Francesco Leone

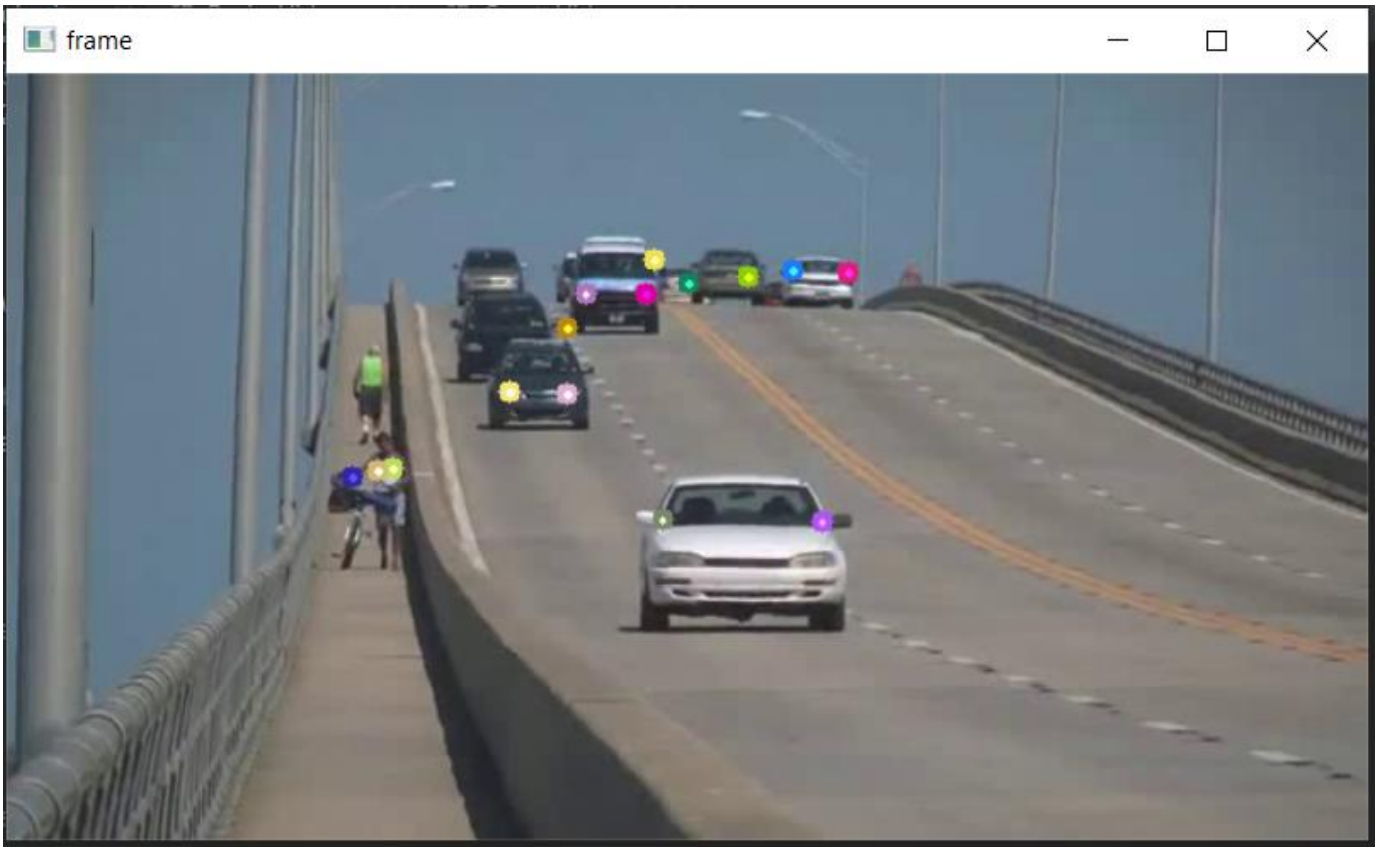
Exercise 7

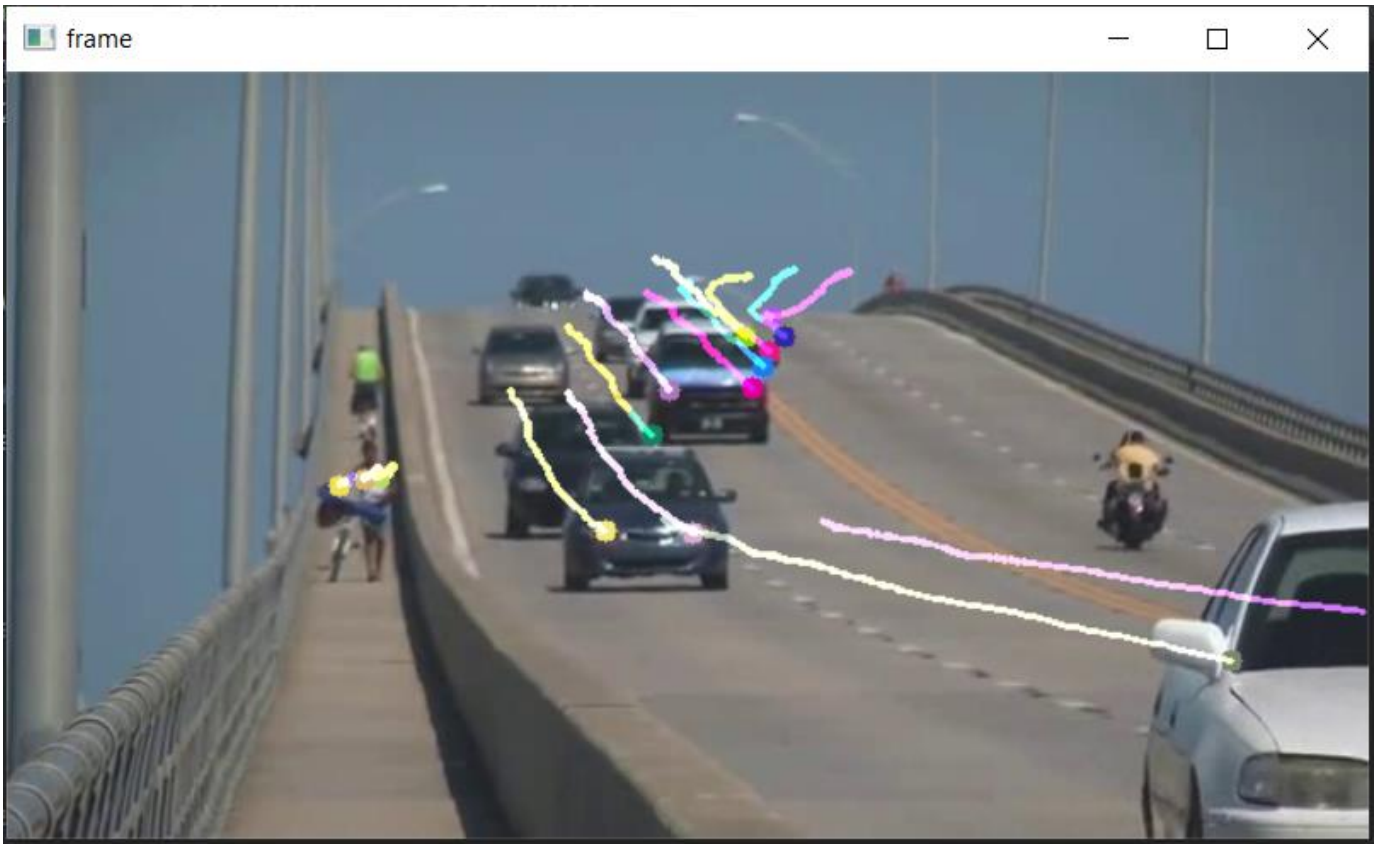
1-2) Program and results

```
import numpy as np
import cv2 as cv

videoName = input("Input video: ")
videoName = videoName + ".avi"
cap = cv.VideoCapture(videoName)

# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15,15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))
# Create some random colors
color = np.random.randint(0,255,(100,3))
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

first = True

while(1):
    try:
        ret,frame = cap.read()

        frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        # calculate optical flow
        p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)
        # Select good points
        if p1 is not None:
            good_new = p1[st==1]
            good_old = p0[st==1]
        # draw the tracks
        for i,(new,old) in enumerate(zip(good_new, good_old)):
            a,b = new.ravel()
            c,d = old.ravel()
            mask = cv.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
            frame = cv.circle(frame,(int(a),int(b)),5,color[i].tolist(),-1)
            img = cv.add(frame, mask)
            cv.imshow('frame', img)

        #show the first frame for a longer period to allow screenshots
        if(first):
            cv.waitKey(0)
            first = False
```

