EE 417 Computer Vision

Term Project

Simple Offside Detection

Utku Gürsoy / 26572

Semih Zaman / 28318

20.01.2023

Fall 2022/2023

Table of Contents

Importance of the Problem & Problem Definition

The offside detection is an important part of the sport of soccer (or football). The offside rule is designed to prevent teams from gaining an unfair advantage by positioning players in a way that allows them to receive passes that they would not be able to receive if they were in an onside position. If a player is offside, they are not allowed to participate in the play and the opposing team is awarded a free kick. Offside detection algorithms are used to help referees and assistant referees make accurate offside calls during a match [2]. These algorithms use computer vision and machine learning techniques to analyze live video feeds of the match and identify players who are in an offside position. This can be a challenging task, as it requires accurately tracking the positions of all players on the field in real-time and determining whether they are in an offside position based on the position of the ball [2]. Overall, the offside detection algorithm plays an important role in ensuring that the offside rule is applied fairly and consistently in soccer matches, and it helps to ensure that the game is played in a fair and sporting manner. Aim of this project is to implement a simple offside detection algorithm by using Computer Vision techniques covered in the scope of EE 417 course. Our algorithm detects whether a player is in offside position or not, given different in-game images from video recordings of soccer games as input.

Problem Formulation & Solution Method

Problem Formulation:

Let say that there are N defending players (P_{D1}, P_{D2}, …, P_{DN}) and M attacking players (P_{A1}, P_{A2}, …, P_{AM}) in a given video frame. Also, let define that the coordinates of the closest region of body (head, chest, shoulder or foot) to the attacking goal post for defending players are (x_{Di}, y_{Di}) for i=1, 2, …, N; for attacking players are (x_{Ai}, y_{Ai}) for i=1, 2, …, M.

The mathematically formulation of the problem basically is as follows:

$$G(\text{frame}) = \begin{cases} "Offside\ detected" & if any\ (x_{Ai}, y_{Ai})\ is\ closer\ than\ (x_{Dm}, y_{Dm}) \\ "Failed\ to\ Detect\ Offside" & otherwise \end{cases}$$

where (x_{Dm}, y_{Dm}) is the coordinates of the defending player who is the closest to the attacking goalpost where $1 \leq m \leq M$.

Solution Method:

In this problem, there are two main issues:

1) How to detect the players and differentiate them as defenders and attackers?
2) How to measure the closeness of the players regardless of perspective?

The first approach is color segmentation as a preprocessing step for player detection:

The implementation of this approach starts with an assumption that blue colored team is the attacking team while the red-colored team is the defending team, and the attack direction is from right to left. This situation can be easily generalizable in future projects. Therefore, the pixels are measured by how they are close to the blue and red, respectively, by using the fuzzycolor toolbox [1]. Based on their closeness to these colors, the algorithm segments the image after grey-scaling it. The pixels convergingto blue and red were whitened, while others were darkened.

Before entering the detection part, it is necessary to minimize the effect of noise as much as possible. In this regard, the ‘imfill’ function is used to fill in holes or gaps in objects, which is useful for object recognition since it allows to consider the entire object as a single entity rather than a collection of disconnected parts. The ‘bwareaopen’ function is used to remove noise or small, unwanted objects from the image. It is essential to consider only the larger, more meaningful objects in the image. By removing the small objects, the image is simplified, and the analysis can be focused on the more important features. Lastly, a morphological structuring element is created by ‘strel’ function in order to apply dilation operation. This aims for the same purpose as ‘imfill’ since it also allows to consider the entire object as

a single entity. The objects in the image are expanded, and any gaps or holes within the objects are filled, making it easier to analyze the entire object as a single entity by dilating the image. The explanation in this paragraph belongs to the part of the implementation in the lines between 90-140 as can be seen in Appendix B-C.

After several preprocessing steps are done for each blue and red segmentation, the remaining regions are the regions where the players we are looking for are located. Thus, the last process is to detect the remaining regions with the bounding box. This process nearly solves the first main issue except when the tribune is in the photo because the tribune causes the detection part since it contains blue and red parts.

To tackle this little problem, it is enough to detect the line determining the football field because the outside of the field is irrelevant. The Hough Transform is preferred to apply for this detection problem [3]. Since we decided that the position of the camera angle, which takes the images we applied in our project, is on the long side of the football field, the longest horizontal line detection tackles our problem. Then, it is possible to discard the region above the line, which corresponds to the tribune. Also, a threshold is determined to pick the longest horizontal line in order to stop discarding the pictures without tribune [3]. The Hough Transform is utilized in the lines between 58-87 as can be seen in Appendix A-B.

The second approach is the vanishing point for determining the position of the player relative to themselves:

Our offside detection problem is based on the comparing the distances of the defensive players and attacking players from the goalpost. However, it is not as easy as comparing the x coordinates and choosing which is displayed further to the left or the right due to the perspective in the image. To make this comparison possible, we will get help from the vanishing point idea since the straight lines in the football field are parallel and converge at a single image point, called a vanishing point [3].

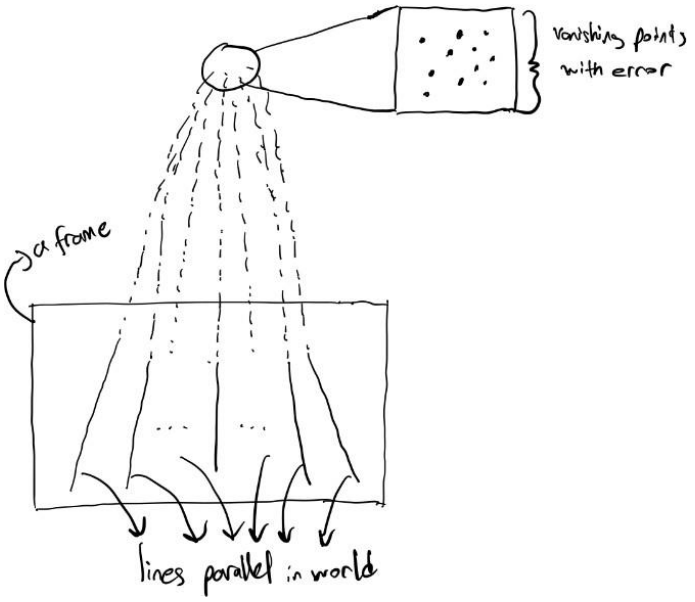


Figure 1: This is the illustration of the vanishing point approach in the algorithm.

Different from the theoretical explanation stating that lines converge at a single point, lines will intersect in pairs, as in Figure 1, since determining all lines perfectly is impractical. This implies that there needs a reconstruction of the vanishing point, which is averaging all possible pairwise intersections in our case. The implementation of this approach in the lines between 21-52 as can be seen in Appendix A.

In the next step, we need to draw an offside line in order to decide whether the position is offside or not. There comes another assumption for simplicity: the offside lines are always drawn according to a fixed point on the bounding boxes of defending players, and offside is also detected according to the same fixed bound bounding boxes of attacking players, as shown in Figure 2.

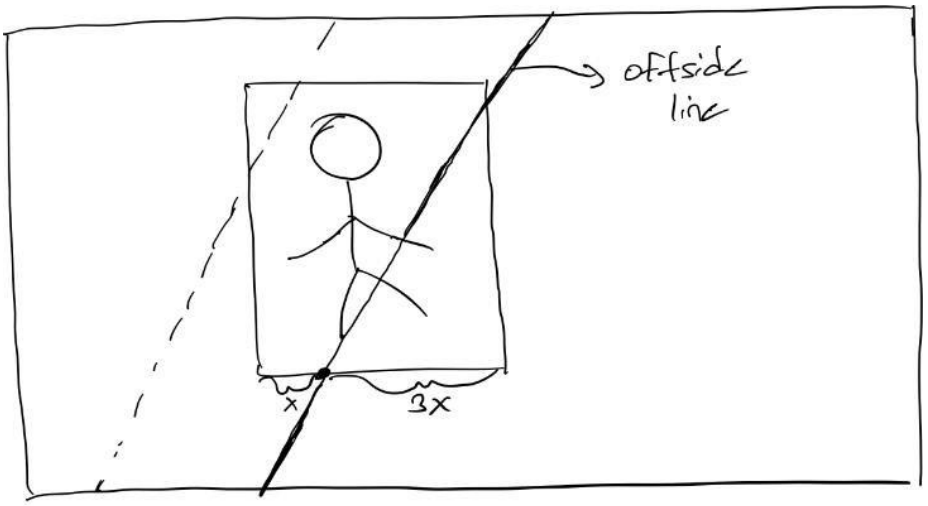


Figure 2: It visualizes the assumption

For drawing the offside line, it is necessary to determine the closest defending player to the attacking goalpost, which is the left-most player in our case [2]. Thanks to the reconstructed vanishing point, it is possible to get a projection of all defending players' positions into a line to understand whom the left-most player is with avoiding the projection illusion. As mathematically speaking, let us say that there are K defending players; therefore, K projections will be made. The points corresponding to each defending player are (x_1, y_1), (x_2, y_2), …, and (x_K, y_K) using the above assumption in Figure 2. For each point, a line expressing mx + c = y will be drawn where the vanishing point is (x_{VP}, y_{VP}), $m = \frac{y_{VP}-y_i}{x_{VP}-x_i}$, and $c = y_i - mx_i$. Then the answer becomes the $\underset{i}{argmax} \frac{s-c}{m}$ while projecting on the line y = s for i=1,2, …, K. This logic improved by us is implemented in the lines between 151-175 as can be seenin Appendix C.

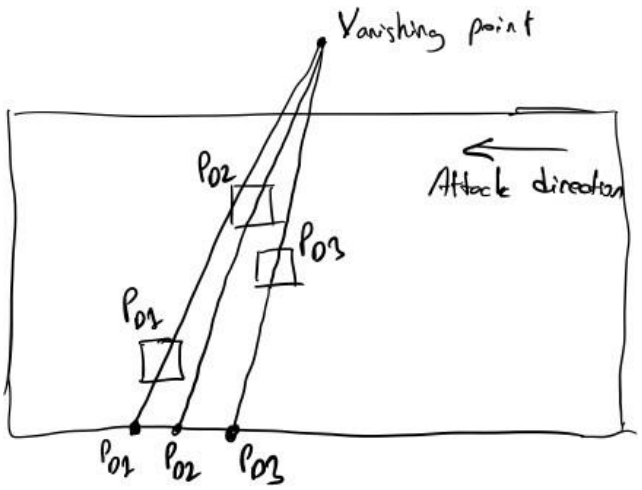


Figure 3: It is an illustration of determining the defending player to draw the offside line, which results in choosing Player 1

After determining the desired player, the offside line was calculated automatically as equals to $mx + c = y$ where $m = \frac{y_{VP} - y_k}{x_{VP} - x_k}$, and $c = y_k - mx_k$ if k = $arg\max_i \frac{s-c}{m}$. Now, everything is ready to reach the ultimate goal, offside detection. We are going to implement the if condition mentioned in the 'Problem Formulation', which checks whether any attacking player is on the left of the offside line. The implementation is mathematically simple because it is enough to check that there exists any attacking player whose coordinates satisfy the equation mx + c $>$ y for deciding whether 'Offside detected' or not, as shown in Figure 4. This part is implemented in the lines between 178-198 as can be seen in Appendix C.

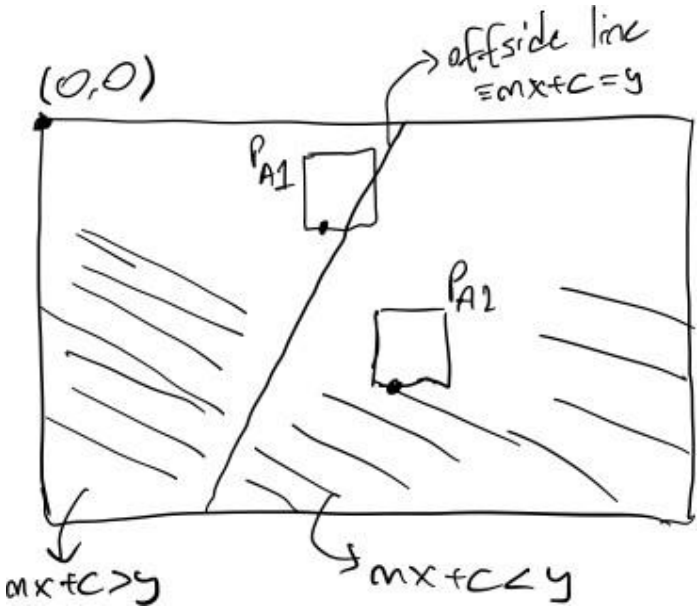


Figure 4: The geometric interpretation of the offside detection

Mathematical Background of Line Detection with Hough Lines and Hough Transform

All points in cartesian coordinates can be defined the tuples of (x_i, y_i) where x_i is the projection of it on x-axis and y_i is the projection f it on y-axis. Moreover, all lines in cartesian coordinates can be expressed as $y = mx+b$ where m and b are the parameters we're looking for. To do further analysis of the parameters, a mapping operation performs from the spatial domain to parameter domain. While x and y are variables in spatial domain, it turns out that m and b are new variables where x and y are parameters. After this stage, it is more convenient to make an interpretation regarding the parameters.

For instance, this operation maps a point in spatial domain to a line in parameter domain. Firstly, the equation transforms into $y - mx = b$ as changing the domain. Since a point is defined as only one tuple of (x_i, y_i). In parameter space, there will be one equation originating from this point, which corresponds to a line equation $y_i - mx_i = b$.

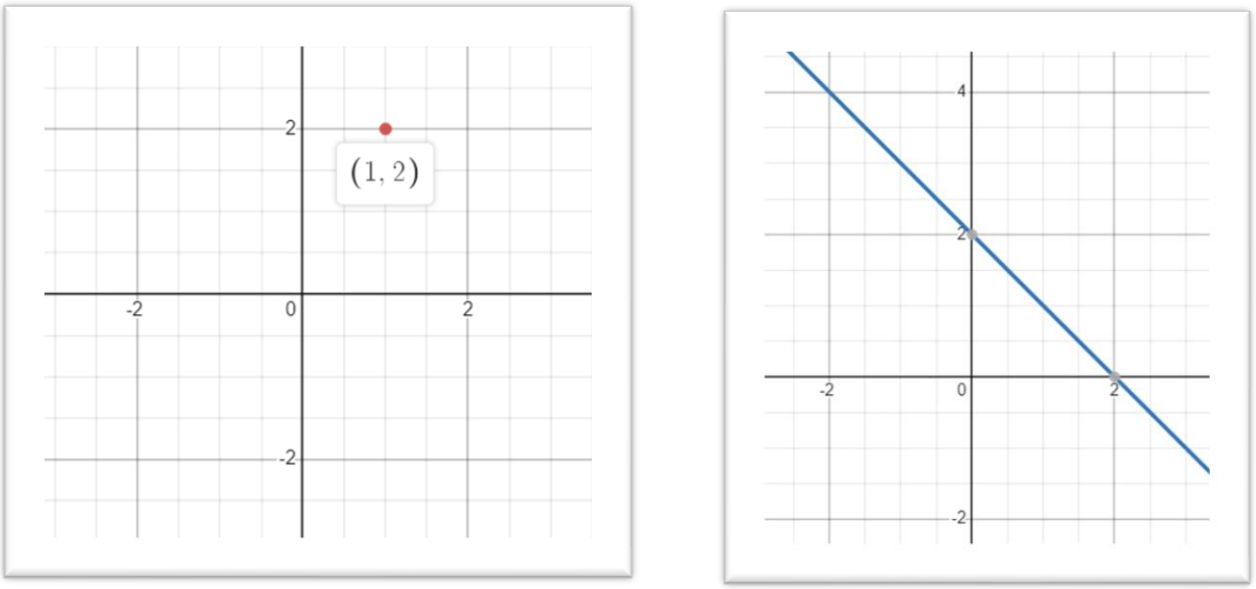


Figure 5: (1,2) is mapping onto b = 1-2m [3]

Also, this operation maps a line in spatial domain to a point in parameter domain. It is because a line contains infinitely many tuples of (x_i, y_i) which originates infinitely many equations, which there exists one common solution for all of it, in parameter domain. This solution is the location where this line will be mapping in parameter domain.

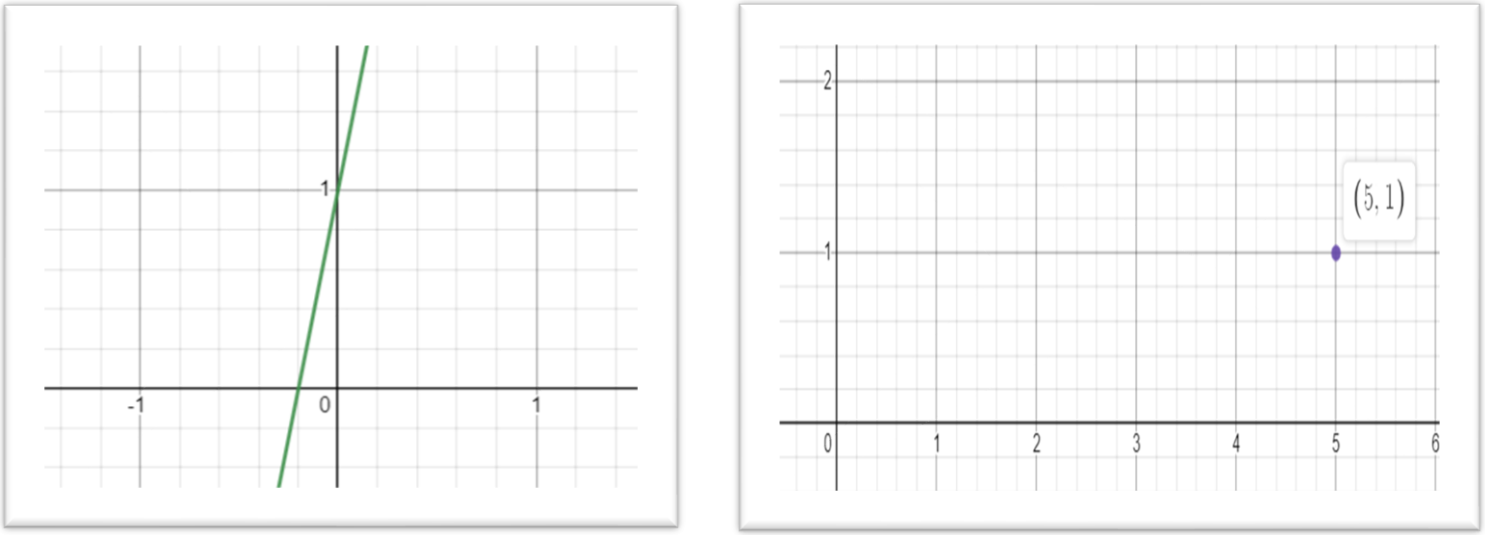


Figure 6: 5x+1=y is mapping onto the point of (5,1) [3]

One more observation will be the case of multiple points in spatial domains. This situation is more realistic in terms of line detection problem than the others. With the help of the previous observations, it is obvious that each point transforms a line in parameter domain. Except the points whose x_i's are the same, lines originated by points will be intercepting pairwise at a point on a space. At the end, an image consisting of lots of intersection points will be appearing. The intersection point,which is crossed by the largest number of lines, will be a line spanning the greatest number of points in the spatial domain as we are searching for. As an example,

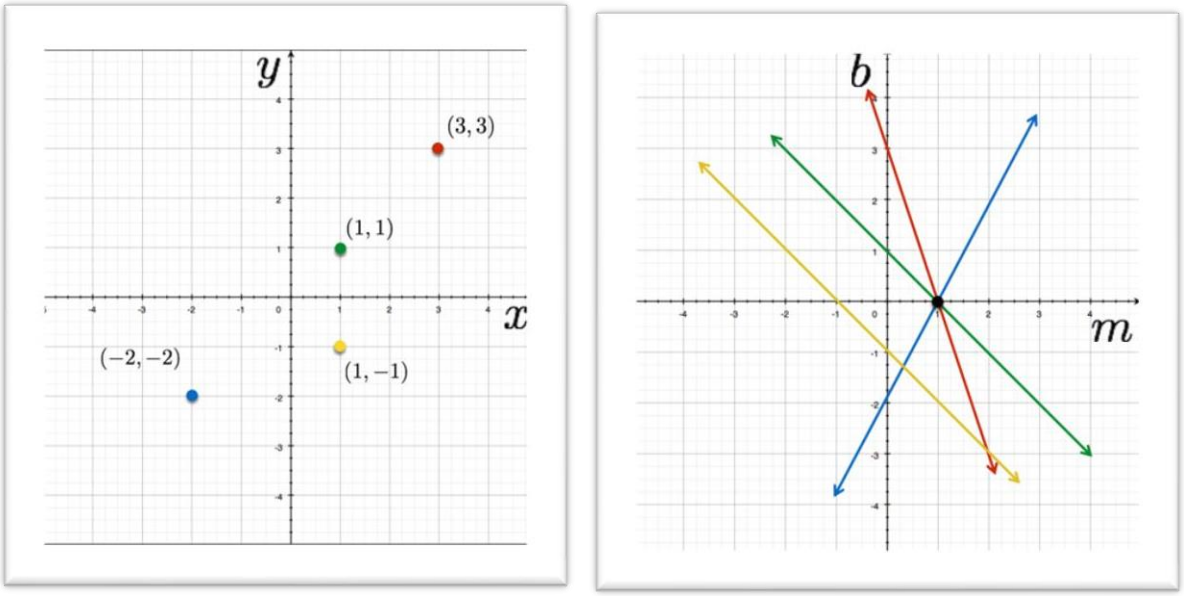


Figure 7: (1,0) is crossed by in total 3 lines in right image, and the line corresponds of that point is the most likely edge while considering given four points [3]

Therefore, the following steps belong to the algorithm which is a line detection algorithm with Hough Transform [3]:

1) Quantizing Parameter Space (m,c)

2) Create Accumulator Array A(m,c)

3) Setting the array A to 0 for all elements

4) Incrementing the entries of A whenever the point of (m,c) is crossed by a line

5) Finding local maxima in A(m,c)

However, this algorithm has its own problem. One of them is that it still problematic when the case of x = a constant. And also, quantizing parameter space is not applicable since the limits of parameters m and c are infinity. For the above problems, we define a new mapping operation as follows:

Firstly, the equation of mx+b= y changes into a new form of an equation by using trigonometric manipulation: $x\cos\theta + y\sin\theta = \rho$ where ρ is the shortest distance between the origin and the line, and θ is the angle of between the positive side of x-axis and this shortest distance. In this stage, $x\cos\theta + y\sin\theta = \rho$ is the equation of parameter space while y = mx+b is the equation of image space. Also, the new axis of parameter space is ρ and θ instead of m and b. The main difference of new mapping operation is that the points in spatial domain map onto a sinusoid in parameter domain.

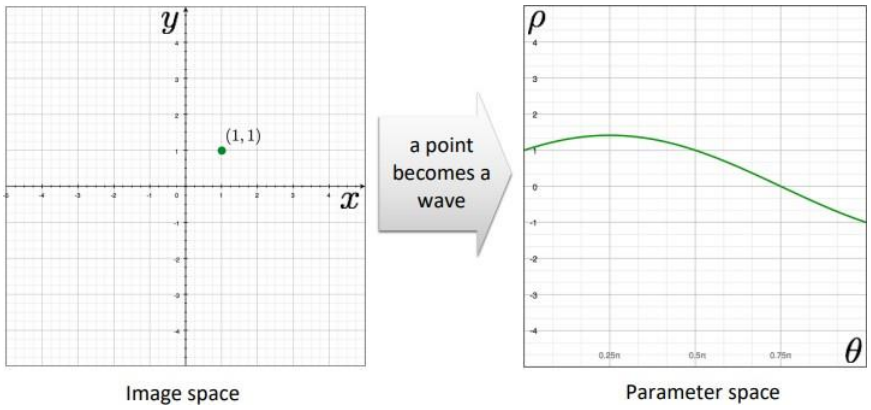


Figure 8: It is an illustration for the case of mapping of one point [3]

Also, a line in spatial domain mapping onto a point in parameter space as in the previous parameterization. Moreover, there occurs again an image consisting of lots of intersection points caused by sinusoids in the case of multiple points in spatial domains. The intersection point,which is crossed by the largest number of sinusioids, will be a line spanning the greatest number of points in the spatial domain as we are searching for. As an example,

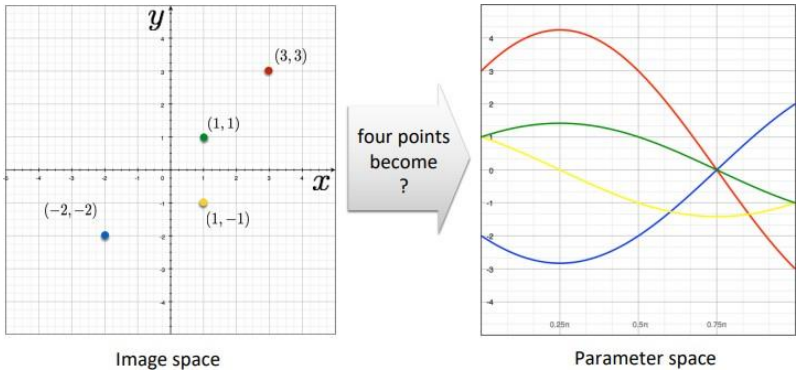


Figure 9: It is an illustration for the case of mapping of multiple points [3]

Consequently, the previous algorithm also changes as the axis of parameter space changes as follows [3]:

1)Quantizing Parameter Space (θ, ρ)

2)Create Accumulator Array A(θ, ρ)

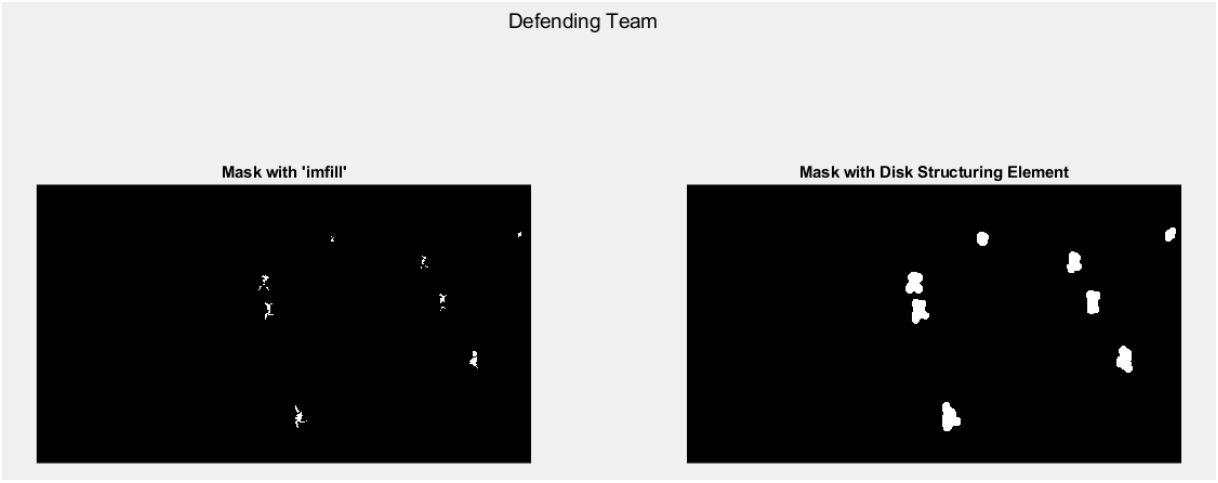
3)Setting the array A to 0 for all elements

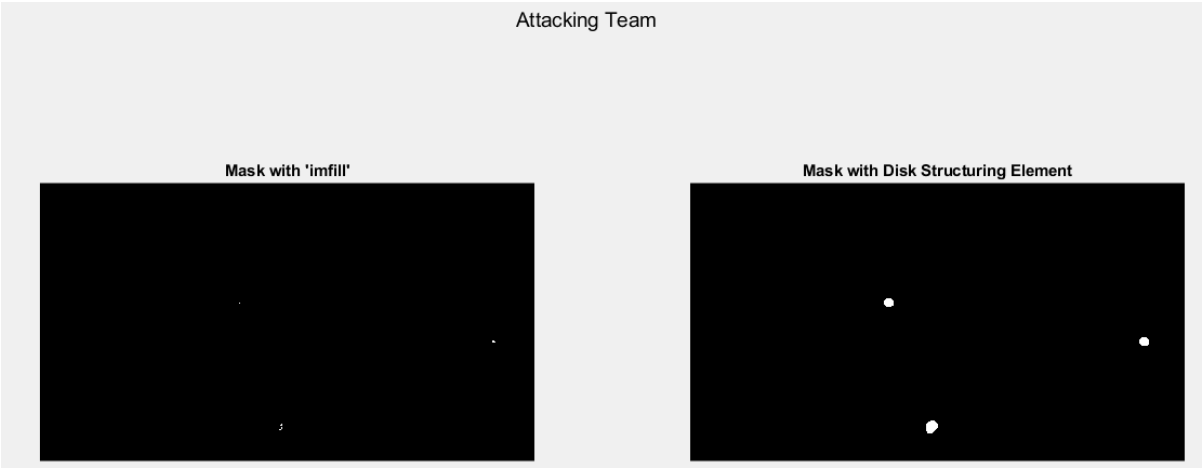
4)Incrementing the entries of A whenever the point of (θ, ρ) is crossed by a sinusoid

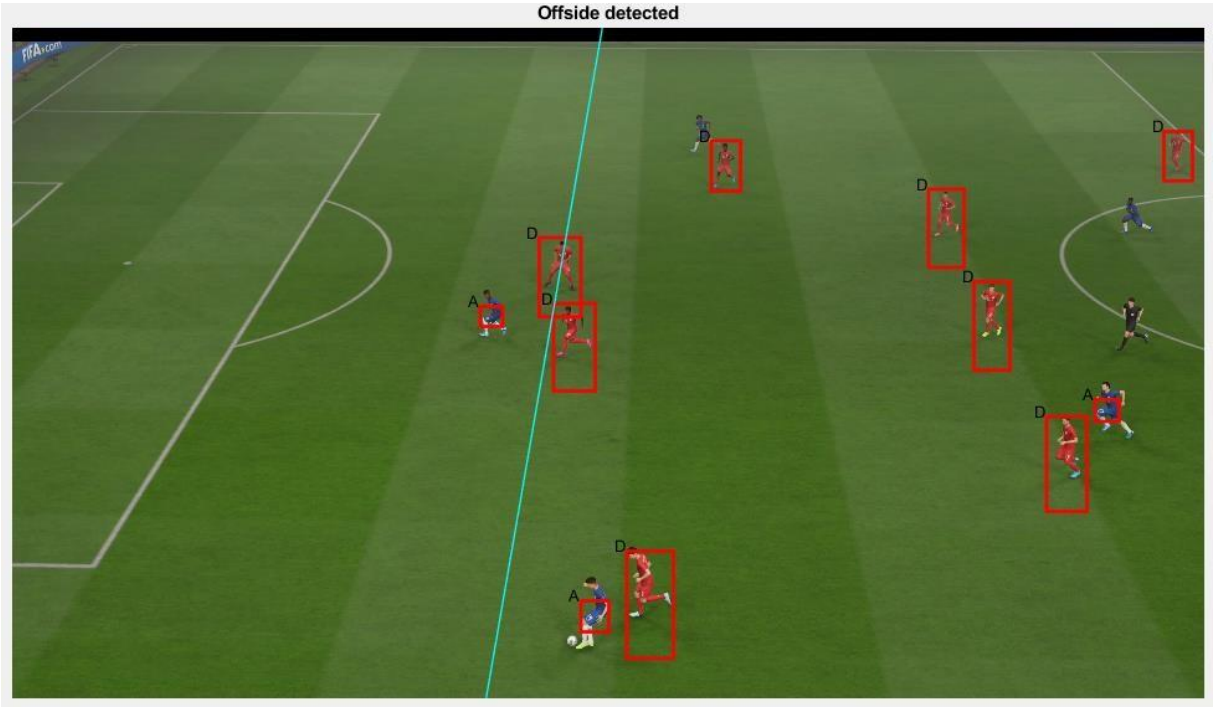
5)Finding local maxima in A(θ, ρ)

This updates on the algorithm solves the issues that the first one is struggle with since the θ can be valued between 0 and 2π, and ρ can be valued between 0 and a distance with a finite value, and the line with the equation of x = a constant corresponds to a sinusoid thanks to new parameterization.

