VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering (SENSE)

PROJECT BASED LEARNING (J Component) - REPORT

COURSE CODE / NAME	CSE3017 - COMPUTER VISION	
PROGRAM / YEAR	B.Tech (ECM)	
LAST DATE FOR REPORT SUBMISSION	10TH APRIL, 2019	
DATE OF SUBMISSION	01ST APRIL 2019	
TEAM MEMBERS DETAILS	**REGISTER NO.**	**NAME**
	16BLC1021	RAJA HARIKESH N V
	16BLC1092	PRANAV N
	16BLC1135	PRAVEEN KUMAR R
PROJECT TITLE	**OPTICAL PLAYER TRACKING SYSTEM IN SOCCER**	
COURSE HANDLER'S NAME	**PROF. KARTHIK.R**	**REMARKS**
COURSE HANDLER'S SIGN		

Table of Contents:

ABSTRACT:

- Performance Analysis of players is a blooming field in the domain of Sports Analytics. Extensive research is being carried out in finding numerous but efficient ways of tracking a player so that his/her performance on the field can be monitored and adjusted to suit the demands of the team.
- This project is not only limited to offline videos, but can be extended to live streams.

OBJECTIVE:

- To develop a mechanism which detects and tracks all on-field participants of a soccer match namely players, referee, and balls.
- Extend the model to derive useful information like heat maps, total distance covered, average team formation etc.

PROGRAM FLOW:

1. Perspective transformation is applied to ensure capturing of complete football pitch. This transformation is very helpful in cases where panoramic videos have player contours which are too small to be detected by naked eye and even the computers.
2. Once the first frame after the transformation is loaded, a Region of Interest selector is triggered to enable manual box-bounding of players in the frame.
3. Once the ROI is selected for intended players, then the app starts tracking all those players frame by frame. The accuracy and tracking speed depends on the type of tracking algorithm chosen.
4. 3 windows display the tracking, radar and heat map of players respectively.

ALGORITHMS:

List of OpenCV tracking algorithms:

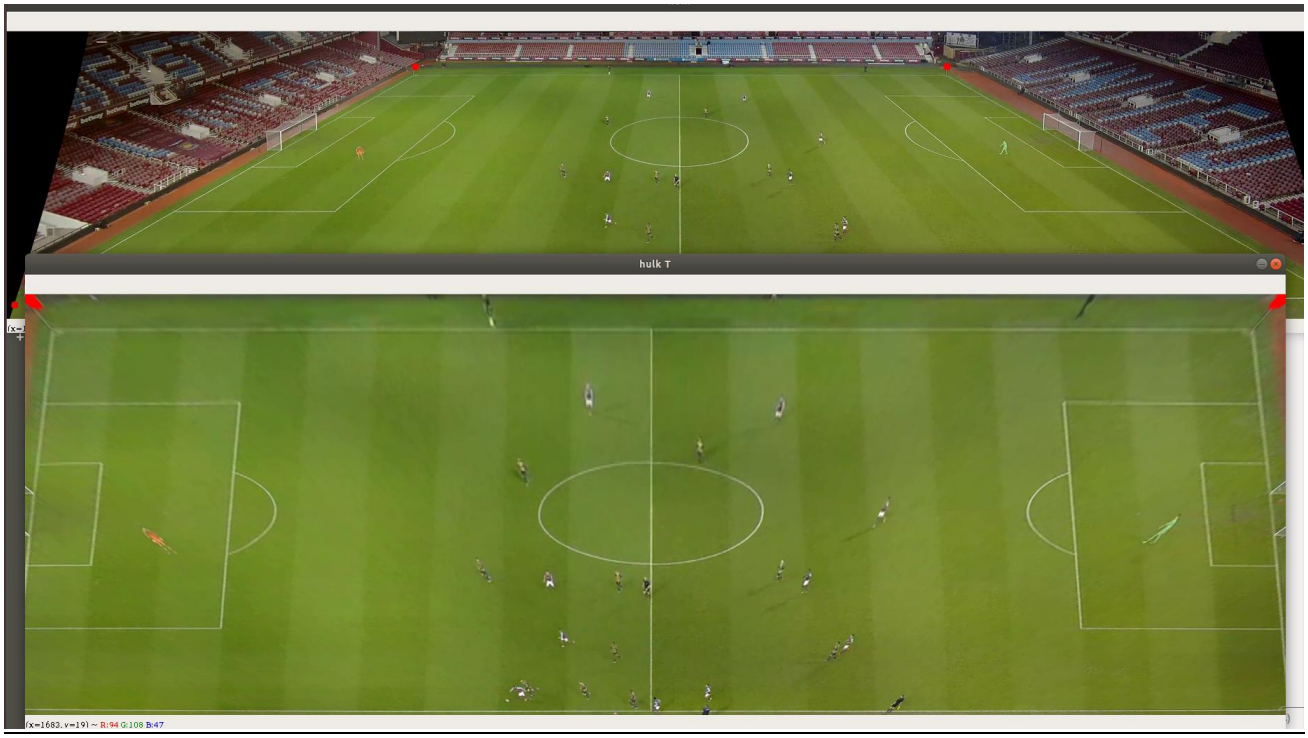
1. **BOOSTING Tracker:** Based on the same algorithm used to power the machine learning behind Haar cascades (AdaBoost), but like Haar cascades, is over a decade old. This tracker is slow and doesn't work very well. Interesting only for legacy reasons and comparing with other algorithms. *(minimum OpenCV 3.0.0)*
2. **MIL Tracker:** Better accuracy than BOOSTING tracker but does a poor job of reporting failure. *(minimum OpenCV 3.0.0)*
3. **KCF Tracker:** Kernelized Correlation Filters. Faster than BOOSTING and MIL. Similar to MIL and KCF, does not handle full occlusion well. *(minimum OpenCV 3.1.0)*
4. **CSRT Tracker:** Discriminative Correlation Filter (with Channel and Spatial Reliability). Tends to be more accurate than KCF but slightly slower. *(minimum OpenCV 3.4.2)*
5. **MedianFlow Tracker:** Does a nice job reporting failures; however, if there is too large of a jump in motion, such as fast moving objects, or objects that change quickly in their appearance, the model will fail. *(minimum OpenCV 3.0.0)*
6. **TLD Tracker:** The TLD tracker is incredibly prone to false-positives. *(minimum OpenCV 3.0.0)*
7. **MOSSE Tracker:** Very, *very* fast. Not as accurate as CSRT or KCF but a good choice if pure speed is needed. *(minimum OpenCV 3.4.1)*
8. **GOTURN Tracker:** The only deep learning-based object detector included in OpenCV. It requires additional model files to run .*(minimum OpenCV 3.2.0)*

Since CSRT provides high tracking accuracy and we can tolerate low FPS throughput (30fps least), we have gone for the same algorithm.

CSRT:

In the Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR), we use the spatial reliability map for adjusting the filter support to the part of the selected region from the frame for tracking. This ensures enlarging and localization of the selected region and improved tracking of the non-rectangular regions or objects. It uses only 2 standard features (HoGs and Colornames). It also operates at a comparatively lower fps (25 fps) but gives higher accuracy for object tracking.

PERSPECTIVE TRANSFORMATION:



getPerspectiveTransform

Calculates a perspective transform from four pairs of the corresponding points.

C++: Mat **getPerspectiveTransform**(InputArray **src**, InputArray **dst**)

C++: Mat **getPerspectiveTransform**(const Point2f **src**[], const Point2f **dst**[])

Python: cv2.**getPerspectiveTransform**(src, dst) → retval

C: CvMat* **cvGetPerspectiveTransform**(const CvPoint2D32f* **src**, const CvPoint2D32f* **dst**, CvMat* **map_matrix**)

Python: cv.**GetPerspectiveTransform**(src, dst, mapMatrix) → None

Parameters:
- **src** – Coordinates of quadrangle vertices in the source image.
- **dst** – Coordinates of the corresponding quadrangle vertices in the destination image.

The function calculates the 3×3 matrix of a perspective transform so that:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

where

$$\texttt{dst}(i) = (x'_i, y'_i), \texttt{src}(i) = (x_i, y_i), i = 0, 1, 2, 3$$

warpPerspective

Applies a perspective transformation to an image.

