Howzatt! How to Build Your Own Ball Tracking System for Cricket

ADVANCED COMPUTER VISION IMAGE OBJECT DETECTION OBJECT TRACKING PYTHON SPORTS TECHNIQUE UNSTRUCTURED DATA UNSUPERVISED

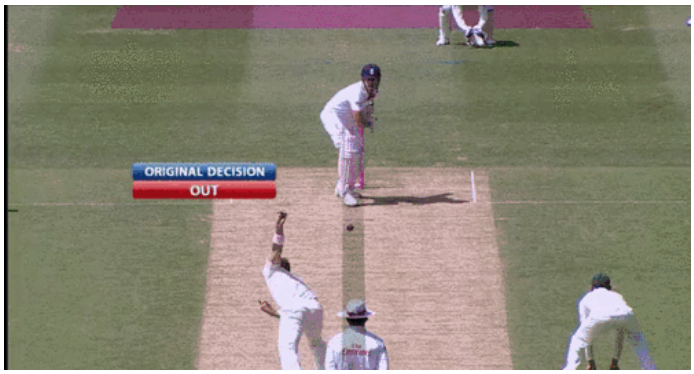
Overview

- Learn how to build your own ball tracking system for cricket using computer vision and Python
- Understand different approaches for tracking fast-moving objects in a sports video
- We will also discuss the various use cases of a ball tracking system

Introduction

The Decision Review System (DRS) is quite ubiquitous in the sport of cricket these days. Teams are starting to rely heavily on the DRS to overturn tight umpiring decisions and that, quite often, can turn the match in their favor.

This ball tracking concept, part of the DRS, is now an all too familiar sight for cricket fans:



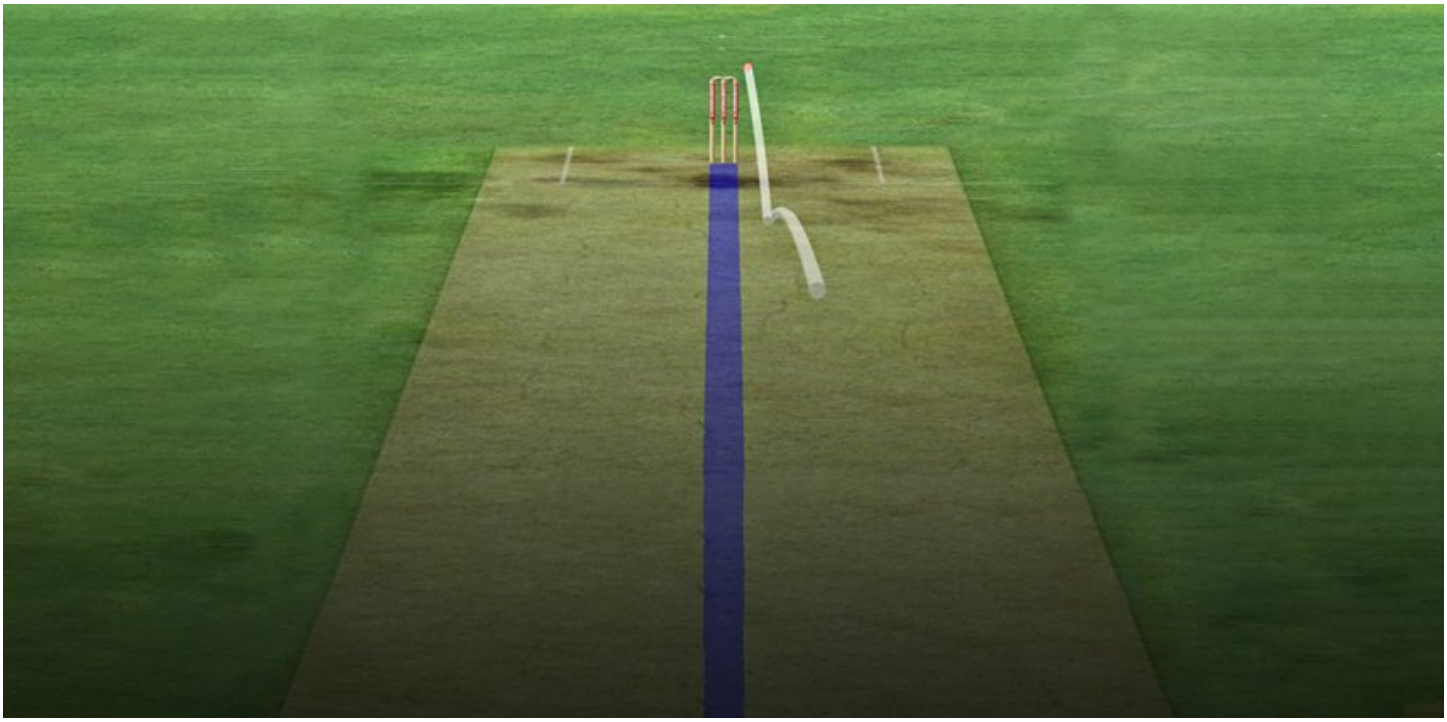
This got me thinking – could I build my own ball tracking system using my knowledge of deep learning and Python?

