Homework Assignment #5

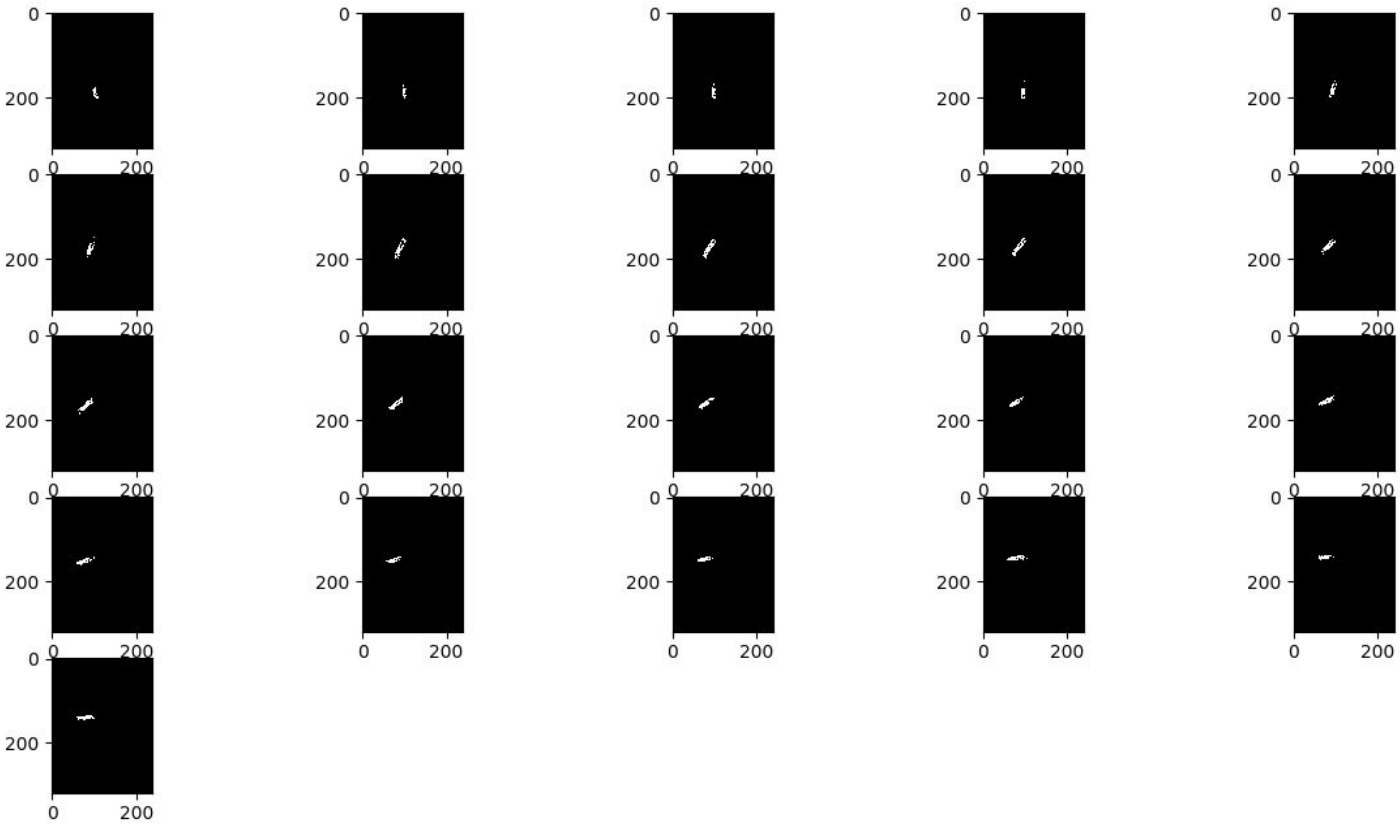
Computer Vision for HCI - CSE 5524 Shailesh Tripathi (tripathi.52)

