A Project Report

On

Automatically Recognizing On Ball Screens

BY

Akash Sebastian

2015A7PS0967H

Under the supervision of

Prof. Tathagat Ray

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

CS F376: DESIGN PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(May 2019)

ACKNOWLEDGMENTS

This project work is carried out with the support of the department of Computer Science, Bits-Pilani, Hyderabad campus, India. I take this opportunity to express my profound gratitude and regard to my guide Prof. Tathagat Ray for the exemplary guidance, constant encouragement, and unwavering support throughout the project.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Recognizing On Ball Screens**" submitted by Mr. Akash Sebastian (ID No. 2015A7PS0967H) in partial fulfillment of the requirements of the course CS F376, Design Project Course, embodies the work done by him under my supervision and guidance.

Date: 9th May 2019 **(Tathagat Ray)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

Analytics is taking the NBA by a storm. Every decision, be it in coaching or front office, is being taken after consulting what the numbers have to say. Observing this analytical boom, the league, to promote research, has allowed researchers to access tracking data of official NBA games for free. Using this tracking data, this project will facilitate the automatic detection of screening action. A screen is used in almost every half court play and the detection of these would be able to facilitate much more research and analysis in these areas such as the defence of the screens, the points the offense is able to score per possession given a particular type of defensive scheme, etc. The data will first be passed undergo data segmentation wherein data points where a screen may have potentially occurred will be separated from the data. To detect the screens, machine learning classifiers on top of a rule-based algorithm will be used on the segmented data.

Data

The data that this project uses is provided by STATS SportsVU. Six cameras are installed in every NBA arena and these specialized cameras track the x and y coordinates of the ten players on the court and the x, y and z coordinates of the ball at 25 frames per second. This data is compiled as a .json file for researcher's use. Included in the json file are all the players in the roster of both teams that are playing. The entire game is broken up into multiple events of varying length in real time. Each game has around 450 – 500 events. A snippet of the data is shown below. The event ID indicates the event number. The rosters of the home and away teams are listed with each player getting a corresponding player ID.


```
{'eventId': '2', 'visitor': {'name': 'Detroit Pistons', 'teamid': 1610612765, 'abbreviation': 'DET', 'players':
[{'lastname': 'Blake', 'firstname': 'Steve', 'playerid': 2581, 'jersey': '22', 'position': 'G'}, {'lastname': 'Ily
asova', 'firstname': 'Ersan', 'playerid': 101141, 'jersey': '23', 'position': 'F'}, {'lastname': 'Anthony', 'first
name': 'Joel', 'playerid': 201202, 'jersey': '50', 'position': 'C'}, {'lastname': 'Meeks', 'firstname': 'Jodie',
'playerid': 201975, 'jersey': '20', 'position': 'G'}, {'lastname': 'Morris', 'firstname': 'Marcus', 'playerid': 20
2694, 'jersey': '13', 'position': 'F'}, {'lastname': 'Jackson', 'firstname': 'Reggie', 'playerid': 202704, 'jerse
y': '1', 'position': 'G'}, {'lastname': 'Drummond', 'firstname': 'Andre', 'playerid': 203083, 'jersey': '0', 'posi
tion': 'C'}, {'lastname': 'Baynes', 'firstname': 'Aron', 'playerid': 203382, 'jersey': '12', 'position': 'F-C'},
{'lastname': 'Caldwell-Pope', 'firstname': 'Kentavious', 'playerid': 203484, 'jersey': '5', 'position': 'G'}, {'la
stname': 'Bullock', 'firstname': 'Reggie', 'playerid': 203493, 'jersey': '25', 'position': 'F'}, {'lastname': 'Din
widdie', 'firstname': 'Spencer', 'playerid': 203915, 'jersey': '8', 'position': 'G'}, {'lastname': 'Johnson', 'fir
stname': 'Stanley', 'playerid': 1626169, 'jersey': '3', 'position': 'F'}, {'lastname': 'Hilliard', 'firstname': 'D
arrun', 'playerid': 1626199, 'jersey': '6', 'position': 'F'}]}, 'home': {'name': 'Atlanta Hawks', 'teamid': 161061
2737, 'abbreviation': 'ATL', 'players': [{'lastname': 'Korver', 'firstname': 'Kyle', 'playerid': 2594, 'jersey':
'26', 'position': 'G'}, {'lastname': 'Sefolosha', 'firstname': 'Thabo', 'playerid': 200757, 'jersey': '25', 'posit
ion': 'G-F'}, {'lastname': 'Millsap', 'firstname': 'Paul', 'playerid': 200794, 'jersey': '4', 'position': 'F'},
{'lastname': 'Horford', 'firstname': 'Al', 'playerid': 201143, 'jersey': '15', 'position': 'F-C'}, {'lastname': 'S
plitter', 'firstname': 'Tiago', 'playerid': 201168, 'jersey': '11', 'position': 'F-C'}, {'lastname': 'Teague', 'fi
rstname': 'Jeff', 'playerid': 201952, 'jersey': '0', 'position': 'G'}, {'lastname': 'Mack', 'firstname': 'Shelvi
```