C++: void **warpPerspective**(InputArray **src**, OutputArray **dst**, InputArray **M**, Size **dsize**, int **flags**=INTER_LINEAR, int **borderMode**=BORDER_CONSTANT, const Scalar& **borderValue**=Scalar())

Python: `cv2.warpPerspective`(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst

C: void **cvWarpPerspective**(const CvArr* **src**, CvArr* **dst**, const CvMat* **map_matrix**, int **flags**=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS, CvScalar **fillval**=cvScalarAll(0))

Python: `cv.WarpPerspective`(src, dst, mapMatrix, flags=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS, fillval=(0, 0, 0, 0)) → None

Parameters:
- **src** – input image.
- **dst** – output image that has the size `dsize` and the same type as `src` .
- **M** – 3×3 transformation matrix.
- **dsize** – size of the output image.
- **flags** – combination of interpolation methods (`INTER_LINEAR` or `INTER_NEAREST`) and the optional flag `WARP_INVERSE_MAP`, that sets `M` as the inverse transformation (**dst** $\rightarrow$ **src**).
- **borderMode** – pixel extrapolation method (`BORDER_CONSTANT` or `BORDER_REPLICATE`).
- **borderValue** – value used in case of a constant border; by default, it equals 0.

The function `warpPerspective` transforms the source image using the specified matrix:

$$\texttt{dst}(x,y) = \texttt{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

when the flag `WARP_INVERSE_MAP` is set. Otherwise, the transformation is first inverted with `invert()` and then put in the formula above instead of `M` . The function cannot operate in-place.