I'm a huge cricket fan and I'm constantly looking for different use cases where I can apply machine learning or deep learning algorithms. The idea of building a ball tracking system came to me when I was working on my previous project focused on generating insights from cricket commentary data.

Cricket teams and franchises use this idea of ball-tracking to understand the weak zones of opposition players as well. Which position is a particular batsman vulnerable in? Where does the bowler consistently pitch in the death overs?

Ball tracking systems help teams analyze and understand these questions. Here is one such example from the recent cricket match:

- New Zealand bowlers followed a strategy against Virat Kohli in the previous tour of India to New Zealand. The very first few balls faced by Virat were in his weak zone. This made him uncomfortable and he regularly lost his wicket early



In this article, we will walk through the various aspects of a ball tracking system and then build one in Python using the example of cricket. This promises to be quite a unique learning experience!

Note: If you're completely new to the world of deep learning and computer vision, I suggest checking out the below resources:

- *Fundamentals of Deep Learning*
- *Computer Vision using Deep Learning*

Table of Contents

What is a Ball Tracking System?

Let's quickly familiarize ourselves with two popular terms in Computer Vision prior to a discussion about the Ball Tracking System – Object Detection and Object Tracking.

Object Detection is one of the most fascinating concepts in computer vision. It has a far-reaching role in different domains such as defense, space, sports, and other fields. Here, I have listed a few interesting use

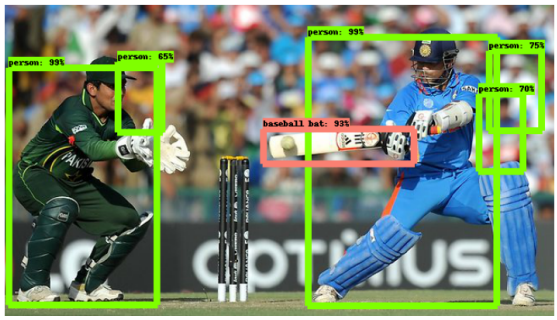
cases of Object Detection in Defense and Space:

- Automatic Target Aimer
- Training Robots in real word simulations to retrieve people in hazardous physical environments
- Detecting Space Debris

But what is object detection?

> Image Classfication + Localization = Object Detection

Object Detection is the task of identifying an object and its location in an image. Object detection is similar to an [image classification] problem but with one additional task – identifying the location of an object as well – a concept known as Localization.



As you can see here, the location of the object is represented by a rectangular box that is popularly known as a Bounding Box**.** Bounding Box represents the coordinates of the object in an image. But wait – how is Object Detection different from Object Tracking? Let's answer this question now.

Object Tracking is a special case of Object Detection. It applies to only video data. In object tracking, the object and its location are identified from every frame of a video.

