Computer Vision and Image Processing Semester Project

Darts Recognition Using Computer Vision

Author: **Máté Bene**

MSc in Automation Engineering

University of Bologna, 2025

Contents

List of Figures

Introduction

1.1 Motivation

In the 21st century, some tasks need to be performed quickly and accurately, with minimal human intervention. Image processing/computer vision is increasingly being used to automate tedious or tiring tasks. For example, in ball sports, computer vision is used to track players, measure the distances they have covered, and is often used to analyse human body position to prevent injuries.

Sports are near and dear to my heart and one of my family members plays darts at a serious level, which is how I became interested in this sport.

1.2 Darts

Darts is a sport of English origin, in which small darts are thrown at sectors of varying value on a circular target. Each player starts the game with 501 points, and after each throw the score is deducted from the total score. The player who reaches 0 points first wins, with the last throw landing in a double sector. Such a winning round is called a leg. Players take turns throwing 3-3 darts per round. The value of the throw is determined by the number of sectors hit by the dart. In most tournaments, a winning leg is automatically counted as a winning set. The player winning the predetermined number of sets wins the match.

The standard international darts board is called a clock board, which is 18 inches in diameter. The board can be divided into three main sections: the singles, doubles and triples multiplier areas. Of the two concentric rings, the outer one is the double multiplier sector and the inner one is the triple multiplier sector. In addition to these, the red circle (bullseye) and the green ring (outer bull) in the middle of the board are worth 50 and 25 points respectively, while the other places, are

worth the number of points indicated at the edge of the board (Figure 1.1). The board is placed 1.73 m high on the wall, measured from the centre, and the players shoot from a distance of 2.73 m.

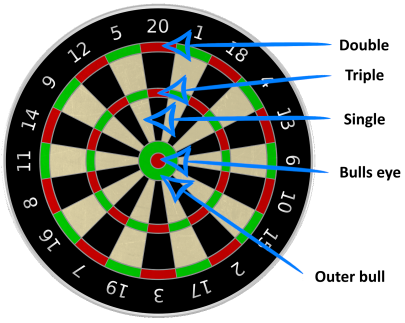

Figure 1.1: Scoring zones

1.3 Calculation of the score

There are a number of methods for calculating the score, from the simplest, counting by hand or keeping track of results on a sheet of paper, or perhaps designating someone to do the counting (on a clipboard in professional competitions) to the most complex, where results are kept via electrical systems (Figure 1.2). Examples of the latter could be the use of various applications, tracking the score by laser or, as in my case, the use of image processing. For me, these solutions are not as practical as counting on paper. For serious players, the electronic board is not a good option, as it is easy for the arrows to pop out, and the feel of the game is completely different. This prompted me to create an application that can determine the score.

Figure 1.2: Electronic board

Dart detection

2.1 Calibration

The camera was placed to the left of the board and a little below it, so as to not be in the way during the game. The goal was to obtain four correspondence points to be able to warp the image as if the camera were straight in front of the board. First I placed four calibration objects (four sticky notes) around the board, whose position can be determined by color detection.

I soon ran into a problem, as the yellow color of the calibration object I used was very similar to the walls in the room, so it was difficult to make a mask of just the sticky notes, so I replaced them with lighter blue ones. I also noticed that if I worked on the project in the morning in sunlight, or in the evening with a lamp on, the minimal change in light had a significant effect on the results. The solution was to perform a calibration before starting the game. During the calibration process, the lower and upper bound of the color you are looking for needs to be set in RGB color coding. In addition to adjusting the color, it is required to position the calibration objects accurately, as the player will get more precise results during the game. After the color detection, if the program finds four contours (the outline of the four objects, Figure 2.1), it connects the center of the objects opposite each other with a line (Figure 2.2), displays the image and the player can check if the two lines intersect exactly at the center of the board, if so, the game can be started.

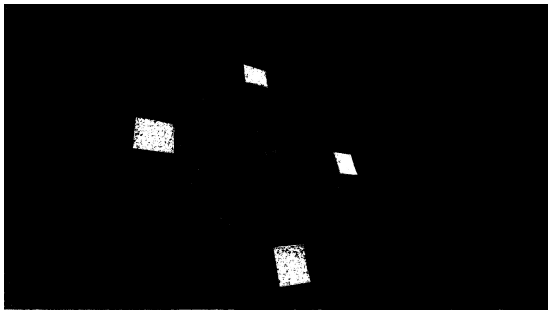

Figure 2.1: Mask of the post its

Figure 2.2: Intersection of the two lines

2.2 Hit detection

The next step is hit detection, which is done by looking at the differences between frames. The previous frame is subtracted from the current frame, resulting in the difference between the images. It is also important to note that sometimes the arrow may hit the board, but the player did not throw it with enough force and it will fall out, so this is not considered a hit. To solve this problem, I introduced a counter that keeps count of the number of frames in which the dart is visible. If this counter reaches 5, only then is it considered a hit. The process of detecting arrows is summarized in Figure 2.3.

