DESIGN OF AN IMAGE PROCESSING ALGORITHM FOR BALL DETECTION

Ikwuagwu Emole
B.S. Computer Engineering '11
Claflin University

Mentor: **Chad Jenkins, Ph.D**
Robotics, Learning and Autonomy Lab
Department of Computer Science
Brown University, RI

Providence, RI 2010

Table of Contents:

1. INTRODUCTION

The aim of this research project was to design an algorithm for ball recognition which will be used in a bigger project – an outdoor soccer game using the iRobot Create, an Asus Eee PC and a camera. This paper focuses on the computer vision methods that were applied towards achieving this aim. The main approach used involves edge detection. The paper contains my attempts and the extent to which the ultimate goal has been accomplished.

The programming language of choice in this project was C++ for the following reasons - it is one of the languages that are supported by ROS (including Python); it is one of the languages that currently support OpenCV (including C and Python); and it's the language with which the author feels most comfortable with.

2. ROS

ROS (Robot Operating System) is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. [1]

This is the main platform upon which all our experiments are run. It is installed on the Asus PC which is then connected to the iRobot Create and camera. It contains libraries for handling streaming camera images and also for communicating with the Create's motors and sensors.

In ROS, programs are called *nodes*. For one node to make certain data available to other nodes, it *publishes* such data to a *topic*. A node that needs such data then *subscribes* to the desired topic. When many nodes need to run at the same time, one can utilize *roslaunch*. By creating a *launch* file that contains all the nodes, all the process can start at the same time.

3. OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. [2] It contains functions and methods of obtaining and manipulating image/video data. Aside the basic functions of displaying, resizing or saving an image, this robust library can aid in methods of computer vision such as edge detection and object recognition. For our project, the methods of interest include Hough Transforms and Template Matching.

4. IMAGE PROCESSING

4.1 Image Conversion from ROS image message to OpenCV

The first step of the process is to obtain ROS images being published by the robot's camera and then convert them to OpenCV type images so that we can easily manipulate the images. In the Brown ROS Package, the node that publishes camera images is called *gscam*. A ROS node was written to subscribe to these streaming images. While *gscam* runs, our new node subscribes to the images being published and then converts them to the OpenCV *IplImage* (Intel 'Image Processing Library' Image) format using *CvBridge*. An image in this format can then be saved or manipulated as wished.

4.2 Getting and Saving Ball Images

Images of the ball are needed for our experiment and such images are to be used as seen by the robot as it drives towards the ball. Two nodes were to run side-by-side – (i) drive towards the ball; and (ii) save camera images to the computer. The concurrency was achieved by creating a *launch* file that contains the two nodes and their dependencies.

With the camera mounted on the Create and a soccer ball placed in its view, the Create is set to drive towards the ball while images of the ball, as seen by the robot, are saved onto the Asus laptop. The images were set to be saved in the order of *pic1.png, pic2.png,* etc.

4.3 Obtaining Ball Information from Saved Images

With a database of over 200 images, the size and location of the ball needed to be obtained and this involves recording the radius of the ball and the position of its center in each image. To do this, a new node was created which uses HighGUI (High Level Graphical User Interface), the OpenCV GUI that enables a user to interact with an image using the keyboard or a mouse. Using this node, the user circumscribes the ball in an image with a circle by dragging a circle across the ball.

