Circle Detection Using Computer Vision for Skeleton Detection and Simulation of Mechanical applications

Mosab Wadea Hussain
s201021320@kfupm.edu.sa
Dept of Computer Engineering, KFUPM
Dhahran, Saudi Arabia

***Abstract*—This work is meant to be a course project for Computer Vision course (COE 487). The course gave the students a clear understanding and guided them into processing images and applying algorithms on images to take a decision based on some logic that can be performed on the images. This paper will go through the process of how the project identifies some circles and detects the movements of these circles and represent them in some form in order to send these values to some other software for simulation.**

I. Introduction

Computer vision field is one of the most important fields in the uprising and rapidly expanding field of inventions and new technologies. The importance of computer vision emerges when there is a need of a computer to analyze a visual thing and especially if that was needed in real time. On the other hand, the mechanical field now is trying to increase the efficiency of the newly designed machines and moving objects. In order to satisfy this, simulations must be performed on the designs and devices. Here comes one of the most important fields of computer vision. Where it can be possible to capture the movements of a device and identify it and represent it.

II. Goal

The main aim of this project will be developing a program that can get live camera capturing for a moving mechanical object that is tagged with colored circles, these circles are detected by an algorithm and represented in a format that can be used to simulate the object's movement in other simulating softwares. This will help the mechanical engineers to spot issues in a design that is still not fully functioning by simulating the movement and apply extreme tests and scenarios during the simulation without the need to damage the device. It can also help to evaluate the movement of a device that can not be reached by workers or can not be stopped.

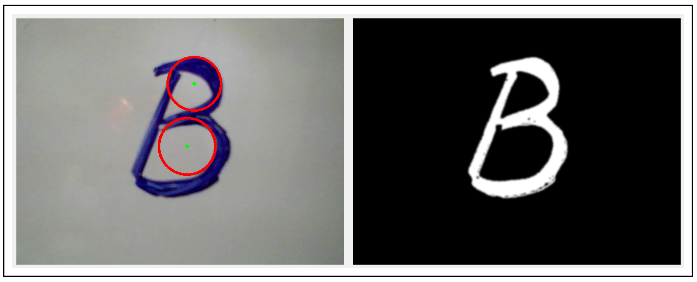


Fig. 1. Left: original image. Right: filtered image

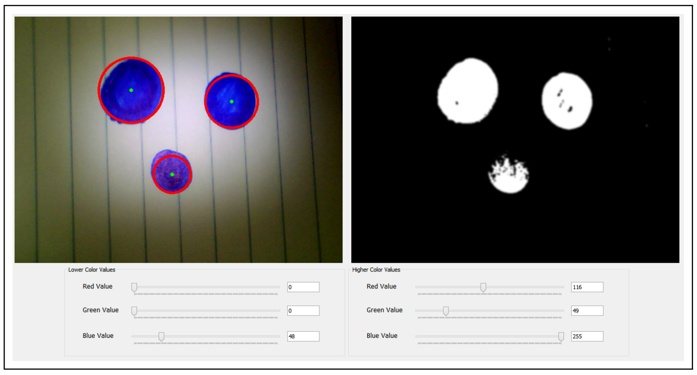
III. Methodology

The project uses an algorithm that was developed in C#, using the EmguCV [1] library in C#. This algorithm detects the circles in an image that is taking by a camera. Before the camera input is processed by the algorithm it must be filtered, the filtration is mainly to remove the unwanted colors in the image. Then a threshold filter is applied to convert the image into a binary image that the algorithm can deal with. The algorithm used in detecting circles is the *Hough* filter for detecting contours. Here is the code developed in C# using Visual Studio:

A. Filtering Code:

The code shown below is for capturing and filtering. It starts by querying the image frame from the camera object. if there is an image it will apply the function *InRange* that will isolate the unwanted colors, the resulting image is assigned to a binary type variable, this will convert the image to a binary image. then it will smooth the resulting image to remove any remaining noise. The result of this process for blue color filtering can be seen in Fig. 1.

```
org = cam.QueryFrame();
if (org == null) return;

proc = org.InRange(bgrL, bgrH);
//bgrL= lowest BGR value, bgrH= highest BGR
```



Lower Color Values

Red Value	0
Green Value	0
Blue Value	48

Higher Color Values

Red Value	116
Green Value	49
Blue Value	255

Fig. 2. The red circles are drawn by the program

```
// value
proc = proc.SmoothGaussian(9);
```

B. *finding and plotting circles:*

The method used for this purpose is called *HoughCircles* this method will return an array of *Circle* objects that can be used in the program. The following code takes each circle in the array and plots it on the original image it self.

