BIOS 7718 Homework 2

Piper Williams

April 8, 2019

Problem 1, Part 1

For this problem, we were asked to use RANSAC to estimate inliers in the presence of outliers. For this problem, 30 inliers were generated as follows. First, a 3D line consisting of x^o, y^o, and z^o exists such that:

$$x^o + 2y^o + 3z^o = 1$$
$$x^o - y^o = 0$$

$n = 30$ inliers $\{(x_i^o, y_i^o, z_i^o)\}_{i=1}^n$ were generated from this line. x_i^o was set to equal $0.4, 0.41, 0.42, ..., 0.68, 0.69$. y_i^o and z_i^o were solved for accordingly. Then, noise was added to the inliers, and this noise followed a Gaussian distribution with a mean of $\mathbf{0}$ and a variance-covariance structure of $(0.03^2\boldsymbol{I})$, where $\boldsymbol{I}$ is a 3×3 identity matrix. Finally, 100 outliers were generated that were uniformly distributed between (0, 0, 0) and (1, 1, 1). The inlier and outlier data were concatenated to make one data set consisting of 130 rows. Then, RANSAC was implemented to estimate the true inliers in this data set. To implement RANSAC accurately, we had to define the value of a few key parameters. First, as stated in this problem, the probability of choosing at least one trial free from outliers was set to 0.99. Then, we needed to determine the number of maximum iterations necessary, k, as well as the distance threshold, t. t is usually set empirically, and was set to 0.08 for this problem. However, k is determined from the following inequality:

$$k > \frac{log(1-p)}{log(1-(1-\epsilon)^n)}$$

Where p equals the probability of choosing at least one trial free from outliers ($= 0.99$), ϵ is equal to the outlier proportion ($= \frac{10}{13}$), and n is equal to the minimum number of points needed to fit the model ($= 2$ when fitting a line). With this information, k was determined to need to be at least 85. For this problem, k was set to 500. RANSAC was then implemented using these parameters (Fig. 1).

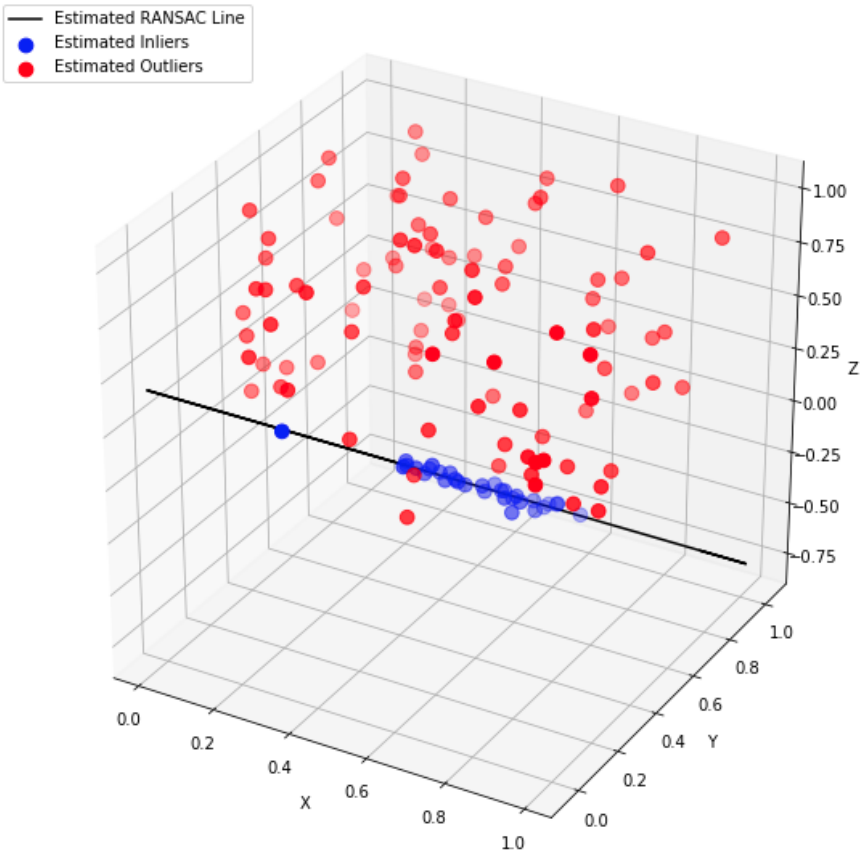


Fig. 1: RANSAC estimation of inliers.

Here, we can see the estimated line fit to the inlier data, the estimated inliers, and the estimated outliers. All 30 of the true inliers were detected via RANSAC, while only 1 of the 100 outliers were incorrectly classified as an inlier.

Problem 1, Part 2

For this problem, we were asked to add another set of inliers to the previously-generated data in Problem 1, Part 1. The new set of inliers ($m = 20$) were generated using the following 3D line:

$$\begin{aligned} x^o + y^o + z^o &= 1 \\ x^o - y^o &= 0 \end{aligned}$$

Similarly to the previous problem, noise was then added to these inliers, and the noise followed a Gaussian distribution with a mean of $\mathbf{0}$ and a variance-covariance structure of $(0.01^2\boldsymbol{I})$, where $\boldsymbol{I}$ is a 3×3 identity matrix. These new inliers were then added to the previously-generated data. Thus, the new data set has 50 inliers and 100 outliers, totaling 150 rows. Parameters for each RANSAC iteration were calculated just as they were in Problem 1, Part 1. RANSAC was fit iteratively to each set of inliers, and the estimated inliers and fitted 3D line for each iteration were plotted in one 3D graph (Fig. 2).

