Optical Flow - Lab 7.5

Recap

This is the Lab on using an Optical Flow based algorithm to track a person as CE6003's Object Tracking section. You should complete the tasks in this lab as part of the Optical Flow section of the lesson.

Please remember this lab must be completed before taking the quiz at the end of this lesson.

First, if we haven't already done so, we need to clone the various images and resources needed to run these labs into our workspace.

In [1]:

```
#!git clone https://github.com/EmdaloTechnologies/CE6003.git
#!git clone https://github.com/mcnamarad1971/CE6003.git
```

Program Description

This program demonstrates a very simple 'tracking' mechanism - derived from a Optical Flow algorithm. We're going to use OpenCV's Lucas Kanade algorithm to track a single object, namely a person.

Imports

Standard imports

In [3]:

```
import os
import re
import io
import cv2
import time
import numpy as np
import base64
from IPython.display import clear_output, Image, display
#from google.colab.patches import cv2_imshow
```

The Story So Far

To illustrate how to track something in a video stream, we have done a little work offline. We have located a bounding box enclosing the object of interest in the first frame of the video.

What Happens Now

For the first frame in the video, we'll use our pre-rolled points to define a region of interest. We know - a priori - where the object we want to track is in the first frame.

We'll find some good features to track in that image (corners mainly) and then pass that set of corners to Lucas-Kanade algorithm along with the old frame and the new frame. It will return a set of corners in the new frame that is its best estimate of where the ROI has moved to.

We"ll find the centre of those points and use that as a tracker.

Key Parameters

We have two key parameters to shape our optical flow tracker. We have a set of parameters for the Lucas-Kanade optical flow estimation and we have a rejection radius. Any points outside the radius are disregarded for tracking purposes.

In [7]:

```
# Parameters for Lucas Kanade optical flow
lkParams = dict( winSize = (15,15),
                 maxLevel = 2,
                 criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03
))

# Rejection radius - points outside a circle with this radius are deemed to be not part
of the object we're tracking
rejRadius = 200

# The bounding box we are interested in in the first frame
defX1 = 500
defX2 = 700
defY1 = 1000
defY2 = 1800

# A video writer
writer = None

# A frame grabber
#fg = cv2.VideoCapture("/content/CE6003/images/lab7/vids/daire.mp4")
fg = cv2.VideoCapture("./images/lab7/vids/daire.mp4")
```

We need a helper function to find Euclidean distance. We use this to locate points inside and outside the radius of interest.

In [8]:

```
# Euclidean Distance term
def findDistance(x1, y1, x2, y2):
        d = (x1-x2)**2 + (y1-y2)**2
        d = np.sqrt(d)
        return d
```

Optical Flow

Look back over Optical Flow Concept video for an insight into how it is operating.

The concept is:

- For a low computational cost
- Generate a set of features - usually corners.
- Give those corners to the Lucas-Kanade algorithm. It will find the region in the next frame that best matches those corners and supplies that region
- Calculate the difference between the predicted measurement of the selected particles and the actual measurement
- Adjust the state update/prediction particles (through a resampling stage to prevent degeneration of filter), and repeat....

Done on Monte Carlo sampled particles.

One key term to watch is how many particles to use for your specific tracking application.

Demo

Program Execution

For each frame:

- get centre of detection (if any) and confidence from Yolo
- feed Particle Filter with these values
- Print internal Particles used by Particle Filter

In [9]:

```
def demo_of():
    global writer
    global fg
    global lkParams
    global rejRadius
    global defX1
    global defX2
    global defY1
    global defY2

    vidScale = 4

    # Read one frame initially
    ret, frame = fg.read()

    # Convert it to grayScale
    newFrameGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # calculate a region of interest - roi
    x1 = defX1
    x2 = defX2
    y1 = defY1
    y2 = defY2
    roi = newFrameGray[y1:y2, x1:x2]

    # get some corners
    newCorners = cv2.goodFeaturesToTrack(roi, 50, 0.01, 10)

    # translate corners from roi frame to image frame
    newCorners[:,0,0] = newCorners[:,0,0] + x1
    newCorners[:,0,1] = newCorners[:,0,1] + y1

    # Draw the corners we're tracking in the original image
    for corner in newCorners:
        cv2.circle(frame, (int(corner[0][0]), int(corner[0][1])), 5, (0,255,0))

    vidout = cv2.resize(frame, (int(frame.shape[1]/vidScale), int(frame.shape[0]/vidSca
le)))
    #cv2_imshow(vidout)

    if writer is None:
        # Initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*'VP80')
        writer = cv2.VideoWriter('video.webm', fourcc, 30,
                                 (vidout.shape[1], vidout.shape[0]), True)

    # Write the output frame to disk
    writer.write(vidout)

    # set up old corners and old frame
    oldFrameGray = newFrameGray.copy()
    oldCorners = newCorners.copy()

    # start tracking
    while True:
        # New we have an old frame, we can get
        # a new_frame, we have old_corners and
        # we can get new_corners, and update accordingly
```

```
        # read new frame and convert to gray
        ret, frame = fg.read()
        if ret == False:
            break

        # Grayscale
        newFrameGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # find the new tracked points
        newCorners, st, err = cv2.calcOpticalFlowPyrLK(oldFrameGray,
                                                       newFrameGray,
                                                       oldCorners,
                                                       None, **lkParams)

        # Prune far away points
        mean = newCorners.mean(axis=0)
        mean = mean.astype(int)

        # draw the centroid
        cv2.circle(frame, (mean[0][0],mean[0][1]),int(rejRadius),(255,0,0), 20)

        # keep only those corners
        # which are at a distance of rejRadius or less
        # use a list expression
        newCorners = np.array([i for i in newCorners if
                               findDistance(i[0][0], i[0][1], mean[0][0],
                                            mean[0][1]) <= rejRadius])

        # draw the new points
        for corner in newCorners:
            cv2.circle(frame, (int(corner[0][0]),int(corner[0][1])),5,(0,255,0))

        if len(newCorners) < 10:
            print ('Lost Object', len(newCorners))
            break

        # Find minimum enclosing circle
        ctr, rad = cv2.minEnclosingCircle(newCorners)

        # Draw this circle
        cv2.circle(frame, (int(ctr[0]),int(ctr[1])),int(rad),(0,0,255),thickness=5)

        # Update old_corners and oldFrameGray
        oldFrameGray = newFrameGray.copy()
        oldCorners = newCorners.copy()

        vidout = cv2.resize(frame, (int(frame.shape[1]/vidScale), int(frame.shape[0]/vi
dScale)))

        # Write the output frame to disk
        writer.write(vidout)

    # Release the file pointers
    writer.release()

demo_of()

!ls video.webm
```

```
'ls' is not recognized as an internal or external command,
operable program or batch file.
```

Video

Thia code plays the video we just made.

The individual corners identified by the Shi-Tomasi / Lucas-Kanade are in green.

The 'object' is in red and the rejection limit is in blue.

As you can see, Optical Flow has a role to play in object tracking but it can struggle with occlusions.

In [10]:

```
# Set this to 1 if video display
# is not working - works with chrome and firefox, not with safari
videoBodge = 0

def arrayShow (imageArray):
    ret, png = cv2.imencode('.png', imageArray)
    encoded = base64.b64encode(png)
    return Image(data=encoded.decode('ascii'))

if(videoBodge == 0):
    from IPython.display import HTML
    from base64 import b64encode
    webm = open('video.webm','rb').read()
    data_url = "data:video/webm;base64," + b64encode(webm).decode()
else:
    video = cv2.VideoCapture("video.webm")
    while(video.isOpened()):
        clear_output(wait=True)
        ret, frame = video.read()
        if(ret == False):
          break
        lines, columns, _ =  frame.shape
        img = arrayShow(frame)
        display(img)
        time.sleep(1)
```

In [11]:

```
# Display Video
HTML("""
<video width=200 controls>
      <source src="%s" type="video/webm">
</video>
""" % data_url)
```

Out[11]:

Conclusion

Exercises

Exercise 1 Answer the question - would taking a new set corners on each iteration improve the algorithm's performance?

Stretch 1 Convert the program to track using a dense optical flow algorithm.

Takeaways

1. You've seen a Sparse Optical Flow used for single object tracking
2. You've seen that it did't deal terribly well with occlusions - i.e. in this example the object being tracked disappeared for a few frames and the Optical Flow based tracker lost it at that point.
3. You've seen that you probably need to tune the Filter to get it working for a particular application.

Next Steps

1. We'll look at one last tracking approach based on a CNN Feature based tracker.