Hough Transform:

```
def houghTransform(img, theta_seg, rho_seg):
    height = img.shape[0]
    width = img.shape[1]

    tan_from_origin = round(np.hypot(height, width))    # Must, use this method for t
he input.bmp to work
    theta_param = np.deg2rad(np.arange(0, 180, theta_seg)) # 0 - 180 degrees
    rho_param = np.arange(-
tan_from_origin, tan_from_origin, rho_seg)    # Ranges from -max from
origin of image to max from origin

    total_thetas = len(theta_param)
    total_rho = tan_from_origin * 2
    accumulator = np.zeros((total_rho, total_thetas))   # Initalize to zero, array si
ze relative to rho, theta
    # Edge Point (x, y)
    edgeY, edgeX = np.nonzero(img)
    # Vote Accumulator
    for i in range(width):
        # Loop through edge pixel
        x = edgeX[i]
        y = edgeY[i]
        for j in range(total_thetas):
            rho = round((x * np.cos(theta_param[j])) + (y * np.sin(theta_param[j])))
+ tan_from_origin
            accumulator[rho, j] = accumulator[rho, j] + 1

    return accumulator, theta_param, rho_param
```

The Hough transform function was straight forward since most of the pseudo code was on the lecture slide. The only problem I ran into is calculating the rho parameter. It did not like it when I used the height of the image as the parameter (y-axis) for some of the test images. test.bmp and test2.bmp ran fine but when I tried input.bmp it did not seem to like it – gave me out of bound error. Therefore, I calculated the max distance from the origin, so the tangent from the origin which is the “hypotenuse” of rho and theta. The accumulator was used to keep track of the ‘intersections’ because every time a vote calculated (drawn) it is tracked. The color will change to lighter at these interactions.

Showing the result images:

```python
def showImages(origImage, cannyImg, accumulator, thetas, rhos):
    fig, ax = plt.subplots(1, 3, figsize=(15, 10))

    # Original Image
    ax[0].imshow(origImage)
    ax[0].set_title('Original')

    # Edge Detection Image
    ax[1].imshow(cannyImg, cmap=plt.cm.gray)
    ax[1].set_title('Edge Detect')

    # Hough Transform Image
    ax[2].imshow(
        accumulator, cmap=plt.cm.gist_earth, origin='lower',
        extent=[
            np.rad2deg(thetas[0]),
            np.rad2deg(thetas[-1]),
            rhos[0], rhos[-1]
        ]
    )
    ax[2].set_title('Hough Transform')
    ax[2].set_xlabel('Theta [Degrees]')
    ax[2].set_ylabel('Rho [Pixels]')

    fig.tight_layout(pad=3.0)

    plt.savefig('result.png')
    plt.show()
```

This function gave me the most problems. First this is because I never use matplotlib before, so majority of the time is me spent researching (the colors is just – wow). The main thing I want to talk about is that after the voting my graph looked “flipped” – this is because the origin starts at the top left – so I called origin=”lower” which essentially flipped the resulting image to match the cartesian coordinates.

Main Function:

```
if __name__ == '__main__':
    img = cv2.imread('input.bmp')
    origImage = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cannyImage = cv2.Canny(gray, 50, 100)

    # For Voting: These parameter seems the best - to me
    theta_seg = 0.01
    rho_seg = 0.01

    accumulator, thetas, rhos = houghTransform(cannyImage, theta_seg, rho_seg)
    showImages(origImage, cannyImage, accumulator, thetas, rhos)
```

Normally I would not analysis the main function but this time there are thing in the function that needed to be mentioned.

In this function, I called the cv2 to change the color to gray. This modified image is then passed into the cv2 built-in canny with the parameter of 50 lower and 100 upper – these values seems to work best. This is also where I would call the theta segments (increasing the value, the more "pixelated" the voting looked). When you increase the rho segments it changes the steps the rho is increased by (see image below for easy understanding).