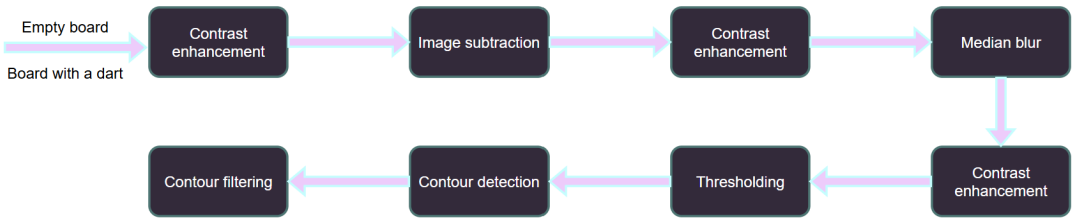

Figure 2.3: Steps of dart detection

2.2.1 Contrast enhancement

Since some parts of the dart are in contact with the black zones of the board, there is minimal difference detectable in those areas after the images are subtracted (since the dart's color is also dark), and this difference is not large enough for thresholding. To solve this problem, I applied a correction, more precisely a contrast enhancement, to both images before subtraction. I applied several methods of contrast enhancement in succession.

Histogram stretching

For contrast stretching, we stretch a narrower part of the intensity range (Figure 2.4) over the full range (Figure 2.5). This allows us to display the narrower range in with higher contrast, i.e. to highlight it.

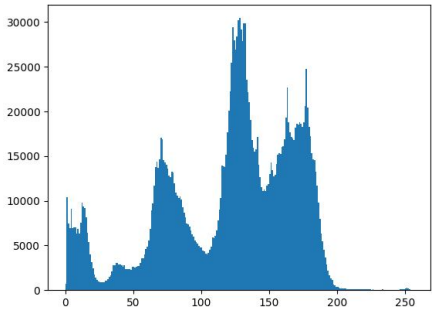

Figure 2.4: Histogram before stretching

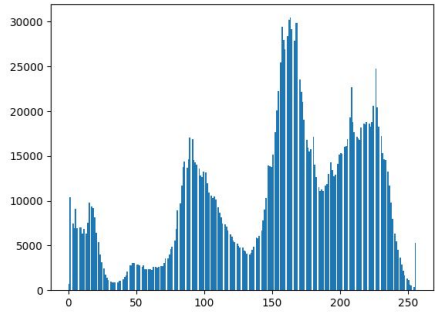

Figure 2.5: Histogram after stretching

Gamma correction

For the first contrast enhancement, I adjusted the darker pixels by stretching the histogram, the pixels with intensities between 20 and 110, and also by gamma correction, where I set gamma to 0.4, which also increases the contrast of the dark pixels. After subtracting the images, I wanted to improve the contrast again, but here I had slightly lighter pixels where the dart was, because this is where the program finds the largest difference between the two images, so I chose a value for gamma larger than 1, to achieve larger contrast in the areas with lighter pixels, which will help during thresholding.

2.2.2 Median blur

After subtracting the two images, the resulting image contains the differences between the two. The goal is to obtain a binary mask of the arrow at the end of the entire detection process. However, after subtraction, it can be seen that a significant amount of noise appears in the image. To reduce the noise, a smoothing operation is needed to smooth out the large variations in intensity. After trying several methods like median and Gaussian blur, I came to the conclusion that the median blur gave me the best results (since most of the noise was impulse or salt-and-pepper noise), especially if I ran it in sequence with several kernel sizes. I noticed that by applying smaller kernels first, the noise was progressively reduced while more image details were retained at each step. In addition it proved to give better results for less uniform noise.

2.2.3 Thresholding

After filtering, a binary image is created from the grayscale output image by thresholding. Since the image has a clear separation between foreground and background intensity levels Otsu's thresholding provided great results.

