Shivani Chepuri
2018122004

CV Assignment 5
Optical Flow
Lucas Kanade Algorithm Implementation

Implemented:

- Detection and segmentation of moving objects in a video (from webcam and a video file)
- Tracking of objects in a video sequence

Please find all results (images and videos) in this drive link
https://drive.google.com/open?id=1Vrvwf3nVgJvRjgEvmfjb7h3P4JLeqgBg

Algorithm Flow:

- First, the optical flow is computed for the pairs of images provided (gray images).
- Then, the scope of this finding of optical flow is done for a video sequence, here using a webcam (a video file can also be given as input).
- The function, *def opticalFlow_LKA(I1g, I2g, window_size, T=1e-2) (returns (u,v)),* involves finding the motion (u, v) that minimizes the sum-squared error of the brightness constancy equations for each pixel in a window. Initially, these equations are 9. But are reduced to 2 using a selection of key points (the points which give the least squared error).
- Here, u and v are the x and y components of the optical flow, I1 and I2 are two images taken at times t = 1 and t = 2 respectively, and window_size is a 1 × 2 vectors storing the width and height of the window used during flow computation.
- In addition to these inputs, a threshold T should be added, such that if τ is larger than the smallest eigenvalue of A'A, then the optical flow at that position should not be computed. A typical value for T is 0.01. Recall that the optical flow is only valid in regions where transpose(A)A matrix has rank 2, which is what the threshold is checking.

$$A^T A = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_y I_x & \sum I_y^2 \end{pmatrix}$$

Note: There are tradeoffs associated with using a small vs. a large window size.

Code and Explanation:
Note: The comments provide the necessary explanation.

#import necessary libraries

from imutils.video import VideoStream
import argparse
import datetime
import imutils
import time
import cv2
import os
import scipy
import numpy as np
from scipy import signal
from numpy import linalg as LA
import matplotlib.pyplot as plt
import math

#The following function calculates the optical flow.
#This function basically finds the solution for A = Ub, here psedo inverese is not taken but least squares are applied, which makes A invertible. That is on the basis of Horn&Snuchk and Lucas Kanade Algorithm.

The following images provide an idea of what is happening in this function

Lucas & Kanade

$$\sum (f_{xi}u + f_{yi}v + f_{ti})f_{xi} = 0$$

$$\sum (f_{xi}u + f_{yi}v + f_{ti})f_{yi} = 0$$

$$\sum f_{xi}^2 u + \sum f_{xi}f_{yi}v = -\sum f_{xi}f_{ti}$$
$$\sum f_{xi}f_{yi}u + \sum f_{yi}^2 v = -\sum f_{yi}f_{ti}$$

$$\begin{bmatrix} \sum f_{xi}^2 & \sum f_{xi}f_{yi} \\ \sum f_{xi}f_{yi} & \sum f_{yi}^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum f_{xi}f_{ti} \\ -\sum f_{yi}f_{ti} \end{bmatrix}$$

$$u = \frac{-\sum f_{yi}^2 \sum f_{xi}f_{ti} + \sum f_{xi}f_{yi} \sum f_{yi}f_{ti}}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi}f_{yi})^2}$$

$$v = \frac{\sum f_{xi}f_{ti} \sum f_{xi}f_{yi} - \sum f_{xi}^2 \sum f_{yi}f_{ti}}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi}f_{yi})^2}$$

```
def opticalFlow_LKA(I1g, I2g, window_size, T=1e-2):
    #Roberts Derivative masks to interpret the optical flow equation
    #These masks are applied to im1 and im2 respectively
    #After the masks are applied, the responses are added to get f_x, f_y, f_t (first derivates)

    im1_derMat_x = np.array([[-1., 1.], [-1., 1.]])
    im1_derMat_y = np.array([[-1., -1.], [1., 1.]])
    im1_derMat_t = np.array([[-1., -1.], [-1., -1.]])#*.25
    im2_derMat_x = im1_derMat_x
    im2_derMat_y = im1_derMat_y
    im2_derMat_t = np.array([[1., 1.], [1., 1.]])#*.25

    # Implement Lucas Kanade for each point, calculate I_x, I_y, I_t
    m = 'same'

    #space convolution of image with the derivate matrices
```

```
    f_x = signal.convolve2d(I1g, im1_derMat_x, boundary='symm', mode=m)
    f_y = signal.convolve2d(I1g, im1_derMat_y, boundary='symm', mode=m)
    f_t = signal.convolve2d(I2g, im2_derMat_t, boundary='symm', mode=m) +
signal.convolve2d(I1g, im1_derMat_t, boundary='symm', mode=m)

    u = np.zeros(I1g.shape)
    v = np.zeros(I1g.shape)

    #Within window, window_size * window_size
    #Calculate optical flow solution, that is [u,v] for each pixel
    #In Lucas-Kanade case calculate for each window, i.e, 3*3. Here, we took 15*15

    for i in range(w, I1g.shape[0]-w):
        for j in range(w, I1g.shape[1]-w):
            f_x1 = f_x[i-w:i+w+1, j-w:j+w+1]
            f_y1 = f_y[i-w:i+w+1, j-w:j+w+1]
            f_t1 = f_t[i-w:i+w+1, j-w:j+w+1]

            Ix = f_x1.flatten()
            Iy = f_y1.flatten()
            It = f_t1.flatten()

            # define A,b matrices in AU = b, where U = [u,v]
            i11 = Ix*It
            i12 = Iy*It
            I11 = Ix*Ix
            I12 = Ix*Iy
            I21 = Iy*Ix
            I22 = Iy*Iy
            b = np.array([-np.sum(i11),-np.sum(i12)])
            A = np.array([[np.sum(I11),np.sum(I12)],[np.sum(I21),np.sum(I22)]])
            a = A.T.dot(A)
            ue,D,ve = LA.svd(a)
            #Initialise (u,v) as zero if threshold is not met, else use the results from above
            if np.min(D) > T:
                U = (LA.inv(A)).dot(b)
                u[i,j] = U[0]
                v[i,j] = U[1]
            else:
                u[i,j]=0
                v[i,j]=0
    return (u,v)
```

```
#The function to display or plot the optical flow arrow, optical flow mask, Threshold,
Delta, #Magnitude, Angle, and the output of Lucas Kanade Algorithm, i.e, (u,v)

def plots_optical_flow(im1,im2,u,v,area_default,arrow_thres,fig_size):

    figure = plt.figure(figsize = fig_size)
    ax = figure.add_subplot(3,2,1)
    ax.imshow(im1, cmap='gray')
    ax.set_title("Image 1")
    ax.axis('on')

    ax = figure.add_subplot(3,2,2)
    ax.imshow(im2, cmap='gray')
    ax.set_title("Image 2")
    ax.axis('on')

    ax = figure.add_subplot(3,2,3)
    ax.imshow(u, cmap='gray')
    ax.set_title("U")
    ax.axis('on')

    ax = figure.add_subplot(3,2,4)
    ax.imshow(v, cmap='gray')
    ax.set_title("V")
    ax.axis('on')

    ax = figure.add_subplot(3,2,5)
    ax.imshow(u*u + v*v, cmap='gray')
    ax.set_title("Magnitude = U^2 + V^2")
    ax.axis('on')

    ax = figure.add_subplot(3,2,6)
    ax.imshow(np.arctan2(v,u), cmap='gray')
    ax.set_title("arc(v/u)")
    ax.axis('on')

    figure = plt.figure(figsize = fig_size)
    ax = figure.add_subplot(1,2,1)
    ax.imshow(im1, cmap='gray')
    ax.set_title("Optical flow Arrows")

    #Find the key points which give the least squared error among all the points in the
    #window. The number of key points we are considering here is 200. We will show the
```

```
    #optical flow arrows on these key points.

    key_points = cv2.goodFeaturesToTrack(im1, 200, 0.01, 10, 3)
    for i in key_points:
        x,y = i[0]
        y = int(y)
        x = int(x)
        ax.arrow(x,y,u[y,x],v[y,x], head_width = 2, head_length = 5, color = (1,0,0))


    ax = figure.add_subplot(1,2,2)
    ax.imshow( (u*u + v*v > arrow_thres), cmap='gray')
    ax.set_title("optical_flow mask")
    ax.axis('on')

    #Considering the 2 images, resizing them and finding the difference between them
    frame1 = imutils.resize(im1, width=500)
    frame2 = imutils.resize(im2, width=500)

    #gray1 = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    gray1 = frame1
    gray1 = cv2.GaussianBlur(gray1, (21, 21), 0)

    #gray2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    gray2 = frame2
    gray2 = cv2.GaussianBlur(gray2, (21, 21), 0)

    # compute the absolute difference between the current frame and first frame
    Delta = cv2.absdiff(gray1, gray2)
    Threshold = cv2.threshold(Delta, 25, 255, cv2.THRESH_BINARY)[1]
    # dilate the thresholded image to fill in holes, then find contours on thresholded image
    Threshold = cv2.dilate(Threshold, None, iterations = 5)
    C = cv2.findContours(Threshold.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    C = imutils.grab_contours(C)

    # loop over the contours
    for c in C:
    # if the contour is too small, ignore it
        if cv2.contourArea(c) < area_default:
            continue
    # compute the bounding box for the contour, draw it on the frame,
        (x, y, w, h) = cv2.boundingRect(c)
```

```
        cv2.rectangle(frame2, (x, y), (x + w, y + h), (0, 255, 0), 2)

    figure = plt.figure(figsize = fig_size)
    ax = figure.add_subplot(3,1,1)
    ax.imshow(frame2, cmap='gray')
    ax.set_title("frame")
    ax.axis('on')

    ax = figure.add_subplot(3,2,1)
    ax.imshow(Threshold, cmap='gray')
    ax.set_title("Threshold")
    ax.axis('on')

    ax = figure.add_subplot(3,3,1)
    ax.imshow(Delta, cmap='gray')
    ax.set_title("Delta")
    ax.axis('on')
    plt.show()
    return None

#function that takes 2 consecutive time evoluting frames(1st and 2nd, 2nd and 3rd
#frames, and so on) in a video and applies the previously defined optical flow function
#to calculate the optical flow between each set of such 2 frames in the video. This
#function is an extension of 2 images to a video.
#Note: It is lagging or the processing is slow, with high and moderate fps, due to the
#heavy computation

def segment(path, flag, w, arrow_thres, area_default):
        # if flag is 0, then we are reading from webcam
        if flag==0:
                vs = VideoStream(src=0).start()
                time.sleep(2.0)
                #vs = cv2.VideoCapture(0)
        # otherwise, we are reading from a video file
        else:
                vs = cv2.VideoCapture(path)
        # initialize the first frame in the video stream
        #Define an empty frame to compare with the video's first frame.

        firstFrame = None
        # loop over the frames of the video
        while (True):
                frame = vs.read()
```

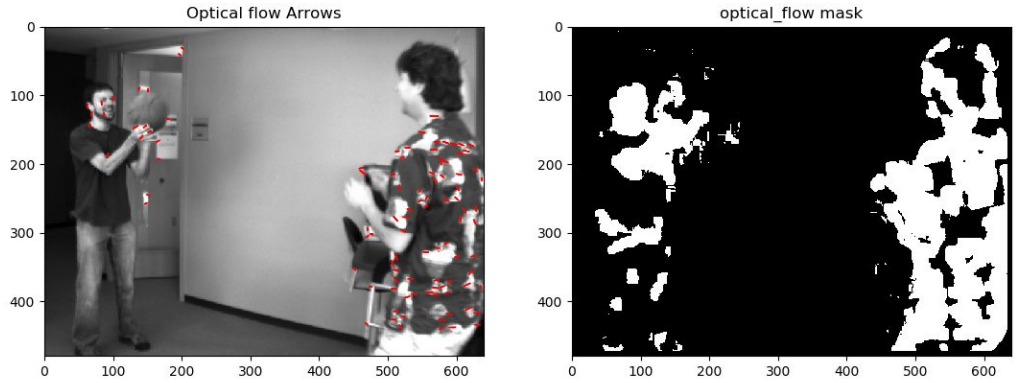
```
#print("this is frame",frame)
if flag == 0:
        frame = frame
else:
        frame = frame[1]
# if the frame could not be grabbed, then we have reached the end of the
#video
if frame is None:
        print("110")
        break
# resize the frame, convert it to grayscale, and blur it
#frame = imutils.resize(f, width=500)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (21, 21), 0)
# if the first frame is None, initialize it
if firstFrame is None:
        firstFrame = gray
        continue
# compute the absolute difference between the current frame and first
#frame
Delta = cv2.absdiff(firstFrame, gray)
Threshold = cv2.threshold(Delta, 25, 255, cv2.THRESH_BINARY)[1]
#dilate the thresholded image to fill in holes, then find contours on
#thresholded image
Threshold = cv2.dilate(Threshold, None, iterations=2)
C = cv2.findContours(Threshold.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
C = imutils.grab_contours(C)

Im1 = firstFrame / 255.
Im2 = gray / 255.

u,v = opticalFlow_LKA(Im1,Im2,w)

# loop over the contours
for c in C:
        # if the contour is too small, ignore it
        if cv2.contourArea(c) < area_default:
                continue
        # compute the bounding box for the contour, draw it on the frame,
        (x, y, w, h) = cv2.boundingRect(c)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
# show the frame and record if the user presses a key
```

```
                cv2.imshow("Frame", frame)
                cv2.imshow("Threshold", Threshold)
                cv2.imshow("Delta", Delta)

                figure = plt.figure(figsize = fig_size)
                ax = figure.add_subplot(1,2,1)
                ax.imshow(gray, cmap='gray')
                ax.set_title("Optical flow Arrows")

                #Get good keypoints
                kps = cv2.goodFeaturesToTrack(gray, 100, 0.01, 10, 3)
                for i in kps:
                        x,y = i[0]
                        y = int(y)
                        x = int(x)
                        ax.arrow(x,y,u[y,x],v[y,x], head_width = 2, head_length = 5, color =
(1,0,0))

                ax = figure.add_subplot(1,2,2)
                ax.imshow( (u*u + v*v > arrow_thres), cmap='gray')
                ax.set_title("optical_flow mask")
                ax.axis('on')
                plt.show()

                cv2.waitKey(1)
                #cv2.imshow("Magnitude", (u*u + v*v > arrow_thres))
                key = cv2.waitKey(1) & 0xFF
                # if the `w` key is pressed, break from the loop and stop streaming
                if key == ord('w'):
                        break

        # cleanup the camera and close any open windows
        if flag==0:
                vs.stop()
        else:
                vs.release()
        cv2.destroyAllWindows()


Dir ='/home/shivani/CV/eval-data-gray/'
path = '/home/shivani/CV/videoplayback.mp4'
```

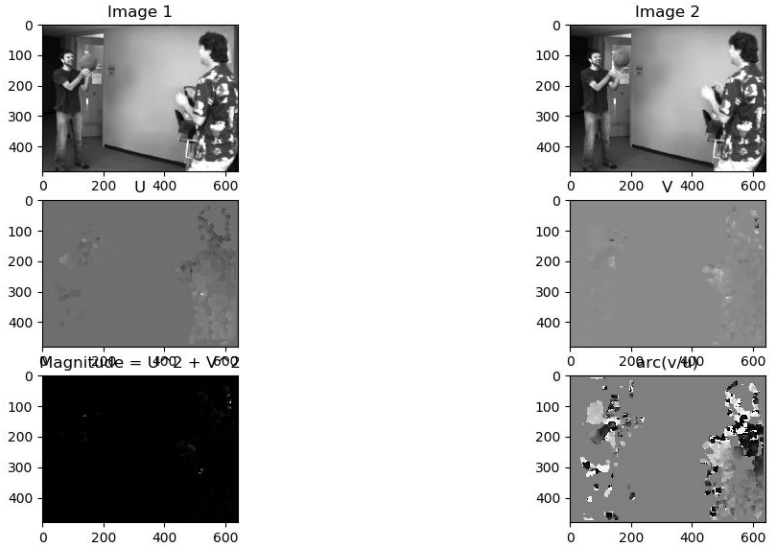
```
img_list = []
for folder in os.listdir(Dir):
    #Get the list of all images in the 'path'
    list_dir = os.listdir(os.path.join(Dir,folder))
    #read images in grayscale
    img1 = cv2.imread(os.path.join(Dir,folder,list_dir[0]),0)
    img2 = cv2.imread(os.path.join(Dir,folder,list_dir[1]),0)
    img_list.append([img1,img2])

cnt = 0
window_size = 15
w = int(window_size/2)
arrow_thres = 0.1
flag = 0
fig_size = (16,16)
area_default = 500
# Calculate optical flow on images
#Note Run either the following commented part or the segment() function, so the
machine's #resources wont be exhausted.
"""
for ims in img_list:
        im1,im2 = ims
        # normalize pixels
        Im1 = im1 / 255.
        Im2 = im2 / 255.
        u,v = opticalFlow_LKA(Im1,Im2,w)
        plots_optical_flow(im1,im2,u,v,area_default,arrow_thres,fig_size)
        print(im1.shape)
        print(im2.shape)
        cnt +=1

#function that applies optical flow on a video
#flag is 0 for live capture from webcam
#flag is 1 for reading from a video file
segment(path, flag, w, arrow_thres, area_default)
```

Please find all results (images and videos) in this drive link
https://drive.google.com/open?id=1Vrvwf3nVgJvRjgEvmfjb7h3P4JLeqgBg

