ACTIVITY 11

BASIC VIDEO PROCESSING

Sheenah A. Ancheta

Objective

To get the acceleration due to gravity g of a free-falling tennis ball with no external force aside from gravity.

Figure 1. Snapshot from the video of a free-falling tennis ball

Camera used:

iPhone 7 Plus

Camera/Video settings:

240 frames per second (*fps*) at 720p resolution

A slow-motion video of a free-falling tennis ball was captured. A snapshot of the video is shown in Figure 1. The video was then converted into image sequences using the *scene video filter* in VLC media player.

Process in the video processing

- Set-up the blob detector
- Determine the scale (*pixel* to *cm*)
- Track the centroid of the ball in terms of x and y position
- Calculate the acceleration due to gravity g

Setting-up the Blob Detector

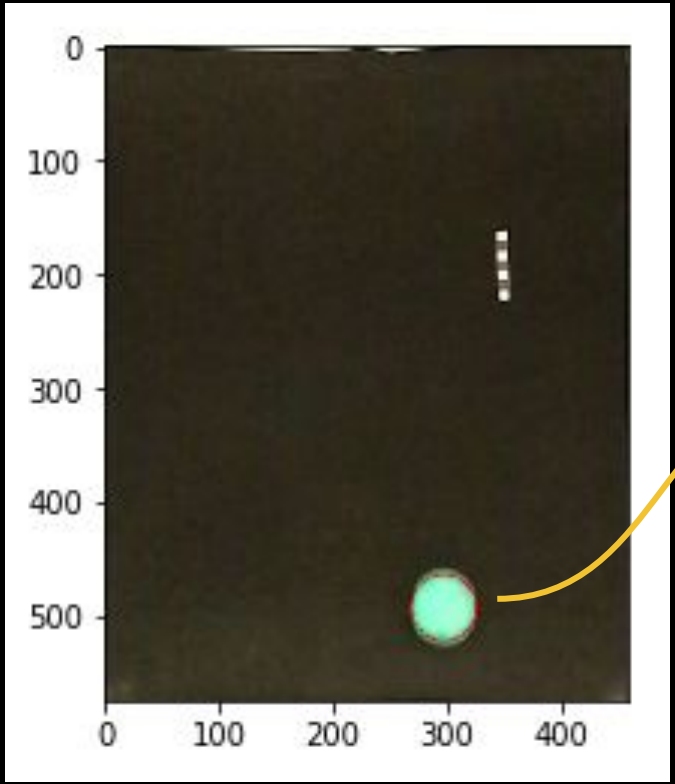
A blob detector was set-up in a way shown in the code snippet in Figure 2. The detector was controlled by parameters, area and circularity. The minimum area was set to 200 to ensure that only the ball gets detected later on. The circularity was also considered. The function *drawKeypoints( )* was used to identify where the features of the blob has been detected.

```
params = cv2.SimpleBlobDetector_Params()

#Filter by Area.
params.filterByArea = True
params.minArea = 200

#Filter by Circularity
params.filterByCircularity = True
params.minCircularity = 0.5

#Set up the detector with default parameters.
detector = cv2.SimpleBlobDetector_create(params)
```