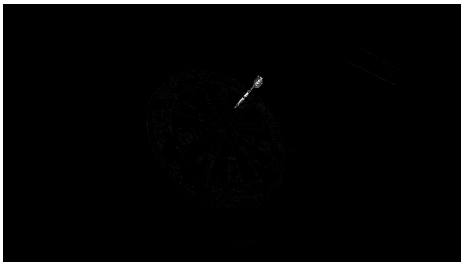

Figure 2.6: Original image

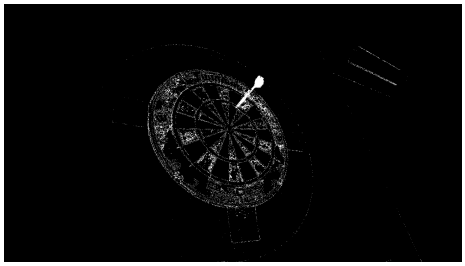

Figure 2.7: Image after Otsu thresholding

2.2.4 Contour detection and filtering

After thresholding, I applied contour detection to find the dart. Since the camera is positioned to the side of the board, some parts of the dart will intersect with the black zones of the board, in these areas the algorithm may find several smaller contours. This occurs because in certain parts of the dart the colour is similar to the black sector on the board. In this case, the contour of the dart is composed of several smaller parts. Since it is possible that, despite all these steps, a small amount of noise still appears in the image, I performed a filter on the contours as a final operation to detect the arrow and remove the contours of negligible size (i.e. the noise) from the resulting mask.

Based on Figure 2.3, I would like to illustrate the steps of the process in practice:

Figure 2.8: Original image of the board with a hit

Figure 2.9: Matching image after contrast enhancement

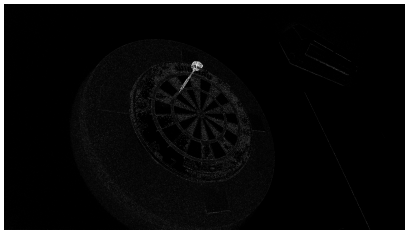

Figure 2.10: Difference between the two images

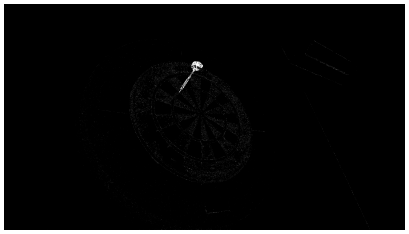

Figure 2.11: Difference image after contrast enhancement

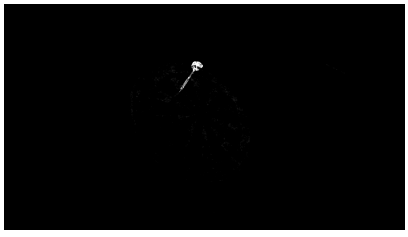

Figure 2.12: Median blur

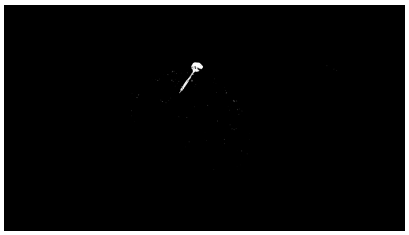

Figure 2.13: Contrast enhancement after median blur

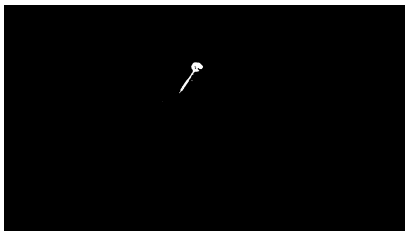

Figure 2.14: Thresholding

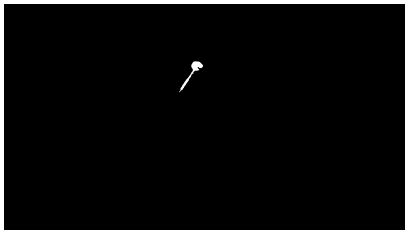

Figure 2.15: Filtering of the contours

Calculation of the score

According to my original plan, the score would have been calculated using color detection. The image would have been segmented according to zones (single, double, triple multiplier zones), and then using Canny edge detection, the point sectors would have been identified by finding the metal separators on the board. In some cases, the metal separators shone so brightly that the algorithm detected a much smaller area than it should have when searching for triple and double sectors, which significantly reduced the hit rate. In Figure 3.1, the areas circled in red show that the red colour was not detected successfully in that area, where it was detected as white instead of red due to the significant glare. This error appears to be minor, but it significantly impaired the accuracy of the score, so a different solution was needed.

