AT70.20: Computer Vision, Midterm Exam

Fri Jul 23, 2021

Name: Solution

Student ID:

Guidelines:

- READ EACH QUESTION CAREFULLY.
- This is exam is **2.5 hours**.
- Fill in your name and student ID above.
- Fill in each question with your answer below.
- Make a copy of this document on Google Docs. When you are finished, turn in your exam on Google Classroom.
- While working on the exam, you may use any resources available in your books, personal notes, and on the Internet, but you MAY NOT INTERACT WITH LIVE HUMAN BEINGS. You cannot send email to anyone (except for the instructor, AFTER you have finished the exam), receive email from anyone, chat with anyone, or do instant messaging with anyone. So that I can monitor your activity during the exam, I will want to see two camera feeds in the Zoom session, one looking at you and one looking at your computer screen. Anyone caught sharing solutions or otherwise violating these rules by the proctors will be dealt with severely, including an F for the course, a letter in your file, and a letter to the Dean.
- Use your time wisely. Note that I give partial credit, so you should be sure to attempt every question, even if your solution is not quite perfect. Show your steps so that I can give you partial credit even if some of the inputs are incorrect.
- Relax, enjoy the exam, use your brain, and good luck!

Background for all questions

We'll use two images the video for Lab 04 for this exam:

https://drive.google.com/file/d/1_QyP5XEuOdMn1Oapq6Z2C6TWFy9vzOSv/view?usp=sharing

https://drive.google.com/file/d/11MQc561Bub12ynPjbCY7HrfaqHSfB8PN/view?usp=sharing

Assume the following camera parameters:

```
%YAML:1.0
```

```
---
K: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 7.96607443e+02, 0.00000000e+00, 1.00207058e+03, 0.00000000e+00,
7.93970516e+02, 4.25696490e+02, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00
]
dist_coeff: !!opencv-matrix
   rows: 5
   cols: 1
   dt: d
   data: [ -1.53874001e-01, 2.36318704e-02, 6.20711711e-05, 1.96694600e-04,
-1.66714132e-03]
```

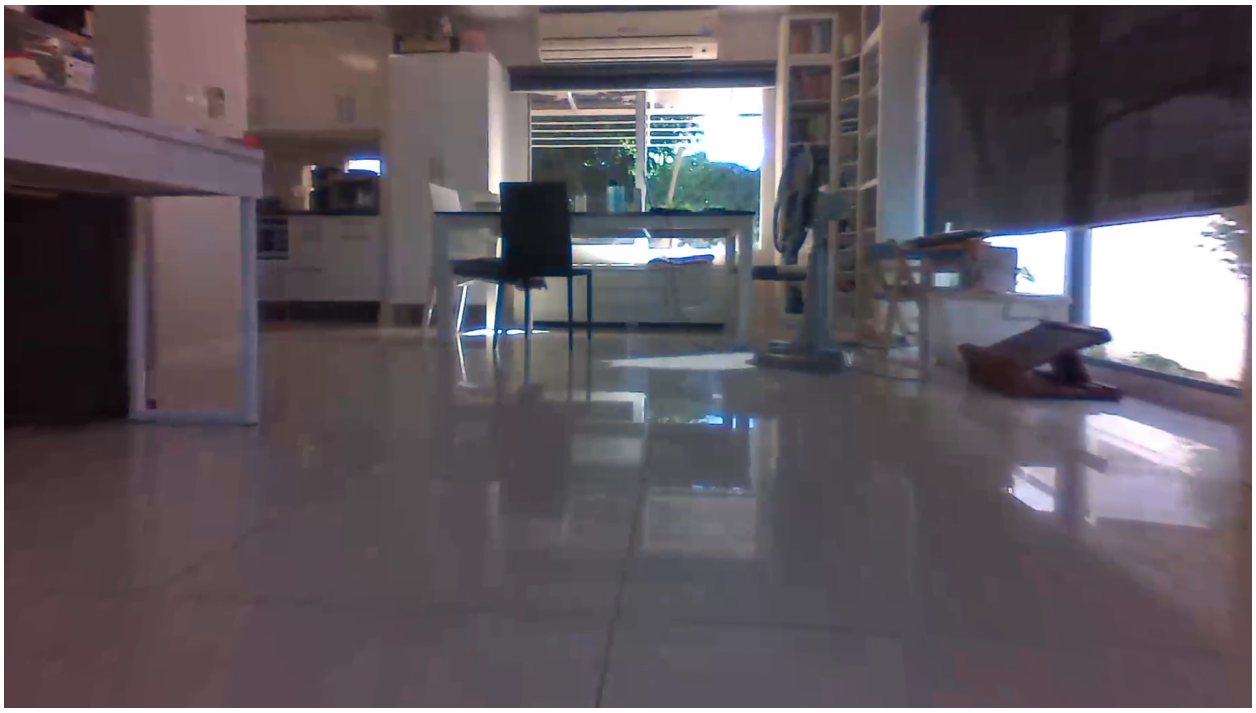
Question 1 (10 points)

