Dice Counter

Patrick Schell
Joseph Gozum
EE146 - Computer Vision
March 15th, 2018

Problem

How can we make a computer read the value of dice?

Reference: Dice recognition program made by Glen De Backer
https://www.simplicity.be/article/recognizing-dices/

Technical Approach

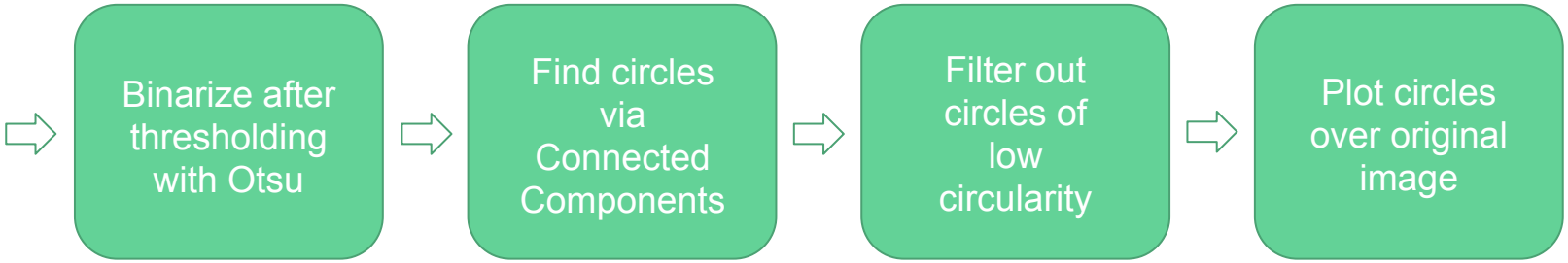
> Binarize image

> Find connected components

> Filter out unwanted components

> Show results

Technical Approach



Result

> The value the computer calculated for one die is displayed

> Location of die pits (circles) are plotted

Results from Connected Components

OVERALL

All Gameboards		Actual Class							
		0	1	2	3	4	5	6	TOTAL
Predicted Class	0	9	1	0	0	1	0	0	11
	1	0	17	3	2	0	0	0	22
	2	0	2	10	4	1	0	0	17
	3	0	0	5	11	3	1	0	20
	4	0	0	0	4	11	3	1	19
	5	0	0	0	0	2	14	4	20
	6	0	0	0	0	0	1	16	17
	>6	0	0	0	0	0	0	4	4
	TOTAL	9	20	18	21	18	19	25	130

FP	TP	TN	FN

n = 130	Predicted		
Actual	TN = 9	FP = 18	27
	FN = 26	TP = 80	106
	35	98	

Accuracy	0.68
Error Rate	0.338
Sensitivity	0.75
False Positive Rate	0.67
Specificity	0.33
Precision	0.82
Prevalence	0.815

Conclusion

1.) Connected components provides scale invariance

2.) Otsu algorithm is optimal in segmenting background and foreground

3.) Background subtraction and hough circle transform constrains the algorithm too much

4.) Expand as a tool for blind people to be able to play boardgames
    > Computer voice to read dice value
    > Board reading
    > Piece placement and score reading

Dice Counter

University of California, Riverside
EE146 - Computer Vision
Joseph Gozum
Patrick Schell

I. Introduction

Sight is an integral part of human interactions. We base many of our actions on what we see. Often times, people have lose their sight for one reason or another. This can have a negative impact in everyday life.

We propose a program to help the blind in one aspect of life, board games. The program is designed using connected components and eccentricity thresholding to analyze the dice roll value. This can allow them to know the roll they get on the board and get them more involved.

II. Technical Approach

The algorithm uses an image binarization with a threshold decided by the Otsu Algorithm. This binarized image is then inverted for the dots to become the foreground. The open morphological method is used to remove any stray pixels.

A connected components approach is taken to provide regions of interests. The centroid and eccentricity are extracted from these regions of interests. A “for loop” checks each regions eccentricity via threshold and records the current centroid and number of successfully detected dots.

III. Data Acquisition

Equipment*:* Several board games were utilized for experimentation, they include: Risk, Zelda-themed Monopoly, Future-themed Monopoly, and Clue. A standard six-sided, black and white die. A tripod and a Canon 7D DSLR with an 18-105mm lens.

Typically, ¼ to ½ of the total game board was roughly used to roll the die on. The die would be rolled within the area of play and the camera, mounted on the tripod would capture the event. The camera was set to roughly ~35mm to minimize barrel distortion, as well as to capture the intended ¼ to ½ area of play.

Between 30-33 images were captured per game board to be used as inputs to our program to determine the die roll value.