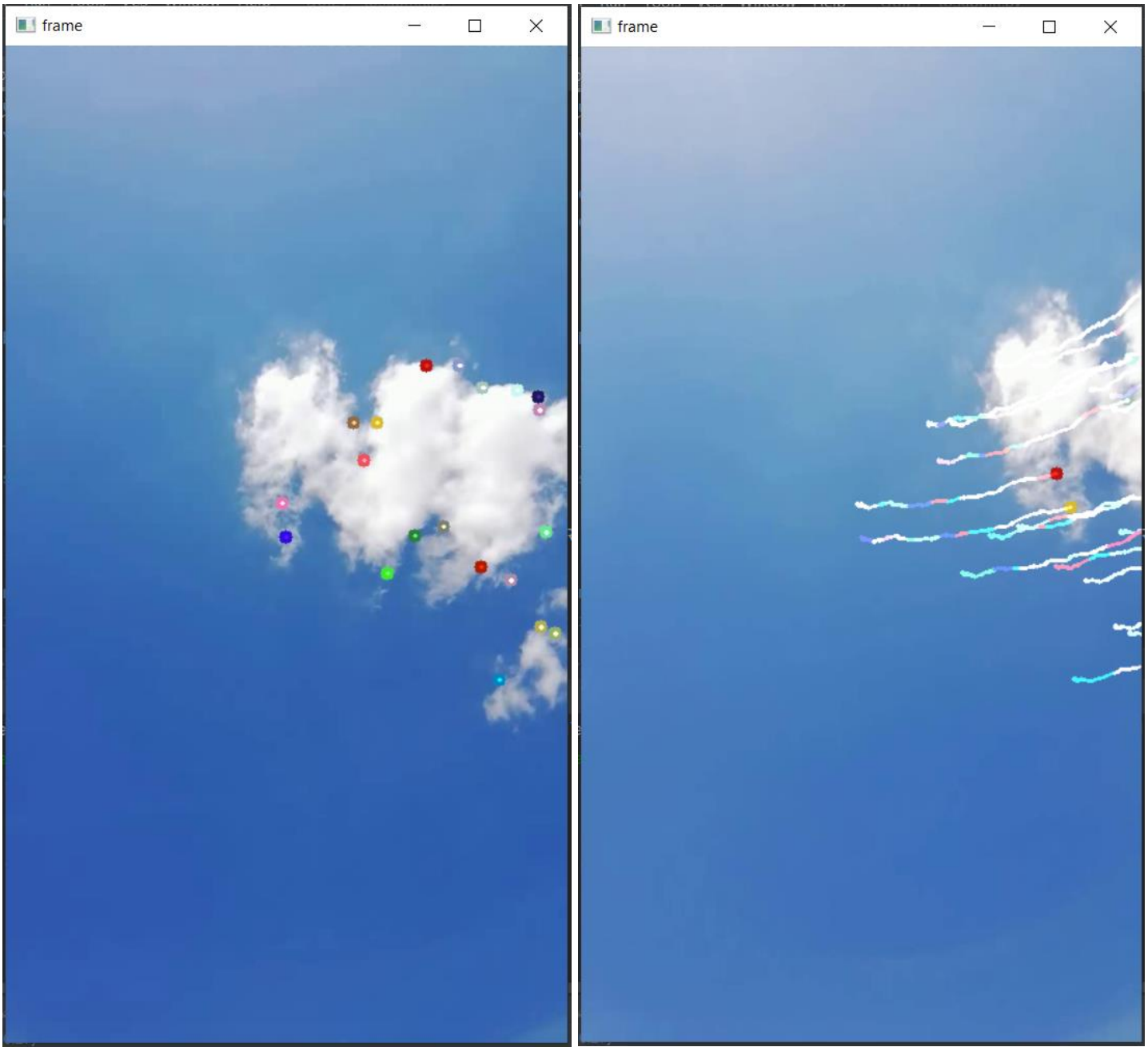
```
        else:
            k = cv.waitKey(30) & 0xff
            if k == 27:
                break
        # Now update the previous frame and previous points
        old_gray = frame_gray.copy()
        p0 = good_new.reshape(-1,1,2)

    #this exception is executed after the last frame
    except:
        cap.release()
        # show the last frame for a longer period to allow screenshots
        cv.imshow('frame', img)
        k = cv.waitKey(0)
        break
cv.destroyAllWindows()
```

