IPCV Dartboard Challenge Report

Kamen Bachvarov (kb17034)
Omran Alhaddad (oa17248)

Introduction

In computer science, image processing refers to analyzing and manipulating an image in order to better visualize it or gain useful information from it. As computers get faster and faster, image processing speeds improve in proportion hence it has become a fundamental part of the digital world. A similar field of study is computer vision. Even though it might be simple for human beings to process images, computers need to be trained. This is the goal of computer vision. It helps them deeply understand the content of an image. In this paper we are going to implement dartboard detection by using image processing and computer vision techniques and eventually produce an appropriate output.

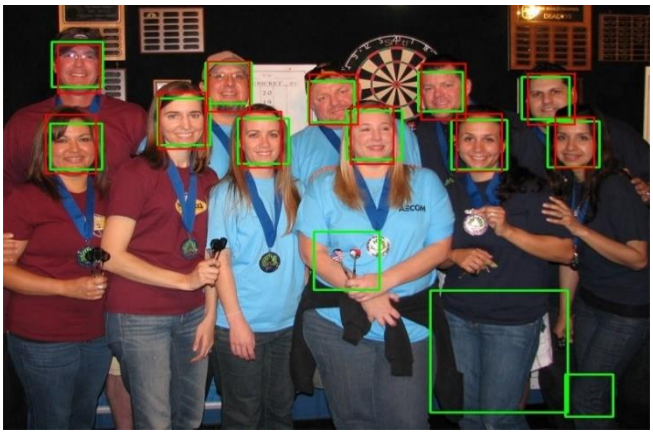
Face detector

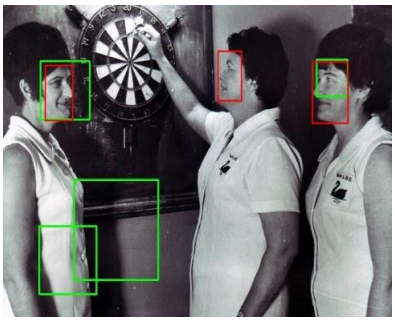
In order to better understand the concept idea of what we are going to implement, we have run a well-trained face detector on the five images below.

Pictures 1-5: Face detections

1 

2 

3 

4 

5 

It can be seen in the examples that the face detector has done his job quite effectively. To confirm that, we had initially made ground truth face detections (the red rectangles), that roughly represent our prediction of where the detected face should be located. After running the face detector, we combined its results (in green) with our predictions.

Furthermore, we have written a small python script that compares a face detection to its prediction respectively. We have run that with 10 of the example images (ones that included faces) with a IoU threshold of 60% (overlap between ground truth and detected box). The results are summarized as follows.

Table 1: Face detection results

Image	Dart0	Dart4	Dart5	Dart6	Dart7
TPR	0	1	1	0	1
IoU	0	0.76	~0.71	0.06	0.66
F1-score	0	1	0.88	0	1
Image	Dart8	Dart11	Dart13	Dart14	Dart15
TPR	0	1	1	1	0
IoU	0	0.72	0.77	~0.75	~0.55
F1-score	0	1	0.66	0.49	0

The detector performs well, detects most faces with decent accuracy, and thus the F1-score is high as well. Calculating the TPR for all the images was impossible though, because some of them had, and rightfully so, no detections (or no true positives). We decided to skip these. However, it is often possible to achieve a TPR of 1, because when we reduce the threshold and therefore precision, it is highly likely that there would be many detected boxes and eventually one will surely be at the correct place to match a ground truth.

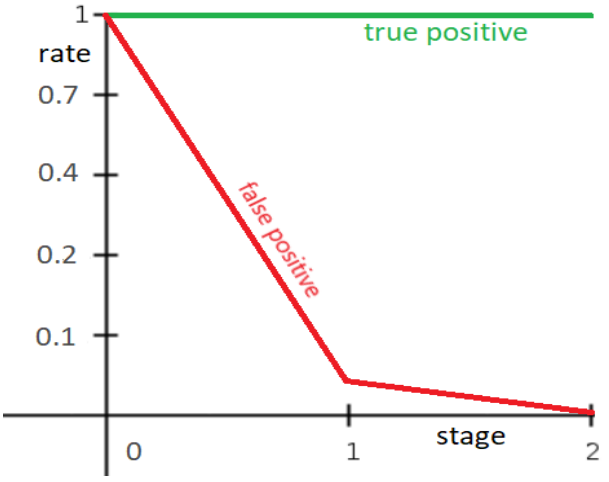
Training boost

Now that we have shown that our face detector can be extremely powerful and efficient, we are going to use similar training methods in order to build and train our own dartboard detector. OpenCV provides us with tools that we use: createsamples and traincascade. The former creates number of random images that contain a dartboard. In addition, we are also given images where there is no dartboard. We are going to supply all of these to the AdaBoost (traincascade). The boosting itself here is a 3-stage procedure where different features are applied in order to generally improve detection efficiency.