General Approach

How I approached this assignment was by studying the lecture that went over Hough transform. Large portion of the algorithm (pseudo) is already on the sides so it made it pretty clear for me to understand. I also used Wikipedia on Hough transform which is helpful because it contained a visual representation of what rho and theta should do and effect the voting. Generally, these are the steps/approaches:

1. Take in bmp image and converts it to gray channel using cv2
2. Run canny on the gray image
3. Pass this image into the Hough transform algorithm
    - Please refer to Hough Transform section for in-depth description
4. Returns accumulator array, theta, and rho to the main
5. Pass the acquired parameter to the showImage function
    - This is where the save and plot image is done
6. [Did Not Finish Draw Line]
   The idea was to take the returned accumulator array and pass it to another function for it to draw on the canny image by using a for loop and cv2 draw line function but unfortunately I was not able to complete it.

Analysis of Result

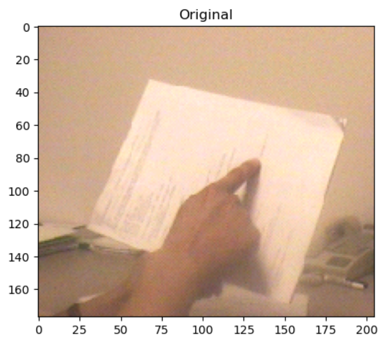
After a couple tests I realized that if I increase the theta over 1 then it is clearer to see the changes which is more pixelated. Therefore, keeping theta small means more refined and accurate but this also means more computer resources must be used to run the algorithm. Rho is in a similar way, but it was not as obvious as theta. I did not see a significant change until I started to increase the value over 25. I realize the higher the more downward shifted the graph is – this might be due to how I implemented rho parameter – but generally it does seem to alter the image. When I tested the rho by increasing the value significantly, I realized that the max of rho might depended on the size of the image that was inputted (this might be a wrong assumption) but for my algorithm anything that is equal or over 2*(image size) spits out an error. For the method of detecting intersection, I used the combination of the plt and accumulator array. Basically, I keep track of all the coordinates inside accumulator array and when I plot them the plt color changes depending on the amount of “intersection” it was happening at a certain point.

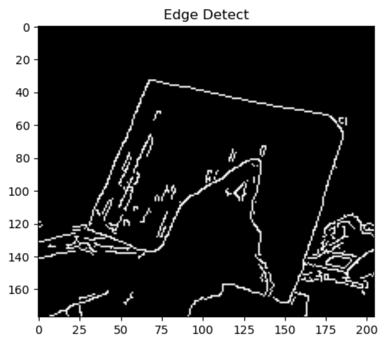
*Side note, I think with the parameter theta = 0.5 and rho = 1 might be the best since anything lower might start tax your computer and see little to no difference. Below I have included the image ran @ theta = 0.01 and rho = 0.01 – to me I see no difference, but it took significantly longer.