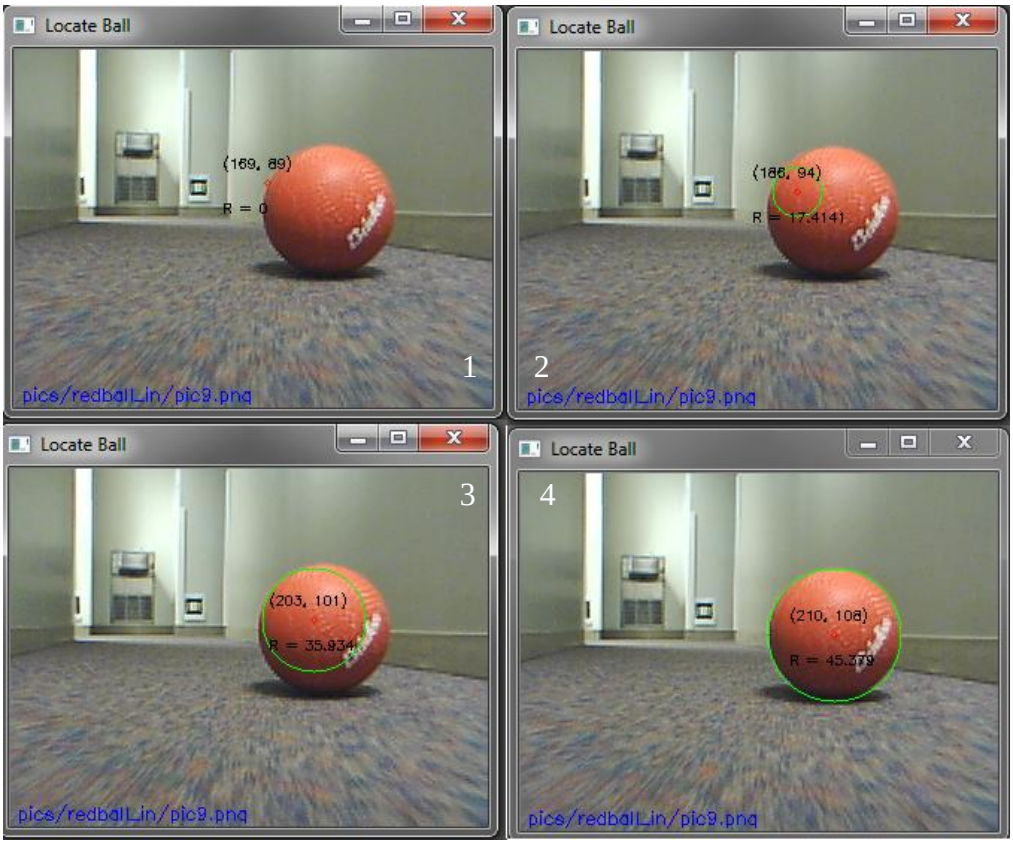

Fig. 1: Using the mouse, a circle is dragged over the ball starting from any point on the circumference.

The radius and center of this circle can then be saved in a text file by hitting a key.

The newly created text file is a record of file names, ball center and radius.

The node also contains certain shortcuts to load next or previous images, delete any undesired circle and save current circle information. This exercise was carried out for all the ball images in the database.

4.4 Ball Detection

The two methods that were attempted are Hough Transforms and Template Matching.

4.4.1 Hough Transforms

Hough Transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a

certain class of shapes by a voting procedure. [3] An OpenCV function called *cvHoughCircle* is used to detect all circles in an image. The reason why this method is being used is because the ball is presumed to be a circle on the 2-D image.
For the *cvHoughCircle* to work correctly, the image has be to converted to grayscale and then a Canny Edge detector is used to convert the image into a format that makes the circles in the image more obvious to the *cvHoughCircle* method. A function called *cvCanny* is used to achieve this aim. The *cvCanny* parameters are as follows: cvCanny(const CvArr* src, CvArr* dest, double threshold1, double threshold2, int aperture_size=3). The function cvCanny finds the edges on the input image src and marks them in the output image dest using the Canny algorithm. The smallest of threshold1 and threshold2 is used for edge linking. [4]

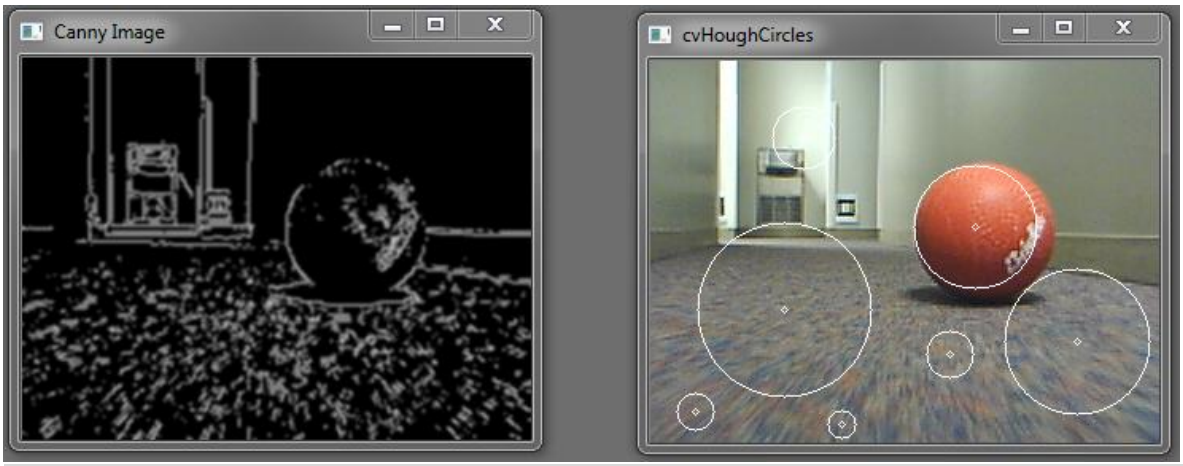

Fig. 2: Canny Edge Detection (on the left) makes it easier for a circle to be detected. `cvCanny(src, dest, 100, 100, 3)`. Image on the left shows the original image after the *cvHoughCircle* have been used to detect the circles.

