Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Programming 2 Report

Introduction

I chose to do the Optical Flow exercise. I'm turning in one python code named "OpticalFlow.py", which contains the Lucas-Kanade implementation. We were given frame1_a.png, frame1_b.png, frame2_a.png and frame2_b.png as our dataset. I have hard coded the image paths to be the same location as the code file. Running the code outputs the results for optical flow on top of frame1_b.png and frame2_b.png, with relevant V_x and V_y vectors shown in magenta and some of the magnitude as green arrows on the image. For the write-up, I commented out the portion I did not need and the magnitude arrows were scaled so they are more visible. I also have a function called graph() that graphs the V_x and V_y vectors (as V) only and has arrow heads that show the direction of V for each pixel point.

Each image is preprocessed by adding zero pads of width 1 and the intensity (the pixel value) of each pixel is normalized by dividing by 255. The gradient filters g_x and g_y as given in the assignment sheet were used to calculate I_x and I_y. It was calculated as I_t is calculated by subtracting frame1's intensity from frame 2. Using the given equation in the assignment sheet, V_x and V_y were calculated. My implementation of Lucas-Kanade is shown below:

OpticalFlowv4.py > lucas_kanade

priny — No results

```python
def lucas_kanade(V, padded1, padded2):  # Lucas-Kanade Function
    r_count = 0

    for row in padded1:
        end = len(row)
        if np.array_equal(row, padded1[0]):  # zero padded row
            pass  # Do nothing
        else:
            stop = end-1
            col = 1
            while col < stop:
                window1 = np.array([[padded1[r_count-1][col-1], padded1[r_count-1][col], padded1[r_count-1][col+1]],
                                    [row[col-1], row[col], row[col+1]],
                                    [padded1[r_count+1][col-1], padded2[r_count+1][col], padded2[r_count+1][col+1]]])
                window2 = np.array([[padded2[r_count-1][col-1], padded2[r_count-1][col], padded2[r_count-1][col+1]],
                                    [padded2[r_count][col-1], padded2[r_count]
                                        [col], padded2[r_count][col+1]],
                                    [padded2[r_count+1][col-1], padded2[r_count+1][col], padded2[r_count+1][col+1]]])

                w1 = 1/255 * window1  # Normalization
                w2 = 1/255 * window2  # Normalization
                I_x = w1 * DoG_gx  # Element-wise multiplication
                I_y = w1 * DoG_gy  # Element-wise multiplication
                I_t = np.subtract(w2, w1)  # Element-wise subtraction

                A = np.array([[np.sum(I_x * I_x), np.sum(I_x * I_y)],
                              [np.sum(I_x * I_y), np.sum(I_y * I_y)]])
                A_inv = np.linalg.inv(A)  # A^(-1)

                b = np.zeros((2, 1))
                b[0][0] = -1 * np.sum(I_x * I_t)
                b[1][0] = -1 * np.sum(I_y * I_t)

                u_v = np.matmul(A_inv, b)

                V[r_count-1][col-1][0] = u_v[0]
                V[r_count-1][col-1][1] = u_v[1]

                col = col + 1

            r_count = r_count + 1
```

Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Results

a) Image 1

Below are the original images of the two frames. Frame 1 is frame1_a.png and Frame 2 is frame1_b.png. We can tell with our eyes that the ball and the hands and arms of the two people move.

Image 1 Frame 1:

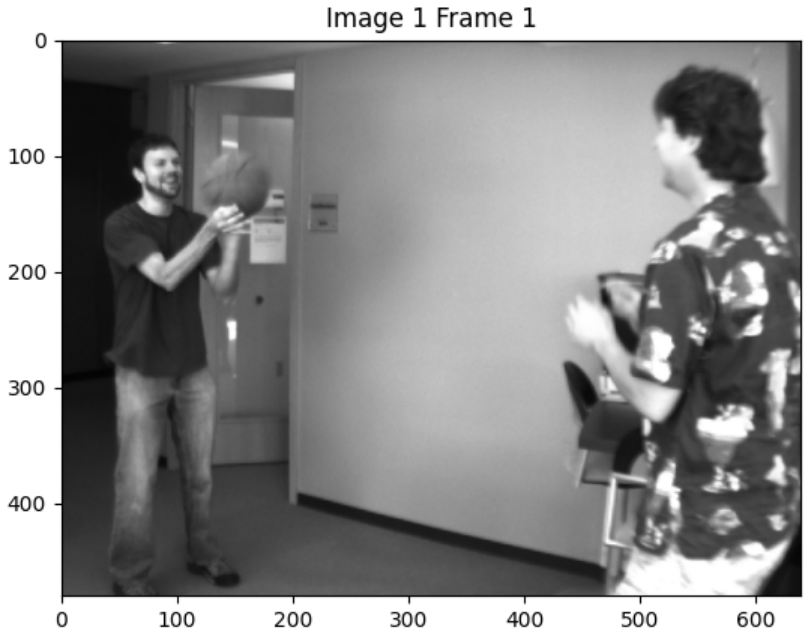
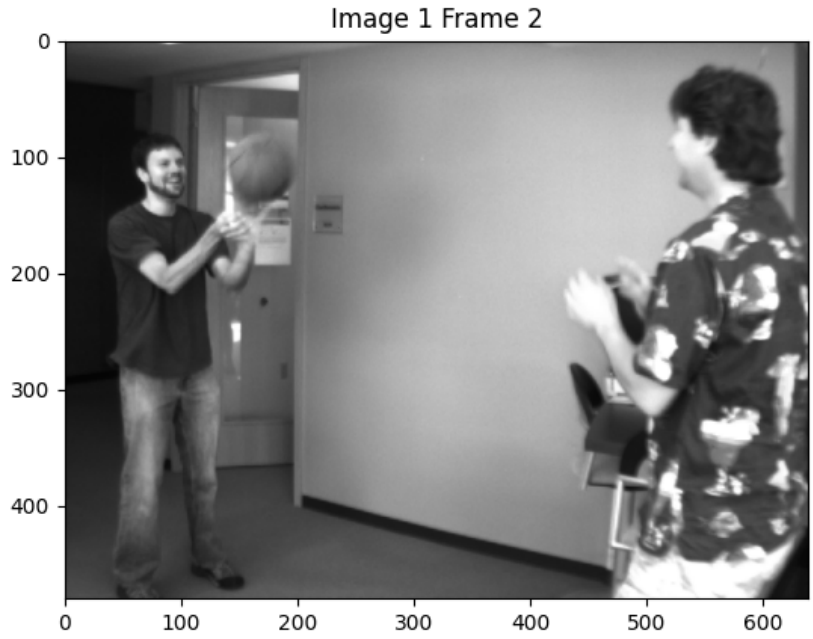


Image 1 Frame 2:



