**Question 1**

**Question 2**

1. X = np.vstack((X\_circ, X\_line)) #All points

2.

3.

4. # Item 2: Total least squares line fitting with scicpy.optimize with RANSAC

5. import math

6. from scipy.optimize import minimize

7. N = X\_line.shape[0] # points

8. X\_ = X\_line

9.

10. def line\_equation\_from\_points(x1, y1, x2, y2):

11. # Calculate the direction vector (Δx, Δy)

12. delta\_x = x2 - x1

13. delta\_y = y2 - y1

14.

15. # Calculate the normalized vector (a, b)

16. magnitude = math.sqrt(delta\_x\*\*2 + delta\_y\*\*2)

17. a = delta\_y / magnitude

18. b = -delta\_x / magnitude

19.

20. # Calculate d

21. d = (a \* x1) + (b \* y1)

22.

23. # Return the line equation in the form ax + by = d

24. return a, b, d

25.

26. # RANSAC to fit a line

27. # returning the error

28. def line\_tls(x, indices):

29. a, b, d = x[0], x[1], x[2]

30. return np.sum(np.square(a\*X\_[indices,0] + b\*X\_[indices,1] - d))

31.

32.

33. # Constraint

34. # x = [a,b]

35. def g(x):

36. return x[0]\*\*2 + x[1]\*\*2 - 1

37.

38. # maintaining a\*\*2 + b\*\*2 = 1

39. cons = ({'type': 'eq', 'fun': g})

40.

41. # Computing the consensus (inliers)

42. def consensus\_line(X\_, x, t):

43. a, b, d = x[0], x[1], x[2]

44. error = np.absolute(a\*X\_[:,0] + b\*X\_[:,1] - d)

45. return error < t

46.

47.

48. t = 1. # Threshold value to determine data points that are fit well by model.

49. d = 0.84\*N # Number of close data points required to assert that a model fits well to data.

50. s = 2 # Minimum number of data points required to estimate model parameters.

51.

52. inliers\_line = [] # Indinces of the inliers

53. max\_iterations = 5000

54. iteration = 0

55. best\_model\_line = [] # Best model normal (a, b) and distance from origin d

56. best\_error = np.inf

57. best\_sample\_line = [] # Three-point sample leading to the best model computation

58. res\_only\_with\_sample = [] # Result (a, b, d) only using the best sample

59. best\_inliers\_line = [] # Inliers of the model computed form the best sample

60. x0 = np.array([1, 1, 0]) # Initial estimate a,b,d

61.

62. while (iteration < max\_iterations):

63. indices = np.random.randint(0, N, s) # A sample of 2 points selected at random

64.

65. #return many things but res.x contains a,b,d

66. res = minimize(fun = line\_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': False})

67.

68. #returns indexes of cordinates of inliners

69. inliers\_line = consensus\_line(X\_, res.x, t) # Computing the inliers

70.

71. # print('rex.x: ', res.x)

72. # print('Iteration = ', iteration, '. No. inliners = ', inliers\_line.sum()) # number of inliners

73.

74. if np.sum(inliers\_line) > d:

75. x0 = res.x

76. print('no of inliners = ',np.sum(inliers\_line), d)

77. # Computing the new model using the inliers

78. #return many things but res.x contains a,b,d

79. res = minimize(fun = line\_tls, args = inliers\_line, x0 = x0, tol= 1e-6, constraints=cons, options={'disp': False})

80.

81. #print(res.x, res.fun) # what is fun

82. if res.fun < best\_error:

83. # print('A better model found ... ', res.fun, best\_error)

84. best\_model\_line = res.x

85. best\_error = res.fun

86. best\_sample\_line = X\_[indices,:]

87. # res\_only\_with\_sample = x0

88. best\_inliers\_line = inliers\_line

89.

90. iteration += 1

91.

92.

93. print('Best line model', best\_model\_line)

94.

A graph of a circle with colored dots

Description automatically generated

Code for RANSAC circle

1. new\_xline = X\_line[~best\_inliers\_line]

2. new\_X = np.vstack((X\_circ, new\_xline)) #All points

3.

4. import math

5. import numpy as np

6. from scipy.optimize import minimize

7. N = new\_X.shape[0] # points

8. # X\_ = new\_X

9.

10. def circle\_tls(x, indices):

11. x0, y0, r = x[0], x[1], x[2]

12. datapoints = new\_X[indices]

13. squared\_distances = np.sum((datapoints - np.array([x0, y0]))\*\*2, axis=1)

14. error = np.abs(squared\_distances - r\*\*2)

15. return np.sum(error)

16.

17. def consensus\_circle(new\_X, x, t):

18. xc, yc, r = x[0], x[1], x[2]

19. point\_to\_center = np.sqrt(np.square(new\_X[:,0] - xc) + np.square(new\_X[:,1] - yc))

20. return (point\_to\_center < r+t) & (point\_to\_center > r-t)

21.

22. t = 1.0 # Threshold value to determine data points that are fit well by model.

23. # increase d to 0.8N

24. d = 0.85\*N # Number of close data points required to assert that a model fits well to data.

25. s = 3 # Minimum number of data points required to estimate model parameters.

26.

27. cons\_circle = ({'type': 'ineq', 'fun': lambda x: x[2] - t})

28.

29. inliers\_circle = [] # Indinces of the inliers

30. max\_iterations = 500

31. iteration = 0

32. best\_model\_circle = [] # Best model normal (a, b) and distance from origin d

33. best\_error = np.inf

34. best\_sample\_circle = [] # Three-point sample leading to the best model computation

35. res\_only\_with\_sample\_circle = [] # Result (xc, yc, r) only using the best sample

36. best\_inliers\_circle = [] # Inliers of the model computed form the best sample

37. x0 = np.array([1, 1, 1]) # Initial estimate xc, yc, r

38.

39. while iteration < max\_iterations:

40. indices = np.random.randint(0, N, s) # A sample of 3 points selected at random

41.

42. #return many things but res.x contains a,b,d

43. res = minimize(fun = circle\_tls, args = indices, x0 = x0, tol= 1e-6, constraints=cons\_circle, options={'disp': False})

44.

45. #returns indexes of cordinates of inliners

46. inliers\_circle = consensus\_circle(new\_X, res.x, t) # Computing the inliers

47.

48. # print('rex.x: ', res.x)

49. # print('Iteration = ', iteration, '. No. inliners = ', inliers\_circle.sum()) # number of inliners

50.

51. if np.sum(inliers\_circle) > d:

52. x0 = res.x

53. print('no of inliners for circle: ', np.sum(inliers\_circle), d)

54. # Computing the new model using the inliers

55. #return many things but res.x contains a,b,d

56. res = minimize(fun = circle\_tls, args = inliers\_circle, x0 = x0, tol= 1e-6, constraints=cons\_circle, options={'disp': False})

57.

58. # print(res.x, res.fun) # what is fun

59. if res.fun < best\_error:

60. # print('A better model found ... ', res.x, res.fun)

61. best\_model\_circle = res.x

62. best\_error = res.fun

63. best\_sample\_circle = new\_X[indices,:]

64. # res\_only\_with\_sample\_circle = x0

65. best\_inliers\_circle = inliers\_circle

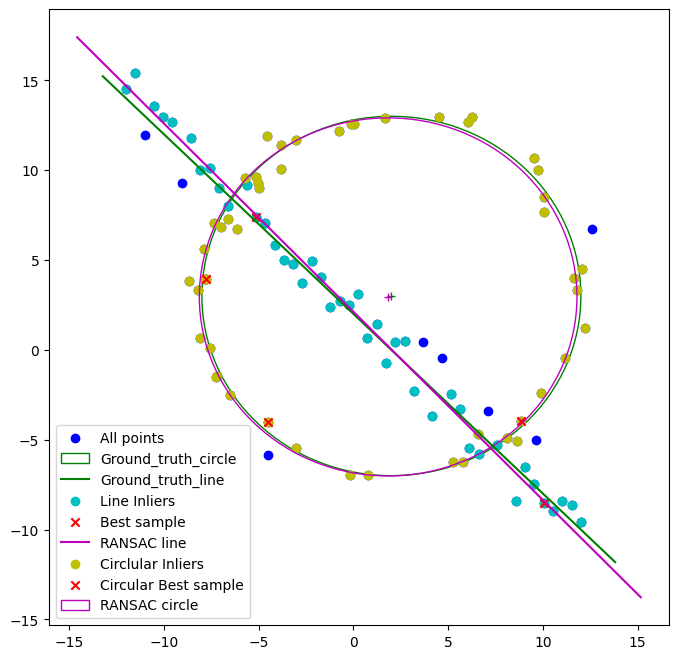
66.