Figure 2. Code snippet on the setting-up of the blob detector

```
image = cv2.imread('pics/frame213.jpg')
image = image[385:960,120:580] #crop
keypoints = detector.detect(image)
im_with_keypoints = cv2.drawKeypoints(image,keypoints,np.array([]),(255,0,0),cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

Figure 3. Code snippet on drawing the keypoints of the detected blob in an image



Keypoints, right exact on the tennis ball

Figure 4. Result after blob detection and drawing the keypoints

Determining the scale

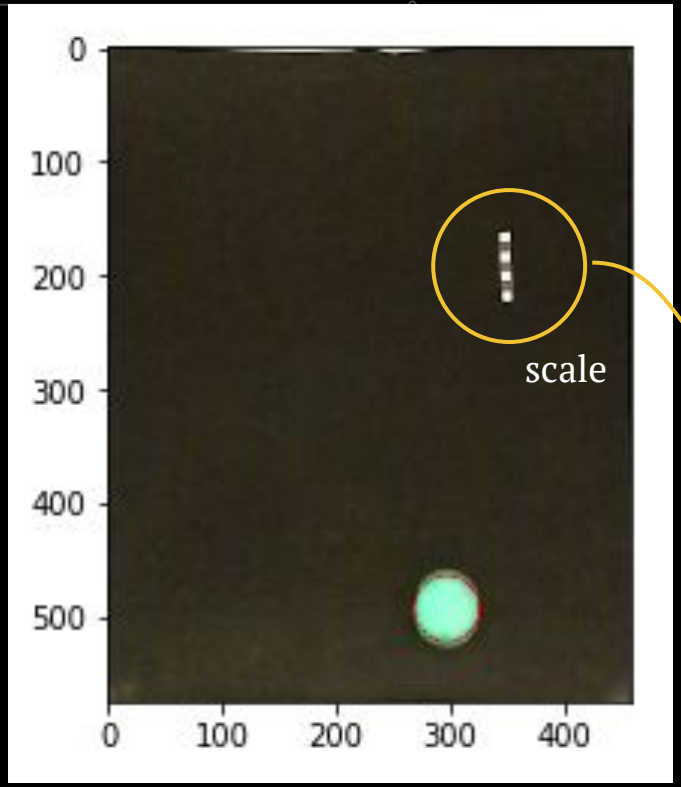


Figure 5. Scale used in the video

Each box on the scale is equivalent to 1 cm. The scale shown here has a total length of 7 cm. In terms of pixel distance, it is 60 px in total.

Therefore, the final scale is:

1 px : 0.117 cm

Tracking the centroid

We keep track of the centroids of the ball at each frame to help us determine the position of the ball

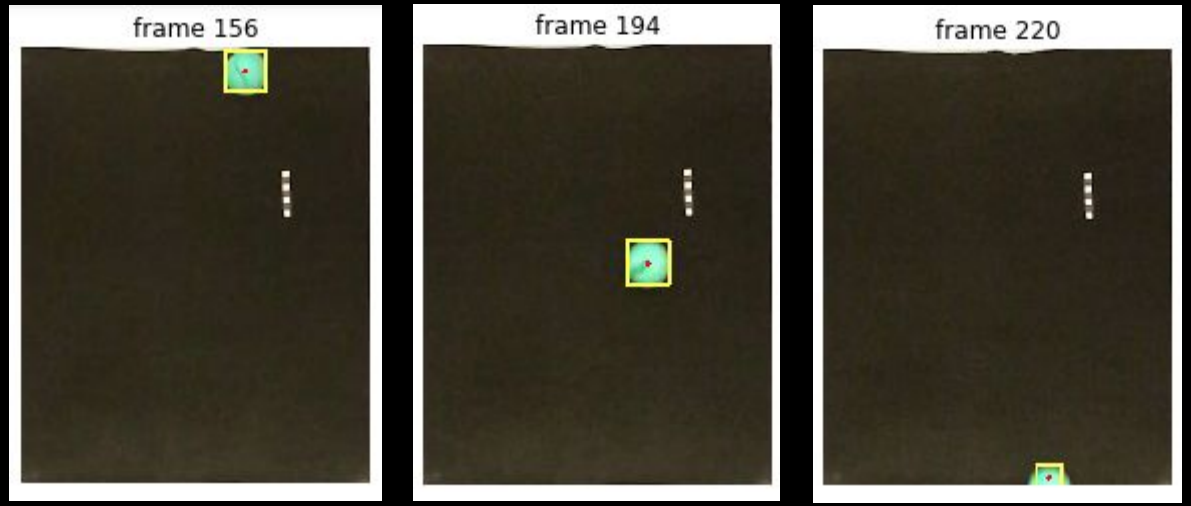


Figure 6. Sample frames from the image sequences obtained from the free-falling video of a tennis ball

Shown in the images are the first frame (frame 156), last frame (frame 220), and a sample frame from when the ball was in the middle of the path (frame 194). A square and centroid dot was placed to keep track of the free-falling tennis ball.