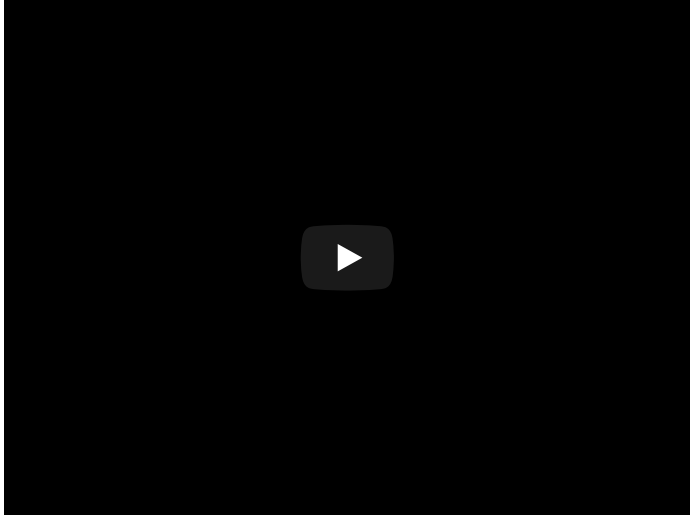
> Object Detection applied on each frame of a video turns into an Object Tracking problem.

Remember that Object Detection is for an image whereas Object Tracking is for the sequence of fast-moving frames. Both of the problems involve the same task but the terms are interchangeably used depending upon the type of data that you're working with.

So how does this apply to ball-tracking?

Ball Tracking System is one of the most interesting use cases of Object Detection & Tracking in Sports. A Ball Tracking System is used to find the trajectory of the ball in a sports video. Hawk-eye is the most advanced ball tracking system used in different sports like cricket, tennis, and football to identify the trajectory of the ball from high-performance cameras.

We can develop a similar system using the concepts of computer vision by identifying the ball and its location from every frame of a video. Here is a demo of what we will be building in this article:



Awesome, right?

Use Cases of Ball Tracking System in Sports

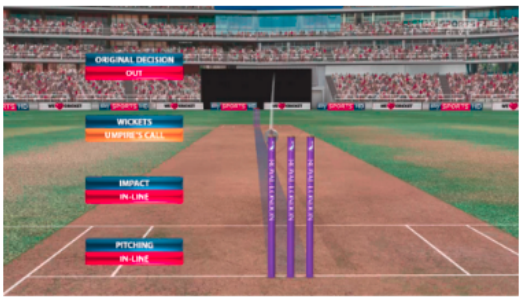
The Ball Tracking System, as I'm sure you've gathered by now, is a powerful concept that transcends industries. In this section, I will showcase a few popular use cases of ball-tracking in sports.

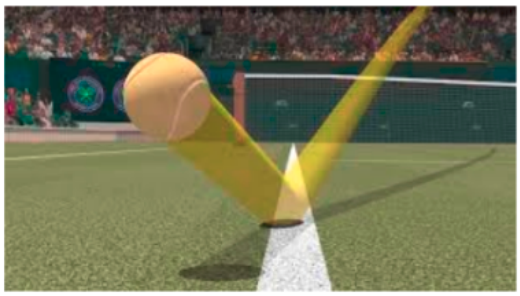
Use Case 1: Critical Decision Making

We've discussed this earlier and I'm sure most of you will be familiar with the hawk-eye in cricket.

The trajectory of the ball assists in making critical decisions during the match. For example, in cricket, during Leg Before Wicket (or LBW), the trajectory of the ball assists in deciding whether the ball has pitched inside or outside the stumps. It also includes information about the ball hitting the stumps or not.

Similarly, in tennis, during serves or a rally, the ball tracking system assists in knowing whether the ball has pitched inside or outside the permissible lines on the court:





Use Case 2: Identifying the Strong and Weak Zones of a Batsman

Every team has a set of match-winning players. Picking their wickets at the earliest opportunity is crucial for any team to win matches. With the help of ball-tracking technology, we can analyze the raw videos and generate heat maps.

From these heatmaps, we can easily identify the strong and weak zone of a batsman. This would help the team to develop a strategy against every player ahead of a match:

Can you think of other use cases of a ball tracking system in sports? Let me know in the comments section below!

Different Approaches to Ball Tracking Systems

There are different tracking algorithms as well as pre-trained models for tracking the object in a video. But, there are certain challenges with them when it comes to tracking a fast-moving cricket ball.

Here are the few challenges that we need to know prior to tracking a fast-moving ball in a cricket video.