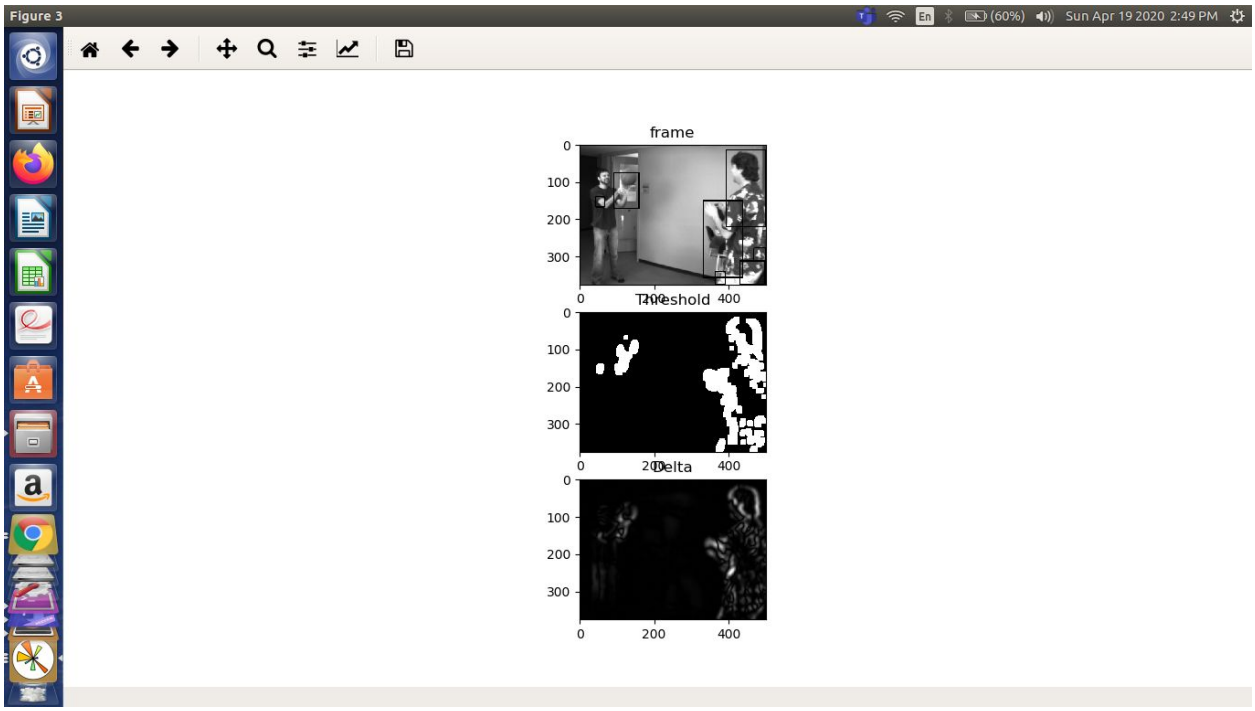
Results:
Showing optical flow arrows and optical flow mask.



Showing Image 1, image 2, U, V, Magnitude, and Angle respectively.



Showing <u>object detection and segmentation</u>, Threshold mask and Delta mask. Here in the Head and hand areas, the movement is detected. Therefore, the object that is being moved/displaced is tracked. We can observe the bounding boxes in those areas.



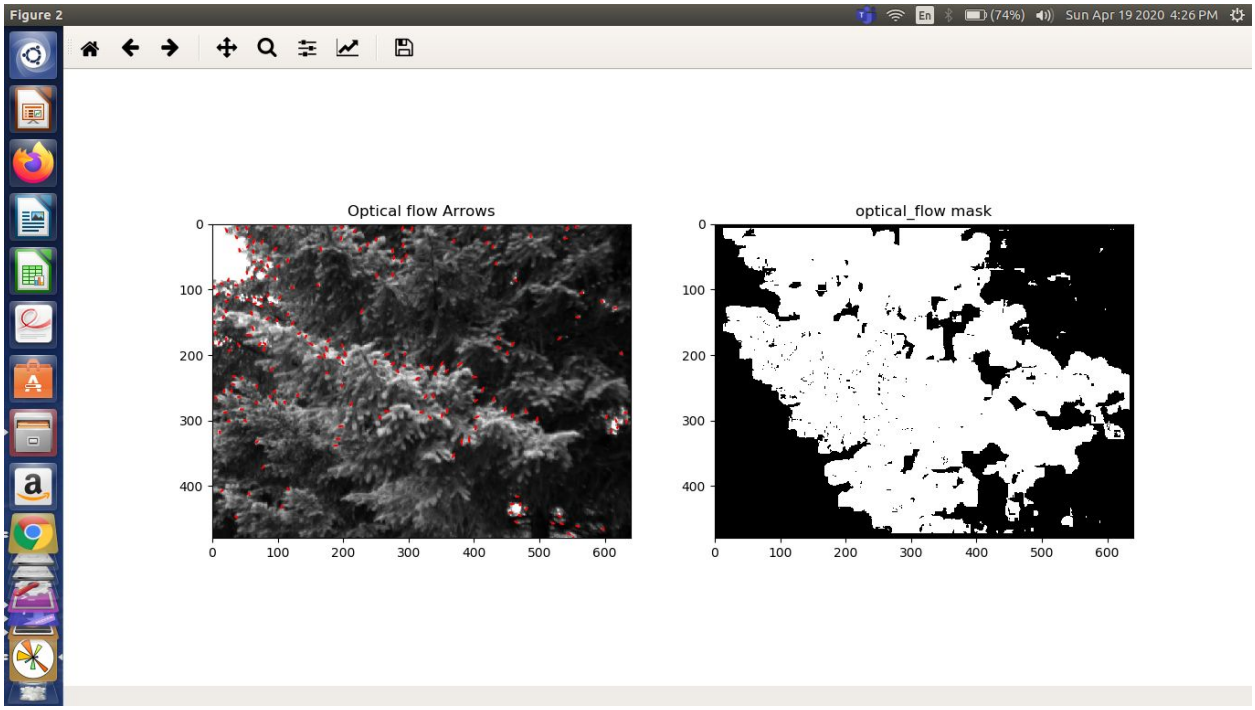


Figure 2
Sun Apr 19 2020 4:26 PM
Optical flow Arrows
optical_flow mask

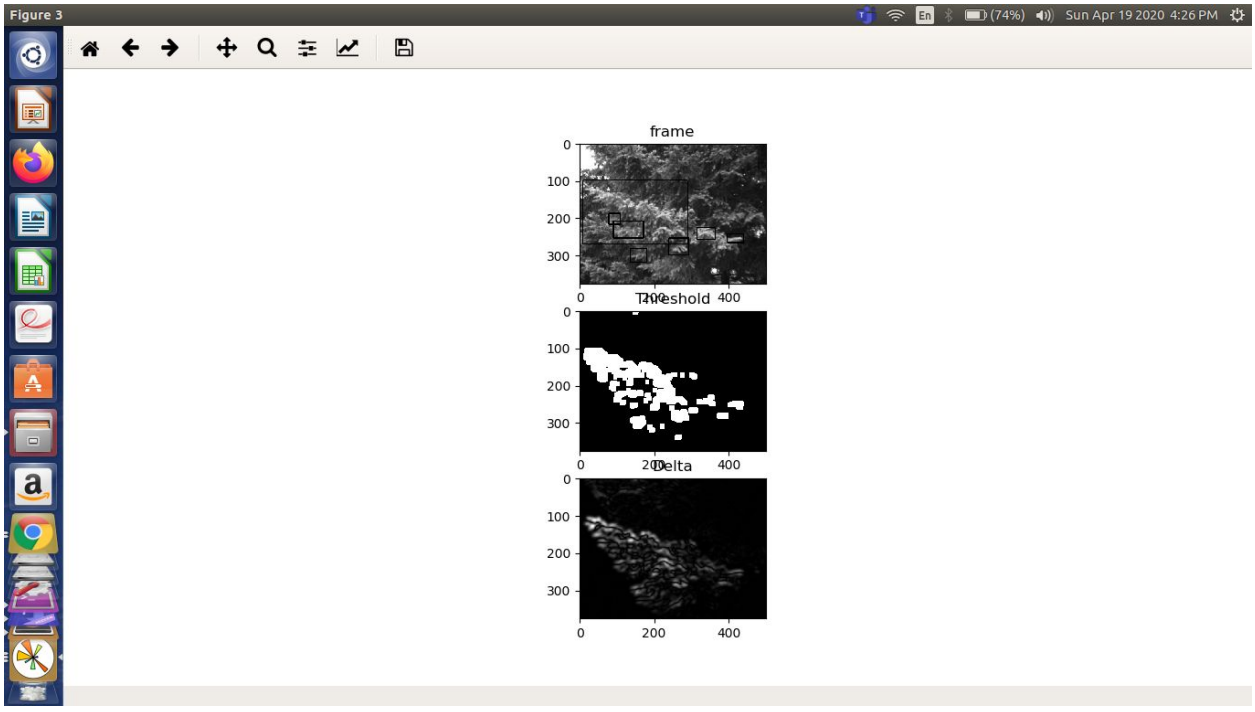
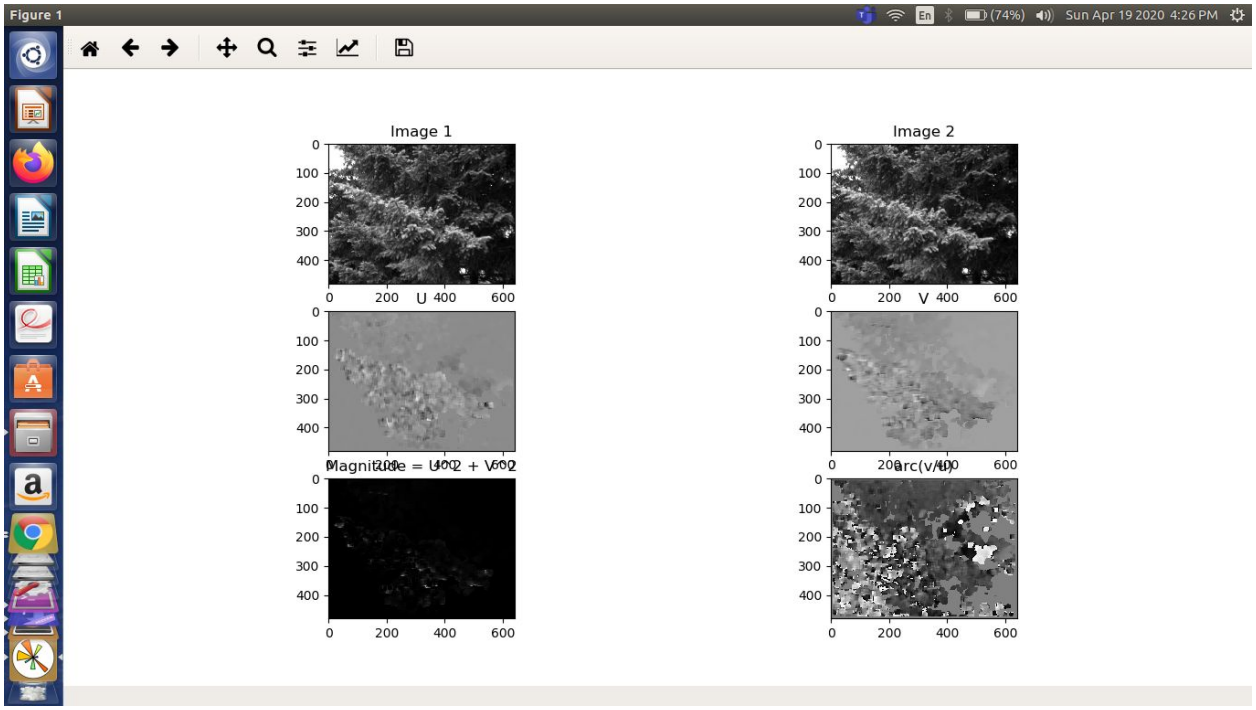
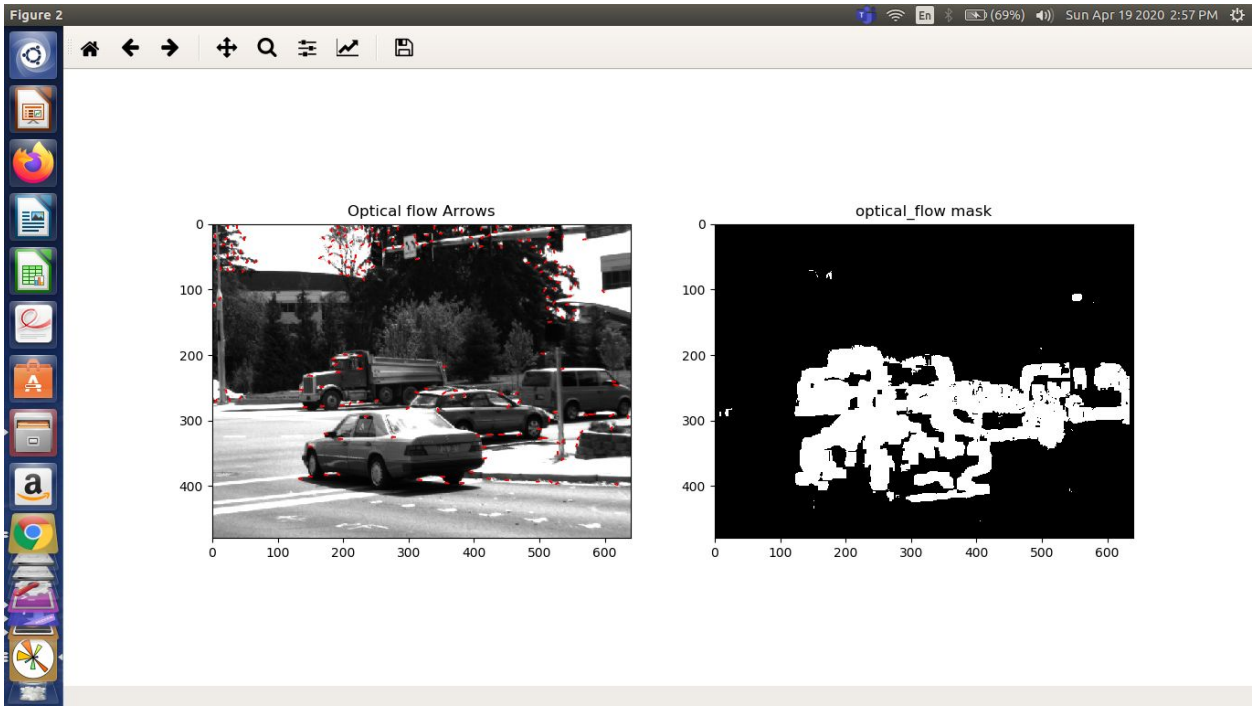


Figure 3
Sun Apr 19 2020 4:26 PM
frame
Threshold
Delta



The following results are for the other image pairs in the dataset provided (in the order described above)