Note: Python libraries required:numpy, matplotlib, skimage

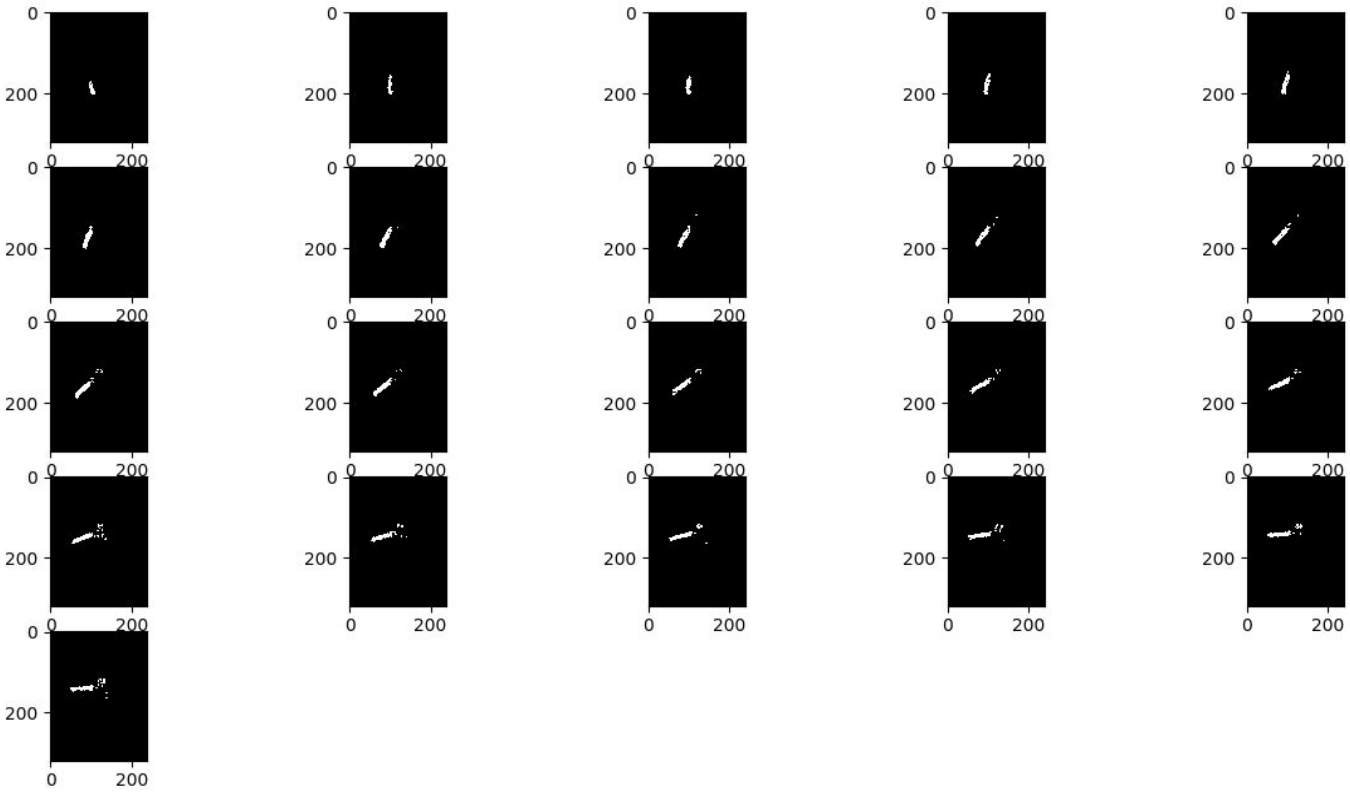
Q1.

In order to get “cleaner” results I have applied open operation once followed by two dilations in order to capture the complete arm without losing the fist.