Undistort the two images using the given distortion parameters.

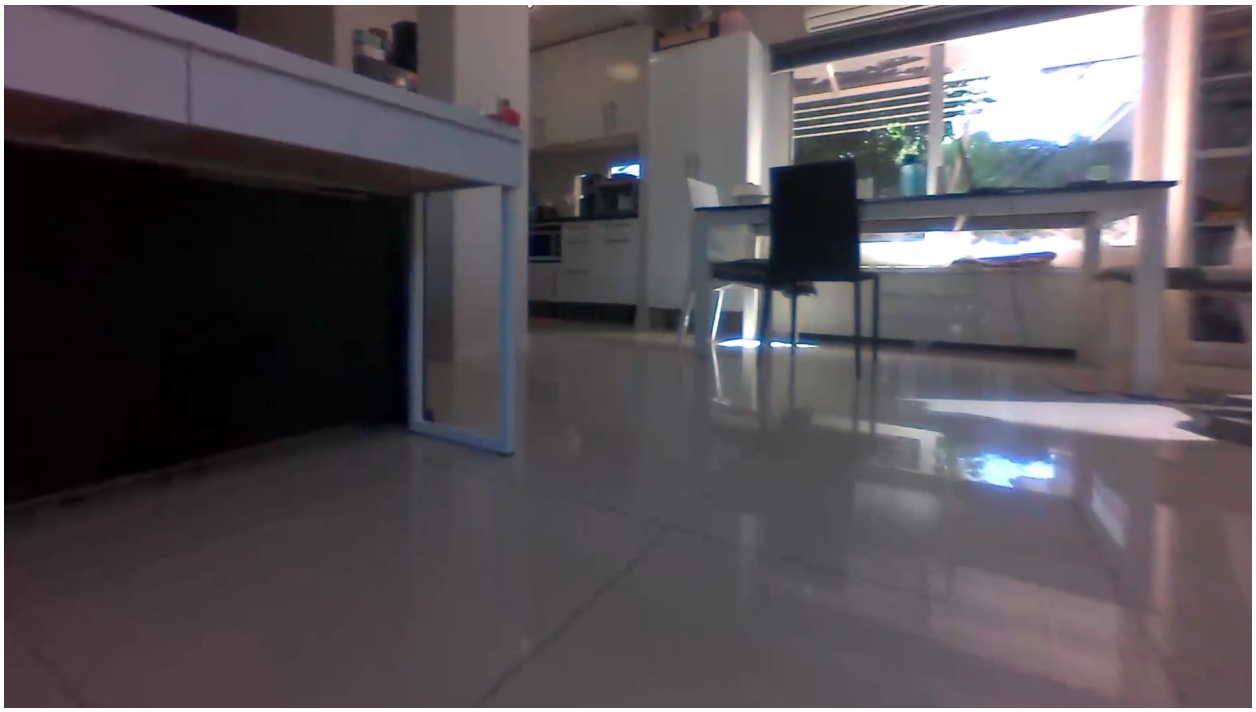
I used the following code:

```
# Question 1


import numpy as np

import cv2


frame1 = cv2.imread('Final-Image-1.png')

frame2 = cv2.imread('Final-Image-2.png')


K = np.array([[7.96607443e+02, 0.00000000e+00, 1.00207058e+03],

              [0.00000000e+00, 7.93970516e+02, 4.25696490e+02],

              [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])

dist_coeff = np.array([-1.53874001e-01, 2.36318704e-02, 6.20711711e-05,
1.96694600e-04, -1.66714132e-03])


frame1_u = cv2.undistort(frame1, K, dist_coeff)

frame2_u = cv2.undistort(frame2, K, dist_coeff)


cv2.imwrite('frame1u.png', frame1_u)

cv2.imwrite('frame2u.png', frame2_u)
```

First image:



Second image:



Question 2 (10 points)

Read the documentation of the OpenCV function `getOptimalNewCameraMatrix()`. Explain a situation in which we would want to use it.

