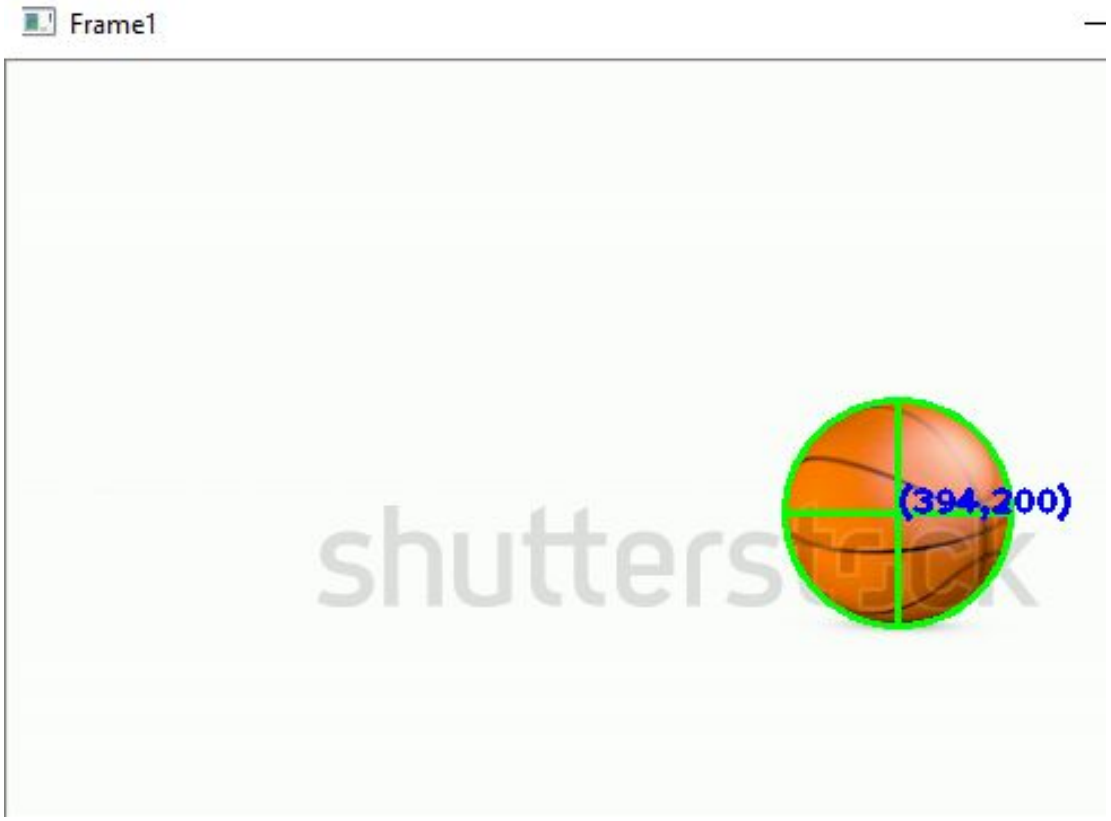
A.I.R.I.S

(Application and Interface for Real-Time Item Seeking-Tracking Movement Using Image Differencing)



Abstract

With the use of the OpenCV Computer Vision library along with Microsoft Visual Studios Integrated Developer's Environment, the goal of creating an application in the C++ programming language can be created to track objects in motion can be achieved and mimic the basic functions and processes of the human eye. The application should first be able to perform the basic principle of motion tracking by using the concept of image differencing-- also known as background subtraction. Background subtraction detections motion by comparing one image with another. In order to achieve background subtraction, the application should first be able to take a video feed and process it to create a threshold image, a difference image, and a resultant final threshold image from the subtraction of the threshold and the difference image. The difference image should detail the difference in the image before and after a frame change. The threshold image should allow users to define the moving image clearly. The final threshold image is the resultant of the threshold image refined by the difference image. The application should also be able to allow users to toggle between multiple views for the final output and allow the user to see the threshold, difference, and final threshold images. Ultimately, the application seeks to then track and find an object's location (relevant to its centroid) from both through a video file and through a real-time camera feed.

First of all, a computer with Windows 10 operating system is needed for the project because Microsoft Visual Studios is needed (Macintosh computers do not have Microsoft Visual Studios). Then, the download of OpenCV 2.4.10 and Microsoft Visual Studios 2013 Update 5

was performed with the appropriate linking of files. A code was then inputted into Microsoft Visual Studios to program a video tracking image which allows the analyses of points.

Further research was conducted and a live-feed video tracking application was produced. It can track any moving targets such as yourself, just like the video tracking image produced in Microsoft Visual Studios.

The hypothesis was supported overall because the program can mimic basic functions of the human eye. By putting the grid on top of the video footage, and using a difference threshold to create a final threshold (to find the differences in each frame), the application is able to pinpoint the center of moving objects and track the item's movement on a live video feed.

Question

- How can we create a program that is capable of performing real-time motion tracking and
- object detection through a live video feed using the C++ programming language?
- How can we process and analyze real-time video footage?
- How does the program mimic the basic functions and processes of the human eye?

Hypothesis

If an application is created to mimic the basic functions of the human eye such as motion tracking and object detection, then the application should be capable of reading real time video footage and analyzing it to detect color and motion.