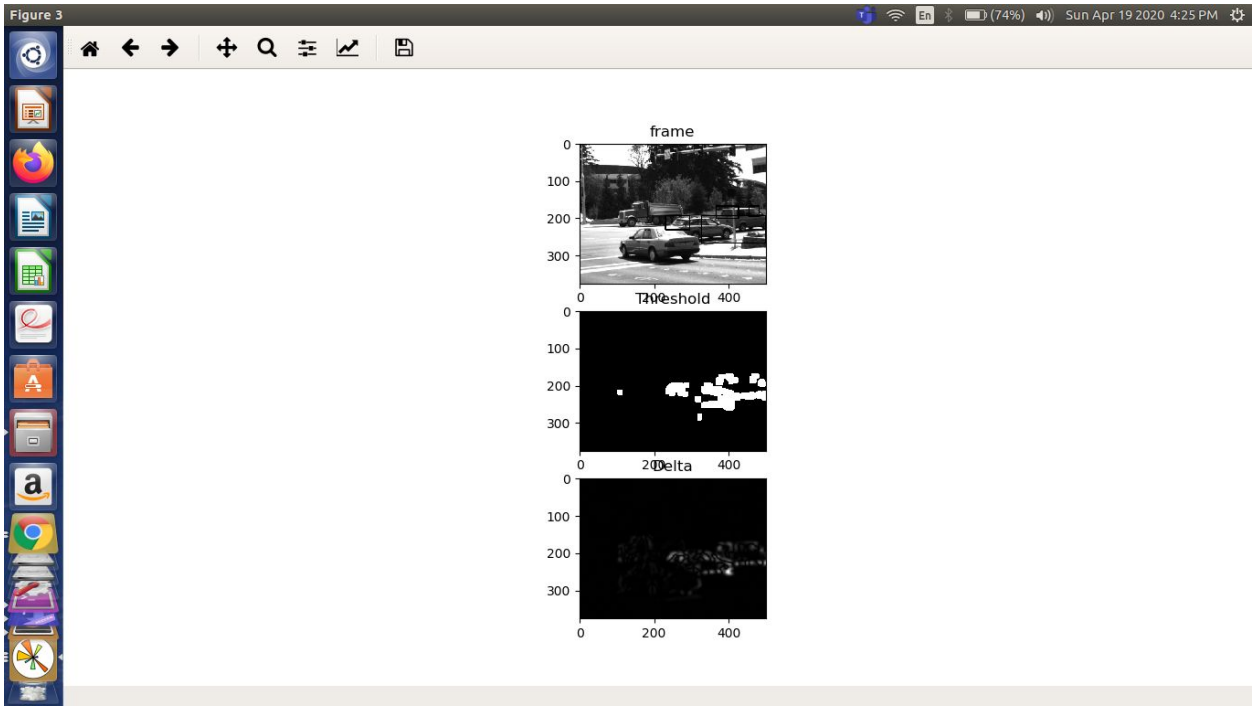


Figure 3
Sun Apr 19 2020 4:25 PM
frame
Threshold
Delta

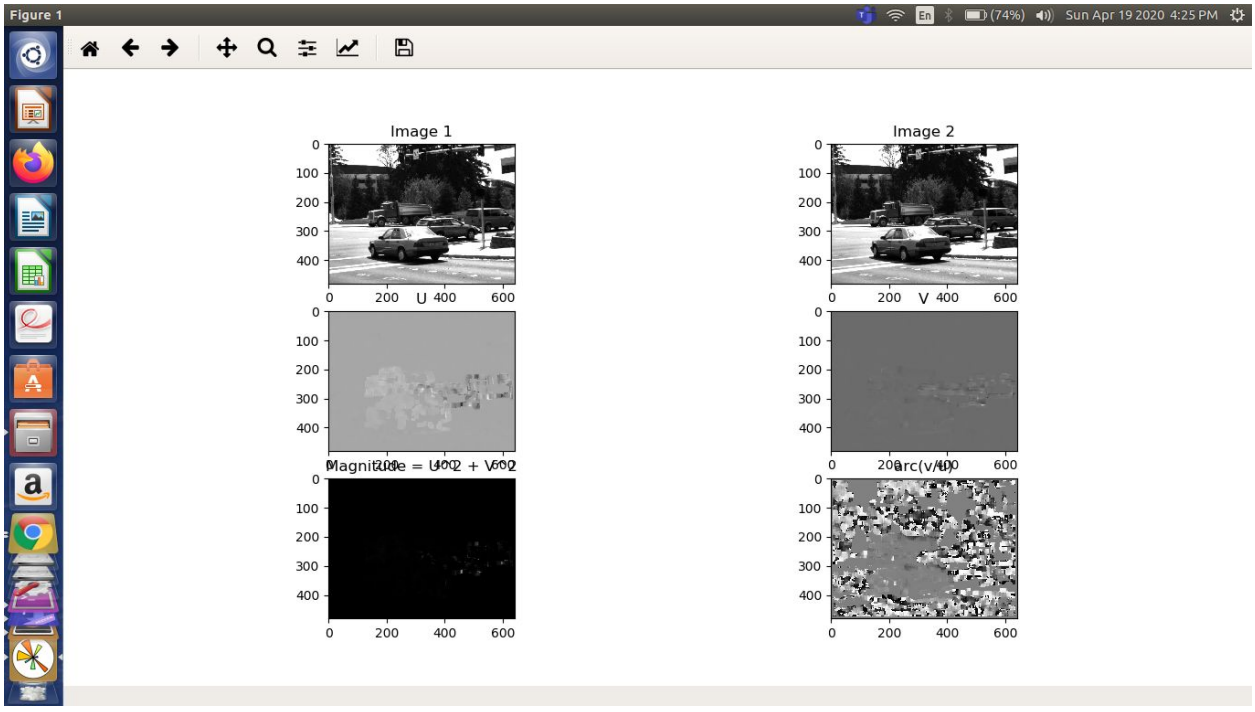
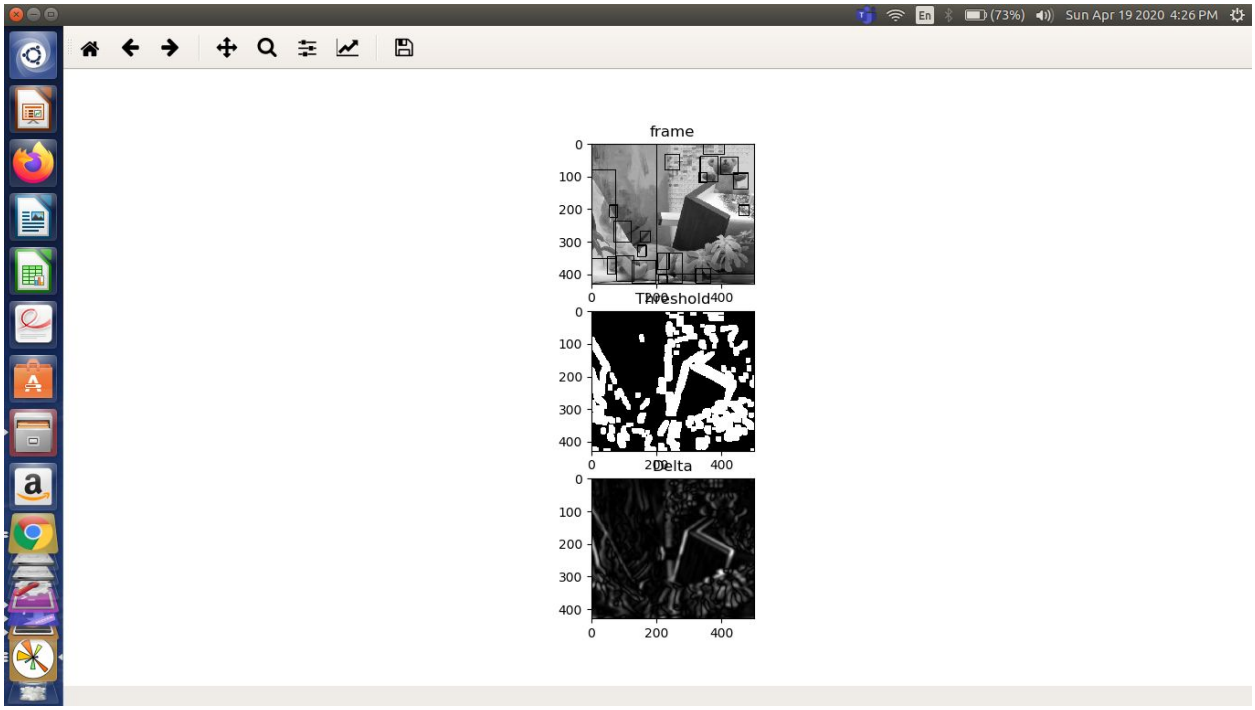


Figure 1
Sun Apr 19 2020 4:25 PM
Image 1
Image 2
U
V
Magnitude = U^2 + V^2
arc(v/u)


frame
Threshold
Delta

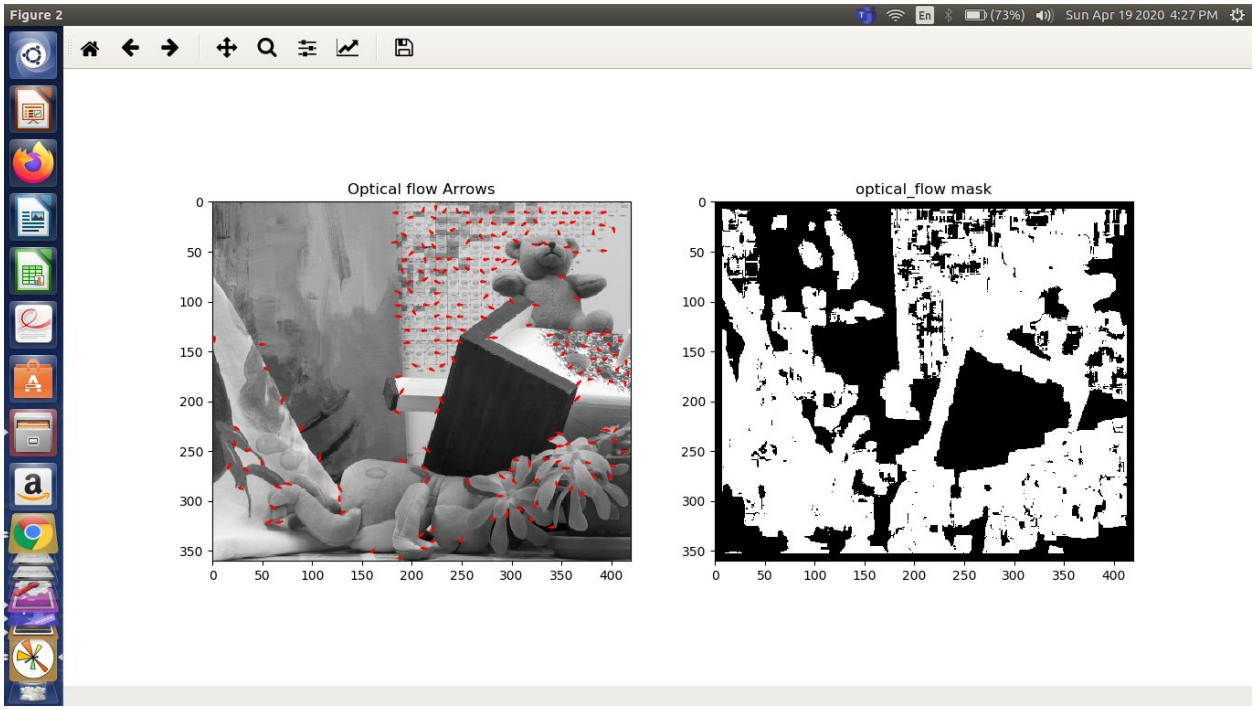


Figure 2
Optical flow Arrows
optical_flow mask

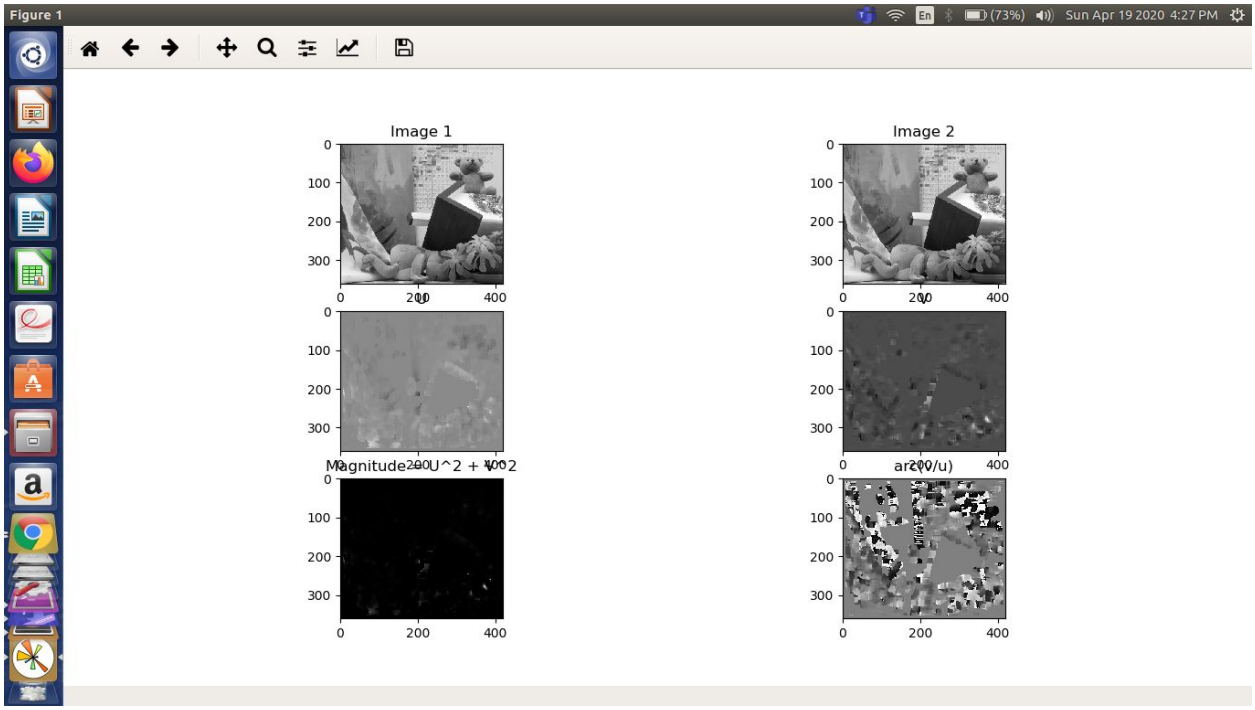


Figure 1
Sun Apr 19 2020 4:27 PM
Image 1
Image 2
Magnitude^U^2 + ^V^2
arctan(v/u)