Graph 1: Stage 0 training results

In the graph above not only are we showing that the false positive rate drastically reduces from 1 to 0.036 but also that we have successfully maintained our hundred percent true positive rate. What is more, only around 3% of the detections are now false alarms. Having in mind this has only been the initial training stage and we have already dramatically improved, that start has been promising. Below is a plot that consists of the total movements of the TPR and FPR during the stages.

Graph 2: TPR and FPR during training process

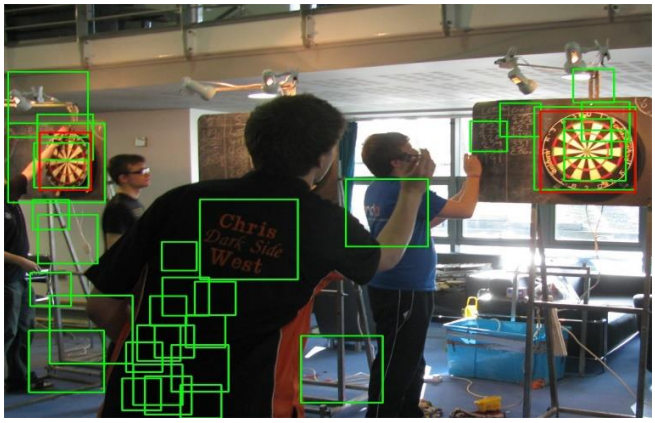


Performance testing

It is now time to put our trained dartboard detector into practice and check how it performs. To do so, we are going to use the same images as before, but this time we should be able to locate some dartboards. Once again, we have placed a bounding box prediction (in red). The three examples we have picked below we think best showcase the detector's abilities.

Pictures 6-8: Trained dartboard detections

6



7

8

These show us that even if the dartboard is different sizes, partly hidden, further into the distance, rotated, affected by light, or any other sort of noise that could have a negative impact on it, the detector would eventually find it. More information in the table below.

Table 2: Trained dartboard performance

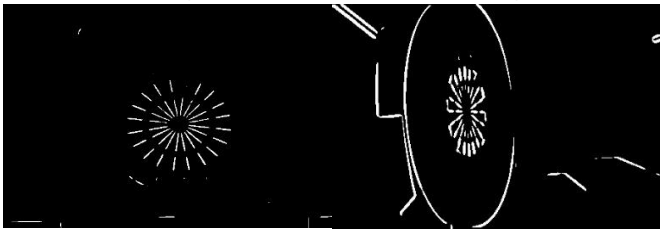
Image	Dart8 (6)	Dart11 (7)	Dart1 (8)
TRP	1	1	1
F1-score	0.06	0.4	0.29

These were appropriate to show because we can clearly see the difference in F1-score This is explained by the high precision of (7) and the low of (6) respectively. The overall TRP for the images remains high – 0.9, however the average value of the F1-score has reduced to 0.19. The detection threshold was the same – 60%. Hence the change was most likely due to the significant decrease in precision after training the detector – that is, many more false positives are apparent on the images (green boxes). So, the next step is to try different Hough transforms.