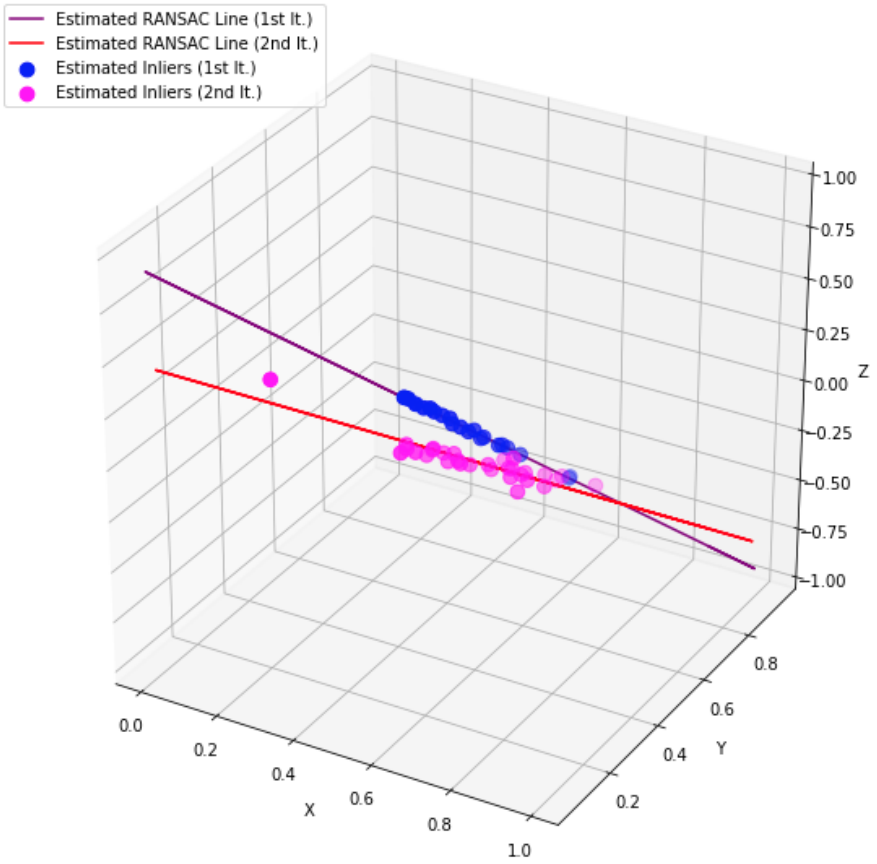


Fig. 2: Iterative RANSAC estimation of two sets of inliers.

The outliers were not plotted in this figure to better visualize the two sets of estimated inliers. Figure 2 shows the estimated inliers for each iteration as well as the corresponding 3D line fit for each iteration. In the first RANSAC iteration, 18 inliers were detected from the first set of inliers ($m = 20$). However, 2 outliers were also incorrectly classified as inliers. In the second RANSAC iteration, a total of 30 inliers were detected. While most of these were correctly detected from the second set of inliers ($n = 30$), a couple of true outliers were incorrectly classified as inliers. Overall, the RANSAC method seemed to do well in correctly identifying inliers.

Problem 2, Part 1

For this problem, we used the Hough circle transform to detect wheels in an image of a car. Before implementing the transform, the image was Gaussian smoothed with $\sigma = 3$ and a kernel size of 19×19. The Hough circle transformed was then applied to the Gaussian-smoothed image, the circles and center of the circles detected are shown in the original image below (Fig. 3).

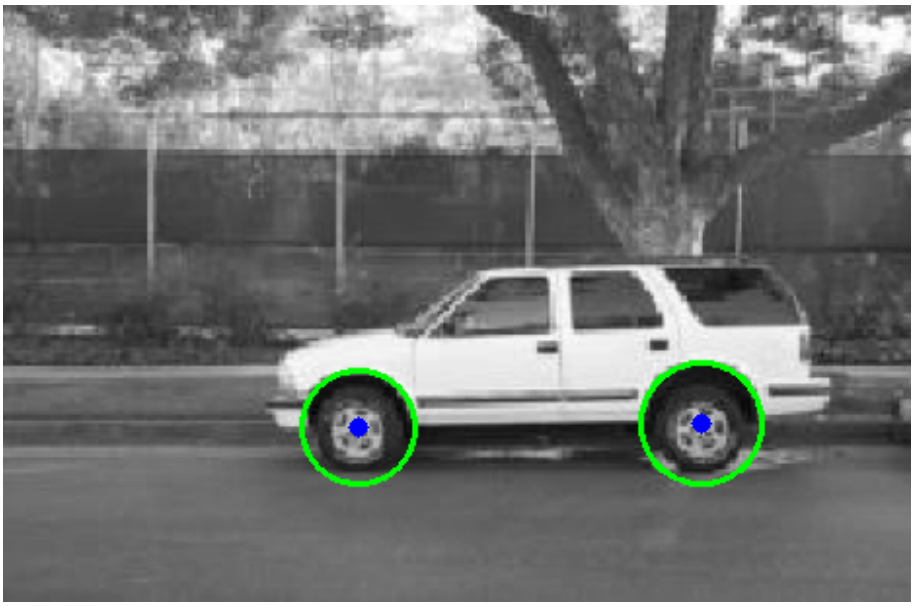


Fig. 3: Hough circle transformed applied to an image of a car to detect the boundary and center of the wheels.