In some situations, we might want to preserve particular parts of the image and not others. For example, if we have extreme fisheye distortion, we might miss valuable information near the corners of the image if we cut off the field of view to exclude any invalid pixels. In this situation, we might prefer to allow the undistortion to put some invalid pixels in the visible region of the image in order that more pixels are visible in other parts of the image. Another situation where this might be useful is if we want the camera's principal point to be the actual center of the image. We see from K that the principal point is around 100 pixels from the center of the image. We may prefer the principal point to be at the center, and getOptimalNewCameraMatrix would do that for us.

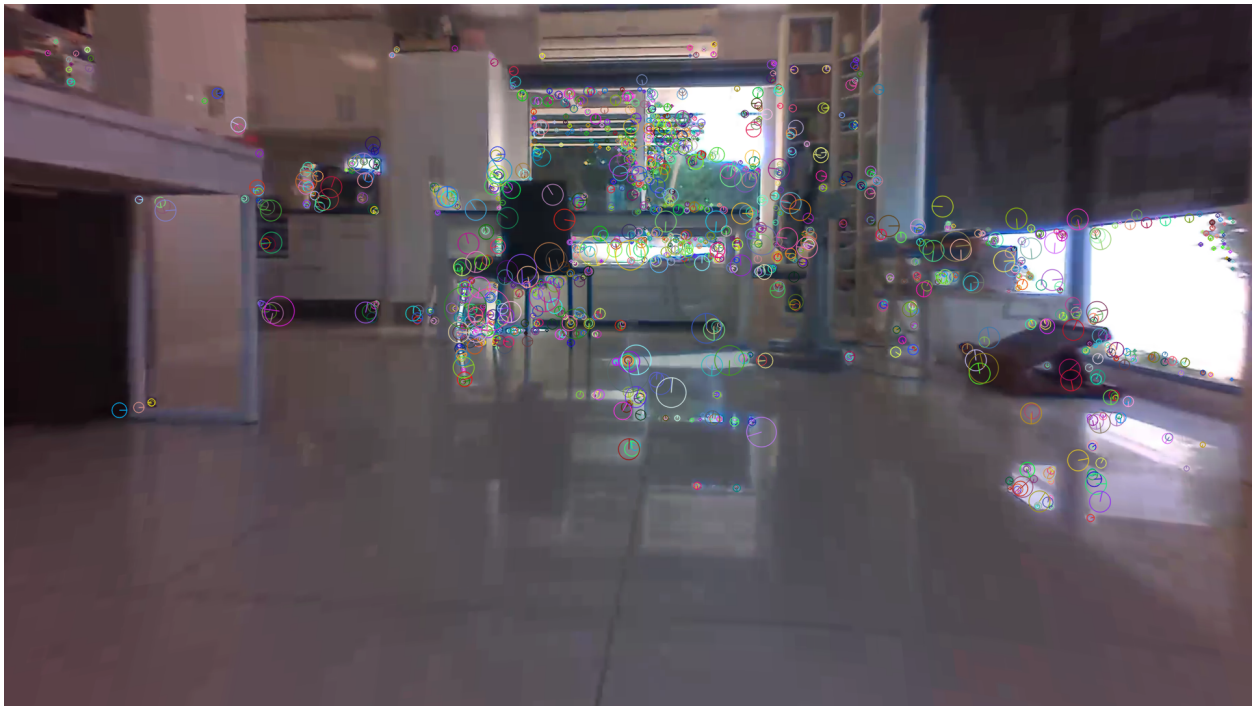
Question 3 (10 points)

Find AKAZE keypoints for the two undistorted images. Display them here.