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

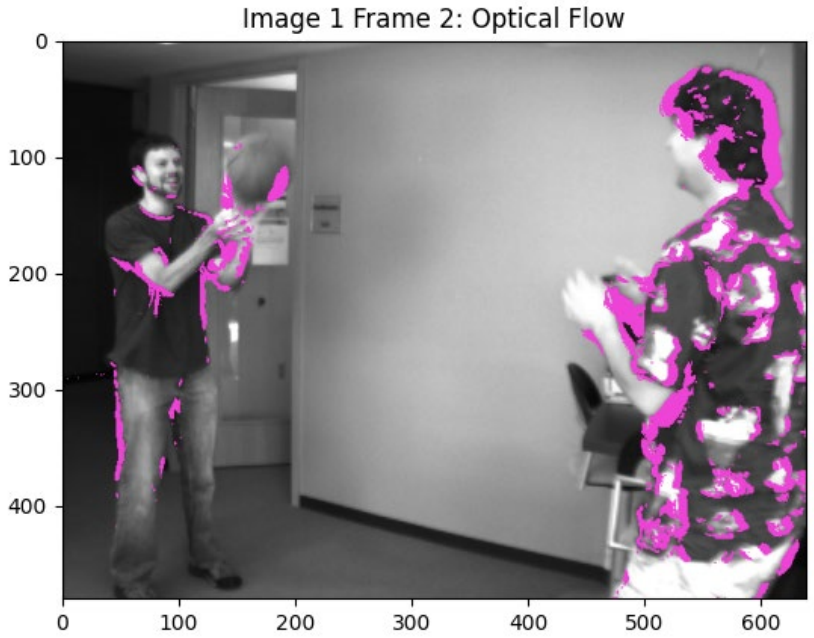
V_x and V_y values for Image 1:

```
Image 1 Vx and Vy: [[[-2.02765512e-02  4.21589235e-03]
  [ 2.41784896e-02 -1.79814989e-02]
  [ 3.35874974e-02 -4.43205907e-02]
  ...
  [-2.37262020e-03  1.84614934e-05]
  [-2.35189191e-03  5.36301255e-05]
  [ 1.28403050e-02 -1.81126726e-03]]

 [[-6.89679548e-02  1.48259947e-02]
  [ 2.66109639e-02  1.55784158e-02]
  [ 4.64741833e-02  5.11488020e-03]
  ...
  [-5.83672342e-03  5.97351090e-03]
  [ 9.19409882e-04  6.78734901e-03]
  [ 9.36752803e-03  7.80227483e-03]]

 [[-3.91630707e-02  4.75342576e-02]
  [ 6.42770283e-03  5.14184975e-02]
  [ 2.06727587e-02  2.57155237e-02]
  ...
  [-2.05219806e-03  2.11329519e-03]
  [ 3.04297446e-03  2.94911584e-03]
  [-3.63847909e-03  2.20003333e-03]]

 ...

 [[ 0.00000000e+00  2.63825122e-03]
  [ 3.22810578e-03  1.12016947e-03]
  [-3.28953720e-03  3.26037997e-03]
  ...
  [-4.45259298e-03  4.45259298e-03]
  [ 2.36541599e-02 -4.45259298e-03]
  [ 0.00000000e+00 -1.14864865e-02]]

 [[-1.14463298e-05  2.60415224e-03]
  [ 2.18796759e-03  2.17855617e-03]
  [-6.57857102e-03  2.17378868e-03]
  ...
  [-4.26386551e-03  4.80763930e-03]
  [ 1.44100054e-02 -4.62207722e-03]
  [ 9.25925926e-03 -1.75675676e-02]]

 [[ 6.56464571e-03  6.50743909e-03]
  [-6.52635839e-03  2.20239117e-03]
  [-7.84861151e-03  4.31380149e-03]
  ...
  [-5.40254237e-03 -9.00423729e-03]
  [ 0.00000000e+00  0.00000000e+00]
  [ 2.32397462e-02 -5.08762012e-03]]]
```

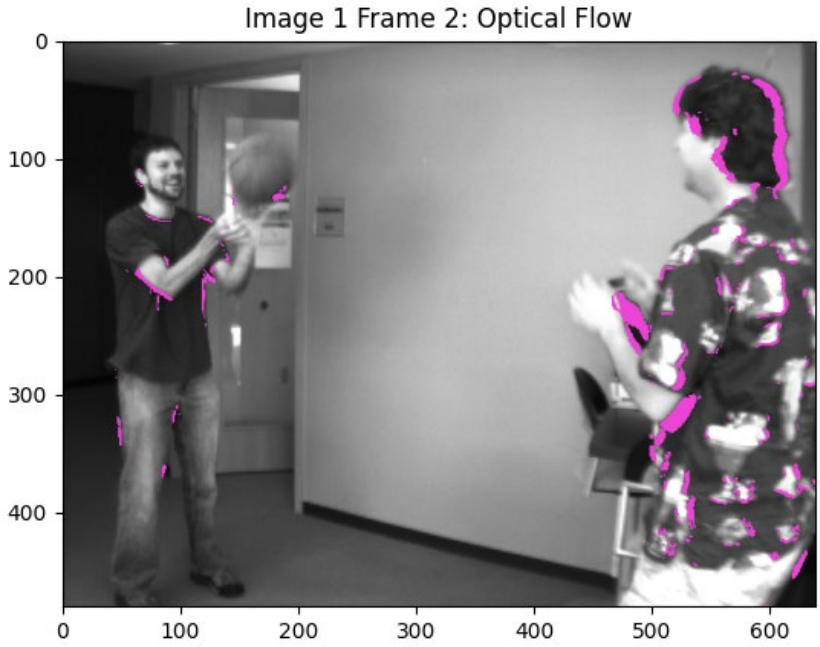
Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Image 1 Optical Flow V_x and V_y plot on image:

My values for V_x and V_y were mostly very small, and hard to visualize. I plotted V_x and V_y as magenta lines from pixel (x, y) using OpenCV's arrowedLine() function. My Lucas-Kanade Implementation detected a lot of noise, so I tried to limit the number of points plotted. Below is after ignoring V_x and V_y values between -0.1 and 0.1:

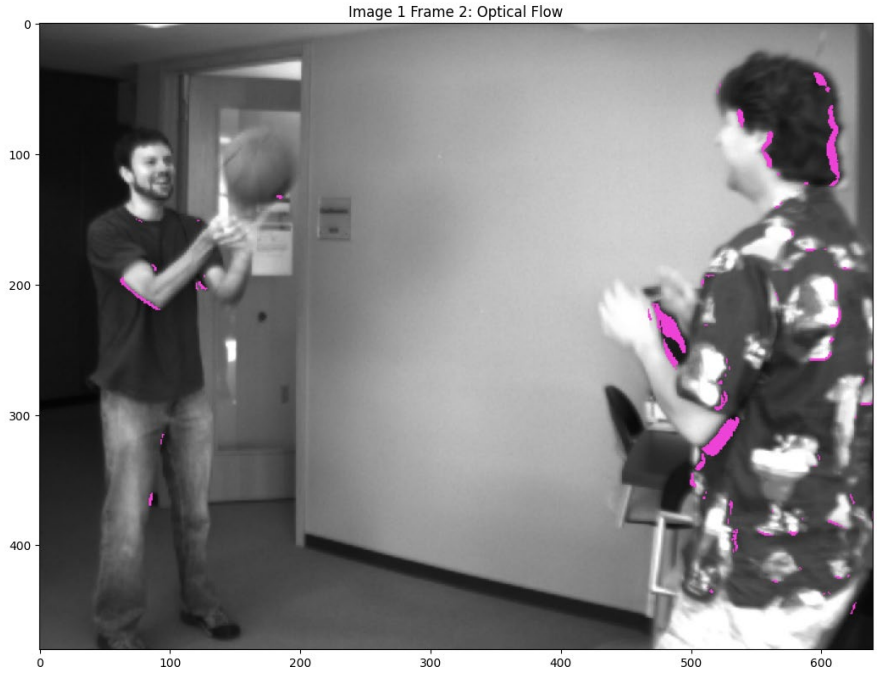


Below is after ignoring V_x and V_y values between -0.2 and 0.2:

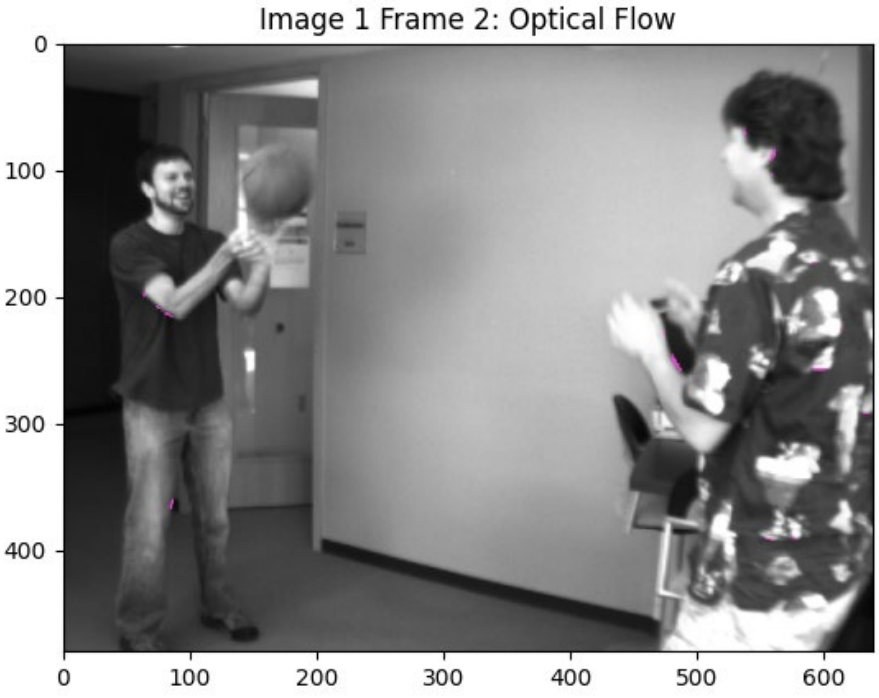


Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Below is after ignoring V_x and V_y values between -0.3 and 0.3:



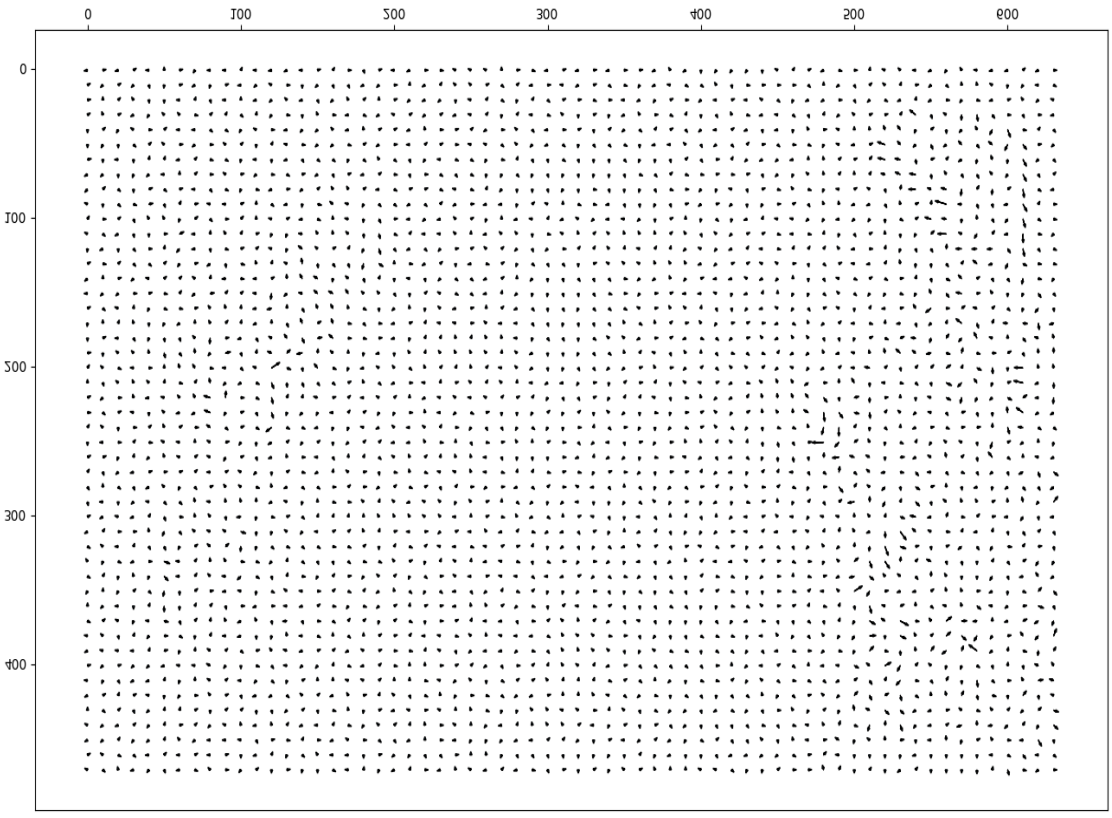
Below is after ignoring V_x and V_y values between -0.5 and 0.5



