Who laughs at the end of a game video based on the gameplay tracking

Lala Ahmadova

Modelling project work 2
Supervisors: Pataki Péter, Tamaga István

Company: ARH ZRT

December 8, 2020

Overview (1st semester)

Goal

Video recording of the well-known board game, called Ludo,must be tracked and reconstructed by using image processing tools. In the first semester, pictures will be added. They may differ by their shooting angle, playfields (playfields may contain circles or squares), different light conditions and etc. With simple table elements (fields, etc), edge finders (Canny, Sobel) and detection by other elementary image processing tools the playfield should be detected. Meanwhile, a text document will be obtained which contains 'x' and 'y' coordinates of playfield center points..

Implementation

To reach this goal the functions and algorithms of the library OpenCV was used. The steps are followings:

- Load a picture and make it black-white.
- Apply Canny edge detection algorithm to find edges .
- Apply Sobel on x and y direction and calculate magnitude.
- Normalize magnitude and find contours (clusters of points).
- Open a text file namely "coordinates.txt" and using moments function find centroid of contours.
- Draw contours and mark their center points.
- Close the text file and show the result.

Result

In conclusion, this algorithm finds the playfield of the game and marks the center points. The main drawback is that a few wrong points are also detected although they are not center points. The second one is working better for computer based images than real-life ones. Detecting playfield despite it being made of circles or squares can be considered as an advantage.

2 nd semester

Goal

Our goal for this semester is to improve algorithm in a way that only center points should be detected. Meanwhile, a text file namely coordinates.txt is generated which contains center points. After removing the complication that we faced last term the dice detection and finding its value is required. To achieve 2 different methods have to be used.

Algorithm

The algorithm for this semester differs from previous one. The basic logic is to use Hough Circle transformation. We can describe it as follows:

1. Import the picture and change color
   Changing color is for reducing noise and avoid false circle detection.

2. Use cv2.HoughCirle() function to detect circles
   With correct parameter values it detects all the circles and also dice.

3. Sort them by ascending order
   The playfield contains small circles where most of them are same in size. After running Hough Circle transformation it can be easily seen. By sorting them in an ascending order we can get the ones which differ in size.

4. Take the 2 biggest one (They are dice and center of the image)

5. Cut the image around these circles one-by-one
   What is left is to detect which one is dice and count it. For this aim we cut all the image around each circle.

6. Use cv2.HoughCircles() with different parameters to find small circles on dice
   At the end we apply the same method to find value of dice. Since dice itself contains small black circles the algorithm can count it.

Explanation of used functions

Houghe Circle Transformation

A circle is defined by the triplet of three parameters $(x_{center}, y_{center}, r)$ where x_{center} and y_{center} is center coordinates and r is the radius of the circle. Mathematically it is represented as

$$(x - x_{center})^2 + (y - y_{center})^2 = r^2$$

The function HoughCircles is used in OpenCV to detect the circles in an image. This method finds the possible circle centers and edges as a fist step. Secondly, it detects most

relevant radius to the corresponding centers.
One can see

$cv2.HoughCircles(image, method, dp, mindst, param1, param2, minradius, maxradius)$

- ***image*** Input image (grayscale)
- ***method*** Detection method.(The only available one is HOUGH-GRADIENT in OpenCV)
- ***dp*** Inverse ratio of accumulator resolution and image resolution.
- ***mindst*** minimum distance between centers od detected circles.
- ***param1*** **and** ***param2*** are method specific parameters
- ***minradius*** minimum circle radius
- ***maxradius*** maximum circle radius

Image shape

The shape of the image is triplet of 3 values: rows, columns and channels and can be get ***img.shape***. In case an image is grayscale then only the number of rows and the number of columns will be get.

Resize the image

cv2.resize() is a function to resize an image. The size of the image can be specified manually, or you can specify the scaling factor.

Conclusion

Main difficulty of this semester is to adjust values for parameters of Hough Circle transformation to find circles. Small changes can cause detection of numerous unknown circles or it finds a few circles. This resulted by wrong values of dice. Generally the function detects the center of circles well. The main problem occurs in the case of radii. To get better results the values for ***minradius*** and ***maxradius*** should be adjusted correctly. If the value of threshold for center detection is low, this is ***param2***, then large amount of circles will be generated.

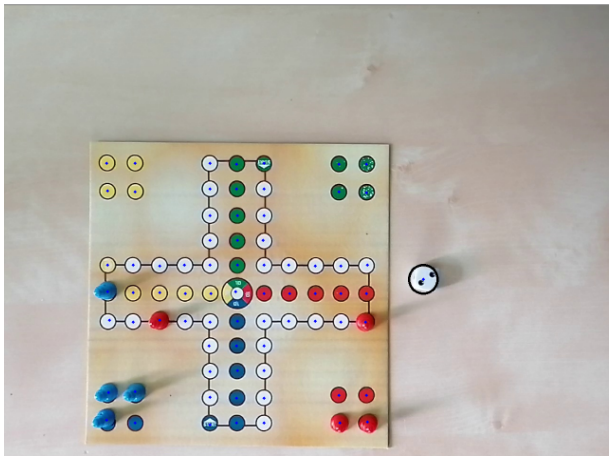
Out of 53 images 10 of them worked well with exact dice detection and value, for other 11 of them it found dice correctly but can not read the value. For others it could not detect the dice.

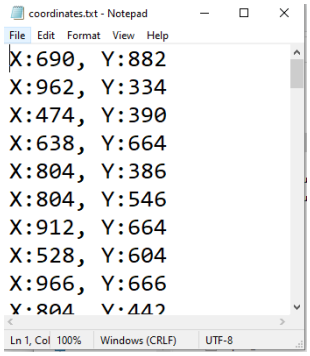
Although the task of this semester is to use two different methods (Haar Cascade and simple image processing tools) to find dice and to read its value I was able to do only one of them.

Addition to the next semester's task I am planning to improve my algorithm for dice detection and to apply Haar Cascade and compare the accuracy level of these two methods.

When algorithm works well:

```
========== RESTART: C:\Users\Asus\Desktop\project-Lala\2.py ==========
2
Reading dice:
 value:  2
```





When program finds dice with wrong value:

```
========== RESTART: C:\Users\Asus\Desktop\project-Lala\2.py ==========
2
Reading dice:
 value:  2
```