CODING:

```
from __future__ import print_function
import numpy as np
import cv2
import os
import sys
from random import randint

trackerTypes = ['BOOSTING', 'MIL', 'KCF','TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE',
'CSRT']

def createTrackerByName(trackerType):
  # Create a tracker based on tracker name
  if trackerType == trackerTypes[0]:
    tracker = cv2.TrackerBoosting_create()
  elif trackerType == trackerTypes[1]:
    tracker = cv2.TrackerMIL_create()
  elif trackerType == trackerTypes[2]:
    tracker = cv2.TrackerKCF_create()
  elif trackerType == trackerTypes[3]:
    tracker = cv2.TrackerTLD_create()
```

```
  elif trackerType == trackerTypes[4]:
    tracker = cv2.TrackerMedianFlow_create()
  elif trackerType == trackerTypes[5]:
    tracker = cv2.TrackerGOTURN_create()
  elif trackerType == trackerTypes[6]:
    tracker = cv2.TrackerMOSSE_create()
  elif trackerType == trackerTypes[7]:
    tracker = cv2.TrackerCSRT_create()
  else:
    tracker = None
    print('Incorrect tracker name')
    print('Available trackers are:')
    for t in trackerTypes:
      print(t)

  return tracker

if __name__ == '__main__':

  #print("Default tracking algoritm is CSRT \n"
  #      "Available tracking algorithms are:\n")
  #for t in trackerTypes:
  #    print(t)

  trackerType = 'CSRT'
  videoPath = sys.argv[1]

  # Create a video capture object to read videos
  cap = cv2.VideoCapture(videoPath)
  # Set video to load
  success, frame123 = cap.read()
  hulk = cv2.imread('pitch.png')
  original = cv2.imread('pitch.png')
  f = open( 'file.txt', 'w' )
  # quit if unable to read the video file
  if not success:
    print('Failed to read video')
    sys.exit(1)

  ## Select boxes
  bboxes = []
  colors = []
  p=0
  # OpenCV's selectROI function doesn't work for selecting multiple objects in Python
  # So we will call this function in a loop till we are done selecting all objects
  while True:
    # draw bounding boxes over objects
```

```
    # selectROI's default behaviour is to draw box starting from the center
    # when fromCenter is set to false, you can draw box starting from top left corner
    # hardcoded for pano.mp4
    cv2.circle(frame123, (583, 50), 5, (0, 0, 255), -1)
    cv2.circle(frame123, (1342, 50), 5, (0, 0, 255), -1)
    cv2.circle(frame123, (11, 390), 5, (0, 0, 255), -1)
    cv2.circle(frame123, (1911, 390), 5, (0, 0, 255), -1)
    pts1 = np.float32([[583, 50], [1342, 50], [25, 390], [1911, 390]])
    pts2 = np.float32([[0, 0], [1800, 0], [0, 600], [1800, 600]])
    matrix = cv2.getPerspectiveTransform(pts1, pts2)

    frame = cv2.warpPerspective(frame123, matrix, (1800, 600))
    print('Select the 10 Home Team Outfield Players')
    print('Select the 10 Away Team Outfield Players')
    bbox = cv2.selectROI('MultiTracker', frame)
    bboxes.append(bbox)
    if (p<10):
      colors.append((0,0,255))
    else:
      colors.append((255,0,0))
    print("Press q to quit selecting boxes and start tracking")
    print("Press any other key to select next object")
    p=p+1
    k = cv2.waitKey(0) & 0xFF
    if (k == 113):  # q is pressed
      break
print('Selected bounding boxes {}'.format(bboxes))

## Initialize MultiTracker
# There are two ways you can initialize multitracker
# 1. tracker = cv2.MultiTracker("CSRT")
# All the trackers added to this multitracker
# will use CSRT algorithm as default
# 2. tracker = cv2.MultiTracker()
# No default algorithm specified

# Initialize MultiTracker with tracking algo
# Specify tracker type

# Create MultiTracker object
multiTracker = cv2.MultiTracker_create()

# Initialize MultiTracker
for bbox in bboxes:
  multiTracker.add(createTrackerByName(trackerType), frame, bbox)
```

```
  # Process video and track objects
  while cap.isOpened():
    success, frame123 = cap.read()
    # hardcoded for pano.mp4
    cv2.circle(frame123, (583, 50), 5, (0, 0, 255), -1)
    cv2.circle(frame123, (1342, 50), 5, (0, 0, 255), -1)
    cv2.circle(frame123, (11, 390), 5, (0, 0, 255), -1)
    cv2.circle(frame123, (1911, 390), 5, (0, 0, 255), -1)
    pts1 = np.float32([[583, 50], [1342, 50], [25, 390], [1911, 390]])
    pts2 = np.float32([[0, 0], [1800, 0], [0, 600], [1800, 600]])
    matrix = cv2.getPerspectiveTransform(pts1, pts2)

    frame = cv2.warpPerspective(frame123, matrix, (1800, 600))
    if not success:
      break

    # get updated location of objects in subsequent frames
    success, boxes = multiTracker.update(frame)
    l,b ,channels = frame.shape  #newaddition
    hulk = cv2.resize(hulk,(b,l))
    original = cv2.resize(original,(b,l))
    radar = original.copy()
    # draw tracked objects
    for i, newbox in enumerate(boxes):
      p1 = (int(newbox[0]), int(newbox[1]))
      p2 = (int(newbox[0] + newbox[2]), int(newbox[1] + newbox[3]))
      cv2.rectangle(frame, p1, p2, colors[i], 2, 1)

      if (i<10):
        cv2.putText(frame, str(i), (int(newbox[0]), int(newbox[1])),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 1, cv2.LINE_AA)
        #cv2.putText(hulk, 'Home', p1, cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2,
cv2.LINE_AA)#newaddition
        #cv2.putText(hulk, str(i), (int(newbox[0]), int(newbox[1])),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 1, cv2.LINE_AA)
        overlay=hulk.copy()
        cv2.circle(overlay,(int(newbox[0]), int(newbox[1])), 10, (0,0,255), -
1)#newaddition
        #cv2.applyColorMap(hulk,cv2.COLORMAP_AUTUMN)
        cv2.circle(radar,(int(newbox[0]), int(newbox[1])), 10, (0,0,255), -
1)#newaddition
        alpha = 0.1  # Transparency factor.

        # Following line overlays transparent rectangle over the image
        hulk = cv2.addWeighted(overlay, alpha, hulk, 1 - alpha, 0)
```