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

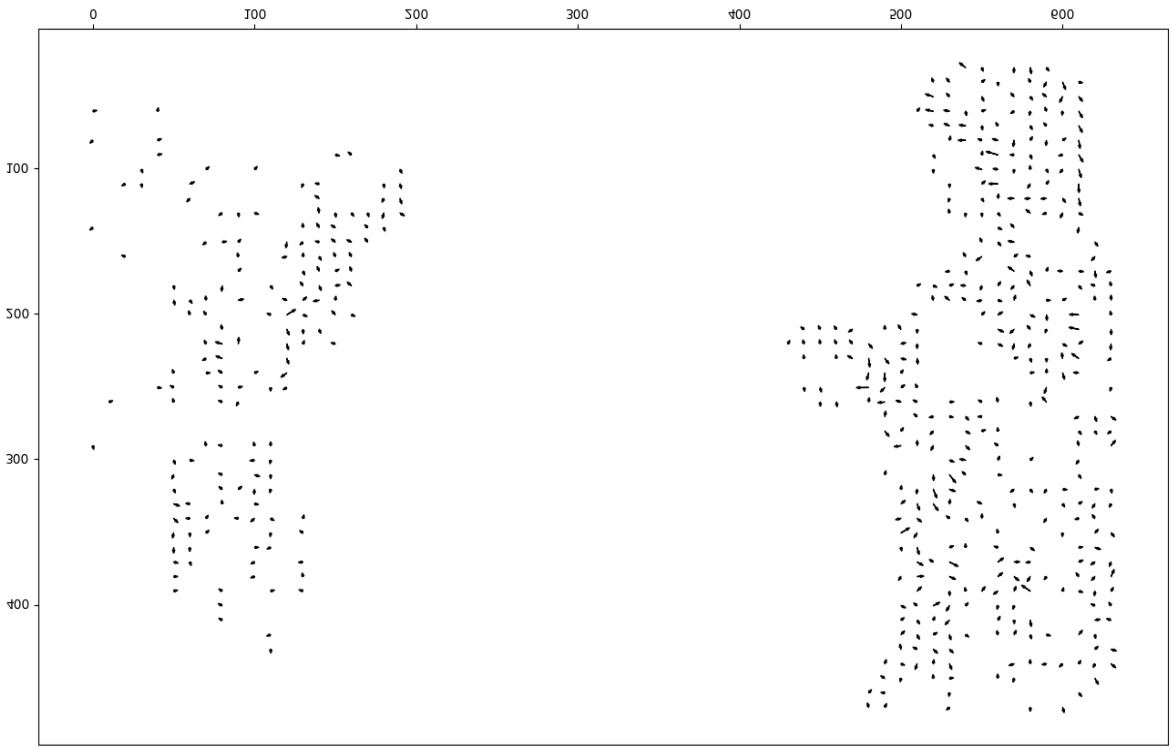
I have also included a graph() function that uses mathplotlib that is able to plot the V_x and V_y (as V) in a coordinate. The result, without any scaling was below:

Because it's still hard to tell the direction, I plotted every 10th pixel point and scaled the arrows to be 10 times longer for below result:



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

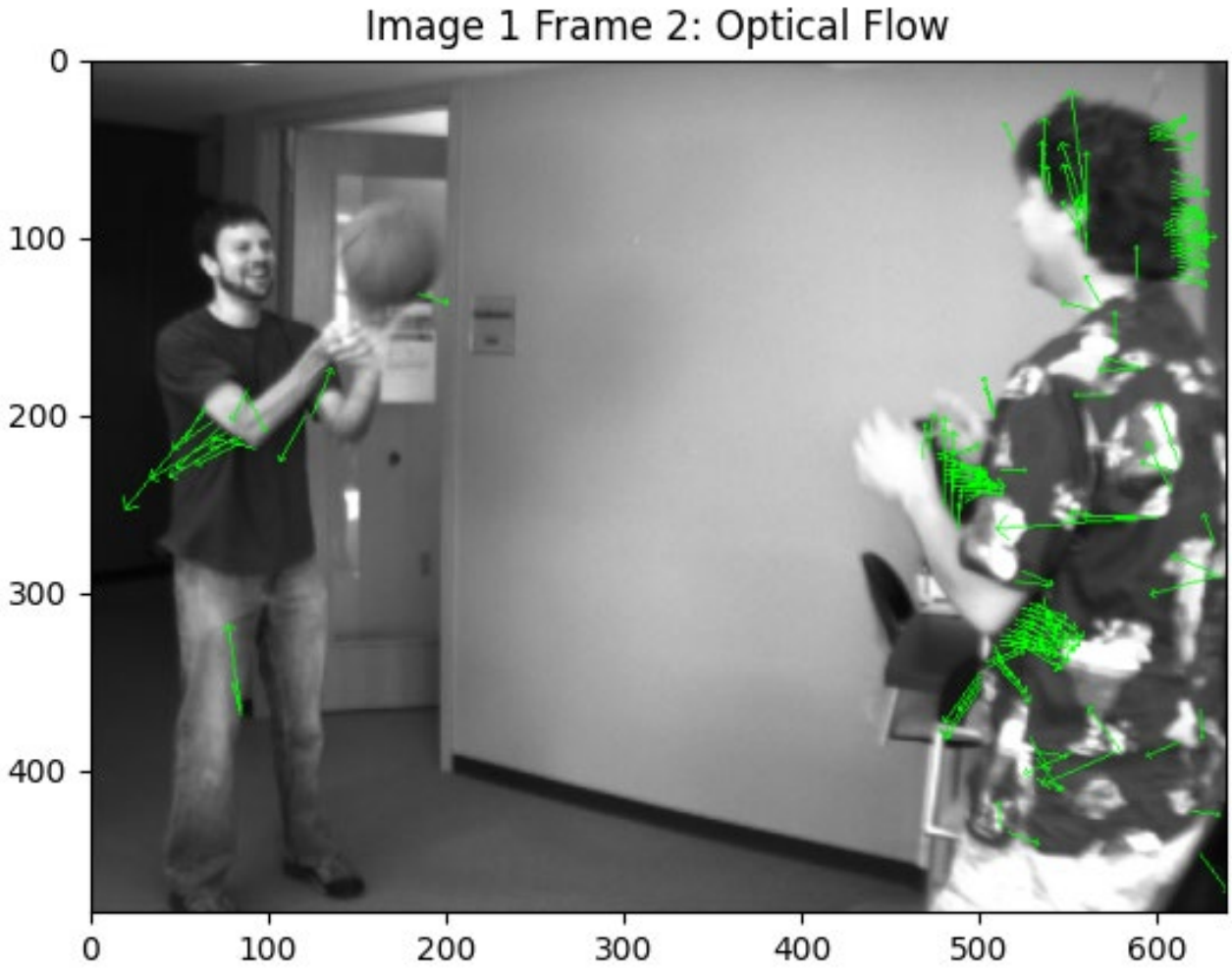
Ignoring V_x and V_y between -0.3 and 0.3 to get rid of some noise gives results below:



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Image 1 Optical Flow Magnitude:

I decided ignoring V_x and V_y between -0.3 and 0.3 looked good and used it for drawing magnitudes (green arrows). To make the arrows more visible, I multiplied the Vx and Vy values by 150 and ignored magnitudes less than 0.2. I also skipped drawing arrows on every 3rd row and every 4th column in the image.