Image Name: result1.png

Theta: 0.5

Rho: 0.5





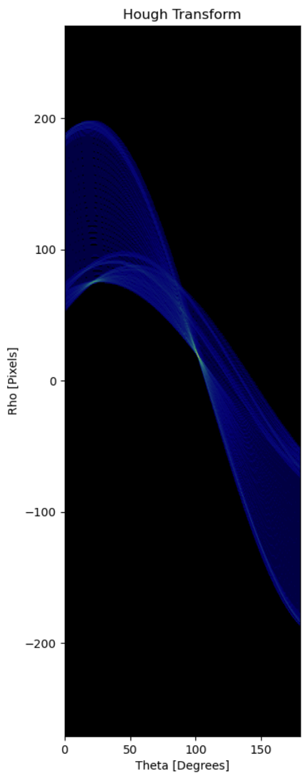


Image Name: result2.png

Theta: 1

Rho: 1

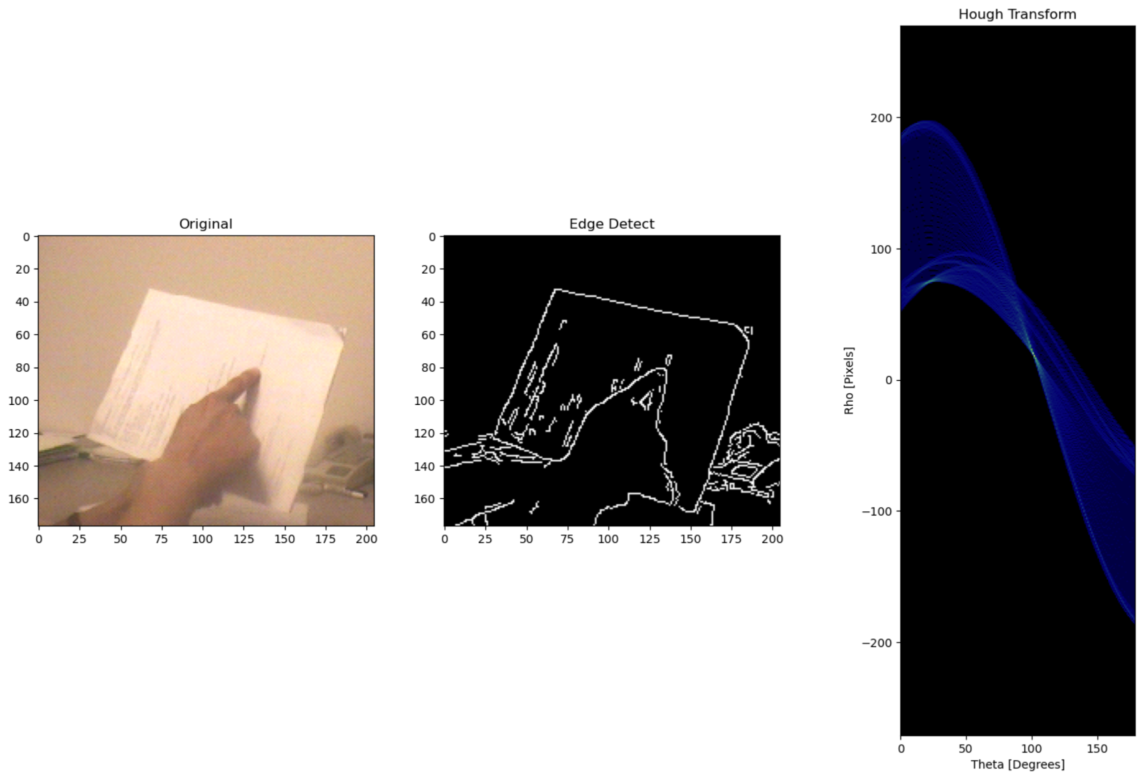