Materials

- Computer with Microsoft Windows 10
- Microsoft Visual Studios 2013 with Update 5
- OpenCV 2.4.10
- Video Footage/Webcam

Object Tracking Procedure

1. Download an integrated development environment (IDE) for the users' corresponding machine capable of running C++ programs such as Microsoft Visual Studio Community 2013.
2. Download a functional version of the OpenCV library.
3. Extract and install OpenCV following the corresponding version documentation.
4. Connect an external or internal camera to the Computer with the latest driver updates, if applicable.
5. Create a .cpp file in your IDE and connect it to the OpenCV library. Also include:

```
#include <opencv\cv.h>
#include <opencv\highgui.h>

using namespace std;
using namespace cv;
```

6. Set up the basic public variables and pointers for future use in the application such as:

```
//Our sensitivity value to be used in the absdiff() function
const static int SENSITIVITY_VALUE;
//The size of blur used to smooth the intensity image output from absdiff() function
const static int BLUR_SIZE;
//In this project we will have just one object to search for and keep track of its position, so an integer array should be made to store the x and y coordinates of the object.
int theObject[2] = {0,0};
//Bounding rectangle of the object, we will use the center of this as its position.
Rect objectBoundingRectangle = Rect(0,0,0,0);
```

7. Also create a int to String helper function to take in a number input and create a string output.
8. Create a method to search for movement that will take in the threshold image and the video feed. The values should be taken from the feed and passed into the function and manipulate them rather than just working with a copy.
9. In this method, there should be to vectors needed to fund the output to find the contours. The filtered image should be found using OpenCV's findContours function. If the contours vector is not empty then that means we have objects found. The largest contour will be found at the end of the contours vector and we will assume that the biggest contour is the object we are looking for.
10. For detection purposes, a bounding rectangle will be around the largest contour then its centroid will be found and help find the object's final estimated position. The object's position will be updated by changing the array values that it is stored in.
11. The position of the object should also be written to the screen for debugging purposes.

12. In the main of the .cpp program, variables should be made for functionality and future use:

```
bool objectDetected = false;
//Check for the modes of the video output
bool debugMode = false;
bool trackingEnabled = false;
//Check if the code is paused and resumed
bool pause = false;
//Set up the matrices that we will need
//The two frames we will be comparing
Mat frame1, frame2;
//Their grayscale images (needed for absdiff() function)
Mat grayImage1,grayImage2;
//Resulting difference image
Mat differenceImage;
//Thresholded difference image (for use in findContours() function)
Mat thresholdImage;
//Video capture object.
VideoCapture capture;
```

13. Create an infinite while loop that is capable of looping a video feed by re-opening or constantly keeping the capture up every time the video or feed reaches the last frame. Also include the following for debugging and error catching:

```
// Check if the video has reached its last frame by adding '-1' because we are
// reading two frames from the video at a time. If this is not included, we get a
// memory error!
while(capture.get(CV_CAP_PROP_POS_FRAMES)<capture.get(CV_CAP_PROP_FRAME_COUNT)-1){
```

14. In the main, the first frame should be read then converted to grayscale for frame differencing. The second frame should be copied and converted to grayscale again for frame differencing. Also perform frame differencing with the sequential images. This will output an "intensity image"
15. Set the threshold intensity image at a given sensitivity value. Show the difference between the difference image and thresholded image in debug mode and if it is not in debug mode, destroy the window.
16. Blur the image to get rid of the image noise in order to output an intensity image.
17. Set a threshold again to receive a binary image from blur output. If debug mode is on, then show the thresholded image after it has been blurred. If it is not in the debug mode, then destroy the windows.
18. If tracking is enabled, then search for contours in our threshold image.
19. Show our captured frame and check to see if a button. Set a preferable 10 millisecond delay

for enough time for the proper operation of this program because without this, the frames will not have enough time to refresh.
20. Create a switch statement where in the following cases:
Case ESC: Exit the program.
Case “t”: Toggle Tracking.
Case “d”: Debug Mode (Show the Difference and Threshold feeds).
Case “p”: Pause/Resume the Code.
21. Lastly, release the capture and video feed before re-opening and looping again.
22. After the program has been written and capable of running with the desired functionality, test the effectiveness of the program by a series of motion detection tests.
23. Set up a grid display relative to the width and height of the feed or camera’s capture. Also measure the distance of the camera or feed from the grid display.
24. Take an object such as a ball or a toy and move it from one point to another. Record whether or not the program was able to effectively detect the movement of the object from one point to another as well as the distance of the feed from the wall.
25. Repeat step 24 for 19 more trials. The following should be recorded:
· Starting Point
· End Point
· Computer Starting Point
· Computer End Point
· Point Difference
26. Analyze the data and find whether or not the program was accurate in terms of motion and distance detection.

In addition, for the implementation of a real-time camera feed, use the VideoCapture object’s open method. The parameter passed onto the method should be 0 if a default camera is already connected to the computer system.

Cell Counting Procedures

1. Create a .cpp file in your IDE and connect it to the OpenCV library. Also include:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace cv;
using namespace std;
```

2. Create four Mats: src (source), gray (grayscale image), small (rectangle), and srcclone (a clone of the source that is used in the result). Also include the threshold value for trackbar

```
//Creation of Mats and default threshold value for the trackbar
        Mat src, gray, small, srcclone;
        int threshval = 25;
```

3. Create a function that allows us to mark-off the leukemia cells based on our trackbar (as it controls the thresholds)

```
        static void on_trackbar(int, void*)
```

4. Create a Mat object and have it grayed

```
        Mat bw = threshval < 128 ? (gray < threshval) : (gray > threshval);
//imshow( "threshold", bw );
```

5. Make two vectors to store the contour points and image information

```
        vector<Vec4i> hierarchy;
        vector<vector<Point> > contours;
```

6. Find the contours in the image

```
        findContours(bw, contours, hierarchy, CV_RETR_LIST,
        CV_CHAIN_APPROX_SIMPLE);
// Have srcclone set to be a clone of src
        srcclone = src.clone();
```

7. Create a Mini Rect called minRect to use for leukemia marking and for each consecutive rectangle

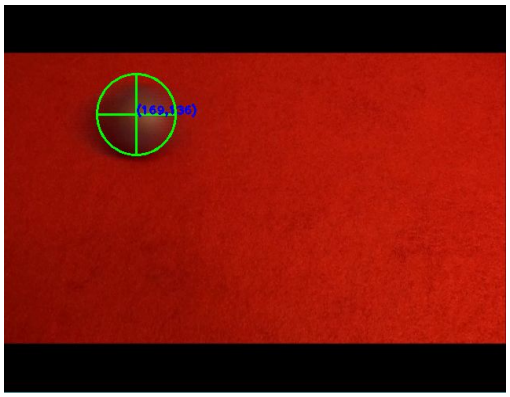
```
        Rect minRect;

// Loop for when i is less than the size of the contours Vector
        for (size_t i = 0; i < contours.size(); ++i)
// Calculate the area of each contour
```

