In [ ]:

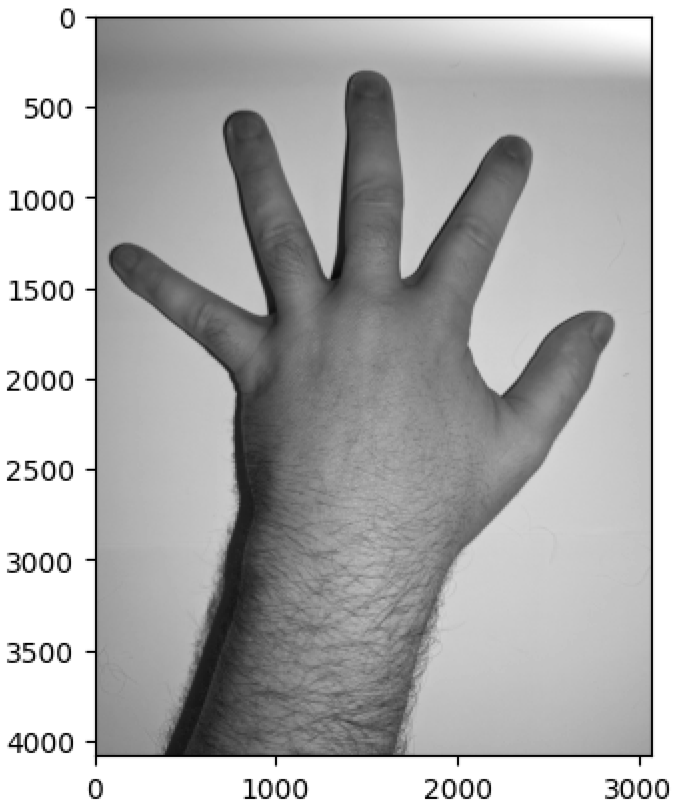
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

from skimage import io
from skimage.color import rgb2gray
import scipy.ndimage as ndimage

import math
```

In [ ]:

```python
handNumber = 5
plainHands = []
gtHands = []
grayHands = []
for i in range(1, handNumber + 1):
    plainIMG =  io.imread(f'hand/handSecondSet{i}.jpg')
    plainHands.append(plainIMG)

    gray_img= rgb2gray(plainIMG.copy())
    grayHands.append(gray_img)
    gtHands.append(plt.imread(f'handGT/handSecondSetGT{i}.png'))

plt.imshow(grayHands[0], cmap=plt.cm.gray)
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x200ab4239d0>
```

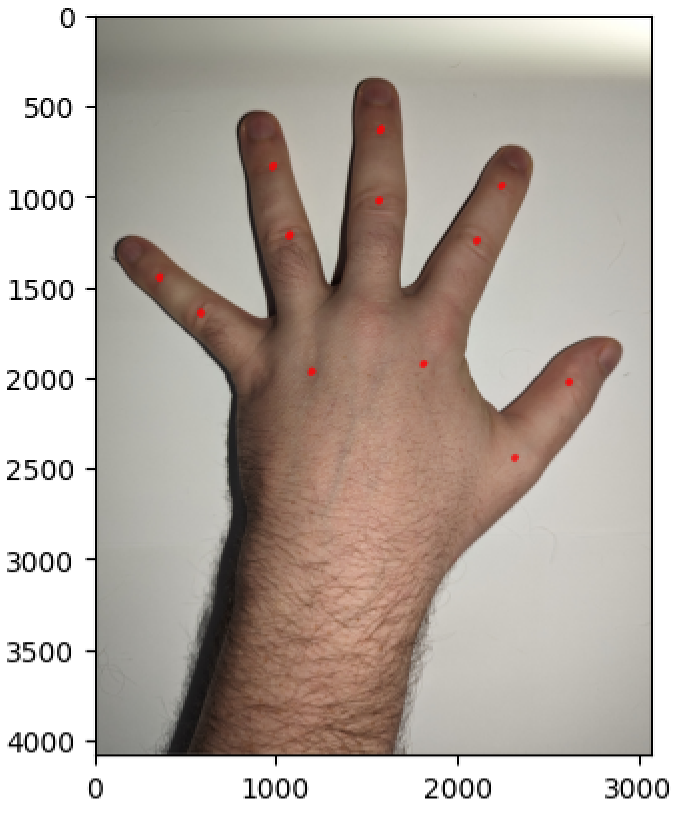


In [ ]:

```python
fPoints = []
handIdx = 4
```

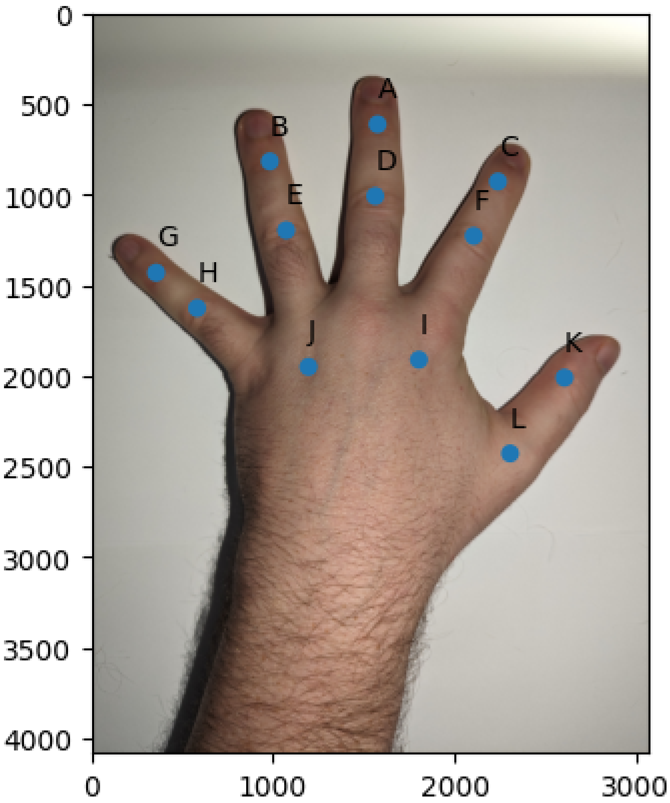
```
for x in range(gtHands[handIdx].shape[0]):
    for y in range(gtHands[handIdx].shape[1]):
        if(gtHands[handIdx][x][y][0] >= .8 and gtHands[handIdx][x][y][1] < .4 and g