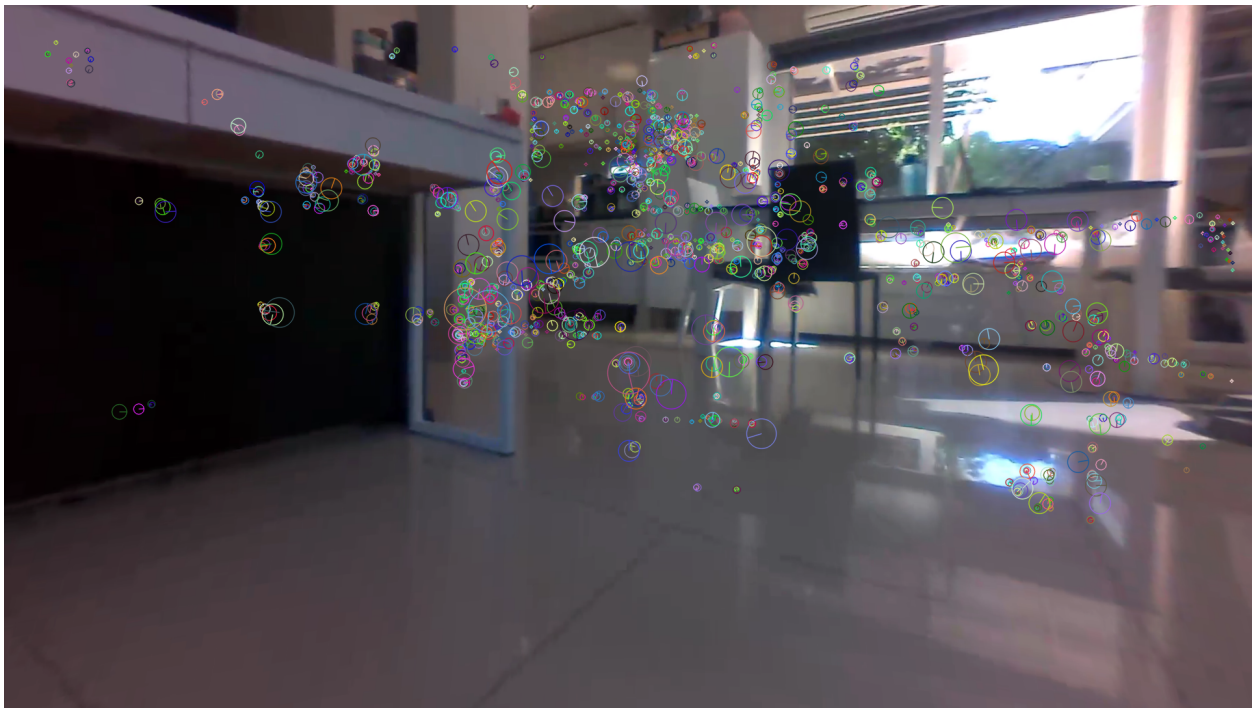
I used the following code:

```
# Question 3


akaze = cv2.AKAZE_create()

kpts1, desc1 = akaze.detectAndCompute(frame1_u, None)

kpts2, desc2 = akaze.detectAndCompute(frame2_u, None)


keypts_image_1 = frame1_u.copy()

keypts_image_2 = frame2_u.copy()

keypts_image_1 = cv2.drawKeypoints(frame1_u, kpts1, keypts_image_1, None,
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

keypts_image_2 = cv2.drawKeypoints(frame2_u, kpts1, keypts_image_2, None,
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)


cv2.imwrite('frame1uk.png', keypts_image_1)

cv2.imwrite('frame2uk.png', keypts_image_2)
```

First image:



Second image:



Question 4 (10 points)

Get keypoint correspondences and remove outlier correspondences using `findEssentialMat()`. Show the remaining point correspondences here.