```
        double area = contourArea(contours[i]);
// Ignore contours that are too small or too large
// if (area < 1e2 || 1e5 < area) continue;
// Set minRect to boundingRect for each of the contours
        minRect = boundingRect(Mat(contours[i]));
```
8. If there are Leukemia cells that are together, group them with a green rectangle.
```
        if (minRect.height < 20) continue;
// Create a color that is Green
        Scalar color = Scalar(0, 255, 0);
// If the rectangle .....
        if (minRect.height < 50 | minRect.height > minRect.width * 2) {
        color = Scalar(0, 0, 255);
//cout << "This is B 244";
// Mark the leukemia blood cells with a bounding rectangle
rectangle(srcclone, minRect, color, 2, 8);
        cout << i;
        cout << "\n";
// Draw each contour only for visualization purposes
// drawContours(srcclone, contours, static_cast<int>(i), Scalar(0, 0, 255), 2, 8, hierarchy, 0);
// Find the orientation of each shape
```
9. Record the thresholded image into a file named “result”
```
imwrite("result.jpg", srcclone);
        resize(srcclone, small, Size(src.cols / 2, src.rows / 2));
        imshow("Connected Components", small);
```
10. Create the main program; read the image inserted and analyze it
```
int main(int argc, const char** argv)
        src = imread("[respectivejpg]");
        if (src.empty())
        cout << "Could not read input image file: " << endl;
        return -1;
        cvtColor(src, gray, COLOR_BGR2GRAY);
// you can try different values below
        GaussianBlur(gray, gray, Size(5, 5), 1, 1);
        Mat kernel = Mat::ones(3, 3, CV_8UC1);
        dilate(gray, gray, kernel);
```
11. Take the data and bring it into a window for visual purposes.
```
        namedWindow("Connected Components", 1);
        createTrackbar("Threshold", "Connected Components", &threshval, 127, on_trackbar);
        on_trackbar(threshval, 0);
```

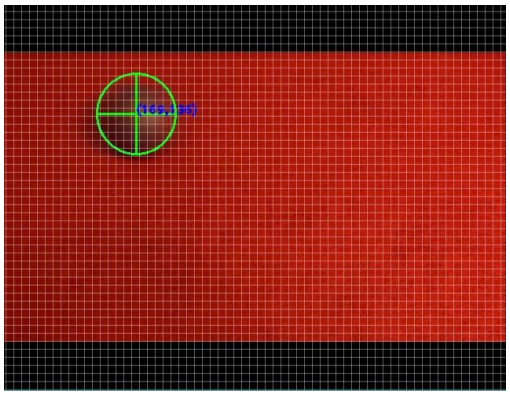
12. Analyze the data by moving the trackbar accordingly.

Data/Data Analyses/Graph of Video of Moving Ball

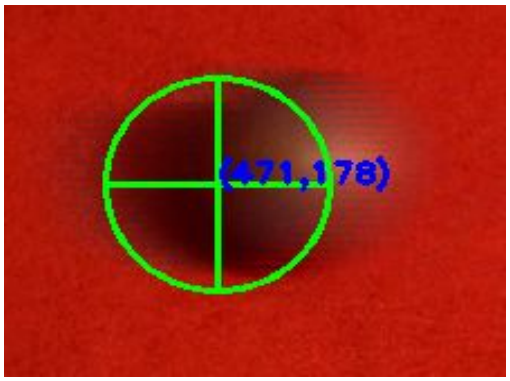
Gridless Display



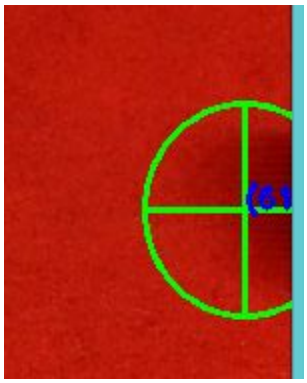
Grid Display (10px line spacing)



Ball Moving Too Fast



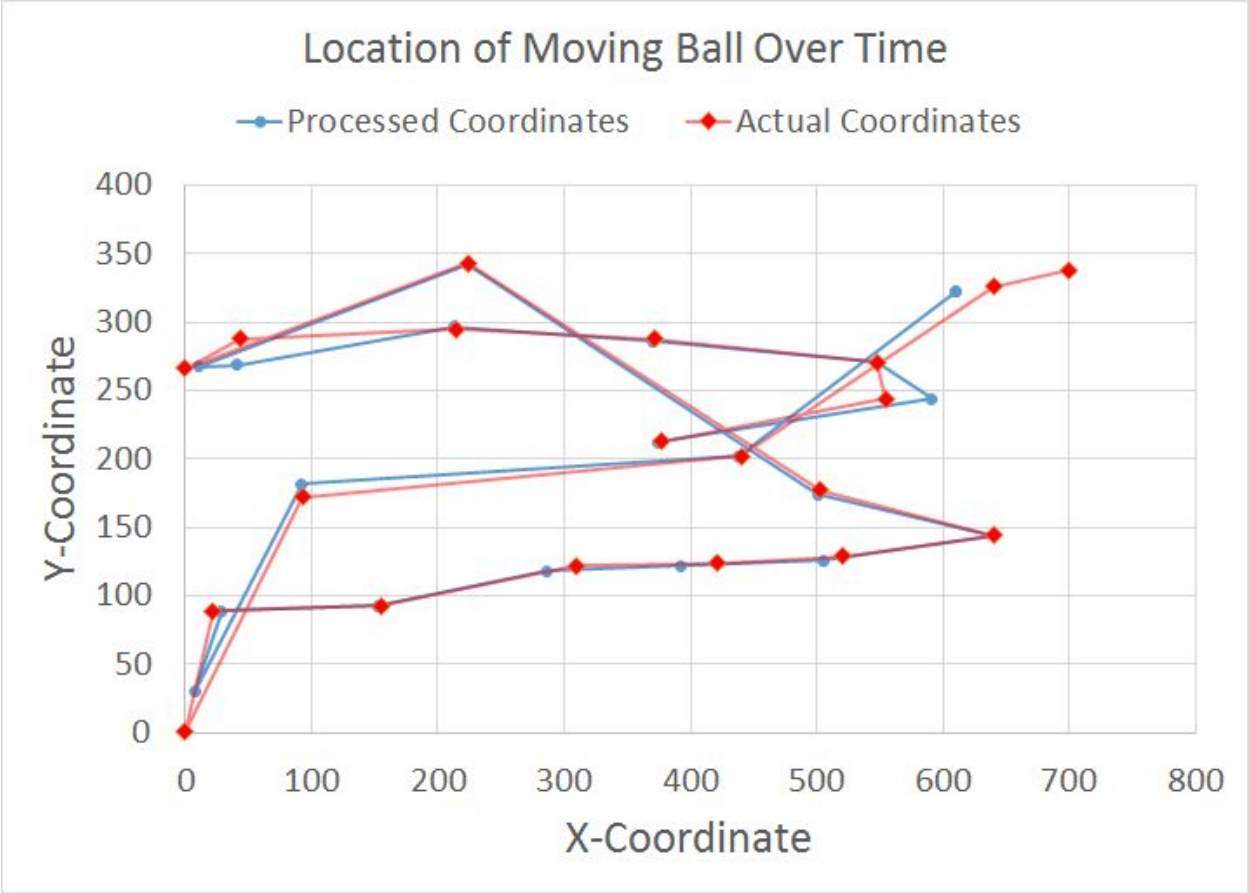
Ball Out of Range



Video Length: 19 seconds

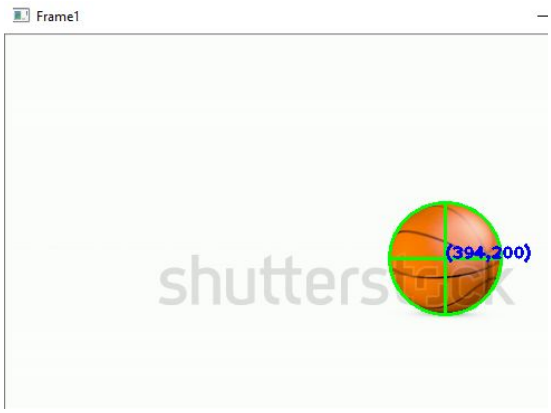
Video paused at intervals of 1 second.

Location of Moving Ball Over Time			