            addPoint = True
            for point in fPoints:
                pfx = point[0]
                pfy = point[1]

                summation = ((x - pfx) ** 2) + ((y - pfy) ** 2)
                distance = math.sqrt(summation)

                if distance < 100:
                    addPoint = False
                    break

            if addPoint:
                fPoints.append((x, y))

plt.imshow(gtHands[handIdx])
```

```
Out[ ]: <matplotlib.image.AxesImage at 0x200ab4fb350>
```



```
In [ ]: handCopy = plainHands[handIdx].copy()

        fig, ax = plt.subplots(1)
        ax.set_aspect('equal')

        ax.imshow(handCopy)
        variable = 65
```

```
pointDict = {}
for point in fPoints:
    circ = Circle((point[1], point[0]), 50)
    ax.add_patch(circ)
    plt.text(point[1], point[0] - 150, f'{chr(variable)}')
    pointDict[chr(variable)] = (point[1], point[0])
    variable += 1

plt.show()
```



In [ ]:

```
class Point():
    def __init__(self, point: tuple) -> None:
        self.x = point[0]
        self.y = point[1]

    def toCartesian(self):
        return (self.x, self.y)

class Feature():
    def __init__(self, point1, point2, color = 'r', isFinger= True) -> None:

        normalVector = np.array([point1[0] - point2[0], point1[1] - point2[1]])

        magnitude = math.sqrt(normalVector[0] ** 2 + normalVector[1] ** 2)

        self.p1 = Point(point1)
        self.p2 = Point(point2)

        self.unitNVector = normalVector / magnitude
        self.unitDVector = np.array([self.unitNVector[1] * (-1), self.unitNVector[0
        self.isFinger = isFinger
        self.color = color

    def slope(self):
        return (self.p1.y - self.p2.y) / (self.p1.x - self.p2.x)

    def orthoganalSlope(self):
        return (-1 / self.slope())
```

In [ ]:

```
class HandFeatures():
    def __init__(self, pDict: dict) -> None:
        self.features = {}
        self.features['F1'] = Feature(pDict['I'], pDict['J'], 'y', False)
        self.features['F2'] = Feature(pDict['K'], pDict['L'], 'r')
        self.features['F3'] = Feature(pDict['C'], pDict['F'], 'b')
        self.features['F4'] = Feature(pDict['A'], pDict['D'], 'g')
        self.features['F5'] = Feature(pDict['B'], pDict['E'], 'black')
        self.features['F6'] = Feature(pDict['G'], pDict['H'], 'r')

    def getFeatures(self) -> dict:
        return self.features

    def getFeature(self, idx) -> Feature:
        return self.features[idx]
```

In [ ]:

```
features = HandFeatures(pointDict)
```

In [ ]:

```
print(f'normal vector: {features.getFeature("F3").unitNVector}')
print(f'Direction vector: {features.getFeature("F3").unitDVector}')
```

```
normal vector: [ 0.4078321  -0.91305694]
Direction vector: [0.91305694 0.4078321 ]
```

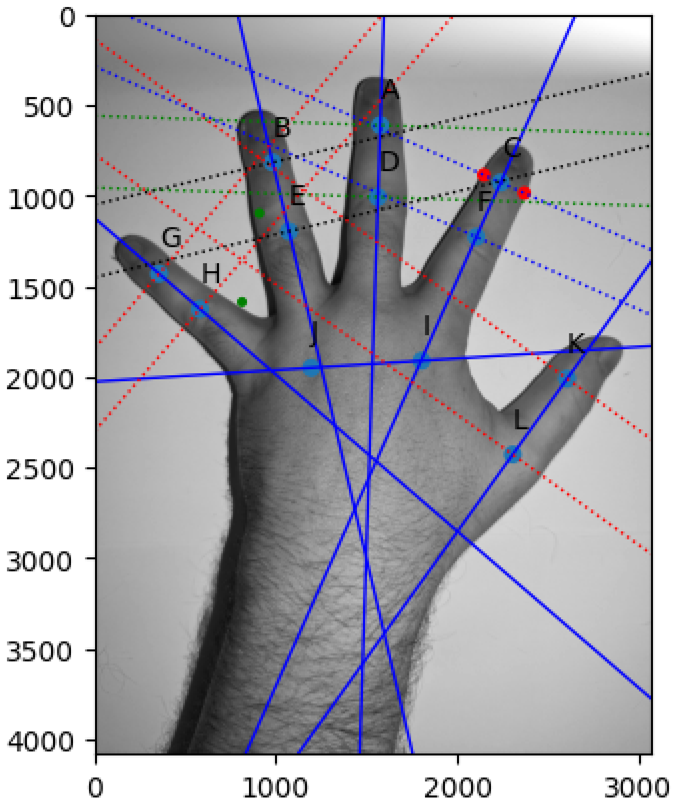
In [ ]:

```
def drawLine(point1: tuple, point2: tuple, ax):
    ax.axline(point1, point2, color='b', linewidth=1)
```

In [ ]:

```
handCopy = grayHands[handIdx].copy()

fig, ax = plt.subplots(1)
ax.set_aspect('equal')

ax.imshow(handCopy, cmap= plt.cm.gray)
variable = 65
pointDict = {}
for point in fPoints:
    circ = Circle((point[1], point[0]), 50)
    ax.add_patch(circ)
    plt.text(point[1], point[0] - 150, f'{chr(variable)}')
    pointDict[chr(variable)] = (point[1], point[0])
    variable += 1

# start of scanning
circ = Circle((909, 1096), 20, color = 'g')
ax.add_patch(circ)
```

```
circ = Circle((813, 1586), 20, color = 'g')
ax.add_patch(circ)

for key in features.features:
    feature = features.features[key]
    ax.axline(feature.p1.toCartesian(), feature.p2.toCartesian(), color='b', linewi