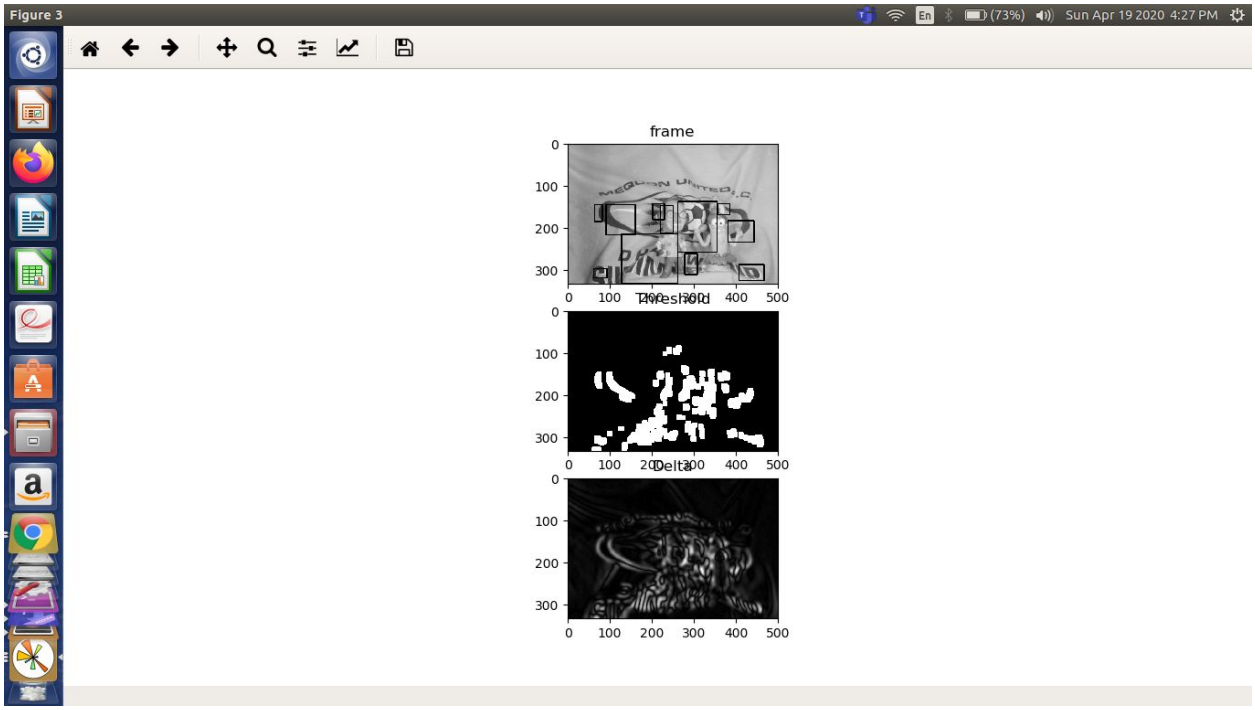


Figure 3
Sun Apr 19 2020 4:27 PM
frame
threshold
Delta

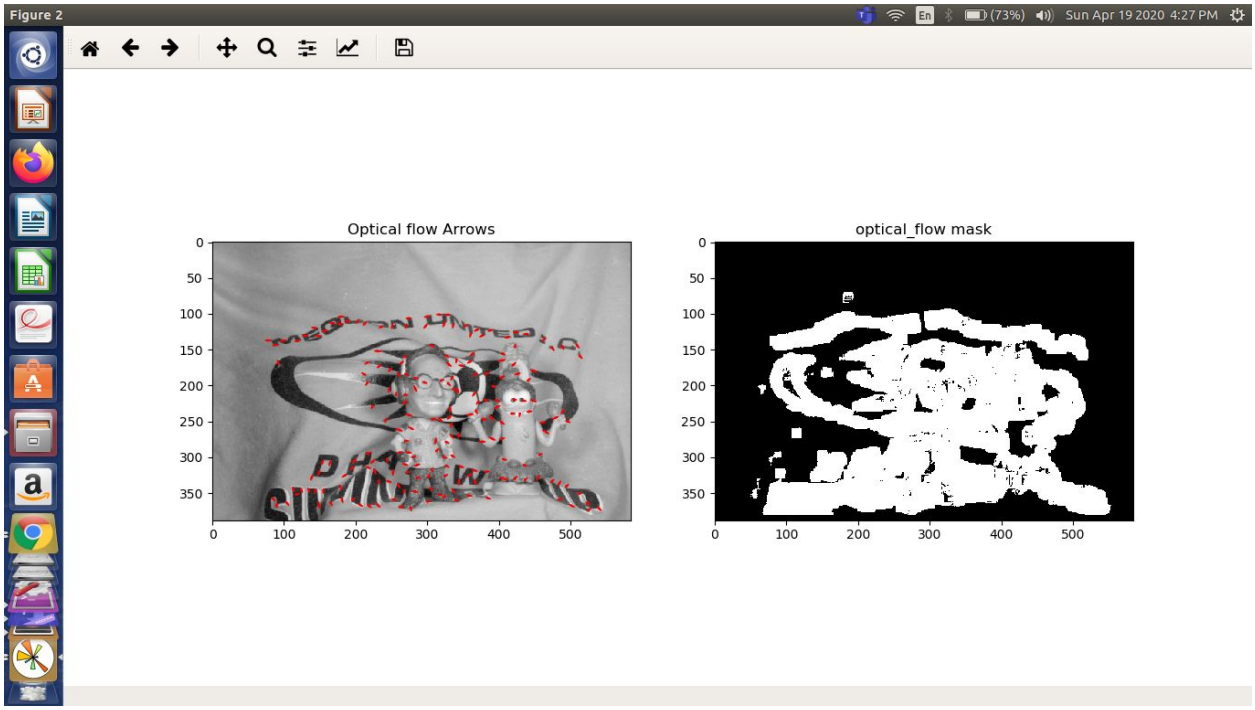


Figure 2
Sun Apr 19 2020 4:27 PM
Optical flow Arrows
optical_flow mask

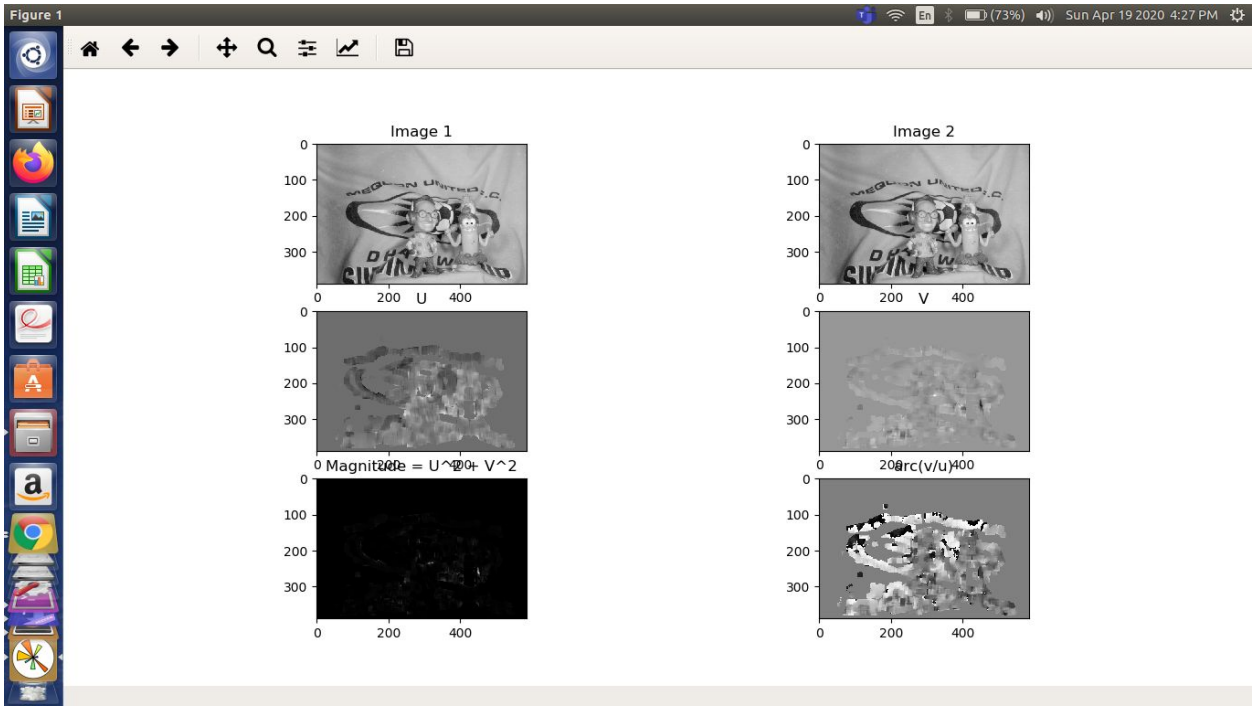


Figure 1
Sun Apr 19 2020 4:27 PM
Image 1
Image 2
U
V
Magnitude = U^2 + V^2
arc(v/u)

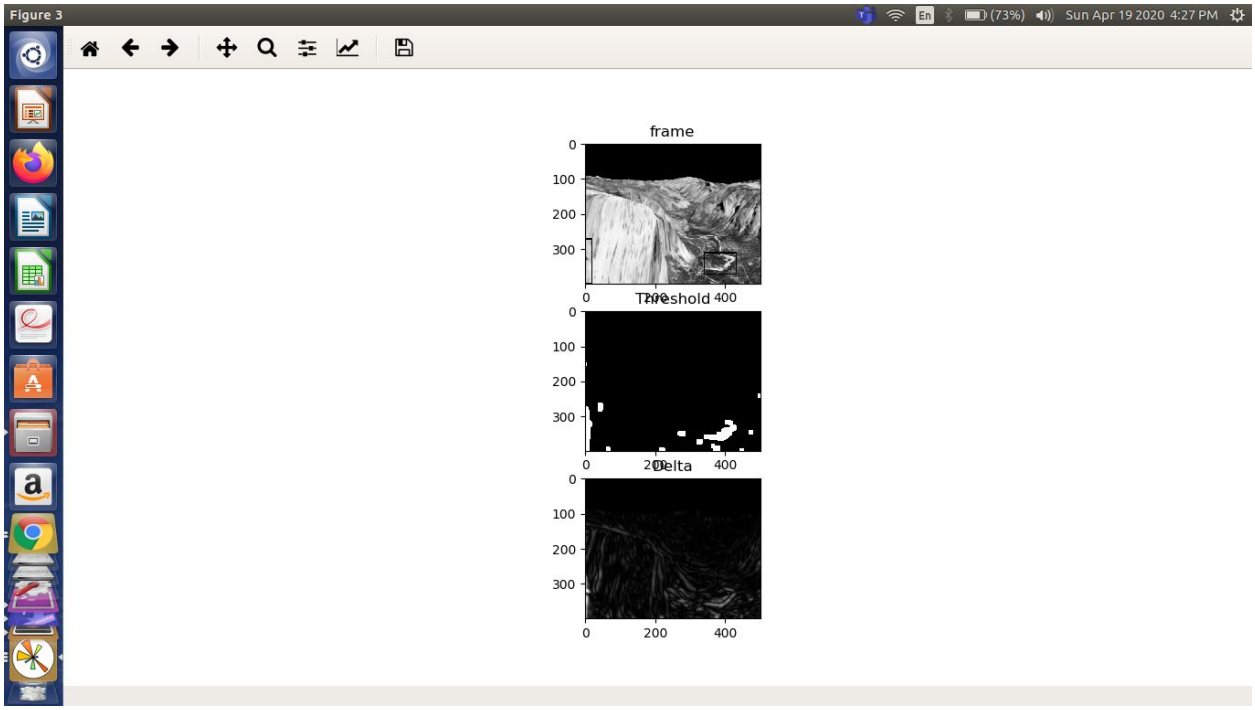


Figure 3
Sun Apr 19 2020 4:27 PM
frame
Threshold
Delta

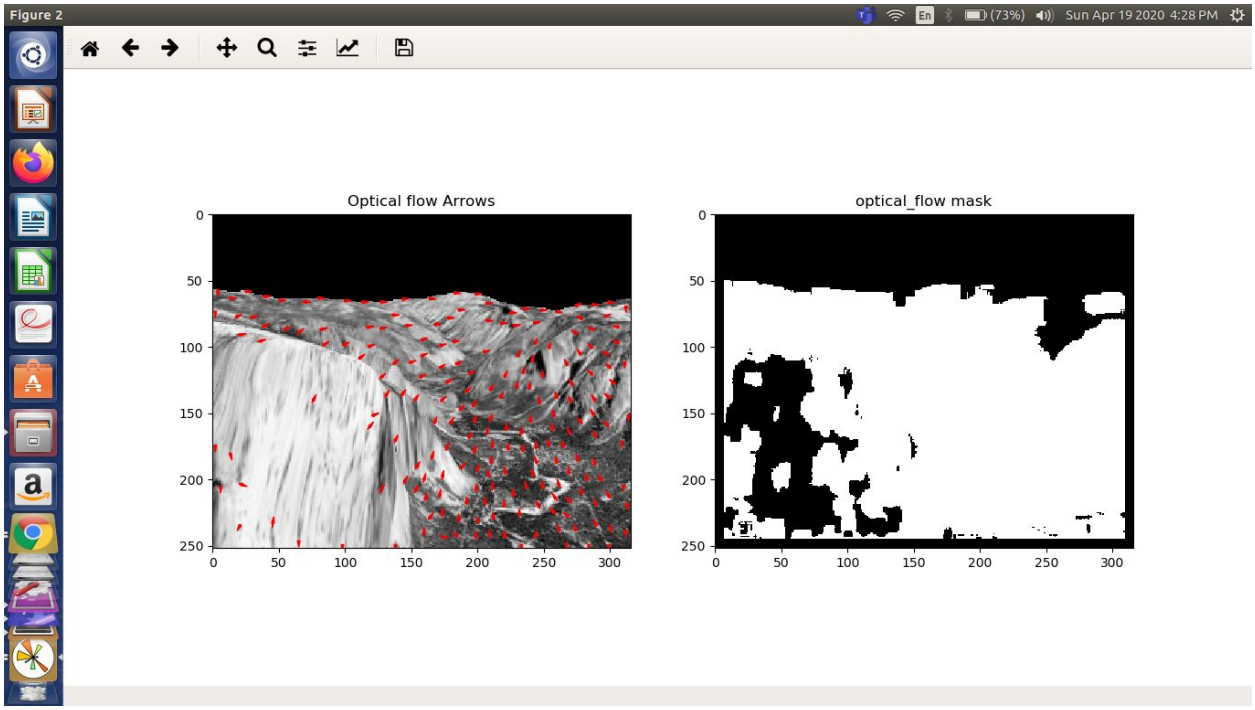


Figure 2
Sun Apr 19 2020 4:28 PM
Optical flow Arrows
optical_flow mask

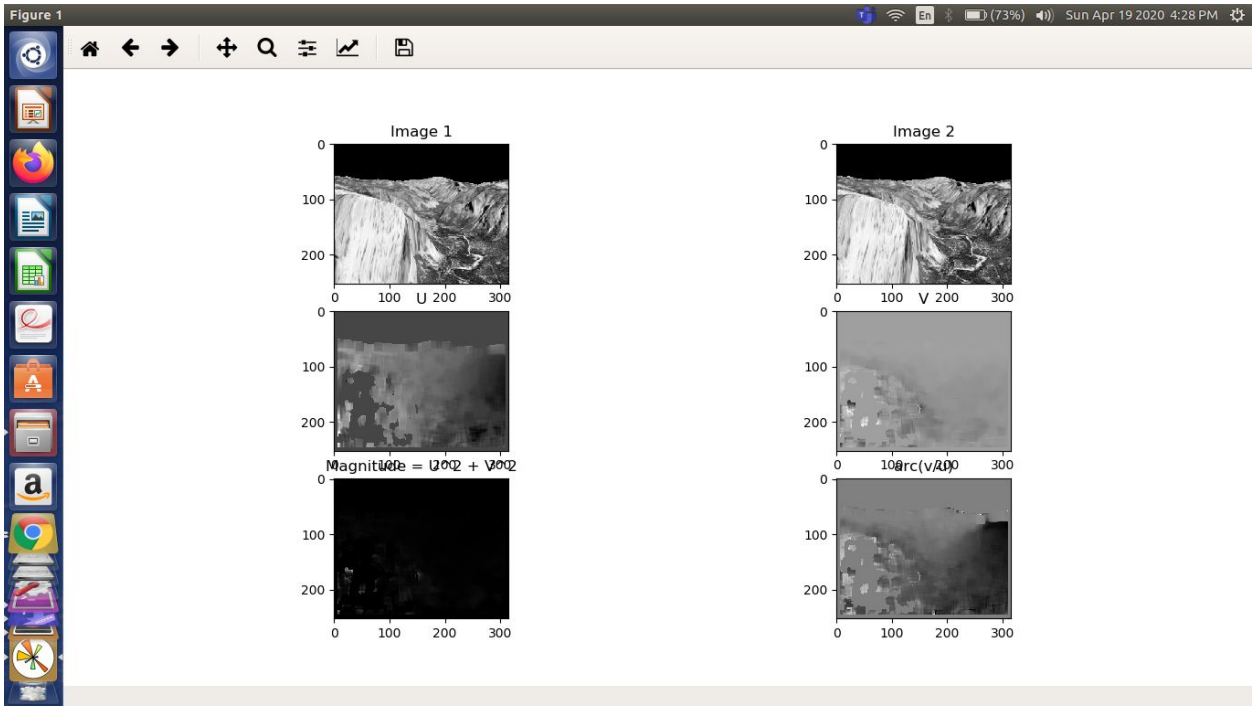
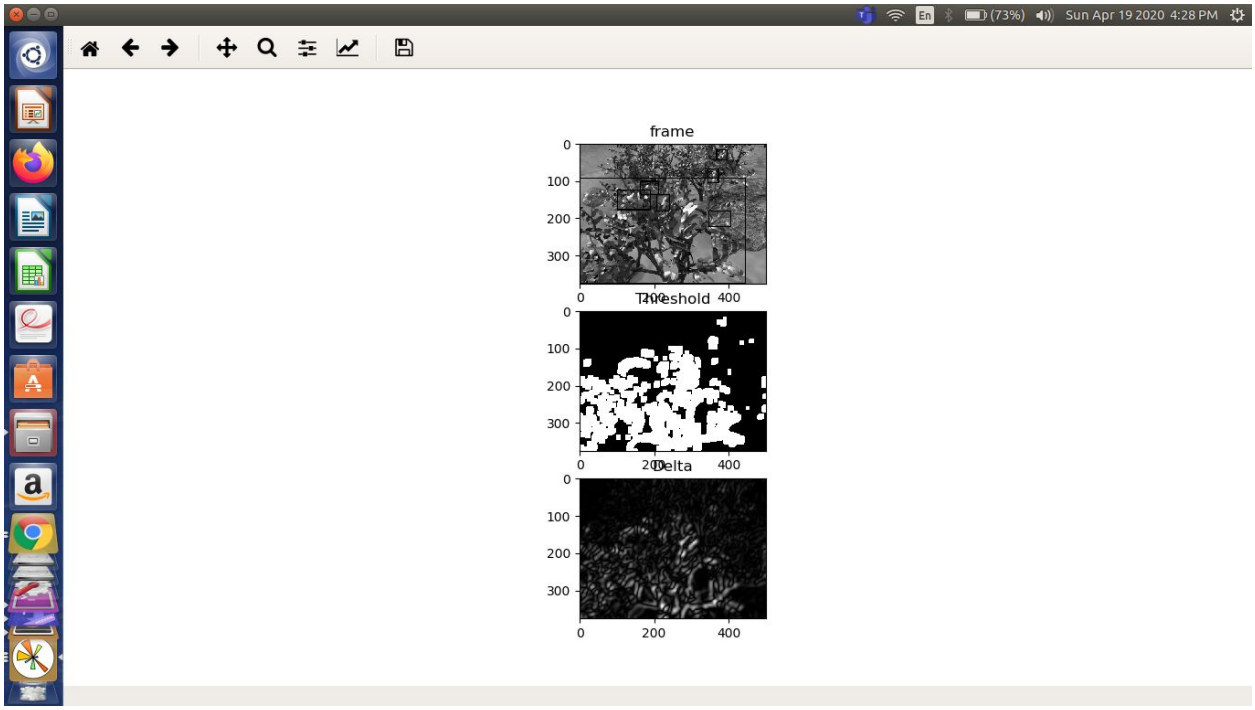


Figure 1
Sun Apr 19 2020 4:28 PM
Image 1
Image 2
U
V
Magnitude = U^2 + V^2
arc(v,u)


Sun Apr 19 2020 4:28 PM
frame
Threshold
Delta

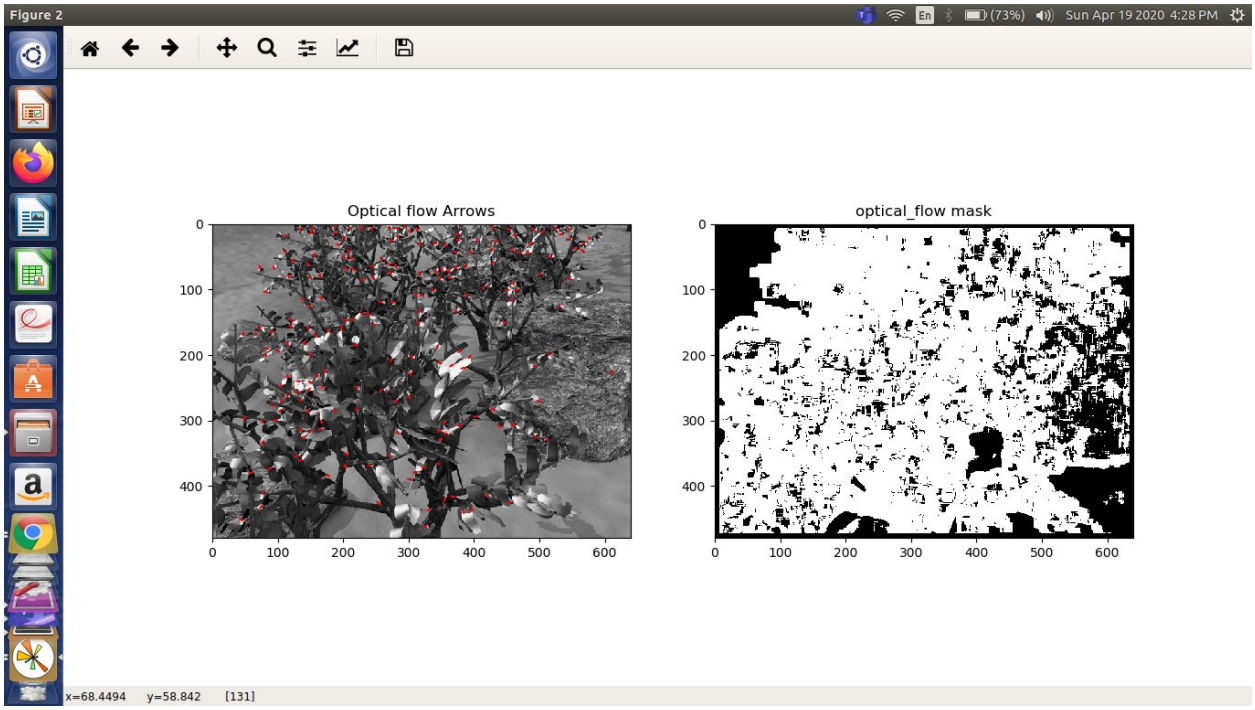


Figure 2
(73%) Sun Apr 19 2020 4:28 PM
Optical flow Arrows
optical_flow mask
0
100
200
300
400
500
600
x=68.4494 y=58.842 [131]