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

b) Image 2

Below are the original images of the two frames for image 2. Frame 1 is frame2_a.png and Frame 2 is frame2_b.png. This one is hard to tell with eyes how the leaves move.

Image 2 Frame 1:

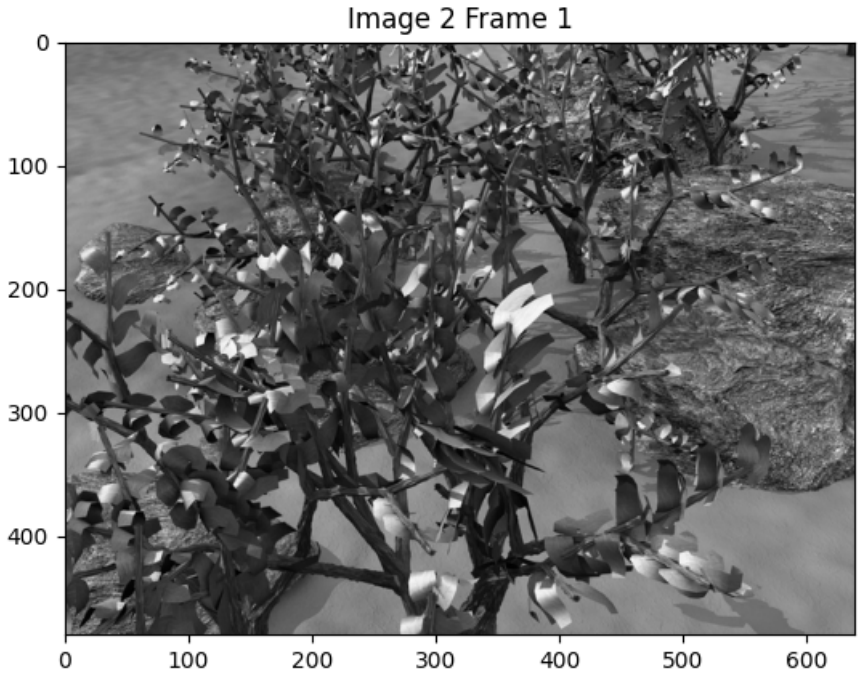
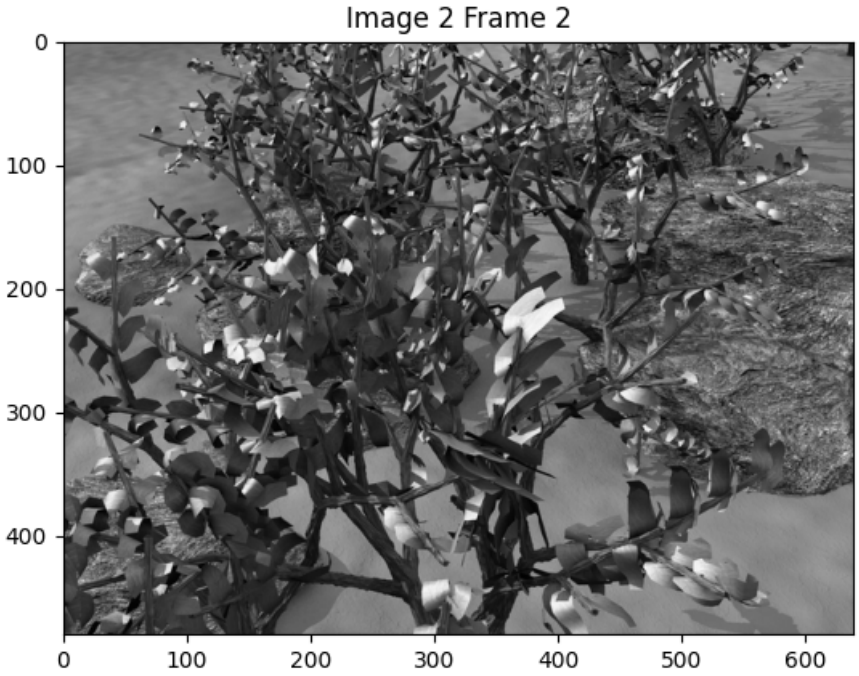


Image 2 Frame 2:



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

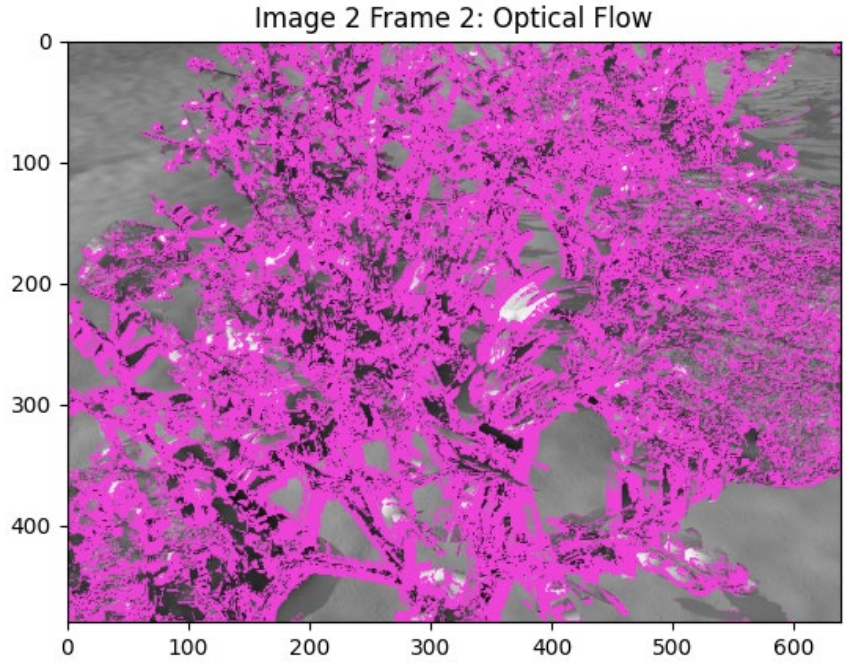
V_x and V_y values for Image 2:

```
Image 2 Vx and Vy: [[[-7.16741731e-03  1.45327234e-03]
  [ 2.56773928e-03 -1.42776310e-03]
  [ 4.20775615e-03 -1.41242938e-03]
  ...
  [-1.70302297e-01 -6.96202899e-02]
  [-3.76550243e-01  1.44141815e-01]
  [ 3.91362924e-01 -1.88455710e-01]]

 [[-5.65277988e-03  3.44090095e-03]
  [ 2.81831387e-03  2.79783792e-03]
  [ 2.07286532e-03  6.76219370e-04]
  ...
  [-3.04052846e-01 -1.27973637e-01]
  [-3.12992261e-01  2.94818124e-01]
  [ 4.50748995e-01  2.77522894e-01]]

 [[-4.21341665e-03  2.54519161e-03]
  [ 1.38855487e-03  2.80883244e-03]
  [-3.10795770e-05  1.39658483e-03]
  ...
  [-2.02535138e-01  2.35705817e-01]
  [-1.92080115e-01  3.22056947e-01]
  [ 4.09892051e-01  2.11977626e-01]]

 ...

 [[-3.47893388e-01  3.37896835e-01]
  [ 1.10344104e-01  2.97924701e-01]
  [ 9.52202828e-02  2.87161226e-01]
  ...
  [ 1.20733466e-02  8.88656948e-03]
  [ 7.87593112e-03  1.10060654e-03]
  [-1.25758711e-02  1.59840086e-02]]

 [[-3.46733223e-01  2.92509916e-01]
  [ 9.52929381e-02  2.81664292e-01]
  [ 7.95735215e-02  2.77271923e-01]
  ...
  [ 2.06762837e-02  1.01314636e-02]
  [ 2.93636286e-02  8.84782609e-03]
  [-1.97309960e-02  7.71696517e-03]]

 [[-2.66151568e-01  3.51518359e-01]
  [ 7.54871384e-03  4.46770981e-01]
  [ 4.04632523e-03  4.45776225e-01]
  ...
  [ 1.41502289e-02  1.57862366e-02]
  [ 2.92322507e-02 -2.59269795e-02]
  [ 4.17055344e-03 -5.76948791e-02]]]
```