Here, we can see that the Hough circle transform did not perfectly detect the boundaries of the wheels and the center of the wheels. However, it did reasonably well. This could be due to the image quality. The parameters that must be specified in the Hough circle transform include 1.) the inverse ratio of accumulator and image resolution, 2.) minimum distance between detected centers, 3.) upper threshold for the internal Canny edge detector, 4.) threshold for center detection, 5.) minimum circle radius, and 6.) maximum circle radius. To detect the wheels in this image, these parameters were altered accordingly.

Problem 2, Part 2

The Hough circle transform was implemented here to detect and localize nuclei in a histology image. First, the RGB image was converted to a gray-scale image. The gray-scale image was then Gaussian smoothed with $\sigma = 5$ and a kernel size of 31×31. Finally, the Hough circle transform was applied to the image, and the center of the circles were marked accordingly (Fig. 4).

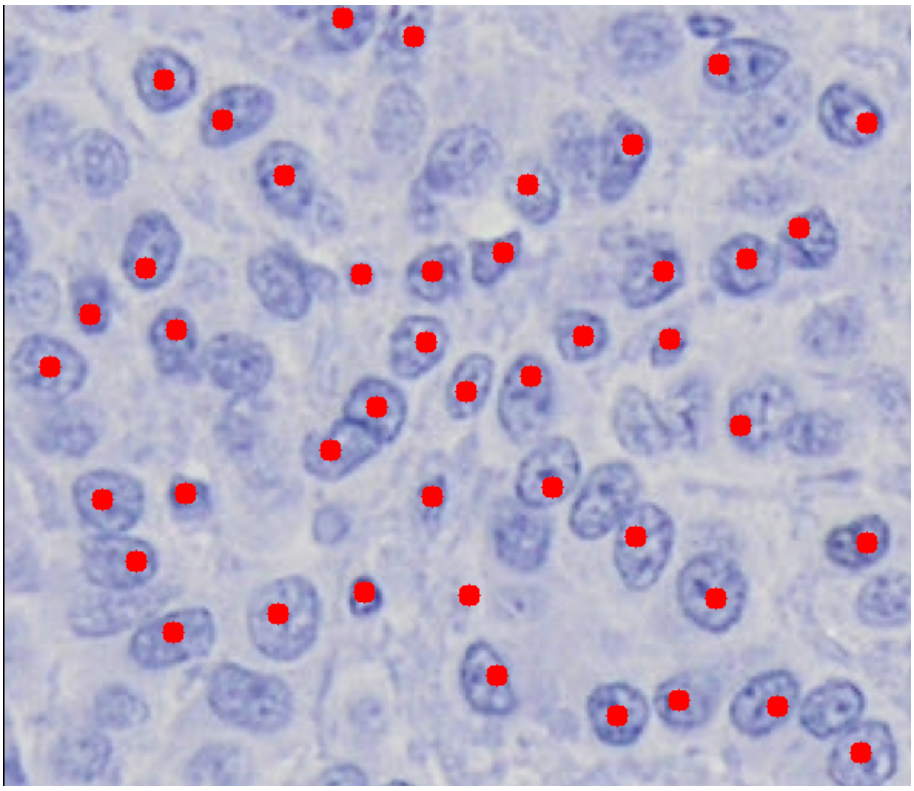


Fig. 4: Hough circle transformed applied to a histology image to detect and localize individual nuclei. 44 nuclei were detected.

Again, the Hough circle transform was not perfect. However, most of the nuclei were accurately detected. Here, 44 nuclei were detected in this image.

Problem 3, Part 1

Here, we used the marker-controlled watershed transform for object segmentation. For this problem, we first read in the image and converted it to gray-scale. Because the foreground in this image was darker than the background, we needed to invert the pixels in the gray-scale image. Otherwise, the erosion and dilation operations will not work appropriately. Next, the image was binarized using Otsu's binarization. Then, to detect the sure foreground and sure background of the image, the binary image was eroded and dilated, respectively. The dilated and eroded images were then subtracted from one another to get the "unknown" image region. The markers were then labeled using the $cv2.connectedComponents$ function in $openCV$, where the background is labeled with 0 and all other objects are labeled with integers starting from 1. Due to the way the watershed transform operates, it considers pixels with a value of 0 as unknown. Thus, to ensure proper operation of the watershed transform, the unknown pixels were labeled with a 0 and the background was labeled with a value of 1. After this, the watershed transform was applied to the image using the markers detected (Fig. 5).