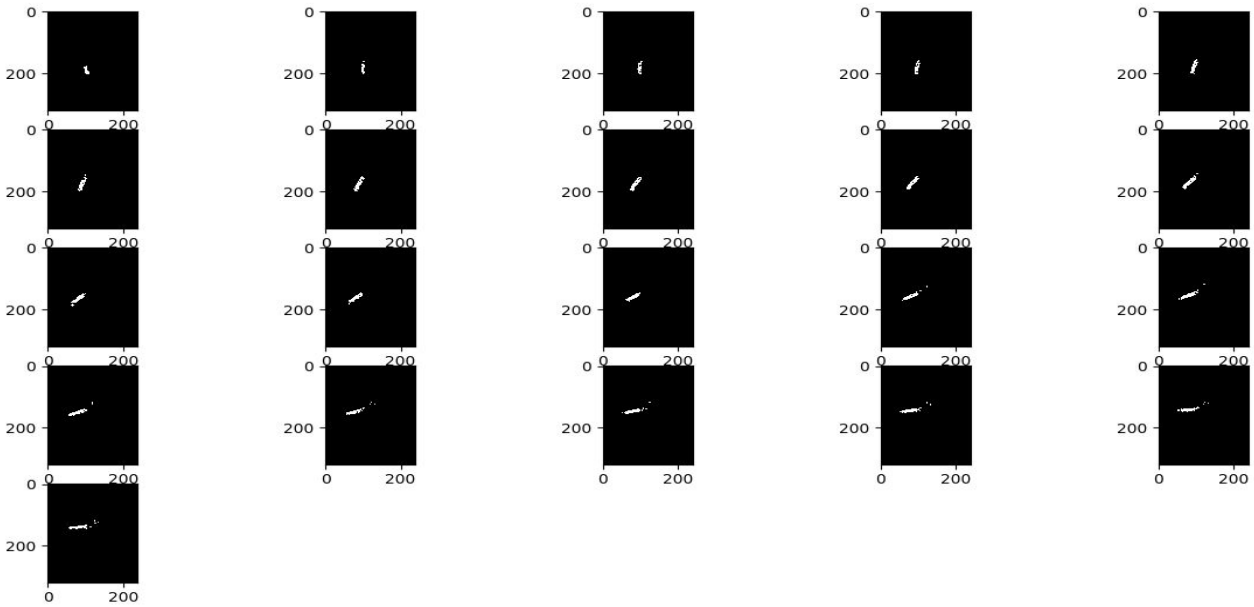
Threshold = 23



Threshold = 9



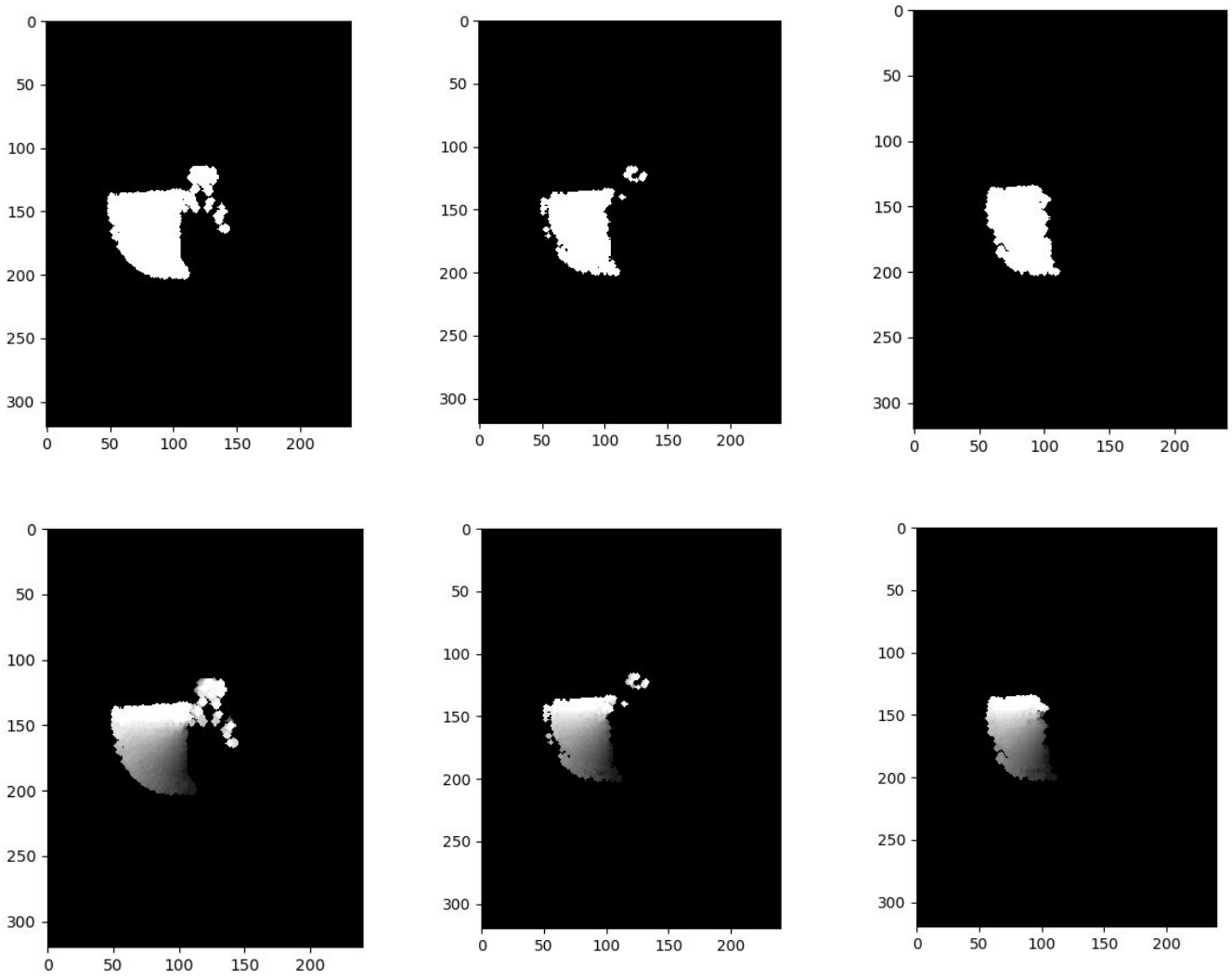
Threshold = 15



Q2.

The MEI and MHI values were calculated. In MHI, since we want to preserve the complete history, I do not drop any older values which will correspond to delta = 22.

Threshold=9 Threshold=15 **Threshold=23**