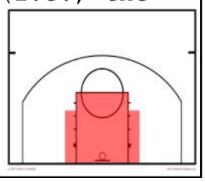

Each moment has a player ID along with the x, y and z coordinates of that player (as the z coordinates of the player isn't tracked, that value is zero for all players). The ball's player ID is –1 and has a value for the z coordinate.


```
3118, 'jersey': '32', 'position': 'F'}, {'lastname': 'Bazemore', 'firstname': 'Kent', 'playerid': 203145, 'jerse
y': '24', 'position': 'G'}, {'lastname': 'Holiday', 'firstname': 'Justin', 'playerid': 203200, 'jersey': '7', 'pos
ition': 'G'}, {'lastname': 'Schroder', 'firstname': 'Dennis', 'playerid': 203471, 'jersey': '17', 'position':
'G'}, {'lastname': 'Muscala', 'firstname': 'Mike', 'playerid': 203488, 'jersey': '31', 'position': 'F-C'}, {'lastn
ame': 'Patterson', 'firstname': 'Lamar', 'playerid': 203934, 'jersey': '13', 'position': 'G-F'}]}, 'moments': [[1,
1445991114020, 717.41, 21.4, None, [[-1, -1, 66.19088, 20.37273, 1.94592], [1610612737, 2594, 28.5862, 21.64312,
0.0], [1610612737, 200794, 47.37216, 24.03445, 0.0], [1610612737, 201143, 24.87104, 24.38831, 0.0], [1610612737, 2
01952, 33.45193, 25.15548, 0.0], [1610612737, 203145, 25.51108, 32.80861, 0.0], [1610612765, 101141, 49.06854, 29.
63809, 0.0], [1610612765, 202704, 66.11508, 19.05359, 0.0], [1610612765, 202694, 29.75039, 30.73939, 0.0], [161061
2765, 203484, 35.09886, 26.92302, 0.0], [1610612765, 203083, 33.69503, 25.29581, 0.0]]], [1, 1445991114060, 717.3
7, 21.36, None, [[-1, -1, 66.01261, 20.43085, 2.29823], [1610612737, 2594, 28.43447, 22.00489, 0.0], [1610612737,
200794, 46.95033, 24.04033, 0.0], [1610612737, 201143, 24.56331, 24.41702, 0.0], [1610612737, 201952, 33.40307, 2
5.11709, 0.0], [1610612737, 203145, 25.18495, 32.62304, 0.0], [1610612765, 101141, 48.68985, 29.58211, 0.0], [1610
612765, 202704, 65.69977, 19.15132, 0.0], [1610612765, 202694, 29.32868, 30.40703, 0.0], [1610612765, 203484, 34.7
2768, 27.32228, 0.0], [1610612765, 203083, 33.50527, 25.43572, 0.0]]], [1, 1445991114100, 717.33, 21.32, None, [[-
1, -1, 65.69083, 20.47676, 2.66002], [1610612737, 2594, 28.29376, 22.36607, 0.0], [1610612737, 200794, 46.54675, 2
4.00085, 0.0], [1610612737, 201143, 24.29185, 24.5029, 0.0], [1610612737, 201952, 33.34369, 25.0202, 0.0], [161061
2737, 203145, 24.86938, 32.38676, 0.0], [1610612765, 101141, 48.29977, 29.53183, 0.0], [1610612765, 202704, 65.280
51, 19.26021, 0.0], [1610612765, 202694, 28.97311, 30.12944, 0.0], [1610612765, 203484, 34.31801, 27.67583, 0.0],
[1610612765, 203083, 33.33357, 25.51744, 0.0]]], [1, 1445991114140, 717.29, 21.28, None, [[-1, -1, 65.25718, 20.64
```

Data Segmentation

Since the data is very large and going through every moment of every event of every game would be very time consuming, the data is passed through a segmentor which identifies which moments could potentially contain a screen action. The list of rules that need to be passed for a moment to get segmented is shown below.

```
IF [
      there is a player who is dribbling the ball (i.e., the ball-handler)
   AND there is an offensive player within 10 feet of the ball-handler (i.e.,
      the screener)
   AND the screener is not in or near the paint (see in-set figure for
      definition of "near")
   AND the screener is no more than 2 ft. further from the basket than the
      ball-handler is
   AND there is a defensive player <= 12 ft. from the ball-handler (i.e., the
      on-ball defender)
   AND the basketball is not in the paint
] for >=13 consecutive frames
AND the ball-handler is the same for every frame
THEN
  identify as action
```