Image Name: result3.png

Theta: 5

Rho: 1

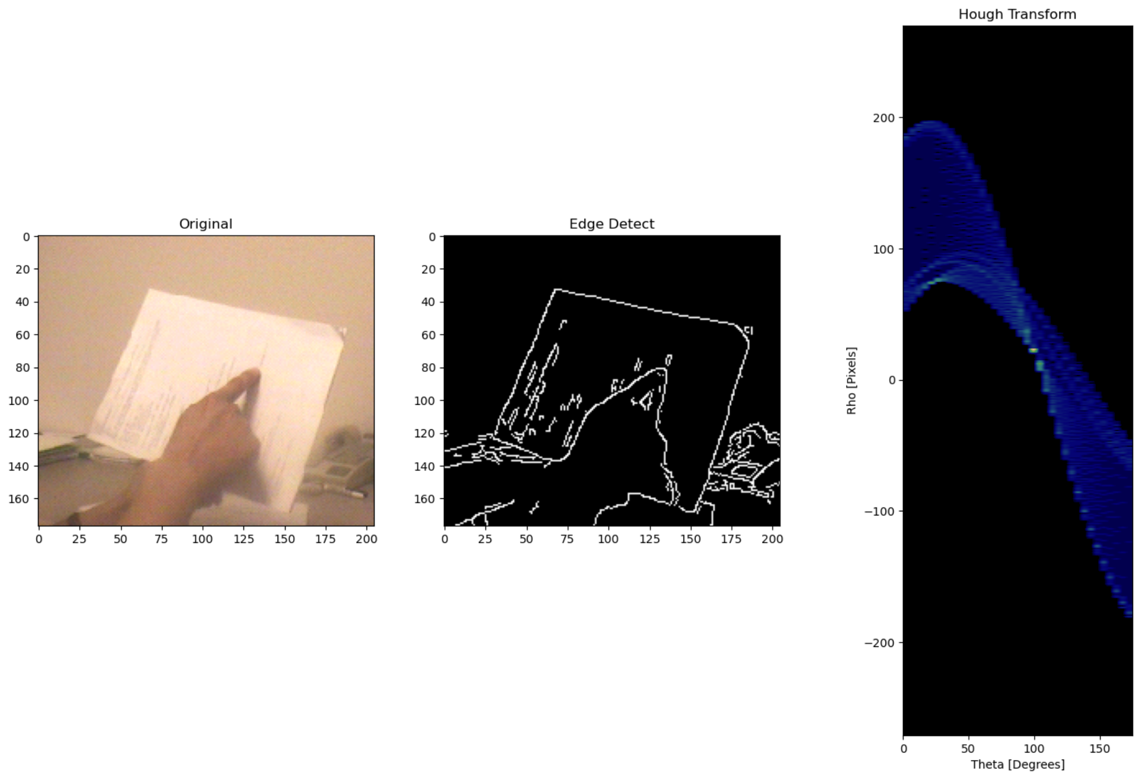


Image Name: result4.png

Theta: 5

Rho: 50

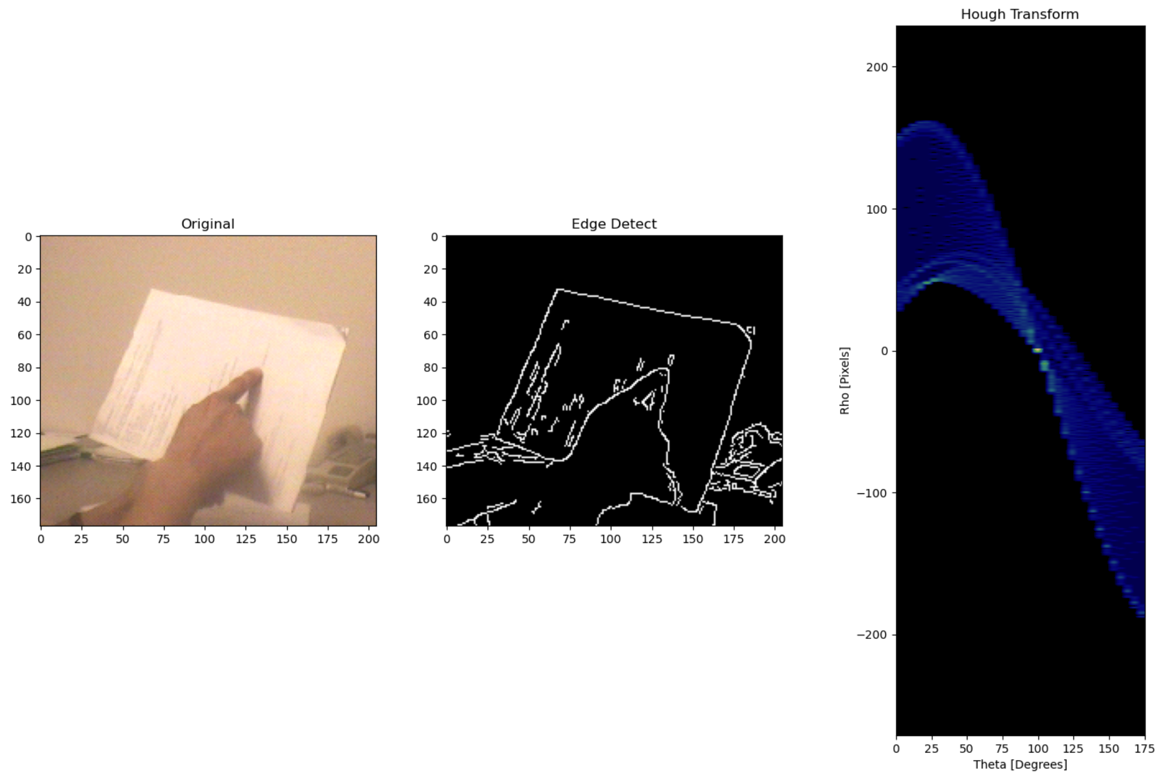