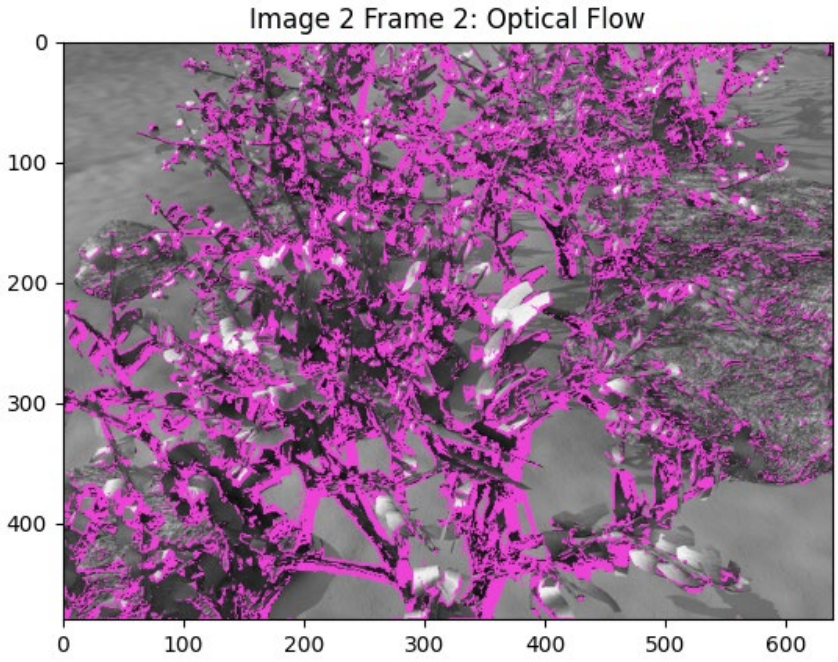
Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Image 2 Optical Flow V_x and V_y plot:

I plotted V_x and V_y as magenta lines from pixel (x, y). My Lucas-Kanade Implementation detected a lot of noise, so I tried to limit the number of points plotted. Below is after ignoring V_x and V_y values between -0.1 and 0.1:

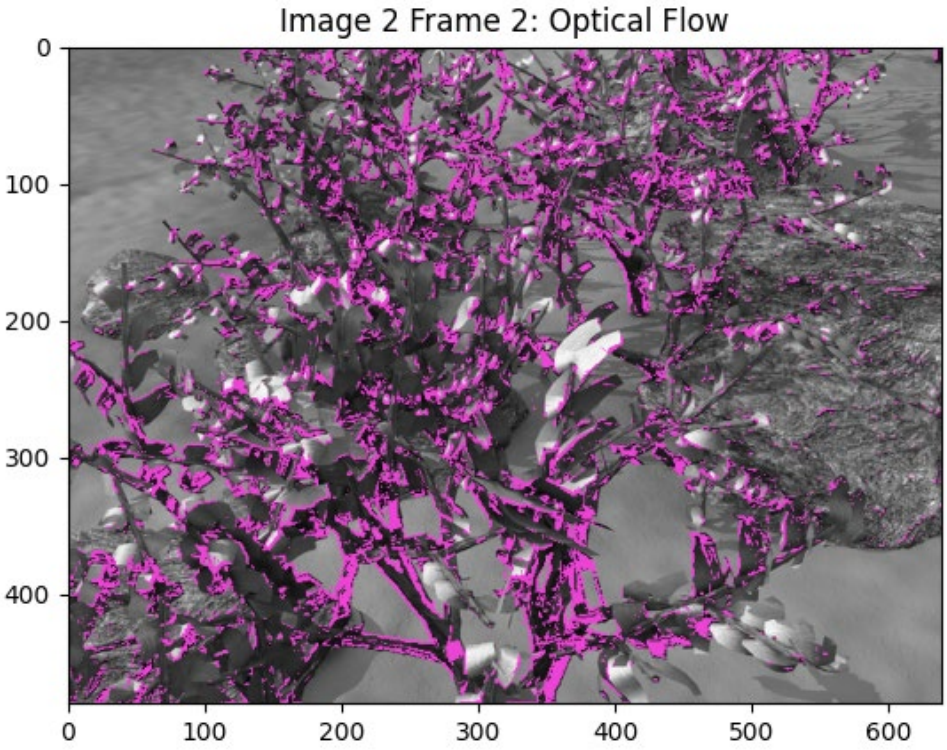


Below is after ignoring V_x and V_y values between -0.2 and 0.2:

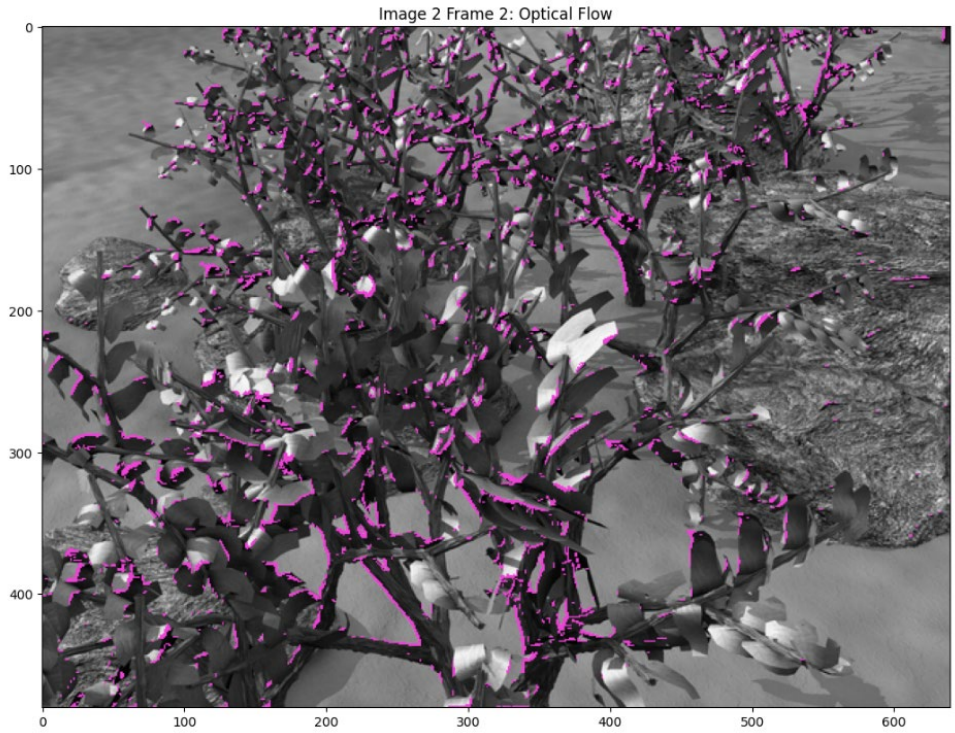


Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Below is after ignoring V_x and V_y values between -0.3 and 0.3:

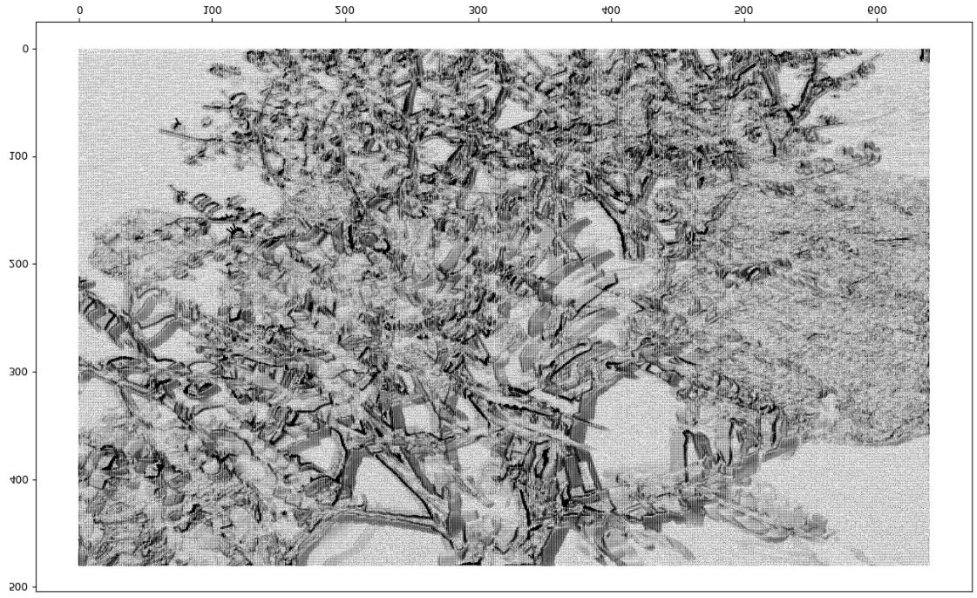


Below is after ignoring V_x and V_y values between -0.5 and 0.5:

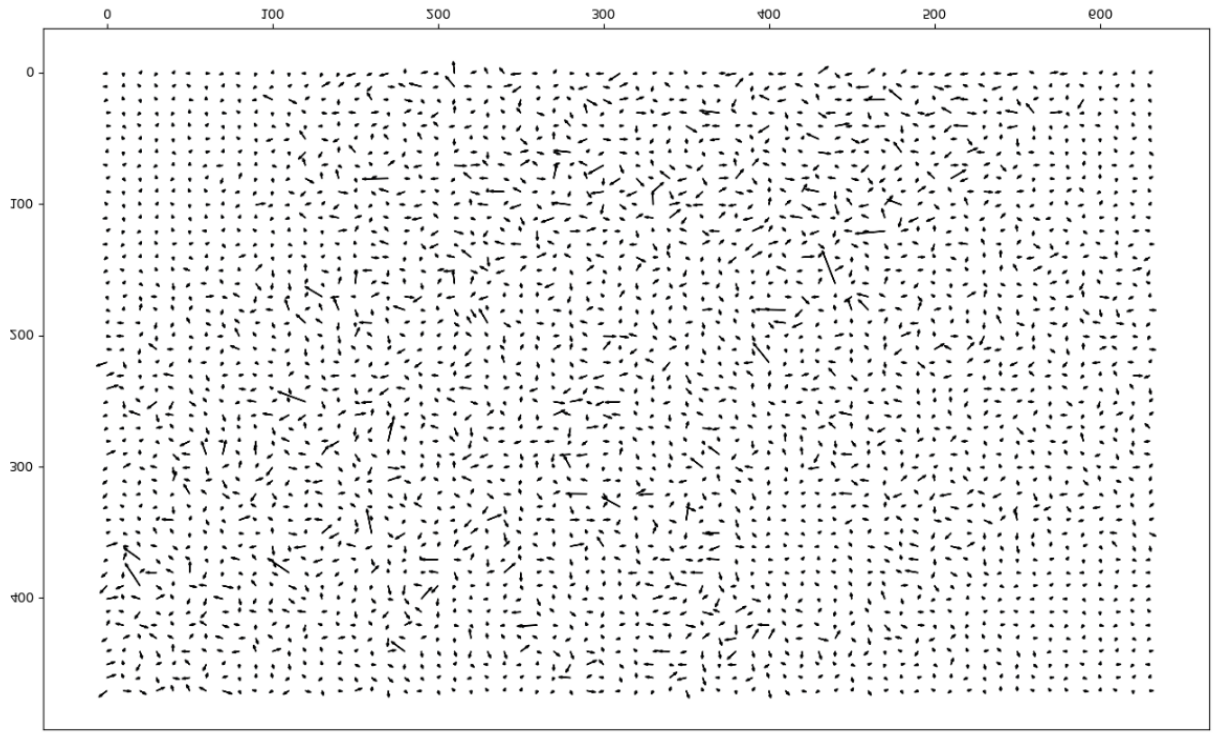


Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

The result from the graph( ) function, without any scaling was below:

