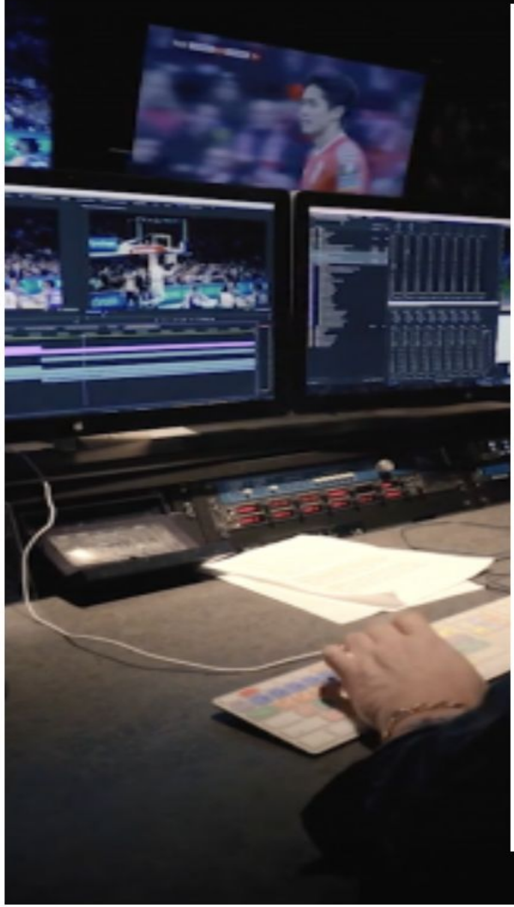
# ACTIVITY 11
# Basic Video Processing

Marc Jerrone R. Castro
2015-07420

## Video Processing

*A short background on the concept of video processing*



### 1 What is Video Processing?

*The core of videos and video processing*

Videos are strings of images which are shown in rapid succession as to simulate an impression of movement. We may simply look at videos as three dimensional images where length, width, and an axis with respect to time is considered.

As such video processing is simply extracting, enhancing, and analyzing these rapid succession of images by decomposing it to a set of images as a function of the frame rate of frames per second of the recording device.

**Note:** Some applications of video processing is for analyzing clips of a kinematic or dynamic system such that one is capable of extracting kinematic variables such as gravity using image processing techniques we have learned so far ona series of images that have been extracted from a video file.

**Video Length**

6 seconds

30 FPS

**Frame Rate**

~25 frames per second

**Video Dimension**

1280 x 720

## RELEVANT INFORMATION

The clip used in this experiment was taken using a Samsung A7 2018 camera, an orange pingpong ball and a experimental setup containing a backdrop of known length.

**Total Images Produced**

152

**Time Between Frames**

1/25 seconds per frame

**Image Dimensions**

1280 x720

## VIDEO PRE-PROCESSING STEP TWO

Using the trimmed video, I loaded it on FreeVideoToJPGConverter where a scene filter was set to convert per video frame into individual images. The resulting images were then loaded into a Python notebook for video processing

# 03. Parametric Segmentation

## INPUT

## CODE

```python
path = "/content/drive/My Drive/186/Activity 11/RAW/*.jpg"
savepath = "/content/drive/My Drive/186/Activity 11/PROCESSED/"

#INITIAL IMAGE TO SET ROI SHADE AND COLOR"
#img = io.imread('00016.jpg')
img = io.imread('00014.jpg')
image = sm.img_as_float(img)
image = np.double(image)
image = image[45:1050,790:1125]
#cropped = image[27:60,130:165]
cropped = image[193:220,188:215]
cropped_imgR = cropped[:,:,0]
cropped_imgG = cropped[:,:,1]
cropped_imgB = cropped[:,:,2]


#Loop for all images"
for file in glob.glob(path):
  #print(file)
  filename = file[-9:]
  #print(filename)
  img = io.imread(file)
  image = sm.img_as_float(img)
  image = np.double(image)
  image = image[45:1050,790:1125]

  image_R = image[:,:,0]
  image_G = image[:,:,1]
  image_B = image[:,:,2]

  I_patch = cropped_imgR + cropped_imgG + cropped_imgB
  I_obj = image_R + image_G + image_B

  I_patch[I_patch==0] = 100000
  I_obj[I_obj==0] = 100000

  r_patch = cropped_imgR/I_patch
  g_patch = cropped_imgG/I_patch

  r_obj = image_R/I_obj
  g_obj = image_G/I_obj

  mean_r = np.mean(r_patch)
  mean_g = np.mean(g_patch)
  stdev_r = np.std(r_patch)
  stdev_g = np.std(g_patch)

  joint_prob = probability(r_obj,r_patch)*probability(g_obj,g_patch)
  cv2.imwrite(savepath+str(filename),joint_prob)
```