Results





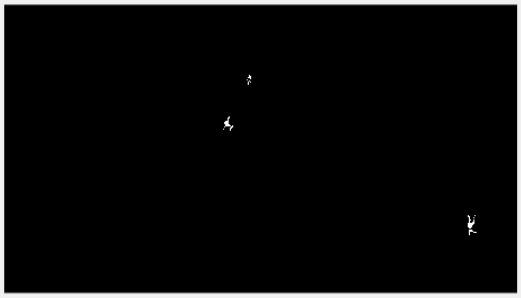


(Figure 1: Test 1 Correct)

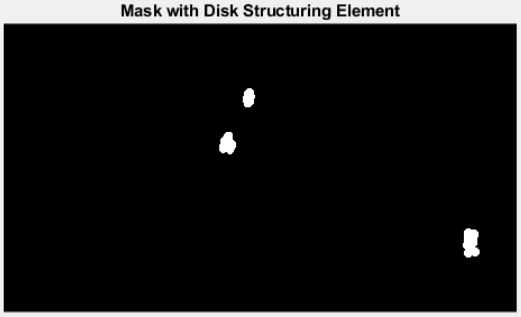
Please select two points on field lines to calculate vanishing point

Defending Team

Mask with 'imfill'

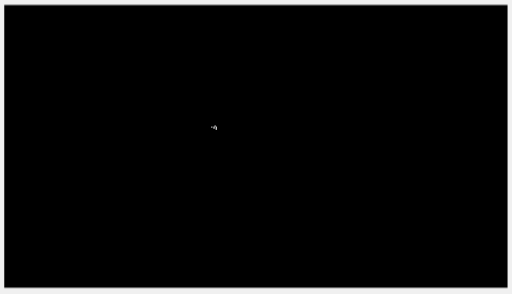


Mask with Disk Structuring Element

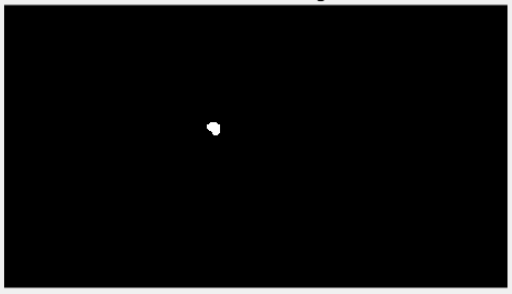


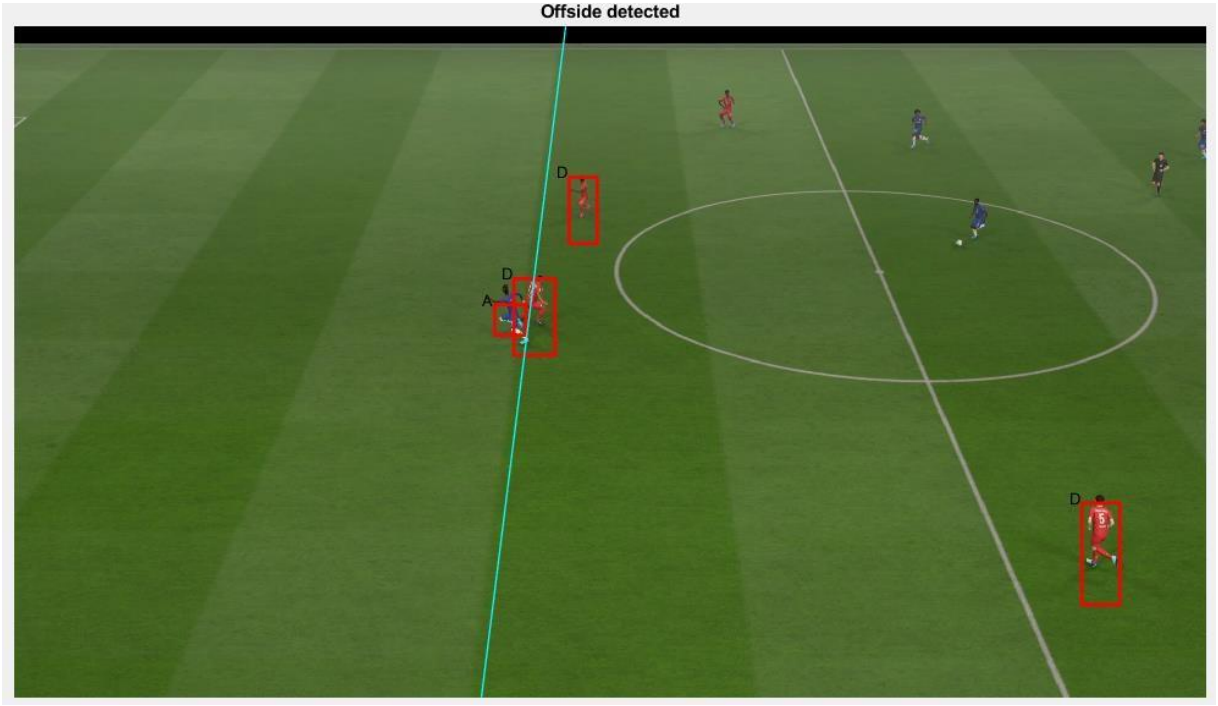
Attacking Team

Mask with 'imfill'

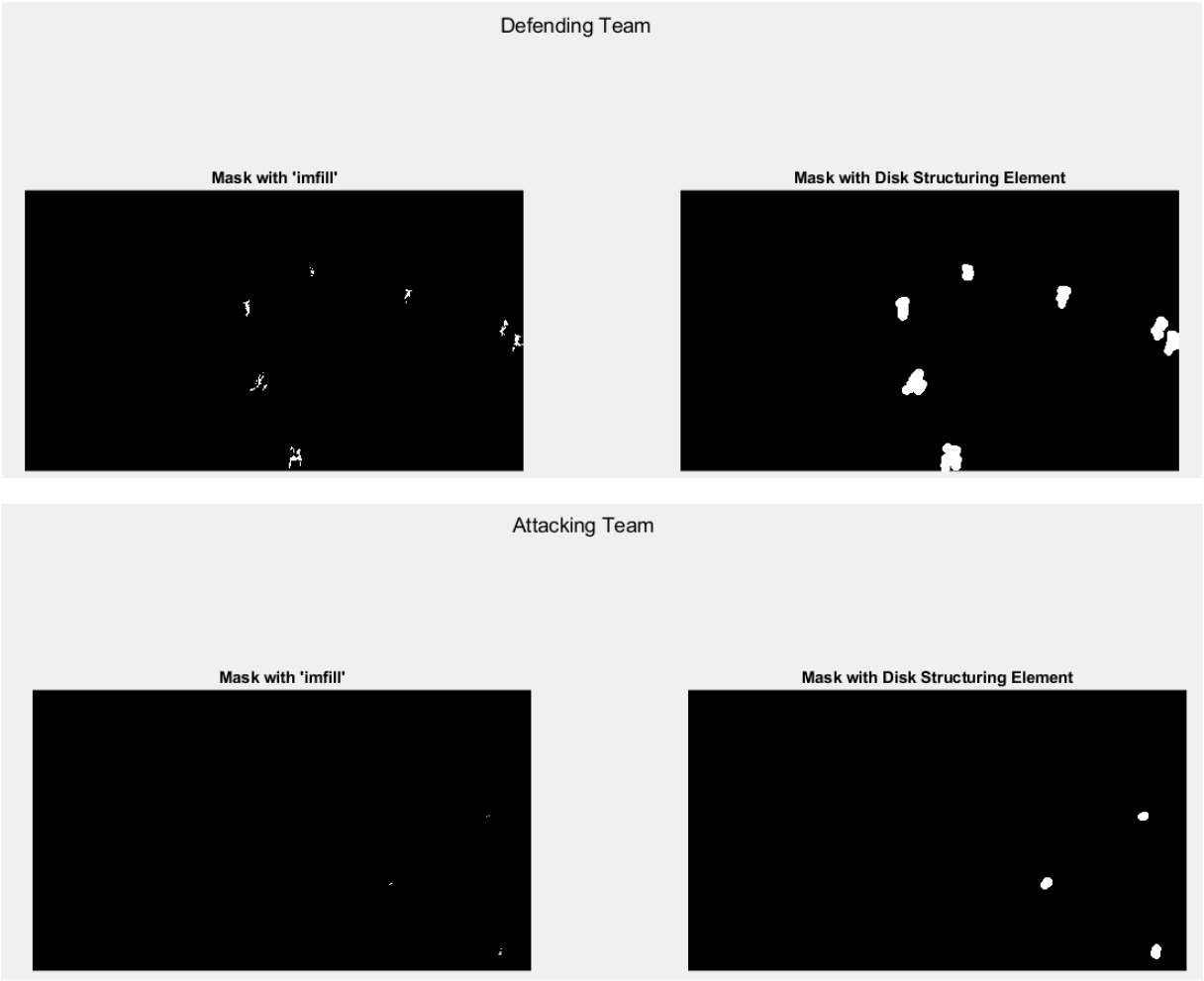


Mask with Disk Structuring Element



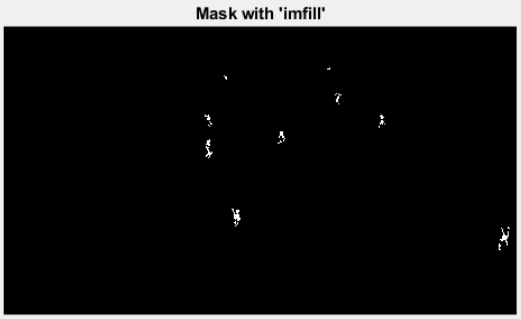


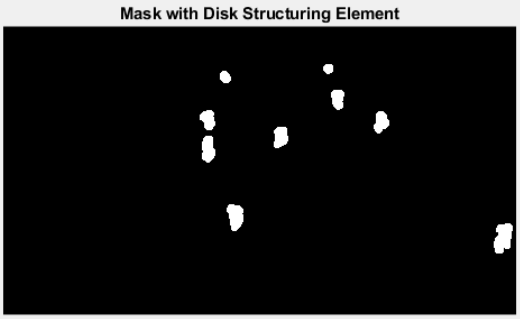
(Figure 2: Test 2 Correct)



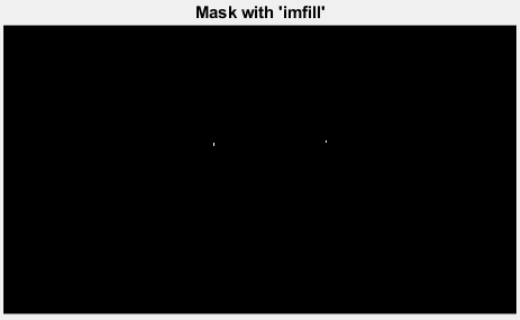
(Figure 3: Test 3 Fail)

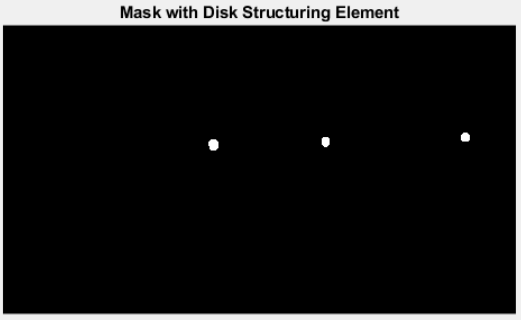
Defending Team

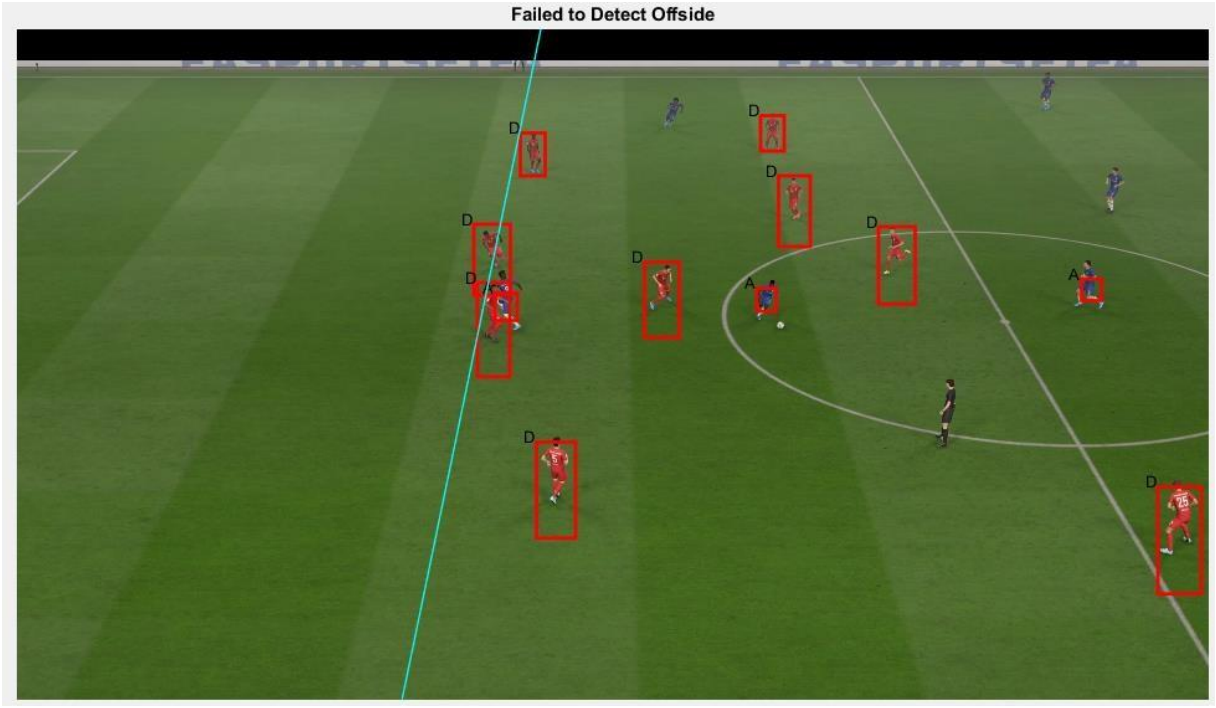




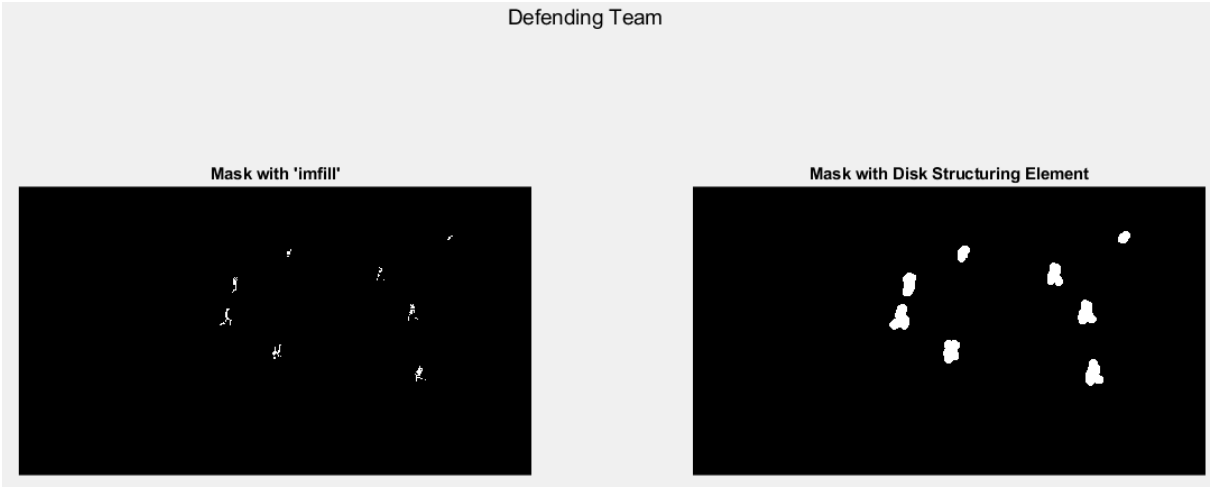
Attacking Team

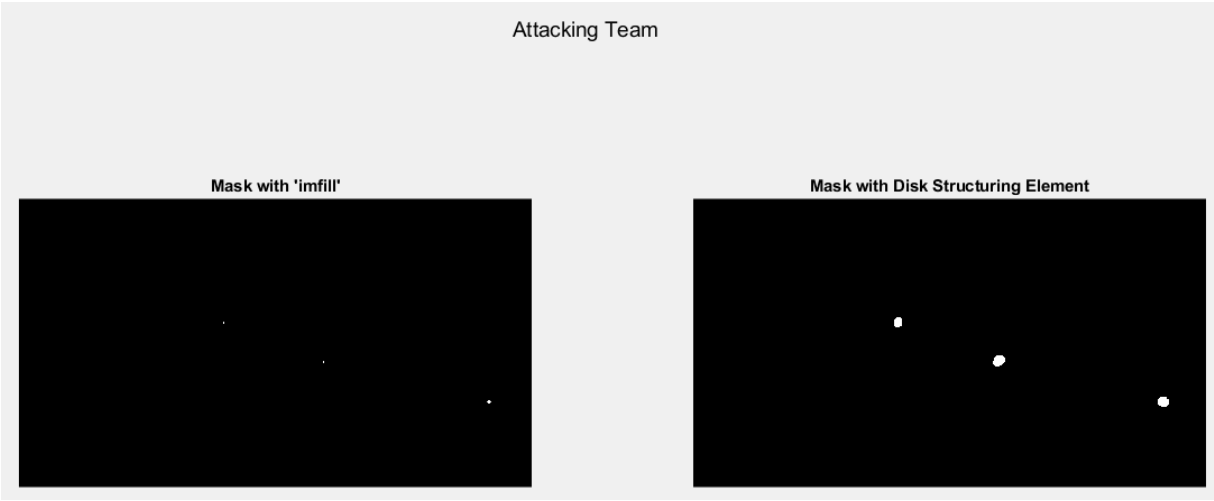


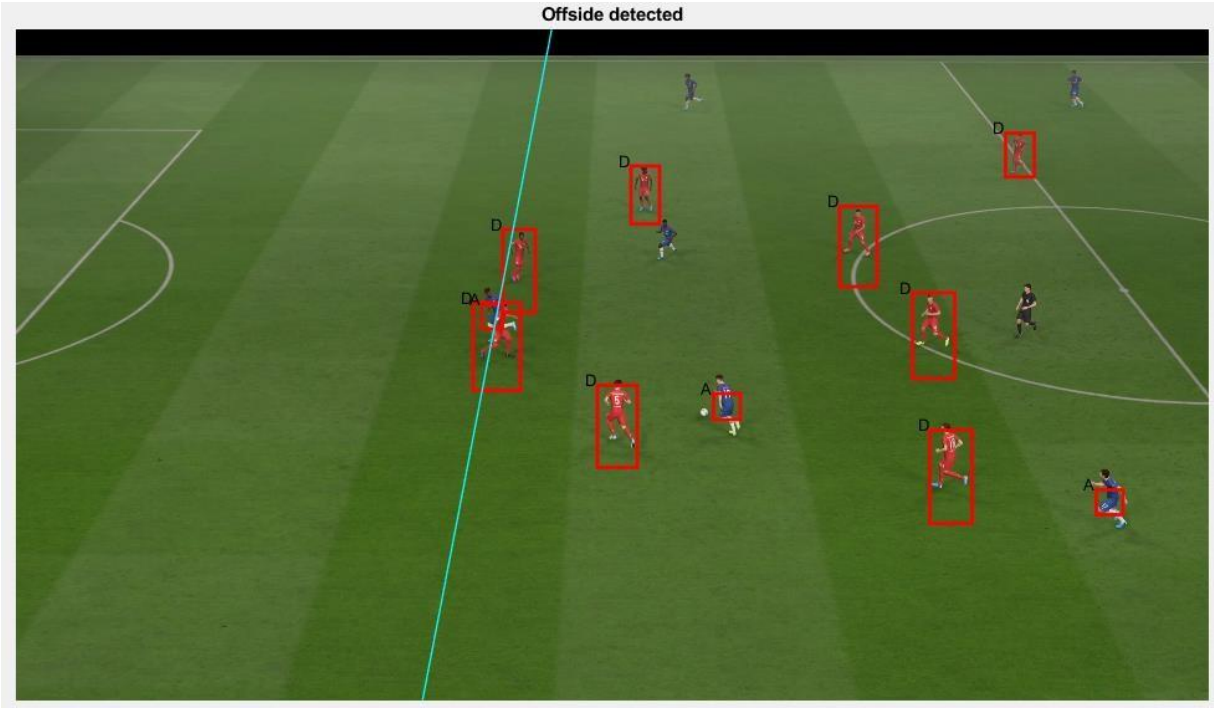




(Figure 4: Test 4 Correct)

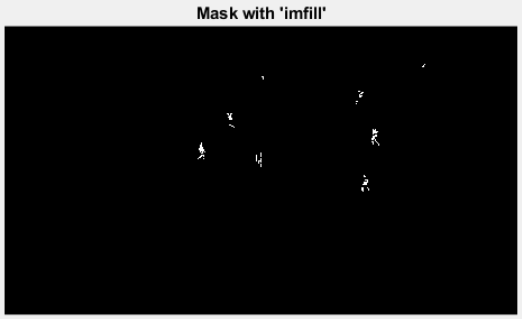


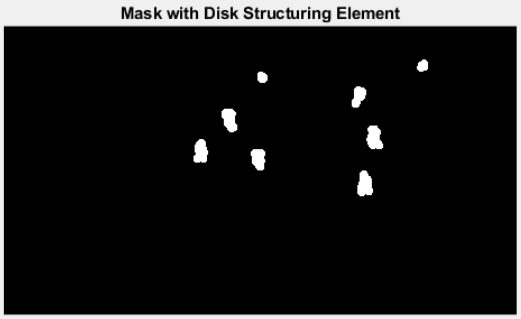




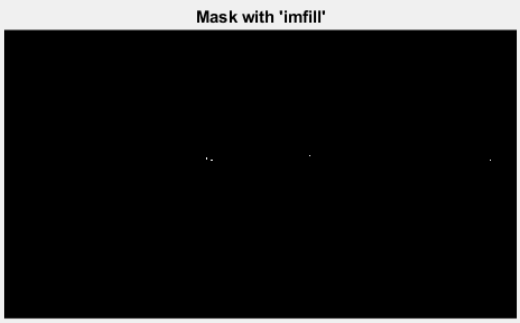
(Figure 5: Test 5 Correct)