Time	Processed Coordinates	Actual Coordinates	Total (x,y) Difference
00:01:00	(374, 212)	(377, 213)	4
00:02:00	(591, 244)	(555, 244)	36
00:03:00	(548, 271)	(548, 271)	0
00:04:00	(370.4, 286)	(371, 288)	2.6
00:05:00	(213, 297)	(215, 295)	4
00:06:00	(41, 269)	(43, 288)	21
00:07:00	(10, 267.5)	(0, 266)	9.5
00:08:00	(224, 342)	(224, 343)	1
00:09:00	(501, 174)	(503, 177)	5
00:10:00	(640, 144)	(640, 144)	0
00:11:00	(505, 126)	(521, 129)	19
00:12:00	(393, 122)	(421, 124)	10
00:13:00	(286.5, 118)	(310, 122)	28.4
00:14:00	(153, 93)	(155, 93)	2
00:15:00	(29, 89)	(22, 89)	7
00:16:00	(7, 29.5)	(0, 1)	35.5
00:17:00	(92, 182)	(93, 172)	11
00:18:00	(438, 202)	(440, 202)	2
00:19:00	(610, 322)	(640, 326)	34
00:20:00	(610, 323)	(700, 338)	105
		Average (x,y) Difference	16.85



The line graphs of both processed coordinates and actual coordinates are very similar in where each point is located. With each coordinate having a total (x,y) difference of less 0-36 (with an outlier of 105 and an average total (x,y) difference of 16.85), it can be seen that the application created was very accurate.

Data/Data Analyses/Graph of Basketball



Normal video footage of the ball with a crosshair showing the centroid of the ball in (x,y) coordinates.

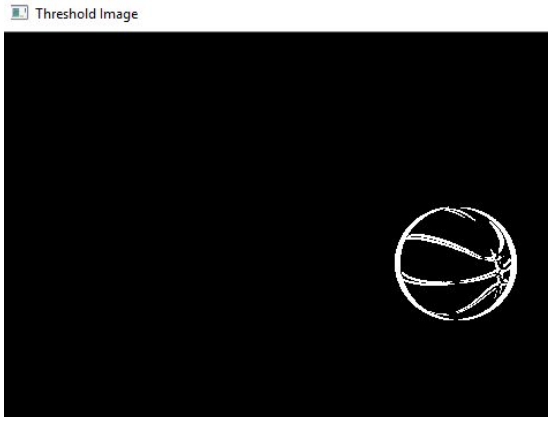


Image of the ball showing the external contours of the basketball at the point where the video was paused.

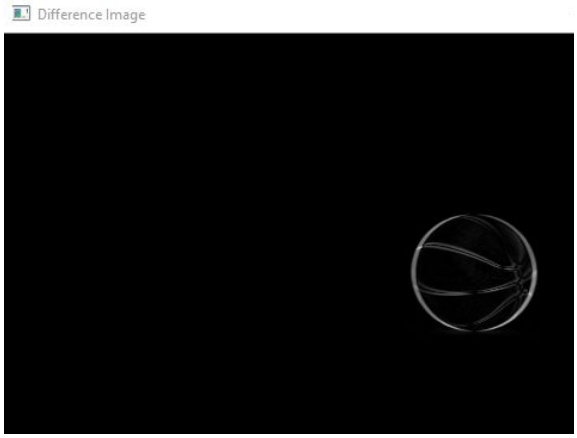


Image of the ball showing the external contours, but with the image noise removed and the contour lines defined smoothly.

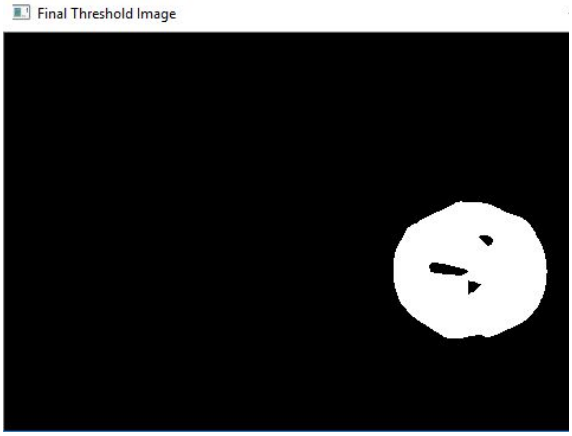


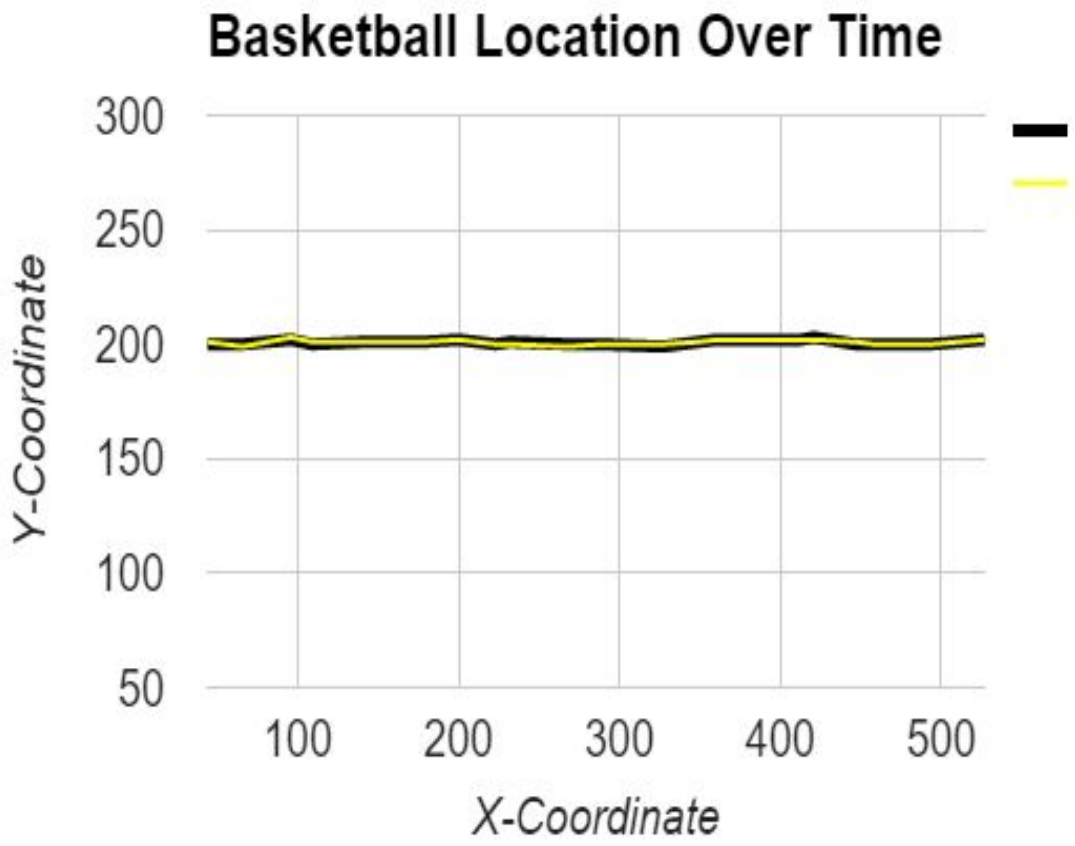
Image of the ball showing the entire image of the basketball as an external contour, recognizing the basketball as a single object.

,

Video Length: 20 seconds

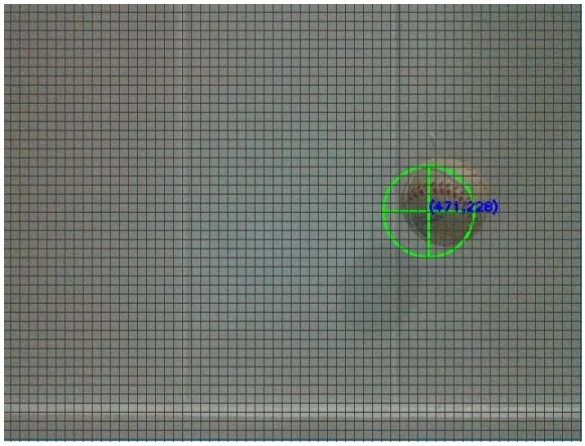
Video paused at intervals of 1 second.

Basketball Location Over Time			
Time	Processed Coordinates	Actual Coordinates	Total (x,y) Difference
00:01:00	(527, 202)	(528, 202)	1
00:02:00	(494, 200)	(494, 200)	0
00:03:00	(458, 200)	(458, 200)	0
00:04:00	(447, 200)	(446, 201)	2
00:05:00	(421, 203)	(420, 202)	2
00:06:00	(413, 202)	(413, 202)	0
00:07:00	(361, 202)	(361, 202)	0
00:08:00	(349, 201)	(349, 201)	0
00:09:00	(327, 199)	(326, 200)	2
00:10:00	(289, 200)	(289, 200)	0
00:11:00	(267, 200)	(268, 199)	2
00:12:00	(232, 201)	(232, 200)	1