I used the following code:

```
# Question 4


matcher =
cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_BRUTEFORCE_HAMMING)
nn_matches = matcher.knnMatch(desc1, desc2, 2)
matched1 = []
matched2 = []
matched_pts_1 = []
matched_pts_2 = []
matches = []
nn_match_ratio = 0.8
for m, n in nn_matches:
   if m.distance < nn_match_ratio * n.distance:
       matched_pts_1.append(kpts1[m.queryIdx].pt)
       matched_pts_2.append(kpts2[m.trainIdx].pt)
       matched1.append(kpts1[m.queryIdx])
       matched2.append(kpts2[m.trainIdx])
       matches.append(m)


E, inlier_mask = cv2.findEssentialMat(np.array(matched_pts_1),
np.array(matched_pts_2), K, method=cv2.RANSAC)
inlier_matches = []
for i in range(len(inlier_mask)):
   if inlier_mask[i]:
       inlier_matches.append(matches[i])


out_img = frame1_u.copy()
out_img = cv2.drawMatches(frame1_u, kpts1, frame2_u, kpts2, inlier_matches,
out_img)
cv2.imwrite('inlier-matches.png', out_img)
```

Inlier correspondences:

Question 5 (10 points)

Draw two pairs of epipolar lines over the two images according to the essential matrix you obtained above.

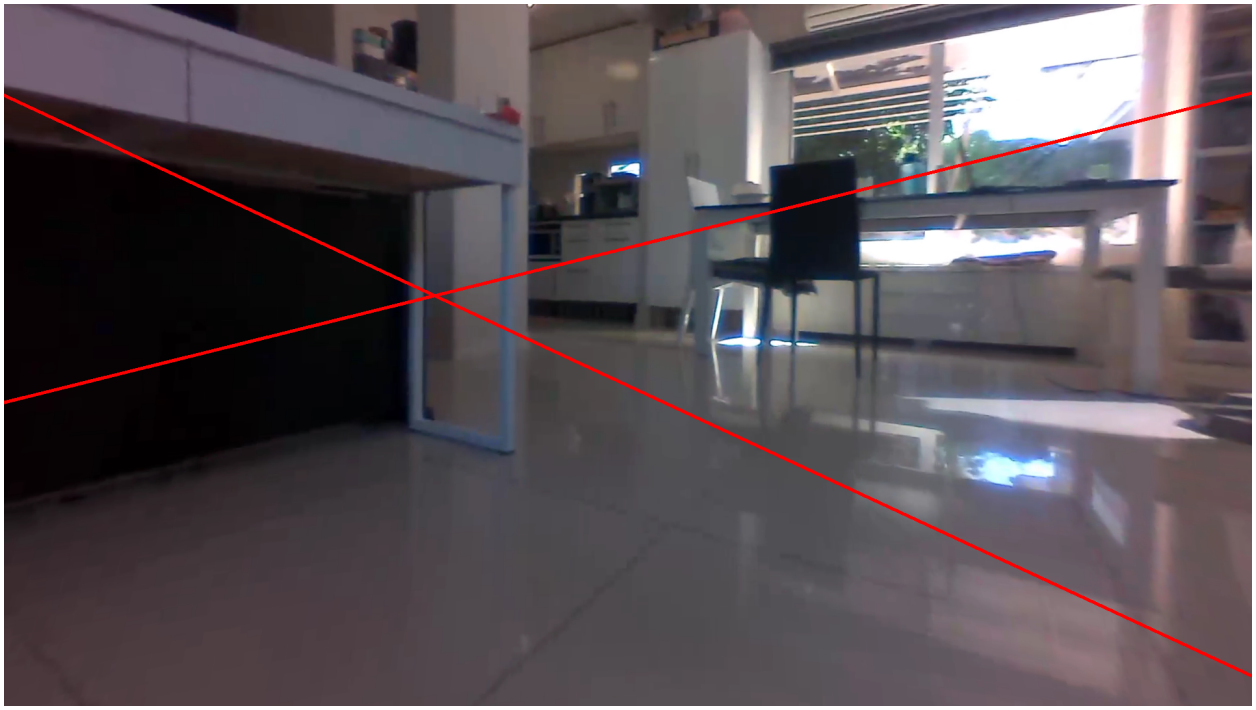
I used this code:

```
# Question 5


def draw_epipolar(frame, l, K):
   p1 = K @ [[-100.0], [-l[0,0] * -100 / l[1,0] - l[2,0] / l[1,0]], [1.0]]
   p2 = K @ [[100.0], [-l[0,0] * 100 / l[1,0] - l[2,0] / l[1,0]], [1.0]]
   cv2.line(frame, (int(p1[0]), int(p1[1])), (int(p2[0]), int(p2[1])), (0, 0,
255))


Kinv = np.linalg.inv(K)
point1 = Kinv @ np.array([[1820], [100], [1]])
point2 = Kinv @ np.array([[1820], [980], [1]])
l1 = E @ point1
l2 = E @ point2
point1p = np.array([[1.0], [-l1[0,0] * 1.0 / l1[1,0] - l1[2,0] / l1[1,0]],
[1.0]])
point2p = np.array([[1.0], [-l2[0,0] * 1.0 / l2[1,0] - l2[2,0] / l2[1,0]],
[1.0]])
l1p = E.T @ point1p
l2p = E.T @ point2p
frame1_e = frame1_u.copy()
frame2_e = frame2_u.copy()
```

```
draw_epipolar(frame1_e, l1p, K)

draw_epipolar(frame1_e, l2p, K)

draw_epipolar(frame2_e, l1, K)

draw_epipolar(frame2_e, l2, K)

cv2.imwrite('frame1_e.png', frame1_e)

cv2.imwrite('frame2_e.png', frame2_e)
```