The key to each rule is either the absolute position of a given player or the ball or the relative position of a player with respect to either another player or the ball. To calculate this information, the euclidian_distances function in the scikit-learn library is used. This function given two arrays outputs a 2D array with all the distances of the first array relative to the second element by element. Passing both arrays as the same one with the ten player's coordinates and the ball's coordinates every frame resulting in a 2D array with the relative distances of every player and the ball with respect to each other.

distances = euclidean_distances(coords, coords)

Once the distances are found, the ball handler is identified as the player with the least distance to the ball.

ball_handler = np.argmin(distances[-1][:-1])

For the first condition, the ball handler must be dribbling the ball. To check this, change in z coordinate of the ball is observed across consecutive frames. If there is a change, the ball is deemed to be dribbled.

```
if radius != self.ball_radius:
    self.ball_radius = radius
    flag[0] = 1
```

For the second condition, categorization of the two teams needed to be done first. The first five entries in the 2D array are of one team and the next five of the other. So, if the ball handler's value is less than five, the distance between an offensive player should be checked from entries 0 to 4 in the array.

```
if ball_handler < 5:
    if np.any(np.array([x for i, x in enumerate(distances[ball_handler][:5]) if i != ball_handler])<10) and self.screener == np.argmin(np.array([x for i, x in enumerate(distances[ball_handler][:5]) if i != ball_handler])):
        flag[1] = 1
else:
    if np.any(np.array([x for i, x in enumerate(distances[ball_handler][5:-1]) if i != ball_handler])<10) and self.screener == np.argmin(np.array([x for i, x in enumerate(distances[ball_handler][5:-1]) if i != ball_handler])):
        flag[1] = 1
```

Storing the Results

Likewise, all six conditions are checked and if all flags are positive, the frame count is incremented. If the frame count reaches 13, the game ID is noted along with the event ID and the moment count at which the segmentation began.

```
row = [self.path_to_json, self.event_index, i]

with open('screen_segmentation.csv','a') as csvFile:

    writer = csv.writer(csvFile)

    writer.writerow(row)
```

The information is stored in screen_segmentation.csv. The game ID, event number and the frame number at which the screen starts is all stored in the csv file. This segmentor is subsequently run for all the games (636) available. The segmented data is then passed through a more stringent test to check if screens really occur or not in these plays.

Deploying the Segmentor

The segmentor was run on a server. All 636 game zip files were transferred to the server through scp and then unzipped. The format of running the script is:

python3 main.py --path=*pathtojson* --event=*eventnumber*

To run all the 636 games with all the events, a wrapper script was written to call the main.py script multiple times.

```
path='.'

files=[]

for r,d,f in os.walk(path):

    for file in f:

        if '.json' in file:

            files.append(file)
```

This code gets all the json files in an array. Then, looping through the array, for each json file we must find the total number events being stored in the json file.

```
for num,file_name in enumerate(files):

    df = pd.read_json('Extract/' + file_name)

    length = len(df)

    for i in range(length):

        cmd = 'python3 main.py --path=Extract/' + file_name + ' --event=' + str(i)

        os.system(cmd)
```

This code reads the json file to find the length which is the number of events that the json file has. Then, each file along with each event that that file contains is called. This script is then executed on the server and the results are stored in screen_segmentation.csv. A sample of the file is shown below.

```
Game_ID,Event_Number,Frame_Number
extract/0021500403.json,0,12
extract/0021500403.json,0,25
extract/0021500403.json,0,44
extract/0021500403.json,0,79
extract/0021500403.json,1,12
extract/0021500403.json,1,25
extract/0021500403.json,1,44
extract/0021500403.json,1,79
extract/0021500403.json,1,371
extract/0021500403.json,1,384
extract/0021500403.json,1,397
extract/0021500403.json,1,410
extract/0021500403.json,1,423
extract/0021500403.json,2,46
extract/0021500403.json,2,59
extract/0021500403.json,2,72
extract/0021500403.json,2,85
```

False Positives after Segmentation

Three types of actions predominantly get caught by the segmentor.

1. A legitimate screen where the on ball defender gets screened by an offensive player.
2. An action where the ball handler and another offensive player cross paths to go to their respective spots on the floor when the team is setting up the offense. At the time of crossing when the offensive player is close to the ball handler, all the conditions of the segmentor like the distances between the players would be positive for 13 frames resulting in a false positive
3. An action where in the flow of the offense, an offensive player and the ball handler run in opposite directions along the perimeter. In this case, a screen may or may not be set.

The segmentor merely filters out actions that may have screens. To accurately classify screens, we need to employ a different method.