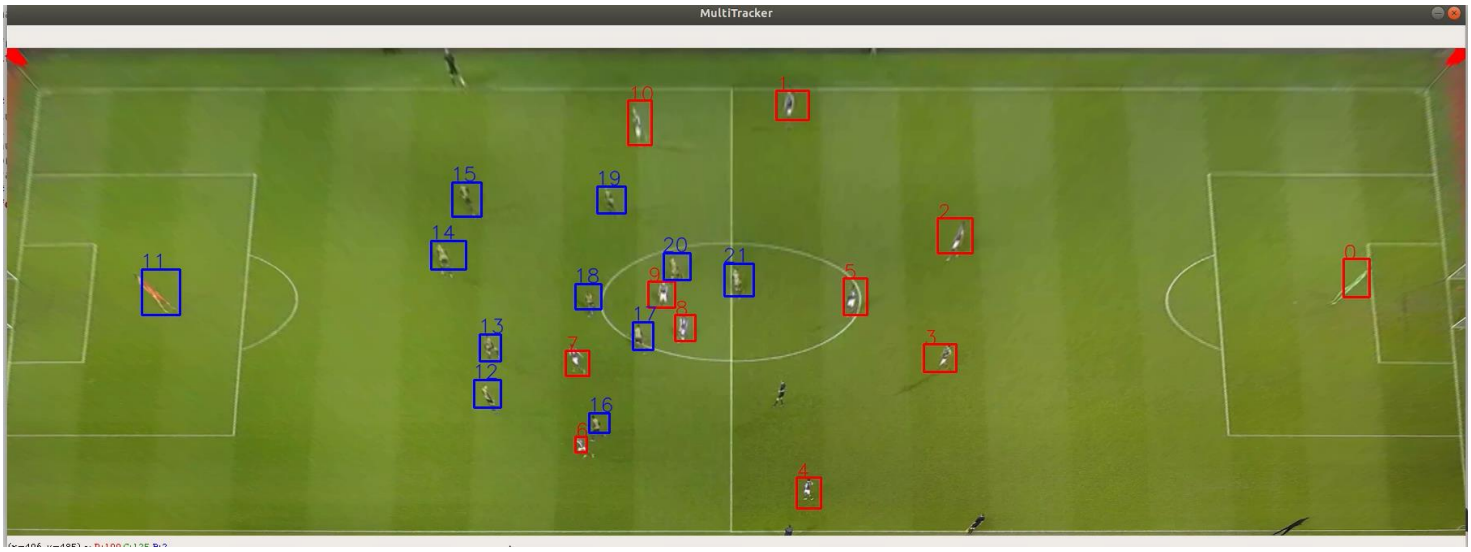
```
        f.write( 'Home: Player '+str(i)+' x,y:
'+str(int(newbox[0]))+','+str(int(newbox[1])) + '\n' )   #####
      else:
        cv2.putText(frame, str(i), (int(newbox[0])-2, int(newbox[1])-2),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,0,0), 1, cv2.LINE_AA)
        #cv2.putText(hulk, 'Away', p1, cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2,
cv2.LINE_AA)#newaddition
        #cv2.putText(hulk, str(i), (int(newbox[0]), int(newbox[1])),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,0,0), 1, cv2.LINE_AA)
        overlay=hulk.copy()
        cv2.circle(overlay,(int(newbox[0]), int(newbox[1])), 10, (255,0,0), -
1)#newaddition
        #cv2.applyColorMap(hulk,cv2.COLORMAP_AUTUMN)
        cv2.circle(radar,(int(newbox[0]), int(newbox[1])), 10, (255,0,0), -
1)#newaddition
        alpha = 0.1  # Transparency factor.

        # Following line overlays transparent rectangle over the image
        hulk = cv2.addWeighted(overlay, alpha, hulk, 1 - alpha, 0)
        f.write( 'Away: Player '+str(i)+' x,y:
'+str(int(newbox[0]))+','+str(int(newbox[1])) + '\n' )   #####
    # show frame
    cv2.imshow('MultiTracker', frame)
    cv2.imshow('HeatMap',hulk) #newaddition
    cv2.imshow('Radar',radar)
    # quit on ESC button
    if cv2.waitKey(1) & 0xFF == 27:  # Esc pressed
        break
f.close()
```