## REASONING

I used parametric segmentation for this step to extract only the pingpong ball in motion. I set the ROI for the entire loop using the color and intensity profile of the ball from one of the images.

Aftewards, I used the joint probability equation we learned from Activity 3 to extract sections where the ball is likely to be located for each image.

# 03.

## Parametric Segmentation

### Sample Results

**Figure 1:** Results of the parametric segmentation for frames 2 to 11. The resulting images depicts the probable locations of the ball using parametric segmentation. As observed, some frames contained non-circular images and there were stray pixel included too. By using morphological operations and thresholding, we clean this image even further by removing stray pixels. In addition, one may observe that some segmentations had holes in it which suggests that we fill the holes using cv2.floodFill.

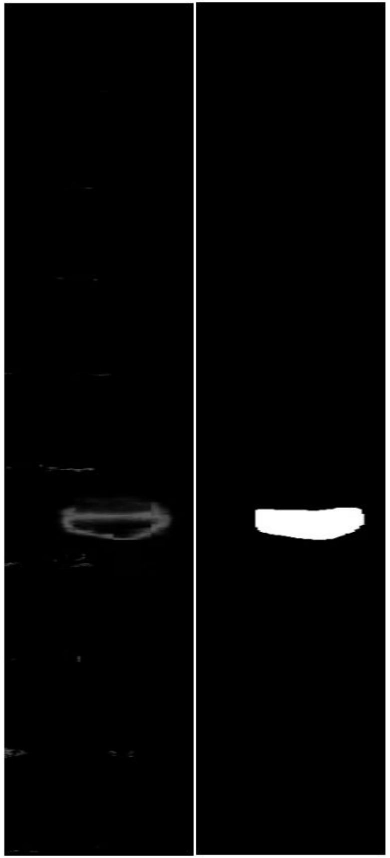


**Figure 2:** Side-by-side comparison of pre-morphed (left) and post-morphed (right) images of Frame 9 in our video.

## THE PROBLEM

If you have noticed,in addition to stray pixels, there were some segmentations that had holes in them. In order to fill these holes and remove stray pixels, I opted to use morphological operations (specifically cv2.MORPH_OPEN) to remove the stray pixels. As for filling the holes, I used method that involves cv2.floodFill which I found from [1].

## CODE SOLUTION

```
newpath = "/content/drive/My Drive/186/Activity 11/PROCESSED/*.jpg"
newsavepath = "/content/drive/My Drive/186/Activity 11/MORPH/"
for newfile in glob.glob(newpath):
  A = cv2.imread(newfile)
  newfilename = newfile[-9:]
  A2 = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
  ret,thresh1 = cv2.threshold(A,20,255,cv2.THRESH_BINARY)

  # Copy the thresholded image.
  im_floodfill = thresh1.copy()

  # Mask used to flood filling.
  # Notice the size needs to be 2 pixels than the image.
  h, w = thresh1.shape[:2]
  mask = np.zeros((h+2, w+2), np.uint8)

  # Floodfill from point (0, 0)
  cv2.floodFill(im_floodfill, mask, (0,0), 255);

  # Invert floodfilled image
  im_floodfill_inv = cv2.bitwise_not(im_floodfill)

  # Combine the two images to get the foreground.
  im_out = thresh1 | im_floodfill_inv

  im_out = cv2.cvtColor(im_out, cv2.COLOR_BGR2GRAY)
  ret2,thresh2 = cv2.threshold(im_out,254,255,cv2.THRESH_BINARY)

  kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10,10))
  opening = cv2.morphologyEx(thresh2, cv2.MORPH_OPEN, kernel)
  cv2.imwrite(newsavepath+str(newfilename),opening)
```