False positives always come up when *cvHoughCircle* is used. These could be reduced by varying the threshold parameters.

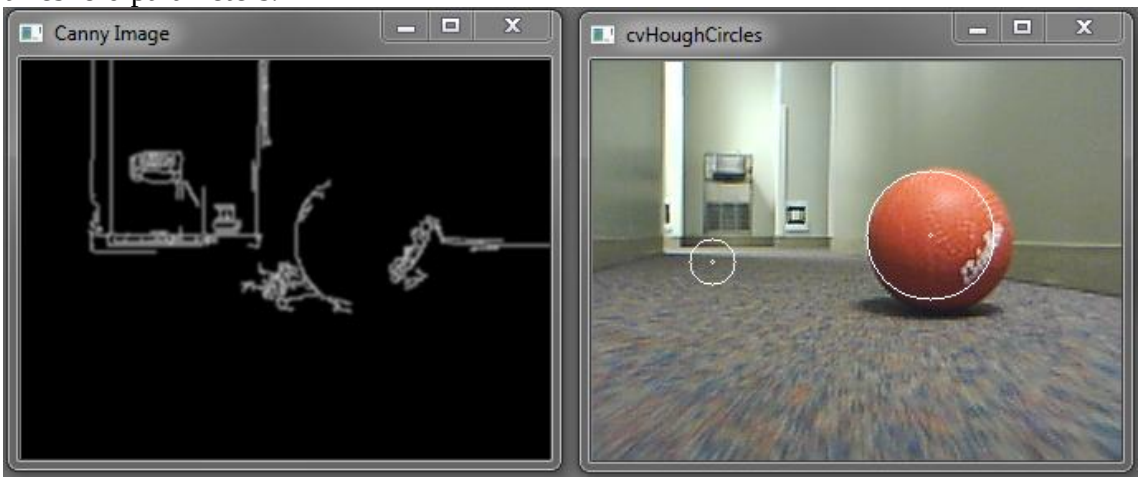

Fig. 3: The false positives were reduced by changing some of the parameters: `cvCanny(src, dest, 10, 500, 3)`.

Shadow Detection

According to R. Woering, the presence of a shadow underneath the ball could be used as a second test to determine which one of the circles detected by the Hough Transform is actually a ball. The paper suggests that if a detected circle is a real ball, there would be shadow underneath the ball. [5]
Based on that assumption, in this research, a section of code was developed which compares the

average color intensity of a rectangular area underneath the circle (where a shadow is expected to be if it's a ball) with that of a rectangular area in the circle somewhere below its center.

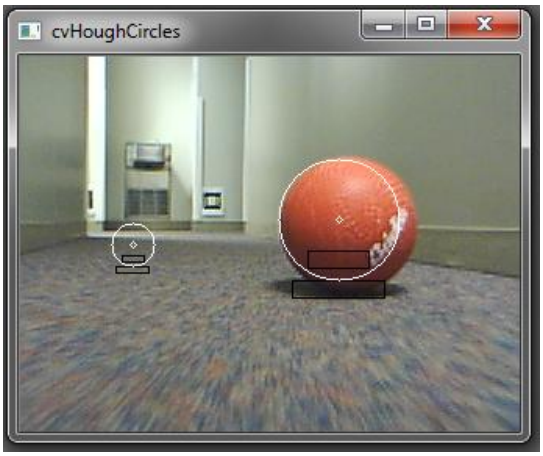

Fig. 4: Image shows the two rectangular areas to be compared for each circle. If the lower rectangle has a darker color, then there's a shadow under the circle.