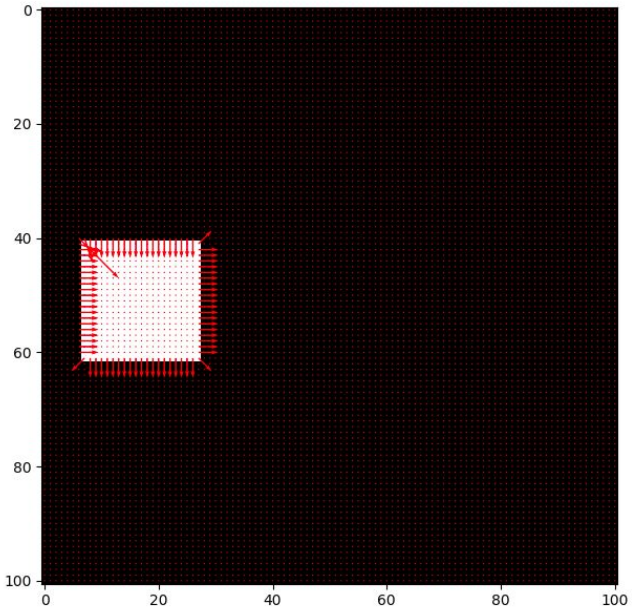
Threshold=23 was selected to compute the similitude moments.
Similitude moments for **MEI**:
[0.1329112746780686, -0.0018802190114880235, -0.00590488677682893, 0.021936601683160195, 0.08873550061034943, -0.01859990547256589, 0.007242148518870271]

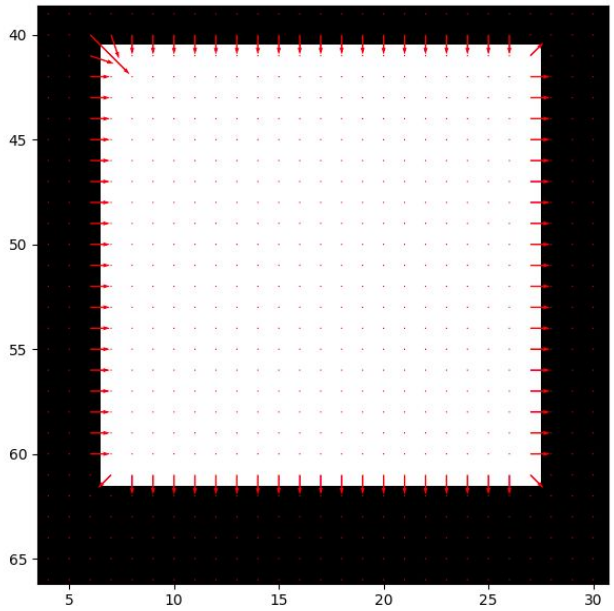
Similitude moments for **MHI**:

[0.17316161340485686, 0.028531902396828242, -0.05310177391385334, 0.03652089012753823, 0.16819950044487714, -0.049822975002065666, 0.04208116598674985]

Q3.

The normal flow for the image(left) and zoomed out image of the box(right)





Observations:

- The flow at the 4 edges have non-zero values which was expected.
- The non-box points has zero magnitude of flow(expected).
- The direction of flow at bottom-right and top-left corners seems correct but the magnitude of flow at top-left is bit large as compared to edges and magnitude of bottom-right is smaller than edges. Reason for large value of top-left is that at that pixel, the value of sqrt(f_x * f_x + f_y * f_y) is small as it has both lower horizontal and vertical flow, and also the difference f_t is large (255). Hence f_t / $f_{magnitude}$ results in a large value. [Note: here $f_{magnitude}$ is not the flow magnitude, but = sqrt(f_x * f_x + f_y * f_y)]
  Even for the bottom left pixel, the value of $f_{magnitude}$ is small, but at this position has f_t=0, the small magnitude which we see there actually corresponds to the just top-left pixel of the above mentioned pixel. For this, $f_{magnitude}$ is larger but same f_t, hence this value of f_t / $f_{magnitude}$ is least.
- Now for the top-right and bottom-left pixel, the direction of flow is inconsistent. The reason being that, this is the “normal” flow and not the actual flow. The normal to the gradient actually lies in that direction. Hence the direction vector which we are getting is inconsistent.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.morphology import dilation, opening
from skimage.filters import sobel_v, sobel_h

## Q1

numfiles = 22
img = []
imDiffBW = []
# Read all images
for filenum in range (1,numfiles+1):
    filename = 'aerobic-' + format(filenum, '03d') + '.bmp'
    img.append(np.double(plt.imread(filename)))
#    img.append(np.double(cv2.imread(filename)))

fig= plt.figure(figsize=(5,5))

fig.set_size_inches(15,7)
t=1
threshold = 23
for t in range(1,numfiles):
    imDiff = abs(img[t] -img[t-1])
    tempImDiffBW = imDiff > threshold
    tempImDiffBW = opening(tempImDiffBW)
    tempImDiffBW = dilation(tempImDiffBW)
#    tempImDiffBW = dilation(tempImDiffBW)

    imDiffBW.append(tempImDiffBW)
    fig.add_subplot(5,5,t)
    plt.imshow(imDiffBW[t-1], cmap='gray')

plt.subplots_adjust(left=0.05,right=0.95, top=0.95, bottom=0.05,hspace=0.2, wspace=0.05)
plt.show()


## Q2

# Define a function to compute Motion Energy Image
def computeMEI(imDiffBW):
    print(len(imDiffBW))
    resultIm = np.zeros((len(imDiffBW[0]), len(imDiffBW[0][0])), np.uint8)
    for i in range(0,len(imDiffBW)):
```

```
            resultIm = resultIm + imDiffBW[i]

    # Compute the binary image from the union
    resultImBW = resultIm > 0
    plt.imshow(resultImBW, cmap='gray')
    plt.show()

    # Convert binary image to float and then return
    return np.array(resultImBW, np.float)

# Function to compute Motion History Image
def computeMHI(imDiffBW):
    resultIm = np.zeros((len(imDiffBW[0]), len(imDiffBW[0][0])))
    for i in range(0, len(imDiffBW)):
            # Since we want to include results from all the diffs therefore delta = 22. But we won't compare
it with the previous values and directly use the values.
            nonZeros = np.where(imDiffBW[i]==1)
            resultIm[nonZeros[0],nonZeros[1]] = i+2 # +2 because the question asks to put the higher
value(as per MATLAB notation)