[1] https://www.learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/

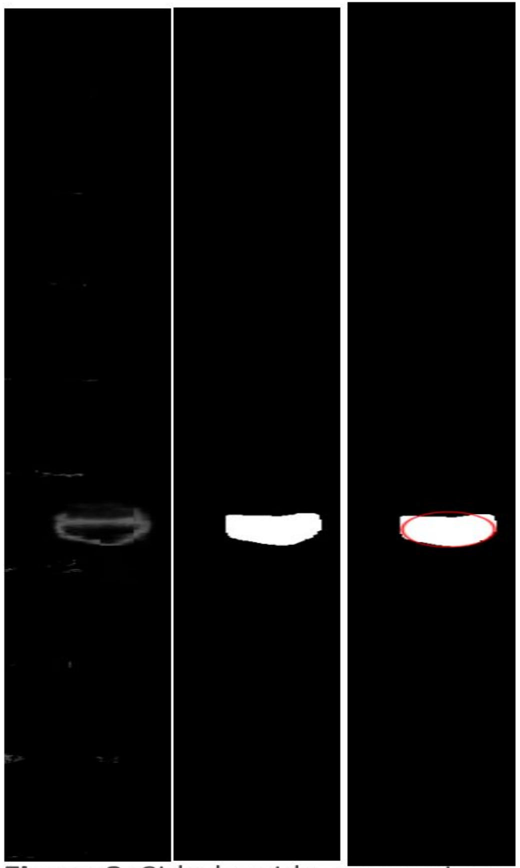


**Figure 3:** Side-by-side comparison of pre-morphed (left), post-morphed (middle), and post-morphed image with its keypoint marked (right) images of Frame 9 in our video.

## DETECT BLOBS

We then perform blob analysis to determine the locations of the centroids of the images. Since the motion is freefall, we are only interested in the y-axis coordinates of each blob centroid.We then import these coordinates to an equation to determine the velocity and acceleration of the ball.

## PROPOSED SOLUTION

```
imgpath = "/content/drive/My Drive/186/Activity 11/MORPH/*.jpg"
centroids = []
reference_file = []
ypos = []
for imgfile in glob.glob(imgpath):
  A = cv2.imread(imgfile)
  A = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
  ret,A2 = cv2.threshold(A,20,255,cv2.THRESH_BINARY)
  imgfilename = imgfile[-6:]
  # Setup SimpleBlobDetector parameters.
  params = cv2.SimpleBlobDetector_Params()

  # Filter by Circularity
  params.filterByCircularity = True
  params.minCircularity = 0

  detector = cv2.SimpleBlobDetector_create(params)
  keypoints = detector.detect(A2)
  im_with_keypoints = cv2.drawKeypoints(A2, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
  #cv2_imshow(opening)
  #cv2_imshow(im_with_keypoints)

  label, N = sm.label(A2, background=0, return_num=True)
  reg = sm.regionprops(label,A2)
  if N == 0:
    print(imgfilename, 'has no labels.')

  imgcent = []
  if N != 0:
    reference_file.append(imgfilename)
    #print(imgfilename, 'has at least',N,'labels.')
    for i in range(N):
      imgcent.append(reg[i].centroid)
    centroids.append(imgcent)
    ys = list(imgcent)[0][0]
    ypos.append(ys)


  #print(imgcent)
    print('The ball at',imgfilename,'was located at an average y-axis coordinate',round(ys))
```

## STATISTICS

Having the y-coordinates of each centroid in the image, we then determine the corresponding pixel location at real time and plotted it using the following code:

```
delta_t = 1/25
time = np.arange(0,10/25,delta_t)
pix_2_dist = .9/1085
scaled = ypos*pix_2_dist
plt.scatter(time,scaled)
plt.plot(time,scaled)
plt.xlabel("Time (s)")
plt.ylabel("Distance (m)")
```