Image 1:

Image 2:



Question 6 (10 points)

Obtain a rotation and translation between the two images using `recoverPose()`. Interpret the rotation and translation you obtained and discuss whether they make sense.

I used this code:

```
# Question 6


retval, R, t, mask = cv2.recoverPose(E, np.array(inlier_pts1),
np.array(inlier_pts2), K)

twist, _ = cv2.Rodrigues(R)
```

I got R:

[[0.92199696 -0.02242264 0.38654733]

[0.01692794 0.99970156 0.01761346]

[-0.38682691 -0.0096961   0.92210137]]

and t:

[[0.39177787]

[-0.02610467]

[-0.91968943]]

R looks like a rotation around the Y axis. Since this is a rotation in the camera coordinate system, that makes sense. Visually we see that between the two frames, the robot is moving forward and to the left, which would indicate a negative rotation around the Y axis (positive point rotation around the Y axis). Indeed, Rodrigues for this rotation matrix gives

[[-0.01402175]

[0.39707927]

[0.02020406]]

i.e., a rotation of 22.795 degrees around Y. Makes perfect sense! The translation looks good too, since it's indicating the position of camera 1 in camera 2's frame. The values indicate that the first camera is to the right of (X=0.39) and behind (Z=-0.92) the second camera, which agrees with what we can see in the two frames.

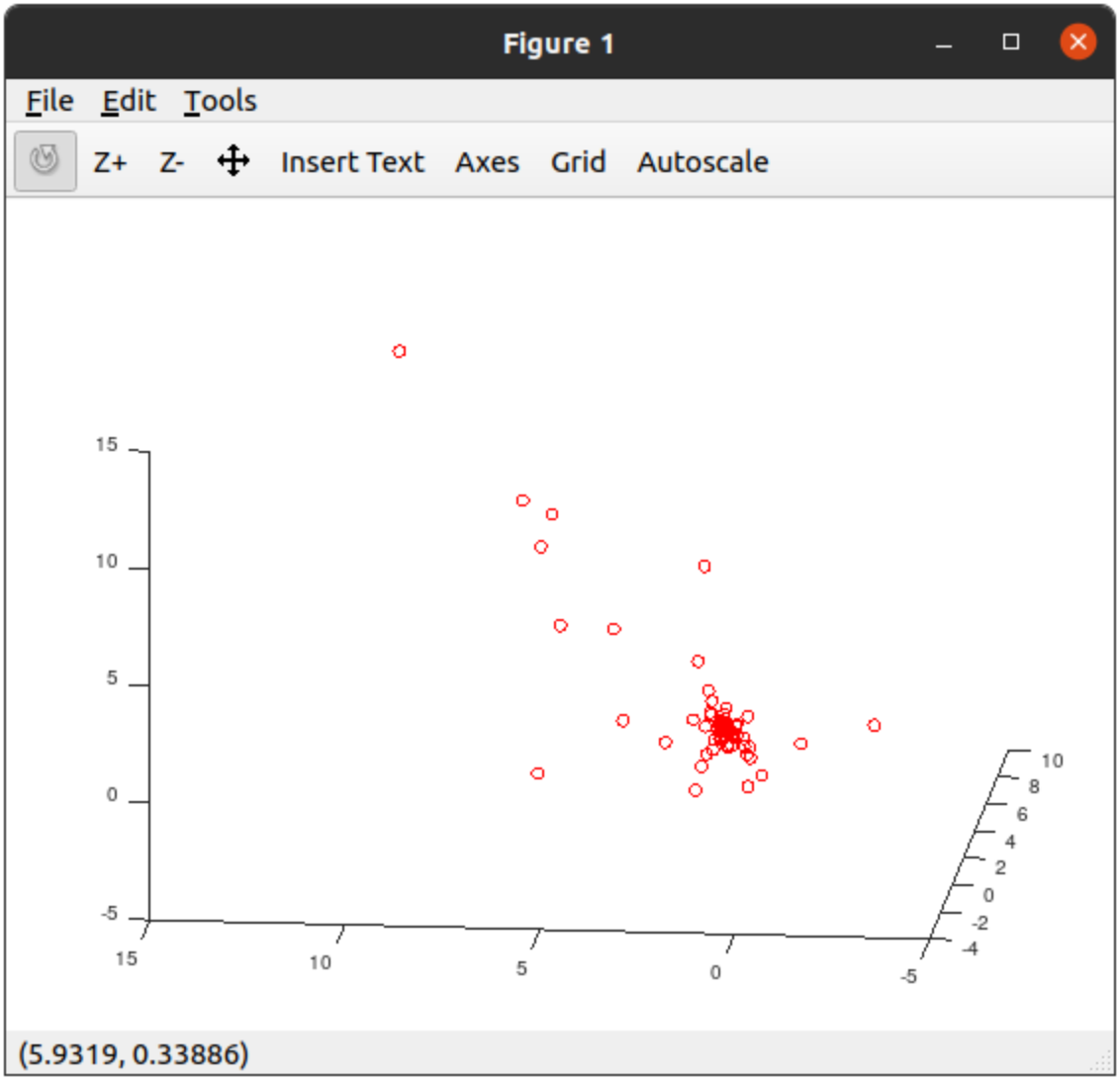
Question 7 (20 points)