    # Scale the image to 255 to have similar display as imagesc() in MATLAB
    resultImDisp = (resultIm - resultIm.min()) / (resultIm.max() - resultIm.min())
    plt.imshow(resultImDisp, cmap='gray')
    plt.colorbar()
    plt.show()

    # The scaling was only for display. Return the actual matrix.
    return resultIm

# Get MEI and MHI
imageMEI = computeMEI(imDiffBW)
imageMHI = computeMHI(imDiffBW)

# Normalize MHI to be between 0-1.
# MEI is already an binary image, hence no normalization is required
imageMHInormalized = np.maximum(0, (imageMHI - 1.0) / 21.0)

# Function to compute mean. Inputs:( image, xOrder(column), yOrder(row))
def computeSpatialMoment(img, p, q):
    moment = 0.0
    for r in range(0, len(img)):
            for c in range(0, len(img[0])):
                    moment += (pow(r, q) * pow(c, p) * img[r][c])
```

```
    return moment

# Function to compute similitude moments. Inputs:(image, xOrder(column), yOrder(row))
def computeSimilitudeMoment(img, i, j):
    moment = 0.0

    # Zeroth order moments
    m00 = computeSpatialMoment(img, 0, 0)

    #First order moments
    m10 = computeSpatialMoment(img, 1, 0)
    m01 = computeSpatialMoment(img, 0, 1)

    centroidC = m10 / m00
    centroidR = m01 / m00

    for r in range(0, len(img)):
        for c in range(0, len(img[0])):
            moment += (pow(r - centroidR, j) * pow(c - centroidC, i) * img[r][c])

    moment = moment / pow(m00, 0.5 * (i+j) + 1)
    return moment



def similitudeMoments(boxIm):
    # Similitude Moments
    N=[]
    # Use the condition 2<= i+j <= 3 to compute values of i and j
    for i in range(0,4):
        jLower = max(0, 2-i)
        jUpper = 3 - i
        for j in range(jLower, jUpper+1):
            N.append(computeSimilitudeMoment(boxIm, i, j))
    return N

print('Please wait while the similitude moments are being calculated...')
# Compute the similitude moments
NvalsMEI = similitudeMoments(imageMEI)
NvalsMHI = similitudeMoments(imageMHInormalized)

print('similitude moments for MEI',NvalsMEI)
```

```
print('similitude moments for MHI',NvalsMHI)

## Q3
imBox1 = np.zeros((101, 101),np.float)
imBox1[40:61,6:27]=255.0

imBox2 = np.zeros((101, 101), np.float)
imBox2[41:62,7:28]=255.0

#  Compute gradient along row and column
Fcol = sobel_v(imBox2)
Frow = sobel_h(imBox2)

# Compute gradient with respect to time
Ft = (imBox2 - imBox1)

# Compute magnitude of gradient
Fmagnitude = np.multiply(Fcol, Fcol) + np.multiply(Frow, Frow)
Fmagnitude = np.sqrt(Fmagnitude)

# The points where flow is zero will give error(division by zero) while computing the unit vector for flow.
# Hence make the zero values as 1.0 as the numerator will be zero anyways while normalizing.
for x in np.nditer(Fmagnitude, op_flags=['readwrite']):
    if x == 0.0:
            x[...] = 1.0


# Compute normalized direction vector
FcolNormalized = np.divide(Fcol, Fmagnitude)
FrowNormalized = np.divide(Frow, Fmagnitude)

# Compute magnitude of flow
FmagnitudeFlow = np.divide(-Ft, Fmagnitude)

#Compute the actual length of normal flow to be displayed
FcolMagnitude = np.multiply(FcolNormalized, FmagnitudeFlow)
FrowMagnitude = np.multiply(FrowNormalized, FmagnitudeFlow)

# Use quiver to show the flow
plt.imshow(imBox2, cmap='gray')
plt.quiver(range(len(imBox1)), range(len(imBox1[0])), FcolMagnitude, -FrowMagnitude, scale=29.0,
color='r')
plt.show()
```