- The cricket ball moves with a very high speed of around 130-150 kph. Due to this, the ball traces along its path
- The similar objects on the ground to that of the ball might be challenging to classify. For example, 30-yard dots in the field when viewed from the top almost look like a ball

Hence, in this article, I will focus on 2 simple approaches to track a fast-moving ball in a sports video:

- Sliding Window
- Segmentation by Color

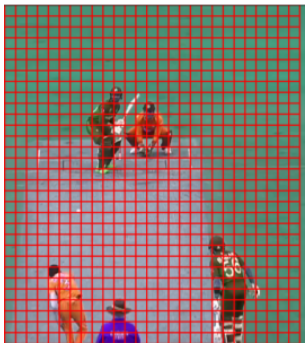
Let's discuss them in detail now.

Approach 1: Sliding Window

One of the simplest ways could be to break down the image into smaller patches, say 3 * 3 or 5 * 5 grids, and then classify every patch into one of 2 classes – whether a patch contains a ball or not. This approach is known as the sliding window approach as we are sliding the window of a patch across every part of an image.

> *Remember that the formation of grids can be overlapping as well. It all depends on the way you want to formulate the problem.*

Here is an example that showcases the non-overlapping grids:



This method is really simple and efficient. But, it's a time taking process as it considers several patches of the image. Another drawback of the sliding window approach is that it's expensive as it considers every patch of an image.

So next, I will discuss the alternative approach to the sliding window.

Approach 2: Segmentation by Color

Instead of considering every patch, we can reduce the patches for classification based on the color of the ball. Since we know the color of the ball, we can easily differentiate the patches that have a similar color to that of the ball from the rest of the patches.

This results in a fewer number of patches to classify. This process of combining similar parts of an image via color is known as **segmentation by color**.

Implementation – Develop Your First Ball Tracking System for Cricket in Python

Time to code! Let's develop a simple ball tracking system that tracks the ball on the pitch using Python. Download the necessary data files from here.

First, let's read a video file and save the frames to a folder:

```
1  import cv2
```

```python
import numpy as np
import imutils

video='28.mp4'

# Create a VideoCapture object and read from input file
# If the input is the camera, pass 0 instead of the video file name
cap = cv2.VideoCapture(video)
cnt=0

# Check if camera opened successfully
if (cap.isOpened()== False):
  print("Error opening video stream or file")

ret,first_frame = cap.read()

# Read until video is completed
while(cap.isOpened()):

  # Capture frame-by-frame
  ret, frame = cap.read()

  if ret == True:

    #removing scorecard
    roi = frame[:800,:]

    #cropping center of an image
    thresh=600
    end = roi.shape[1] - thresh
    roi = roi[:,thresh:end]

    cv2.imshow("image",roi)
    # Press Q on keyboard to  exit
    if cv2.waitKey(25) & 0xFF == ord('q'):
      break

    cv2.imwrite('frames/'+str(cnt)+'.png',roi)
    cnt=cnt+1

  # Break the loop
  else:
    break

cv2.destroyAllWindows()
```

view raw

11_0.py hosted with ❤ by **GitHub**

Reading frames:

```python
import matplotlib.pyplot as plt
import cv2
import numpy as np
import os
import re

#listing down all the file names
frames = os.listdir('frames/')
frames.sort(key=lambda f: int(re.sub('\D', '', f)))

#reading frames
images=[]
for i in frames:
    img = cv2.imread('frames/'+i)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.GaussianBlur(img,(25,25),0)
    images.append(img)

images=np.array(images)
```