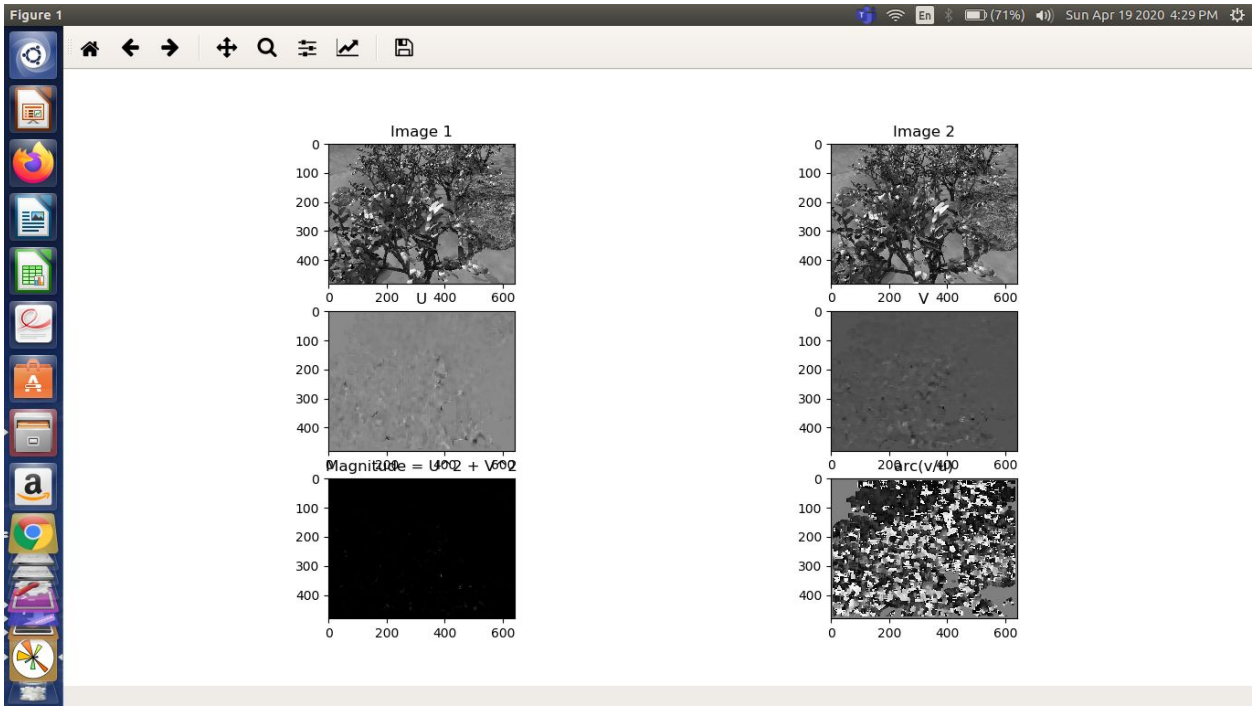
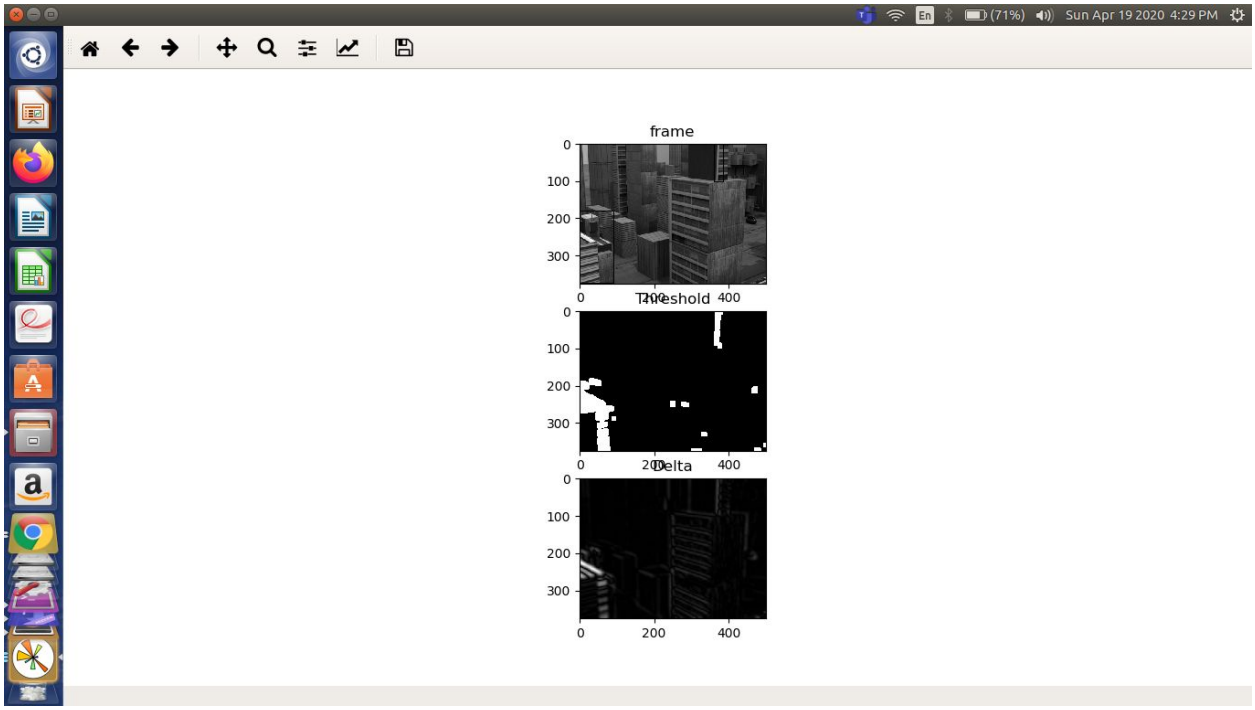
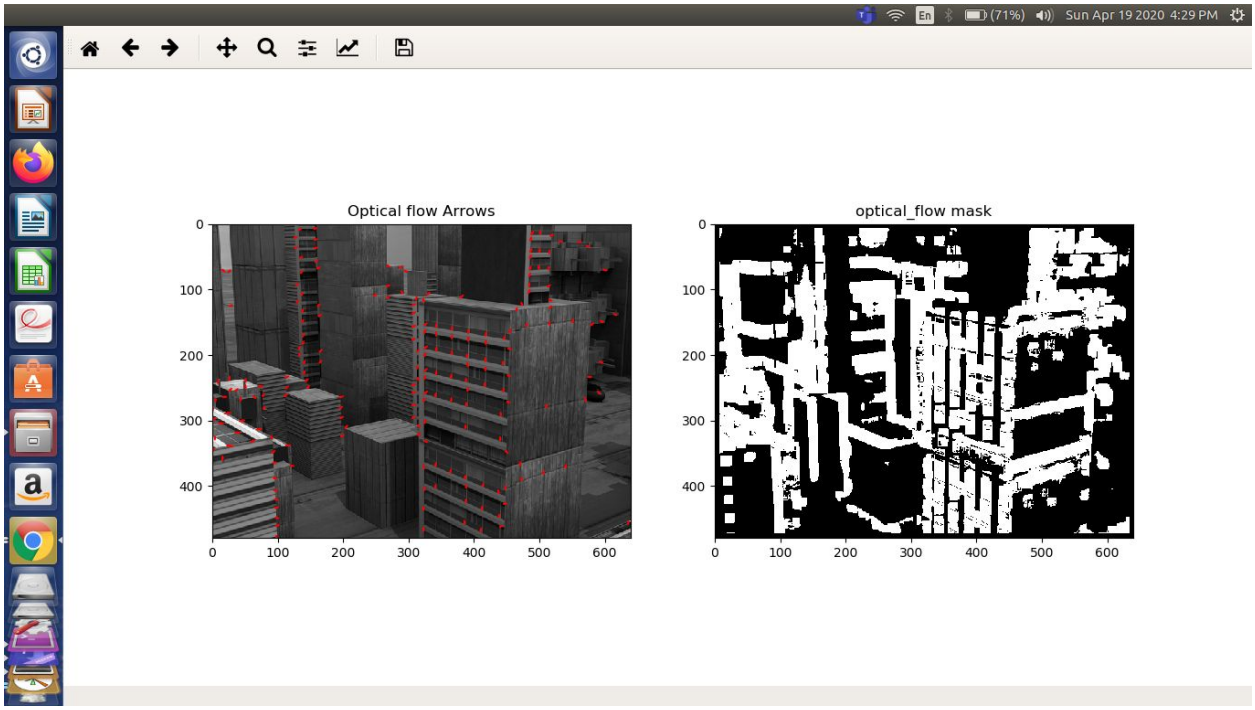


Figure 1
(71%) Sun Apr 19 2020 4:29 PM
Image 1
Image 2
U
V
Magnitude = U^2 + V^2
arc(v/u)
0
100
200
300
400
600


Sun Apr 19 2020 4:29 PM
frame
Threshold
Delta


Sun Apr 19 2020 4:29 PM
Optical flow Arrows
optical_flow mask

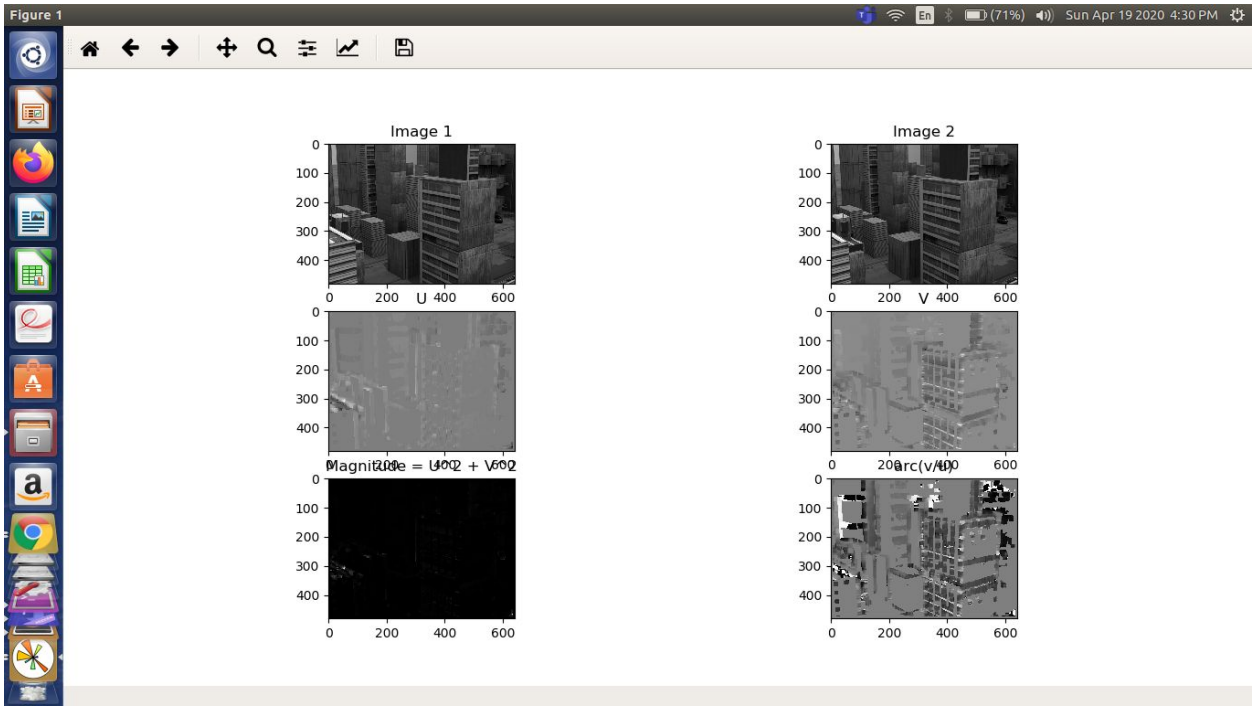


Figure 1
Sun Apr 19 2020 4:30 PM
Image 1
Image 2
U
V
Magnitude = U^2 + V^2
arc(v/u)