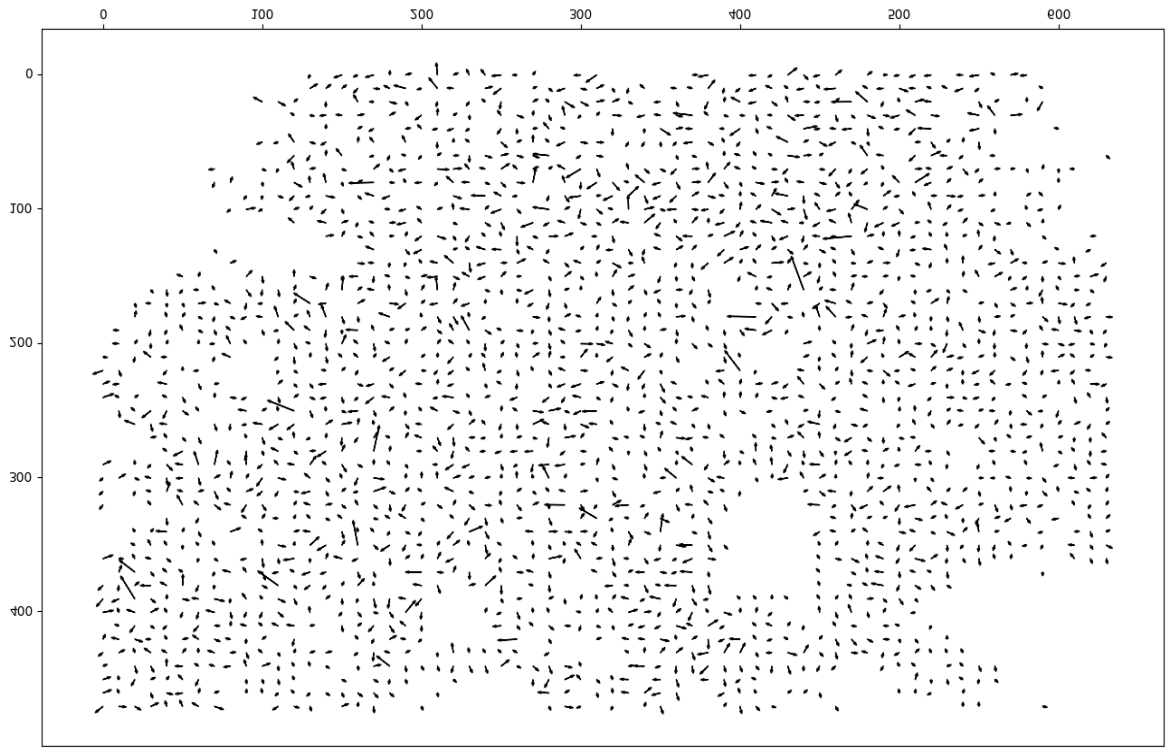


Because it's still hard to tell the direction, I plotted every 10th pixel point and scaled the arrows to be 10 times longer for below result:



Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

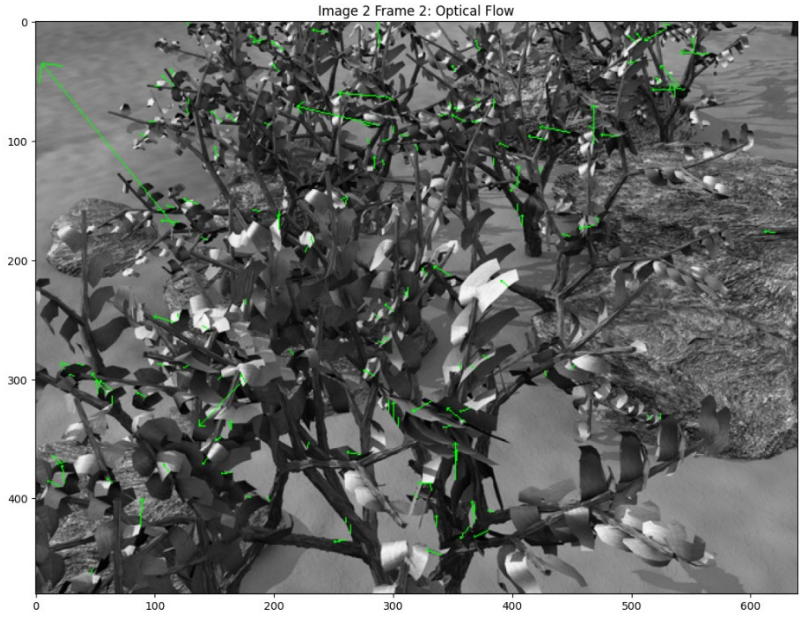
Ignoring V_x and V_y between -0.5 and 0.5 to get rid of some noise gives results below:



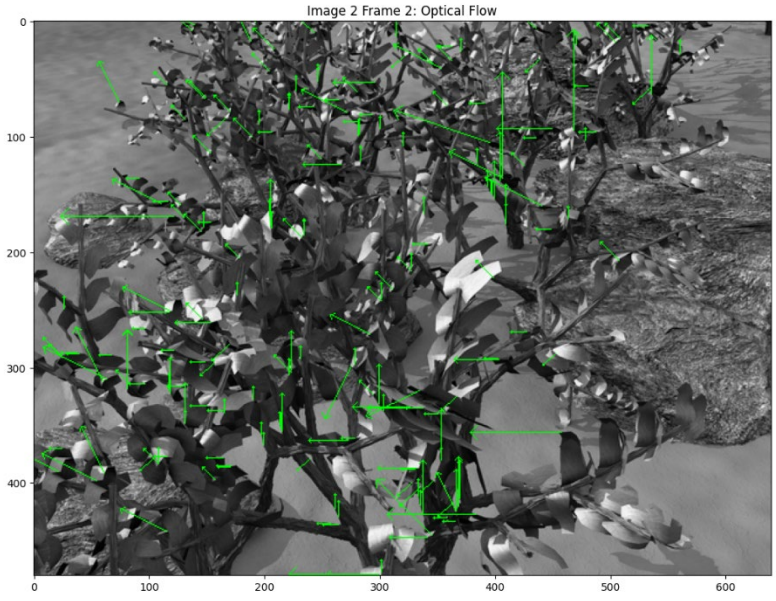
Cera Oh
CS510 CV and DL
Summer 2023
7/21/2023

Image 2 Optical Flow Magnitude:

I decided ignoring V_x and V_y between -0.5 and 0.5 looked good and used it for drawing magnitudes (green arrows) for image 2. To make the arrows more visible, I multiplied the V_x and V_y values by 4 and ignored magnitudes less than 1. I also skipped drawing arrows on every 3rd row and every 4th column in the image.



Below is another result using different scaling and plotting limit as an example:



The general movement direction is the same, even though different points' magnitudes are shown.

Conclusion

My implementation of Lucas-Kanade Optical Flow method can find the objects that move and ignore the background. It picked up a lot of noise, and perhaps I should have rounded off some of the values in the matrices. Ignoring some of the V_x and V_y values that were closer to zero helped dealing with noise. I think the general direction of the movements graphed seems correct as most movements are seen around the persons' arms in Image 1, and the person on the right is moving to the right side of the frame. For image 2, the algorithm focused mostly on the tree and its shadow, which are what you would expect to move.