Video 1: normal objects’ movement

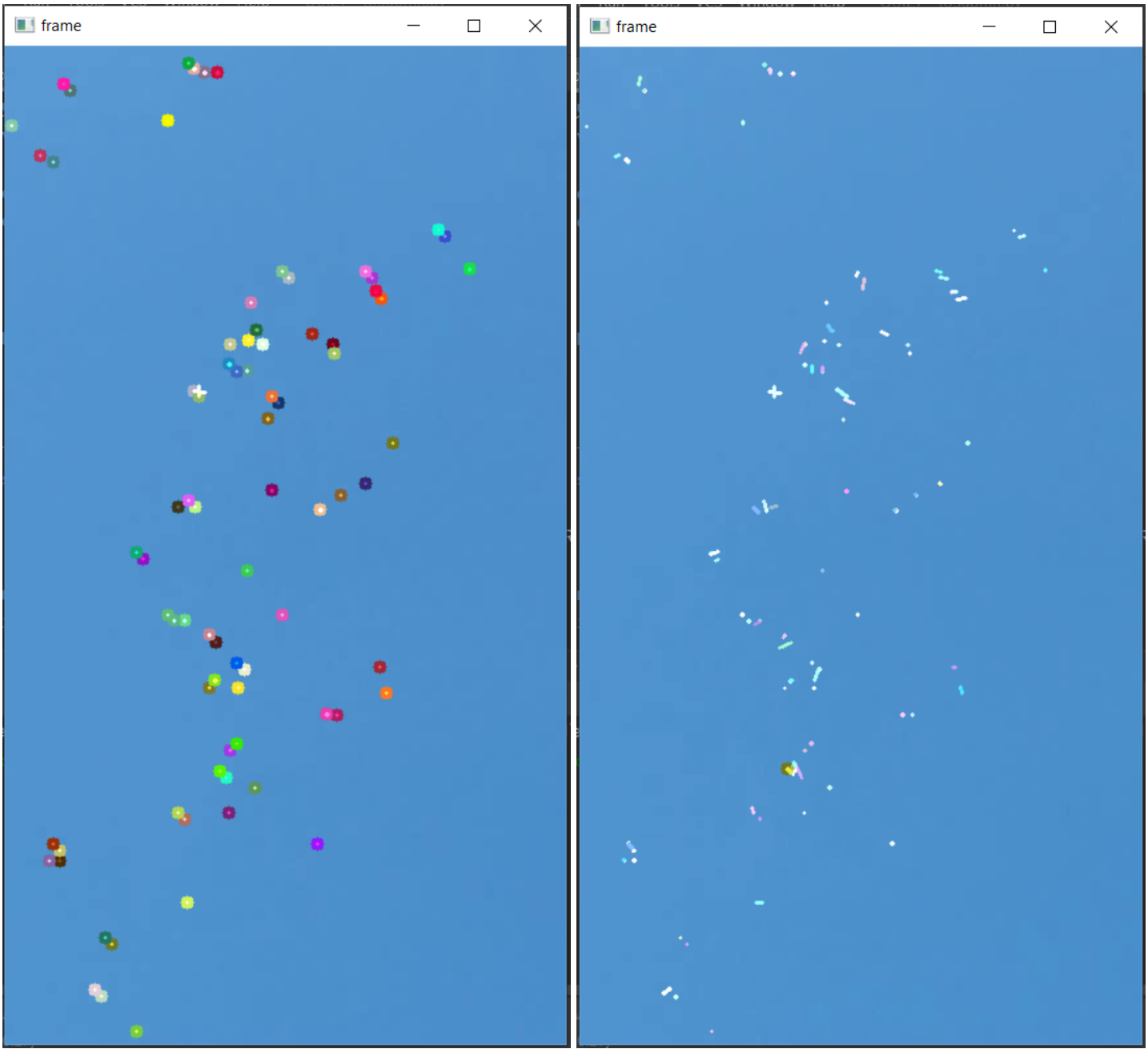




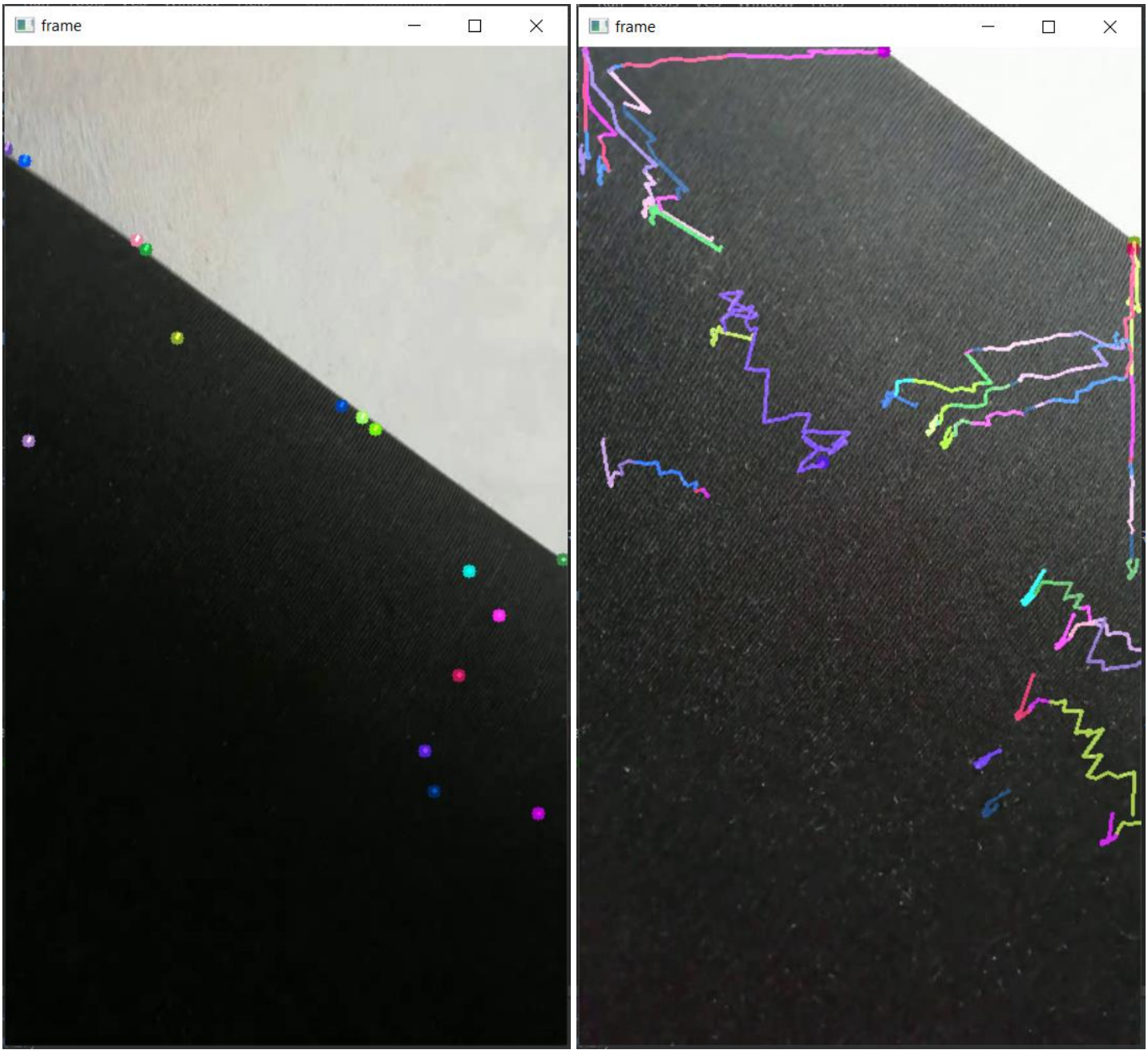
Video 2: poor textures



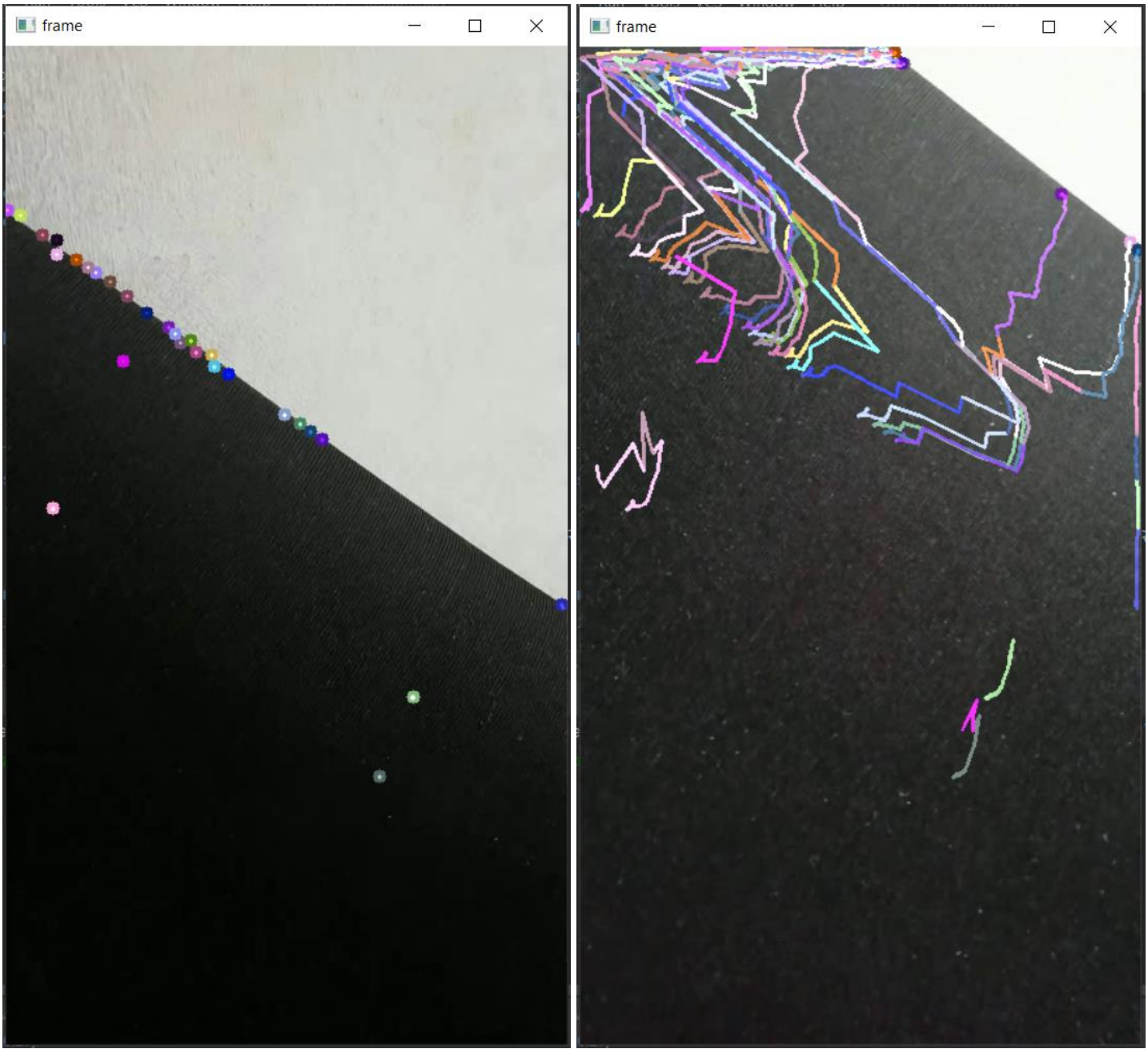
Video 3: almost absent textures



Video 4: edge’s horizontal movement



Video 5: edge’s vertical movement



3-4) Comments and discussion

In order to analyze the behavior of the Lukas-Kanade Optical Flow algorithm, I have used five different videos. The first video can be considered as the normal use case of the algorithm, since the objects, their positions and their movements are clear. The other four videos can be considered as corner cases, since the objects cannot be distinguished or even identified (as in the second and third video), or the movements are not clear (as in the fourth and fifth videos).

In the Video 1 the Lukas-Kanade Optical Flow does a great job, as expected: the movements of the feature points are very clear and smooth, and the lines that draw the movements do not have sudden changes in the direction that appear when the algorithm cannot find the path that the pixels are following.

The Video 2 is characterized by poor textures. In fact, it is possible to identify only an object (a cloud), which cover only a small part of each frame. In this case, the Lukas-Kanade Optical Flow manages only in keeping track of the movements of the cloud, but there is no information at all for what regards the remaining part of the scene.

The Video 3 shows that the Lukas-Kanade Optical Flow does not work when the scene is characterized by particularly poor (or quite inexistent) textures. In fact, even if some feature points can be found in the first frame, after a small number of frames the algorithm completely loses track of the movements. Also, even the movements detected in the first frames are incorrect since it looks like each pixel follows a different direction.

The Video 4 and the Video 5 have been used to test how the Lukas-Kanade Optical Flow works when the features points can be found only on an edge. In the Video 4 the edge moves horizontally, whereas in the Video 5 it moves vertically. The results show that in these cases the algorithm is inaccurate for most frames: at the end, it looks like the edge moves diagonally in both the videos.

To sum up, the Lukas-Kanade Optical Flow algorithm seems to work well only when the textures of the scene are not exaggeratedly poor, and the objects and their movements are enough clear. It must be said that it is easier to match these minimal requirements than not. The corner cases showed before can be easily a part of a bigger scene, but I think it would be uncommon that they represent the whole scene.

References

Source code:

https://docs.opencv.org/master/d4/dee/tutorial_optical_flow.html

Video1 (provided by the author of the source code):

https://www.bogotobogo.com/python/OpenCV_Python/images/mean_shift_tracking/slow_traffic_small.mp4