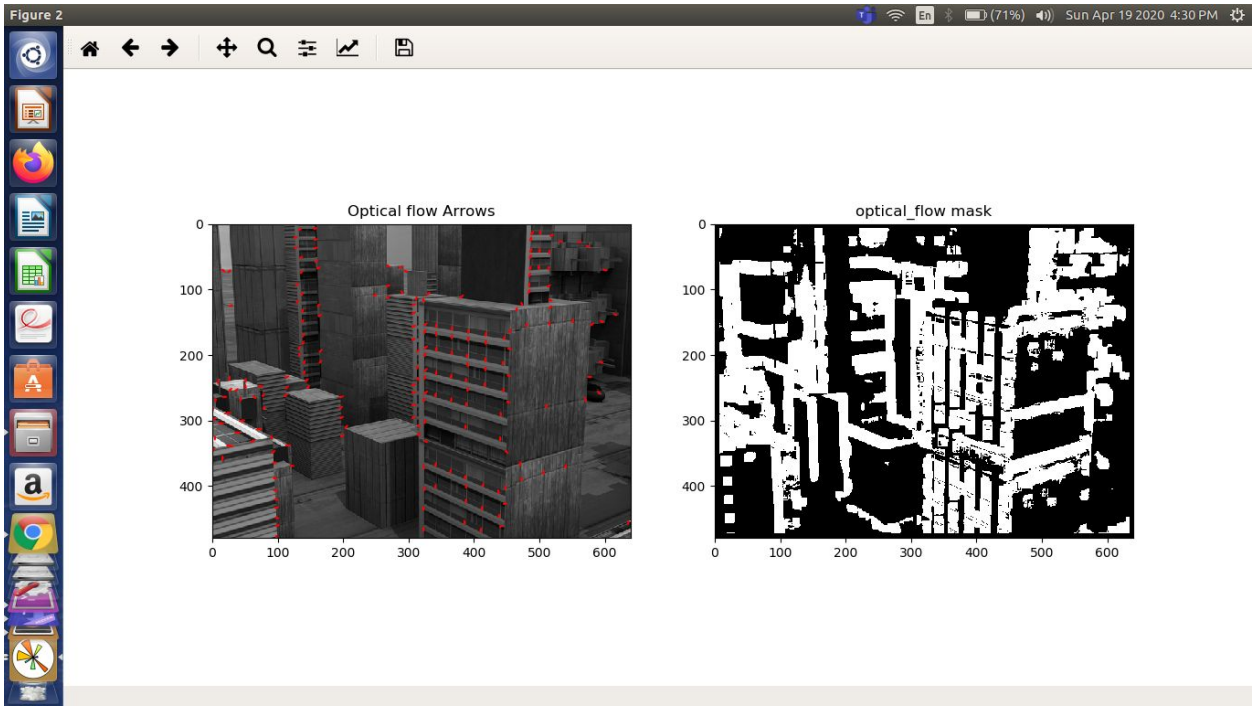


Figure 2
Sun Apr 19 2020 4:30 PM
Optical flow Arrows
optical_flow mask

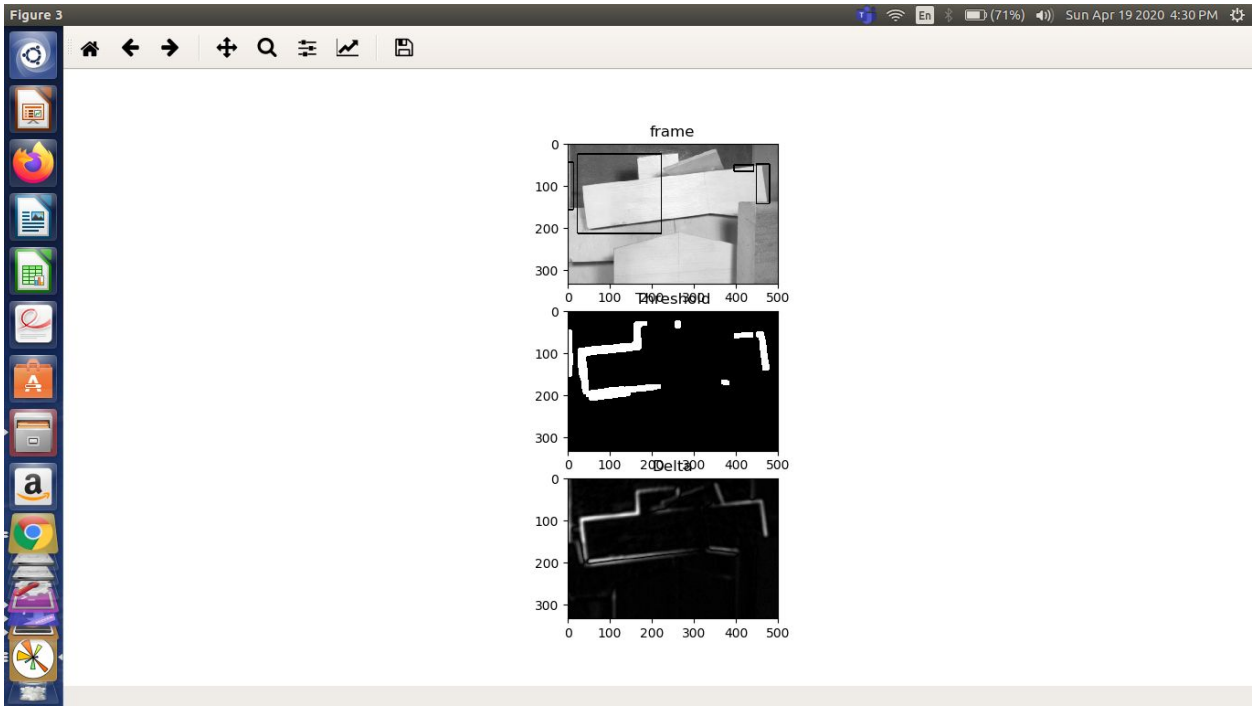


Figure 3
Sun Apr 19 2020 4:30 PM
frame
Threshold
Delta

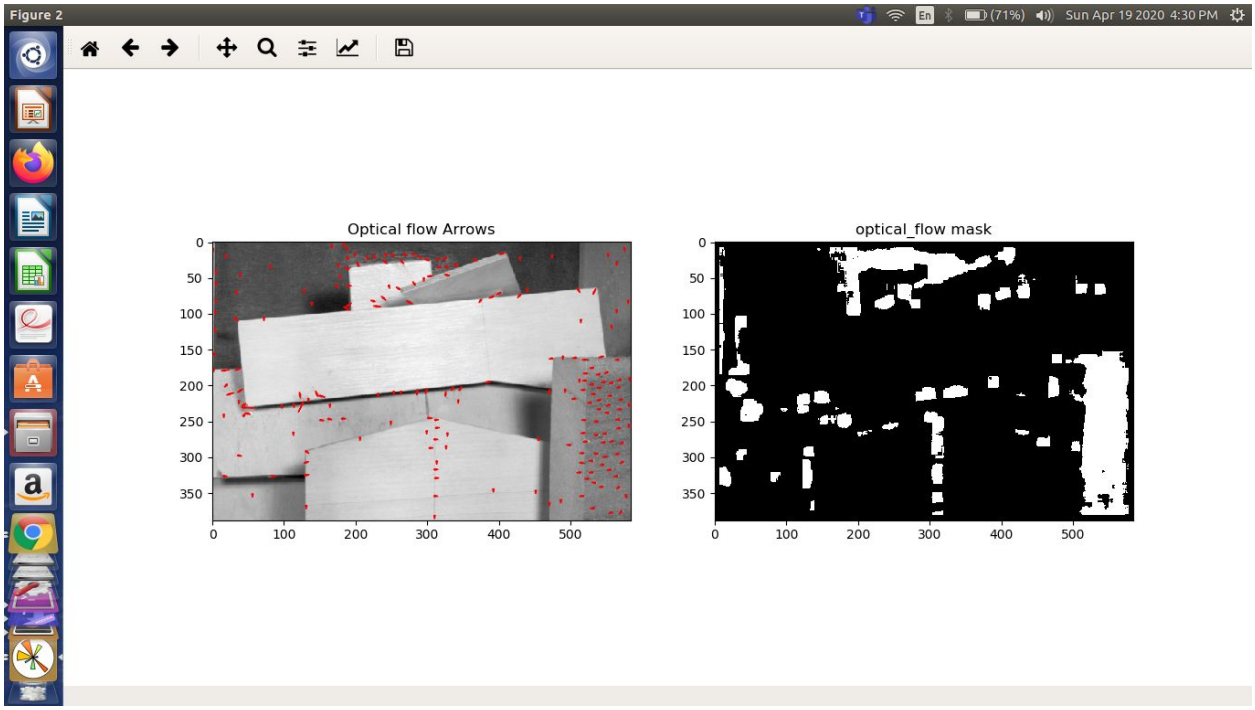


Figure 2
Sun Apr 19 2020 4:30 PM
Optical flow Arrows
optical_flow mask

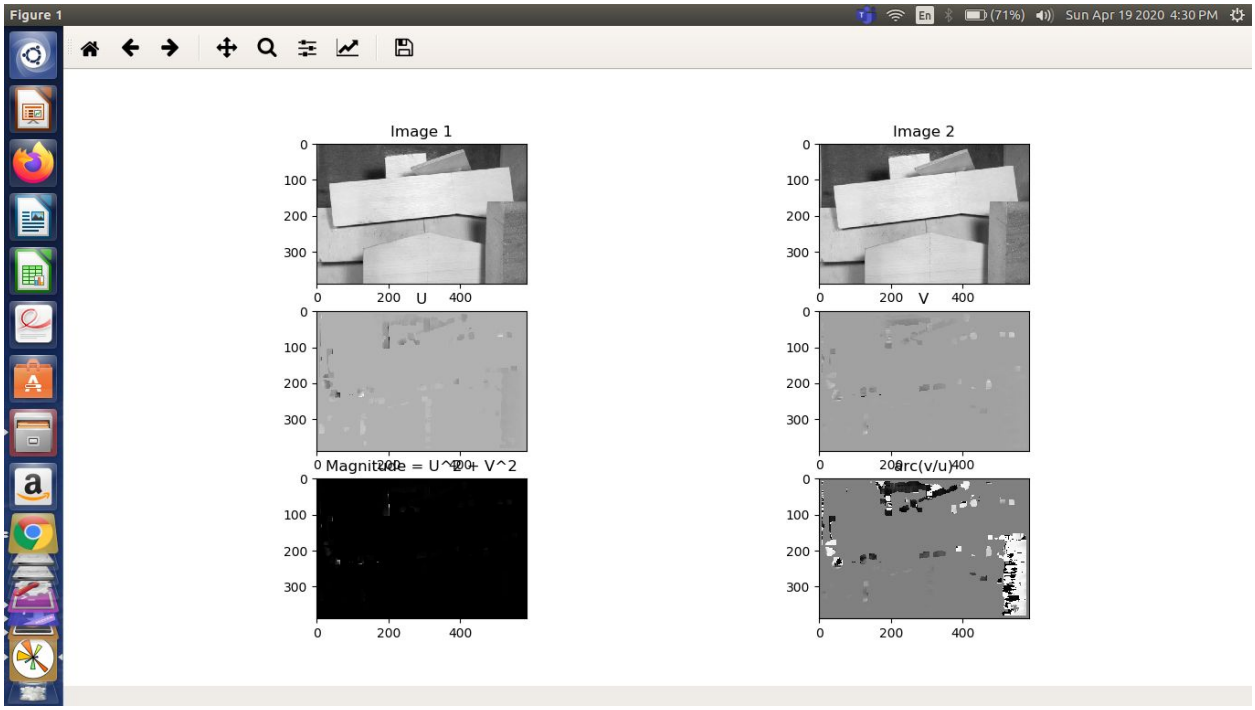


Figure 1
(71%) Sun Apr 19 2020 4:30 PM
Image 1
U
Magnitude = U^2 + V^2
Image 2
V
arc(v/u)

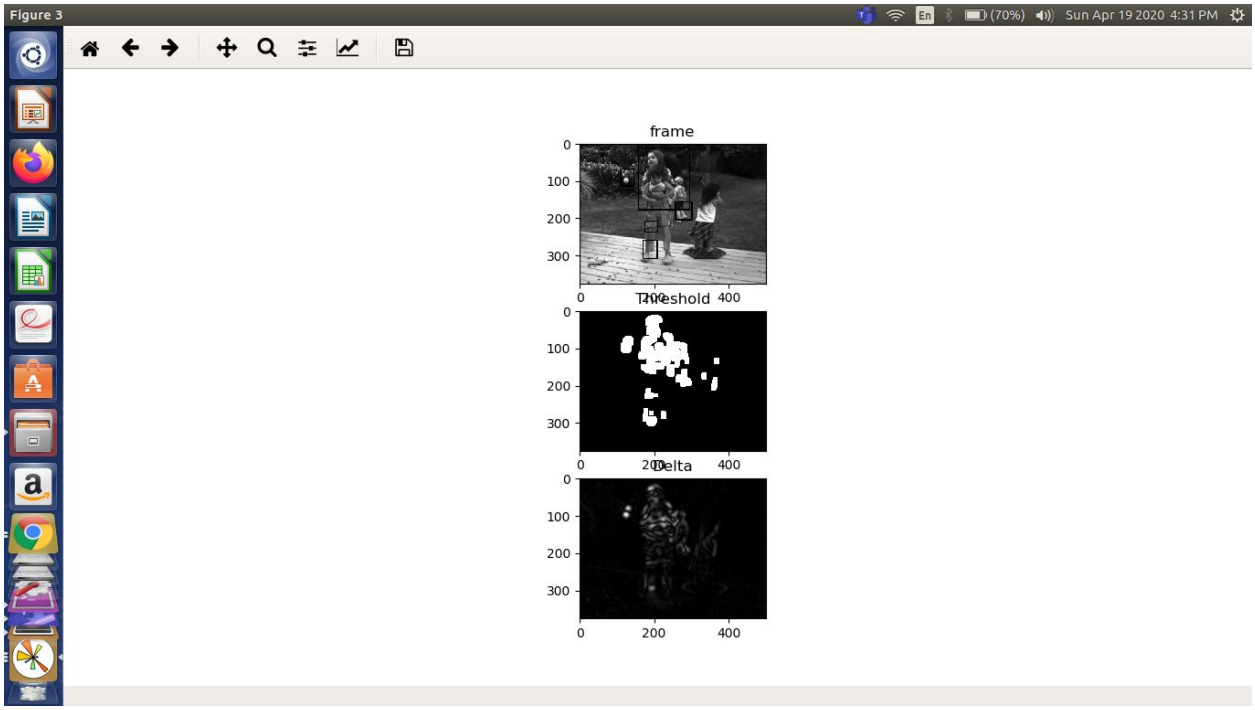


Figure 3
(70%) Sun Apr 19 2020 4:31 PM
frame
Threshold
Delta

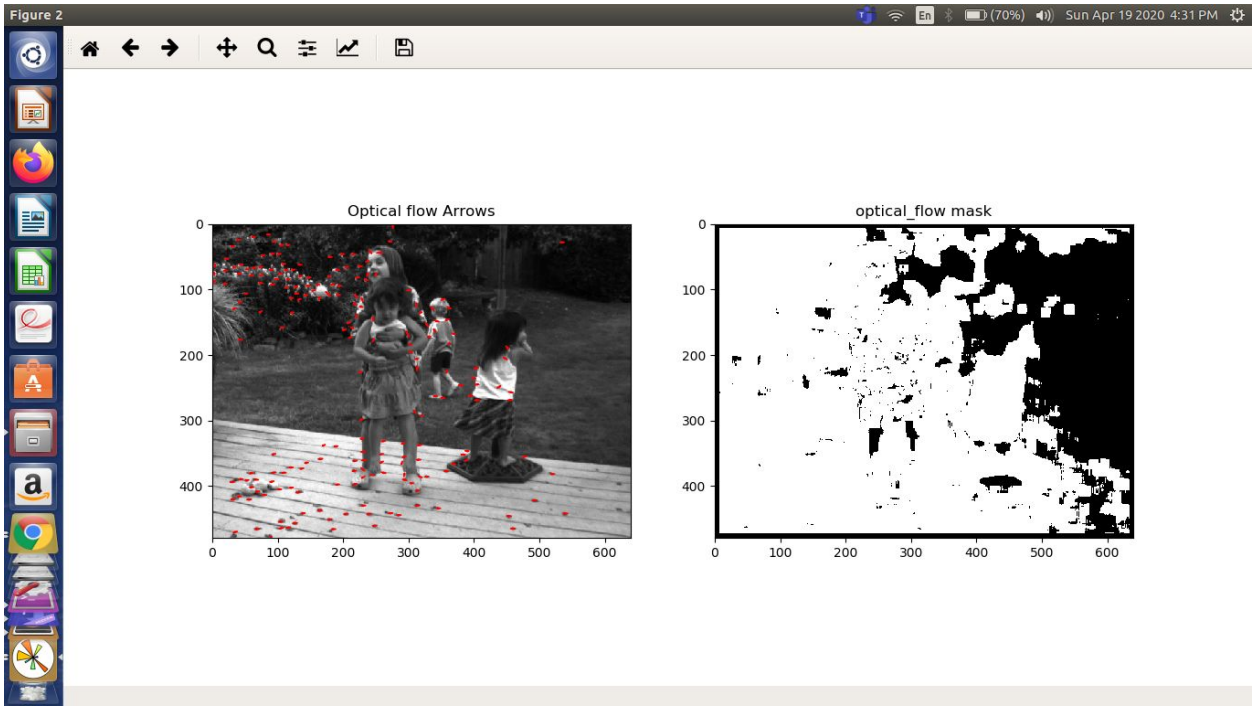


Figure 2
Sun Apr 19 2020 4:31 PM
Optical flow Arrows
optical_flow mask

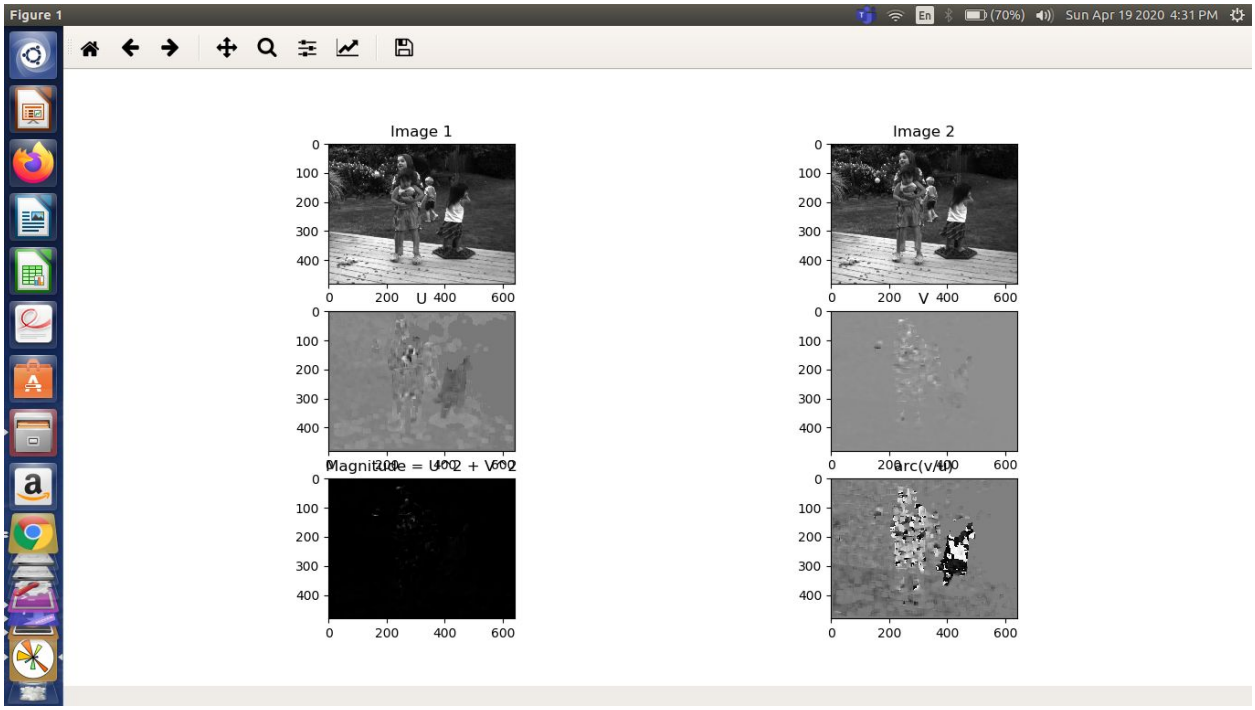
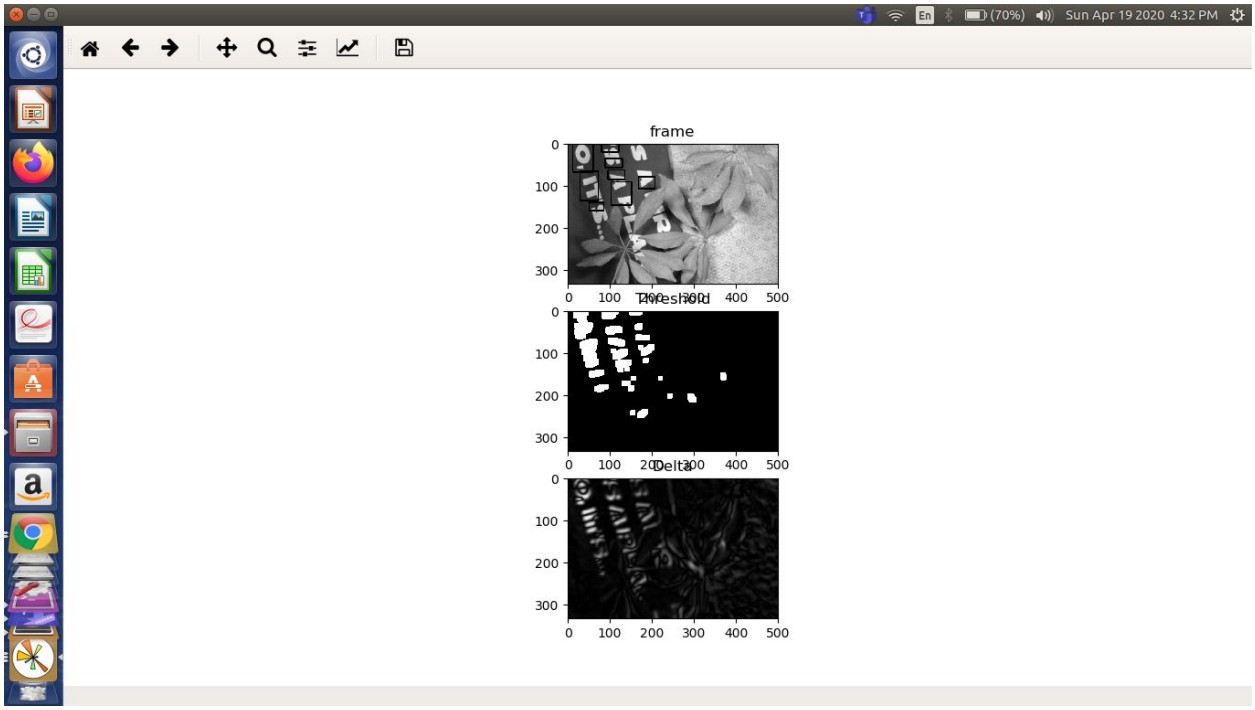