67. iteration += 1

68.

69. print('best model found: ', best\_model\_circle, ', no of inliners = ', np.sum(best\_inliers\_circle))

70.



**Question 3**

1. 1.  import cv2 as cv

2. import numpy as np

3. import matplotlib.pyplot as plt

4.

5. clicked\_coordinates = []

6. click\_count = 0

7.

8. # Mouse callback function

9. def mouse\_callback(event, x, y, flags, param):

10. global clicked\_coordinates, click\_count

11.

12. # If left mouse button is clicked, capture the coordinates and draw a marker

13. if event == cv.EVENT\_LBUTTONDOWN:

14. clicked\_coordinates.append((x, y))

15. click\_count += 1

16. cv.circle(image, (x, y), 5, (0, 0, 255), -1) # Draw a red circle as a marker

17. cv.imshow("Image", image)

18.

19. # If four clicks are captured, stop capturing

20. if click\_count == 4:

21. cv.destroyAllWindows()

22.

23.

24. image = cv.imread('images/collage.jpg')

25. cv.imshow("Image", image)

26.

27. cv.setMouseCallback("Image", mouse\_callback)

28.

29. cv.waitKey(0)

30.

31. # Output the coordinates

32. print("Clicked Coordinates:")

33. for i, (x, y) in enumerate(clicked\_coordinates, start=1):

34. print(f"Point {i}: ({x}, {y})")

35.

36. # display the image with markers in Matplotlib

37. image\_rgb = cv.cvtColor(image, cv.COLOR\_BGR2RGB)

38. # Set the plot size to 15x15 inches

39. fig, ax = plt.subplots(figsize=(5, 5))

40.

41. ax.imshow(image\_rgb)

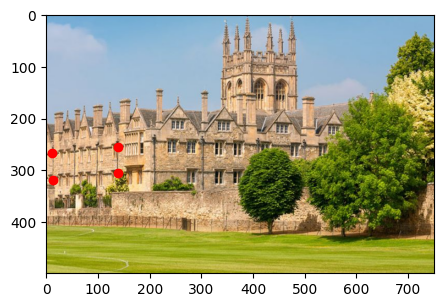
42. for (x, y) in clicked\_coordinates:

43. ax.plot(x, y, 'ro') # Draw a red marker

44.

45. plt.show()

46.



Code for Warping

1. import matplotlib.pyplot as plt

2.

3. img\_building = cv.imread('images/collage.jpg')

4. img = cv.cvtColor(img\_building, cv.COLOR\_BGR2RGB)

5. img\_flag = cv.imread('images/flag2.jpg')

6.

7. points\_building = np.array(clicked\_coordinates, dtype=np.float32)

8.

9. # should be in cliking order

10. point\_flag = np.array([[0, 0], [img\_flag.shape[1], 0], [img\_flag.shape[1], img\_flag.shape[0]], [0, img\_flag.shape[0]]], dtype=np.float32)

11.

12. # Calculate the homography matrix

13. homography\_matrix, \_ = cv.findHomography(point\_flag, points\_building)

14.

15. # Warp the flag image

16. flag\_warped = cv.warpPerspective(img\_flag, homography\_matrix, (img\_building.shape[1], img\_building.shape[0]))

17. warped\_flag=cv.cvtColor(flag\_warped, cv.COLOR\_BGR2RGB)

18.

19. # Adjust transparency

20. alpha = 0.5

21.

22. # # Create the composite image

23. composite\_image = cv.addWeighted(img\_building, 1, flag\_warped, alpha, 0, dst=img\_building)

24. composite=cv.cvtColor(composite\_image, cv.COLOR\_BGR2RGB)

25.

26. fig, ax = plt.subplots(1, 3, figsize=(15,15))

27.

28. ax[0].imshow(img)

29. ax[0].axis('off')

30. ax[0].legend('Image of the building')

31. ax[1].imshow(warped\_flag)

32. ax[1].axis('off')

33. ax[1].legend('Image of the building')

34. ax[2].imshow(composite)

35. ax[2].axis('off')

36. ax[2].legend('Image of the building')

37. plt.show()

38.

A flag on a black background

Description automatically generated

**Question 4**