Triangulate the inlier correspondences from the previous steps and plot the resulting 3D points. Interpret them and discuss whether they make sense.

I used the following code:

```
# Question 7


P1 = K @ np.array([[1.0, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]])

P2 = K @ np.concatenate((R, t), 1)

points = cv2.triangulatePoints(P1, P2, np.array(inlier_pts1).T,
np.array(inlier_pts2).T)

points = points[0:3,:] / points[3:4,:]
```

I got this result. Two outliers with very large Z values had to be removed. The cluster is the group of points on and around the counter in the pantry.



Question 8 (10 points)

Imagine we are designing an extended Kalman filter to track the position and orientation of the robot that captured the images used in Questions 1-7. Assume our system state is [x, y, θ], where (x, y) specifies a position on the ground plane and θ is the rotation of the robot around its z axis. Suppose we plant several orange balls 10cm in diameter in the environment at known 3D points, and we write a program capable of detecting the *largest visible* orange ball in the image, outputting its x position, y position, and diameter in pixels. Suppose we have a known camera calibration matrix K as well as a known transformation from camera coordinates to robot coordinates Tcr and its inverse Trc. Provide pseudocode for the nonlinear sensor model function **z** = h(**x**) below, using Trc, K, and an array of orange ball positions.

def h(x):

z = null

Dbest = 0

Compute Twr for x

For each landmark ball position X:

Xrobot = Twr * X

Let Xcamera1 = [X, Y, Z] from Trc * Xrobot

Ximage1 = K * Xcamera1

Let Xcamera2 = [X+0.05, Y, Z]

Ximage2 = K * Xcamera2

D = ||Ximage2-Ximage1|| * 2

If D > Dbest:

Dbest = D

z = Ximage1 concatenated with D

return z

Question 9 (10 points)

Explain what might happen with the system described in Question 8 if the robot happens to be at a position in the environment in which two orange balls' projections into the image are equal in diameter. Is there anything that could be done to address this problem?

If two balls have approximately the same diameter when projected into the image, we might detect either one as the largest, arbitrarily. If the ball the sensor model function selects doesn't match the ball the sensor detected, we'll have a huge error and this will throw the filter off. We could fix the problem by keeping a list of all balls' expected positions and sizes and matching the one returned by the sensor with the closest projection in the list. This would prevent a mismatch in most cases.