Figure 1
Sun Apr 19 2020 4:31 PM
Image 1
Image 2
U
V
Magnitude = U^2 + V^2
arc(v/u)


Sun Apr 19 2020 4:32 PM
frame
Threshold
Delta

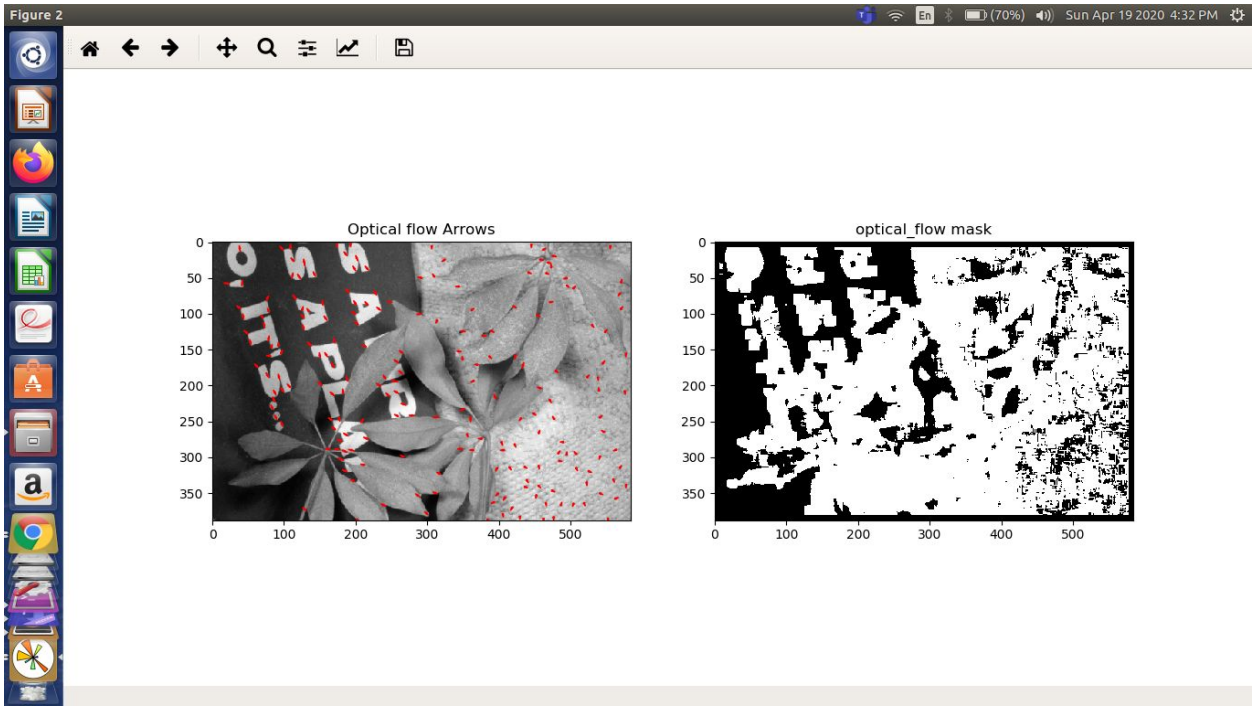


Figure 2
Sun Apr 19 2020 4:32 PM
Optical flow Arrows
optical_flow mask

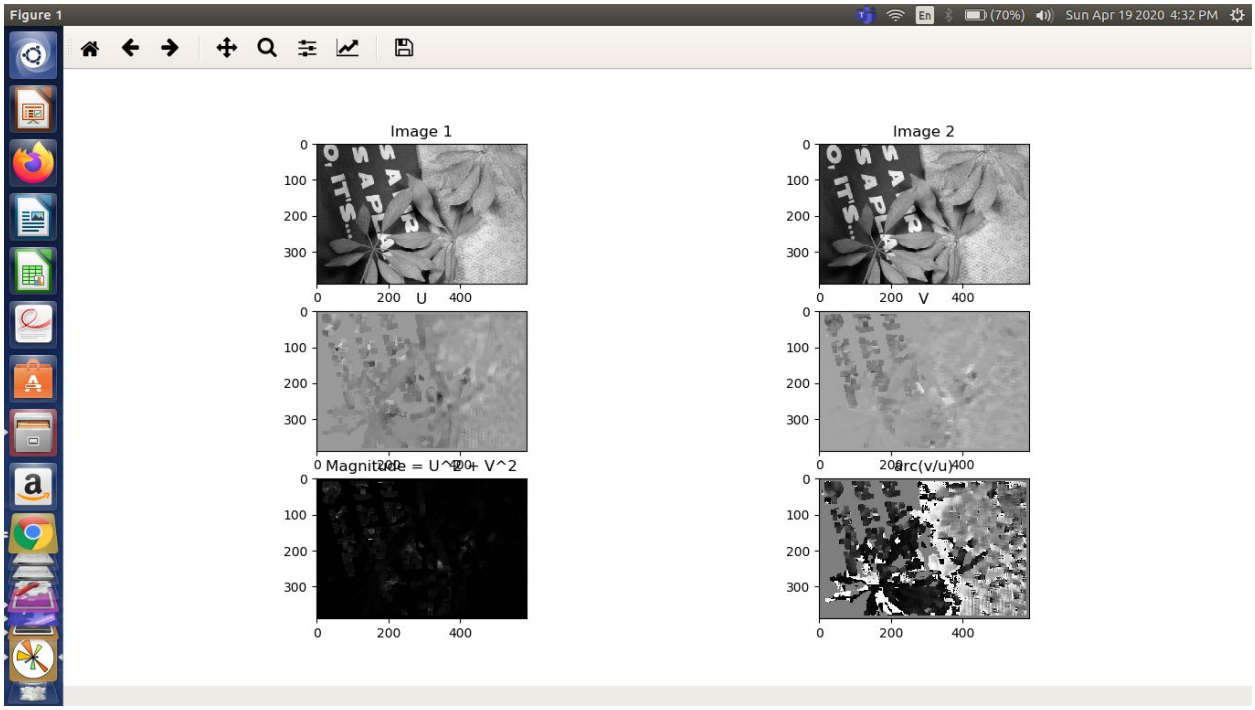
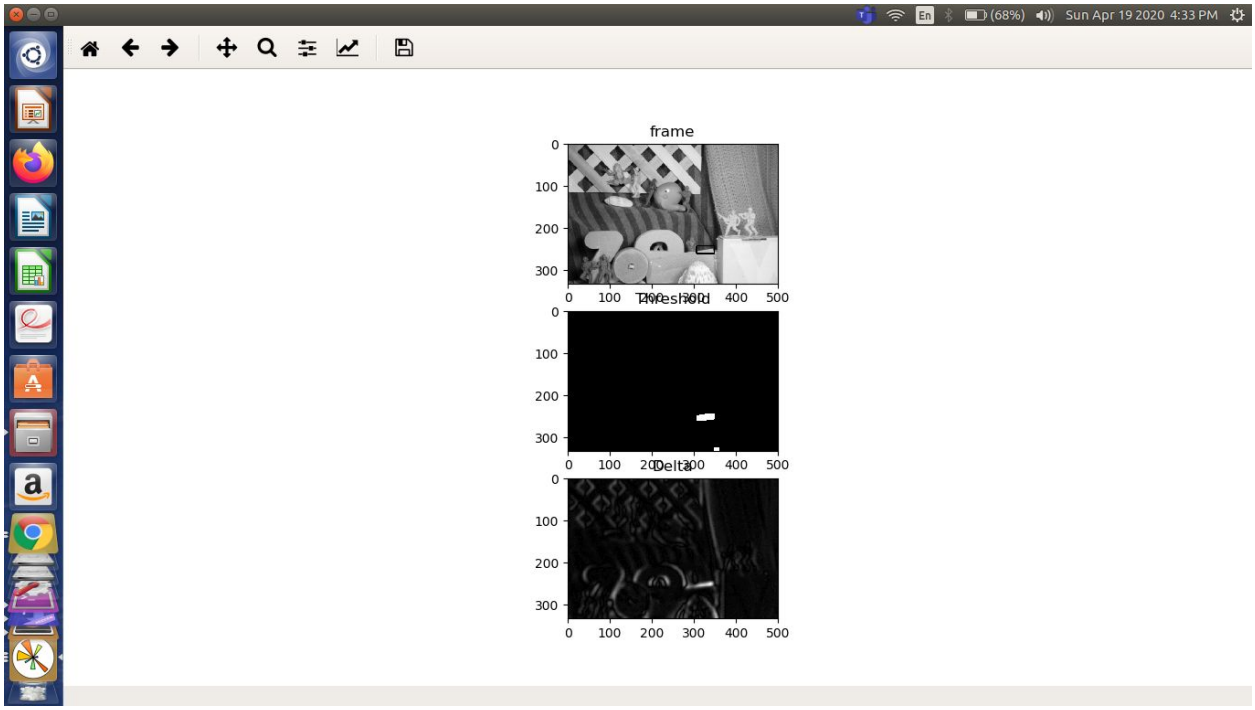
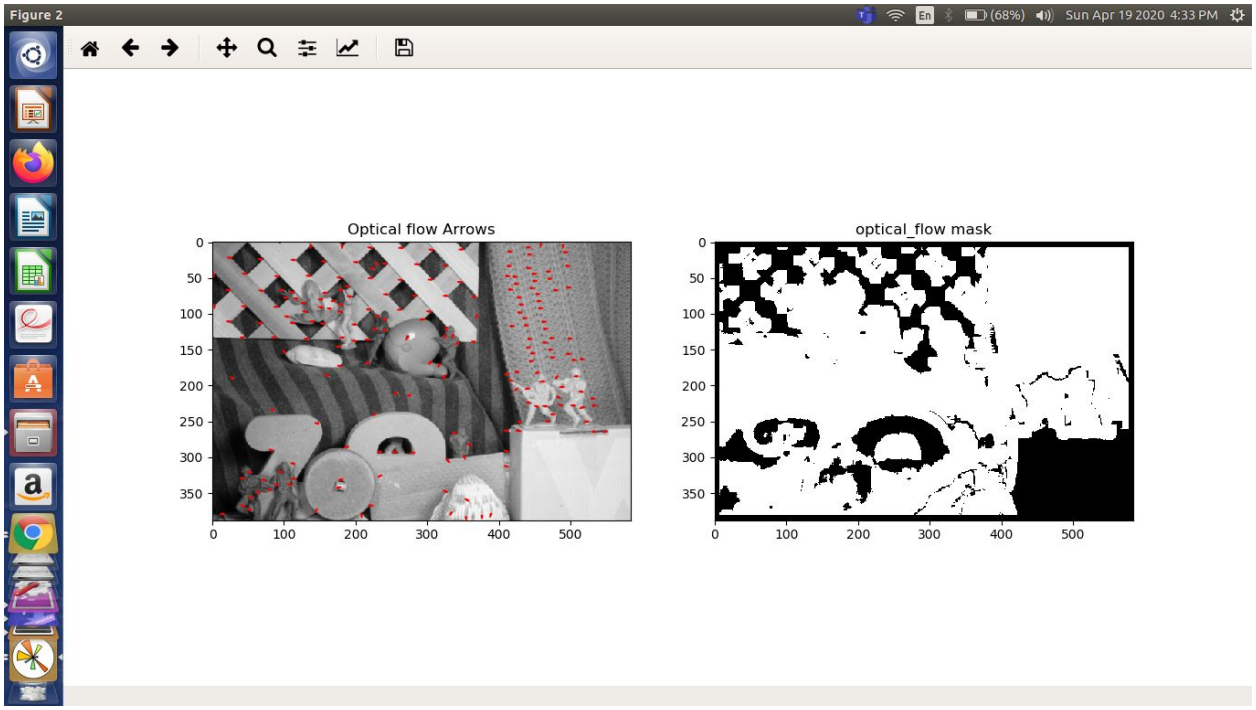
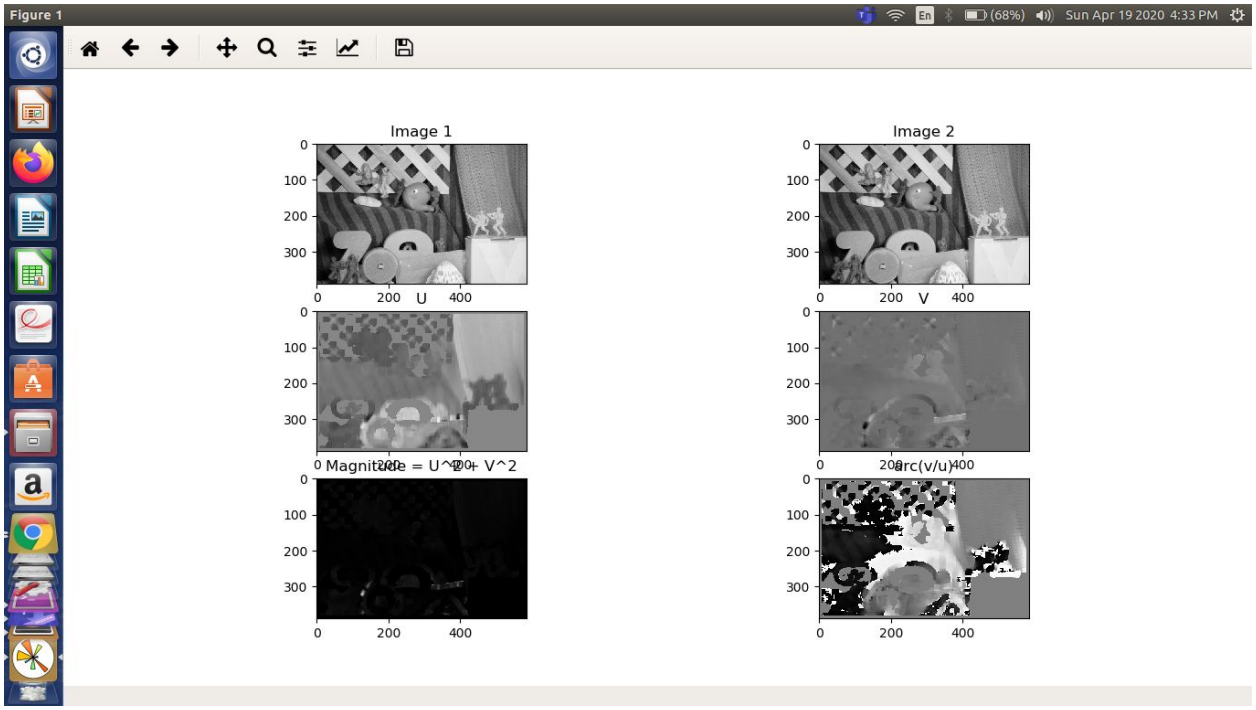


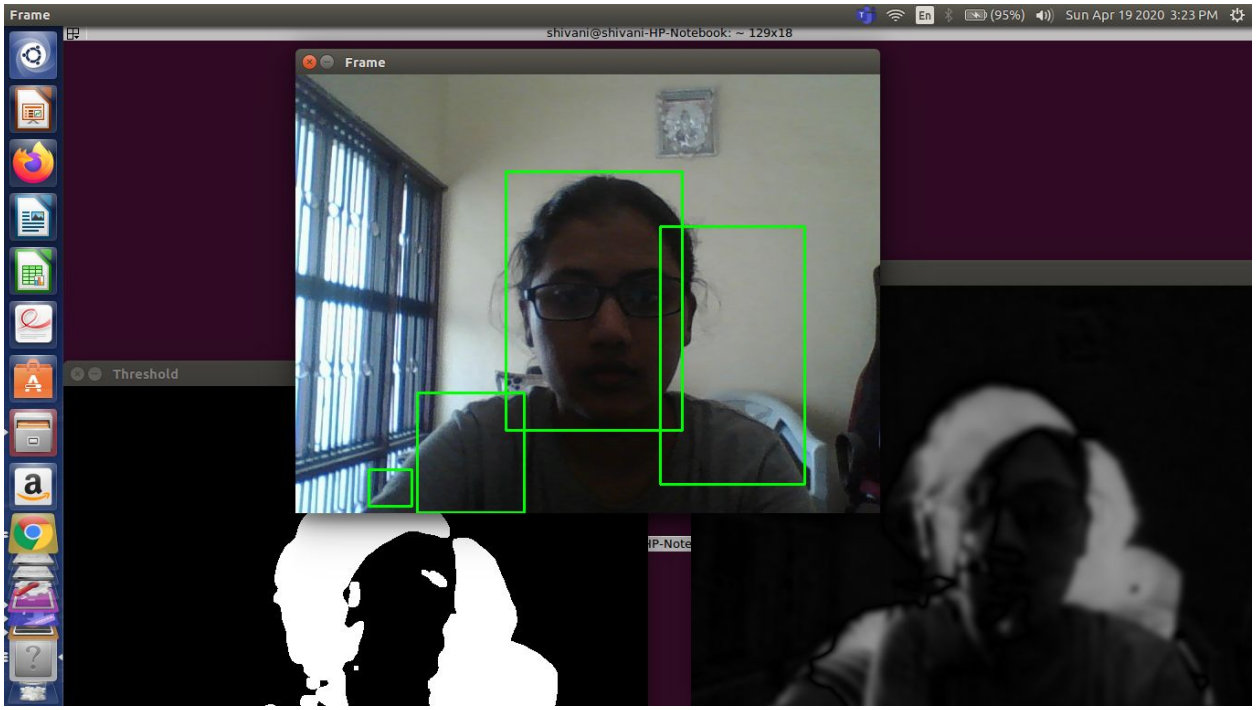
Figure 1
Sun Apr 19 2020 4:32 PM
Image 1
U
Magnitude = U^2 + V^2
Image 2
V
arc(v/u)


Sun Apr 19 2020 4:33 PM
frame
threshold
Delta





Results/Screenshots from Webcam (in the order of captured frames). The images are being displayed with a lag. These are the screenshots taken while streaming webcam.


