication of matrix and its transpose:

```
[[4.15693829e+11 1.93010442e+11 4.70486132e+11 2.35689607e+11
  5.08408868e+11 2.63927665e+11 5.51057450e+11 2.93918165e+11
  5.71826573e+11 3.11757155e+11 6.14841403e+11 3.46204558e+11]
 [1.93010442e+11 8.96174454e+10 2.18451037e+11 1.09433850e+11
  2.36058928e+11 1.22545092e+11 2.55861077e+11 1.36470011e+11
  2.65504382e+11 1.44752847e+11 2.85476584e+11 1.60747149e+11]
 [4.70486132e+11 2.18451037e+11 5.34454180e+11 2.67734375e+11
  5.78826152e+11 3.00483060e+11 6.28586899e+11 3.35270175e+11
  6.52888629e+11 3.55951844e+11 7.03581236e+11 3.96172174e+11]
 [2.35689607e+11 1.09433850e+11 2.67734375e+11 1.34122384e+11
  2.89962497e+11 1.50527869e+11 3.14890124e+11 1.67954508e+11
  3.27064067e+11 1.78315005e+11 3.52458510e+11 1.98463412e+11]
 [5.08408868e+11 2.36058928e+11 5.78826152e+11 2.89962497e+11
  6.27734929e+11 3.25872843e+11 6.82493323e+11 3.64022333e+11
  7.09280136e+11 3.86696254e+11 7.65387955e+11 4.30974296e+11]
 [2.63927665e+11 1.22545092e+11 3.00483060e+11 1.50527869e+11
  3.25872843e+11 1.69169888e+11 3.54299313e+11 1.88974304e+11
  3.68205027e+11 2.00744936e+11 3.97332018e+11 2.23730841e+11]
 [5.51057450e+11 2.55861077e+11 6.28586899e+11 3.14890124e+11
  6.82493323e+11 3.54299313e+11 7.42764751e+11 3.96169406e+11
  7.72289047e+11 4.21048447e+11 8.34342227e+11 4.69801063e+11]
 [2.93918165e+11 1.36470011e+11 3.35270175e+11 1.67954508e+11
  3.64022333e+11 1.88974304e+11 3.96169406e+11 2.11306679e+11
  4.11916830e+11 2.24576482e+11 4.45014239e+11 2.50579802e+11]
 [5.71826573e+11 2.65504382e+11 6.52888629e+11 3.27064067e+11
  7.09280136e+11 3.68205027e+11 7.72289047e+11 4.11916830e+11
  8.03174574e+11 4.37887107e+11 8.68194073e+11 4.88862358e+11]
 [3.11757155e+11 1.44752847e+11 3.55951844e+11 1.78315005e+11
  3.86696254e+11 2.00744936e+11 4.21048447e+11 2.24576482e+11
  4.37887107e+11 2.38735396e+11 4.73335464e+11 2.66526946e+11]
 [6.14841403e+11 2.85476584e+11 7.03581236e+11 3.52458510e+11
  7.65387955e+11 3.97332018e+11 8.34342227e+11 4.45014239e+11
  8.68194073e+11 4.73335464e+11 9.39728579e+11 5.29142004e+11]
 [3.46204558e+11 1.60747149e+11 3.96172174e+11 1.98463412e+11
  4.30974296e+11 2.23730841e+11 4.69801063e+11 2.50579802e+11
  4.88862358e+11 2.66526946e+11 5.29142004e+11 2.97950551e+11]]
```

Final homography matrix:



PART B: MATLAB Prototyping

3. Write a MATLAB script to find the real world dimensions (e.g. diameter of a ball, side length of a cube) of an object using perspective projection equations. Validate using an experiment where you image an object using your camera from a specific distance (choose any distance but ensure you are able to measure it accurately) between the object and camera.

You can use MATLAB's *ginput()* function to record the pixel coordinates of the object's end points on the image.

