Part 3

For this part of the project, we have tried various approaches. In the end, we have used dense optical flow estimation in the OpenCV to calculate the movement of each pixel. Then, we have found which pixels belong to which circles by segmenting the objects by their color.

Our method is as follows:

1. Read the video and the frames.

```
vid = mpy.VideoFileClip("TermProjectFiles/Part3/movie_001.avi")
frame_count = vid.reader.nframes
video_fps = vid.fps

vid_frame = list()
for i in range(frame_count):
    vid_frame.append(vid.get_frame(i*1.0/video_fps))
```

2. For each frame, use colors to find object coordinates.

```
def give_coord(mask_arr):
    return int(np.median(np.where(mask_arr)[0])),
int(np.median(np.where(mask_arr)[1]))

def find_circles(img):

    img_cp = deepcopy(img)
    img_cp = cv.cvtColor(img_cp, cv.COLOR_RGB2HSV)
    green_mask = cv.inRange(img_cp, (60, 25, 25), (80, 255,255))
    blue_mask = cv.inRange(img_cp, (105, 25, 25), (125, 255,255))
    red_mask = cv.inRange(img_cp, (165, 50, 50), (180, 255,255))
    pink_mask = cv.inRange(img_cp, (150, 25, 25), (170, 255,255))

    green_imask = green_mask > 0
    green = np.zeros_like(img, np.uint8)
    green[green_imask] = img[green_imask]

    blue_imask = blue_mask > 0
    blue = np.zeros_like(img, np.uint8)
    blue[blue_imask] = img[blue_imask]

    red_imask = red_mask > 0
    red = np.zeros_like(img, np.uint8)
    red[red_imask] = img[red_imask]

    pink_imask = pink_mask > 0
    pink = np.zeros_like(img, np.uint8)
    pink[pink_imask] = img[pink_imask]

    p0 = np.array([
    [give_coord(red_imask)],
    [give_coord(green_imask)],
    [give_coord(blue_imask)],
    [give_coord(pink_imask)]
    ])

    return p0
```

3. Independent of the circles, find the optical flow vectors for each pixel in each frame with `cv.calcOpticalFlowFarneback` (`video_dense_of_part3.mp4` video is generated by this technique). Then, determine which pixels belong to the circles.

```
red_vec = []
green_vec = []
blue_vec = []
pink_vec = []

for i in range(frame_count-1):
    if i % 25 == 0:
        print(i)
    circle_coords = find_circles(vid_frame[i])
    prvs = cv.cvtColor(vid_frame[i], cv.COLOR_RGB2GRAY)
    nxt = cv.cvtColor(vid_frame[i+1], cv.COLOR_RGB2GRAY)
    flow = cv.calcOpticalFlowFarneback(prvs, nxt, None, 0.5, 3, 15, 3, 5,
1.2, 0)

    red_vec.append(flow[circle_coords[0, 0, 0], circle_coords[0, 0, 1],
:].astype(np.float32))
    green_vec.append(flow[circle_coords[1, 0, 0], circle_coords[1, 0, 1],
:].astype(np.float32))
    blue_vec.append(flow[circle_coords[2, 0, 0], circle_coords[2, 0, 1],
:].astype(np.float32))
    pink_vec.append(flow[circle_coords[3, 0, 0], circle_coords[3, 0, 1],
:].astype(np.float32))
```

4. Now, we know the optical flow of the circles in each frame. We can sum their absolute values to find the speed.

```
def calc_speed(arr):
    a = np.sum(np.array(arr)**2, axis=1)
    return np.sum(np.sqrt(a))/(frame_count-2)

print(f"Red: {calc_speed(red_vec)} pixels/frame")
print(f"Green: {calc_speed(green_vec)} pixels/frame")
print(f"Blue: {calc_speed(blue_vec)} pixels/frame")
print(f"Pink: {calc_speed(pink_vec)} pixels/frame")
```

```
Red: 3.503964013401109 pixels/frame
Green: 1.5493203549199976 pixels/frame
Blue: 3.0940148993472736 pixels/frame
Pink: 0.30066538474035354 pixels/frame
```

Failed Methods

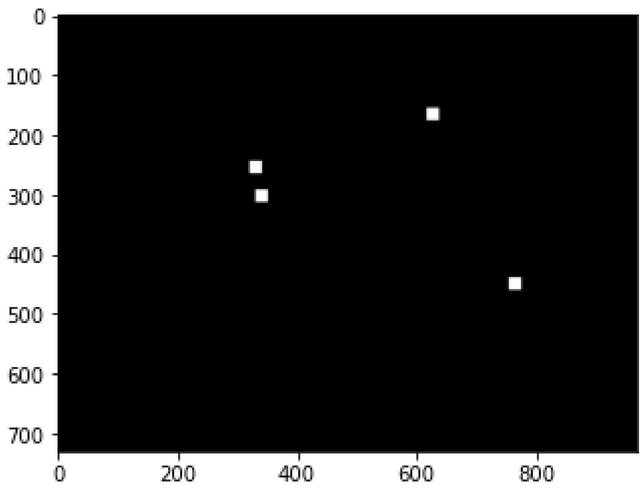
Lucas Kanade Algorithm cannot work because the movements are not small. A possible approach might be to interpolate the intermediate frames with the help of a more advanced technique, but developing and using such a technique is a challenge.

Also, since objects are round, they have no good features to track. Therefore, good features to track implementation of OpenCV does not return useful points. To overcome this issue, we have implemented the following function:

```
def good_frame(frame):
    new_frame = np.zeros((frame.shape[0] + 250, frame.shape[1] + 250))
    pts = find_circles(frame)
    for i in range(pts.shape[0]):
        new_frame[125+(pts[i, 0, 0]-10):125+(pts[i, 0, 0]+11), 125+(pts[i, 0,
1]-10):125+(pts[i, 0, 1]+11)] = 1

    return new_frame.astype(np.uint8)
```

This function creates a new, larger frame with squares instead of circles. For example:



This way, we are able to find good features to track. However, we were not able to run the optical flow algorithm in a stable way. A fun, albeit unsuccessful output of this method can be seen in the `video_square_method_part3.mp4`.