00:13:00	(223, 200)	(224, 200)	1
00:14:00	(200, 202)	(200, 202)	0
00:15:00	(178, 201)	(178, 201)	0
00:16:00	(141, 201)	(141, 201)	0
00:17:00	(109, 200)	(110, 201)	2
00:18:00	(96, 202)	(97, 203)	2
00:19:00	(66, 200)	(67, 199)	2
00:20:00	(44, 200)	(45, 201)	2
Average (x,y) Difference			0.95



The location of the ball graphed on two lines is seen to be very similar when comparing the actual and processed coordinates. One reason for this may be due to the fact that this was a 2 dimensional image and thus there are no discrepancies with shadows or lighting. The basketball was also moving at a very slow pace, and thus did not prove challenging to track. This can be a model for the perfect conditions as to which the object tracking application is most successful, as both lighting and velocity (which cause the most discrepancies) are not issues.

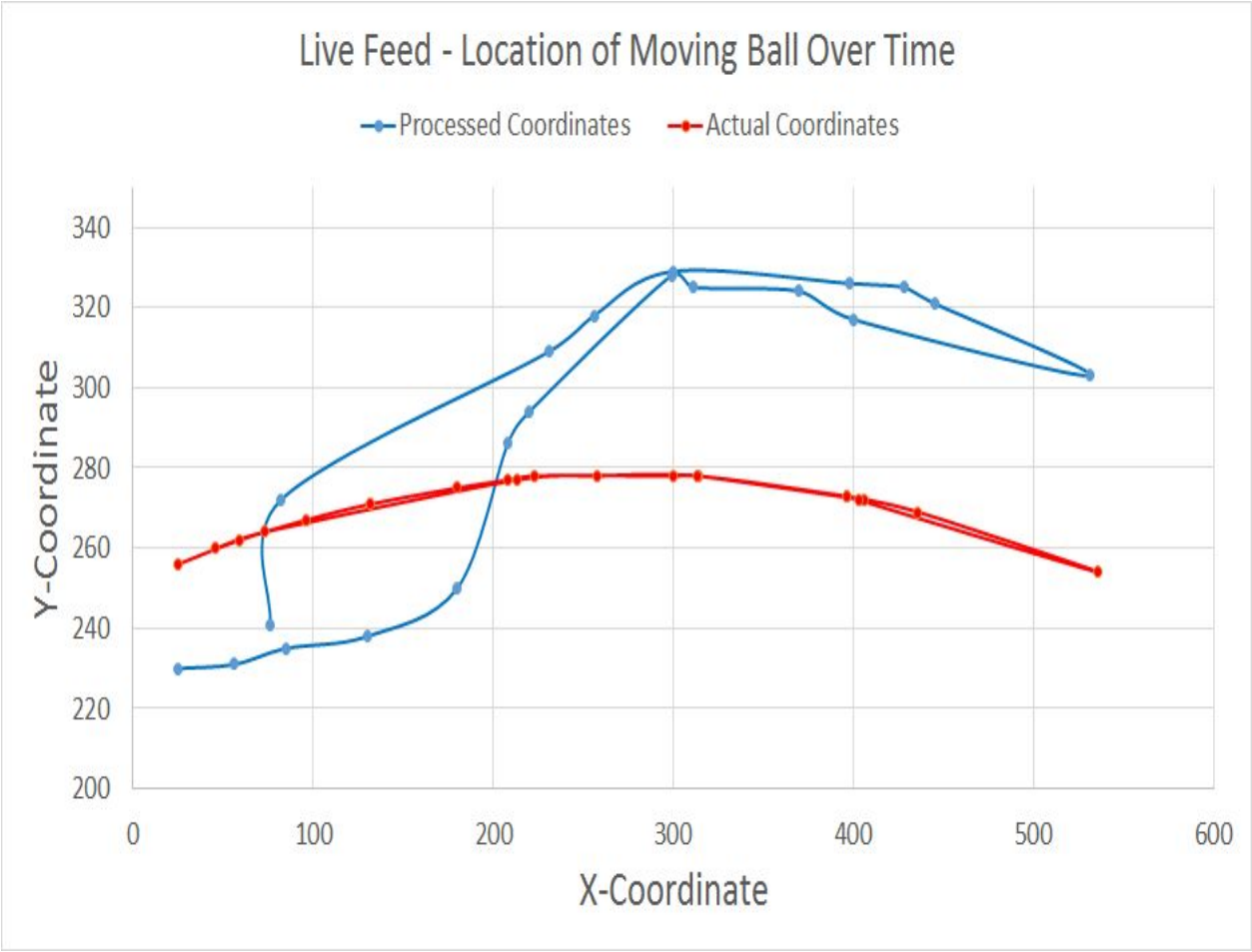
Data/Data Analyses/Graph of Live Video Feed



Video of baseball during a live-feed on how it is tracked while thrown across the camera.

Video of 20 different locations

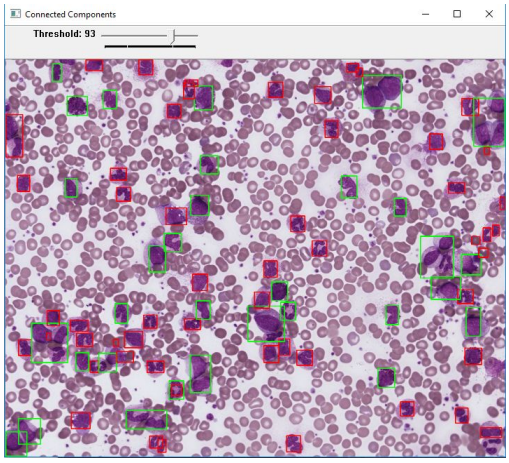
Live Feed - Location of Moving Ball Over Time			
Location #	Processed Coordinates	Actual Coordinates	Total (x,y) Difference
1	(25, 230)	(25, 256)	26
2	(56, 231)	(59, 262)	34
3	(85, 235)	(96, 267)	43
4	(130, 238)	(132, 271)	35
5	(180, 250)	(180, 275)	25
6	(208, 286)	(208, 277)	9
7	(220, 294)	(223, 278)	19
8	(299, 328)	(300, 278)	51
9	(311, 325)	(314, 278)	50
10	(370, 324)	(406, 272)	88
11	(440, 317)	(403, 272)	82
12	(531, 303)	(536, 254)	54
13	(445, 321)	(436, 269)	61
14	(428, 325)	(396, 273)	84
15	(398, 326)	(396, 273)	55
16	(300, 329)	(313, 278)	64
17	(256, 318)	(258, 278)	42
18	(231, 309)	(213, 277)	50
19	(82, 272)	(73, 264)	17
20	(76, 241)	(46, 260)	49
Average (x,y) Difference			46.90

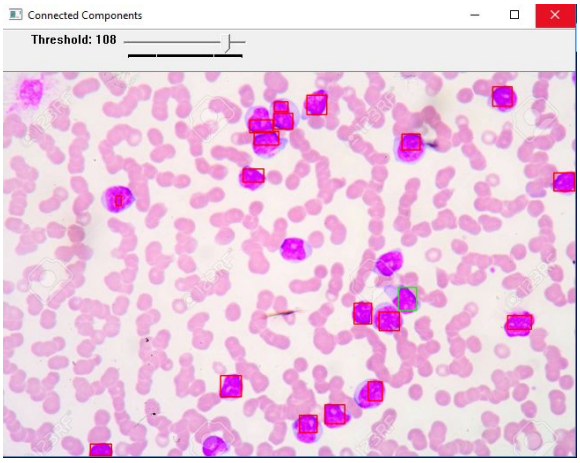


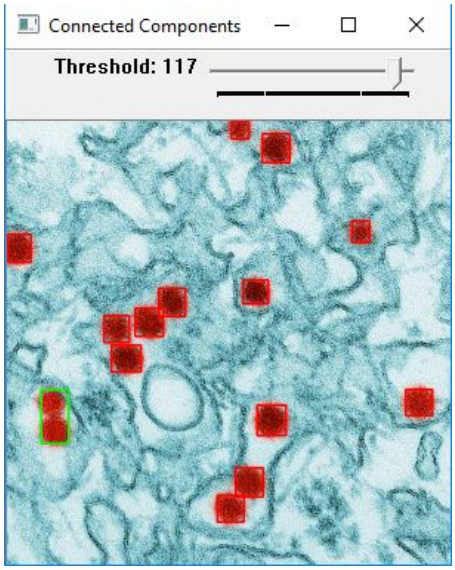
The actual location of the ball is seen to be different and very off at certain points due to the fact that the shadow of the ball is included. But the object is still able to be tracked with an approximation and is still able to follow the motion of the ball as the ball moves.

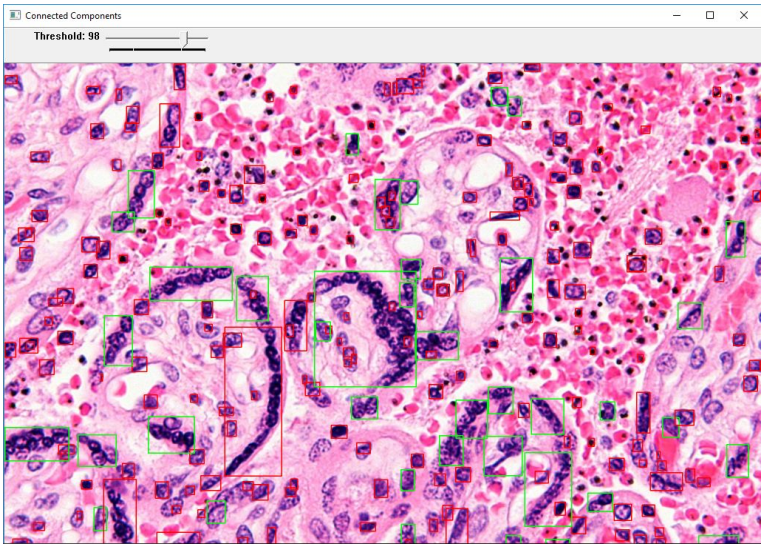
Use of Application for Leukemia Cell Count

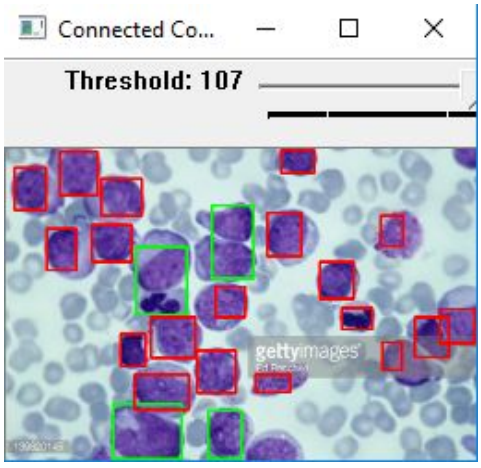
Actual Cell Number vs Processed Cell Number				
Image Number	Actual	Processed Total	Green	Red
Leukemia 1	22	21	2	19