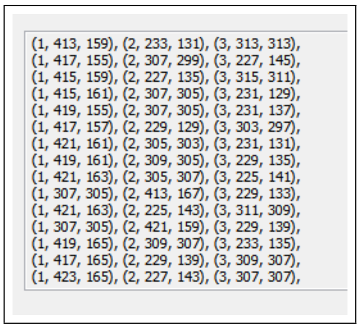
```
CircleF[] circles = proc.HoughCircles(new
    Gray(100), new Gray(50), 2,
    proc.Height/4, 10, 400)[0];

for (int i =0; i<circles.Length; i++) {
  CvInvoke.cvCircle(org, new
      Point((int)circles[i].Center.X,
      (int)circles[i].Center.Y), 3, new
      MCvScalar(0, 255, 0), -1,
      LINE_TYPE.CV_AA, 0);
  org.Draw(circles[i], new Bgr(Color.Red), 3);
  con += "(" + (i+1) + ", " +
      circles[i].Center.X + ", " +
      circles[i].Center.Y+"), ";
 }
if(circles.Length>0)
  consol.AppendText(con+"\n");
imageBox1.Image = org;
imageBox2.Image = proc;
```

The result seen in Fig. 2 is the final output of the algorithm. The application gives the user the ability to choose and try different color values for the filtering as show in Fig. 2.

IV. Results

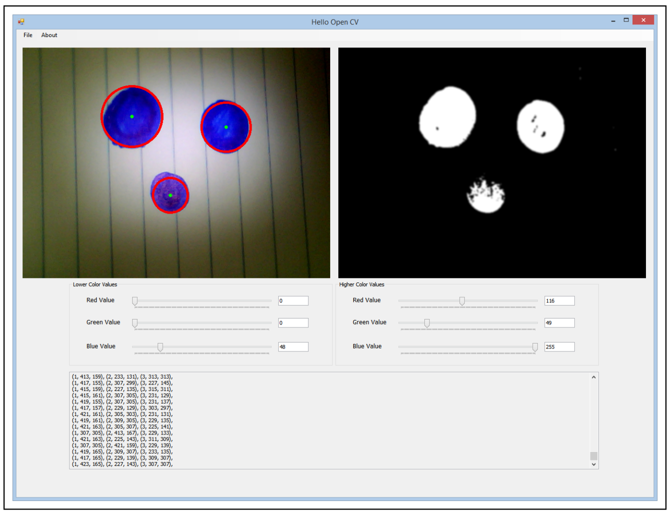
The output of this software is written directly in a console component. The values printed are the circles and their locations in the image. They are printed in the format ([*circle_num*], [*x_coordinate*],[*y_coordinate*]) as can be seen Fig. 3. It can be seen in the output that the camera was moving, since the coordinates of the circles keeps changing.

In order to take the values to other program the user can copy the values to clipboard from the file menu and then past the result in another software.
An over view of the application is shown in Fig. 4.



```
(1, 413, 159), (2, 233, 131), (3, 313, 313),
(1, 417, 155), (2, 307, 299), (3, 227, 145),
(1, 415, 159), (2, 227, 135), (3, 315, 311),
(1, 415, 161), (2, 307, 305), (3, 231, 129),
(1, 419, 155), (2, 307, 305), (3, 231, 137),
(1, 417, 157), (2, 229, 129), (3, 303, 297),
(1, 421, 161), (2, 305, 303), (3, 231, 131),
(1, 419, 161), (2, 309, 305), (3, 229, 135),
(1, 421, 163), (2, 305, 307), (3, 225, 141),
(1, 307, 305), (2, 413, 167), (3, 229, 133),
(1, 421, 163), (2, 225, 143), (3, 311, 309),
(1, 307, 305), (2, 421, 159), (3, 229, 139),
(1, 419, 165), (2, 309, 307), (3, 233, 135),
(1, 417, 165), (2, 229, 139), (3, 309, 307),
(1, 423, 165), (2, 227, 143), (3, 307, 307),
```

Fig. 3. This is the output for the same image in Fig. 2



Lower Color Values

Red Value	0
Green Value	0
Blue Value	48

Higher Color Values

Red Value	116
Green Value	49
Blue Value	255

```
(1, 413, 159), (2, 233, 131), (3, 313, 313),
(1, 417, 155), (2, 307, 299), (3, 227, 145),
(1, 415, 159), (2, 227, 135), (3, 315, 311),
(1, 415, 161), (2, 307, 305), (3, 231, 129),
(1, 419, 155), (2, 307, 305), (3, 231, 137),
(1, 417, 157), (2, 229, 129), (3, 303, 297),
(1, 421, 161), (2, 305, 303), (3, 231, 131),
(1, 419, 161), (2, 309, 305), (3, 229, 135),
(1, 421, 163), (2, 305, 307), (3, 225, 141),
(1, 307, 305), (2, 413, 167), (3, 229, 133),
(1, 421, 163), (2, 225, 143), (3, 311, 309),
(1, 307, 305), (2, 421, 159), (3, 229, 139),
(1, 419, 165), (2, 309, 307), (3, 233, 135),
(1, 417, 165), (2, 229, 139), (3, 309, 307),
(1, 423, 165), (2, 227, 143), (3, 307, 307),
```

Fig. 4. Over view of the application

V. Conclusion

This project is just scratching the surface of the vast variety and many applications of computer vision. This might not be the best approach of skeletonization and identifying movements, but it can be further improved by adding more algorithms that can link the circles, also if the output was processed by some of the graphing algorithms where it will make it more accurate and precise. A future work for this project is to implement a third axis detection and representation.

VI. References

[1] "Emgu cv: Opencv libray for .net platform, *http://www.emgu.com/wiki/index.php/Main*$_{Page}$''.