view raw

11_1.py hosted with ❤ by **GitHub**

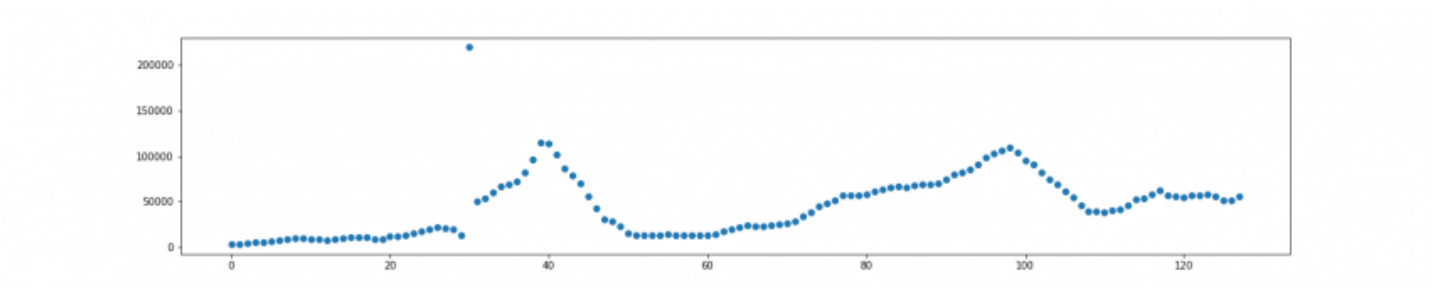
As our objective is to track the ball on the pitch, we need to extract the frames that contain the pitch. Here, I am using the concept of scene detection to accomplish the task:

```
nonzero=[]
for i in range((len(images)-1)):

    mask = cv2.absdiff(images[i],images[i+1])
    _ , mask = cv2.threshold(mask, 50, 255, cv2.THRESH_BINARY)
    num = np.count_nonzero((mask.ravel()))
    nonzero.append(num)


x = np.arange(0,len(images)-1)
y = nonzero

plt.figure(figsize=(20,4))
plt.scatter(x,y)
```

view raw

11_2.py hosted with ❤ by GitHub

Output:



The outlier in the plot indicates the frame number during which the scene changes. So, fix the threshold for obtaining the frames before a scene change:

```
threshold = 15 * 10e3
for i in range(len(images)-1):
    if(nonzero[i]>threshold):
        scene_change_idx = i
        break

frames = frames[:(scene_change_idx+1)]
```

view raw

11_3.py hosted with ❤ by GitHub

Now, we have obtained the frames that contain a pitch. Next, we will implement a segmentation approach that we discussed earlier in the article. Let's carry out all the steps of the approach for only a single frame now.

We will read the frame and apply Gaussian blur to remove noises in an image:

```
img= cv2.imread('frames/' + frames[10])
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray,(25,25),0)

plt.figure(figsize=(5,10))
plt.imshow(gray,cmap='gray')
```

view raw

11_4.py hosted with ❤ by GitHub

Output:

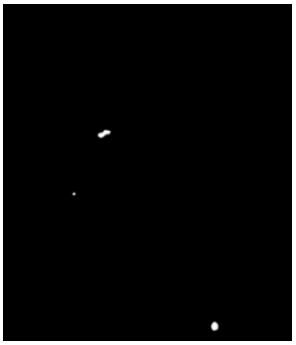
As the color of the ball is known, we can easily segment the white-colored objects in an image. Here, 200 acts as a threshold. Any pixel value below this 200 will be marked as 0 and above 200 will be marked as 255.

```
_ , mask = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)

plt.figure(figsize=(5,5))
plt.imshow(mask,cmap='gray')
```

view raw

11_5.py hosted with ❤ by **GitHub**

Output:



As you can see here, the white-colored objects are segmented. The white color indicates white colored objects and black indicates the rest of the colors. And that's it! We have separated the white-colored objects from the others.

Now, we will find the contours of segmented objects in an image:

```
image, contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

view raw

ball_tracking_1.py hosted with ❤ by **GitHub**

Draw the contours on the original image:

```
img_copy = np.copy(gray)
cv2.drawContours(img_copy, contours, -1, (0,255,0), 3)
plt.imshow(img_copy, cmap='gray')
```

view raw

ball_tracking_2.py hosted with ❤ by **GitHub**

Output:

Next, extract the patches from an image using the contours:

```
!rm -r patch/*

num=20
cnt=0
for i in range(len(contours)):
    x,y,w,h = cv2.boundingRect(contours[i])

    numer=min([w,h])
    denom=max([w,h])
    ratio=numer/denom

    if(x>=num and y>=num):
        xmin, ymin= x-num, y-num
        xmax, ymax= x+w+num, y+h+num
    else:
        xmin, ymin=x, y
        xmax, ymax=x+w, y+h

    if(ratio>=0.5 and ((w<=10) and (h<=10)) ):
        print(cnt,x,y,w,h,ratio)
        cv2.imwrite("patch/"+str(cnt)+".png",img[ymin:ymax,xmin:xmax])
        cnt=cnt+1
```

view raw

11_6.py hosted with ❤ by **GitHub**

It’s time to build an image classifier to identify the patch containing the ball.

Reading and preparing the dataset:

```
import os
import cv2
import numpy as np
import pandas as pd
folders=os.listdir('data/')

images=[]
labels= []
for folder in folders:
    files=os.listdir('data/'+folder)
    for file in files:
        img=cv2.imread('data/'+folder+'/'+file,0)
        img=cv2.resize(img,(25,25))

        images.append(img)
        labels.append(int(folder))

images = np.array(images)
features = images.reshape(len(images),-1)
```

view raw

11_7.py hosted with ❤ by **GitHub**

Split the dataset into train and validation:

```
from sklearn.model_selection import train_test_split
```

```
x_tr,x_val,y_tr,y_val = train_test_split(features,labels, test_size=0.2, stratify=labels,random_state=0)
```

view raw

ball_tracking_3.py hosted with ❤ by GitHub

Build a baseline model for identifying the patch containing ball:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(max_depth=3)
rfc.fit(x_tr,y_tr)
```

view raw

ball_tracking_4.py hosted with ❤ by GitHub

Evaluate the model on the validation data:

```
from sklearn.metrics import classification_report
y_pred = rfc.predict(x_val)
print(classification_report(y_val,y_pred))
```

view raw

ball_tracking_5.py hosted with ❤ by GitHub

Repeat the similar steps for each frame in a video followed by classification:

```
!rm -r ball/*
ball_df = pd.DataFrame(columns=['frame','x','y','w','h'])

for idx in range(len(frames)):

    img= cv2.imread('frames/' + frames[idx])
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray,(25, 25),0)
    _ , mask = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
    image, contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    !rm -r patch/*

    num=20
    cnt=0
    df = pd.DataFrame(columns=['frame','x','y','w','h'])
    for i in range(len(contours)):
        x,y,w,h = cv2.boundingRect(contours[i])

        numer=min([w,h])
        denom=max([w,h])
        ratio=numer/denom

        if(x>=num and y>=num):
            xmin, ymin= x-num, y-num
            xmax, ymax= x+w+num, y+h+num
        else:
            xmin, ymin= x,y
            xmax, ymax= x+w, y+h

        if(ratio>=0.5):
            #print(cnt,x,y,w,h,ratio)
            df.loc[cnt,'frame'] = frames[idx]
            df.loc[cnt,'x']=x
            df.loc[cnt,'y']=y
            df.loc[cnt,'w']=w
            df.loc[cnt,'h']=h

            cv2.imwrite("patch/"+str(cnt)+".png",img[ymin:ymax,xmin:xmax])
            cnt=cnt+1


    files=os.listdir('patch/')
    if(len(files)>0):

        files.sort(key=lambda f: int(re.sub('\D', '', f)))

        test=[]
        for file in files:
            img=cv2.imread('patch/'+file,0)
```

```
            img=cv2.resize(img,(25,25))
            test.append(img)

        test = np.array(test)

        test = test.reshape(len(test),-1)
        y_pred = rfc.predict(test)
        prob=rfc.predict_proba(test)

        if 0 in y_pred:
            ind = np.where(y_pred==0)[0]
            proba = prob[:,0]
            confidence = proba[ind]
            confidence = [i for i in confidence if i>0.7]
            if(len(confidence)>0):

                maximum = max(confidence)
                ball_file=files[list(proba).index(maximum)]

                img= cv2.imread('patch/'+ball_file)
                cv2.imwrite('ball/'+str(frames[idx]),img)

                no = int(ball_file.split(".")[0])
                ball_df.loc[idx]= df.loc[no]
            else:
                ball_df.loc[idx,'frame']=frames[idx]

        else:
            ball_df.loc[idx,'frame']=frames[idx]
```

view raw

11_8.py hosted with ❤ by **GitHub**

Have a glance at the frames containing the ball along with the location:

```
ball_df.dropna(inplace=True)
print(ball_df)
```