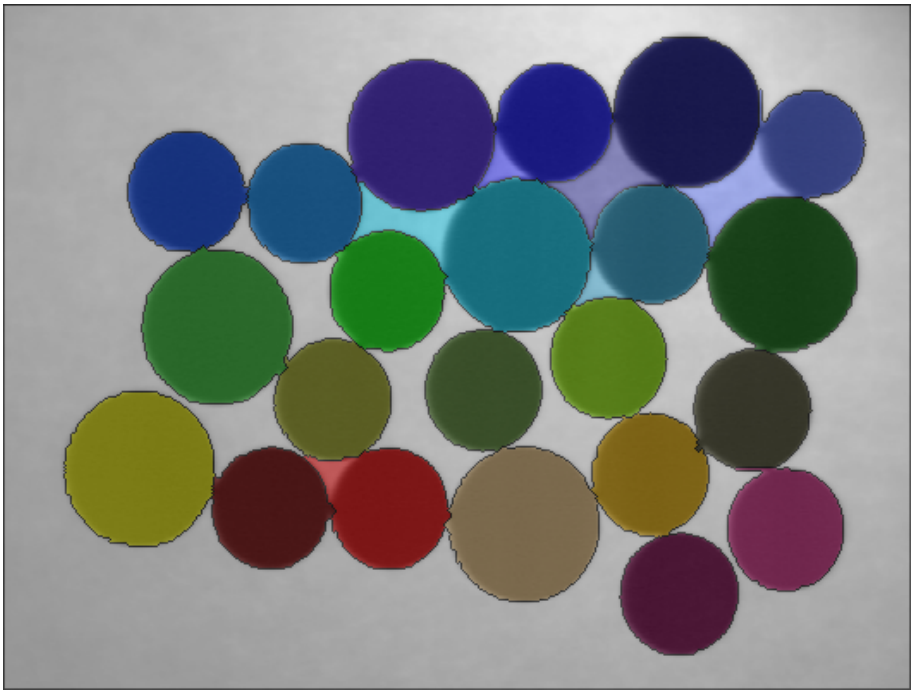


Fig. 5: Object segmentation using the watershed transform.

Based on how the watershed function operates in *openCV*, I could not figure out how to change the color of the boundaries/contours specifically. This is due to the fact that the *boundaries* of the image objects all shared the same pixel value of negative 1. Thus, the boundaries for all objects would be the same color. However, I was able to label the actual image objects themselves with different colors, each color indicating a different segment. The color-segmented image was overlaid on the original image to get a better sense of how well the watershed transform performed (Fig. 5). It seems that the watershed transform did fairly well. Some of the edges did not properly form, as can be seen in the areas where the color leaked into background areas. However, each coin was detected as a separate segment, indicated by the fact that each coin is a different color.

Problem 3, Part 2

The watershed transform was then applied to an image of cells/particles. Similarly to Problem 3, Part 1, the gray-scale image was binarized using Otsu's binarization method and the sure background was determined using dilation methods. However, before dilating the image, noise was removed using the opening morphology operation. Then, the sure foreground was determined using a distance transform as opposed to erosion. If the objects are touching, the distance transform is usually preferred over simple erosion to better segment individual objects. The sure foreground was then subtracted from the sure background to obtain the "unknown" region. The remainder of this problem followed the same steps followed in Problem 3, Part 1 once the unknown region was obtained. The results from the watershed transform are shown below (Fig. 6).

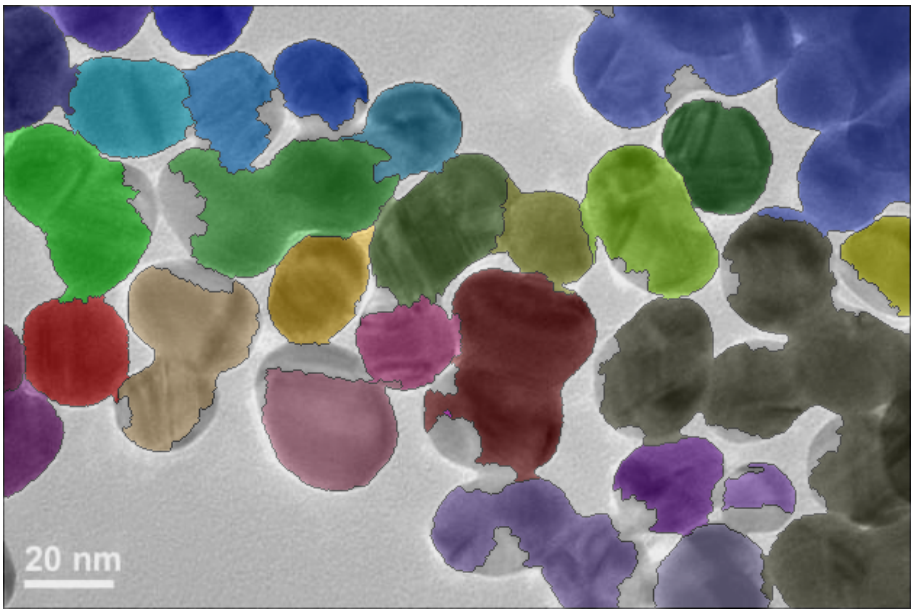


Fig. 6: Object segmentation using the watershed transform.