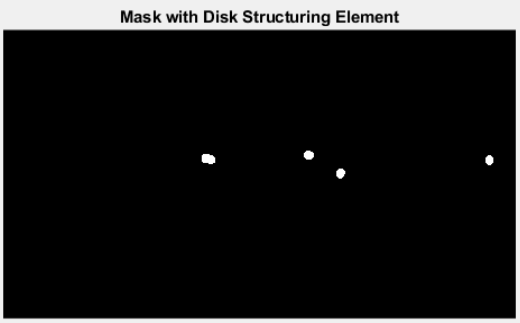
Defending Team

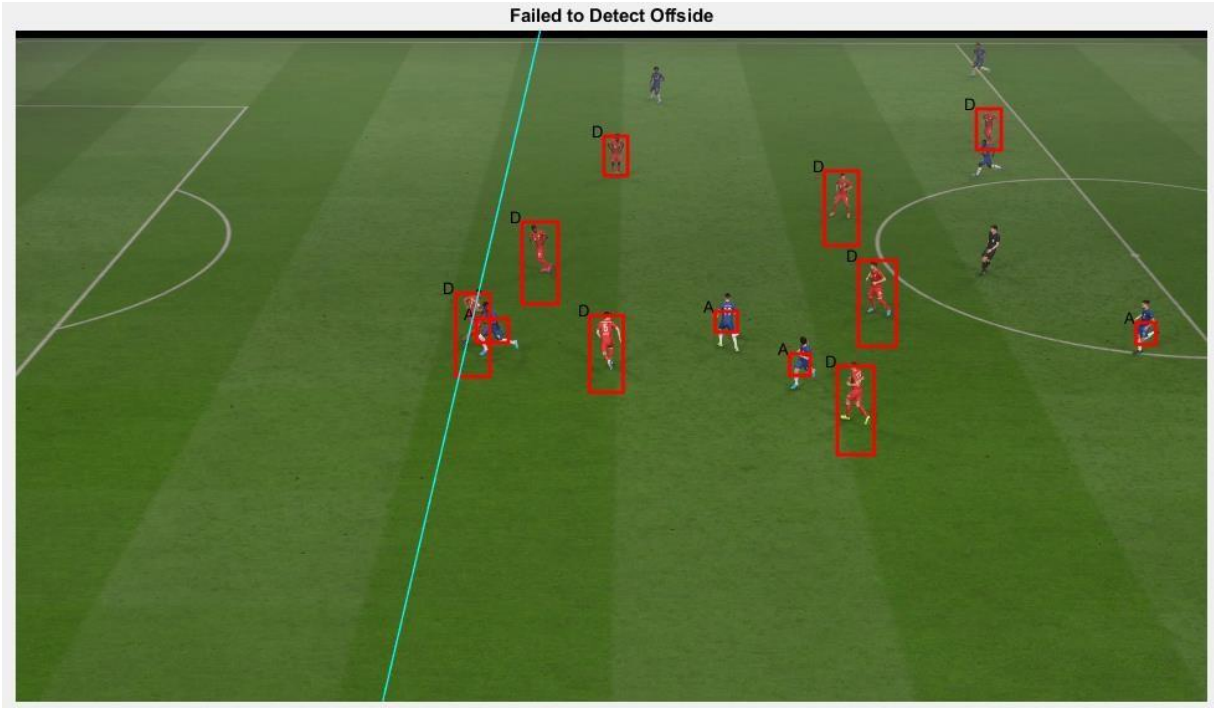




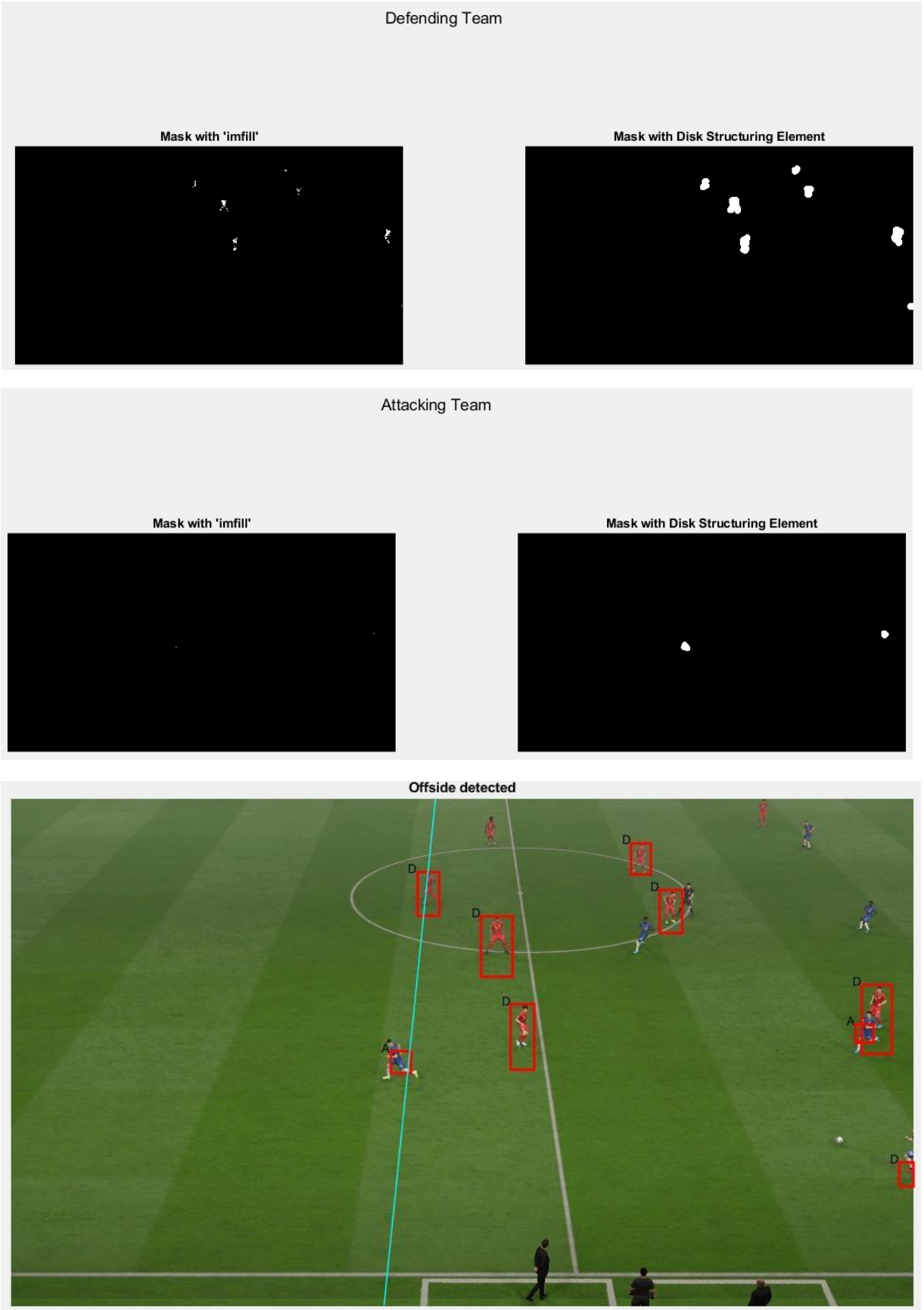
Attacking Team



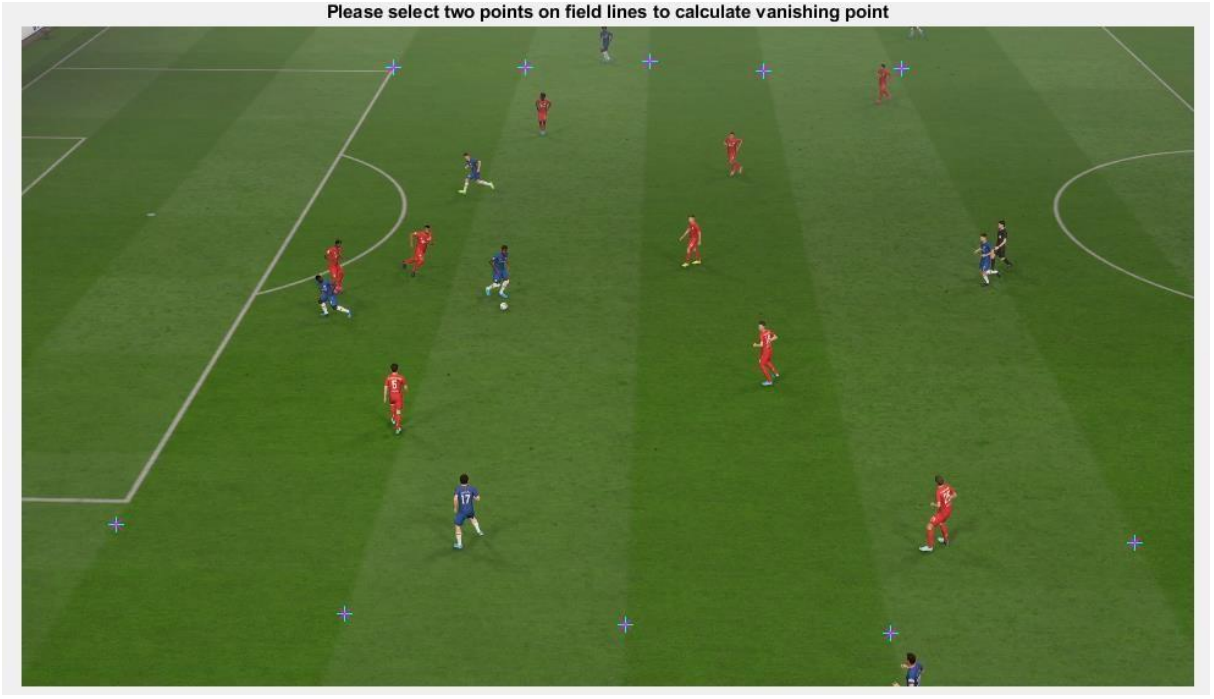




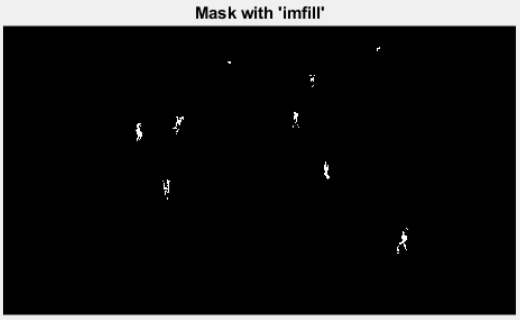
(Figure 6: Test 6 Correct)

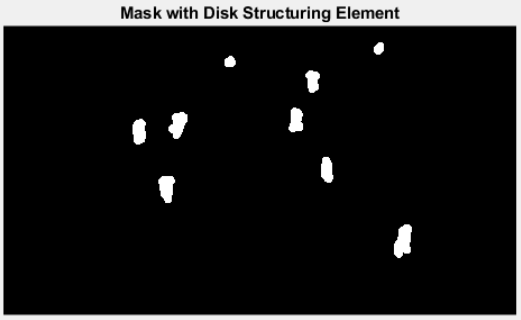


(Figure 7: Test 7 Correct)

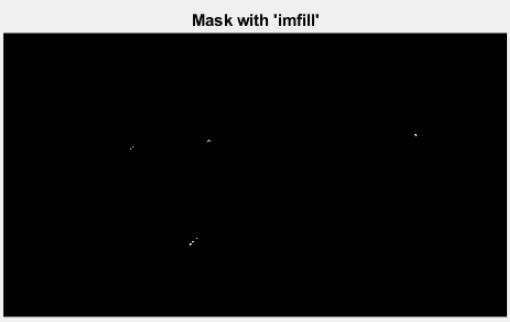


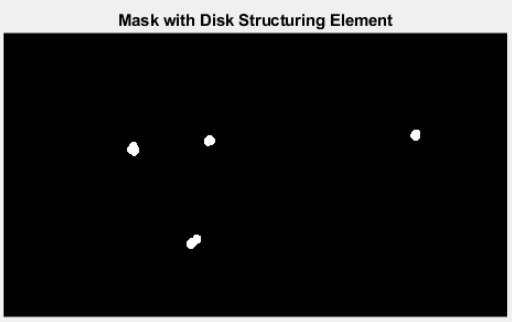
Defending Team

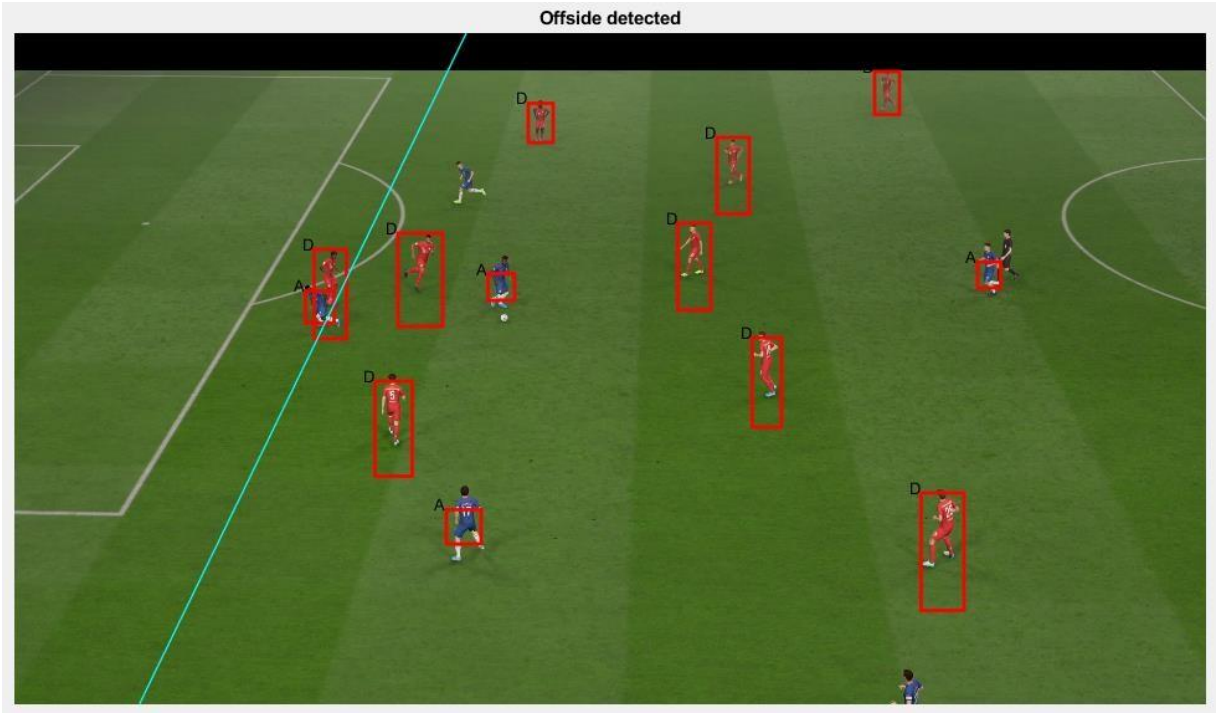




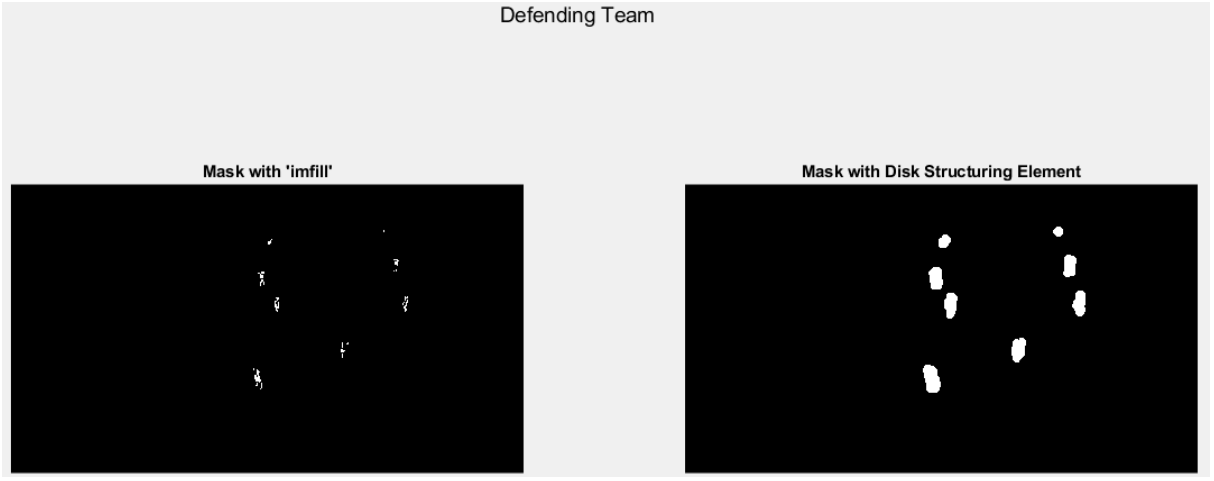
Attacking Team

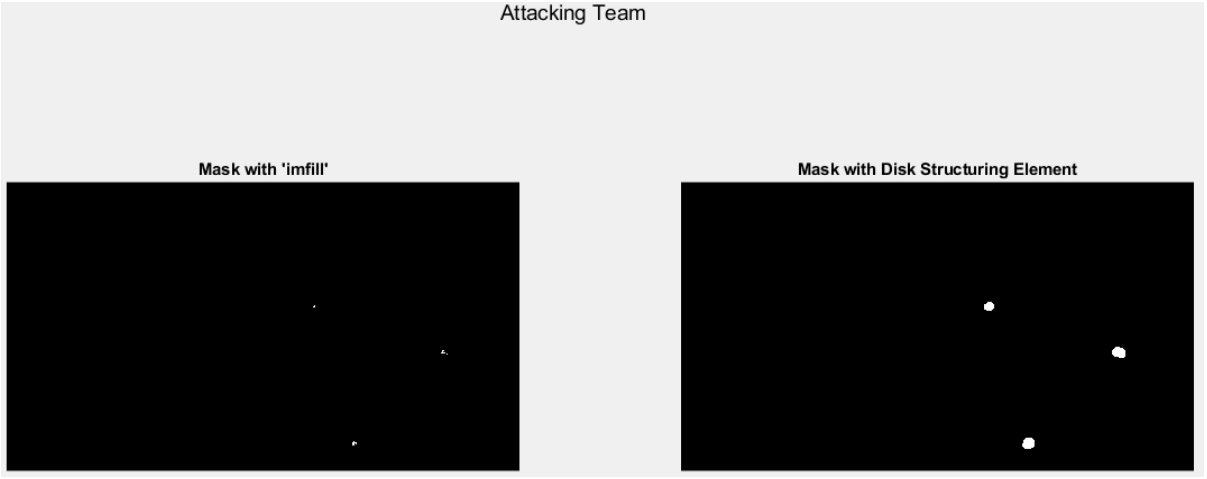


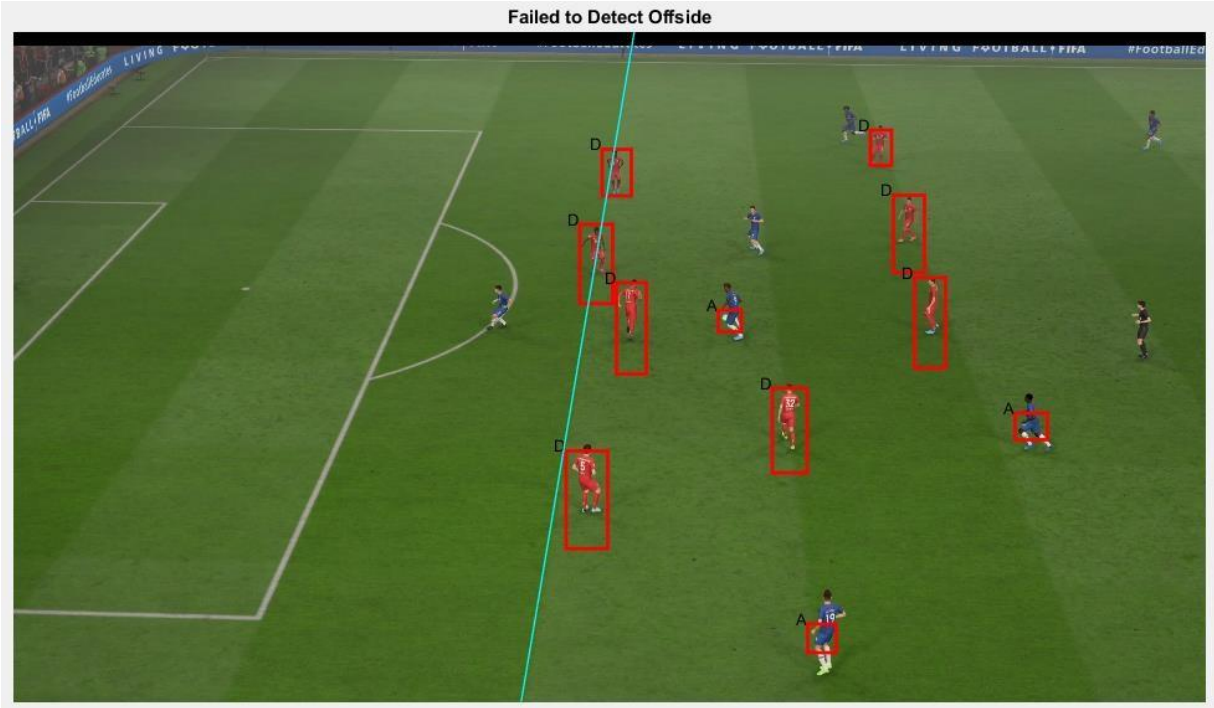




(Figure 8: Test 8 Correct)

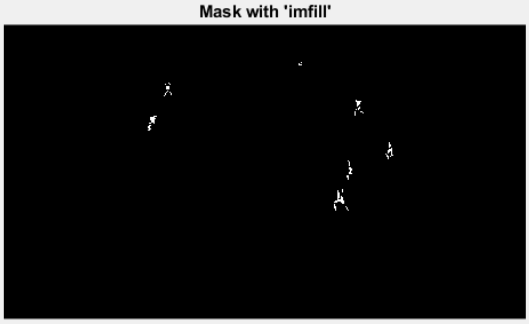


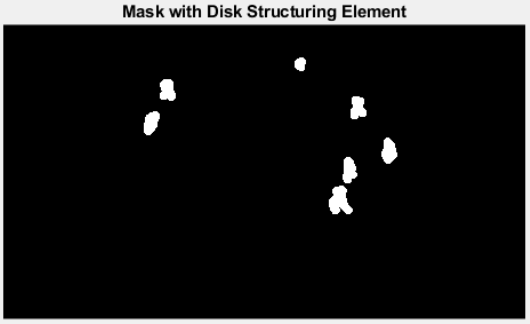




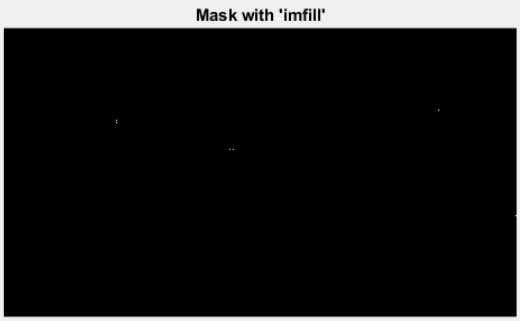
(Figure 9: Test 9 Failed)

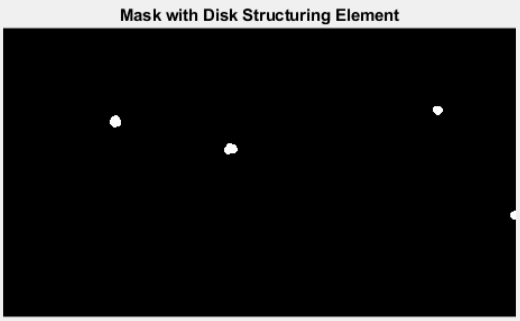
Defending Team

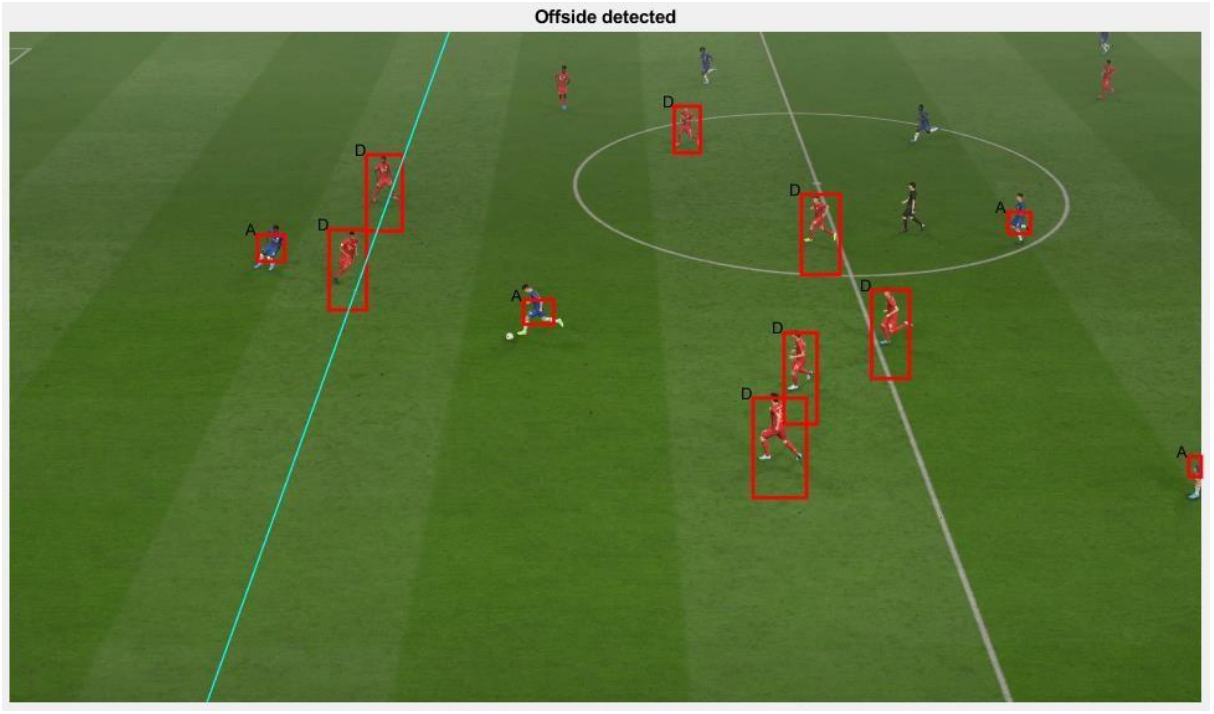




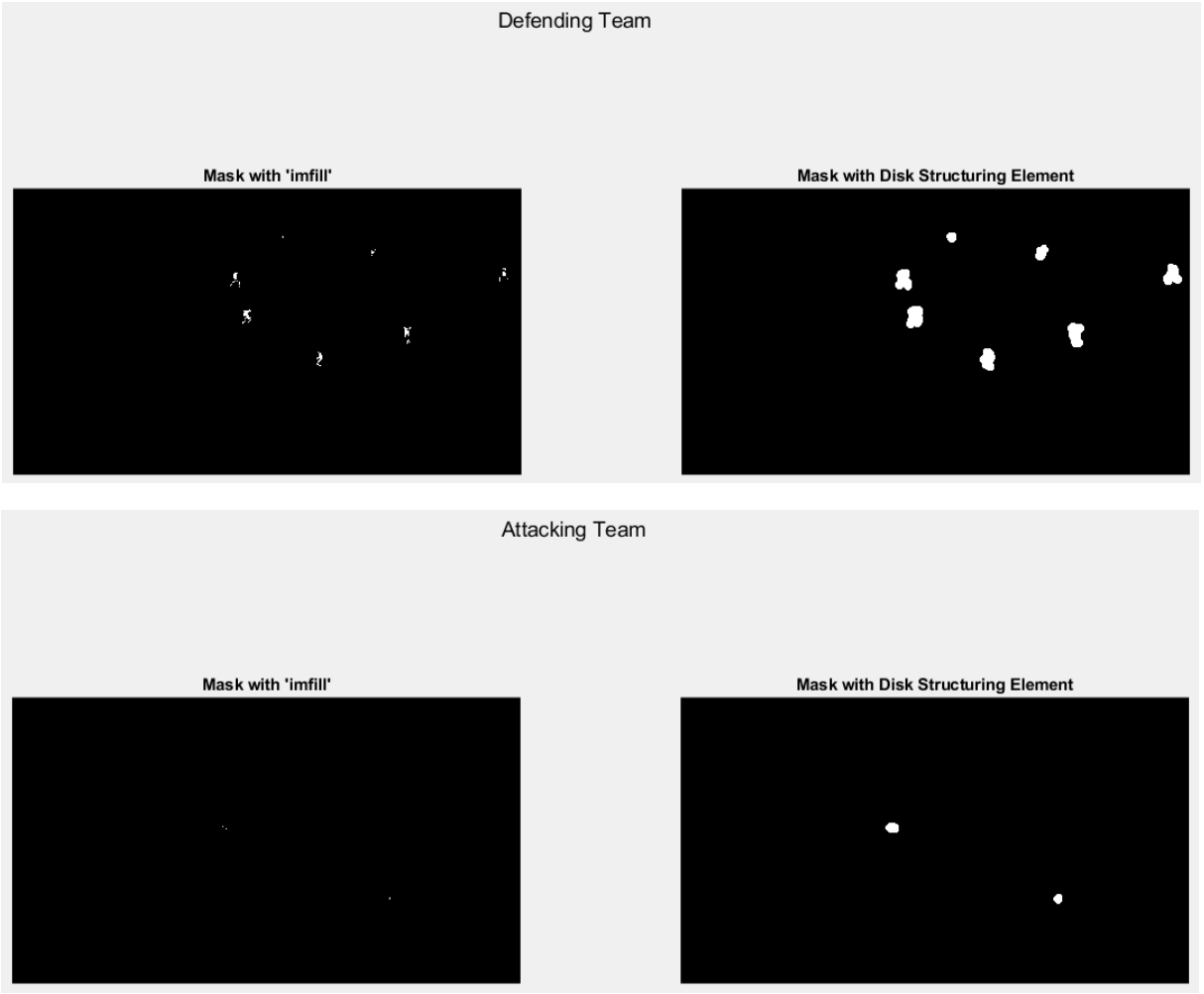
Attacking Team





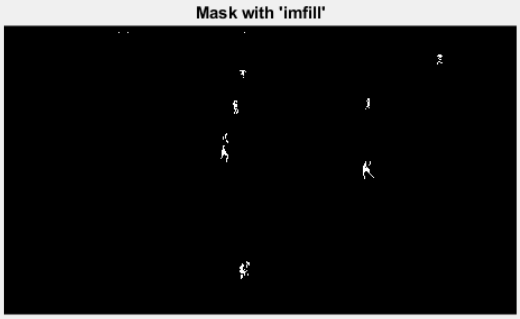


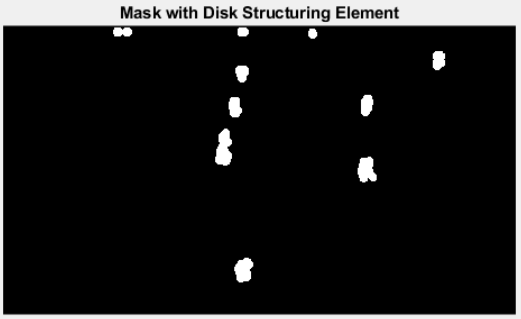
(Figure 10: Test 10 Correct)



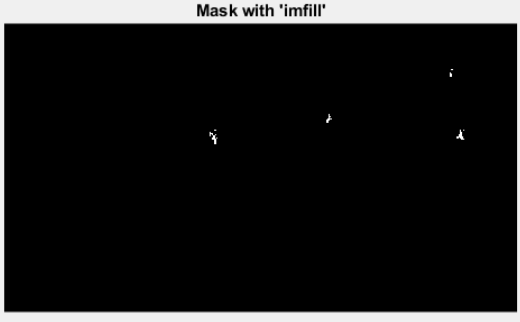
(Figure 11: Test 11 Correct)

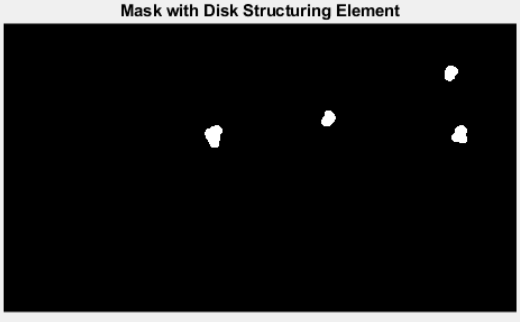
Defending Team

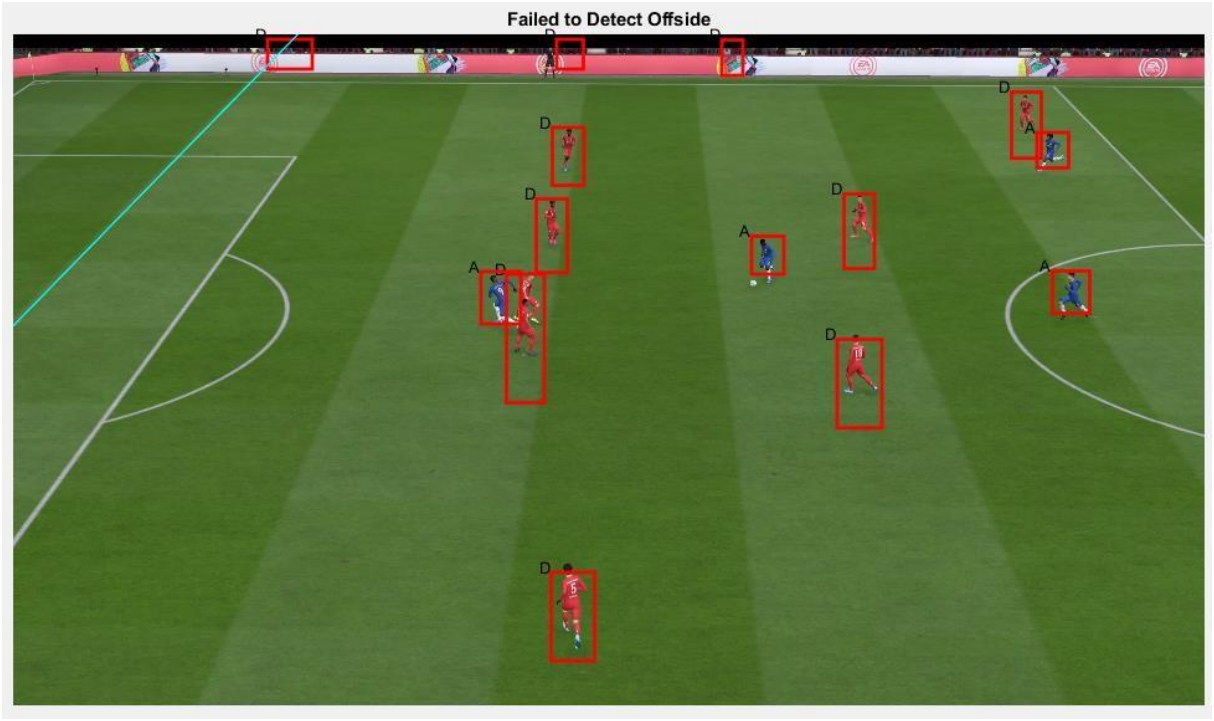




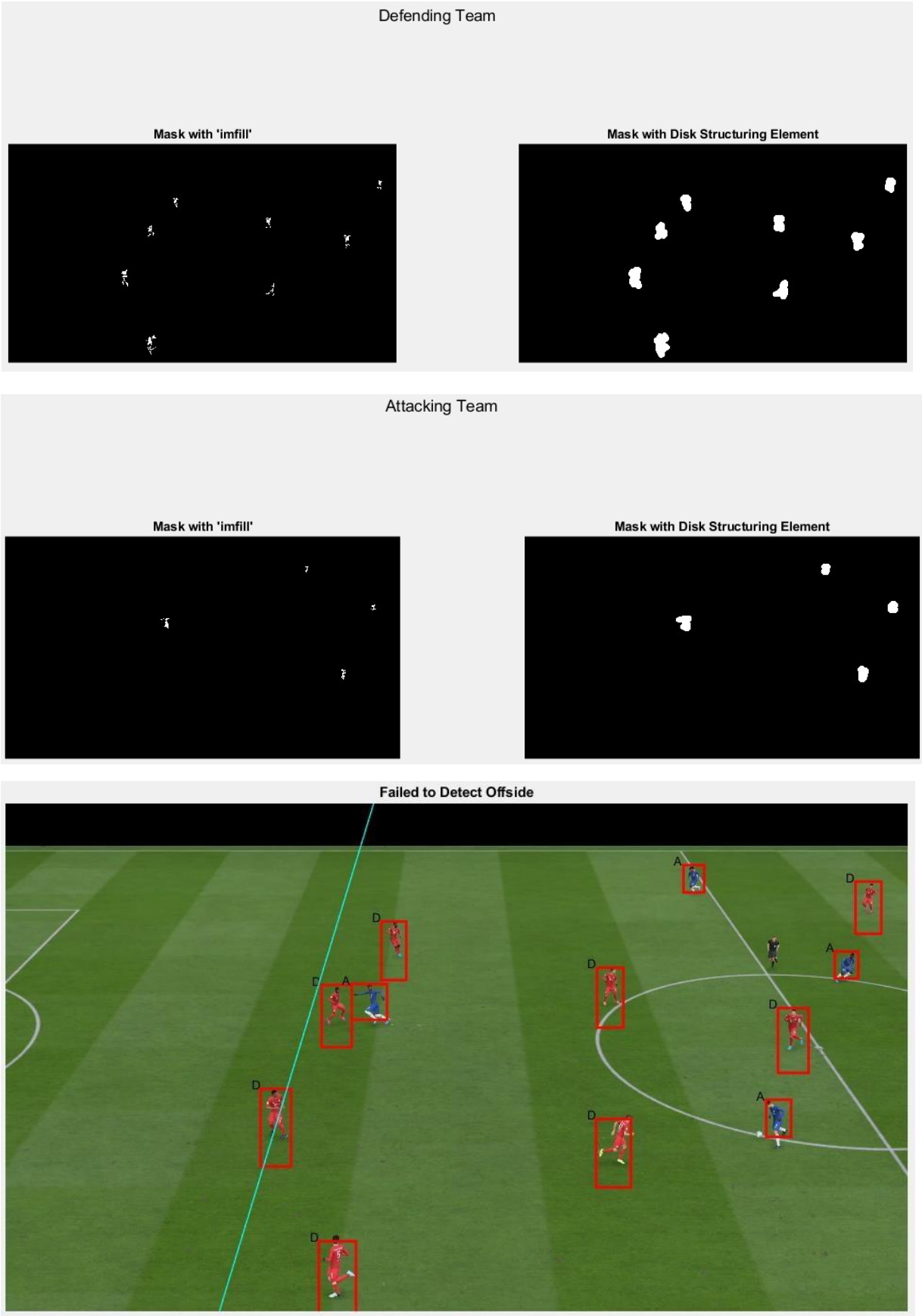
Attacking Team



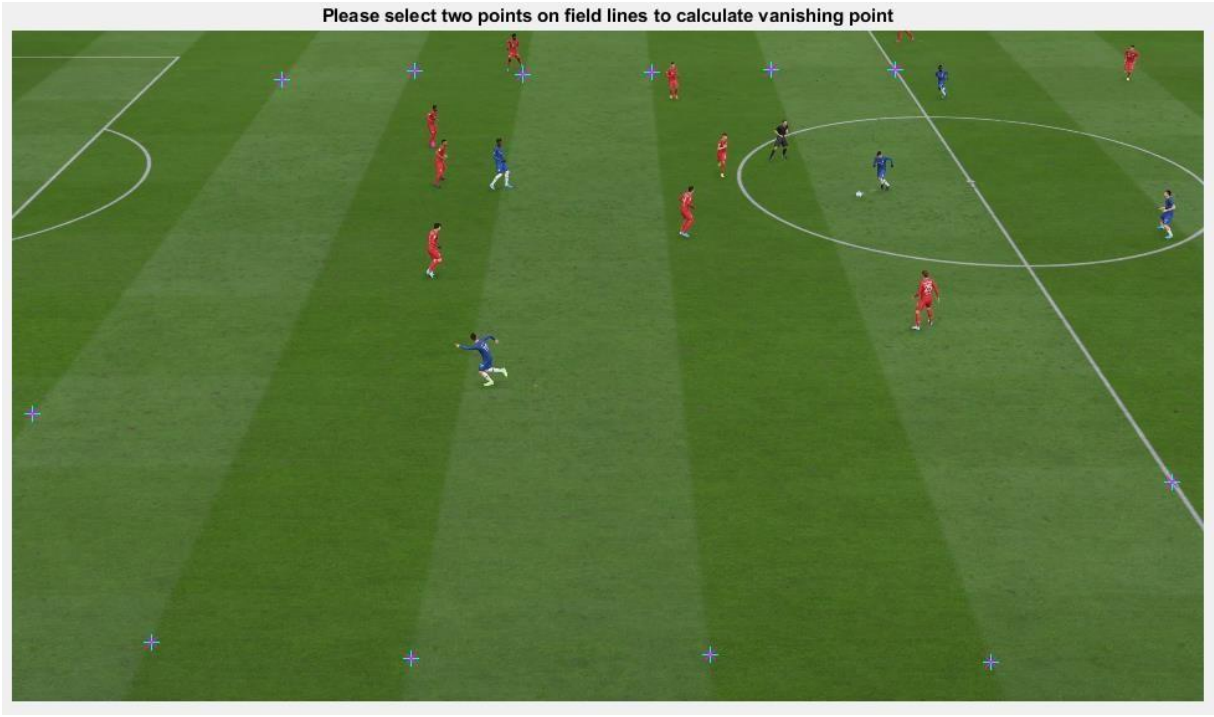




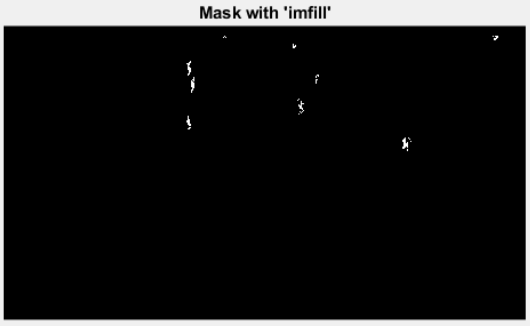
(Figure 12: Test 12 Failed, necessity of cutting tribune)

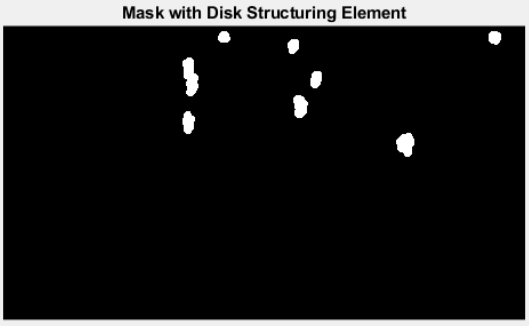


(Figure 13: Test 13 Correct)

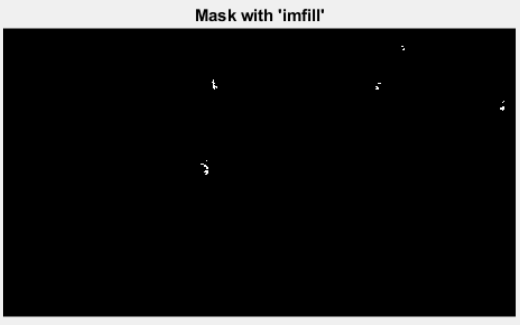


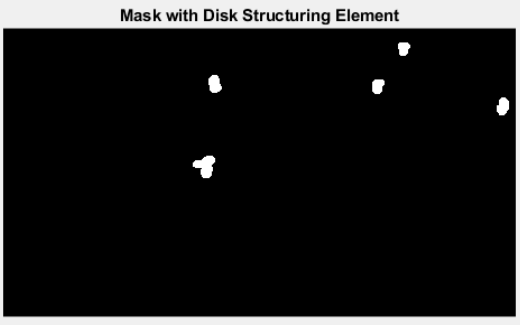
Defending Team

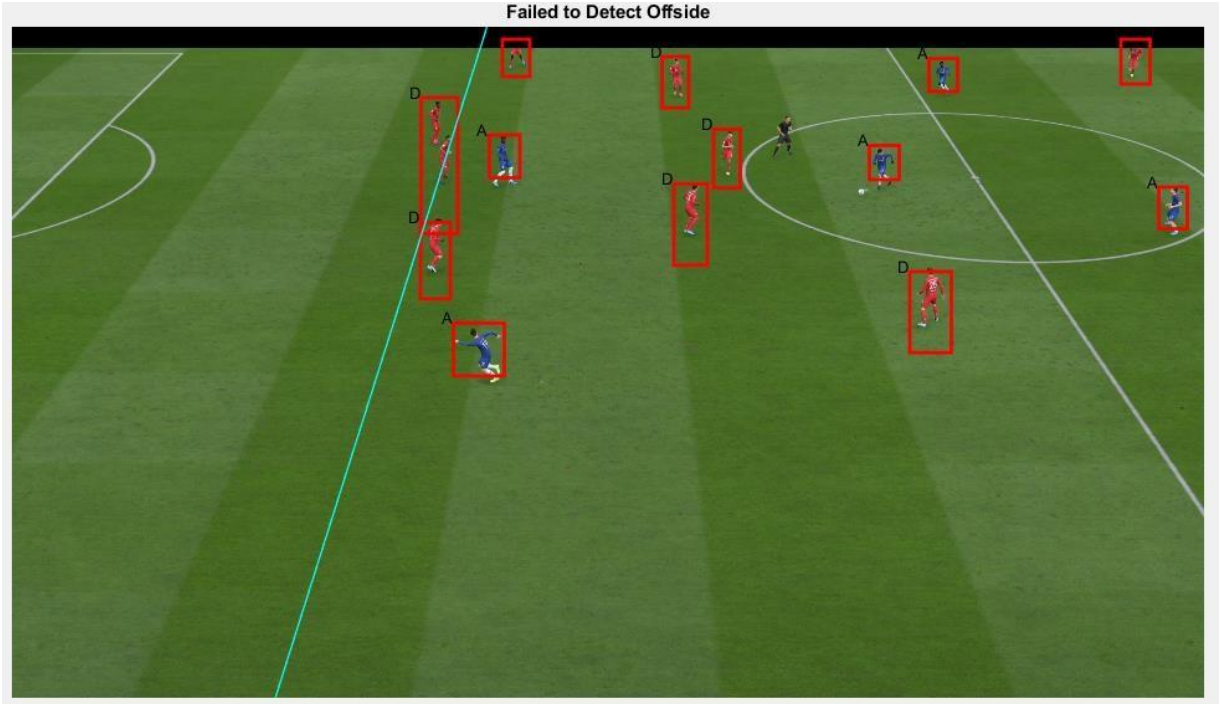




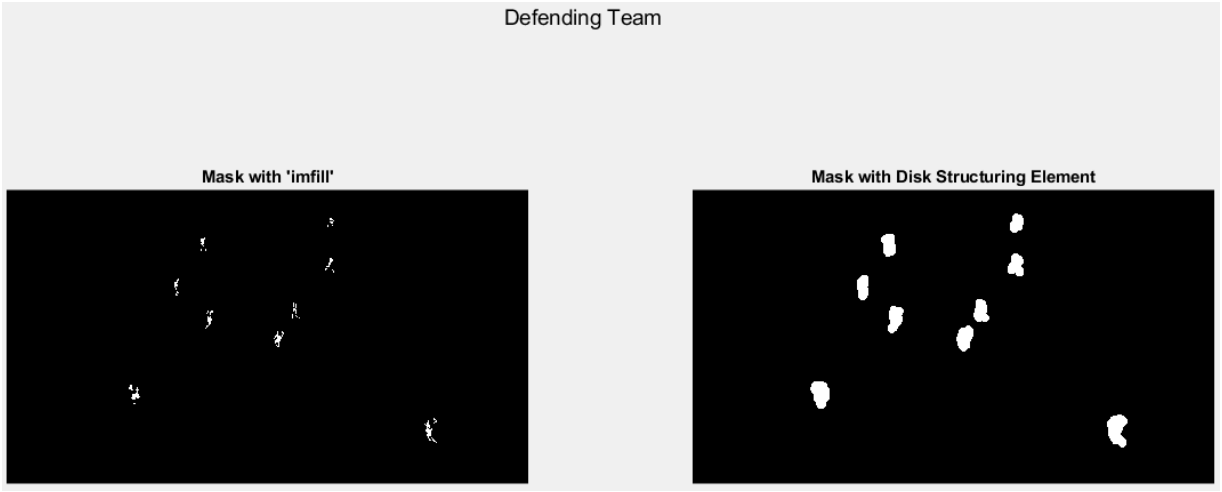
Attacking Team

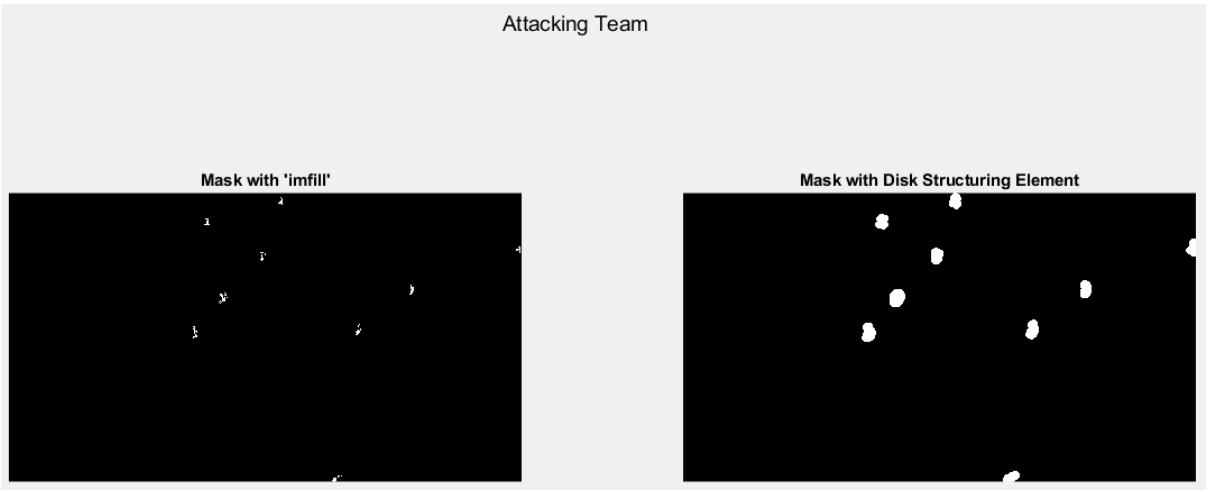


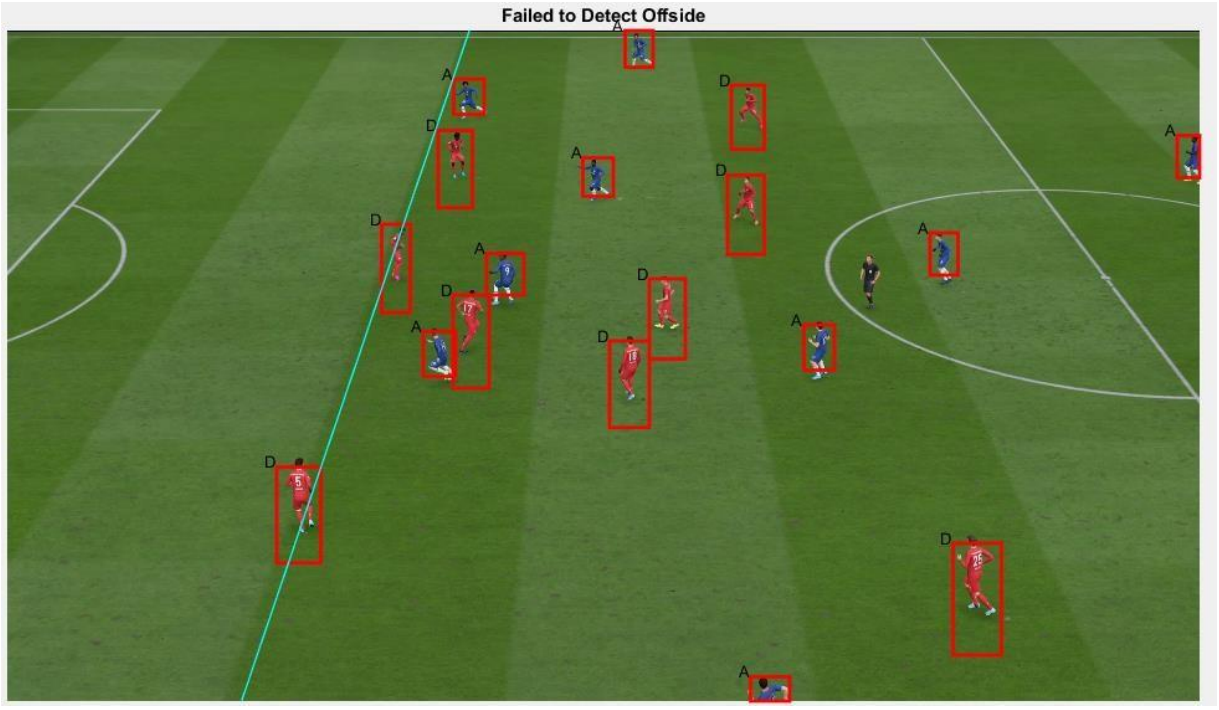




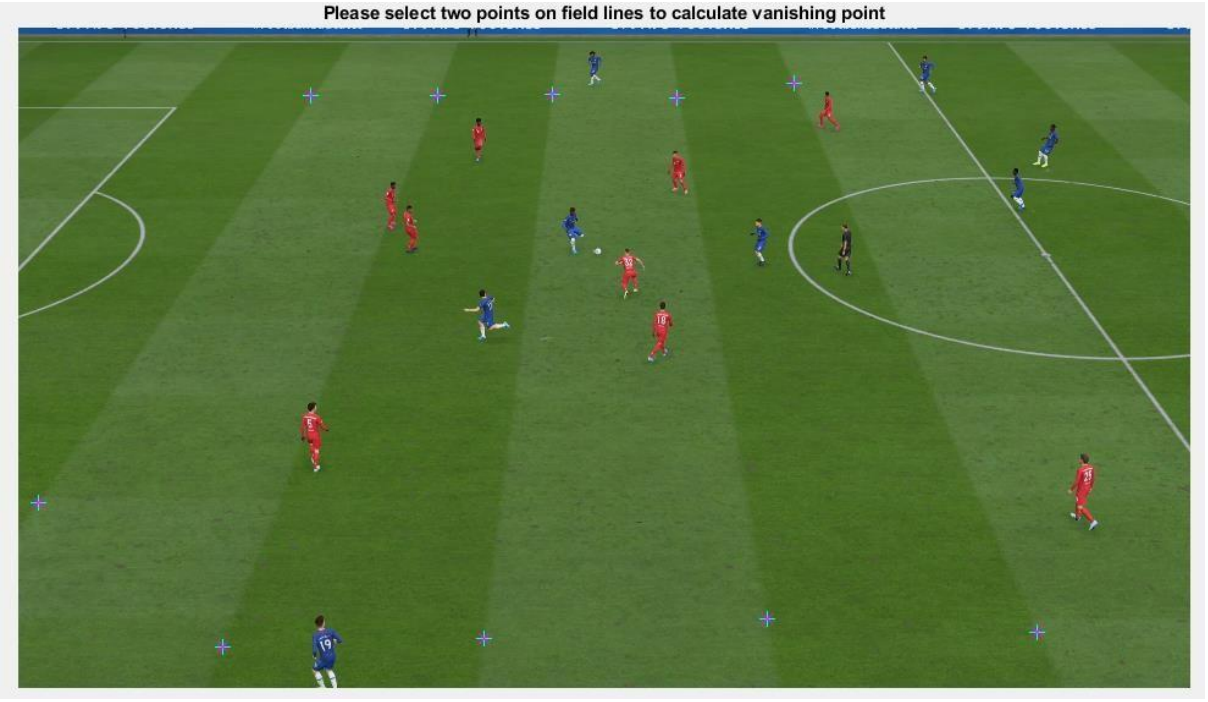
(Figure 14: Test 14 Correct, but a bit problematic player detection)



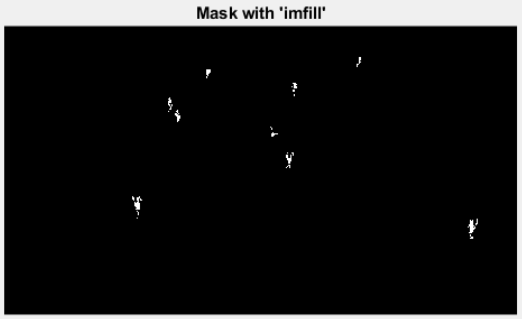


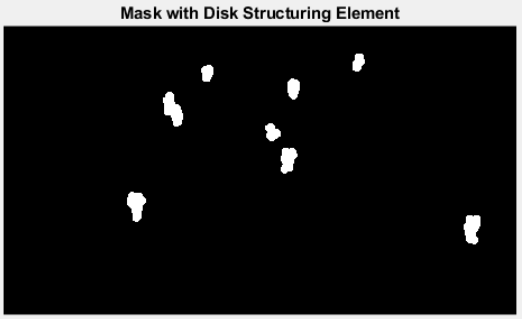


(Figure 15: Test 15 Correct)

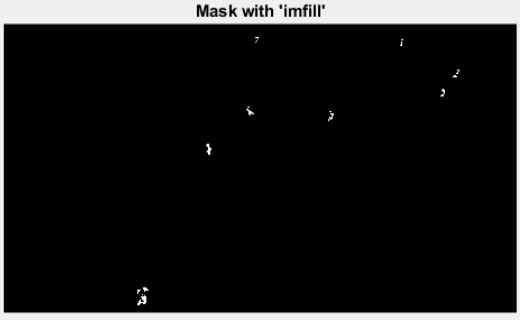


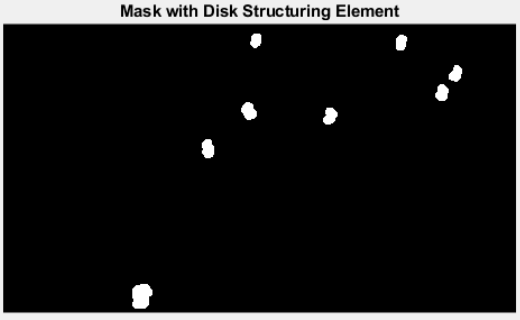
Defending Team

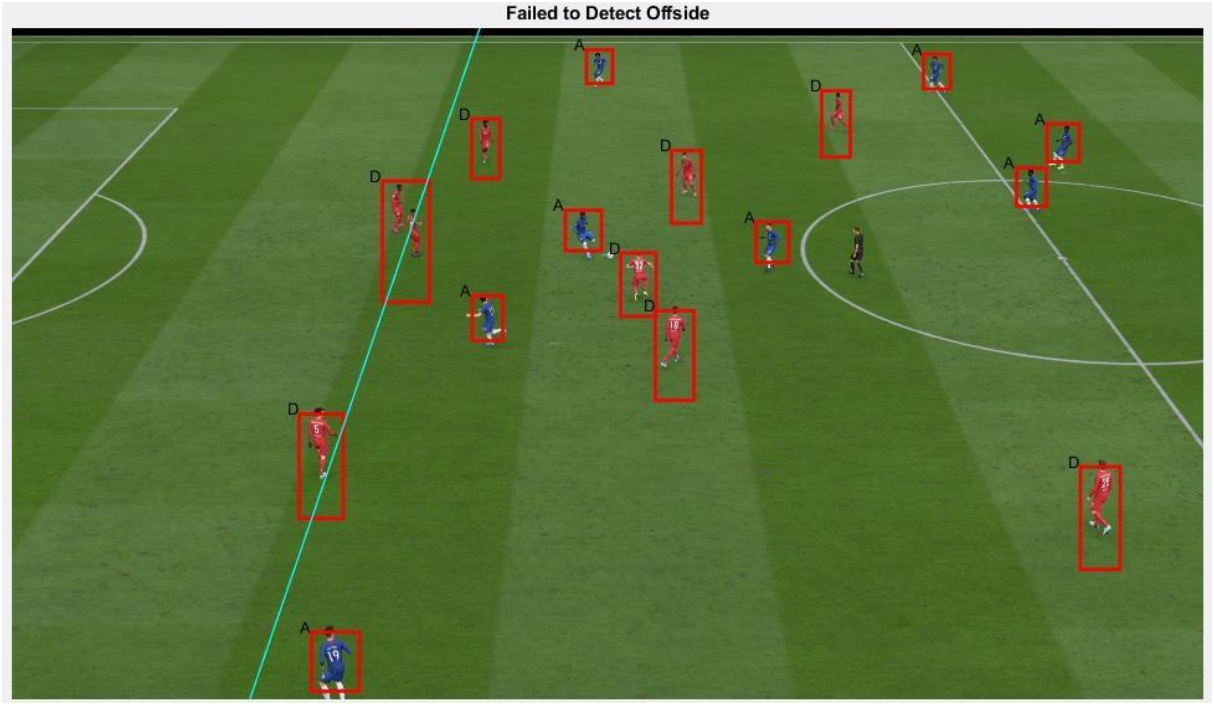




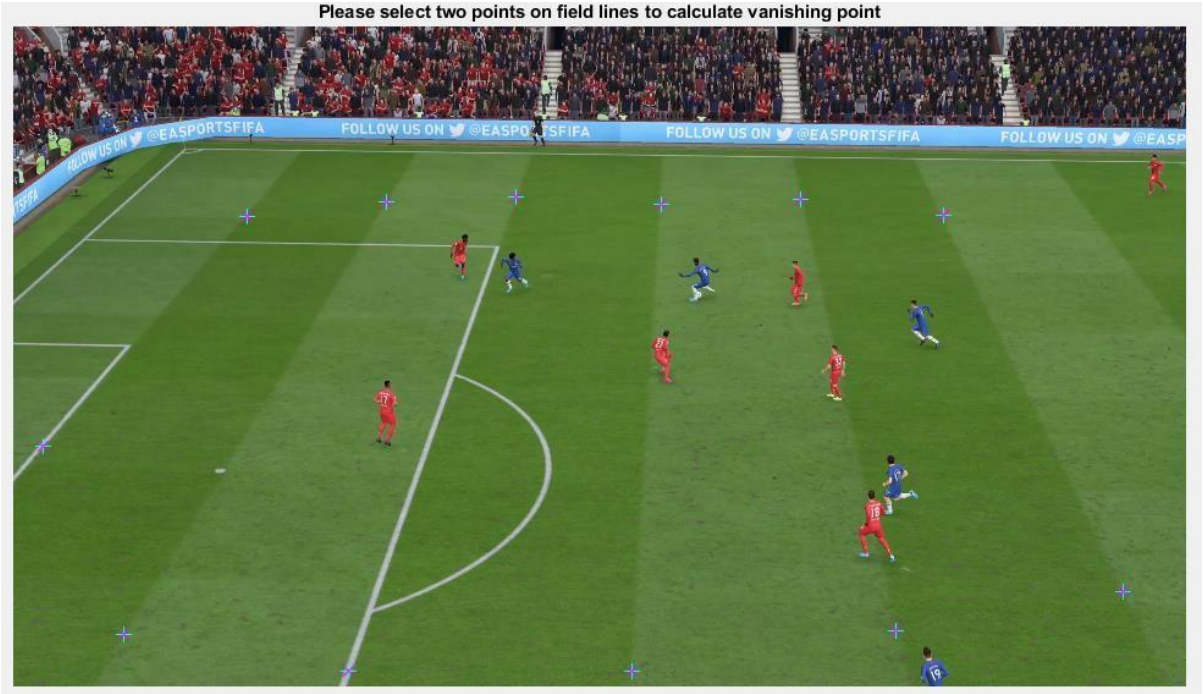
Attacking Team

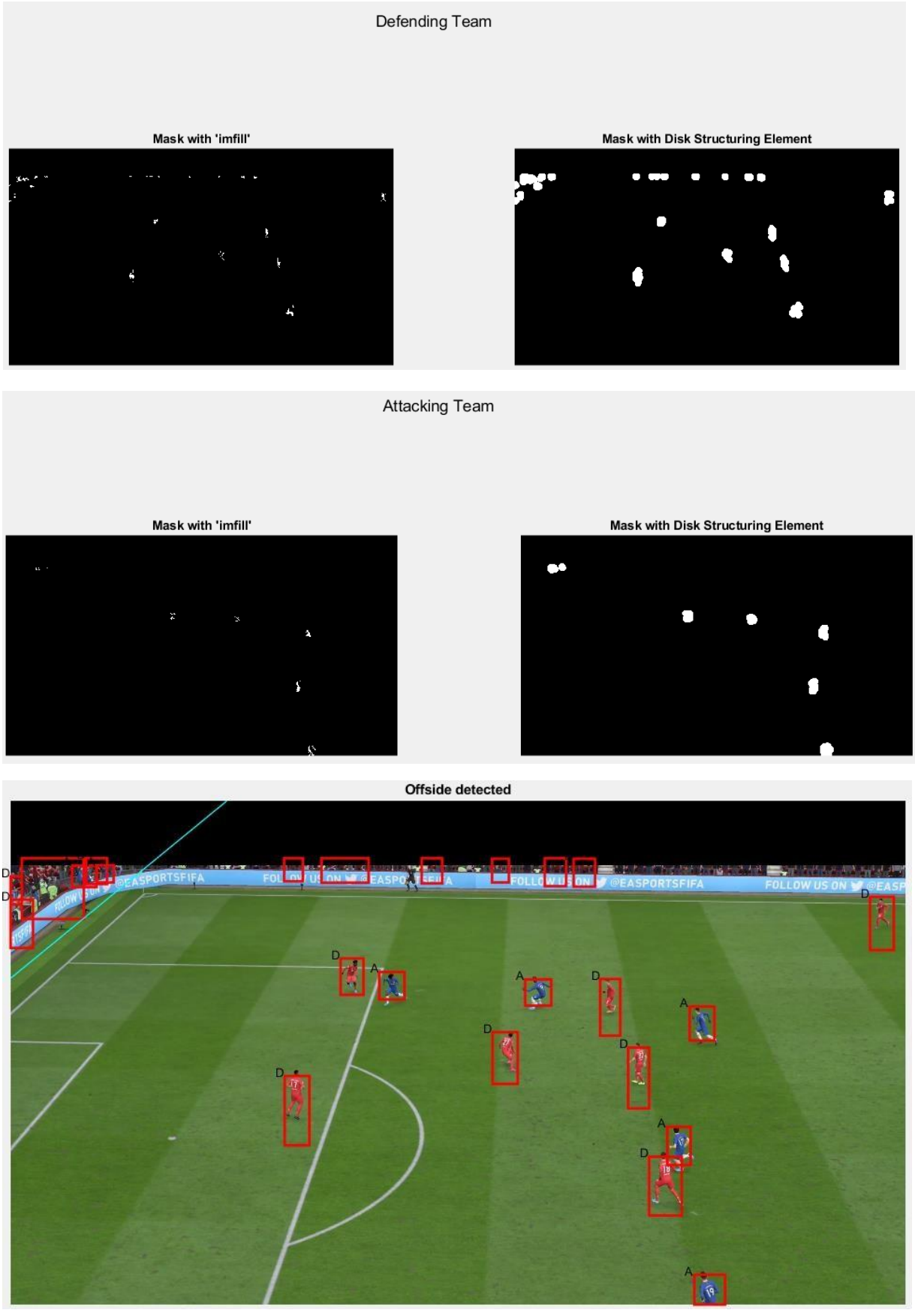






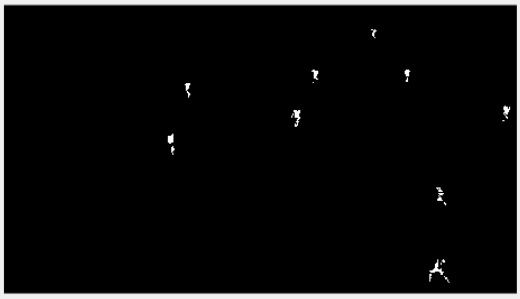
(Figure 16: Test 16 Correct, but a bit problematic player detection)

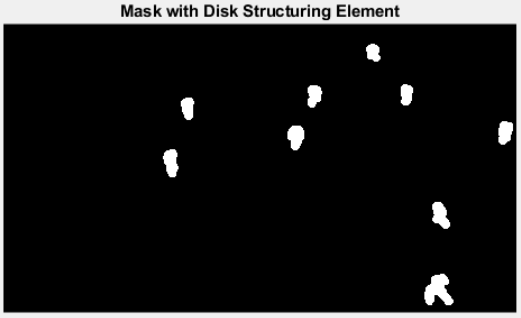




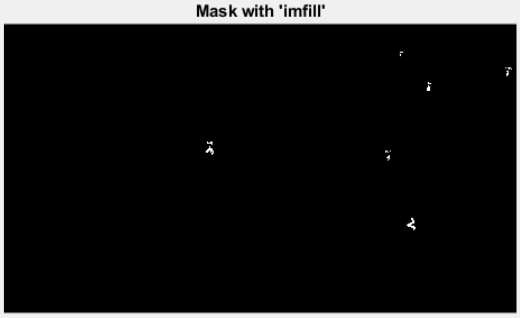
(Figure 17: Test 17 Failed, due to tribune behind the goal)

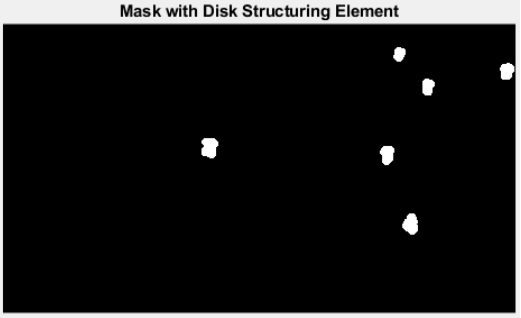
Defending Team

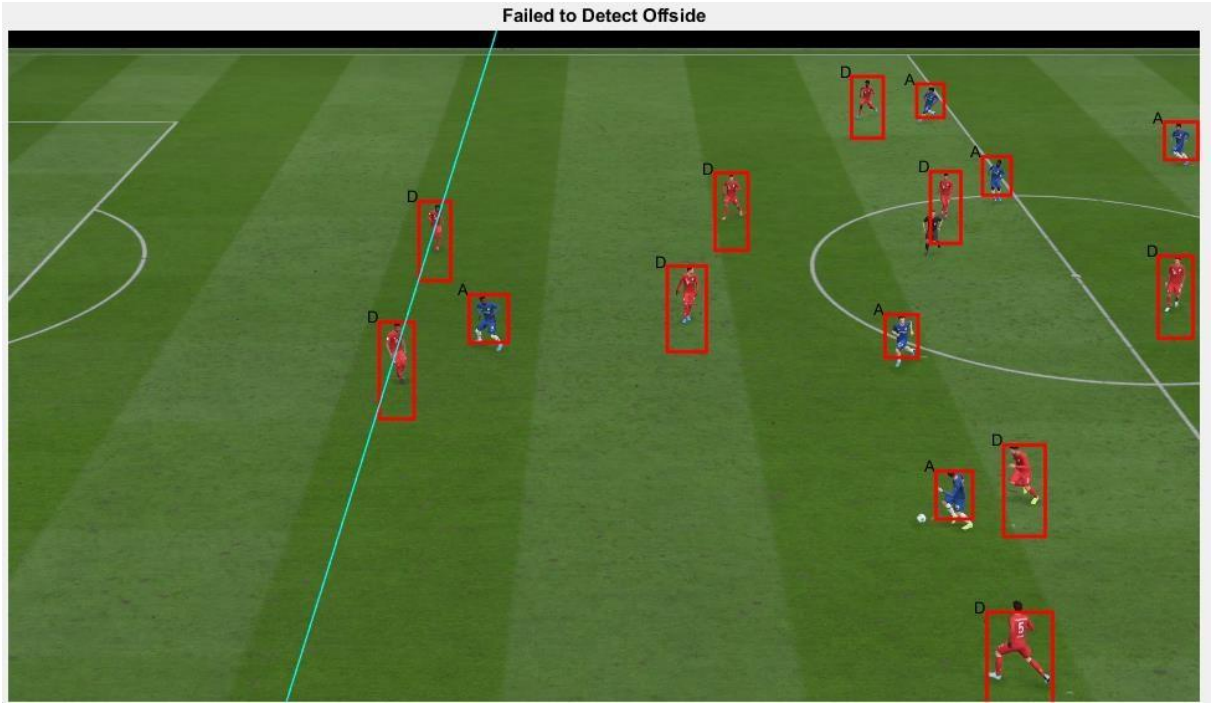




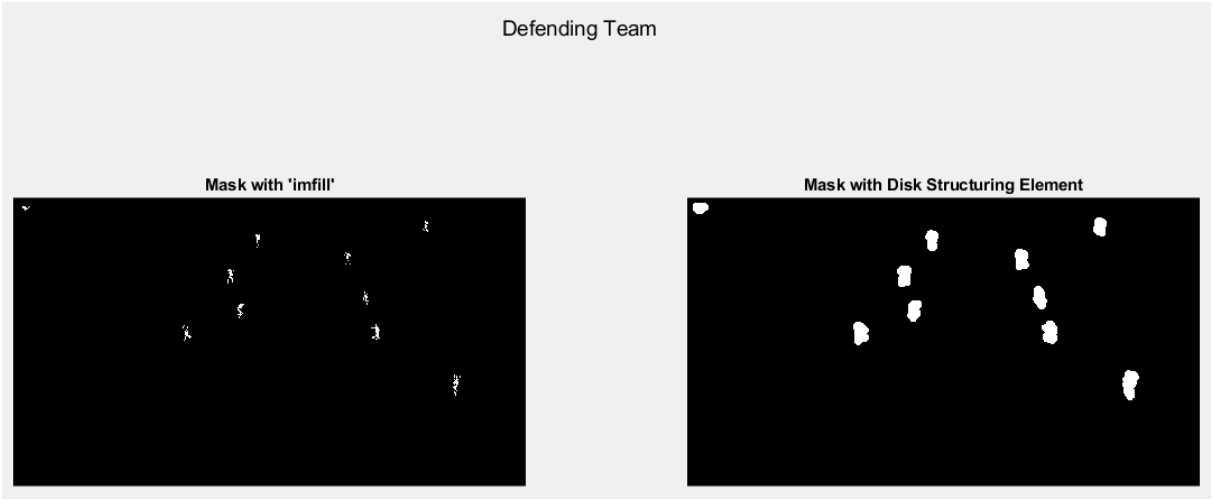
Attacking Team

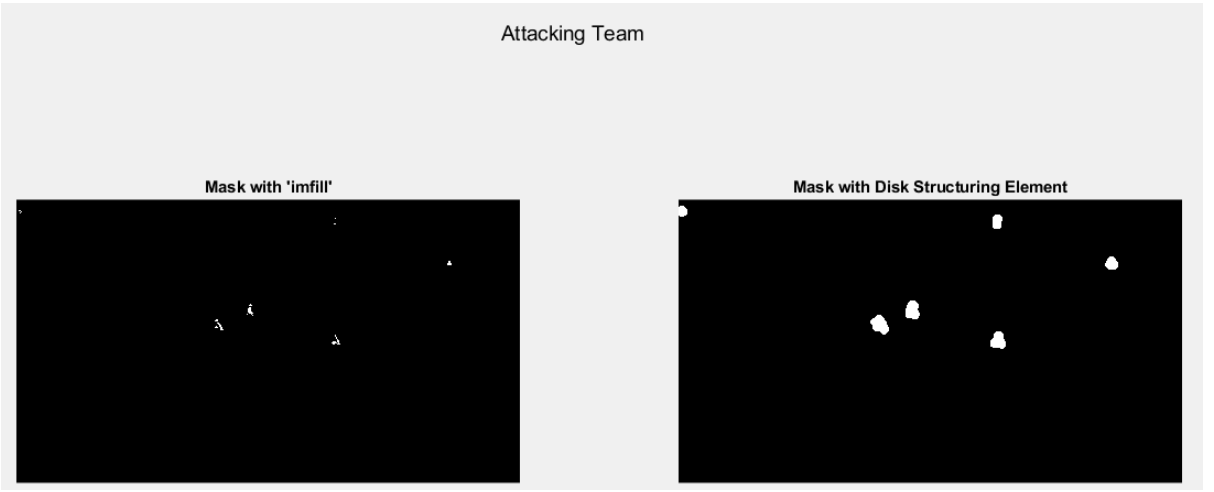


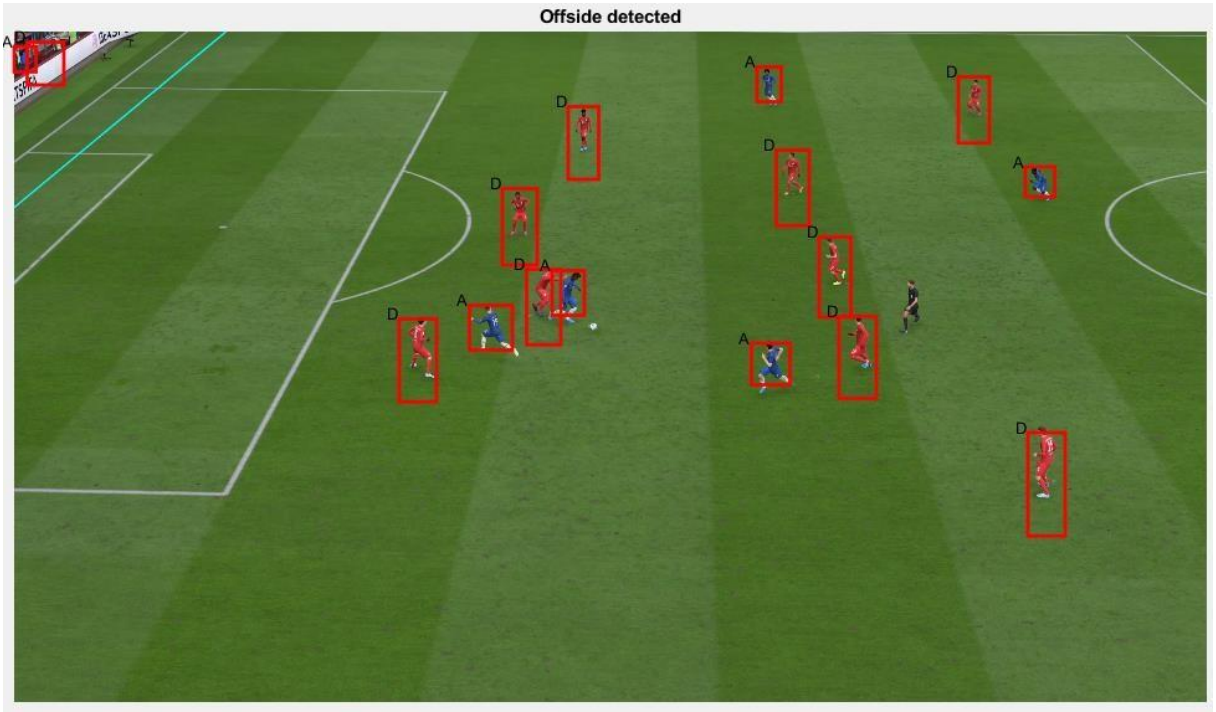




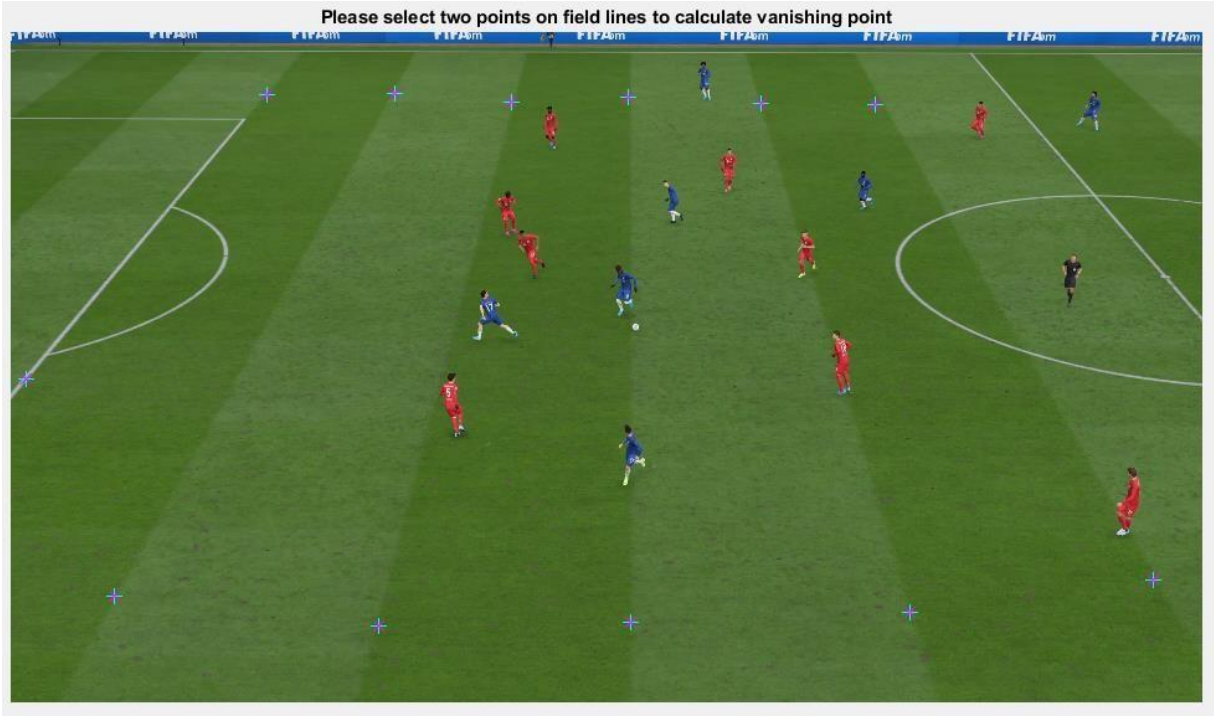
(Figure 18: Test 18 Correct)



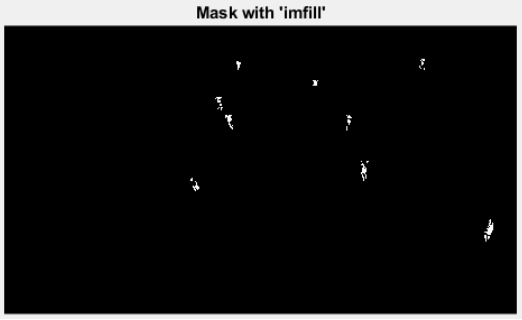


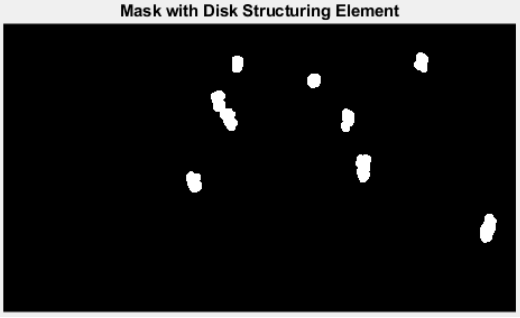


(Figure 19: Test 19 Failed, non-player objects must be discarded)

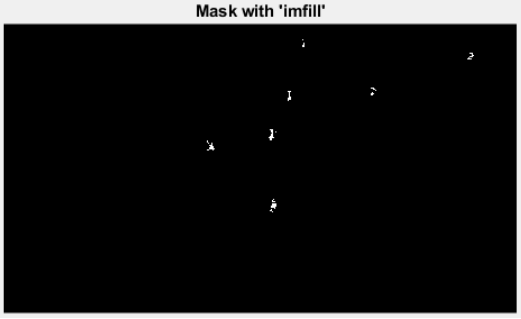


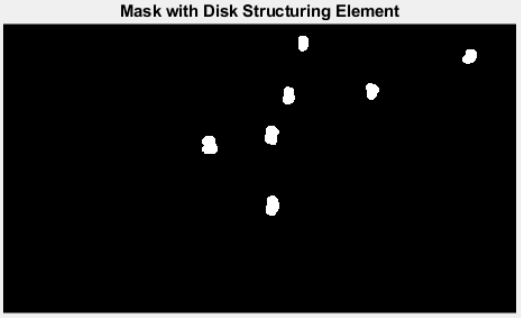
Defending Team

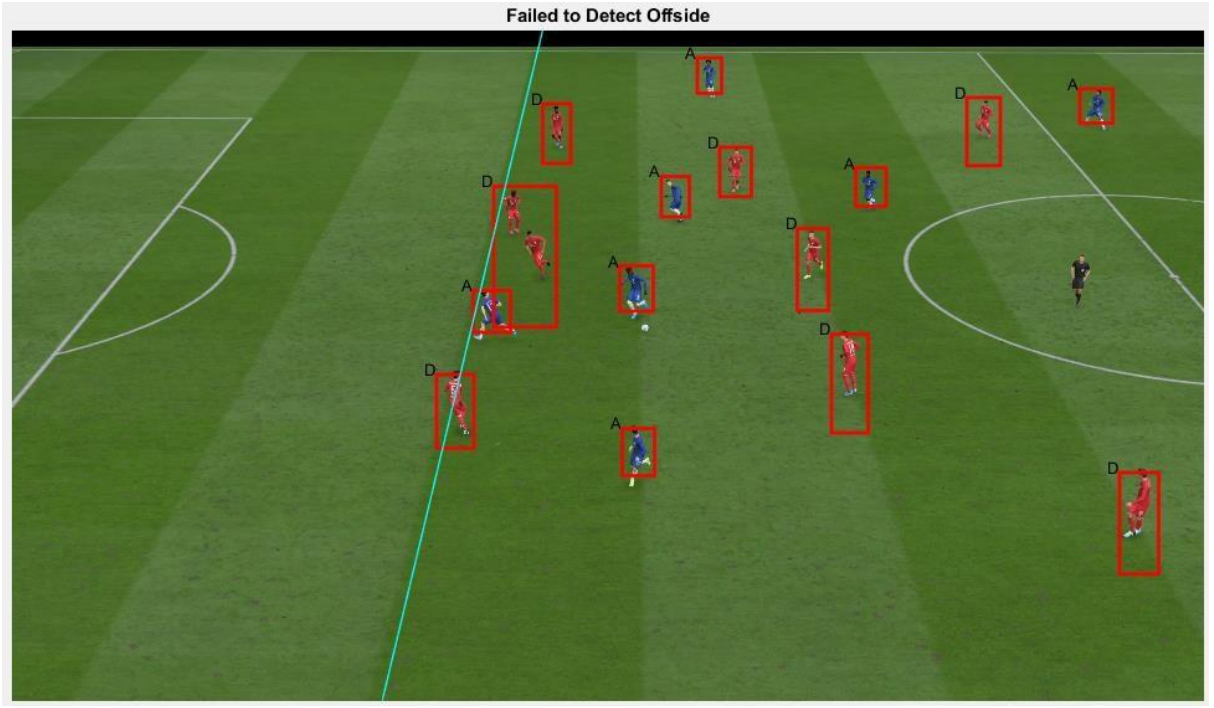




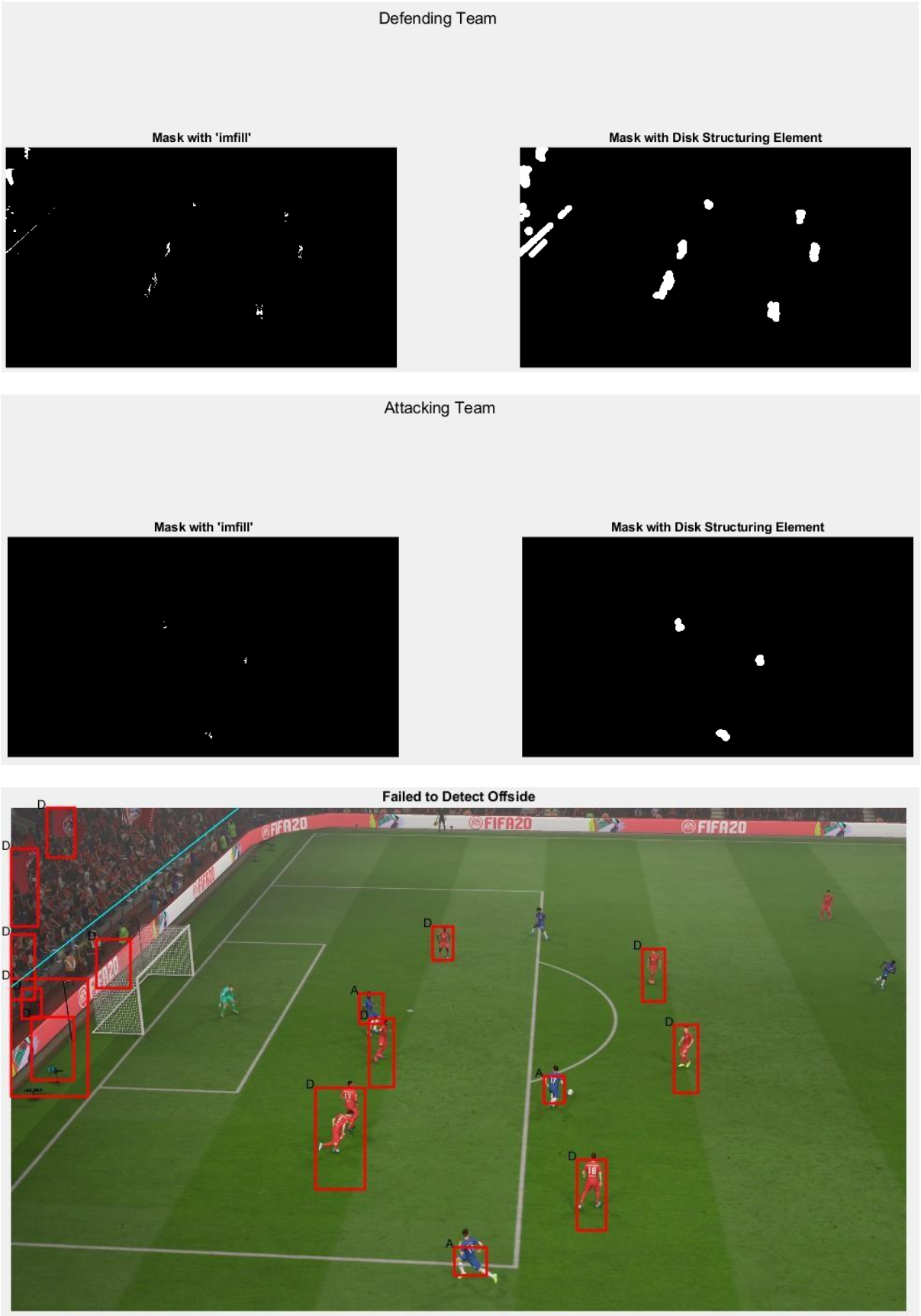
Attacking Team



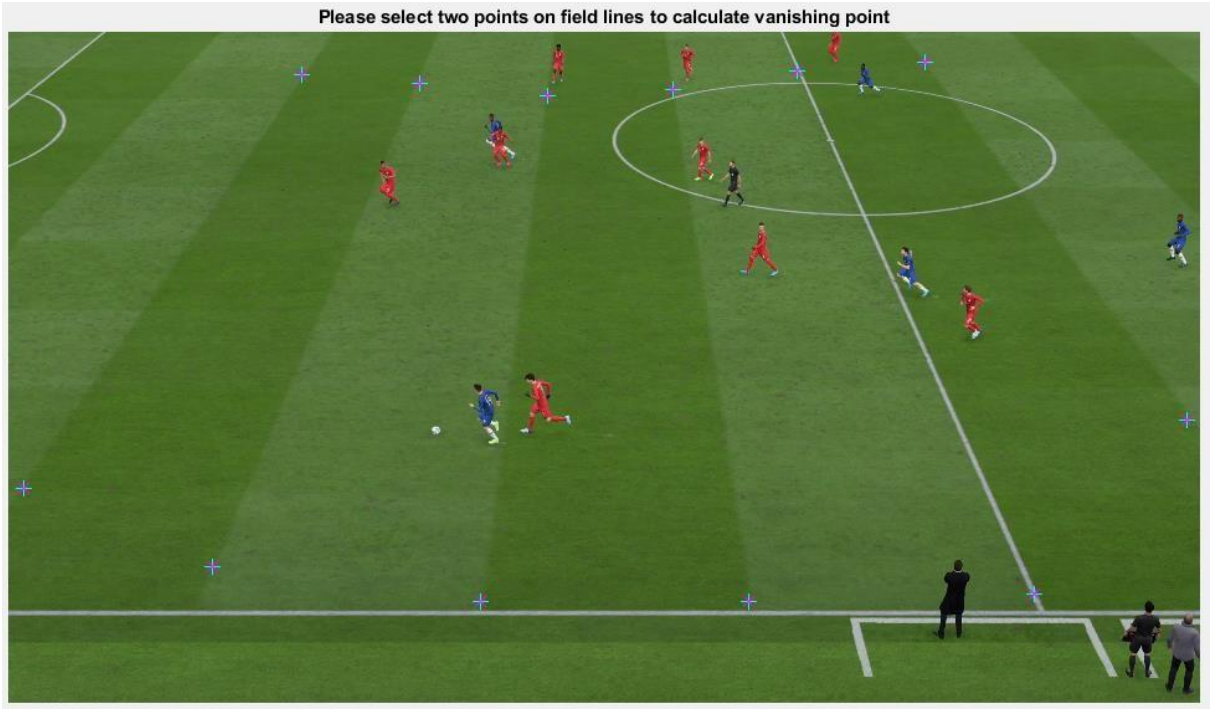




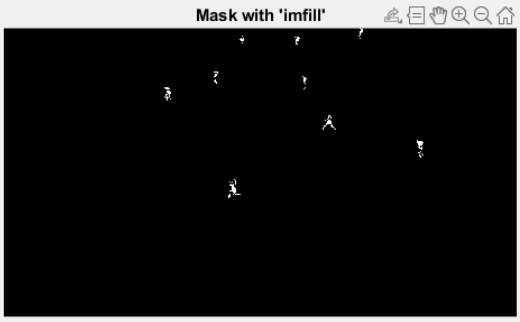
(Figure 20: Test 20 Correct)

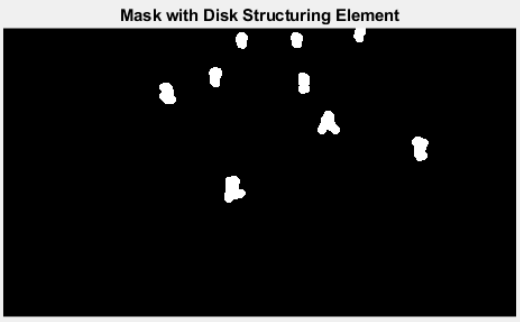


(Figure 21: Test 21 Failed, for the sake of demonstration)

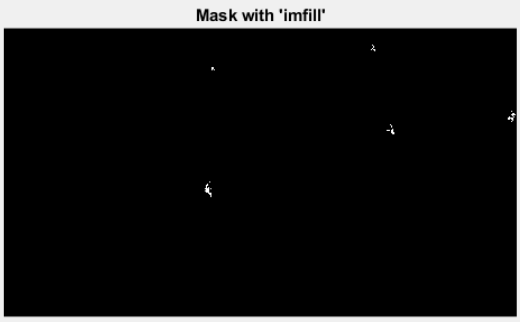


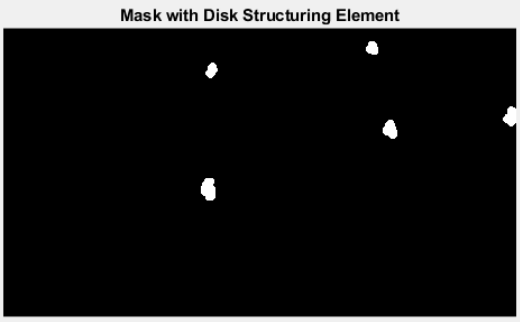
Defending Team

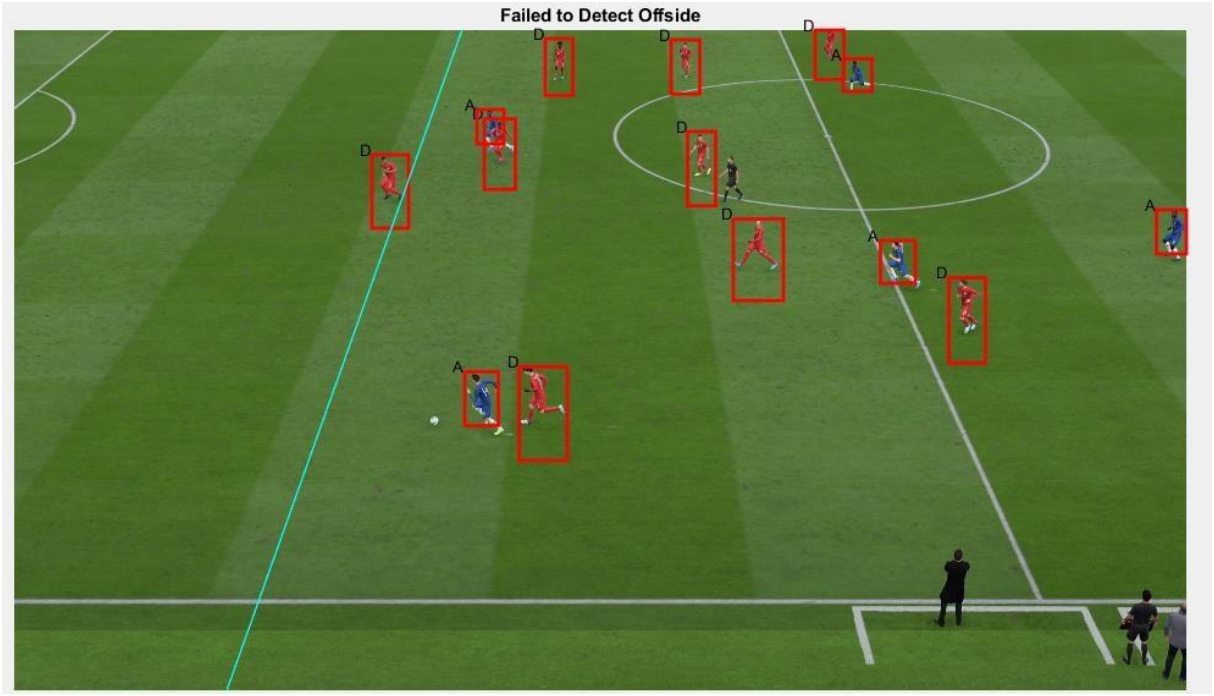




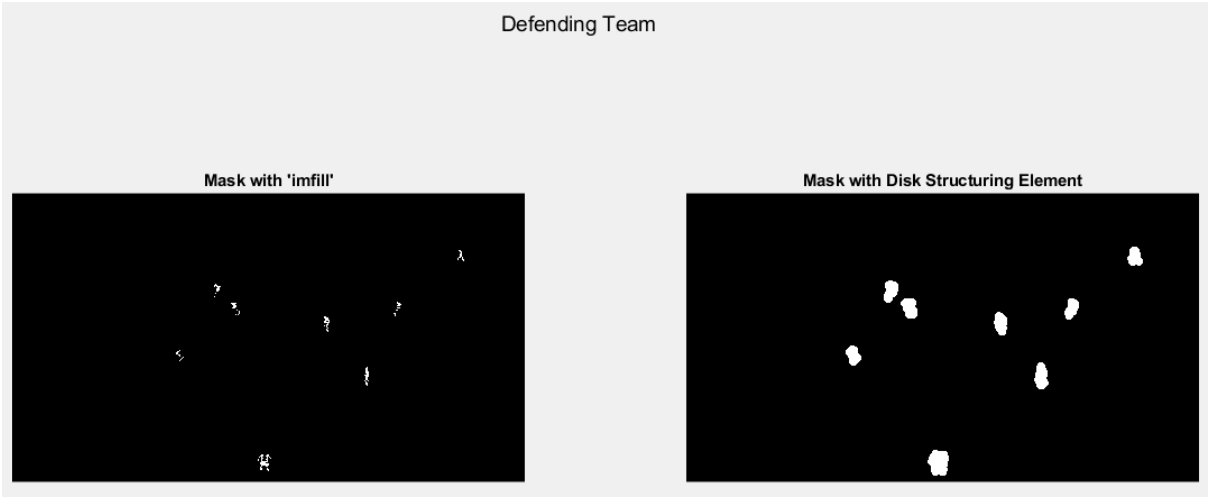
Attacking Team

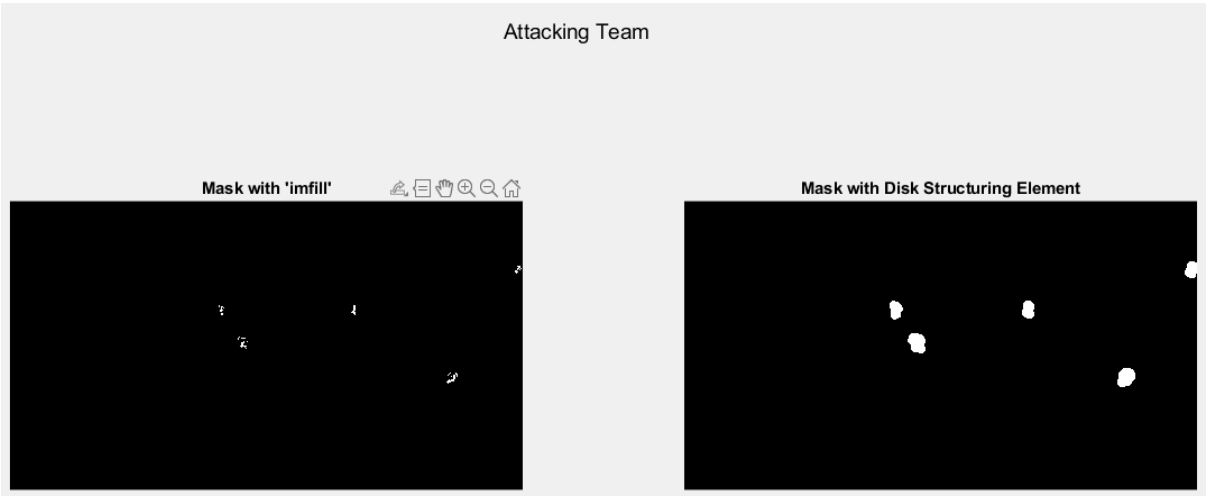


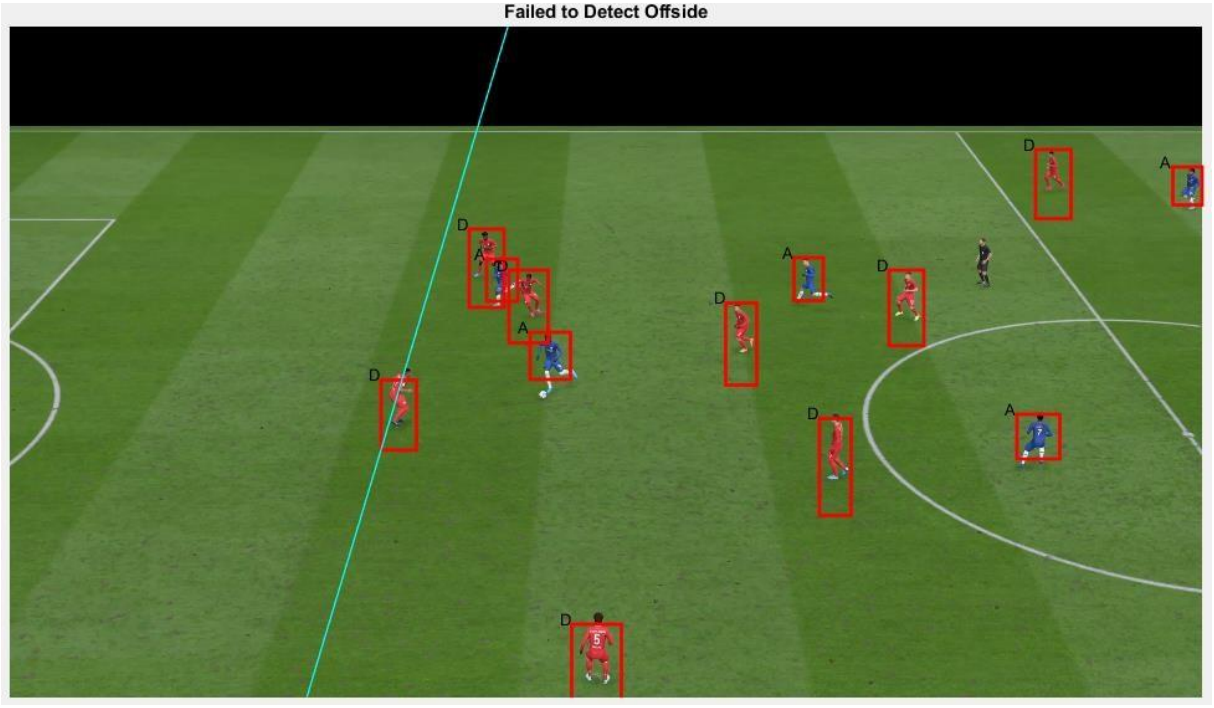




(Figure 22: Test 22 Correct)

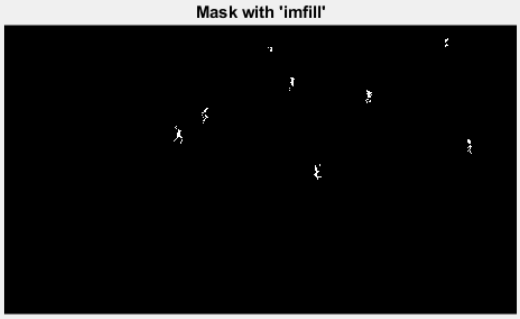


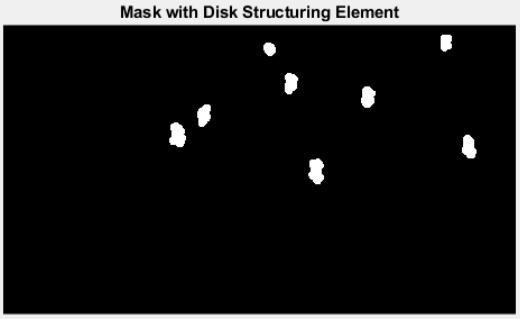




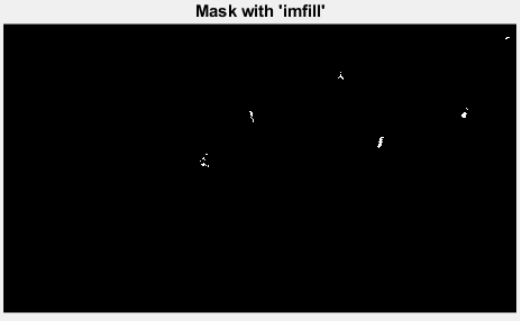
(Figure 23: Test 23 Correct, importance of clipping the tribune)

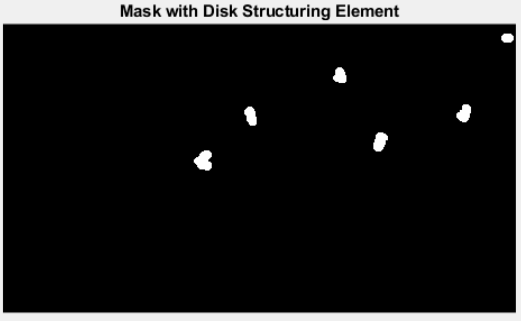
Defending Team

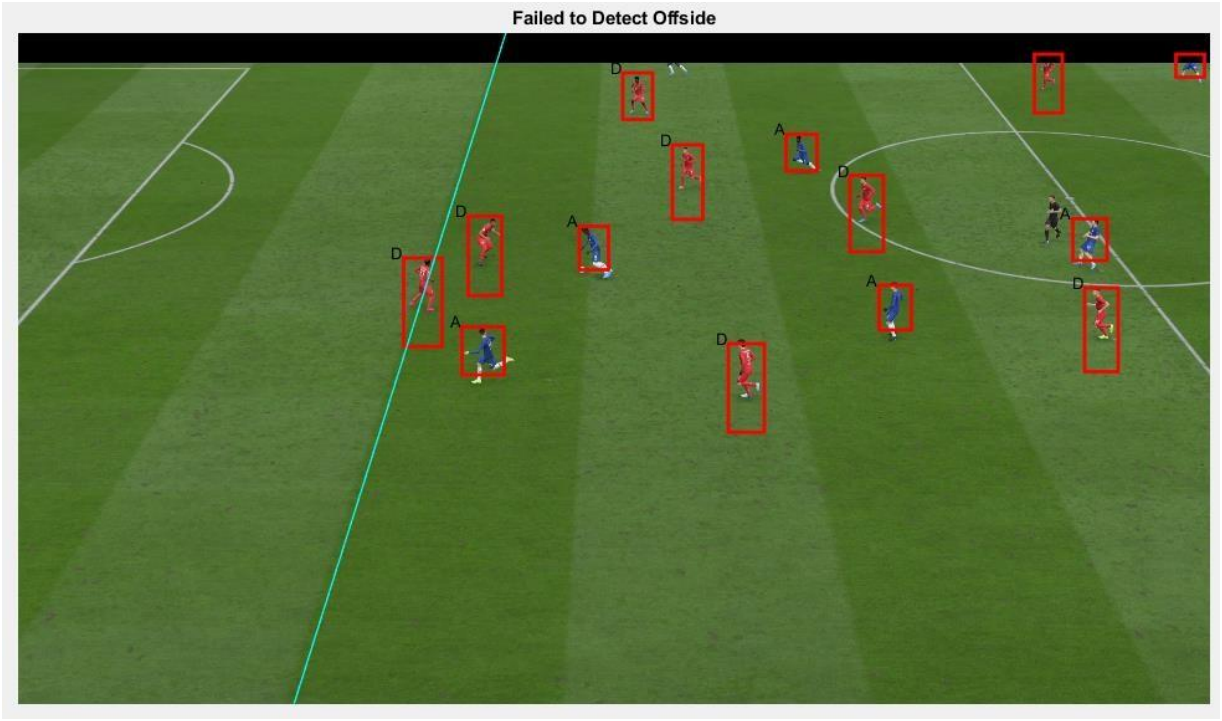




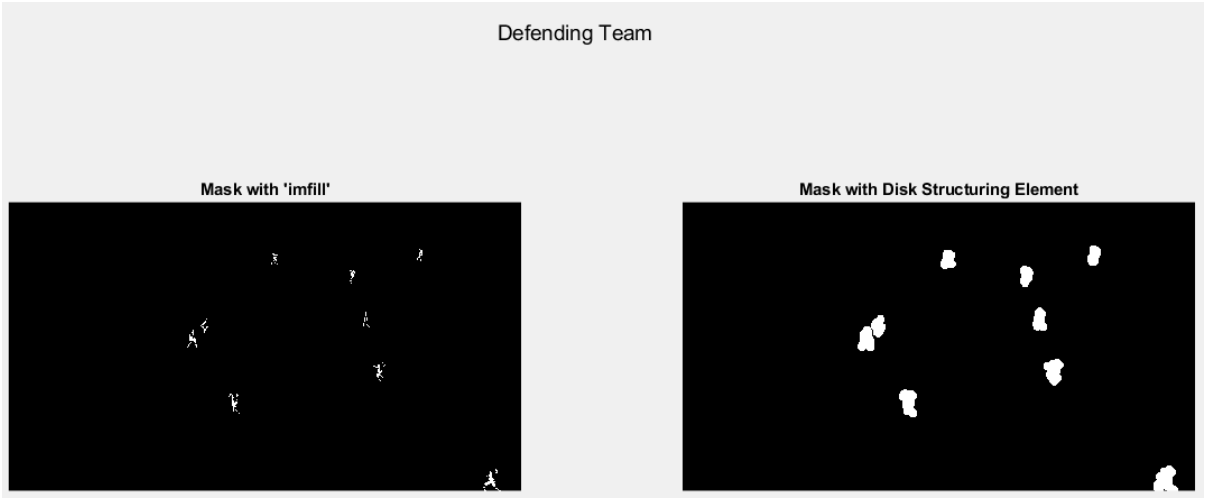
Attacking Team

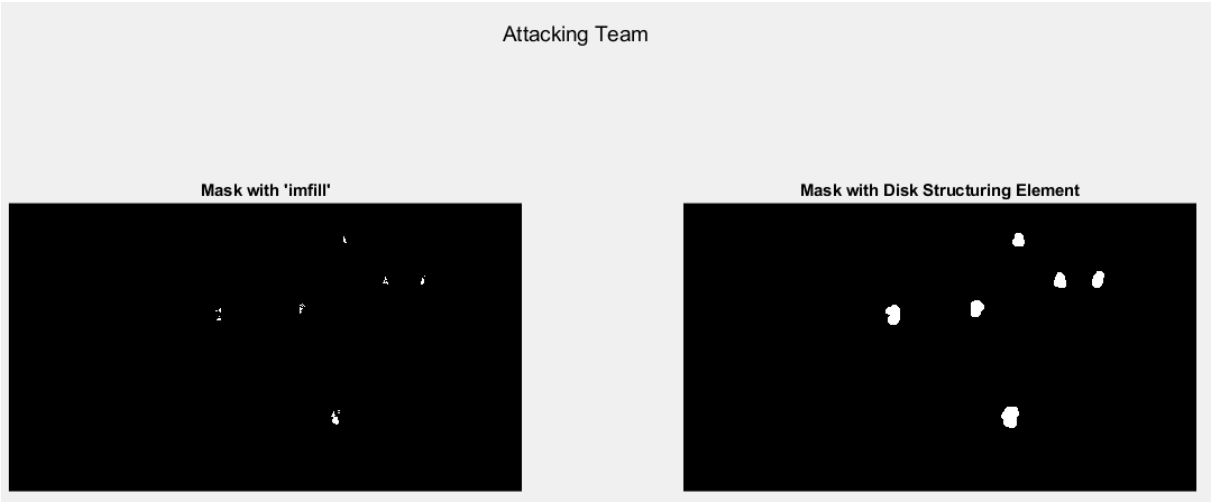


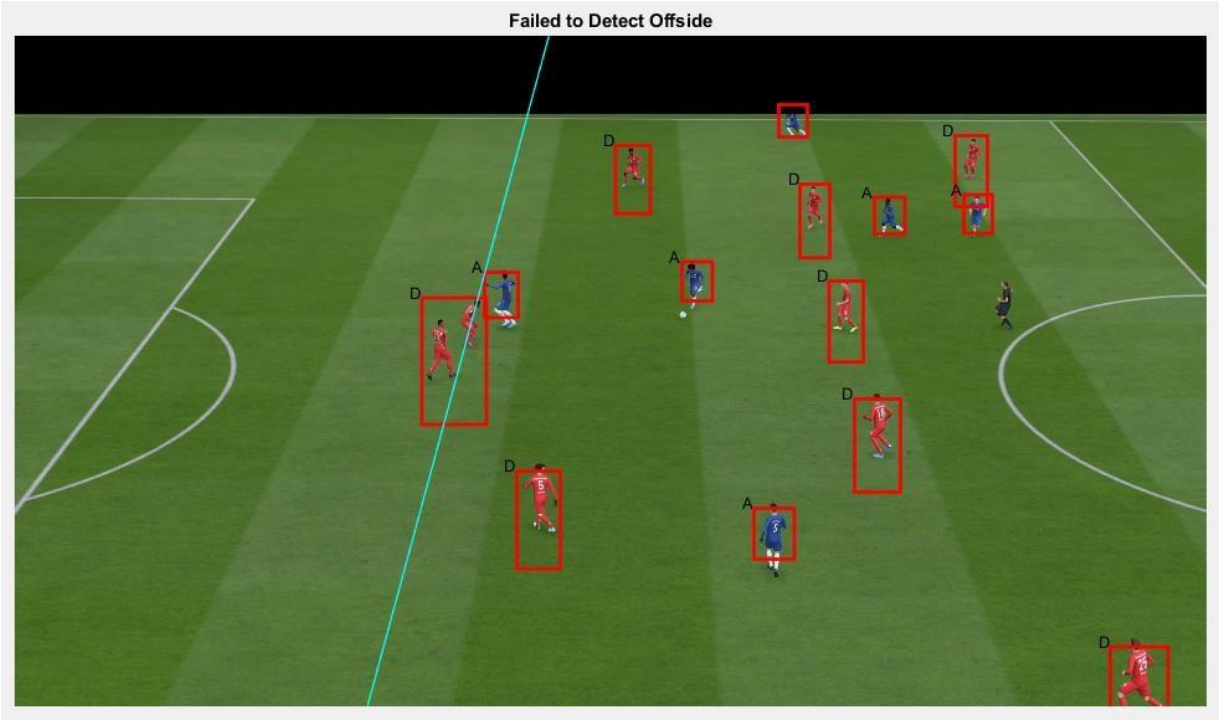




(Figure 24: Test 24 Correct)

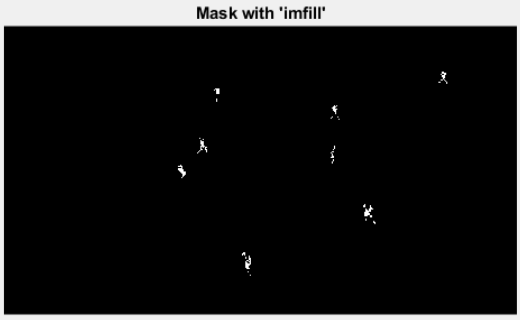


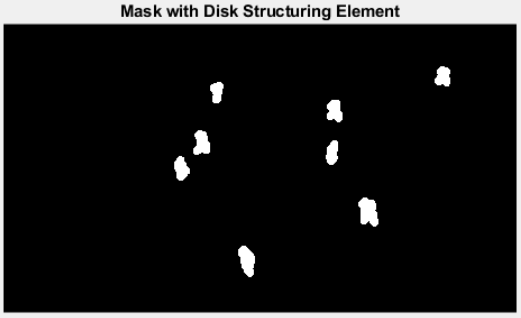




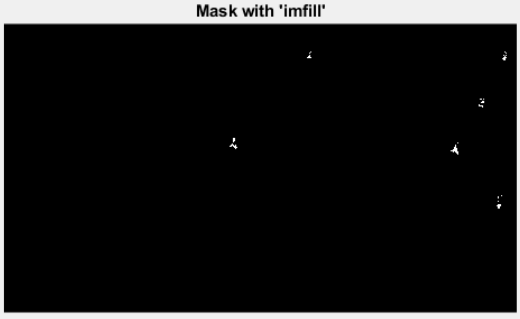
(Figure 25: Test 25 Correct)

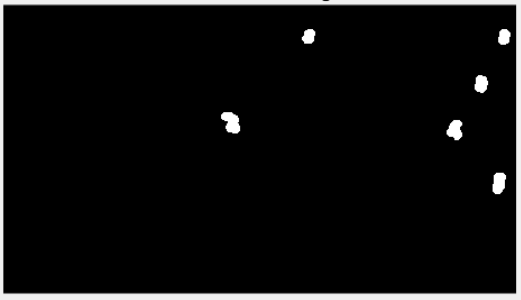
Defending Team

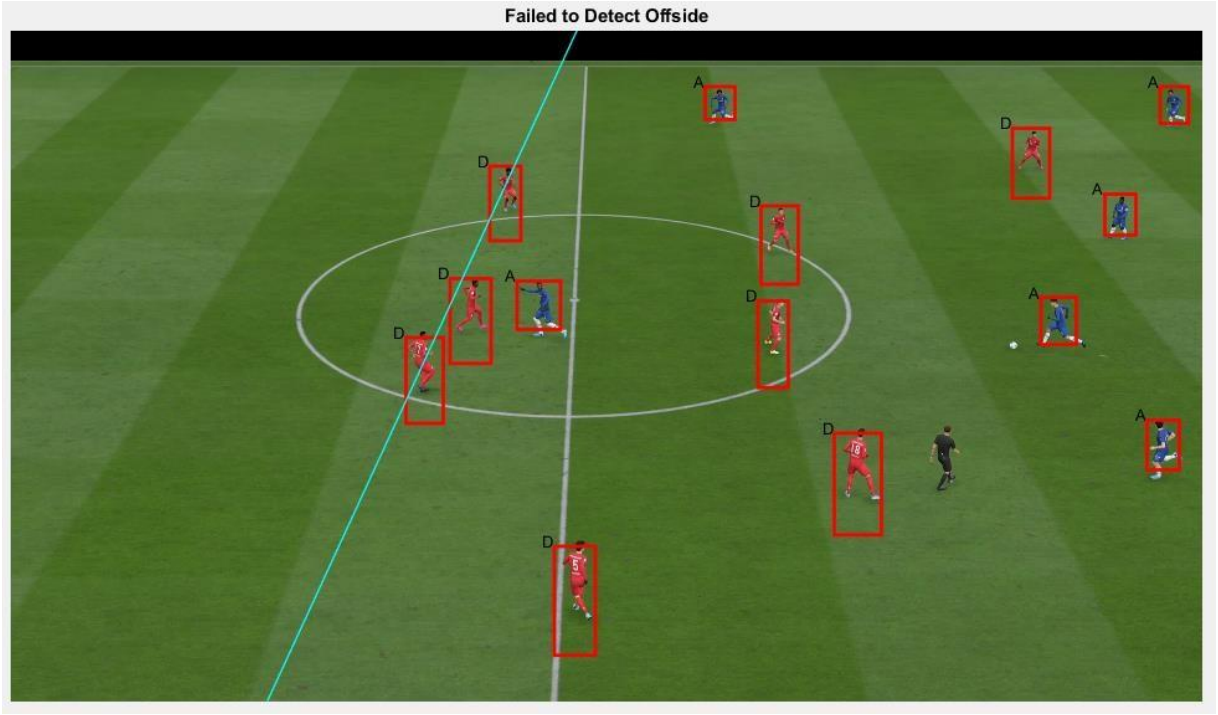




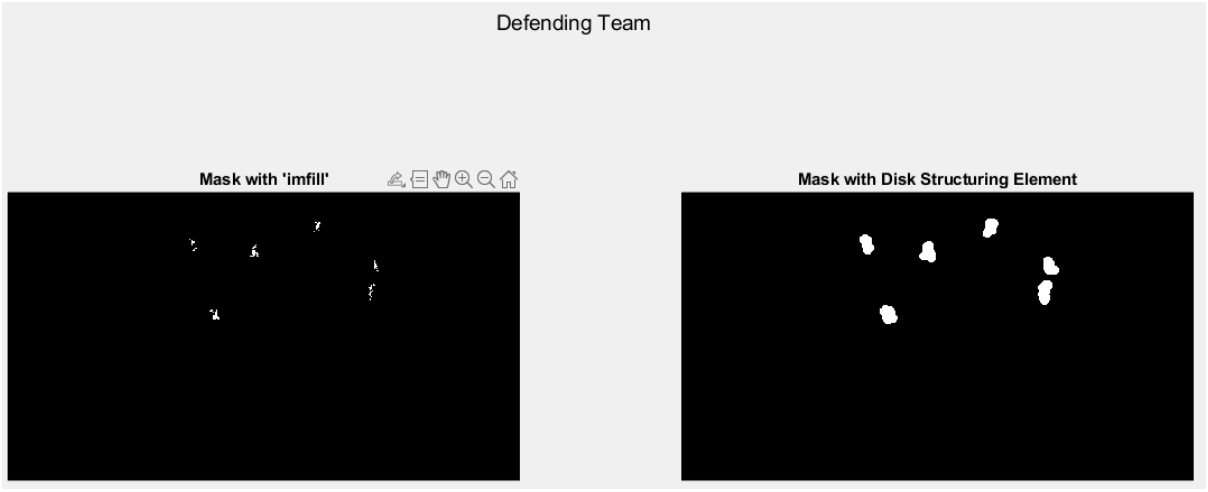
Attacking Team

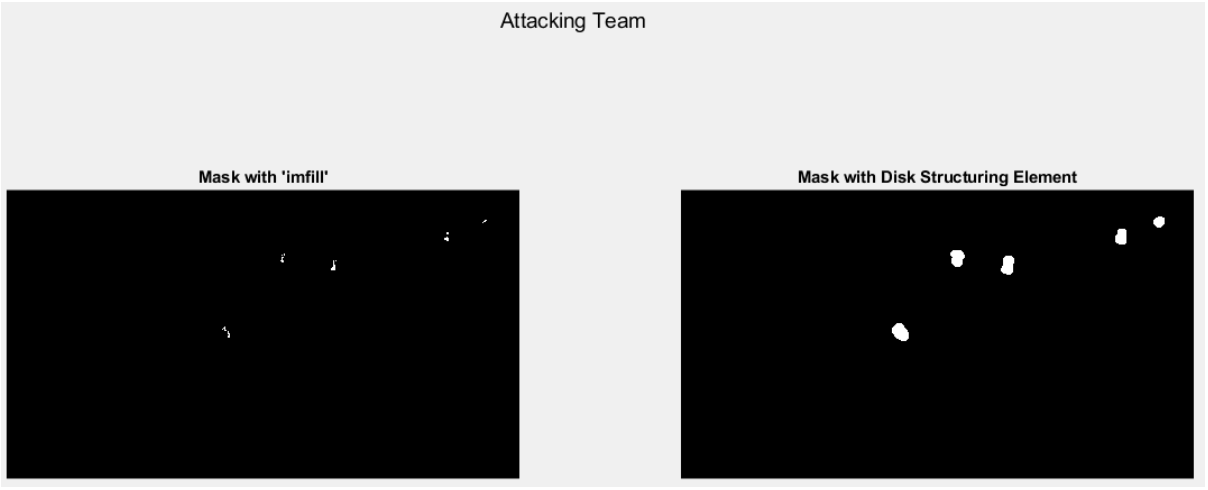


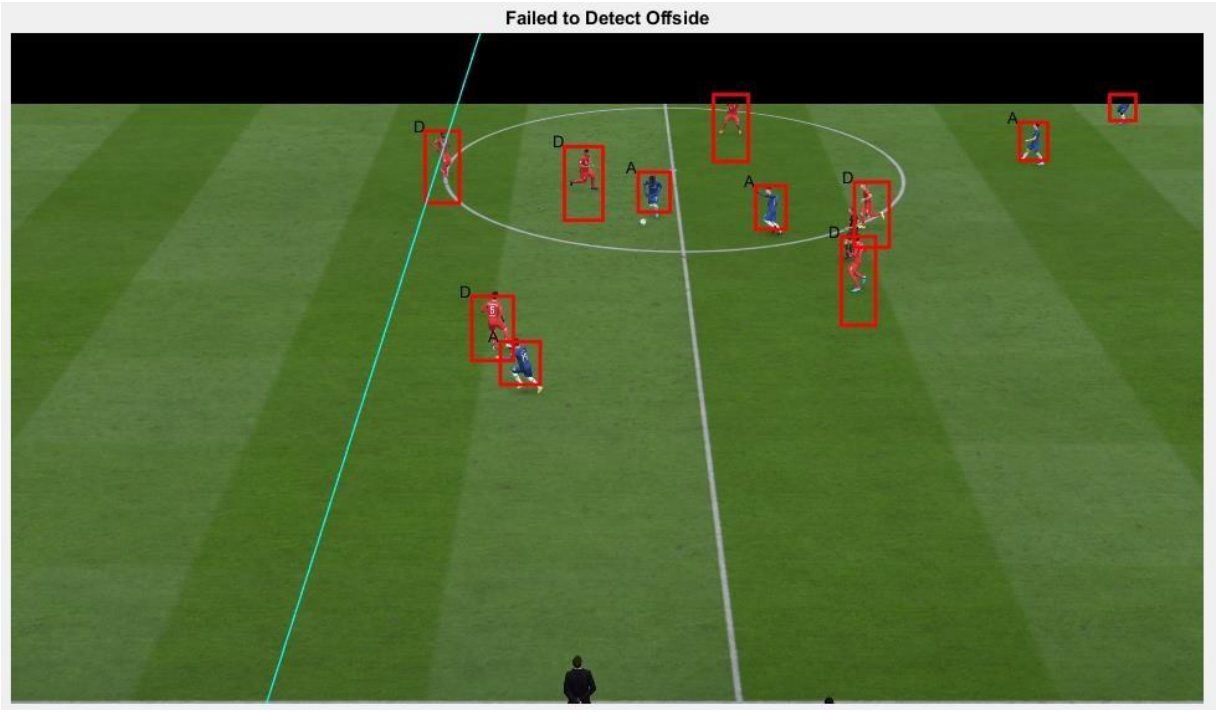




(Figure 26: Test 26 Correct, but effect of choosing to many field lines)

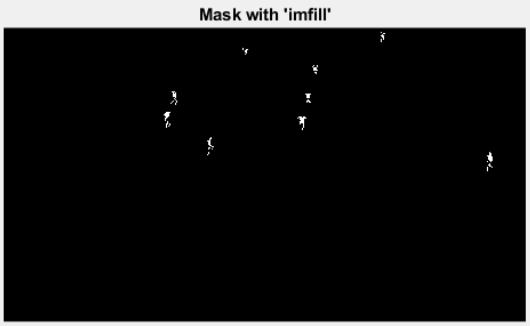


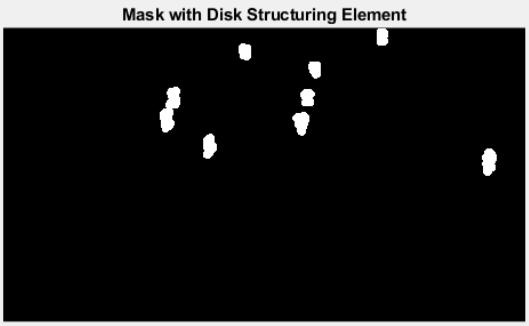




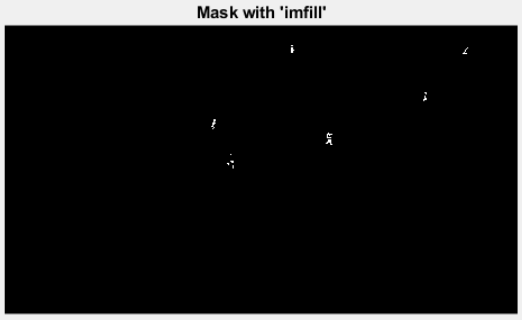
(Figure 27: Test 27 Correct)

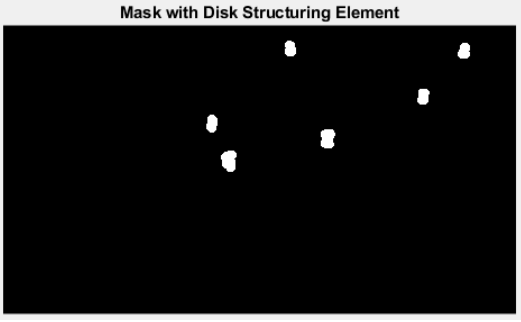
Defending Team

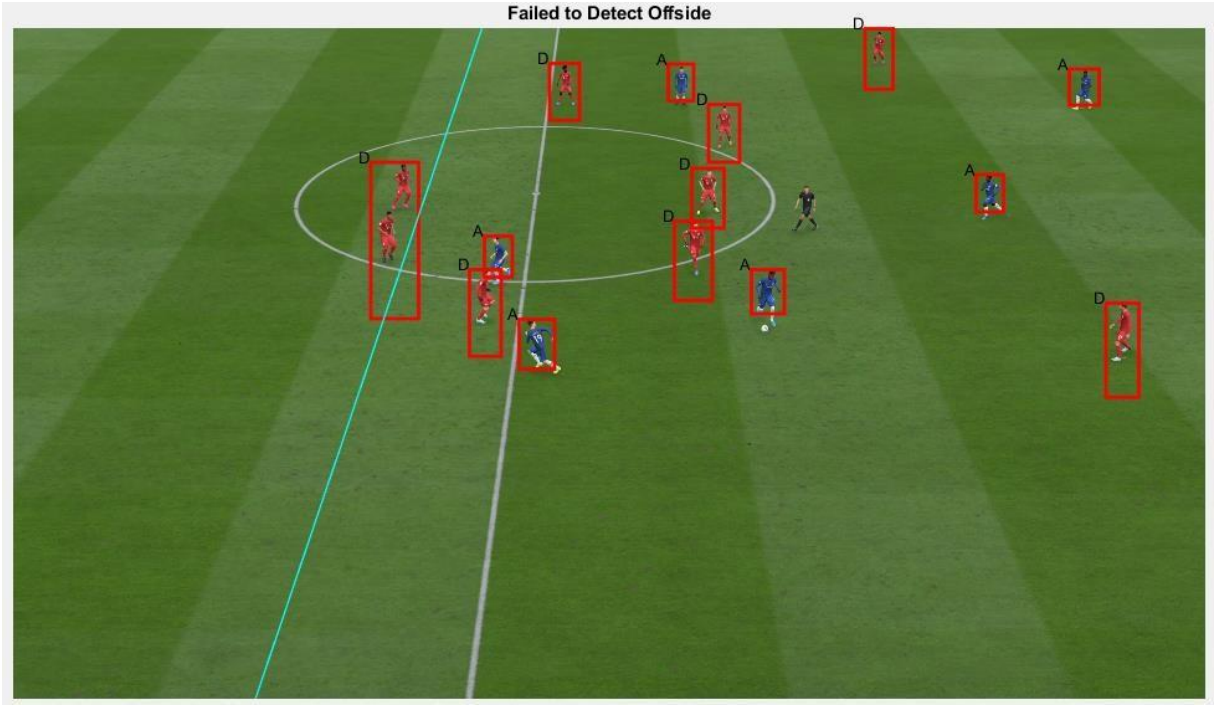




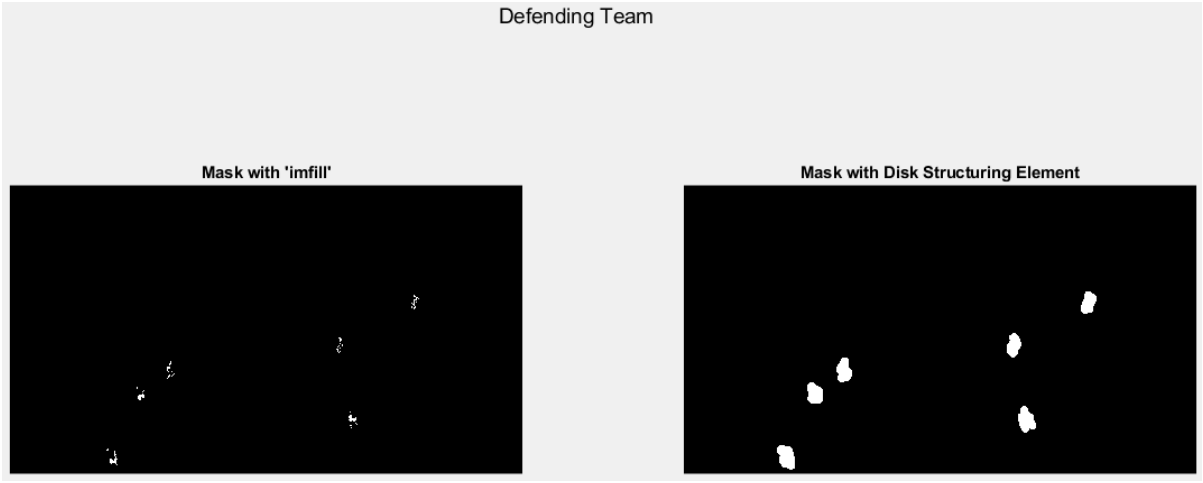
Attacking Team

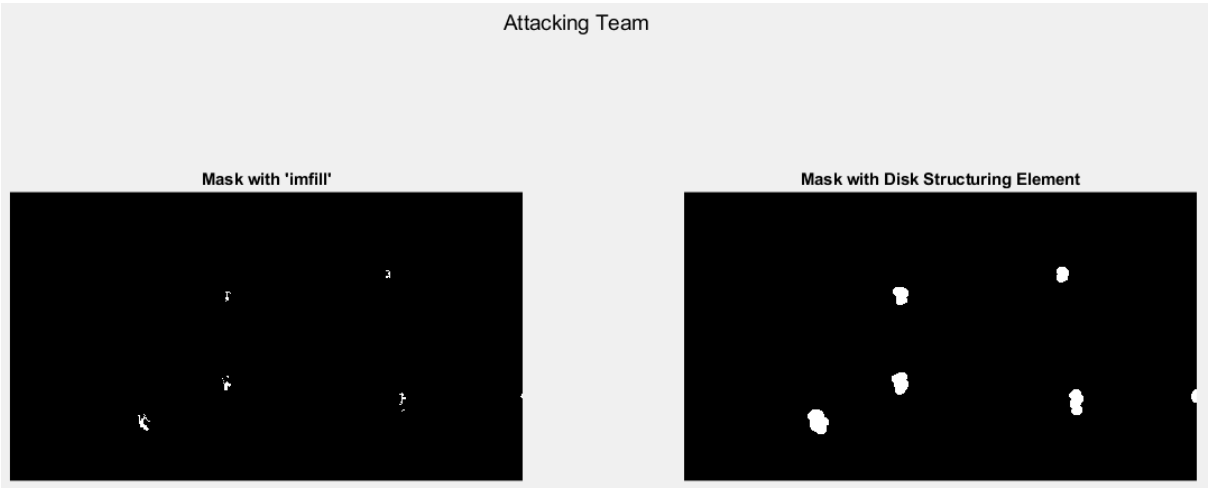


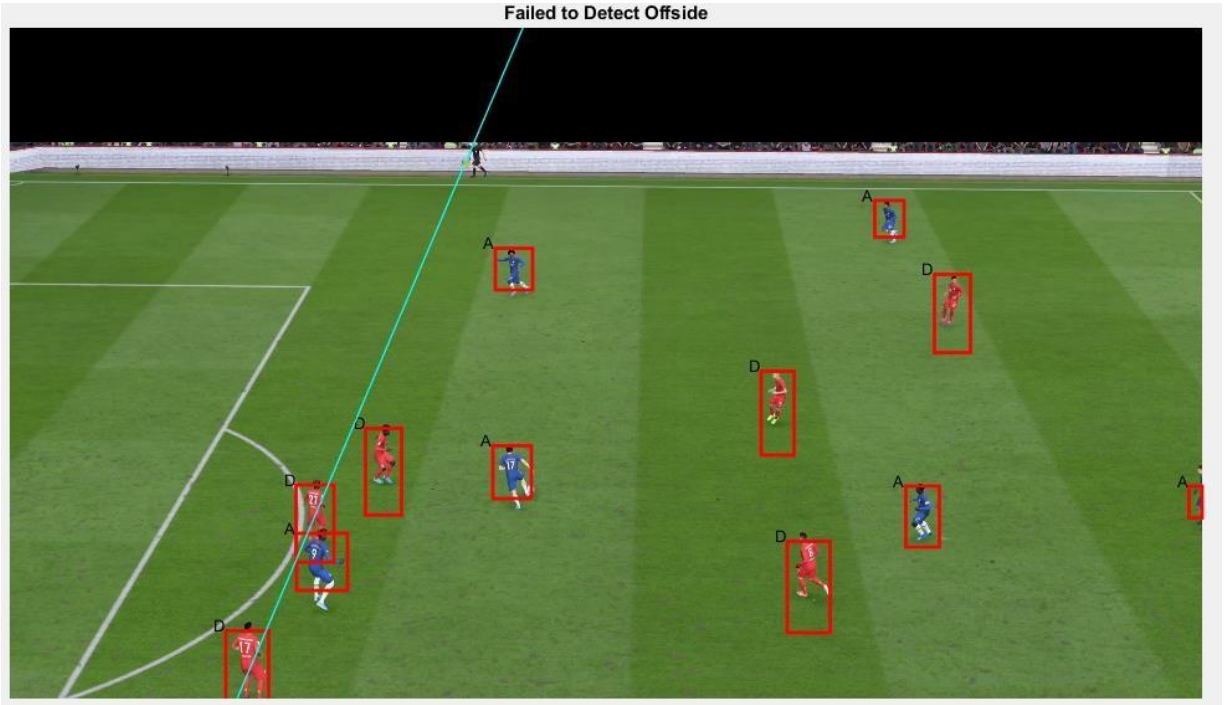




(Figure 28: Test 28 Correct)

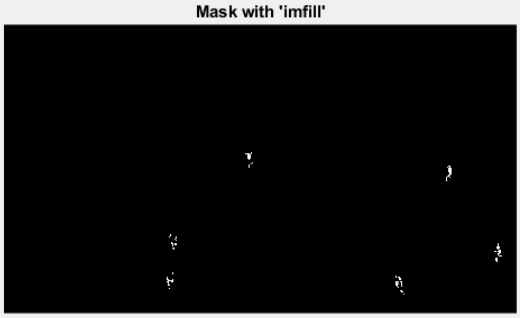


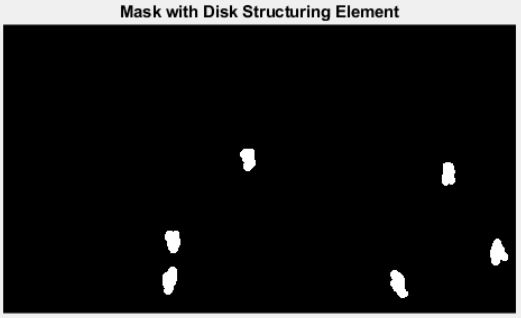




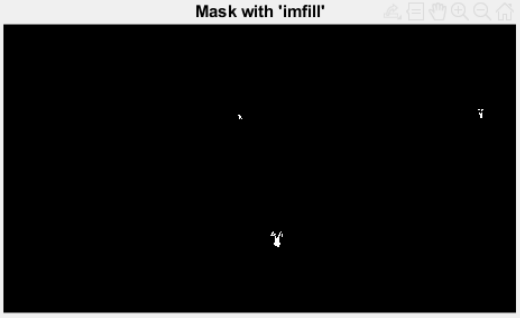
(Figure 29: Test 29 Correct)

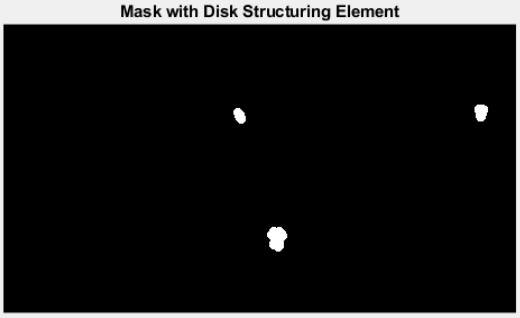
Defending Team

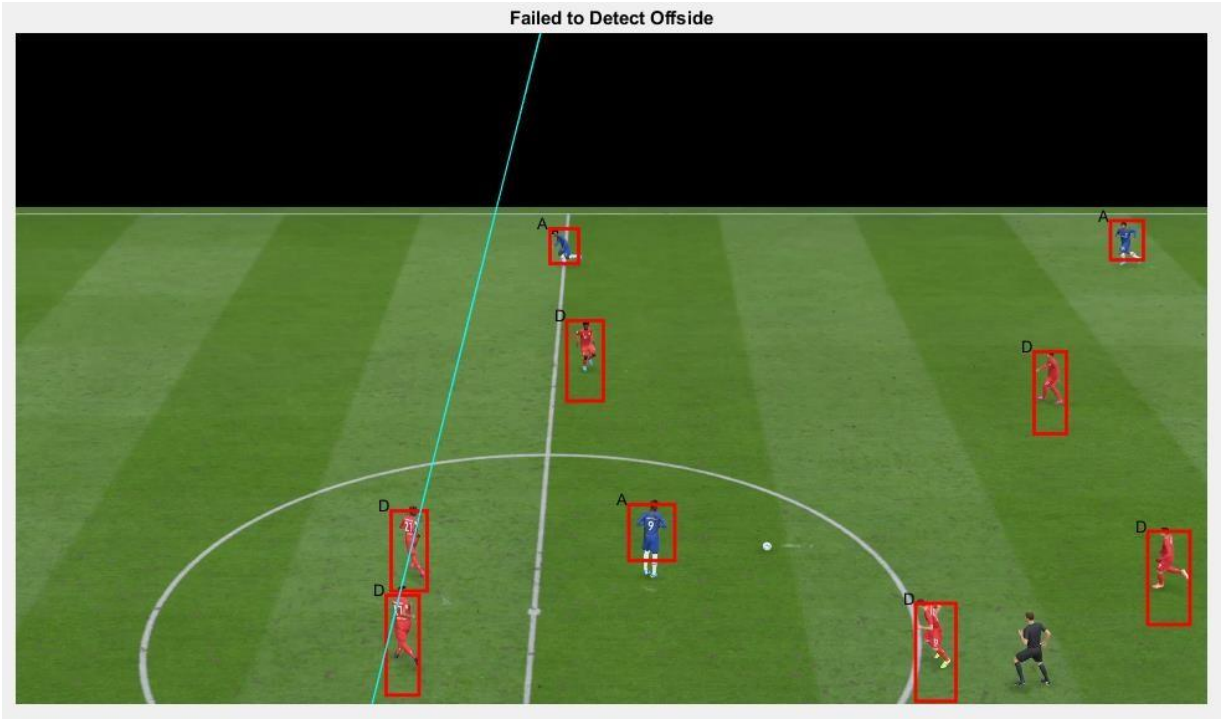




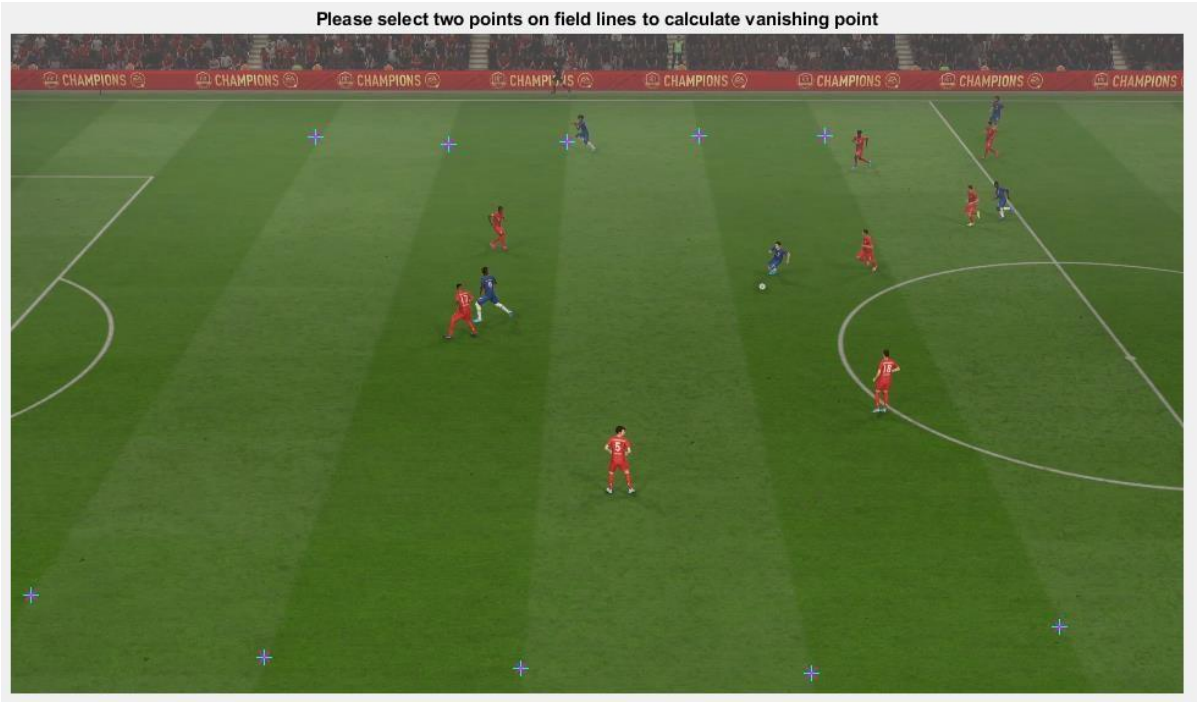
Attacking Team

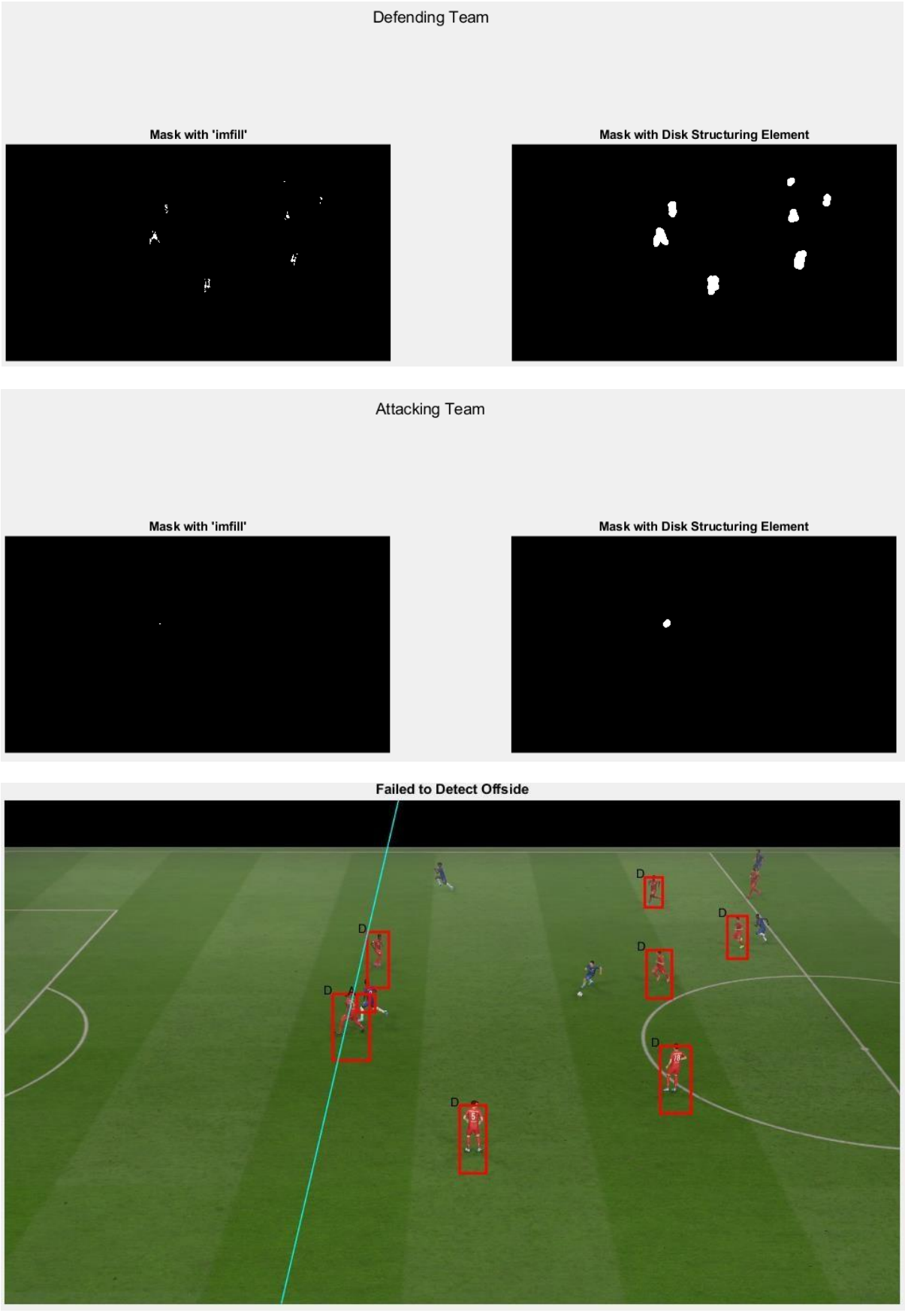






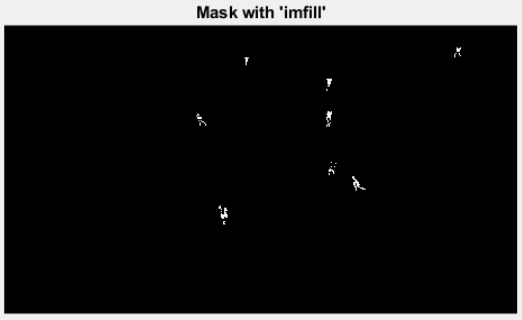
(Figure 30: Test 30 Correct)