view raw

ball_tracking_6.py hosted with ❤ by **GitHub**

	frame	x	y	w	h
4	4.png	190	481	9	10
5	5.png	191	468	8	9
6	6.png	192	459	7	7
7	7.png	193	453	6	6
8	8.png	194	450	6	5
9	9.png	195	450	6	5
10	10.png	196	453	7	6
11	11.png	197	459	7	7
12	12.png	199	469	7	7
13	13.png	201	482	9	10
15	15.png	208	520	12	14
16	16.png	215	548	14	15
17	17.png	223	579	14	19
18	18.png	233	616	15	23
19	19.png	239	624	15	21
20	20.png	243	586	11	16

Next, we will draw the bounding box around the frames that contain the ball and save it back to the folder:

```
files = ball_df['frame'].values

num=10
for idx in range(len(files)):

    #draw contours
    img = cv2.imread('frames/'+files[idx])
```

```
    x=ball_df.loc[idx,'x']
    y=ball_df.loc[idx,'y']
    w=ball_df.loc[idx,'w']
    h=ball_df.loc[idx,'h']

    xmin=x-num
    ymin=y-num
    xmax=x+w+num
    ymax=y+h+num

    cv2.rectangle(img, (xmin, ymin), (xmax, ymax), (255,0,0), 2)
    cv2.imwrite("frames/"+files[idx],img)
```

view raw

11_9.py hosted with ❤ by **GitHub**

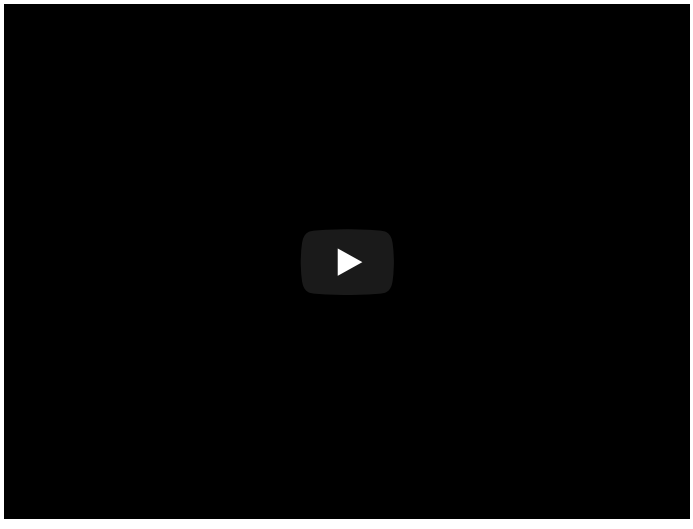
Let's convert back the frames into a video now:

```
frames = os.listdir('frames/')
frames.sort(key=lambda f: int(re.sub('\D', '', f)))

frame_array=[]

for i in range(len(frames)):
    #reading each files
    img = cv2.imread('frames/'+frames[i])
    height, width, layers = img.shape
    size = (width,height)
    #inserting the frames into an image array
    frame_array.append(img)

out = cv2.VideoWriter('28.mp4',cv2.VideoWriter_fourcc(*'DIVX'), 25, size)

for i in range(len(frame_array)):
    # writing to a image array
    out.write(frame_array[i])
out.release()
```

view raw

11_12.py hosted with ❤ by **GitHub**

Output:



How cool is that? Congratulations on building your own ball tracking system for cricket!

End Notes

That's it for today! This brings us to the end of the tutorial on ball tracking for cricket. Please keep in mind that a baseline model is built for image classification tasks. But there is still a lot of room for improving our

model. And also, there are few hyperparameters in this approach such as the size of the Gaussian filter and the thresholding value that must be adjusted depending on the type of video.

What are your thoughts on the system we built? Share your ideas and feedback in the comments section below and let's discuss.

Article Url - https://www.analyticsvidhya.com/blog/2020/03/ball-tracking-cricket-computer-vision/

Aravind Pai

Aravind is a sports fanatic. His passion lies in developing data-driven products for the sports domain. He strongly believes that analytics in sports can be a game-changer