Based on the fact that darker colors have a lower pixel value than brighter colors, this section of code compares the average pixel values in these areas:

[**Note:** The formulae used in this function are the opinions of the author of this paper in an attempt to allocate proportional rectangular areas to circles of different sizes. For the purpose of this test, the image was converted to grayscale for easy comparison of color intensities.]

```
void detect_shadow(CvPoint c, int r1) //c - center of circle, r1 - radius
{
  //top rectangle below the center
  CvPoint p1 = cvPoint(c.x-(0.50*r1),c.y+r1-(r1/2));
  CvPoint p2 = cvPoint(c.x+(0.50*r1),c.y+r1-(r1/2)+(0.25*r1));
  int top = find_avg_pixel(p1, p2, image);
  //bottom rectangle below ball should contain shadow
  CvPoint p3 = cvPoint(c.x-(0.75*r1),c.y+r1);
  CvPoint p4 = cvPoint(c.x+(0.75*r1),c.y+r1+(0.25*r1));
  int bottom = find_avg_pixel(p3, p4,image);
  if(bottom < top)return true;
  else return false;
}

int find_avg_pixel(CvPoint p1, CvPoint p2, IplImage* img) //p1 & p2 correspond to
//the diagonal points of a rectangle. img is in grayscale (1-channel)
{
  int total_p=(p2.x-p1.x)*(p2.y-p1.y);     //total no. of pixels
  int sum=0, avg_p=0;
  for(int x=p1.y; x<p2.y; x++)             //loop through all pixels in the area
  {
      for(int y=p1.x; y<p2.x; y++)
      {
            CvScalar s = cvGet2D(img,x,y); //get current pixel value (intensity)
            sum += s.val[0];               //add value to sum
      }
  }
  avg_p = sum/total_p;
  return avg_p;
}
```

For each circle, the code compares the average color intensities of the two rectangles – top and bottom. If the latter is smaller, then there exists a shadow underneath the circle and that qualifies it to be a ball.

This was the outcome:

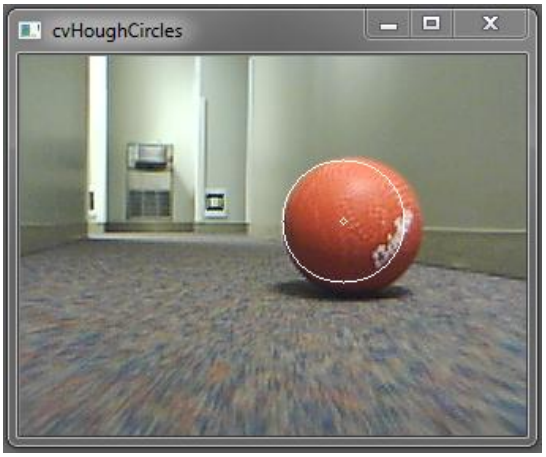

Fig. 5: This shadow-detecting technique worked out fine for this image. The false positive circle was eliminated.

Further tests showed that most images still came out with many false positive circles after the shadow detection techniques has been applied. There are two main reasons for such observation:

1. The backgrounds in these images have an uneven color. Therefore, one can detect a darker section which is not actually a shadow.
2. The ball was not detected as a Hough Circle. The rate of success of accurately detecting a ball as a circle (even if it includes false positives) was quite low.

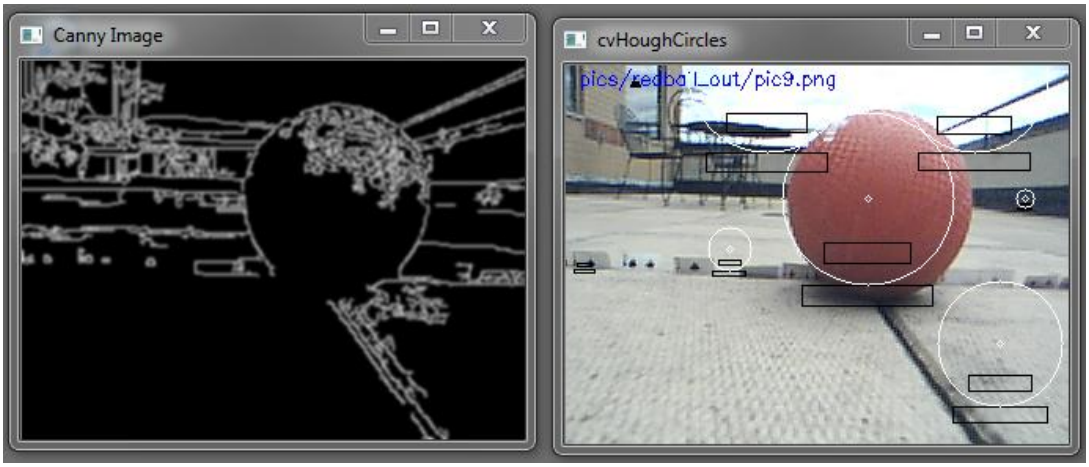

Fig. 6: In this example, the shadow detection was not sufficient in eliminating all the false positives due to uneven background colors.