# endpoints (2147, 885) (2371, 985)
circ = Circle((2147, 885), 30, color = 'r')
ax.add_patch(circ)
circ = Circle((2371, 985), 30, color = 'r')
ax.add_patch(circ)


ax.imshow(handCopy, cmap= plt.cm.gray)
# drawLine(pointDict['C'], pointDict['D'], ax)
# drawLine(pointDict['A'], pointDict['B'], ax)
# drawLine(pointDict['F'], pointDict['G'], ax)
# drawLine(pointDict['H'], pointDict['I'], ax)
# drawLine(pointDict['K'], pointDict['L'], ax)
# drawLine(pointDict['E'], pointDict['J'], ax)


for key in features.getFeatures().keys():
    if key != 'F1':
        feature = features.getFeature(key)

        ax.axline(feature.p1.toCartesian(), slope= feature.orthoganalSlope(), ls= '
        ax.axline(feature.p2.toCartesian(), slope= feature.orthoganalSlope(), ls= '

plt.show()
```



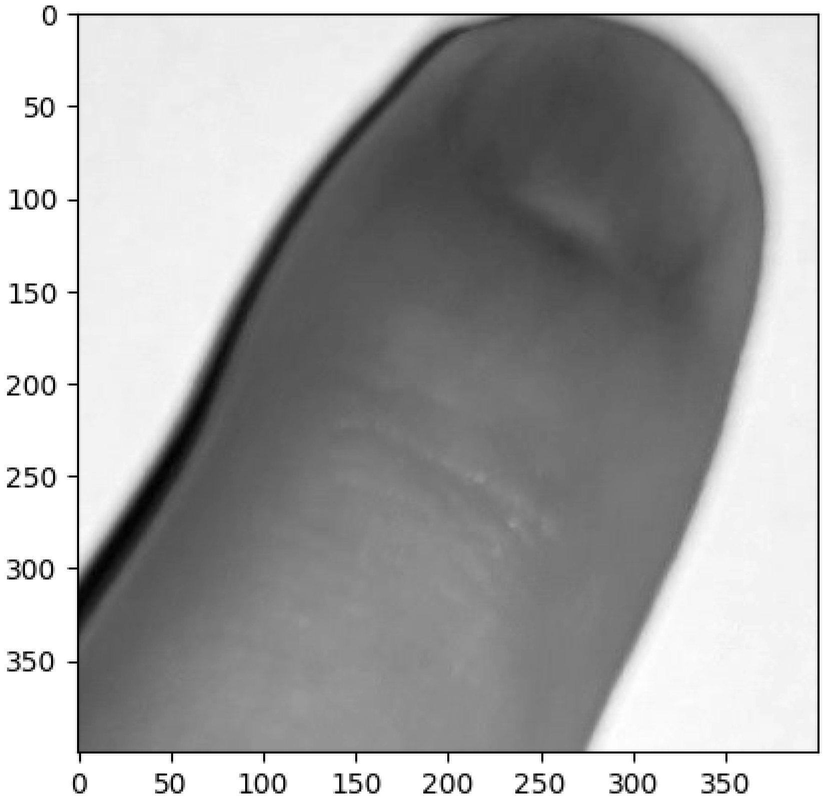
In [ ]:

```
def getIntensityGraph(hand_gray_img, point: Point, unitDVector):
    scale = 200
    step = 0
    newPoint = [point.x, point.y] + (unitDVector * step)

    currentPoint = Point((newPoint[0], newPoint[1]))
    scanningArray = np.array(
        hand_gray_img[
            round(currentPoint.y) - scale : round(currentPoint.y) + scale,
            round(currentPoint.x) - scale : round(currentPoint.x) + scale])

    maxValue = []
    minValue = []

    minValue.append(np.min(scanningArray))
    print(f'Minvalue: {minValue[0]}')
    maxValue.append(np.max(scanningArray))
    print(f'Maxvalue: {maxValue[0]}')



    print(f'')

    return scanningArray

currentFeature = features.getFeature('F3')
array = getIntensityGraph(grayHands[handIdx], currentFeature.p1, currentFeature.uni
smoothArray = ndimage.gaussian_filter(array, sigma=0)
```

```
plt.imshow(smoothArray, cmap=plt.cm.gray)
```

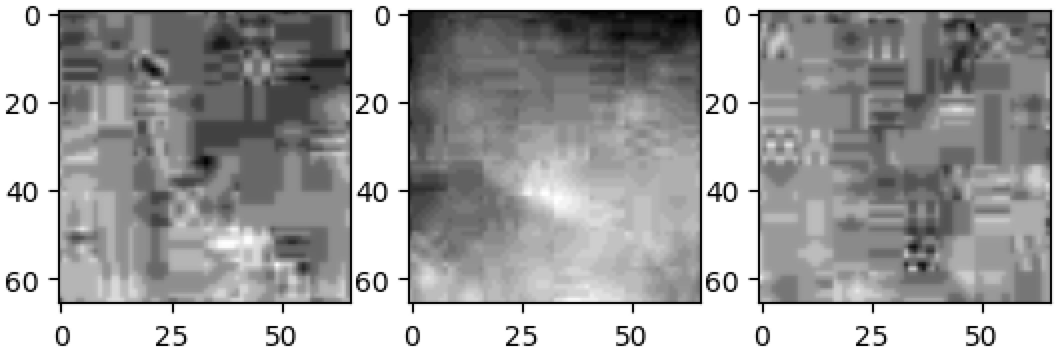
```
Minvalue: 0.03605215686274509
Maxvalue: 0.777556862745098
```

```
Out[ ]: <matplotlib.image.AxesImage at 0x200ab5c1390>
```



```
In [ ]: def computeAllWindows(hand_gray_img, point: Point, unitDVector):
            scale = 33
            start = -350
            end = 350
            sigma = .5
            currentPoint = [point.x, point.y] + np.multiply(unitDVector, start)

            lastPoint = [point.x, point.y] + (unitDVector * end)
            windStart = np.array(
                    hand_gray_img[
                        round(currentPoint[1]) - scale : round(currentPoint[1]) + scale,
                        round(currentPoint[0]) - scale : round(currentPoint[0]) + scale])

            windEnd = np.array(
                    hand_gray_img[
                        round(lastPoint[1]) - scale : round(lastPoint[1]) + scale,
                        round(lastPoint[0]) - scale : round(lastPoint[0]) + scale])

            fig, (ax1, ax2, ax3) = plt.subplots(1, 3)

            ax1.imshow(ndimage.gaussian_filter(windStart, sigma=sigma), cmap=plt.cm.gray)
            ax2.imshow(hand_gray_img[point.y - scale: point.y + scale, point.x - scale : po
            ax3.imshow(ndimage.gaussian_filter(windEnd, sigma=sigma), cmap=plt.cm.gray)
```