## VELOCITY AND ACCELERATION

We then calculate for velocity and acceleration using the following codes:

```
vel = []
for i in range(len(scaled)-1):
  delta_d = scaled[i+1] - scaled[i]
  v = delta_d/delta_t
  vel.append(v)
```

```
accel = []
for i in range(len(vel)-1):
  delta_v = vel[i+1] - vel[i]
  a = delta_v/delta_t
  accel.append(a)
```

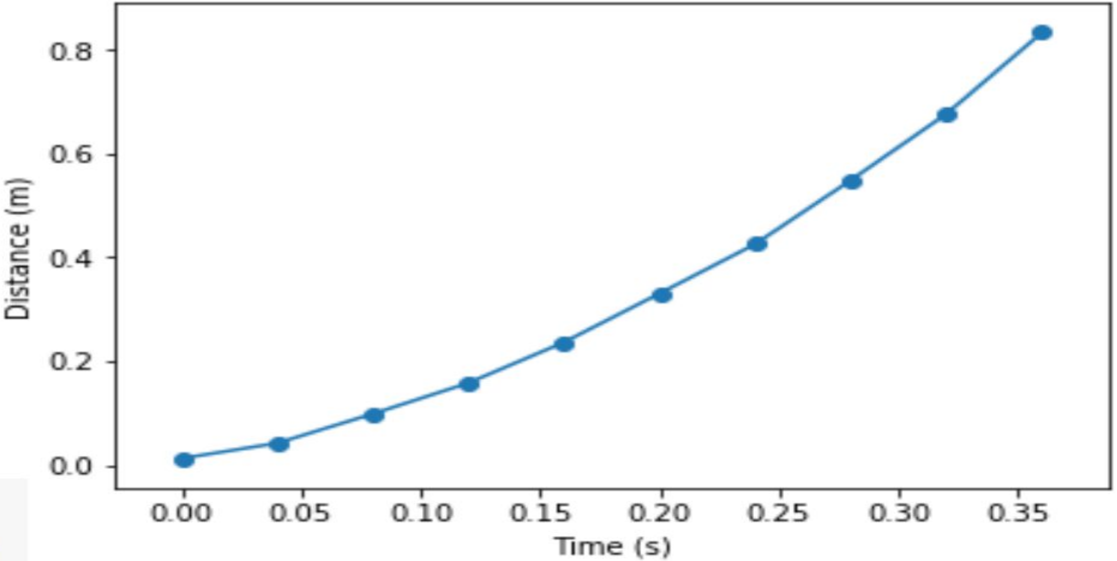



**Figure 4:** Distance vs Time plot of the center of the ball for each frame. This plot signifies very similar properties to the theoretical motion of objects in freefall.

## STATISTICS

We then see that the velocity increases through time as expected while acceleration flunctuated - maybe due to device limitations. As such we also determine the mean value of acceleration.

```
1 vel
```

```
[0.734255673775125,
 1.4104466805446012,
 1.487114052988527,
 1.9606170697245995,
 2.350169509819115,
 2.377597856232404,
 3.0772725236015708,
 3.191226603376337 7,
 3.91985631647479]
```

```
1 accel
```

```
[16.904775169236903,
 1.916684311098143 5,
 11.837575418401814,
 9.73881100236289,
 0.6857086603322182,
 17.491866684229173,
 2.848851994369172 6,
 18.215742827461312]
```

We determined that the average acceleration is:

(9.955 +/- 6.8735)m/s^2

which has an error of,

1.5%

which means that our video processing resulted to a fairly accurate estimation on acceleration due to gravity (9.8m/s^2).