Frame
shivani@shivani-HP-Notebook: ~ 129x18
Frame
Threshold

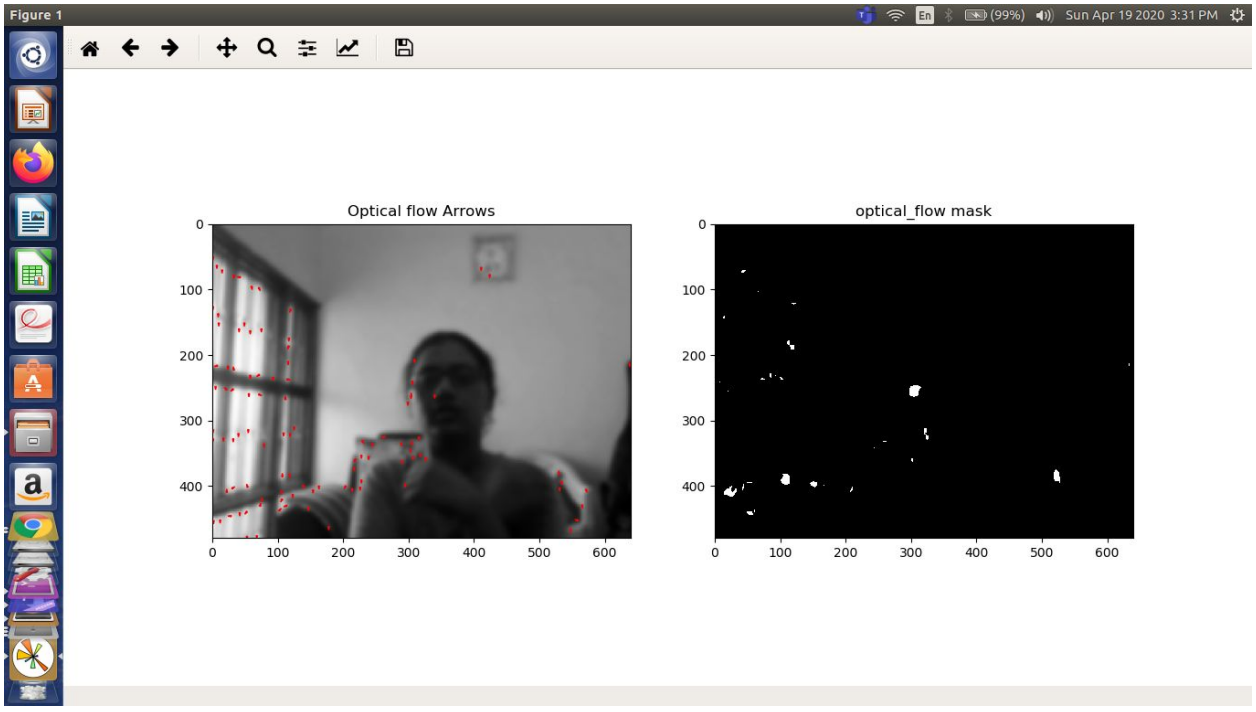


Figure 1
Optical flow Arrows
optical_flow mask
0
100
200
300
400
500
600

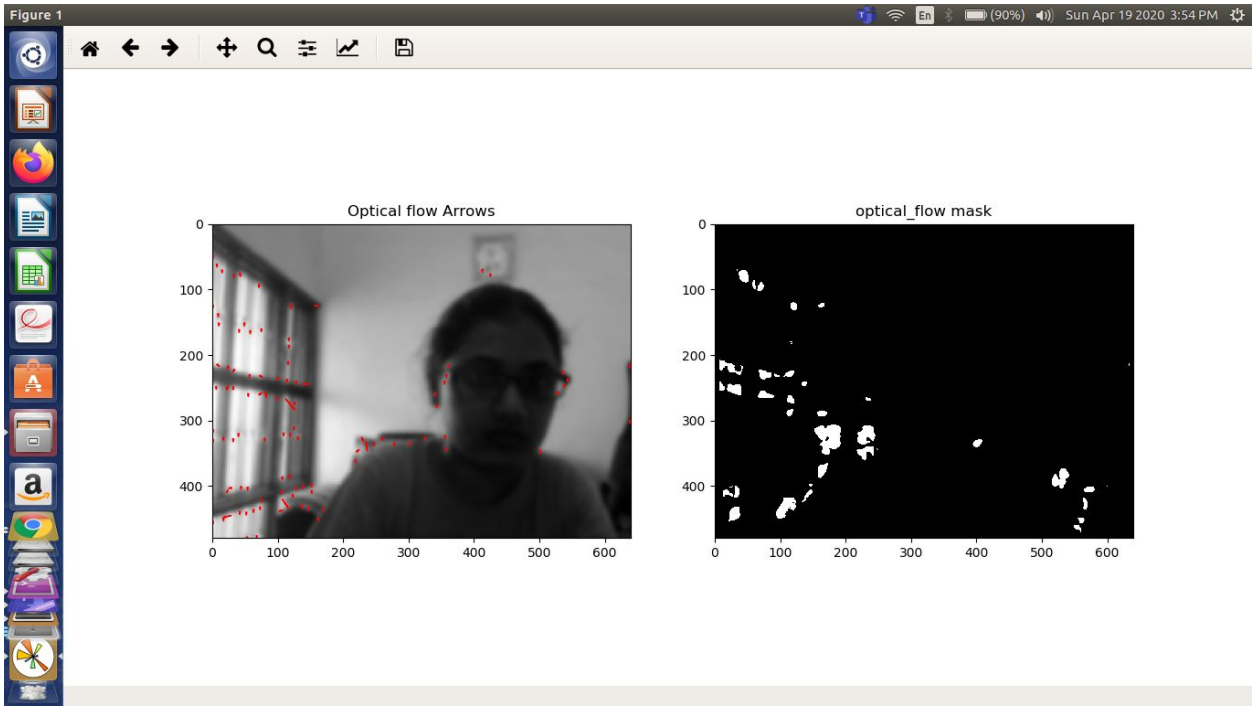
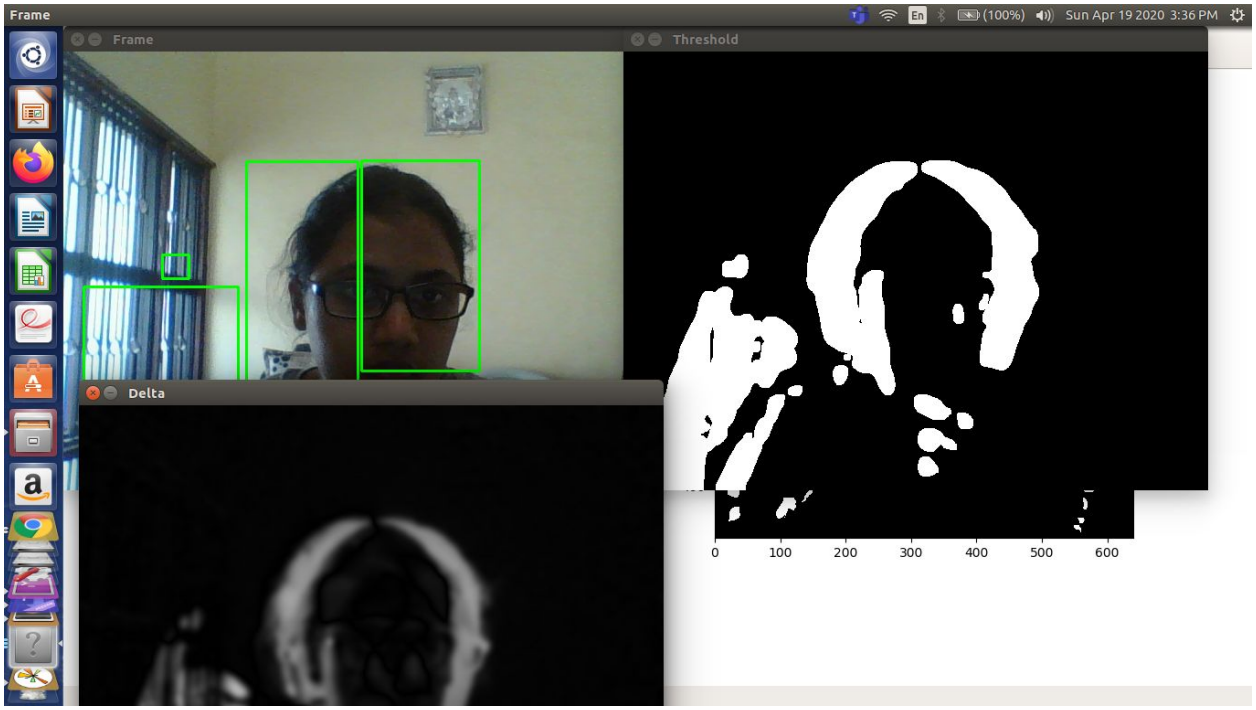
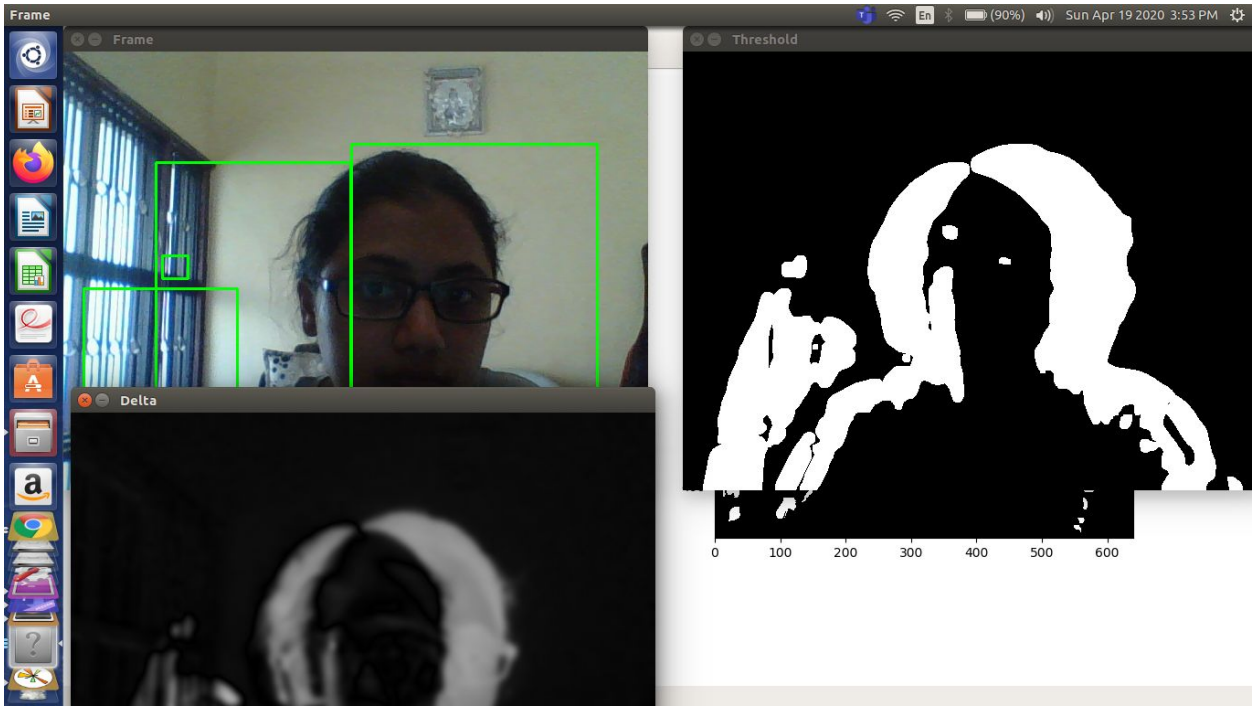
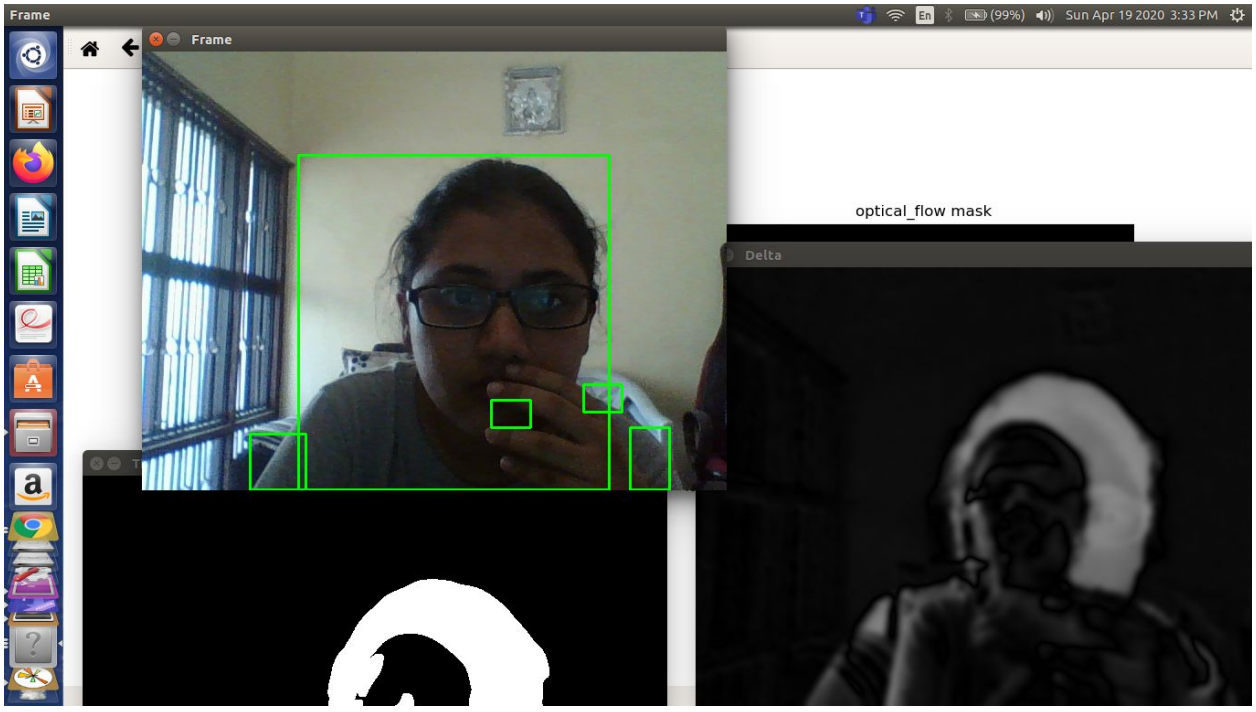


Figure 1
Sun Apr 19 2020 3:54 PM
Optical flow Arrows
optical_flow mask


Frame
Sun Apr 19 2020 3:36 PM
Frame
Threshold
Delta





Please find all results (images and videos) in this drive link
https://drive.google.com/open?id=1Vrvwf3nVgJvRjgEvmfjb7h3P4JLeqgBg