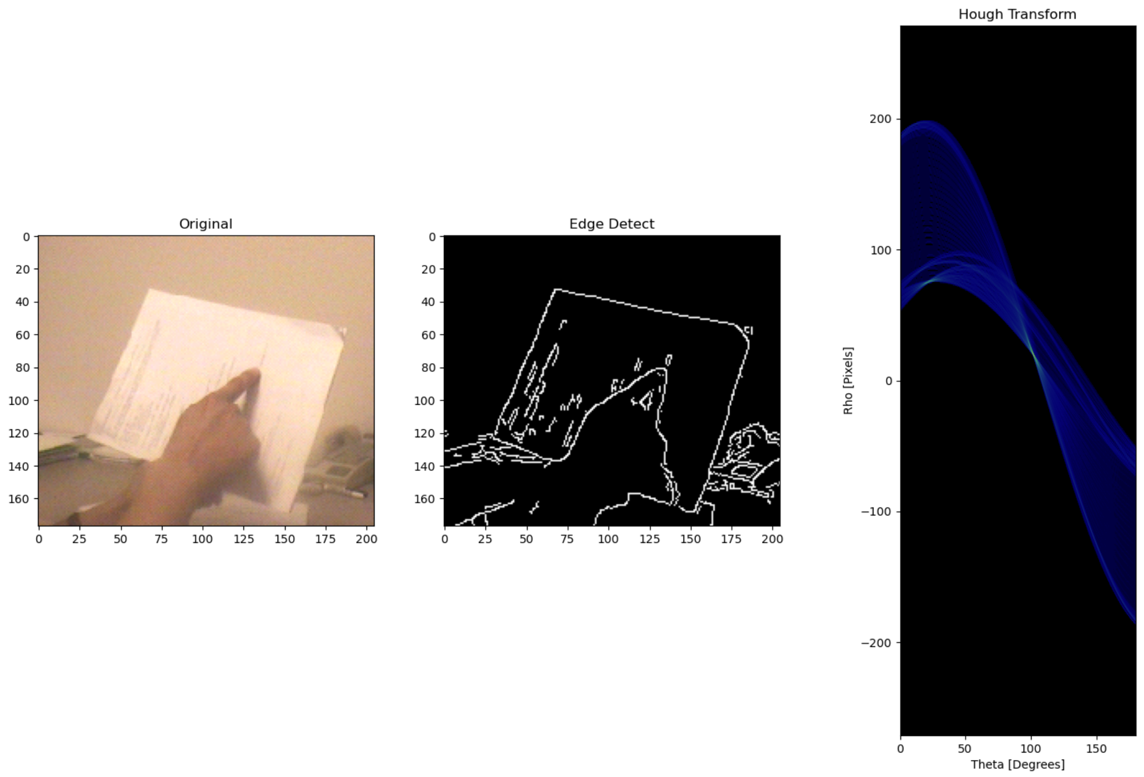


Image Name: resultPrefered.png

Theta: 0.01

Rho: 0.01


Original
Edge Detect
Hough Transform
Rho [Pixels]
Theta [Degrees]