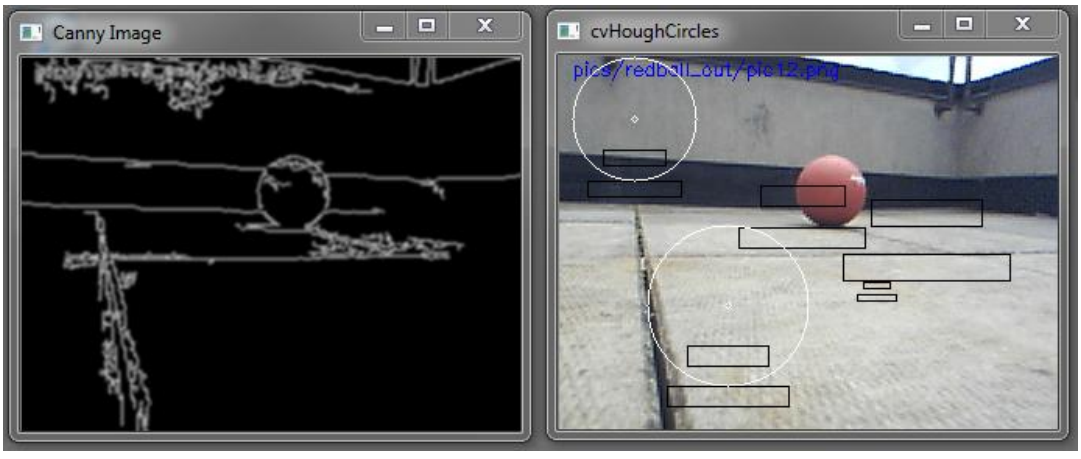

Fig. 7: In this example, the shadow detection was unnecessary because the ball was not among the circles detected by the Hough Transform circle detection technique. All the detected circles were false positives. This was the case for most test images.

4.4.2 Template Matching

Template Matching is a technique in digital image processing for finding small parts of an image which matches with a template image. It can be used to detect edges in a reference image. [6]
The algorithm searches through the reference image to find a section that matches the template image. For our experiment, a cropped out image of the soccer ball from an image was used as a template. This worked fine when used to find the ball in the original image where it was cropped from. Subsequently, when the template image size was reduced, we obtained a false positive. This suggests that template matching gets inefficient when the scale is varied. Another method that could help combat this problem is called SIFT (Scale-Invariant Feature Transform). Due to the limited time we had for the project, this could not be done.

CONCLUSION & FUTURE WORK

In conclusion, the experiment shows that if the Hough Transform appropriately detects a ball as a circle (amongst other false positives), the shadow detection technique could aid in eliminating some of the false positives. If shadow detection has to used, future test images should be taken on backgrounds with even colors. In this case, a shadow would be more distinct.

Also, through the process of this research, it has been observed that ball recognition will require more than circle detection. Future research work will require techniques that involve some feature description. Such techniques include HOG (Histogram of Oriented Gradients) and SIFT (Scale-Invariant Feature Transform). Using the database of images of the ball and the records of its position and size, other machine learning methods could be used to train the robot to recognize a ball in its view.

I would like to thank the Computing Research Association for this research opportunity and Brown University's RLAB (Robotics, Learning & Autonomy at Brown) for giving me the opportunity to perform research at the lab. Through this experience, I have gained a lot of new knowledge in a very interesting field of computer science. The skills that I have gained in ROS and OpenCV will be relevant to my future career and graduate degree pursuit.

REFERENCES

[1] "ROS Wiki" <http://www.ros.org/wiki>

[2] Bradski, G., and A. Kaehler. *Learning OpenCV*. Sebastopol: O'Reilly, 2008.

[3] "Hough Transform" Wikipedia. 2010. Wikimedia Foundation, Inc. 8 Aug 2010 < <http://en.wikipedia.org/wiki/Hough_transform>

[4] *OpenCV: Image Processing and Computer Vision Reference Manual*. University of Pennsylvania. <http://www.seas.upenn.edu/~bensapp/opencvdocs/ref/opencvref_cv.htm>.

[5] Design of a video processing algorithm for detection of a soccer ball with arbitrary color pattern. R. Woering. March 2009. Technische Universiteit Eindhoven. Netherlands <http://www.mate.tue.nl/mate/pdfs/10395.pdf>

[6] "Template Matching" Wikipedia. 2010. Wikimedia Foundation, Inc. 8 Aug 2010 < http://en.wikipedia.org/wiki/Template_matching>