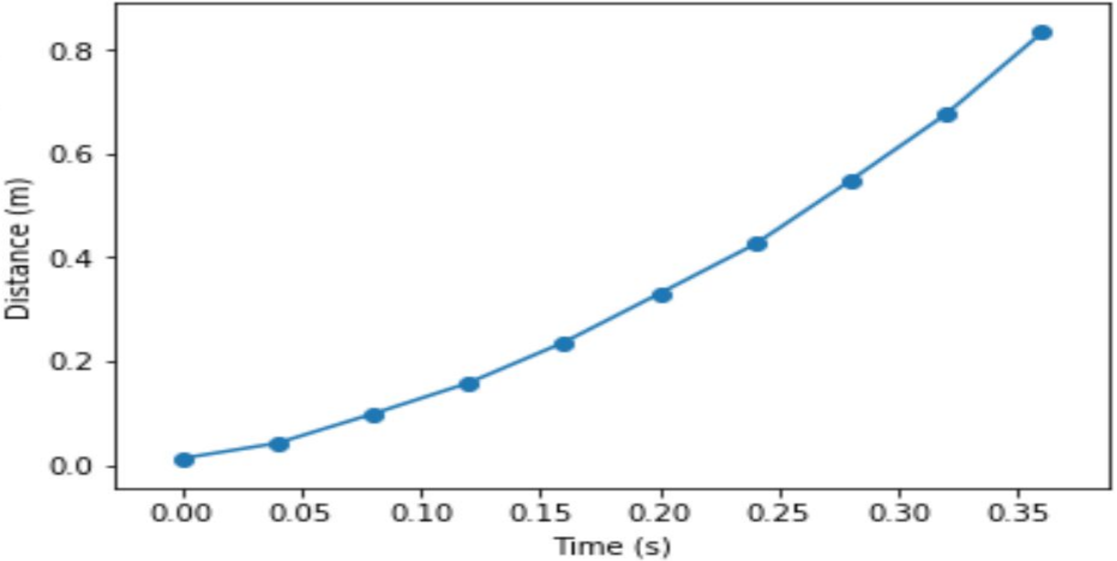



**Figure 4:** Distance vs Time plot of the center of the ball for each frame. This plot signifies very similar properties to the theoretical motion of objects in freefall.

## STATISTICS

As an additional step, I also determined the best fit line by importing the scaled velocity and the time steps into a spreadsheet. Afterwards, I determined the equation of the line for a polynomial fit.

The equation of the line suggests the equation of motion is given by,

x + vt+0.5 (a*t^2)

Therefore looking at the third term of the equation, we can assume that 4.37 = 0.5 a. Which suggests that acceration is 8.74m/s^2.

which has an error of,

10.8%

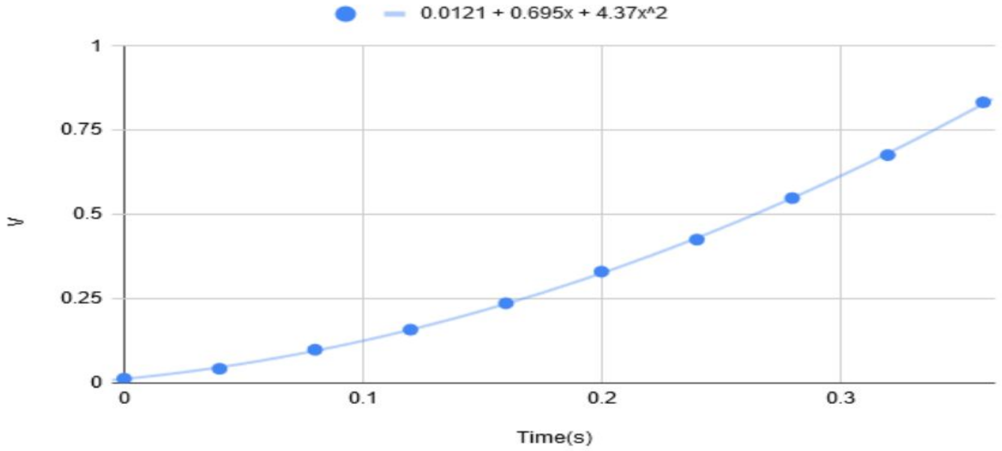



**Figure 5:** Distance vs Time plot of the center of the ball for each frame using Google Spreadsheets.

# SELF EVALUATION

**QUALITY OF PRESENTATION: 5/5**

**TECHNICAL CORRECTNESS: 5/5**

**INITIATIVE: 2/2**

FILE LINKS: INPUT VIDEO; IMAGE FILES