```
first_frame = 156
last_frame = 220
frames = range(first_frame,last_frame+1)

centroids = []

for i in frames:

    #Read image
    image = cv2.imread('pics/frame'+str(i)+'.jpg')

    #Crop image
    image = image[385:960,120:580]

    #Detect ball
    keypoints = detector.detect(image)

    #Draw center and square
    pt = keypoints[0].pt
    d = keypoints[0].size
    im_with_sq = cv2.rectangle(image,(int(pt[0]-d/2),int(pt[1]-d/2)),(int(pt[0]+d/2),int(pt[1]+d/2)),(255,255,0),4)
    im_with_sq = cv2.rectangle(image,(int(pt[0]),int(pt[1])),(int(pt[0]+1),int(pt[1]+1)),(255,0,0), 5)

    #Record position
    centroids.append(keypoints[0])
```

Figure 7. Code snippet on tracking the centroid of the ball in each frame of the video clip

Calculating *g*

Acceleration due to gravity

Now that we already kept track of the centroids of the ball, it is time for us to take the centroids in terms of x and y position for the ball at each frame. We do this to give us an approximation on the average x and y positions of the ball along its trajectory.

Shown in the code snippet is how we obtained the trajectory of the tennis ball.

```
traj_x = []
traj_y = []

#Getting x and y values
for keypoint in centroids:
    x,y = keypoint.pt
    traj_x.append(x)
    traj_y.append(y)
```

Figure 8. Code snippet on getting the trajectory of the tennis ball

The next step was for us to convert the frames to time. We do this by taking note of the camera setting when the video was taken, which I previously mentioned was 240 *fps*. Using this *frame per second* value, we could get the time in *seconds* by implementing the code shown in the snippet in Figure 6.

```
frames = range(first_frame,last_frame+1)

N = len(centroids)
fps = 240
time = (1/240)*np.array(frames)
```

Figure 9. Code snippet on converting frames to time in seconds

```
#Define function
def y(t,g,v0,y0):
    return y0 + v0*t + (1/2)*g*(t**2)

from scipy.optimize import curve_fit

popt,pcov = curve_fit(y,time,traj_y)
g_fit, v0_fit, y0_fit = popt

#value of g in px/s^2
print(str(g_fit)+' px/s^2')
```

```
8118.752006789035 px/s^2
```

```
print("g = " + str(g_fit*((7/60)*0.01))) #converting to m/s^2
```

```
g = 9.471877341253874
```

Figure 10. Code snippet on converting frames to time in seconds

Finally, we know that the ball is moving along the *y*-axis. I assumed that the value along *x* is constant and therefore, I only considered the vertical component values--position and the trajectory of the ball along *y*. The position of the ball along *y* is obtained by the formula:

$$y = y_o + V_{o,y}t - 1/2gt^2$$

```
#Define function
def y(t,g,v0,y0):
    return y0 + v0*t + (1/2)*g*(t**2)

from scipy.optimize import curve_fit

popt,pcov = curve_fit(y,time,traj_y)
g_fit, v0_fit, y0_fit = popt

#value of g in px/s^2
print(str(g_fit)+' px/s^2')
```

```
8118.752006789035 px/s^2
```

```
print("g = " + str(g_fit*((7/60)*0.01))) #converting to m/s^2
```

```
g = 9.471877341253874
```

Figure 11. Code snippet on converting frames to time in seconds

From the code snippet shown, the value of the acceleration due to gravity g was found to be $\mathbf{g = 8118.75\ px/s^2}$. Converting this to m/s^2 using the pixel-to-cm scale we got earlier, the final acceleration due to gravity g is

$$\mathbf{g = 9.47\ m/s^2}.$$

This final value of g deviates by around 3.37% from the theoretical value of 9.8 m/s^2.

Self-evaluation

Technical correctness	5
Quality of presentation	5
Initiative	2
Total	**12**