IV. Results

Fig. 1 in the appendix shows the resulting performance of the program using the data collected in a confusion matrix. From the figure we can see that our implementation tends to under-detect the correct value of the die (number of black circular pits in our experiments). We also have have many false positives in which the program detected a higher value then there

actually was, usually due to the program identifying other circular objects in the image. However, more than half of the 130 data images had a correct value detection, which are the values lying on the blue diagonal of the confusion matrix.

Based on Fig. 2, we have following derived statistics in the table below:

Accuracy	$\frac{TP+TN}{TOTAL}$	0.68	68%
Error Rate	$\frac{FP+FN}{TOTAL}$	0.338	33.8%
Sensitivity	$\frac{TP}{P}$	0.75	75%
Precision	$\frac{TP}{TP+FP}$	0.82	82%

Here we can see a 68% accuracy of the program, correctly identifying the value of the die roll slightly greater than ⅔ of the time. The accuracy suffers from true negative cases being difficult to define. This is something that can be improved upon in future revisions.

The error rate of 33.8% indicates that typically our program misidentifies the value of the die about a ⅓ of the time.

The sensitivity indicates the number of correct positive predictions divided by the total number of positives (excluding false positives) and the best sensitivity is a value of 1.0. We have a value of 0.75 indicating we have a good level of sensitivity.

Precision is similar to sensitivity but it is divided by the total number of positives (both true and false positives). Again value of 1.0 is ideal and our program has a level of precision at 0.82, indicating a high level of precision.

V. Conclusion

1. Connected Components coupled with an eccentricity thresholding provides scale invariance with minimal loss in accuracy.
2. Otsu algorithm is useful for segmenting the foreground and background even when the ground has some intensity similar to the foreground.
3. These two methods combined provide a flexible and fairly accurate approach to the problem.
4. Future work for this project would be computer voice reading of the number rolled. This method could be expanded to help the blind have more involvement in board games. This can include future work in reading a players current board placement, keeping track of their score, and reading the board for players to strategize.

VI. References

Glen De Baker, Dice Recognition Program, https://www.simplicity.be/article/recognizing-dices/

n = 130	Predicted		
Actual	TN = 9	FP = 18	27
	FN = 26	TP = 80	106
	35	98	

Appendix

Fig. 1 - Example Output

Fig. 2a - Original Confusion Matrix

OVERALL

All Gameboards		Actual Class							
		0	1	2	3	4	5	6	TOTAL
Predicted Class	0	9	1	0	0	1	0	0	11
	1	0	17	3	2	0	0	0	22
	2	0	2	10	4	1	0	0	17
	3	0	0	5	11	3	1	0	20
	4	0	0	0	4	11	3	1	19
	5	0	0	0	0	2	14	4	20
	6	0	0	0	0	0	1	16	17
	>6	0	0	0	0	0	0	4	4
	TOTAL	9	20	18	21	18	19	25	130

	FP	TP	TN	FN

Fig. 2b - Simplified Confusion Matrix

SOURCE CODE

```
clear all;
close all;
clc;

for a = 1:32
   %Used to loop through all files in the specified directory
   filename = ['C:\Users\josep\Desktop\CV_DATA\z_mono\' num2str(a) '.JPG'];
   img = imread(filename);
   TEMP = img;
   img = rgb2gray(img);    %Grayscaled version of image

   %============== Finding the circles in the Dice ============================
   threshold = graythresh(img);                %Finds optimal threshold for binarization
   BW = imcomplement(imbinarize(img, threshold));  %Thresholds images and inverts it
   BW = bwareaopen(BW, 600);                   %Removes small objects from image
   %Connected components
   CC = bwconncomp(BW);
   stats = regionprops('table', CC, 'Centroid', 'Eccentricity', 'MajorAxisLength','MinorAxisLength');
   %Finds the circles of interests
   c = stats.Centroid;
   cnt = 0;
   centers = zeros(length(stats.Centroid), 2);
   diameters = zeros(length(stats.MajorAxisLength), 1);
   for i = 1: length(stats.Eccentricity)
      if (stats.Eccentricity(i,1) <= 0.28)
         centers(i, 1) = c(i, 1);
         centers(i, 2) = c(i, 2);
         diameters(i, 1) = mean([stats.MajorAxisLength(i,1) stats.MinorAxisLength(i,1)],2);
         cnt = cnt + 1;
      end
   end
   radii = diameters/2;
   %Overlays some information on image
   cnt = num2str(cnt);
   I_ovrlay = insertText(TEMP,[0,0],cnt,'FontSize',200,'BoxColor','red','BoxOpacity',1.0,
'TextColor', 'white');
   figure, imshow(I_ovrlay), title('GAMEBOARD');
   hold on
   viscircles(centers,radii);
   hold off
end
```