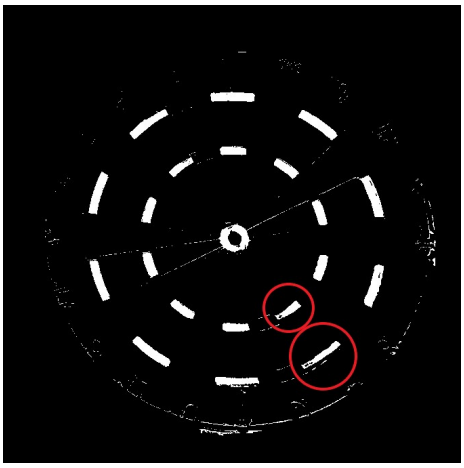

Figure 3.1: Glare caused incorrect color detection

3.1 Detection of the tip of the dart

For more serious darts boards and darts, I have found that the arrow will either go firmly into the board or fall off it, so if the algorithm detects a hit, the arrow will most likely be in the same position. Since the camera is looking at the board from the left, bottom angle, the tip of the arrow is always the pixel of the found contour whose x value is the smallest and y the largest (Figure 4.2).

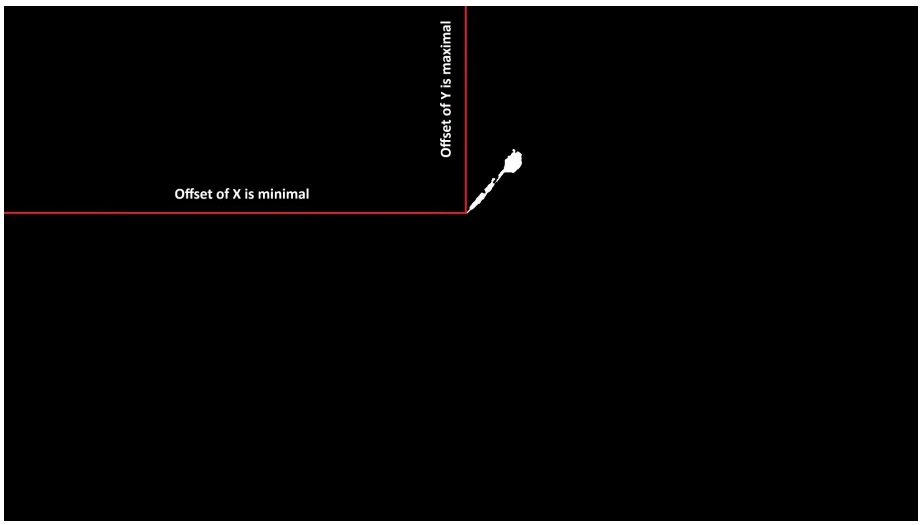

Figure 3.2: Determining the tip of the arrow

3.2 Virtual eye view transformation

At the beginning of the game, the calibration process is necessary to transform the images into a virtual eye view image. The centers of the calibration objects will be used by the algorithm to transform the input image.

Figure 3.3: Board before transformation

Figure 3.4: Board after warping

3.3 Determining the score

After the transformation into a virtual eye view image, the task is to determine the score based on the point and multiplier zones. The multiplier value is calculated based on the distance from the centre. And to calculate the point value, I consider the angle between two lines, the line spanning from the tip of the arrow to the centre of the board, and the horizontal line passing through the centre. So, in reality, I am assigning a polar coordinate to the hit point.

3.3.1 Transformation of the coordinates

Since the coordinates in OpenCV differ from the Cartesian coordinate system, I first transform the coordinate of the hit to simplify the calculation.

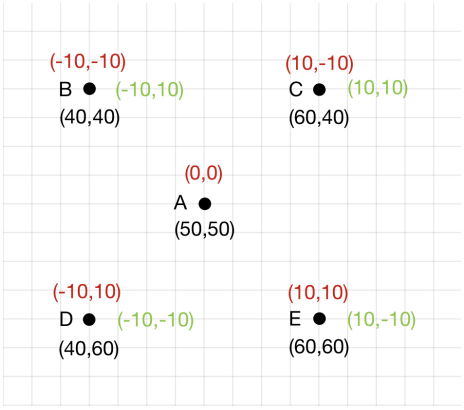

Figure 3.5: Transforming the points

With Figure 3.5, I want to show how I converted the coordinates of the hit point into Cartesian coordinates. Assume that the center of the image is the point $(50, 50)$ and that the hit point is any point marked in black. The goal is to have the center of the image represent the origin, so I subtracted the value of the coordinates of the center (50 in this case) from the value of each coordinate, resulting in the points represented in red. The correct result was obtained by inverting the y value of the coordinates, i.e. multiplying it by -1, and the result was the points in green.