Line intersection

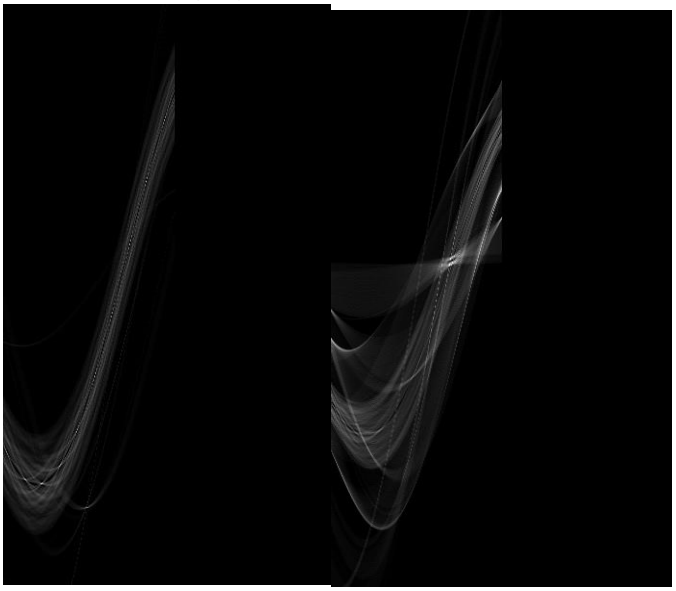
To start with, we are going to implement out Hough transform so that it can detect the intersection points of lines on the image. For this part, we are going to use dart1.jpg and dart12.jpg images. After we convert them to gray scale and apply the Gaussian blur, we build the Sobel operator – that is – computing the derivative on the x direction, the derivative on the y direction, the magnitude of the gradient and the angle of the gradient. We make these accessible at any time. Below are the magnitude images after applying a suitable threshold.

Pictures 1-2: Magnitudes (dart1 left, dart 12 right)



From this point we begin to build our Hough space – 2D for line intersection (all black initially). The idea here is, for each pixel we calculate the rho and theta values. Then, as they represent the Hough space, we increment the pixel at point (rho, thetha). After normalization via threshold, we end up having a sinusoidal figure which represents a straight line on the original image. For the dart1 and dart12 it looks something like this.

Pictures 3-4: Hough spaces (dart1, dart2)



So that now we have the lines, we store the points where they intersect with each other and count the number of those intersections. Previously, when we ran the well-trained Viola-Jones detector, we had some possible detections (bounding boxes) For each of them we get our line intersection points and find the one with the maximum number of intersections at it. Now we are confident enough to make a prediction – it is likely that this point is a center of a dartboard. The corresponding bounding boxes can be seen below.

Pictures 5-6: Dartboard detections vs ground truth

We can see that there are some incorrect detections (or false positives), but that does not affect the successful dartboard detection on these images. There might be many false positives due to the initial boosting performed on the detector via OpenCV as for each of Viola-Jones' dartboard detections, we simply find the likely center (where most lines intersect) and produce a better bounding box from there. The table 1 below provides true positive rate, F1-score (of both) and improvement of F1 on standard Viola-Jones in %.

F1-score comparison

Considering the Viola-Jones' detector has kept its consistency of having TPR of 1, it obviously meant that the precision is low, that is - the number of true positives over all positives is low. Therefore, while applying different thresholds on the Hough space and magnitude, we found the perfect values where the improvement on precision was the biggest. As usual, it resulted in 9% less TPR when compared to original Viola-Jones. However, this trade-off, helped us improved the F1 score (precision) by an average of 245% That is, from initial precision of ~14% to 41%

.

Table 1: Viola-Jones combined with line intersection

Image	Dart0	Dart1	Dart2	Dart3	Dart4	Dart5	Dart6	Dart7	Dart8	Dart9	Dart10	Dart11	Dart12	Dart13	Dart14	Dart15
TRP	1	1	1	1	1	1	1	1	0.5	1	0.66	0.5	1	1	1	1
F1	0.4	1	0.4	0.29	0.66	0.4	0.4	0.13	0.4	0.5	0.2	0.33	0.5	0.2	0.36	0.5
F1(VJ)	0.13	0.29	0.15	0.15	0.08	0.08	0.13	0.04	0.08	0.15	0.1	0.33	0.25	0.11	0.06	0.15
Diff	208	244	166	93	725	400	208	225	400	233	100	0	100	82	500	233

Improvements

- In order to further develop our already enhanced detector we have tried to implement a circle detection and combine them. The idea differentiates from the lines' intersections in the way of creating the Hough space. Here we needed to store a radius as well, hence it detects circles in each image, rather than lines

- Another speedup possibility that was interesting to research on was the OpenCV training process itself. We found that we could change the number of samples and the number of stages. However, it didn't seem to affect the performance on the detector and took a lot of time to train.

Team member: Omran Alhaddad
Contribution: 1

Signature: 

Team member: Kamen Bachvarov
Contribution: 1

Signature: