Rapid and Automated Dice Detection System

Mohandass Muthuraja
Autonomous systems
Hochschule Bonn Rhein Sieg
Bonn, Germany
mohandass.psgrobo@gmail.com

Abstract

The goal of this project is to design an automated dice detection system. This automated dice detection system reports the position of each die being rolled and the values(number of pips) in it. This paper describes the workflow of such a system where the most appropriate image from the video is extracted and then various computer vision techniques and algorithm such as simple blob detection, hierarchical clustering, etc., were used to process the image and detect the dice values and position from a particular region of interest(tray) of the image.

Index Terms

computer vision, dice , blob detection, hierarchical clustering, openCV

I. Introduction

The automatic dice detection system detects the position of each die and reports the value in it. It has numerous uses as a lot of dice games are played everywhere either as a hobby or as gambling. Also, humans take some time to think it over the values of rolled dice. Hence, it is essential to have an automated dice detection system with more reliability and accuracy which automatically detects the position and value of the dice. It is also critical to have such a system to perform the detection as quickly as possible.

Computer vision deals with the operations performed on the digital images to extract useful information from it. Lot of research work has been done in this field and has proved to solve many real-time problems such as automatic inspection, assisting humans in identification tasks, autonomous vehicles, tumor detection, etc., [10]. Recent advancements in computer vision techniques and software computer vision libraries like OpenCV are utilized in solving real-time applications with high computational efficiency. These computer vision techniques are exploited here to detect the values from the rolled dice rapidly.

In this paper, a work-flow for such a system that uses computer vision techniques and algorithms are discussed. A good enough image frame with all the dice after being rolled is extracted from a video, and this image frame is used to localize the dice and determine the number of pips in each die. The entire operations are done in the image frame, and the challenges faced during each operation are explained. Finally, the performance of this system is described based on the experimental evaluation and run-time analysis.

II. Related Work

Haung et al. (2008) developed a machine vision system to determine the value of dice using image processing techniques and a grey clustering algorithm. First, the image was acquired based on the auto-acquisition method, and then the position of the dice is determined after applying various machine vision techniques like binary image conversion, hole filling and morphological operation on the acquired image. At last, the grey clustering algorithm is used to count the number of pips in each die. The author claims the system has good flexibility, high pace, and high efficiency but doesn't explain the performance on the various environmental condition. [1]

Correia et al. (1995) developed a dice detection system for game 'Banca Francesa' that is played in the casino. In contrast to the extracting image from a video, the monochrome CCD camera was used to capture an image after being rolled, and this is used to detect and classify the dice values. [2]

Chung et al. (2009) proposed an image identification scheme that uses features on the image of dice such as the distance between pip, color, size, etc., and the least distance criterion. Images were converted into gray-scale and a binary image (using Ostu method) which is followed by eliminating noise and noncircular patterns in the image. Non - Circular Object Discard stage described by the author eliminates patterns other than circles. Least Distant Segmentation Criterion (LDSC) was used to segment the dice. [3]

Bergant et al. (2000) built an electro-mechanical dice gambling machine for detecting the dice. The authors used an electronic ID on the dice to localize the dice and scanner to read the values on the face of the dice using image processing techniques. [4]

Glenn De Backer, an active computer vision enthusiast, has written code for recognizing dice that first captures a frame from videos. Contour detection on the grayscale image was used to detect the dice. After detecting the dice, thresholding, flood filling and simple blob detection algorithm were performed in sequence to determine the values on each die. [5]

Kyle Schulz, an associate professor through his presentation, has discussed various challenges in detecting dice and methodologies to handle the same in his presentation. [8]

III. Methodology

The work flow of the automated dice detection is shown in the Fig.1

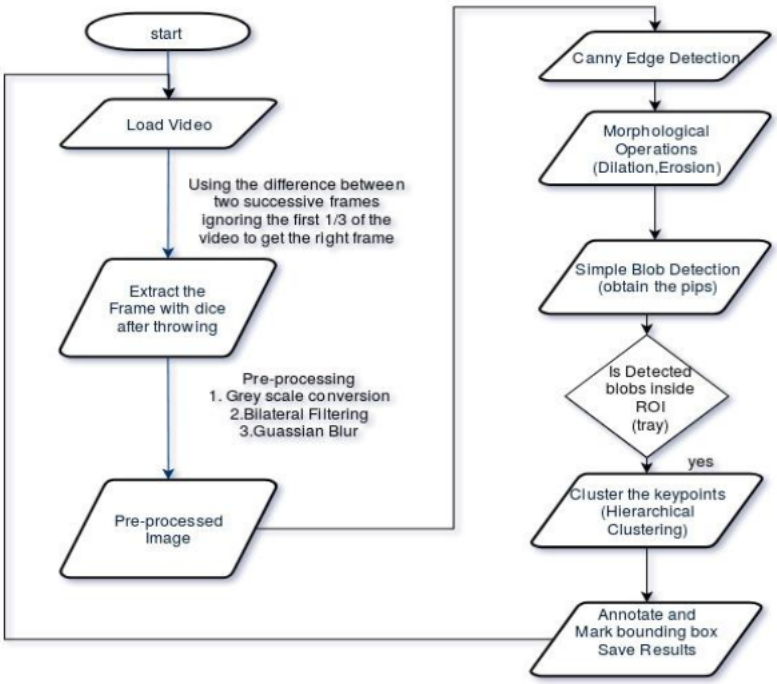


Fig. 1: Work flow of Automated Dice Detection System

The following steps are performed in sequence to localize the die and report the number of pips in it. All the below operation are done using openCV library. [6].

Step 1: First, the videos are loaded from a given directory.

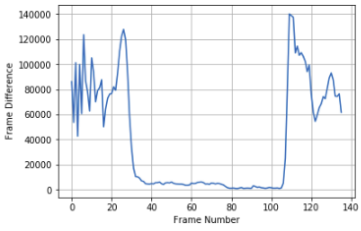
Step 2: The next challenge is to extract a good enough image frame from the video. This is done by using the frame difference between two successful frames.

Step 3: Once the image frame is obtained, it is converted into a grayscale image to perform the preprocessing steps.
Step 4: After preprocessing, edge detection is done using canny edge detection.
Step 5: Canny edge detection is followed by a series of morphological operation to improve the detected edges.
Step 6: Simple Blob detection is then performed to find the all the pips in the image.
Step 7: Unwanted blobs which are exposed outside the region of interest(Tray where the dice are rolled)in the image are then removed. A region of interest is defined based on contour detection and Hough lines.
Step 8: The selected pips are then clustered based on hierarchical clustering algorithm which works based on the Euclidean distance.
Step 9: Clustered centers are used to localize the dice, and the number of pips in each cluster is used to determine the value of each die.

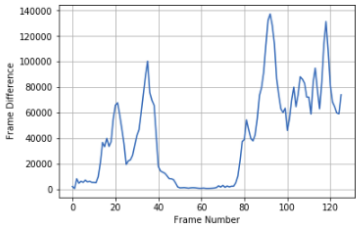
Each of the above steps are discussed in detail below.

A. Frame Extraction

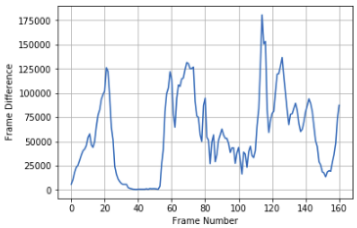
Extracting a suitable frame from the video is very significant as all the other operation can be performed only if the right image frame with all the rolled dice is extracted. Initial $1/3^{rd}$ of the image frames are skipped as it is mostly empty without any dice in it. The difference between two successive frames is used to determine the suitable image frame. Before computing the frame difference, each frame is converted into, and Gaussian blur is performed to remove the noise as it profoundly affects this process. The absolute difference between the successive image frame is normalized and are then summed up to give a scalar value.



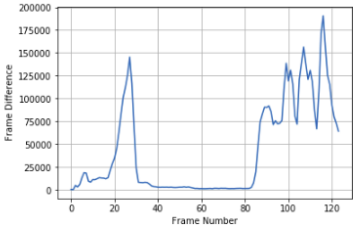
(a) Frame Difference value(Video:2018-10-08@16-17-02.avi)



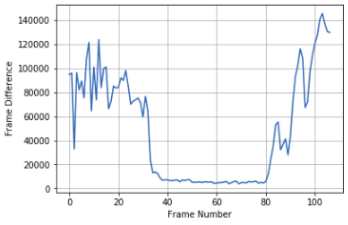
(b) Frame Difference value(Video:2018-10-08@15-41-24.avi)



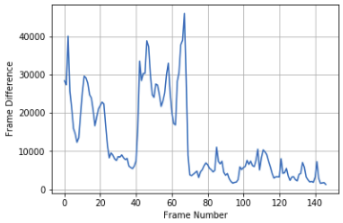
(c) Frame Difference value(Video:2018-10-08@13-38-29.avi)



(d) Frame Difference value(Video:2018-10-08@15-51-36.avi)



(e) Frame Difference value(Video:2018-10-08@15-14-43.avi)



(f) Frame Difference value(Video:2018-10-08@15-00-41.avi)

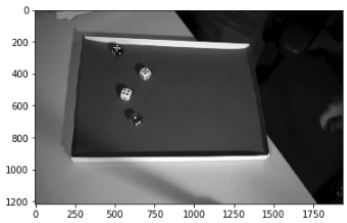
Fig. 2: Frame Difference between successive image frames

Fig. 2 shows the frame difference between successive image frames and it very clear that every video has a plateau region with less frame difference value. By setting a proper threshold value of frame difference, the suitable image frame is easily extracted from the plateau region. Fig 2f shows the result of a noisy image. In this case, the image frame which has very less frame difference is selected. This is experimented in all the provided 40 videos and can get the right frame from all the videos.

B. Preprocessing

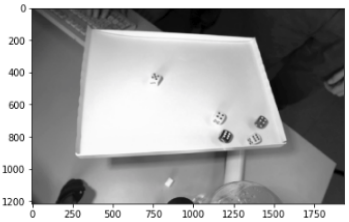
Preprocessing is very vital, and it massively affects the performance. Therefore proper preprocessing need to be done. In this task, the images are initially converted into a gray-scale image. Converting into RGB image to gray-scale image enables to perform complex operations in a shorter time. This is followed by Gaussian blurring of the gray-scale image to remove noise. Proper threshold and kernel size are selected after performing repeated trial and error. Since Gaussian Blur can affect the edges, heavy Gaussian blurring is avoided. Bilateral Filtering of the image is performed next as it removes noise by preserving the edges. Also, Bilateral filters are fast as they are non-iterative. Fig .3 shows the comparison of the original and preprocessed image.

(a) Original image(Video:2018-10-08@16-17-02.avi)



(b) Preprocessed image(Video:2018-10-08@16-17-02.avi)

(c) Original image(Video:2018-10-08@15-41-24.avi)



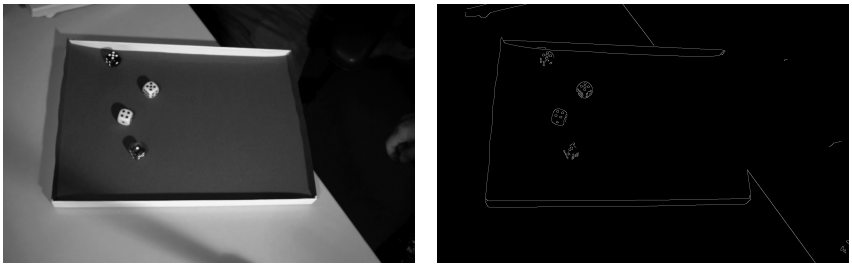
(d) Preprocessed image(Video:2018-10-08@15-41-24.avi)

Fig. 3: Images before and after preprocessing

C. Edge Detection

To obtain the edges, canny edge detection algorithm is used. Canny edge detection algorithm was framed by John F. Canny OpenCV's [6]. Canny edge algorithm is used where images noise is reduced. After noise reduction, the intensity gradient of the image is found. To remove the unwanted pixels, non-maximum suppression is done. At-last, hysteresis thresholding is done to get the real edges. Parameters such as the lower and upper threshold have to be specified for hysteresis thresholding. The parameters values are determined based on the auto-canny algorithm by [8]. In this auto-canny algorithm, the top and the lower threshold for the images are determined based on the median of the pixel intensities of the image. By using auto canny, better edge detection results were obtained than combining strong and weak edges. Fig. 4 shows how a grayscale image is converted after performing auto canny edge detection.

In order to close the holes of pip edge, the morphological operation such as dilation and erosion are performed. The performance of the system is improved by doing the morphological operation.

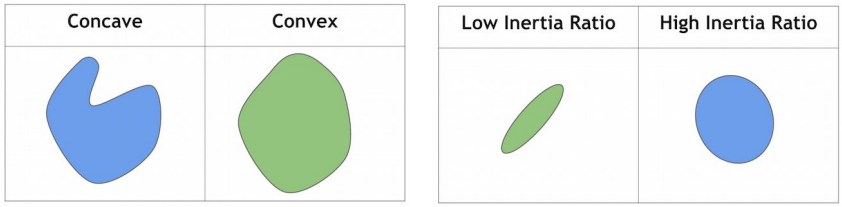


(a) Gray Scale image(Video:2018-10-08@16-17-02.avi) (b) Edge detected image(Video:2018-10-08@16-17-02.avi)

Fig. 4: Auto canny Edge Detection

D. Simple Blob Detection:

As mentioned in [9] "A Blob is a group of connected pixels in an image that shares some common property." OpenCV library SimpleBlobDetector is used to detect the blob. Parameters such as area, circularity, threshold, inertia, and convexity are tuned so that it catches only the pips on the top ignoring other circular blobs in the image. Simple blob detection in openCV primarily converts the image into a binary scale based on the threshold given. Then the contour centers are detected. Contours which have centers very close to each other constitutes a single blob. These blobs then filtered out based on the parameters mentioned before.



(a) Convexity (b) Inertia

Fig. 5: Convexity and Inertia Image from [9]

Fig. 5a shows the difference in an image with low and high convexity. In our task, the pips will have a decent convexity. The right amount of convexity is determined by trial and error. Fig. 5b shows the difference in the image with low and high inertia. This parameter is helpful as the pips on the side have low inertia whereas the pips os the top face of the dice has high inertia.

TABLE I: Simple Blob Detection - Parameters

Parameters	**Min**	**Max**
Threshold	10	230
Area	50	220
Circularity	0.3	1.0
Convexity	0.25	1.0
Inertia	0.5	1.0

Table I shows the parameters which are set after running several times. This has a drawback when the camera position changes a lot, or the images are zoomed-in or zoomed-out. Fig 6 shows the performance of the blob detection algorithm.

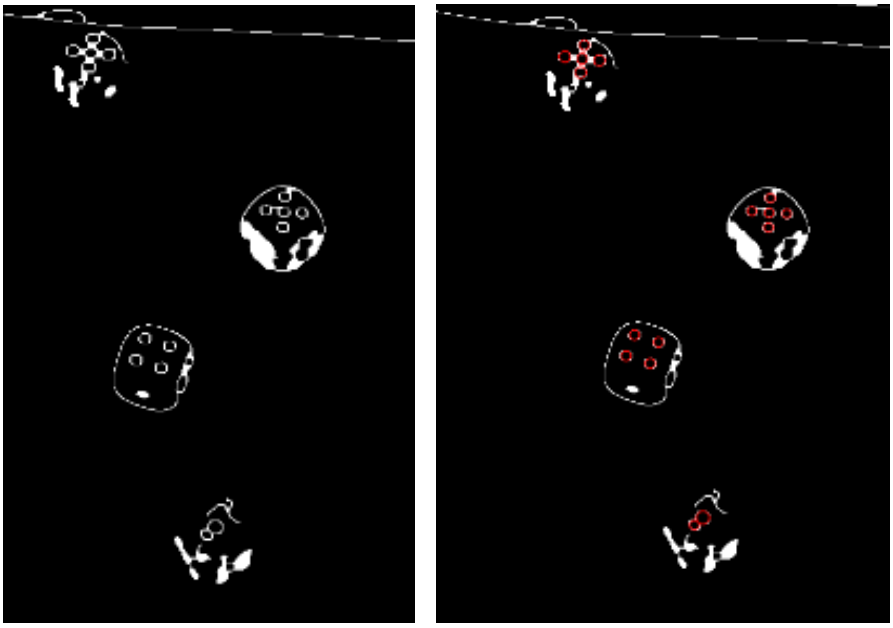
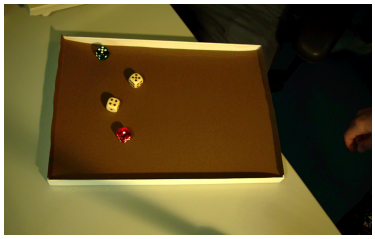


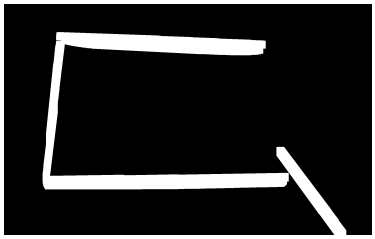
Fig. 6: Simple Blob Detection Video:2018-10-08@16-17-02.avi)

E. Region of interest

Simple blob detection algorithm sometimes detects the circular objects in the image that is not in the tray where the dice are rolled. The challenge here is to eliminate those detected blobs. Hence a region where the tray is located is found to ignore the other circular objects outside the tray. Hough lines are used to find the tray edges, and these are dilated in a large amount to form a rough rectangle. This rectangle contour is then detected using OpenCV's contour detection. The identified rectangle has used a region of interest where the blobs are detected. Blobs outside the rectangle contour are ignored. Fig. 7shows the

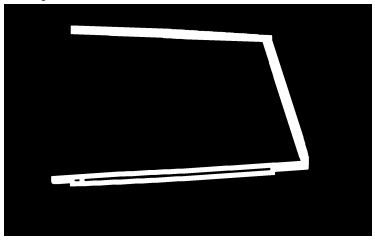


(a) Original image(Video:2018-10-08@16-17-02.avi)



(b) Detected Tray borders(Video:2018-10-08@16-17-02.avi)

(c) Original image(Video:2018-10-08@15-41-24.avi)



(d) Detected Tray borders(Video:2018-10-08@15-41-24.avi)

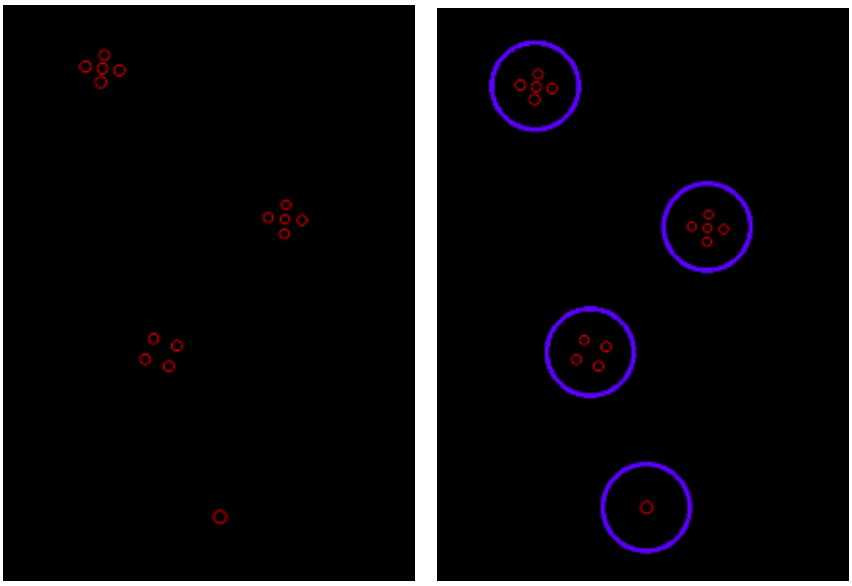
Fig. 7: Region of interest

border of the detected tray from the extracted image frame. An Alternative approach to determine the tray is k-means clustering. As the tray color is the same throughout, k-means can cluster the tray region.

The main shortcoming of using the k-means is that it consumes a lot of time. Since time is one of the critical concern in our task, this approach is not implemented.

F. Clustering

Blobs detected from simple blob detection algorithm has to be clustered to determine the values of each die. Hierarchical clustering is used to cluster the detected blobs belonging to the respective die. Hierarchical clustering is preferred over k-means as it takes very less time and also there is no need to specify the number of clusters. In k-means clustering, the number of clusters has to be mentioned. In hierarchical clustering, only the Euclidean distance has to be mentioned.



(a) Unclusterred blobs (b) clusterred blobs

Fig. 8: Hierarchical Clustering Video:2018-10-08@16-17-02.avi)

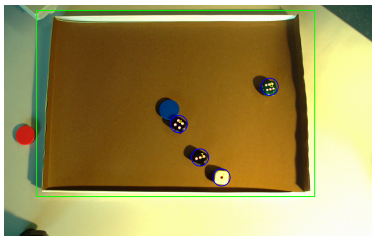
Fig. 8 clearly shows how the closest detected blobs of individual dice are clustered together. Proper euclidean distance for determining the closest blobs is determined after running several trials. This is done to prevent the neighboring dice blobs getting merged into a single cluster.

IV. Results and Evaluation

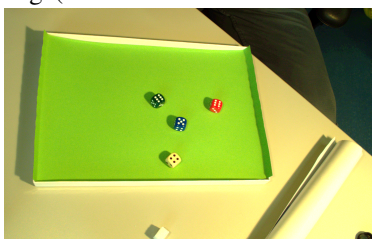
Rapid and automated dice detection system described above is evaluated on 20 videos from Hochschule Bonn Sieg - Computer Vision lab provided by the teaching assistants. Some of the results are shown in Fig. 9.

Table II shows the complete results evaluated on 20 videos. Out of 190 points, the system was able to achieve 149 points. The system failed in images which has heavy noise in it. From Table II, it is confirmed that no video takes more than 3 seconds with Intel Core i5-7200U processor (2.5 GHz, up to 3.1 GHz, 3 MB cache, two cores). The average time taken is only 2.2 seconds per video. The efficiency of the system can be increased by properly determining the region of interest around the tray. Also, a fast object detection model can be built to determine the dice and perform the operations mentioned in this paper in the dice region will increase the performance of the system.

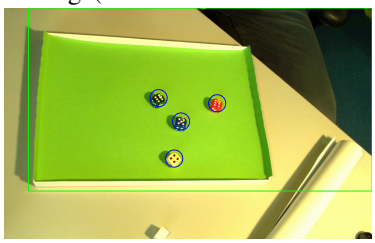
(a) Image(Video:2018-10-08@14-45-58.avi)



(b) Result image(Video:2018-10-08@14-45-58.avi)



(c) Image(Video:2018-10-08@16-03-10.avi)



(d) Result(Video:2018-10-08@16-03-10.avi)

Fig. 9: Results from Rapid and automated dice detection system

TABLE II: Results

Filename	**Total number of Dice Rolled**	**Total number of Dice Detected with correct value**	**Total number of Dice Detected with incorrect value**	**Other objects detected as dice**	**Total Points**	**Time Taken(seconds)**
2018-10-08@16-03-10.avi	4	4	0	0	8	1.24
2018-10-08@15-55-36.avi	4	1	3	8	-3	3.73
2018-10-08@16-25-11.avi	4	4	0	0	8	1.81
2018-10-08@15-47-40.avi	6	2	0	1	3	1.80
2018-10-08@15-41-24.avi	5	3	1	0	7	1.58
2018-10-08@16-28-46.avi	4	3	1	1	6	0.99
2018-10-08@16-17-02.avi	4	4	0	0	8	2.33
2018-10-08@13-38-29.avi	6	6	0	2	12	3.88
2018-10-08@13-54-00.avi	5	5	0	0	10	3.00
2018-10-08@15-00-41.avi	5	4	1	0	9	2.64
2018-10-08@16-13-24.avi	5	4	1	0	9	1.59
2018-10-08@16-09-35.avi	3	3	0	0	6	1.93
2018-10-08@15-34-45.avi	5	2	2	0	6	2.25
2018-10-08@15-26-37.avi	6	6	0	2	10	1.23
2018-10-08@15-20-53.avi	5	3	0	0	6	2.19
2018-10-08@14-55-15.avi	6	6	0	0	12	2.78
2018-10-08@16-20-32.avi	3	3	0	0	6	2.50
2018-10-08@15-14-43.avi	5	4	1	0	9	2.84
2018-10-08@14-45-58.avi	4	4	0	0	8	2.16
2018-10-08@15-51-36.avi	6	4	1	0	9	1.85

ACKNOWLEDGMENT

I would like to thank Prof. Dr. R. Herpers for providing me an opportunity to work in this awesome project. Also,I would like to extend my gratitude for my supervisor Katharina Stollenwerk for her support in this project.

REFERENCES

[1] Huang, K.Y., 2008. An auto-recognizing system for dice games using a modified unsupervised grey clustering algorithm. Sensors, 8(2), pp.1212-1221.

[2] Correia, B.A.B., Silva, J.A., Carvalho, F.D., Guilherme, R., Rodrigues, F.C. and de Silva Ferreira, A.M., 1995, March. Automated detection and classification of dice. In Machine Vision Applications in Industrial Inspection III (Vol. 2423, pp. 196-203). International Society for Optics and Photonics.

[3] Chung, C.H., Chen, W.Y. and Lin, B.L., 2009. Image identification scheme for dice game. In Proc. of 2009 Int. Conf. on Advanced Information Technologies (pp. 24-25).

[4] Bergant, U., Zimic, N. and Lapanja, I., 2000, January. Design considerations of an electromechanical dice gambling machine. In Industrial Technology 2000. Proceedings of IEEE International Conference on (Vol. 1, pp. 444-447). IEEE.

[5] Glenn De Backer, "Recognizing Dices", 28 September 2016, Retrieved from: https://www.simplicity.be/article/recognizing-dices/.

[6] Bradski, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[7] Kyle Schulz, "Dice Roll Recognition", 2013, Retrieved from: http://inside.mines.edu/whoff/courses/EENG510/projects/2013/Schulzpresentation.pdf.

[8] Adrian Rosebrock , "Zero-parameter, automatic Canny edge detection with Python and OpenCV ", 2015, Retrieved from: https://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/

[9] Satya mallick , "Blob Detection Using OpenCV (Python, C++) ", 2015, Retrieved from: https://www.learnopencv.com/blob-detection-using-opencv-python-c/

[10] Wikipedia , "Computer Vision ", 2019, Retrieved from: https://en.wikipedia.org/wiki/Computer_vision