```
    plt.show()

    maxValues = []
    minValues = []
    maxIDXs = []
    minIDXs = []
    difference = []

    step = start

    lastMaxVal = 0
    lastMinVal = 0

    beginFinger: tuple = (0, 0)
    endFinger: tuple = (0, 0)

    startPoint = np.round(currentPoint)
    endPoint = np.round([point.x, point.y] + np.multiply(unitDVector, end))
    print(startPoint)
    print(endPoint)

    searchingForStart = True
    while(step < end):
        # print(f'step {step}')
        currentPoint = [point.x, point.y] + np.multiply(unitDVector, step)
        windowArray = np.array(
            hand_gray_img[
                round(currentPoint[1]) - scale : round(currentPoint[1]) + scale,
                round(currentPoint[0]) - scale : round(currentPoint[0]) + scale])

        smoothWindow = ndimage.gaussian_filter(windowArray, sigma=sigma)


        maxVal = np.max(smoothWindow)
        minVal = np.min(smoothWindow)

        maxIDX = np.argmax(smoothWindow)
        minIDX = np.argmin(smoothWindow)

        maxValues.append(maxVal)
        minValues.append(minVal)

        maxIDXs.append(maxIDX)
        minIDXs.append(minIDX)

        difference.append(maxVal - minVal)

        if step == start:
            lastMaxVal = maxVal
            lastMinVal = minVal

        else:
            # print(f'maxIDX: {maxIDX} - min: {minIDX}')
            # print(f'maxVal: {format(maxVal, ".5f")} - minVal: {format(minVal, ".5
            # print(f'lastMax: {format(lastMaxVal, ".5f")} - lastMin: {format(lastM
            # print('\n')
            if minIDX > maxIDX and searchingForStart:
                # print(f'First Gate: {maxVal - minVal} >= {lastMaxVal - lastMinVal
                if (maxVal - minVal) > (lastMaxVal - lastMinVal):
                    # print((round(currentPoint[0]), round(currentPoint[1])))
                    beginFinger = (round(currentPoint[0]), round(currentPoint[1]))
                else:
                    searchingForStart = False


            if not searchingForStart:
                # print(f'Second Gate: {maxVal - minVal} >= {lastMaxVal - lastMinVa
                currentDiff = maxVal - minVal
                lastDiff = lastMaxVal - lastMinVal
                if currentDiff > lastDiff and not math.isclose(lastDiff, currentDif
                    endFinger = (round(currentPoint[0]), round(currentPoint[1]))

            lastMaxVal = maxVal
            lastMinVal = minVal

        step += 1



    print(beginFinger, endFinger)

    # print(f'maxValue: {maxValue}')
    # print(f'minValue: {minValue}')
    # print(f'maxiDX: {maxIDX}')
    # print(f'minIDX: {minIDX}')

    return maxValues, minValues, maxIDXs, minIDXs, difference




maxValues, minValues, maxIDXs, minIDXs, difference = computeAllWindows(grayHands[ha

# plt.plot(grayscaleProfile)
# plt.show()
```



```
[1920.  784.]
[2560. 1070.]
(2088, 859) (2549, 1065)
```

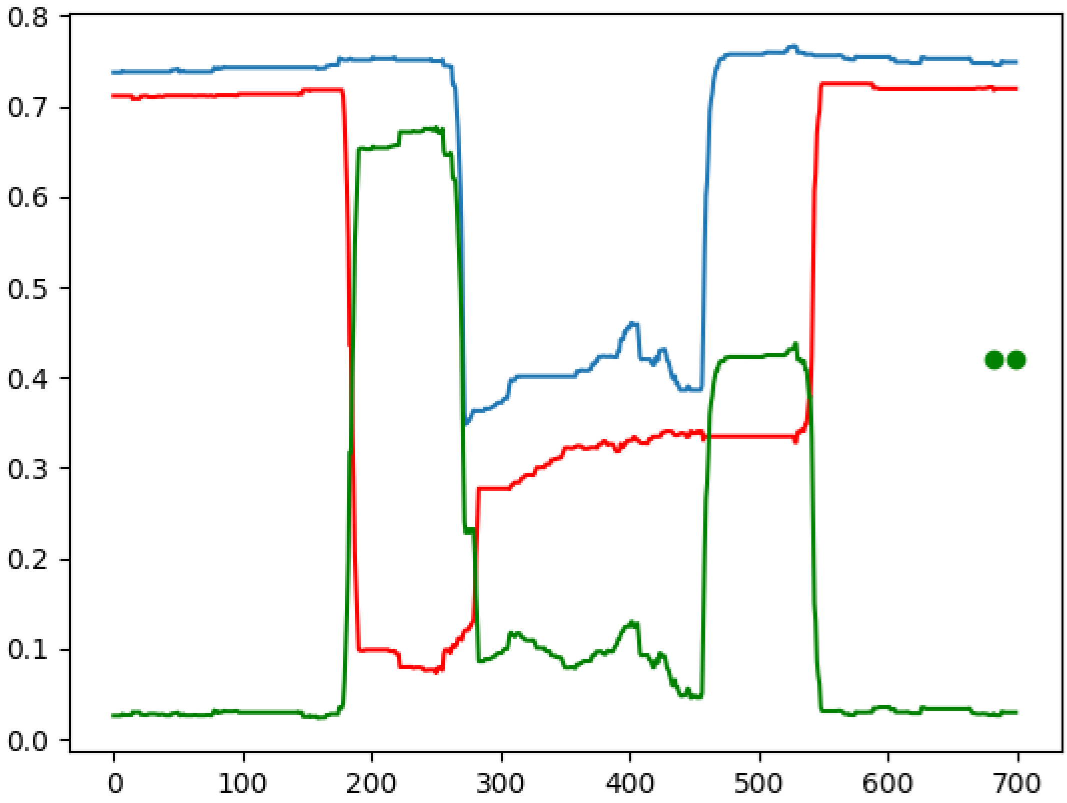
In [ ]:

```
start = 0
end = 0


combined = []

searchingForStart = True
idifference = []
for k in range(1, len(maxValues) - 1):
    combined.append(maxValues[k] - maxValues[k])

    currentMax = maxValues[k] - minValues[k]

    for i in range(0, k):
        oldDifference = (maxValues[i] - minValues[i])

        if minIDXs[k] > maxIDXs[k] and not math.isclose(currentMax, oldDifference,
            if currentMax > oldDifference:
                start = k


        elif minIDXs[k] < maxIDXs[k] and not math.isclose(currentMax, oldDifference
            if currentMax > oldDifference :
                end = k


            # print(f'max_K: {maxValues[k]} - min_K: {minValues[k]} max_i: {maxValu
            # print(f'K: {maxValues[k] - minValues[k]} i: {maxValues[k - 1] - minVa
            # print(f'index: {k}')




print(f'Start: {start}')
print(f'End: {end}')
print(maxIDXs[len(maxIDXs) - 1])

print(f"Start Index: {np.unravel_index(start, (100, 100))}")
print(f"End Index: {np.unravel_index(end, (100, 100))}")
```

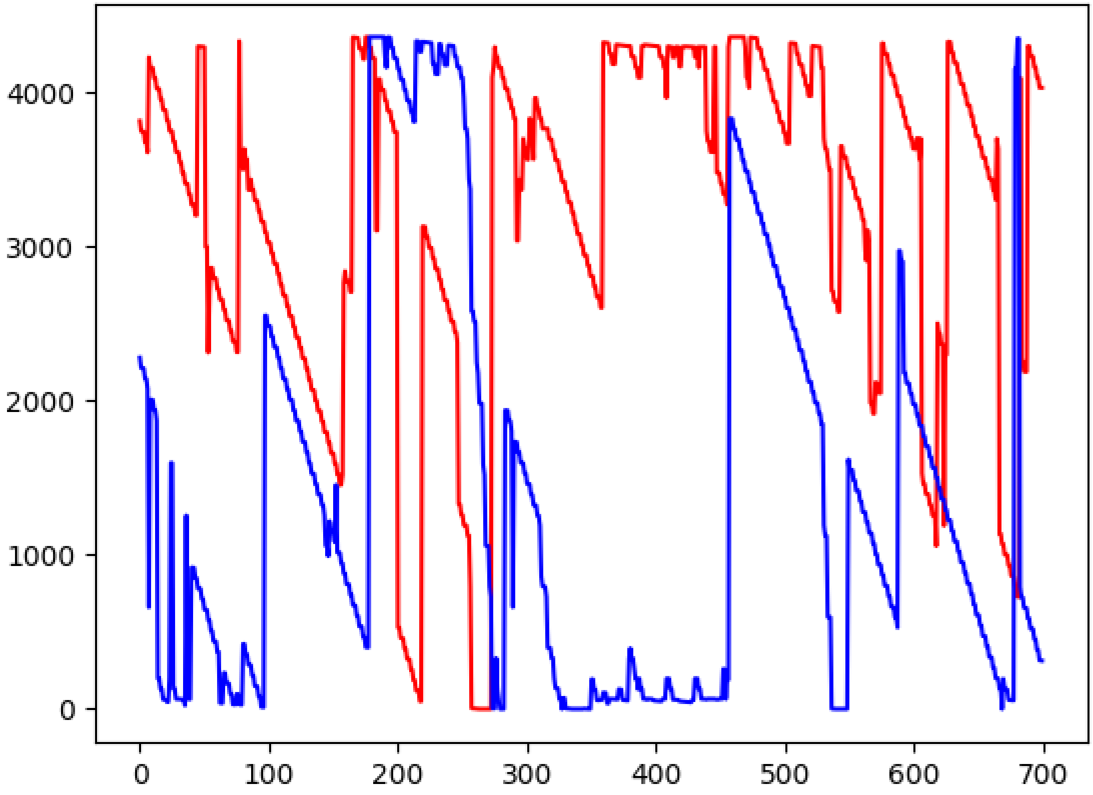
```
Start: 681
End: 698
4026
Start Index: (6, 81)
End Index: (6, 98)
```

In [ ]:

```
plt.plot(maxValues)
plt.plot(minValues, color = 'r')
# plt.plot(difference[100:-1], color='orange')
plt.plot(difference, color='g')
plt.plot([start], 0.42, 'go')
plt.plot([end], 0.42, 'go')
# plt.plot(icombined, color='black')

plt.show()
```



```
In [ ]: plt.plot(maxIDXs, color ='red')
        plt.plot(minIDXs, color = 'blue')
        plt.show()
```



In [ ]:

```
def clusterFeatures(featurePoints):
    variable = 65
    pointDict = {}
    for point in featurePoints:
        pointDict[chr(variable)] = (point[1], point[0])
        variable += 1

    features = HandFeatures(pointDict)

    return features


def loadHandImages(limit = 5):
    plainHands = []
    gtHands = []
    grayHands = []
    for i in range(1, limit + 1):
        plainIMG =  io.imread(f'hand/handSecondSet{i}.jpg')
        plainHands.append(plainIMG)

        gray_img= rgb2gray(plainIMG.copy())
        grayHands.append(gray_img)
        gtHands.append(plt.imread(f'handGT/handSecondSetGT{i}.png'))

    return plainHands, gtHands, grayHands



def getFeaturePointsFromGT(gt_image):
    featurePoints = []
    width = gt_image.shape[0]
    height = gt_image.shape[1]

    for x in range(width):
        for y in range(height):
            # looks for the reddest pixel, ground truthing was done with a red pen
            if(gt_image[x][y][0] >= .8 and gt_image[x][y][1] < .4 and gt_image[x][y

                addPoint = True
                for point in featurePoints:
                    pfx = point[0]
                    pfy = point[1]

                    summation = ((x - pfx) ** 2) + ((y - pfy) ** 2)
                    distance = math.sqrt(summation)

                    if distance < 100:
                        addPoint = False
                        break

                if addPoint:
                    featurePoints.append((x, y))

    return featurePoints
```

```python
def plotEachFeature(plain_image, gray_image, features):
    fig, axis = plt.subplots(2, 4)

    # axis[1][0].set_xticks([])
    # axis[1][0].set_yticks([])
    axis[1][0].set_title('Original')
    axis[1][0].imshow(plain_image)
    drawFeatureLines(gray_image, features, 'F6', axis[0][0])
    drawFeatureLines(gray_image, features, 'F5', axis[0][1])
    drawFeatureLines(gray_image, features, 'F4', axis[0][2])
    drawFeatureLines(gray_image, features, 'F3', axis[0][3])
    drawFeatureLines(gray_image, features, 'F2', axis[1][3])
    drawFeatureLines(gray_image, features, 'F1', axis[1][2])

    plt.show()

def drawFeatureLines(gray_image, features, featureLabel, ax):
    feature = features.features[featureLabel]
    ax.set_title(f'{featureLabel}')
    ax.set_xticks([])
    ax.set_yticks([])
    ax.imshow(gray_image, cmap=plt.cm.gray)
    ax.axline(feature.p1.toCartesian(), feature.p2.toCartesian(), color=feature.col
    ax.axline(feature.p1.toCartesian(), slope= feature.orthoganalSlope(), ls= ':',
    ax.axline(feature.p2.toCartesian(), slope= feature.orthoganalSlope(), ls= ':',
```

In [ ]:

```python
def plotValueGraph(maxValues, minValues, maxIDXs, minIDXs, difference, point1, poin
    fig, axis = plt.subplots(2, 1)
    fig.tight_layout(pad=0.5)

    axis[0].set_title("Values Max vs Min")
    axis[0].plot(maxValues, color = 'r')
    axis[0].plot(minValues, color = 'b')
    # plt.plot(difference[100:-1], color='orange')
    axis[0].plot(difference, color='g')
    axis[0].plot([point1], 0.42, 'mx', markersize= 10, markeredgewidth = 4, label='
    axis[0].plot([point2], 0.42, 'mx', markersize= 10, markeredgewidth = 4, label =

    # plt.plot(icombined, color='black')

    axis[1].set_title('Index High vs Low')
    axis[1].plot(maxIDXs, color ='red', label = 'Max indexs')
    axis[1].plot(minIDXs, color = 'blue', label = 'Min indexes')
    axis[1].legend()
    plt.show()
```

In [ ]:

```python
def showFeatures(fig, gray_img, featurePoints, x_axis, y_axis):
    axes = fig.add_axes([x_axis, y_axis, 1, 1])
    axes.imshow(gray_img, cmap=plt.cm.gray)

    for key in featurePoints:
        if key == 'F1':
            value = featurePoints[key]

            x_values = [value[0][0], value[1][0]]
            y_values = [value[1][1], value[1][1]]

            axes.plot(x_values , y_values, 'r--', linewidth = 2)
            axes.plot(value[0][0], value[0][1], 'bo', markersize=2)
            axes.plot(value[1][0], value[1][1], 'bo', markersize=2)
        else:

            value = featurePoints[key]

            x_values1 = [value[0][0][0], value[0][1][0]]
            y_values1 = [value[0][0][1], value[0][1][1]]

            axes.plot(x_values1 , y_values1, 'r--', linewidth = 2)
            axes.plot(value[0][0][0], value[0][0][1], 'bo', markersize=2)
            axes.plot(value[0][1][0], value[0][1][1], 'bo', markersize=2)

            x_values2 = [value[1][0][0], value[1][1][0]]
            y_values2 = [value[1][0][1], value[1][1][1]]

            axes.plot(x_values2 , y_values2, 'r--', linewidth = 2)
            axes.plot(value[1][0][0], value[1][0][1], 'bo', markersize=2)
            axes.plot(value[1][1][0], value[1][1][1], 'bo', markersize=2)
```

In [ ]:

```
def scanFeatureR2L(hand_gray_img, point: Point, unitDVector, start=-250, end=250, s
    currentPoint = [point.x, point.y] + np.multiply(unitDVector, start)

    maxValues = []
    minValues = []
    maxIDXs = []
    minIDXs = []
    difference = []

    step = end

    maxStartDif = 0.0
    maxEndDif = 0.0
    beginFinger: tuple = (0, 0)
    endFinger: tuple = (0, 0)

    fingerStart = 0
    fingerEnd = 0
    while(step > start):
        # print(f'step {step}')
        currentPoint = [point.x, point.y] + np.multiply(unitDVector, step)
        windowArray = np.array(
            hand_gray_img[
                round(currentPoint[1]) - scale : round(currentPoint[1]) + scale,
                round(currentPoint[0]) - scale : round(currentPoint[0]) + scale])

        smoothWindow = ndimage.gaussian_filter(windowArray, sigma=smoothing)
        maxVal = np.max(smoothWindow)
        minVal = np.min(smoothWindow)

        maxIDX = np.argmax(smoothWindow)
        minIDX = np.argmin(smoothWindow)

        maxValues.append(maxVal)
        minValues.append(minVal)

        maxIDXs.append(maxIDX)
        minIDXs.append(minIDX)

        difference.append(maxVal - minVal)

        currentDifference = maxVal - minVal

        if minIDX - maxIDX > leftThreshold:
            if currentDifference > maxStartDif:
                maxStartDif = currentDifference
                fingerStart = step
                beginFinger = (round(currentPoint[0]), round(currentPoint[1]))



        if maxIDX - minIDX > rightThreshold:
            if currentDifference > maxEndDif:
                maxEndDif = currentDifference
                fingerEnd = step
                endFinger = (round(currentPoint[0]), round(currentPoint[1]))


        step -= 1


    # p1 = abs(fingerStart - end)
    # p2 = abs(fingerEnd - end)
    # plotValueGraph(maxValues, minValues, maxIDXs, minIDXs, difference, p1, p2)




    return beginFinger, endFinger
```

In [ ]:

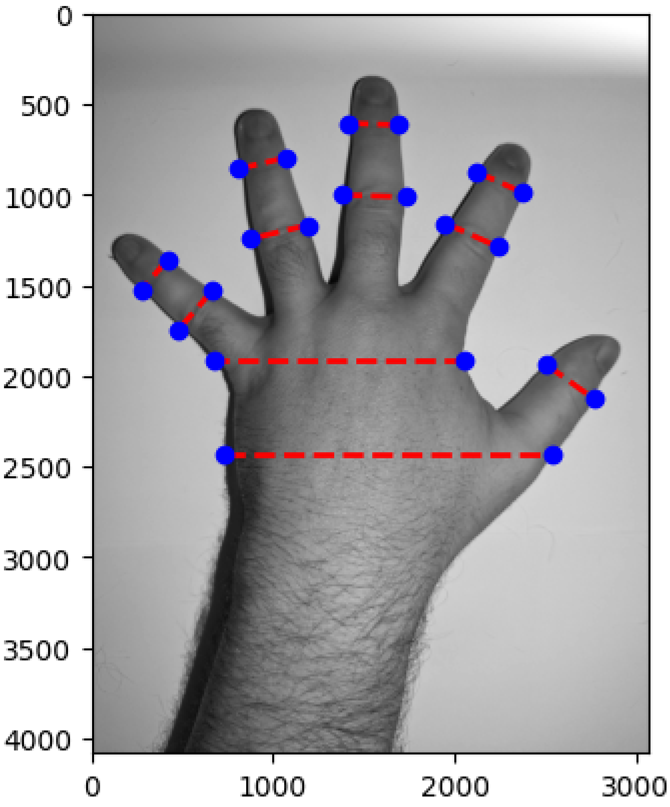
```
def scanFeatureL2R(hand_gray_img, point: Point, unitDVector, start=-250, end=250, s
    currentPoint = [point.x, point.y] + np.multiply(unitDVector, start)