Feature Extraction (Pre Processing)

To perform machine learning to classify actions, feature extraction must be done first to allow the machine to understand what exactly a screen is. This is done using pairwise distances relatively between the three players involved and their respective distances with the basket.

$$s = \underset{t}{\operatorname{argmin}}\ d_{t,p_b,p_s}$$

S is defined as the frame number at which the distance between the ball handler and the screener is at minimum. To calculate the s value, the distances between the ball handler and the screener must be noted during the entire duration of the action. Since the distances of all players relative to each other have already been calculated, storing these values in arrays are all that needs to be done.

self.handler_screener[self.j] = distances[self.screener][ball_handler]

The distances array is a 2D array which holds all of the relative distances. The ball handler and the screener are already identified during the segmentation as the offensive player closest to the ball and the offensive player closest to the ball handler respectively. Likewise, all the other pairwise distances between players are stored in arrays.

self.handler_defender[self.j] = distances[ball_handler][self.on_ball_defender]

self.screener_defender[self.j] = distances[self.screener][self.on_ball_defender]

The distances between the players and the basket must also be stored. This is done by calculating the distance of the players with respect to both baskets and taking the smaller one.

self.handler_basket[self.j] = min(euclidean_distances([[coords[ball_handler][0],coords[ball_handler][1]]],[[0,25]])[0][0],euclidean_distances([[coords[ball_handler][0],coords[ball_handler][1]]],[[94,25]])[0][0])

The same is done for the other two players.

Once all the distances are stored in arrays for the entire duration of the potential screen, the feature extraction can begin.

Feature Extraction

For each pairwise relation, 5 features are extracted which results in a total of 30 features per action. The metrics to be calculated are shown below:

1	2	3	4	5
$\min_t d_{t,a,b}$	$\frac{(d_{s,a,b} - d_{1,a,b})}{s}$	$\frac{1}{s}\sum_{t=1}^{s} d_{t,a,b}$	$\frac{(d_{n,a,b} - d_{s,a,b})}{(n-s)}$	$\frac{1}{(n-s)}\sum_{t=s}^{n} d_{t,a,b}$

N is the total number of frames which were tracked. S is the frame where the distance between the screener and the ball handler is minimum.

S is calculated by finding the minimum value of the screener-ball_handler distance array

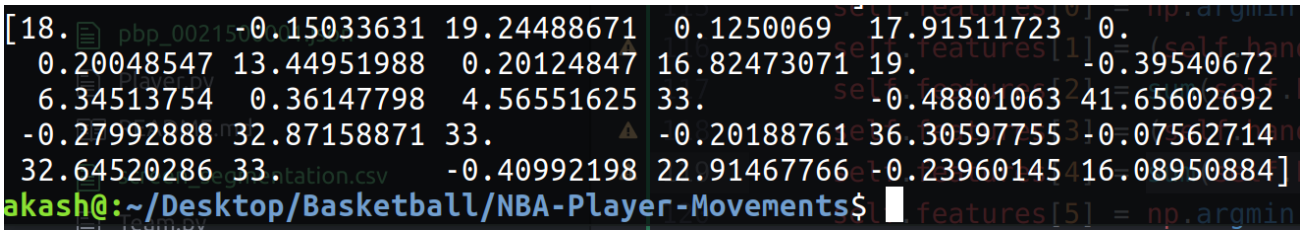
s = np.argmin(self.handler_screener)

The d notation with three subscripts indicate the frame at which distance should be calculated between the two players denoted by the last two letters.

Metric calculation was done in five steps:

1. The first metric is calculated by finding the minimum of the array of distances: *self.features[0] = np.argmin(self.handler_defender)*
2. The second metric is calculated by finding the difference of the distance at frame s and the initial frame and dividing it by s: *self.features[1] = (self.handler_defender[s] - self.handler_defender[0]) / s*
3. The third metric is calculated by finding the sum of the array from the start till the s frame and divide it by s: *self.features[2] = sum(self.handler_defender[:s+1])/s*
4. The fourth metric is calculated by dividing the difference between the final distance and the distance at frame s by the differnece between the total frames and s: *(self.handler_defender[-1] - self.handler_defender[s])/(len(self.handler_defender) - s)*
5. The fifth metric is calculated by dividing the sum of distances from the s frame onwards by the difference between the total number of frames and s: *sum(self.handler_defender[s:])/(len(self.handler_defender) - s)*

The final feature vector comprises of these 5 metrics calculated for all six pairwise interactions i.e. all three players amongst themselves and all three players with respect to be basket.



References

1. Recognizing on ball screens in the NBA - http://www.sloansportsconference.com/wp-content/uploads/2014/02/2014_SSAC_Recognizing-on-Ball-Screens.pdf
2. NBA Player Movements - https://github.com/linouk23/NBA-Player-Movements
3. Exploring NBA SportsVU Data - http://projects.rajivshah.com/sportvu/EDA_NBA_SportVu.html