Example usage:

```
I = imread('rice.png'); % Read the image
imshow(I); % Display the image
[x y] = ginput(2); % reads two points. x is a 2x1 column vector with x
                   coordinates and y is a 2x1 column vector with y
                   coordinates.
```

```
I = imread('image1.jpg'); % Read the image
imshow(I); % Display the image
[x y] = ginput(2);

fx = 13876.2;
fy = 13039.7;
z0 = 315.2;
x1 = z0*(x(1)/fx);
x2 = z0*(x(2)/fx);
y1 = z0*(y(1)/fy);
y2 = z0*(y(2)/fy);

disp(x(1));
distance = sqrt((x2-x1)^2 +(y2-y1)^2);
disp(distance);
```

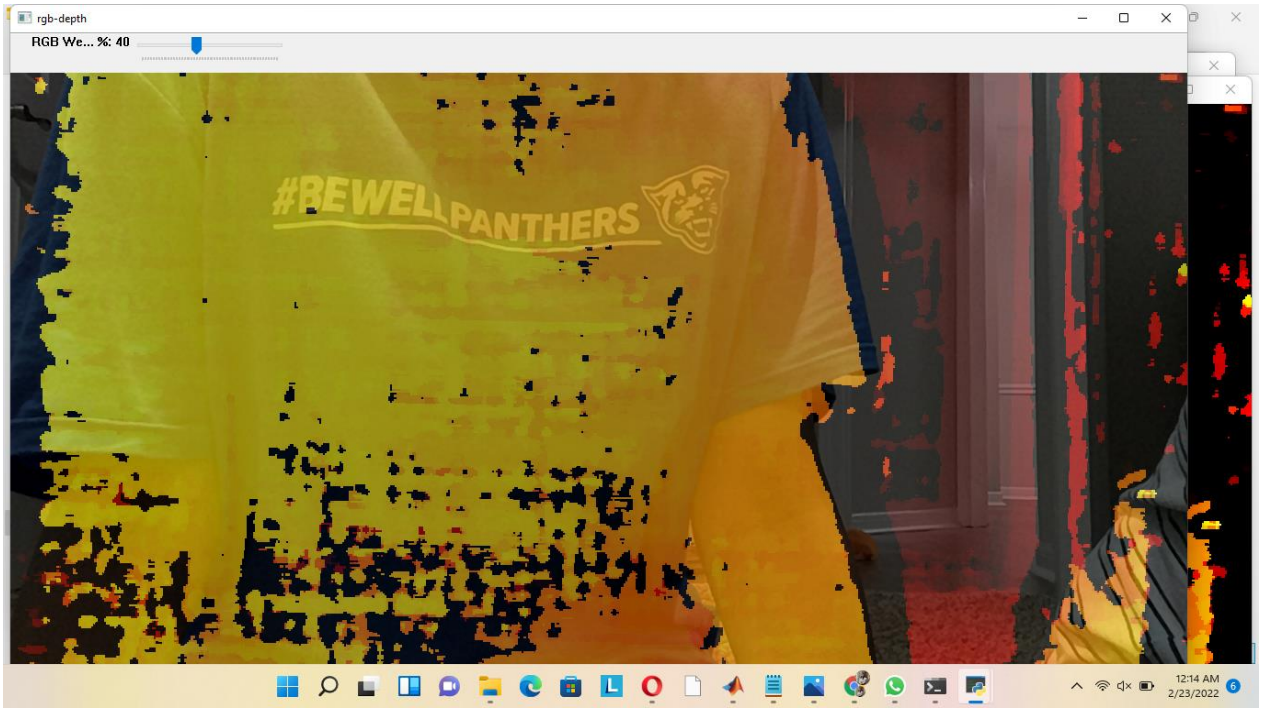