Leukemia 2	89	86	32	54
Random Cells	27	23	4	19
Zika	15	14	1	13

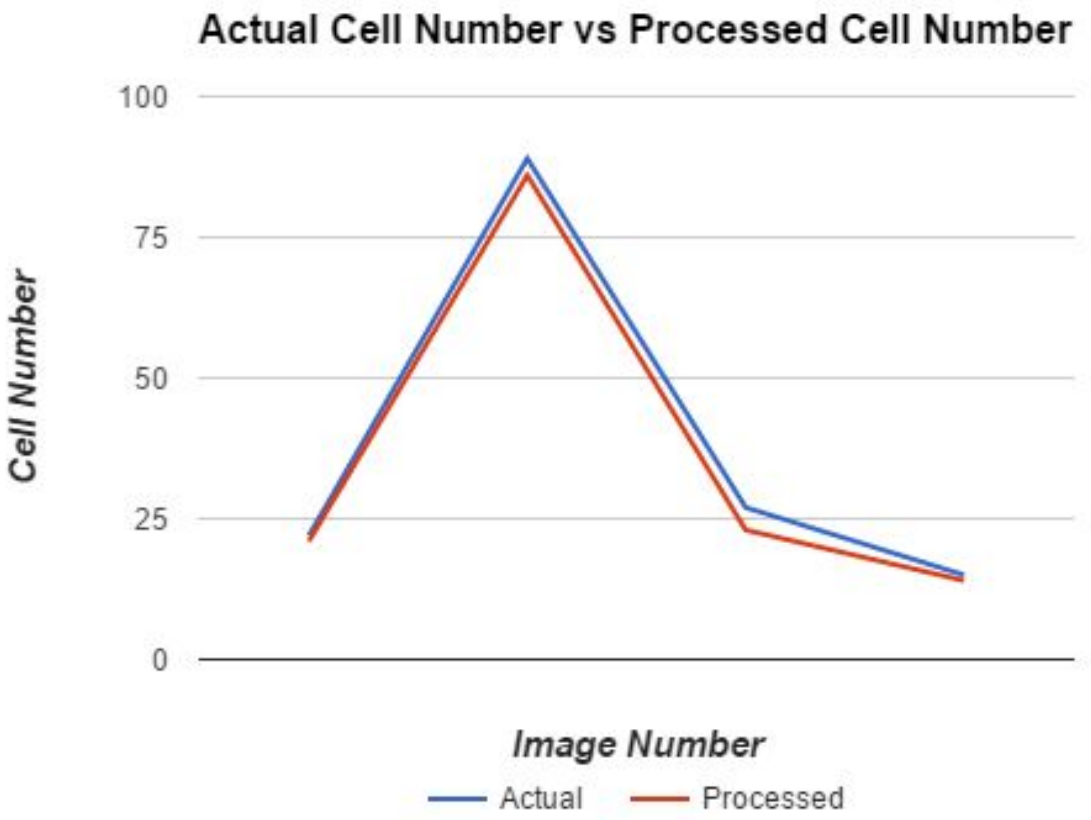

Connected Components
Threshold: 93


Connected Components
Threshold: 108


Connected Components
Threshold: 117


Connected Components
Threshold: 98


Connected Co...
Threshold: 107
gettyimages



The data collected compares the number of cells on the image that was obtained by counting by hand and the processed number of cells found by the program. Due to a limitation of not having actual cellular images found from a lab, images were found of cells and used for analysis. The number of cells counted was very similar to the processed number, as modeled by the line graph meaning this is an accurate program that is capable of counting the number of cells within the image.

Conclusion

By using Microsoft Visual Studios 2013 Update 5 along with OpenCV 2.4.10 to create AIRIS, the hypothesis was found to be supported. The idea that an application can be created to mimic the basic functions of the human eye (motion tracking and object detection) so that the application is capable of reading real time video footage and analyzing it to detect motion is held true because of the fact that the application is able to follow a moving object on footage. By putting the grid on top of the video footage, and using a difference threshold to create a final threshold (to find the differences in each frame), the application is able to pinpoint the center of the moving object and follow it. Data collection accuracy was then established by comparing the actual location of the ball on the grid and the expected location pinpointed by the application, resulting in the program being accurate but off by 2-3 points on the grid. Aside from this degree of error, the application was able to detect a moving object in the two videos that we tested. In addition, the live camera feed was able to detect any form of motion and also pinpoint the center of the moving object and follow it. Like the video-feed, it also had a 2-3 point error. With a ball tested in real-time through a live-action camera feed also had a 2-3 point error. We found that the application was able to successfully track the ball as long as it was in a closed-environment with limited background noise as the application focuses on detecting the largest object in the feed. Ultimately, despite difficulty locating the centroid of a fast moving object, it is evident that the object is still able to be tracked as seen in the final threshold image.