    maxValues = []
    minValues = []
    maxIDXs = []
    minIDXs = []
    difference = []

    step = start

    maxStartDif = 0.0
    maxEndDif = 0.0
    beginFinger: tuple = (0, 0)
    endFinger: tuple = (0, 0)

    fingerStart = 0
    fingerEnd = 0
    while(step < end):
        # print(f'step {step}')
        currentPoint = [point.x, point.y] + np.multiply(unitDVector, step)
        windowArray = np.array(
            hand_gray_img[
                round(currentPoint[1]) - scale : round(currentPoint[1]) + scale,
                round(currentPoint[0]) - scale : round(currentPoint[0]) + scale])

        smoothWindow = ndimage.gaussian_filter(windowArray, sigma=smoothing)

        maxVal = np.max(smoothWindow)
        minVal = np.min(smoothWindow)

        maxIDX = np.argmax(smoothWindow)
        minIDX = np.argmin(smoothWindow)

        maxValues.append(maxVal)
        minValues.append(minVal)

        maxIDXs.append(maxIDX)
        minIDXs.append(minIDX)

        difference.append(maxVal - minVal)

        currentDifference = maxVal - minVal

        if minIDX - maxIDX > leftThreshold:
            if currentDifference > maxStartDif:
                maxStartDif = currentDifference
                fingerStart = step
                beginFinger = (round(currentPoint[0]), round(currentPoint[1]))



        if maxIDX - minIDX > rightThreshold:
            if currentDifference > maxEndDif:
                maxEndDif = currentDifference
                fingerEnd = step
                endFinger = (round(currentPoint[0]), round(currentPoint[1]))



        step += 1

    # p1 = abs(fingerStart - end)
    # p2 = abs(fingerEnd - end)
    # plotValueGraph(maxValues, minValues, maxIDXs, minIDXs, difference, p1, p2)


    return beginFinger, endFinger
```

In [ ]:

```
def scanAllFeatures(features, grayCopy):
    featureLabels = ['F1', 'F2', 'F3', 'F4', 'F5', 'F6']

    featurePoints = {}
    featureDistances = []


    for label in featureLabels:
        currentFeature: Feature = features.getFeature(label)

        if label == 'F1':
            horizontalVector = [1, 0]
            scanSize = 44

            begin_hand, end_hand = scanFeatureR2L(
                grayCopy,
                currentFeature.p1,
                horizontalVector,
                (-1) * (currentFeature.p1.x - scanSize - scanSize),
                400,
                scanSize, 13)

            distance = math.sqrt((begin_hand[0] - end_hand[0]) ** 2 + (begin_hand[1

            featureDistances.append(distance)

            featurePoints[label] = (begin_hand, end_hand)

        elif label == 'F2':
            horizontalVector = [1, 0]
            scanSize = 33
            # plot only a horizontal line from the second point

            begin_finger_pt1, end_finger_pt1 = scanFeatureR2L(grayCopy, currentFeat


            begin_finger_pt2, end_finger_pt2 = scanFeatureL2R(
                grayCopy,
                currentFeature.p2,
                horizontalVector,
                (-1) * (currentFeature.p2.x - scanSize - 400),
                400,
                scanSize,
                13,
                1300,
                1500)

            distance1 = math.sqrt((begin_finger_pt1[0] - end_finger_pt1[0]) ** 2 +
            distance2 = math.sqrt((begin_finger_pt2[0] - end_finger_pt2[0]) ** 2 +

            featureDistances.append(distance1)
            featureDistances.append(distance2)


            featurePoints[label] = [(begin_finger_pt1, end_finger_pt1), (begin_fing
        else:
            begin_finger_pt1, end_finger_pt1 = scanFeatureR2L(grayCopy, currentFeat
            begin_finger_pt2, end_finger_pt2 = scanFeatureR2L(grayCopy, currentFeat

            distance1 = math.sqrt((begin_finger_pt1[0] - end_finger_pt1[0]) ** 2 +
            distance2 = math.sqrt((begin_finger_pt2[0] - end_finger_pt2[0]) ** 2 +

            featureDistances.append(distance1)
            featureDistances.append(distance2)

            featurePoints[label] = [(begin_finger_pt1, end_finger_pt1), (begin_fing

    return featureDistances, featurePoints
```

In [ ]:

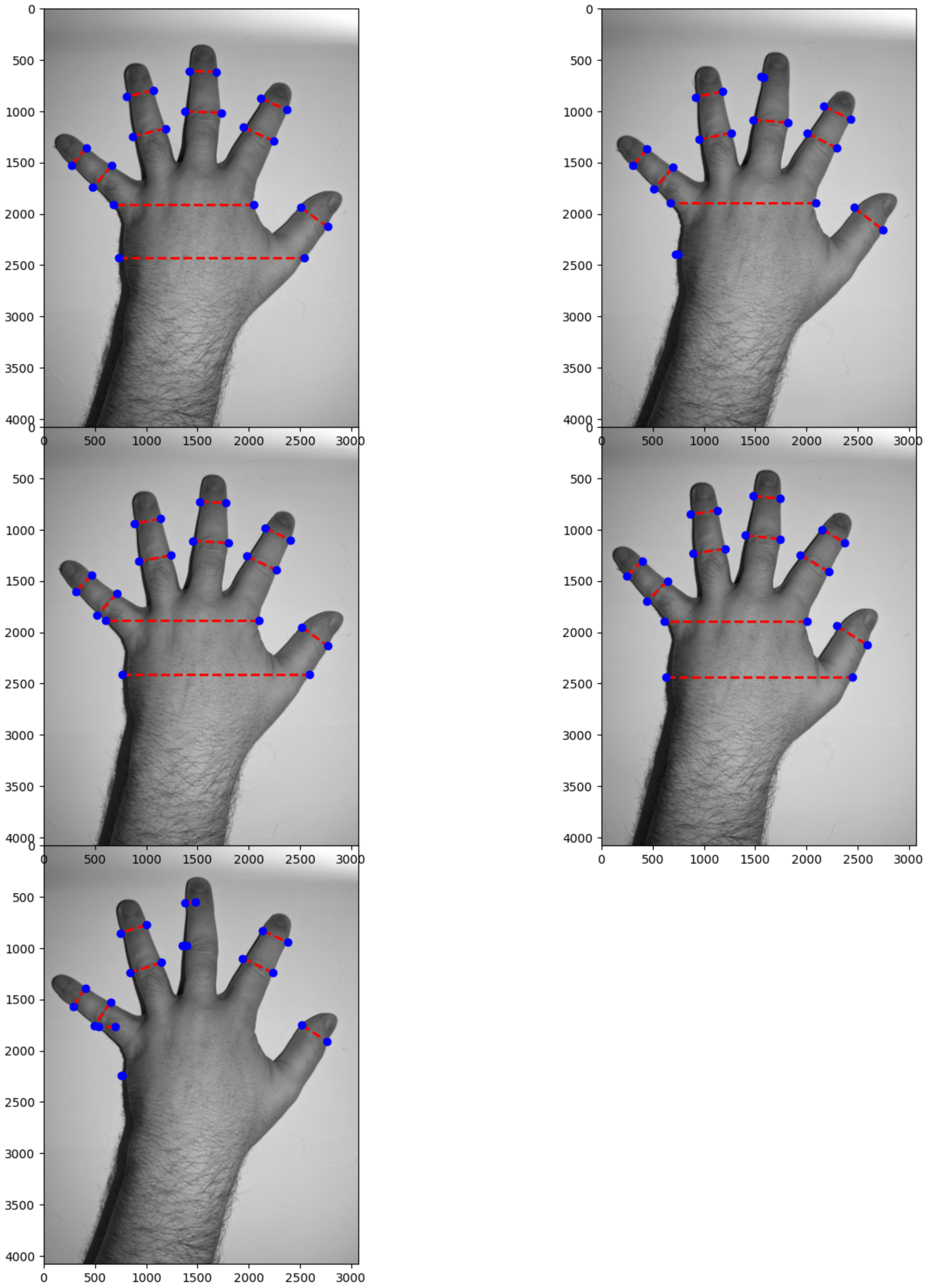
```
# plainHands, gtHands, grayHands = loadHandImages()
handIdx = 4

# featurePoints = getFeaturePointsFromGT(gtHands[handIdx])
# features = clusterFeatures(featurePoints)
grayCopy = grayHands[handIdx].copy()

# plotEachFeature(plainHands[handIdx], grayCopy, features)
featureDistances, featurePoints = scanAllFeatures(features, grayCopy)
showFeatures(gray_img, featurePoints)
```



In [ ]:

In [ ]:

```
# plainHands, gtHands, grayHands = loadHandImages()
allFeatures = []

fig = plt.figure()

x_axis = 0
y_axis = 0
for img_index in range(len(grayHands)):
    featurePoints = getFeaturePointsFromGT(gtHands[img_index])
    features = clusterFeatures(featurePoints)
    grayCopy = grayHands[img_index].copy()

    # plotEachFeature(plainHands[handIdx], grayCopy, features)
    featureDistances, featureStartToEndPoints = scanAllFeatures(features, grayCopy)
    if x_axis > 0:
        showFeatures(fig, grayCopy, featureStartToEndPoints, x_axis, y_axis)
        x_axis -= 1
    else:
        showFeatures(fig, grayCopy, featureStartToEndPoints, x_axis, y_axis)
        x_axis += 1
        y_axis += 1

    allFeatures.append(featureDistances)
    # print(featureDistances)

plt.show()
```



In [ ]:
```
def eucDistance(array1, array2):
    total = 0
    for i in range(len(array1)):
        total += (array1[i] - array2[i]) ** 2
    return math.sqrt(total)
```

In [ ]:

```
euc_distances = np.zeros([len(allFeatures), len(allFeatures)])
print(euc_distances.shape)

for i in range(len(allFeatures)):
    for ii in range(len(allFeatures)):
        if ii <= i:
            continue

        euc_distances[i][ii] = eucDistance(allFeatures[i], allFeatures[ii])
```

```
(5, 5)
```

In [ ]:

```
euc_distances
```

Out[ ]:

```
array([[   0.        , 2228.32975094, 2289.01493217, 1296.94563339,
        2208.94684602],
       [   0.        ,    0.        ,  111.95772279, 1810.80871494,
          47.53311183],
       [   0.        ,    0.        ,    0.        , 1814.34049053,
         127.56175201],
       [   0.        ,    0.        ,    0.        ,    0.        ,
        1800.24028487],
       [   0.        ,    0.        ,    0.        ,    0.        ,
           0.        ]])
```

In [ ]:

```
allFeatures
```

```
Out[ ]:  [[160.0,
           296.2228890548467,
           2.0,
           264.3671689147501,
           325.6316937891642,
           105.17128885774862,
           39.05124837953327,
           262.79459659589656,
           325.0876804802052,
           215.29514625276622,
           277.72108310317384],
          [1397.0,
           353.50954725438464,
           1822.0,
           259.4937378820537,
           328.09297462761987,
           264.48629454094595,
           345.878591416121,
           267.8544380815819,
           323.2893440866865,
           214.97906874856446,
           282.1577572919093],
          [1501.0,
           317.1277345171816,
           1824.0,
           274.76535443901946,
           323.51352367404985,
           259.4937378820537,
           350.69074695520555,
           265.7461194448566,
           322.4437935516824,
           223.60679774997897,
           285.1613578309656],
          [1416.0,
           359.42454006369684,
           27.0,
           293.3751864081214,
           323.7838785362854,
           29.154759474226502,
           344.223764432382,
           274.9727259202265,
           324.98153793715727,
           216.27991122617004,
           280.8166661720775],
          [1376.0,
           325.26297053307496,
           1810.0,
           283.38313287844073,
           324.09875038327436,
           271.14940531006147,
           360.1999444752872,
           266.08645211660064,
           327.9283458318296,
           218.3437656540713,
           280.0589223717038]]
```