fx = 1.3876e+04

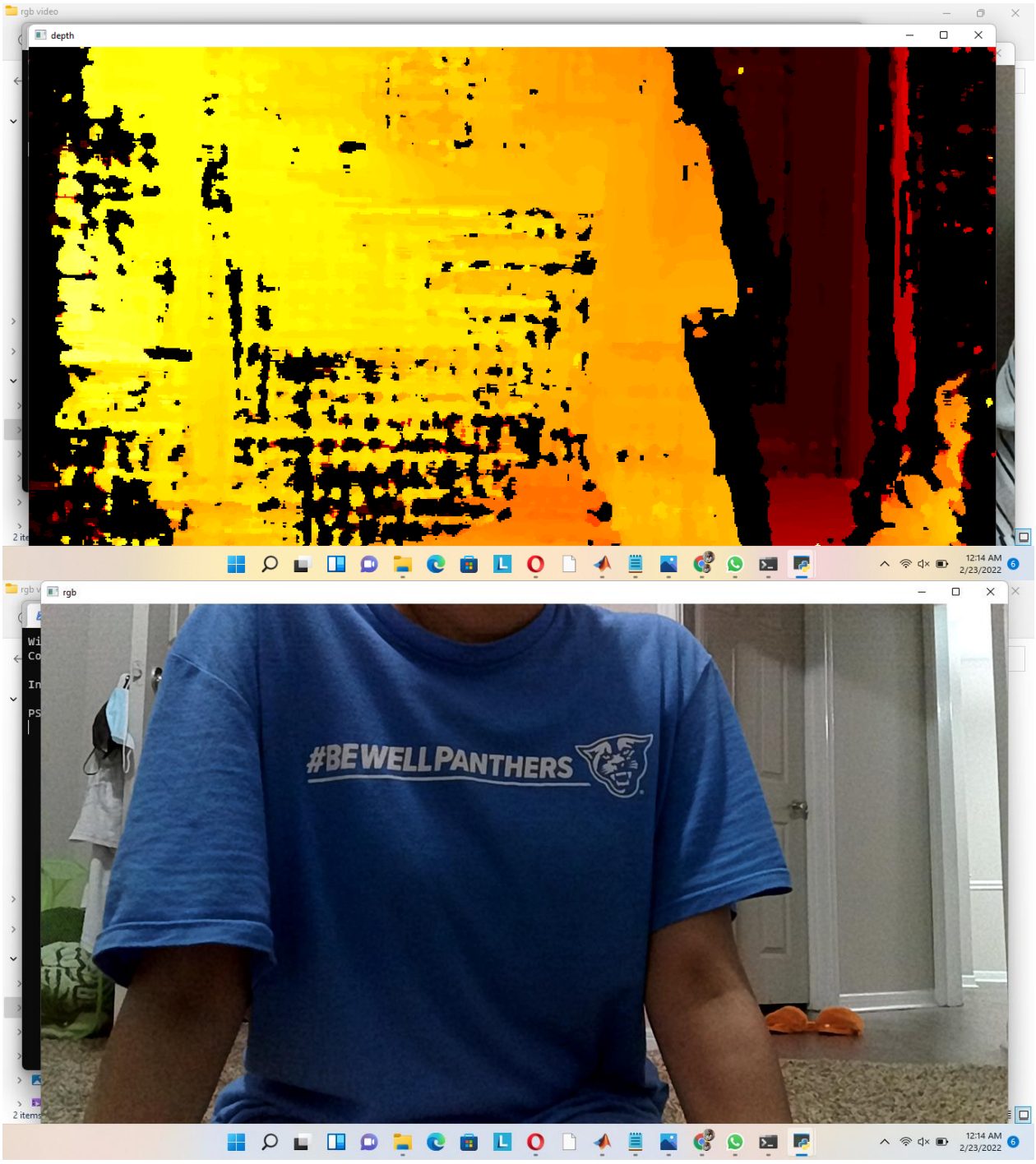
```
605.5000

 14.1858
```

PART C: Application development

4. Familiarize with the Depth AI SDK. Run the tutorials/examples: https://docs.luxonis.com/projects/sdk/en/latest/ (need not submit this)

5. Setup your application to show a RGB stream from the mono camera and a depth map stream from the stereo camera simultaneously. Is it feasible? What is the maximum frame rate and resolution achievable?





6. Run the camera calibration tutorial. Compare the output with answers from Part A and Matlab calibration exercise.