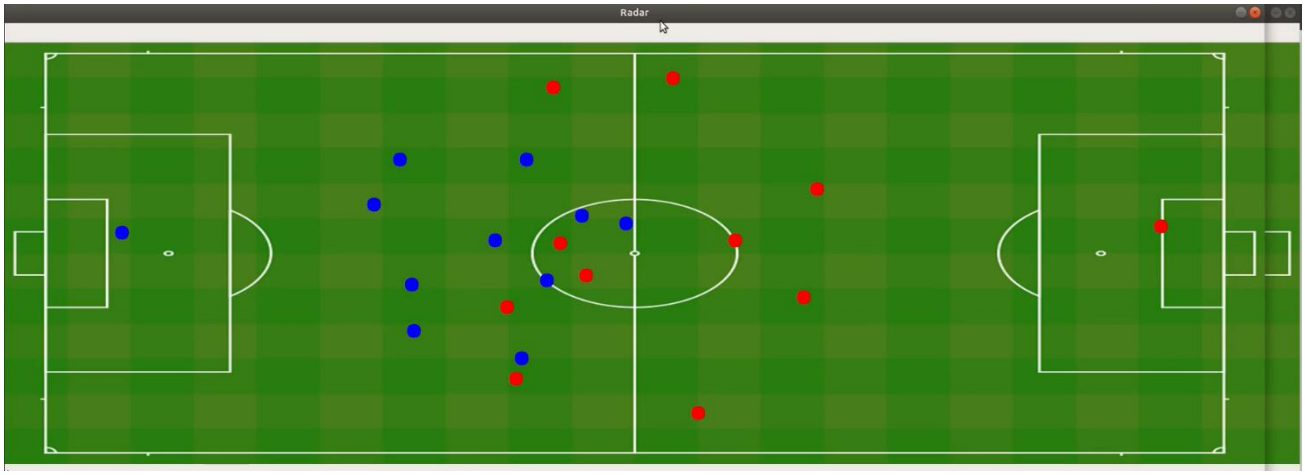
IMPLEMENTATION:

1. Selection of players using ROI selector

2. Multitracker window



3. Radar window



4. Heatmap window

5. Location coordinates stored in a text file

```
Home: Player 0 x,y: 1650,260
Home: Player 1 x,y: 954,50
Home: Player 2 x,y: 1159,210
Home: Player 3 x,y: 1140,363
Home: Player 4 x,y: 984,529
Home: Player 5 x,y: 1038,282
Home: Player 6 x,y: 721,479
Home: Player 7 x,y: 709,375
Home: Player 8 x,y: 828,330
Home: Player 9 x,y: 793,287
Home: Player 10 x,y: 779,64
Away: Player 11 x,y: 167,270
Away: Player 12 x,y: 582,410
Away: Player 13 x,y: 581,346
Away: Player 14 x,y: 524,229
Away: Player 15 x,y: 559,166
Away: Player 16 x,y: 732,450
Away: Player 17 x,y: 774,339
Away: Player 18 x,y: 700,285
Away: Player 19 x,y: 738,165
Away: Player 20 x,y: 819,248
Away: Player 21 x,y: 886,258
Home: Player 0 x,y: 1650,259
Home: Player 1 x,y: 952,51
```

FUTURE WORK:

1. The current work is limited to a single panoramic video of soccer game.
2. Also the perspective transformation can be automated for any kind of panoramic video.
3. Currently only heat-maps of players are stored in a text file and displayed in a windows of the app. More features like distance covered and passing lines of each player, average team formation can be derived from the video.

CONCLUSION:

Tracking the players has been done successfully with the help of OpenCV APIs. CSRT Algorithm is found to be more robust to Occlusion than other trackers available and has better tracking accuracy than 4/7 algorithms. 3 windows respectively display the real time position of players, their heatmap and bounding box in each frame. The location details are stored in a text file, in case it's required in the future.

REFERENCES:

1. https://docs.opencv.org/3.4/d5/d07/tutorial_multitracker.html

2. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html