3.3.2 Determining the distance and the angle

Distance

After transforming the points, the distance of the hit point from the centre can be calculated using a simple Pythagoras theorem. The value of the virtual distance is converted to a real centimetre distance. I measured the distance between the centre of the calibration objects with a tape measure, divided by the width of the transformed image. This value determines the ratio between the real and the virtual distance of the image. I then multiplied the calculated distance by this ratio to get the distance in centimetres. Using this distance, the multiplication zones can now be determined.

Distance	Multiplier/point
$d < 0.635$	50 point
$0.635 < d < 1.6$	25 point
$9.9 < d < 10.7$	Triple multiplier
$16.2 < d < 17$	Double multiplier
$17 < d$	Zero multiplier

Figure 3.6: Multiplier zones (d = distance)

Since the bullseye is a full circle and the outer bullseye is a ring, these two scores can be calculated from the distance data alone.

Angle

I simply formed a right triangle, and obtained the cosine of the required angle. As with the distance, I have also prepared a table of possible scores and their corresponding angular ranges (Figure 3.7).

Angle range	Point value
$0 < \alpha \leq 9,\ 351 < \alpha \leq 360$	6
$9 < \alpha \leq 27$	13
$27 < \alpha \leq 45$	4
$45 < \alpha \leq 63$	18
$63 < \alpha \leq 81$	1
$81 < \alpha \leq 99$	20
$99 < \alpha \leq 117$	5
$117 < \alpha \leq 135$	12
$135 < \alpha \leq 153$	9
$153 < \alpha \leq 171$	14
$171 < \alpha \leq 189$	11
$189 < \alpha \leq 207$	8
$207 < \alpha \leq 225$	16
$225 < \alpha \leq 243$	7
$243 < \alpha \leq 261$	19
$261 < \alpha \leq 279$	3
$279 < \alpha \leq 297$	17
$297 < \alpha \leq 315$	2
$315 < \alpha \leq 333$	15
$333 < \alpha \leq 351$	10

Figure 3.7: Angle ranges and their corresponding score (α = angle)

Summary

4.1 Results, conclusions

4.1.1 General findings

I encountered many difficulties during the calibration in terms of detecting the color of the calibration objects. The yellow object was similar to the color of the surrounding wall, making it difficult to correctly identify the center of the object. Even after switching to a darker blue color, I did not get a satisfactory result as the shadow cast by the board interfered with the calibration. The optimal solution was to use a lighter shade of blue, as in this case the color was strongly contrasted with the surrounding colors.

I experimented with a number of methods during hit detection. After subtracting the image of the empty board from the image with a hit, the contrast was found to be too low, so using different types of contrast enhancement proved to be a very effective solution. Initially, I only used contrast enhancement after image extraction, but during my research I found several examples of image correction before image subtraction, which gave very good results. Since the pixels of the image on which I am thresholding have an intensity values that fall into two ranges, Otsu's method was the most effective when thresholding.

As for the point calculation, I initially used color recognition to determine the multiplier zones. I added steps to the calibration process to set the lower and upper bounds of all the colors on the board for efficient color recognition. This method did not give a precise enough solution, so I discarded it. Since I wanted to simplify the scoring process, I tried to reduce the problem to simpler mathematical calculations.

4.1.2 Results

Based on initial testing, a hit rate of 79% out of 200 shots was achieved. In the case of triple and double multiplier zones, a high degree of accuracy is required, as we are talking about very narrow zones. Consequently, there is a need to further increase the precision. In white zones, the hit rate is extremely high, as the arrow is more in contrast to the color of the board. In the red and green (double and triple zones) areas the score is also relatively successful, but the inaccuracy is most pronounced in the black areas, where the darthead blends in significantly, and therefore contrast needs to be increased in several areas. These results are illustrated in Figure 4.1. Very rarely, but there have been instances during daytime gameplay, where sudden changes in lighting conditions (e.g. fast moving clouds) have caused a small amount of noise to appear on the mask of the dart.

Zone	Number of hits	Successful detection	Success rate
White zone	87	79	90.8%
Black zone	71	48	67.6%
Double multiplier zone	11	8	72.7%
Triple multiplier zone	17	11	64.7%
Outer bullseye	3	3	100%
Bullseye	2	1	50%
0 point	9	8	88.9%

Figure 4.1: Hit rates

4.2 Future plans

The next step is to detect multiple darts thrown in sequence. Based on my experience, it is likely that in the future I will have to explore the two-camera option, as it is possible that darts may obscure each other during successive throws. In addition, the precision of the system could be increased by using two cameras.