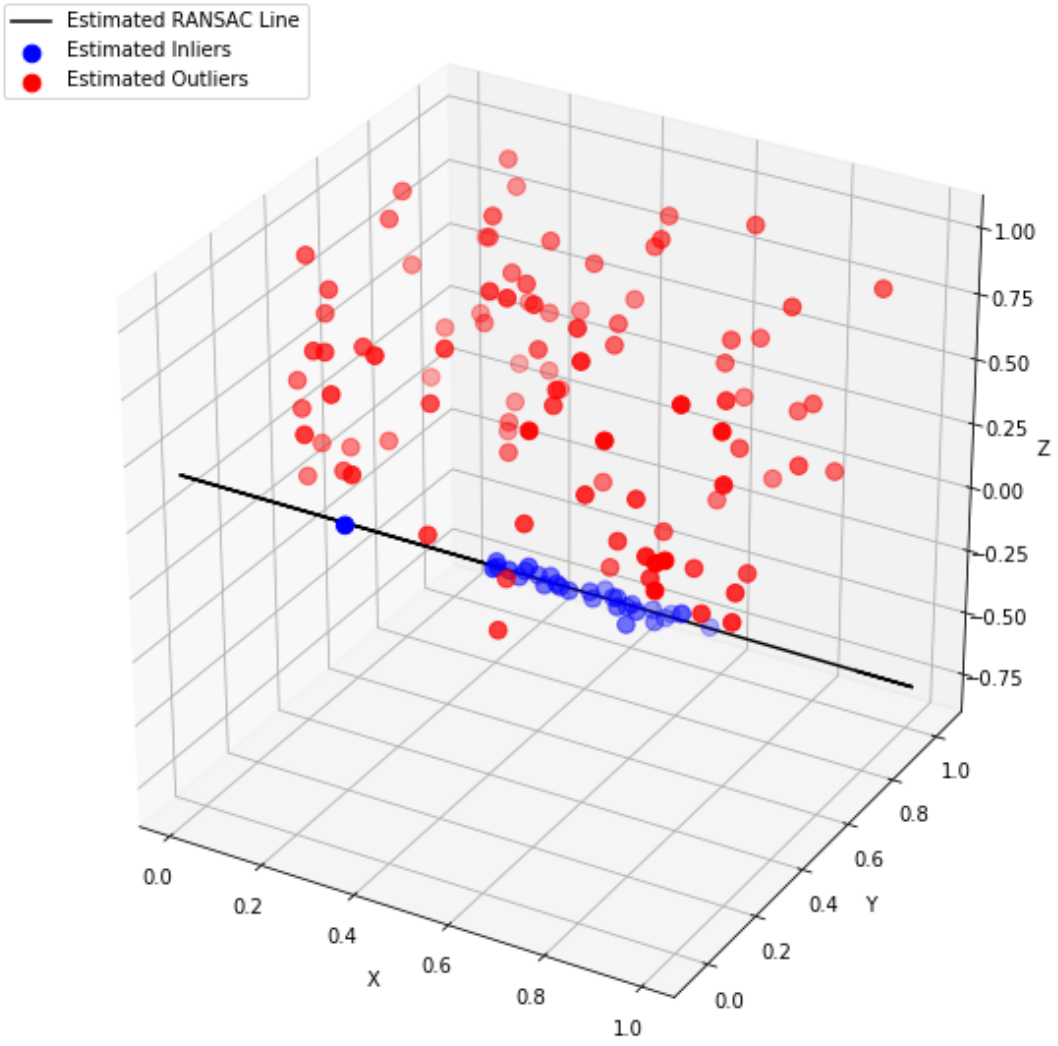
Here, we can see that the watershed transform did relatively well. All of the cells in the image were detected. However, the boundaries calculated from the watershed transform do not match the actual boundaries of the cells perfectly. Also, the grouping of cells in the upper and lower right regions of the image were not recognized as individual cells. Thus, multiple cells were colored as one segment.

BIOS 7718-Assignment 2 Source Code

April 8, 2019

In [44]:
```
##### Problem 1.1 #####
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from skimage.measure import LineModelND, ransac

# generate inliers prior to adding Gaussian noise
x = np.arange(0.4, 0.70, 0.01).reshape(-1,1)
y = x
z = (1 - x - 2*y)/3
inl = np.concatenate((x, y, z), axis = 1)

# generate Gaussian noise
# set seed
np.random.seed(1212)
mu = (0, 0, 0)
cov = [[(0.03)**2,0,0], [0,(0.03)**2,0], [0,0,(0.03)**2]]
err = np.random.multivariate_normal(mu, cov, 30)

# add noise to inliers
inl_final = inl + err

# generate outliers
# set seed
np.random.seed(1212)
outl = []

for i in range(1,101):
    single_outl = np.random.uniform([0,0,0], [1,1,1])
    outl.append(single_outl)
outl_final = np.array(outl)

# combine all data
all_data = np.vstack((inl_final, outl_final))

# designate inliers and outliers
indices = np.arange(all_data.shape[0])
```

```
inlier_indices = indices[0:30]
outlier_indices = indices[30:]

# robustly fit line only using inlier data with RANSAC algorithm
# set seed
np.random.seed(1212)
model_ransac, inliers = ransac(all_data, LineModelND,
                               min_samples=2,
                               residual_threshold=0.08,
                               max_trials=500,
                               stop_probability = 0.99)
outliers = inliers == False

# 3D line coordinates
pred_line = model_ransac.predict(all_data[:,0])

# plot 3D line and data points fit using RANSAC
fig = plt.figure(figsize = (10,10))
ax = plt.axes(projection='3d')
ax.scatter(all_data[inliers][:, 0],
           all_data[inliers][:, 1],
           all_data[inliers][:, 2], c='b',
           marker='o', label='Estimated Inliers', s=75)
ax.scatter(all_data[outliers][:, 0],
           all_data[outliers][:, 1],
           all_data[outliers][:, 2], c='r',
           marker='o', label='Estimated Outliers', s=75)
ax.plot(pred_line[:,0], pred_line[:,1],
        pred_line[:,2], color = 'black',
        label='Estimated RANSAC Line')
ax.set_xlabel('X'), ax.set_ylabel('Y'), ax.set_zlabel('Z')
ax.legend(loc='upper left')
plt.show()
```



In [45]:
```
# number of true inliers detected
np.sum(inliers[0:30])
```

Out[45]: 30

In [46]:
```
# number of outliers classified as inliers
np.sum(inliers[30:])
```

Out[46]: 1

In [47]:
```
##### Problem 1.2 #####
# generate inliers prior to adding Gaussian noise
x_2 = np.arange(0.4, 0.60, 0.01).reshape(-1,1)
y_2 = x_2
```

```
z_2 = 1 - x_2 - y_2
inl_2 = np.concatenate((x_2, y_2, z_2), axis = 1)

# generate Gaussian noise
# set seed
np.random.seed(1212)
mu_2 = (0, 0, 0)
cov_2 = [[(0.01)**2,0,0], [0,(0.01)**2,0], [0,0,(0.01)**2]]
err_2 = np.random.multivariate_normal(mu_2, cov_2, 20)

# add noise to inliers
inl_2_final = inl_2 + err_2

# combine all data
all_data_2 = np.vstack((inl_2_final, all_data))

# fit first line
# set seed
np.random.seed(1212)
model_ransac_1, inliers_1 = ransac(all_data_2, LineModelND,
                                   min_samples=2,
                                   residual_threshold=0.02,
                                   max_trials=500,
                                   stop_probability = 0.99)
outliers_1 = inliers_1 == False

# 3D line coordinates
pred_line_1 = model_ransac_1.predict(all_data_2[:,0])
```

In [48]:
```
# number of inliers detected (1st It.)
np.sum(inliers_1)
```

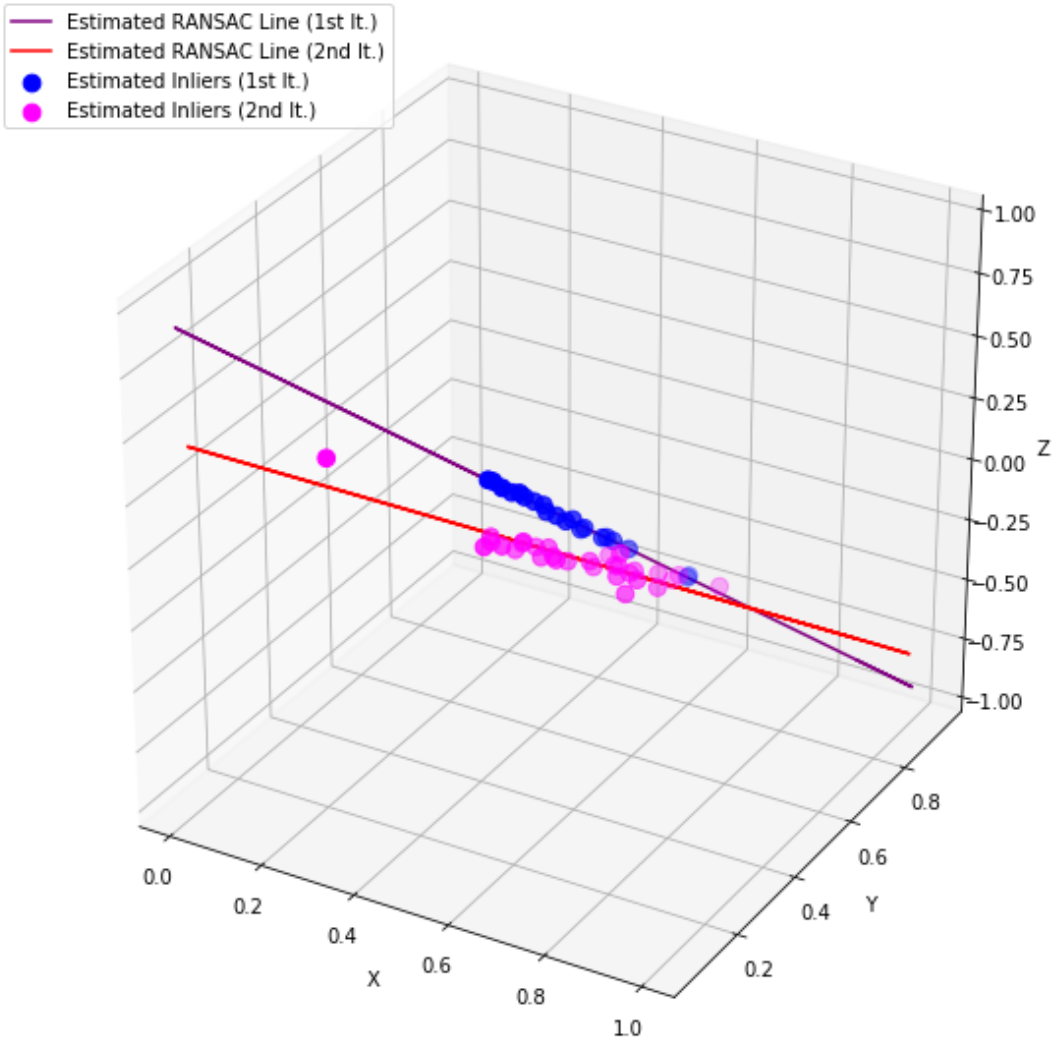
Out[48]: 20

In [49]:
```
# number of TRUE inliers detected (1st It.)
np.sum(inliers_1[0:20])
```

Out[49]: 18

In [50]:
```
# exlude inliers detected from second RANSAC fitting
all_data_2_exc = all_data_2[outliers_1]

# fit first line
# set seed
np.random.seed(1212)
model_ransac_2, inliers_2 = ransac(all_data_2_exc,
                                   LineModelND,
                                   min_samples=2,
                                   residual_threshold=0.08,
```

```
                                    max_trials=500,
                                    stop_probability = 0.99)
outliers_2 = inliers_2 == False

# 3D line coordinates
pred_line_2 = model_ransac_2.predict(all_data_2_exc[:,0])

# plot 3D line and data points fit using RANSAC
fig = plt.figure(figsize = (10,10))
ax = plt.axes(projection='3d')
ax.scatter(all_data_2[inliers_1][:, 0],
           all_data_2[inliers_1][:, 1],
           all_data_2[inliers_1][:, 2], c='b',
           marker='o',
           label='Estimated Inliers (1st It.)', s=75)
ax.plot(pred_line_1[:,0],
        pred_line_1[:,1],
        pred_line_1[:,2],
        color = 'purple',
        label='Estimated RANSAC Line (1st It.)')
ax.scatter(all_data_2_exc[inliers_2][:, 0],
           all_data_2_exc[inliers_2][:, 1],
           all_data_2_exc[inliers_2][:, 2],
           c='magenta',
           marker='o',
           label='Estimated Inliers (2nd It.)', s=75)
ax.plot(pred_line_2[:,0], pred_line_2[:,1],
        pred_line_2[:,2], color = 'r',
        label='Estimated RANSAC Line (2nd It.)')

ax.set_xlabel('X'), ax.set_ylabel('Y'), ax.set_zlabel('Z')
ax.legend(loc='upper left')
plt.show()
```




```
In [51]: # number of inliers detected (2nd It.)
         np.sum(inliers_2)
```

```
Out[51]: 30
```

```
In [52]: ##### Problem 2.1 #####
         import cv2

         # read image in
         car_img = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Ass
         car_img_grey = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments

         # smooth image with Gaussian filter
         car_img_smooth = cv2.GaussianBlur(car_img_grey, (19,19), 3)
```

```
        # Hough transform to detect circles
        circles = cv2.HoughCircles(car_img_smooth, cv2.HOUGH_GRADIENT, 1, 80,
                                   param1=100, param2=30, minRadius=0, maxRadius=0)
        circles = np.uint16(np.around(circles))

        for i in circles[0,:]:
            # draw the outer circle
            cv2.circle(car_img,(i[0],i[1]),i[2],(0,255,0),2)
            # draw the center of the circle
            cv2.circle(car_img,(i[0],i[1]),2,(255,0,0),5)

        cv2.imwrite('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignment 2/
```

Out[52]: True

```
In [53]: ##### Problem 2.2 #####
         # read image in
         nuclei_img = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/A
         nuclei_img_grey = cv2.cvtColor(nuclei_img, cv2.COLOR_BGR2GRAY)

         # smooth image with Gaussian filter
         nuclei_img_smooth = cv2.GaussianBlur(nuclei_img_grey, (31,31), 5)

         # Hough transform to detect circles
         circles = cv2.HoughCircles(nuclei_img_grey, cv2.HOUGH_GRADIENT, 1, 40,
                                    param1=150, param2=12, minRadius=0, maxRadius=30)

         circles = np.uint16(np.around(circles))

         for i in circles[0,:]:
             # draw the center of the circle
             cv2.circle(nuclei_img,(i[0],i[1]),2,(0,0,255),10)

         cv2.imwrite('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignment 2/
```

Out[53]: True

```
In [54]: # number of detected nuclei
         np.shape(circles)[1]
```

Out[54]: 44

```
In [55]: ##### Problem 3.1 #####
         coin_img = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Ass
         coin_img_grey = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignment
         coin_img_grey_final = ~coin_img_grey

         # create kernel
```

```
kernel = np.ones((11,11), np.uint8)

# thresholding grey-scale image
ret, thresh_img = cv2.threshold(coin_img_grey_final, 0, 255, cv2.THRESH_BINARY + cv2.

# erode image to get foreground
erode_img = cv2.erode(thresh_img, kernel, iterations=2)

# dilate image to get background
dilate_img = cv2.dilate(thresh_img, kernel, iterations=2)

# subtract eroded and dilated images to get 'unknown' region
unknown = cv2.subtract(dilate_img, erode_img)

# label markers
ret, markers = cv2.connectedComponents(erode_img)

# add 1 to all labels so that sure background is 1
markers = markers+1

# mark unknown region with 0
markers[unknown==255] = 0

# apply watershed transform
markers = cv2.watershed(coin_img, markers)

# make each segment a different color
coin_img[markers == -1] = [0,0,0]
coin_img[markers == 1] = [211,211,211]
coin_img[markers == 2] = [128,0,0]
coin_img[markers == 3] = [220,20,60]
coin_img[markers == 4] = [255,0,0]
coin_img[markers == 5] = [255,99,71]
coin_img[markers == 6] = [255,69,0]
coin_img[markers == 7] = [255,140,0]
coin_img[markers == 8] = [255,215,0]
coin_img[markers == 9] = [218,165,32]
coin_img[markers == 10] = [0,100,0]
coin_img[markers == 11] = [0,255,0]
coin_img[markers == 12] = [50,205,50]
coin_img[markers == 13] = [0,250,154]
coin_img[markers == 14] = [46,139,87]
coin_img[markers == 15] = [32,178,170]
coin_img[markers == 16] = [47,79,79]
coin_img[markers == 17] = [0,255,255]
coin_img[markers == 18] = [0,191,255]
coin_img[markers == 19] = [135,206,250]
coin_img[markers == 20] = [0,0,128]
```

```
coin_img[markers == 21] = [0,0,255]
coin_img[markers == 22] = [138,43,226]
coin_img[markers == 23] = [75,0,130]

# save image
cv2.imwrite('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignment 2/
```

Out[55]:
```
True
```

In [56]:
```
np.unique(markers)
```

Out[56]:
```
array([-1,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23], dtype=int32)
```

In [57]:
```
##### Problem 3.2 #####
particle_img = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments
particle_img_grey = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assig
particle_img_grey_final = ~particle_img_grey

# thresholding
ret, thresh_img = cv2.threshold(particle_img_grey_final, 0, 255, cv2.THRESH_BINARY +

# noise removal using opening operation
kernel = np.ones((11,11), np.uint8)
open_img = cv2.morphologyEx(thresh_img, cv2.MORPH_OPEN, kernel, iterations = 1)

# sure background area
sure_bg = cv2.dilate(open_img, kernel, iterations = 1)

# sure foreground area
dist_transform = cv2.distanceTransform(open_img, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.15*dist_transform.max(), 255, 0)

# unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

# label markers
ret, markers = cv2.connectedComponents(sure_fg)

# add 1 to all labels so that sure background is 1
markers = markers + 1

# mark unknown region with 0
markers[unknown==255] = 0

# apply watershed transform
markers = cv2.watershed(particle_img, markers)
```

```
        # make each segment a different color
        particle_img[markers == -1] = [0,0,0]
        particle_img[markers == 1] = [211,211,211]
        particle_img[markers == 2] = [128,0,0]
        particle_img[markers == 3] = [220,20,60]
        particle_img[markers == 4] = [255,0,0]
        particle_img[markers == 5] = [255,99,71]
        particle_img[markers == 6] = [255,69,0]
        particle_img[markers == 7] = [255,140,0]
        particle_img[markers == 8] = [255,215,0]
        particle_img[markers == 9] = [218,165,32]
        particle_img[markers == 10] = [0,100,0]
        particle_img[markers == 11] = [0,255,0]
        particle_img[markers == 12] = [50,205,50]
        particle_img[markers == 13] = [0,250,154]
        particle_img[markers == 14] = [46,139,87]
        particle_img[markers == 15] = [32,178,170]
        particle_img[markers == 16] = [47,79,79]
        particle_img[markers == 17] = [0,255,255]
        particle_img[markers == 18] = [0,191,255]
        particle_img[markers == 19] = [135,206,250]
        particle_img[markers == 20] = [0,0,128]
        particle_img[markers == 21] = [0,0,255]
        particle_img[markers == 22] = [138,43,226]
        particle_img[markers == 23] = [75,0,130]
        particle_img[markers == 24] = [147,112,219]
        particle_img[markers == 25] = [128,0,128]
        particle_img[markers == 26] = [255,0,255]
        particle_img[markers == 27] = [199,21,133]
        particle_img[markers == 28] = [255,105,180]
        particle_img[markers == 29] = [219,112,147]
        particle_img[markers == 30] = [255,192,203]

        # save image
        cv2.imwrite('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignment 2/
```

Out[57]:
```
True
```

In [58]:
```
np.unique(markers)
```

Out[58]:
```
array([-1,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
      dtype=int32)
```

In [59]:
```
# generate overlay images for Problem 3
img_1 = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assig
img_2 = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assig
w1 = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignme
w2 = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignme
```

```
combine_1 = cv2.addWeighted(img_1, 0.6, w1, 0.4, 0)
combine_2 = cv2.addWeighted(img_2, 0.6, w2, 0.4, 0)

# save images
cv2.imwrite('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignment 2/
cv2.imwrite('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignment 2/
```

```
Out[59]: True
```