For the application in locating cells of Leukemia, the program was able to get a very accurate approximation in the number of cells that are within the image. Data collection was done by counting the image for Leukemia cells by hand as they would do in laboratories and comparing this to a count that is done by the application by changing the threshold value accordingly. This was done with 5 images of cells and graphed with the number of cells in respect to the image (two lines - processed vs actual.

Difficulties experienced through making the application consisted of having problems with reading and finding code in the IDE, linking OpenCV with Microsoft Visual Studios, correcting and fixing errors within the code, getting rid of image noise, and lastly fixing issues when debugging and running the video. Image noise was fixed by using blur() to get rid of the parts of the video detected by the application that didn't consist of the actual moving object. By using blur, the noise was removed and the application didn't mistake image noise with the moving object. An issue that came up when running video when debugging was that previous code and videos that were run conflicted with the current code and video. Thus CCleaner was used to clean up unnecessary background files that were still running and Avast antivirus was used for ease of access to remove firewall that conflicted with the IDE. Some difficulties for the cell application involves the fact that finding good cell images of Leukemia was not easily obtained online and the threshold value was not always accurate in detecting all cells without overlapping.

Possible errors within the application is that the application does not pinpoint the exact location of the object, instead the location is 2-3 points off. Video quality and resolution may be the reason for error stated above. For the application concerning Leukemia cells, possible errors can be that the number of rectangles created to locate Leukemia cells is sometimes overlapping. Thus, the count can have an excess of counted Leukemia cells. Errors also correspond to bias in counting cells by hand.

Works Cited

"Face Recognition: Challenges, Achievements and Future Directions." *IEEE Xplore*. N.p., n.d. Web. 22 Nov. 2015.