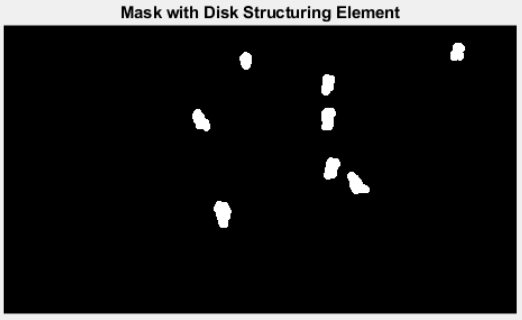




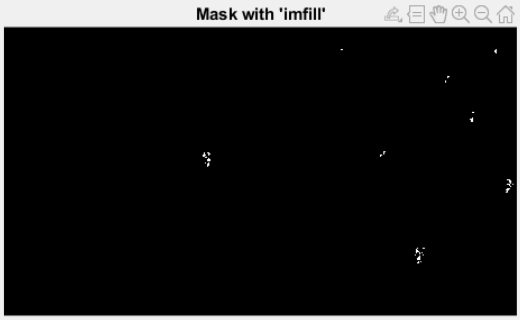
(Figure 31: Test 31 Correct)

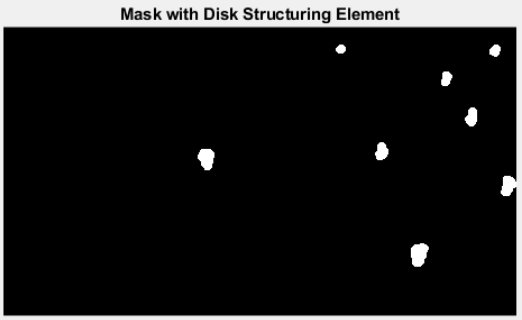
Defending Team

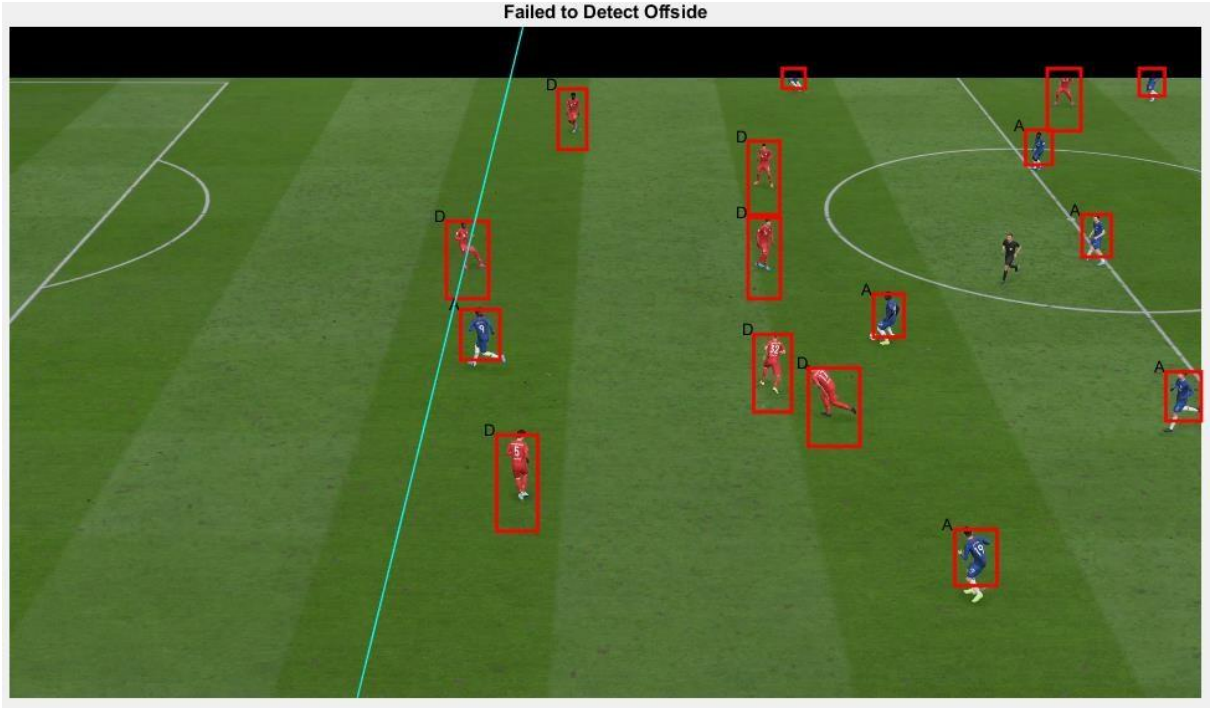




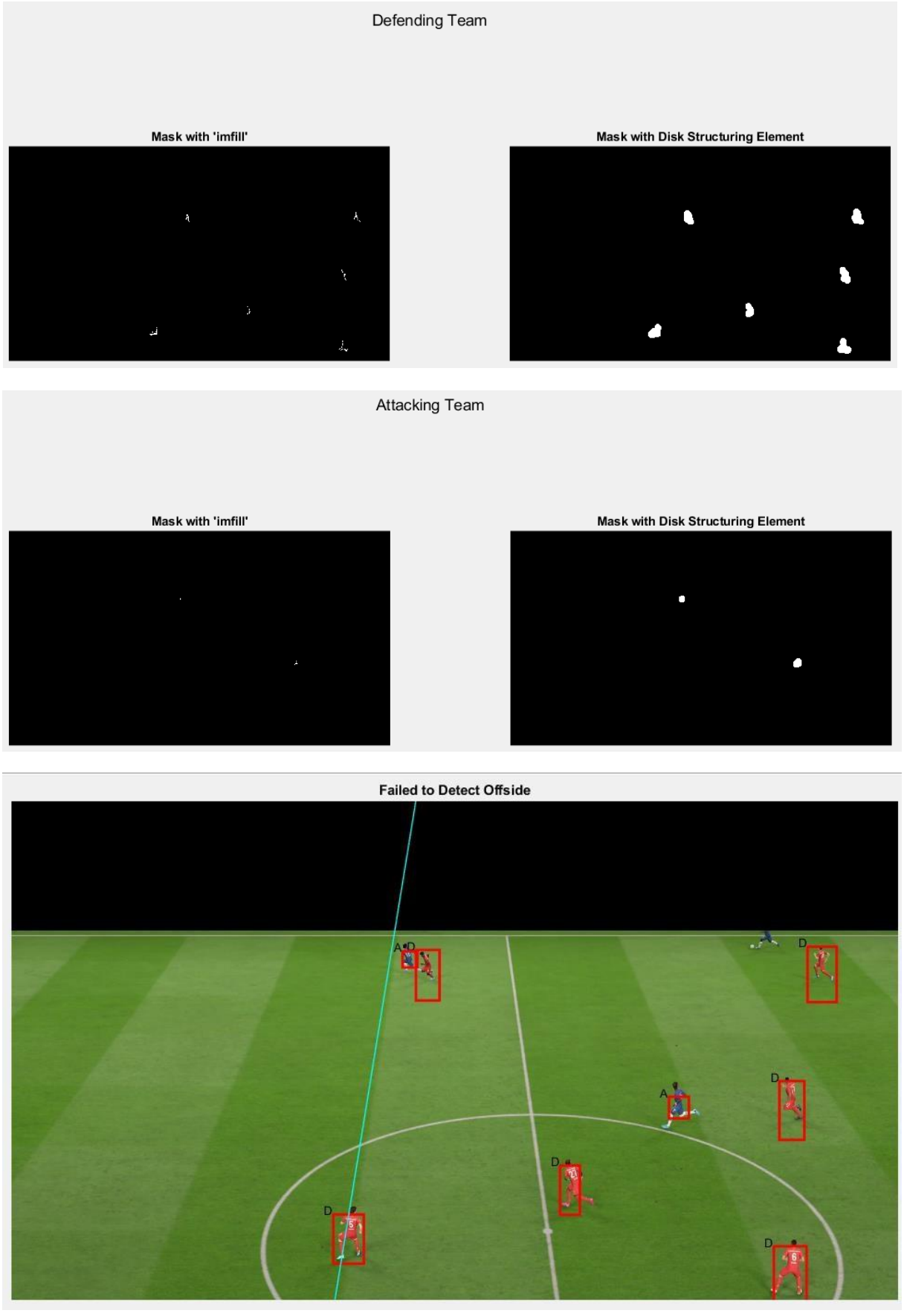
Attacking Team







(Figure 32: Test 32 Correct)



(Figure 33: Test 33 Correct)

Discussion

In total, there are thirty-three different football positions shot from different angles to test our designed offside detection algorithm and analyze the effect of the number of selected points for vanishing point, the tribune, the number of the players in the field. Also, each position is considered as if the ball is coming out the foot right now. The results are organized in such a way that there are outputs of same position image from point selection part, color segmentation steps and detection with offside line, respectively for each position.

The number of selected points becomes very critical while the decision of the corresponding position cannot be finalized obviously. In other words, it really influences the course of the decision when two opposite players are very close each other around the offside line. The small shifts may result in a wrong offside decision. Therefore, it is important to increase the number of selected points as much as possible and as accurate as possible. However, it is also necessary to be careful not to pick the points very close to the edges of the image where the lines may increase the margin of error in vanishing point reconstruction as seen in Figure 26. The tribune affects the directly result of the decision by deflecting the offside line and doing misdetection the players. In this regard, we have developed a method based on Hough Transform to reduce its negative effects on our decisions. Thus, this method managed to eliminate this negative effect in 16 out 20 positions, working with 80% accuracy. While the method is completely failed in Figure 12, 17 and 21, we see that the method lacks discarding the tribune behind the goal in Figure 17, 19 and 21. On the other hand, it is difficult to say clear statement about the direct or indirect effect of the number of the players in the field on the result. Because, the algorithm works well or does not work correctly regardless of the number of the players in the field.

Tests on Figure 3 and Figure 9 show the importance of masking both with “imfill” and “disk”. Initially, Imfill identifies the attacking team players and defending team players. Then, disk structuring element emphasizes the locations of corresponding player locations by magnifying the detected areas with white‘disks’. Figure 12 and Figure 17 show the importance and necessity of shadowing tribune areas, in order to prohibit incorrect detection of non-player objects as players. All the tests, where tribunes are discarded from the player detection phase, gave correct outputs. Figure 21 shows the importance of discarding non-player objects from the scene to have tangible results. In this test case, red flags and advertisement panels are classified as a defense player which leads an incorrect placement of the offside line. Actually, according to the rulebook, all the defense players behind the goalkeeper are not considered into account for the decision. This edge case is not covered in our scope, and in all our cases, it is assumed that there is not any defense player behind the goalkeeper. In Figure 26, all visible field lines are chosen in order to calculate the vanishing point. Intuitively, this selection seemed to lead the most accurate vanishing point to draw an offside line; however, it turns out that abusing the selection

of field lines may deteriorate the drawing of offside line. In Figure 26, the resultant offside line is not correct, even though the algorithm did not decide an offside in this position. The offside line should be parallel to the field lines, with respect to the perspective projection of the camera.

Accuracy	# of positions for defensive player	# of positions for attacking player
acc = %100	19	15
%100< $acc \leq$ %80	7	1
%80< $acc \leq$ %60	3	4
%60< $acc \leq$ %40	0	7
%40< acc	0	2

Table 1: It shows how detection of the players works well using color segmentation based on the positions except the cases resulted in wrong decision due to the tribune.

Based on all football position displayed in the Result section, our offside detection algorithm works with 81.81% accuracy. The reason for two of the wrong decisions is the failure of the detection of an attacking player while the reason for four of the wrong decisions is the failure of the eliminate of the tribune deceptive effect on the algorithm. On the other hand, Table-1 shows that the number of the positions where the identification of defensive players results in high odds makes up most of the cases. One of the reasons that we can observe in the results section is that two defensive players are perceived as a single defensive player due to the mixing with each other during the preprocessing stage. Although the number of the positions where the identification of defensive players results in high odds is not as much as the positions for defensive player, their numbers are also pretty enough. According to the table, there are 9 positions with having less than 60% detection accuracy although the total number of positions resulted in wrong decision are 2. It demonstrates that undetectable players are those who are far from the main issue, that is, the office line, in many cases. It is a good inference since our main problem is to detect whether the position is offside or not.

The proposed technique only solves the offside problem for one attacking direction. It may easily be extended to a technique for both attacking direction. On the other hand, elimination of the effect of the tribune behind the goalpost can be studied on; or the human pose estimation can be studied to get rid of our assumption of projecting the human body to a fixed point on the bounding boxes for offside line and come up with a more serious solution as a future work.

Appendix

A) Implementation of Offside Detection Algorithm

offside_detection.m

```
%% Initialization
clear
close all
clc

%% Read Video
obj = VideoReader('35.mp4'); % create video object
start_frame = 1;            % start frame
offside_frame = 3;          % frame at which the pass is made

for frame_index = 1:4       % parsing through video frames
    if(exist('img'))
        prev_img = img;
    end
    img = readFrame(obj);

    if(frame_index<start_frame)
        continue
    end

%% Getting Vanishing Point lines
    % Requesting parallel vanishing lines input from user
    if (frame_index == start_frame)
        imshow(img)
        title("Please select two points on field lines to calculate vanishing point");
        [x,y] = getpts;
        points = [x,y];
        close all
        %tStart = cputime;
    % Calculating Slope
    m = zeros(size(points,1)/2,1);          % slope matrix
    c = zeros(size(points,1)/2,1);          % intercept matrix
    k = 1;
    vp = zeros(2,1);                        % vanishing point matrix
    for j = 1:2:size(points,1)
        m(k) = (points(j+1,2) - points(j,2)) / (points(j+1,1) - points(j,1));
        c(k) = - points(j,1) * m(k) + points(j,2);
        k = k+1;
    end
    % Calculating the Vanishing point
    count = 0;
    for p = 1:size(points,1)/2
       for q = p+1:size(points,1)/2
           count = count + 1;
           A = [-m(p),1;-m(q),1];
           b = [c(p);c(q)];
           vp = vp + A\b;
       end
    end
    vp = int32(vp/count);
    end

%% Actual Detection starts (one every 20 frames).
    BW_img = rgb2gray(img);                 % Coverting the image to grayscale
    Edge_img = edge(BW_img,'sobel');        % Converting greyscale image to edge image using Sobel

% Removing the TOP Boundary using Hough Transform
%   Defining Hough Parameters
    start_angle = 89;
    end_angle = 89.99;
    theta_resolution = 0.01;
%   Obtaining Hough coefficients
    [hou,theta,rho] = hough(Edge_img(1:floor(size(Edge_img,1)/2),:), 'Theta', start_angle:theta_resolution:end_angle);
    peaks = houghpeaks(hou,2,'threshold',ceil(0.3*max(hou(:))));
    lines = houghlines(Edge_img(1:floor(size(Edge_img,1)/2),:),theta,rho,peaks,'FillGap',5,'MinLength',7);
%   Identifying longest horizontal lines
    min_row = lines(1).point1(2);
    xy_long = [lines(1).point1; lines(1).point2];
```

B) Implementation of Offside Detection Algorithm cont'd

```
    max_len = 0;
    sizes = size(img);
    for k = 1:length(lines)
       xy = [lines(k).point1; lines(k).point2];

       % Determine the endpoints of the longest line segment
       len = abs(lines(k).point1(1) - lines(k).point2(1));
       if ( len > max_len)
          max_len = len;
          xy_long = xy;
       end
    end
    if(xy_long(2,1)-xy_long(1,1) > 70)
    %   Removing top boundary pixels
        img(1:xy_long(:,2)-10,:,:)=0;
        BW_img(1:xy_long(:,2)-10,:,:)=0;
        Edge_img(1:xy_long(:,2)-10,:,:)=0;
    end

%

%% Determining the players and Team_Ids
    img_valid = img;
%   Define defending team colours
    indg = find(fuzzycolor(im2double(img_valid),'red')<0.1);
    n = size(img,1)*size(img,2);

    img_team_read = img_valid;
    img_team_read([indg;indg+n;indg+2*n]) = 0;

%   Image processing
    mask = imbinarize(rgb2gray(img_team_read));
    mask = imfill(mask,'holes');
    mask_open = bwareaopen(mask,30);
    mask_open = imfill(mask_open,'holes');
    if(frame_index == offside_frame)
        figure; sgtitle('Defending Team'); subplot(1,2,1);imshow(mask_open);title("Mask with 'imfill'");
    end
    S_E_D = strel('disk',15);
    mask_open = imdilate(mask_open,S_E_D);
    Conn_Comp_team_red = bwconncomp(mask_open,8);
    S_team_red = regionprops(Conn_Comp_team_red,'BoundingBox','Area');
    if(frame_index == offside_frame)
       subplot(1,2,2); imshow(mask_open);title("Mask with Disk Structuring Element");
    end

%   Define attacking team colours
    indg = find(fuzzycolor(im2double(img_valid),'blue')<0.1);
    n = size(img,1)*size(img,2);
    img_team_read = img_valid;
    img_team_read([indg;indg+n;indg+2*n]) = 0;
    mask = imbinarize(rgb2gray(img_team_read));
    mask = imfill(mask,'holes');
    mask_open = bwareaopen(mask,15);
    mask_open = imfill(mask_open,'holes');
    if(frame_index == offside_frame)
        figure; sgtitle('Attacking Team'); subplot(1,2,1);imshow(mask_open);title("Mask with 'imfill'");
    end
    S_E_D = strel('disk',15);
    mask_open = imdilate(mask_open,S_E_D); % dilate (genişletmek) the binary image with disk structuring element
    Conn_Comp_team_blue = bwconncomp(mask_open,8);
    S_team_blue = regionprops(Conn_Comp_team_blue,'BoundingBox','Area');
    if(frame_index == offside_frame)
         subplot(1,2,2);imshow(mask_open);title("Mask with Disk Structuring Element");
    end
```

C) Implementation of Offside Detection Algorithm cont'd

```
135
136 %   Getting all players/teamIds in one list
137     S = [S_team_red; S_team_blue];
138     Team_Ids = [ones(size(S_team_red,1),1); 2*ones(size(S_team_blue,1),1)]; % Mark defense team as 1 and attacker team as 2
139     Players = cat(2,[vertcat(S(1:size(S,1)).BoundingBox)], Team_Ids); % Concatenate team_ids with their corresponding teams
140
141     %% Mark the bounding boxes
142     f = figure('visible','off');
143     left_most = 9999;
144     team_index = 5;
145
146     if(frame_index == offside_frame)
147         figure; imshow(img)
148         hold on;
149     end
150
151     for i =1:size(S,1)                  % For all detected players
152         BB = S(i).BoundingBox;          % Get their bounding box
153         if(Team_Ids(i)==1)              % If defense team
154             text(BB(1)-20, BB(2)-10,'D'); % Display D above its bounding box
155             BB(4)  = 1.5*BB(4);
156             S(i).BoundingBox(4) = BB(4);  % Bounding boxes of defense team is larger (?)
157         end
158         if(Team_Ids(i)==2)              % If attack team
159             text(BB(1)-20, BB(2)-10,'A'); % Display A above its bounding box
160         end
161         rectangle('Position',[BB(1),BB(2),BB(3),BB(4)],...
162         'LineWidth',2,'EdgeColor','red')  % Draw a rectangle to display bounding box
163
164         x1 = floor(BB(1)+BB(3)/4);      % Coordinates of bounding box
165         y1 = floor(BB(2) + BB(4));
166         ly = size(img,1);
167         slope = int32((vp(2) - y1)/(vp(1) - x1));% Slope of the line from vanishing point to bounding box
168         y_int = int32(- x1 * slope + y1);
169         lx = int32((ly - y_int)/slope);         % Left x coordinate of the player
170         if(lx<left_most && Team_Ids(i) == 1) % Determine x coord of the leftmost defender
171             left_most = lx;
172             slope_last_def = slope;
173             y_int_last_def = y_int;
174         end
175     end
176
177     offside = 0;
178     if(frame_index == offside_frame)
179         m = (ly - vp(2))/(left_most-vp(1));
180         c = vp(2) - m*vp(1);
181         for z = 1:size(Players,1)
182             if(Players(z,team_index) == 2)
183                 x = Players(z,1) + Players(z,3)/4;
184                 y = Players(z,2) + Players(z,4);
185                 if(m*x + c > y)
186                     title("Offside detected")
187                     offside=1;
188                     break
189                 end
190             end
191         end
192         if offside == 0
193             title("Failed to Detect Offside")
194         end
195     end
196
197 %   Plot offside line
198     plot([left_most,vp(1)],[ly ,vp(2)],'c','LineWidth',1)
199  %{ ...
245 end
```

D) Fuzzy Color Function

fuzzycolor.m

```
function [iscolor,colornames]=fuzzycolor(RGB,colorquery)
% fuzzycolor: identify a color name for any RGB color
% usage: [iscolor,colornames]=fuzzycolor(RGB)
% usage: [iscolor,colornames]=fuzzycolor(RGB,colorquery)
%
% Note: because fuzzycolor must initially load a large
% (persistent) dataset, the first call to this function
% will take extra time. Successive calls will be faster.
% If you change the database file, make sure you issue a
% "clear functions" command before calling it again.
%
% arguments: (input)
%  RGB - [1x3] vector, or nx3 array, or nxmx3 array
%       of colors. Each color in this list will be tested
%       against the potential colornames specified by
%       colorquery.
%
%       Note: RGB is assume to be scaled as an RGB intensity,
%       in the interval [0,1]. It can also be a uint8 array,
%       scaled as [0,255].
%
%  colorquery - (OPTIONAL) - allows you to specify which
%       colorname to test for. colorquery must be either
%       empty, the string 'all', or a character string
%       containing the name of a single colorname from this
%       list of colornames:
%
%       'red' 'green' 'blue' 'neutral' 'pastel' 'yellow'
%       'flesh' 'cyan' 'magenta' 'black' 'white' 'purple'
%       'brown'
%
%       DEFAULT: colorquery = 'all'
%
%       If 'all' is specified, then all colors on the
%       list will be tested for, for each row of RGB.
%
%       Any single color name may be specified, or any
%       unambiguous contraction of that name. 'Thus 'g',
%       'gr', 'gre', etc., are all valid contractions for
%       'green'.
%
% arguments: (output)
%  iscolor - double array, indicating if I think that
%       each given color triad is the color in question.
%
%       == 0 --> it does not appear to be the color in question
%       == 1 --> it does appear to be the color in question
%       0<iscolor<1 --> means this color was right on the edge.
%
%  colornames - cell array of the color names tested for
%
% Example:
%  What color is [1 1 0.1]?
%  [iscolor,cn]=fuzzycolor([1 1 0.1])
%
% iscolor =
%    0  0  0  0  0  1  0  0  0  0  0  0  0
%
% cn =
%   Columns 1 through 8
%    'red' 'green' 'blue' 'neutral' 'pastel' 'yellow' 'flesh' 'cyan'
%   Columns 9 through 13
%     'magenta' 'black' 'white' 'purple' 'brown'
%
%  Fuzzycolor correctly predicts that [1 1 0.1] is a yellow.
%
% Example:
%  Is [1 1 0.1] a red?
%  iscolor = fuzzycolor([1 1 0.1],'red')
%
%  ans =
%      0
%
%  Fuzzycolor correctly predicts that [1 1 0.1] is not red.
%
% Author: John D'Errico
% e-mail: woodchips@rochester.rr.com
% Release: 1.0
% Release date: 9/18/06

```

E) Fuzzy Color Function cont'd

```
% have we loaded in the fuzzycolordata array yet?
persistent FuzzyColorData
if isempty(FuzzyColorData)
  % not yet loaded.
  load FuzzyColorData
end
ncolors = FuzzyColorData.ncolors;

% default for colorquery?
if (nargin<2) || isempty(colorquery)
  colorquery = 'all';
  colorind = 1:ncolors;
else
  % which color name was requested?
  colorquery = lower(colorquery);
  if strcmp(colorquery,'all')
    % all was requested, so
    colorind = 1:ncolors;
  else
    % must have been a color name
    colornames = FuzzyColorData.colornames;
    colorind = find(strncmp(colorquery,colornames,length(colorquery)));
    if isempty(colorind)
      error 'colorquery is not a match for any color name in the database'
    elseif length(colorind)>1
      error 'colorquery is an ambiguous color name'
    end
  end
end

% verify that RGB is a valid color or set of colors
if mod(numel(RGB),3)~=0
  error 'RGB array must be a 1x3 vector, a nx3 or an nxmx3 array'
end
if isa(RGB,'uint8')
  RGB = double(RGB)/255;
else
  if (max(RGB(:))<0) || (min(RGB(:))>1)
    error 'RGB array does not appear to be scaled as [0,1] intensity'
  end
end

RGB = reshape(RGB,[],3);
np = size(RGB,1);

% initialize the result array
iscolor = zeros(np,length(colorind));

for i = 1:length(colorind)
  iscolor(:,i) = interp3(FuzzyColorData.rnodes, ...
     FuzzyColorData.gnodes,FuzzyColorData.bnodes,...
     FuzzyColorData.colorlut{colorind(i)},RGB(:,1),RGB(:,2),RGB(:,3),'linear');
end

% return the list of colonames tested for
colornames = FuzzyColorData.colornames(colorind);
```

F) Fuzzy Color Data

x.FuzzyColorData

Field	Value
ncolors	13
colornames	*1x13 cell*
colorlut	*1x13 cell*
rnodes	*1x52 double*
gnodes	*1x52 double*
bnodes	*1x52 double*

References

[1] Color name identification: fuzzycolor, 2006, https://www.mathworks.com/matlabcentral/fileexchange/12326-color-name-identification-fuzzycolor

[2] Deblom, Axel, DETECTING OFFSIDE POSITION USING 3D RECONSTRUCTION, Lund University, 2020

[3] Ünel, Mustafa, EE 417 Computer Vision Lecture Slides, 2022