"Local Binary Pattern and Its Derivatives for Face Recognition." *IEEE Xplore*. N.p., n.d. Web. 22 Nov. 2015.

Szeliski, Richard. *Computer Vision: Algorithms and Applications*. Springer, 2010: n.p., n.d. Web.

"Test Article." *ScienceOpen Research* (2014): n. pag. Web.

"Welcome to Opencv Documentation!¶." *Welcome to Opencv Documentation! — OpenCV 2.4.7.0 Documentation*. N.p., n.d. Web. 21 Nov. 2015.

Documentation Works Cited

Basic Structures¶. (n.d.). Retrieved February 11, 2016, from http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/core/doc/basic_structures.html#rect

Basic Structures¶. (n.d.). Retrieved February 11, 2016, from http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/core/doc/basic_structures.html#mat

Basic Structures¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#scalar

Basic Structures¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#point

Drawing Functions¶. (n.d.). Retrieved February 11, 2016, from http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/core/doc/drawing_functions.html?highlight=circle#void circle

Drawing Functions¶. (n.d.). Retrieved February 11, 2016, from http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/core/doc/drawing_functions.html?highlight=circle#line

Drawing Functions¶. (n.d.). Retrieved February 11, 2016, from http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/core/doc/drawing_functions.html?highlight=circle#puttext

Image Filtering¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=blur#blur

Miscellaneous Image Transformations¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=cvtcolor#cvtcolor

Miscellaneous Image Transformations¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold#threshold

Operations on Arrays¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html?highlight=absdiff#absdiff

Reading and Writing Images and Video¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html?highlight=videocapture#videocapture

Structural Analysis and Shape Descriptors¶. (n.d.). Retrieved February 11, 2016, from http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours

User Interface¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=imshow#void imshow

User Interface¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=imshow#destroywindow

User Interface¶. (n.d.). Retrieved February 11, 2016, from http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=waitkey#waitkey