Embedded Vision System and Wide Gamut Cameras

Jonathan Chu

June 2011

Advisor: Gopi Meenakshisundaram

Department of Computer Science

Abstract:

Embedded Systems are ubiquitous in everyday life. There are embedded systems in cars, homes, parks, etc. This project focuses on the development of an embedded camera system using the Ebox. It is about running video processing applications through this embedded system. It also uses a touch screen interface to control the system. It has potential applications in cars such as in detecting traffic light signals and signs. Algorithms such as pulse coupled neural networks and the Hough transform have been used to show how to potentially identify signs without human intervention. Color thresholding has also been done to extrapolate regions of interest from a scene.

For the second project, I also conducted research on developing wide gamut cameras to try to capture colors in the human gamut range more effectively. The fundamental idea is to use spatial resolution to capture the spectral resolution at every pixel. In other words, many pixels in the camera will be used to represent many wavelengths representing a single pixel in the conceptual image. In the calibration step, three lasers corresponding to the red, green, and blue are used to identify the corresponding camera pixels that capture these narrow wavelength bands. Intermediate wave-lengths will be converging to intermediate camera pixels between the pixels corresponding to red, green, and blue. Once the camera-pixel to wavelength-band correspondence is established, any image can be resolved into its wavelength components, from which true wide gamut image can be synthesized. My specific project will be to resolve the exact pixel location at which the wavelength bands used in the calibration converge, using blob detection algorithm. I will then convert the values read from the camera pixels to CIE-XYZ coordinates and synthesize the image.

Table of Contents

Introduction:

People face a variety of dangers on the road while they are driving. There are often distractions that may cause drivers to be unaware of potential collisions, such as cell phones, music, movie players, etc. Also, people may have vision problems which may hinder their ability to distinguish traffic lights or signs. People with color blindness may not be able to tell red from green lights which may hinder their ability to drive around. One possible device which may assist people in driving their vehicles would be to develop a traffic vision system in the car that could read traffic lights and signs. Such a system could be able to notify the driver of potential dangers or to determine whether a driver has been neglecting important traffic signs. For example a drunk driver may not realize he is driving dangerously and a computer may be able to detect when a driver has run a red light or stop sign. For my research, I explored different computer vision techniques and embedded systems which could aid in recognizing traffic lights or signs.

For my second project, my research centered around the development of wide gamut cameras. In today's world, the cameras that one can purchase in stores may not actually represent a scene as accurately as possible. If a color in a scene has no possible representation, in an existing color space then the camera will only approximate the color to whatever it deems closest, which may or may not be appropriate. Also, monitors and television sets may not have the capability to represent colors accurately. Often times only high end laser display systems have the ability to represent every color perceived by the gamut of human vision. My research with graduate student Maxim Lazarov and visiting Prof. Robert Brown focused on novel techniques for creating more accurate spectral representations of scenes.

Background:

There has been much research done in the area of computer vision for automobile systems. For example, the DARPA Grand Challenge program was established to encourage universities to create driverless vehicles using such advanced systems as cameras, LIDAR, radar, GPS, etc. This has caused many universities to pursue research in vehicle technologies. Many computer vision experts have applied their techniques for the application of driving automobiles. Prof. Masako Omachi, from Tohoku University explored the use of normalizing color spaces and applying the Hough transform to traffic lights. He showed that it could be done quickly and accurately. Lu Kuo-Hao, from Yuan Ze University, used morphology techniques coupled with region labeling and border detection to identify traffic lights. HCS Rughooputh from the University of Mauritius, explored the use of pulse coupled neural networks to create barcodes of images of traffic signs and lights for fast and easy recognition.

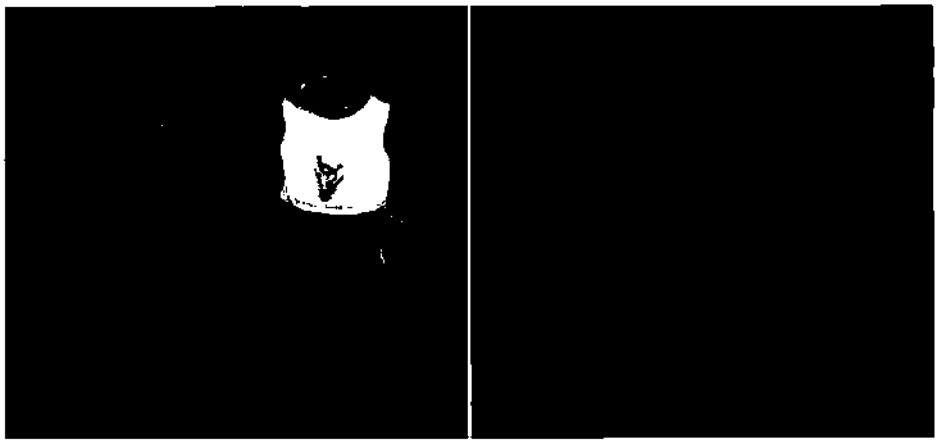
Hyperspectral camera research has been done by others as well, but most of that technology is very expensive to obtain. A hyperspectral camera can cost hundreds of thousands of dollars and are employed by such people as NASA to take images of the land and monitoring mineral or crop fields. They may also be used by the military for surveillance purposes. There are many different techniques employed by these cameras such as staring array systems or push broom systems. Researchers from Shizuoka University, explored the development of a wide gamut camera using an XYZ still camera with three optical filters. They worked on developing methods to more accurately capture and represent the tristimulus values picked up by their camera and those captured by an actual spectrometer. Currently, typical camera technology can only capture colors within the limited RGB color space. Our goal has been to create a new

camera design that would expand the gamut to include a larger area of the CIE color space. Most cameras use prisms, so instead we studied the effects of using a microlens array combined with a diffraction grating to achieve greater color fidelity.

Methodology:

For the initial application, I programmed it using C++ and the OpenCV library. I developed it with the Visual Studio environment. OpenCV stands for Open Source Computer Vision and it is a library of programming functions. It is under a BSD license and has over 2000 algorithms for computer scientists to utilize. It has been used by many successful researchers in their projects including the DARPA grand challenge winner. By using OpenCV, one does not have to spend time programming algorithms and can focus on the high level idea and algorithm. Most of the OpenCV library is in C++, so the C++ language was used with it. Visual Studio is commonly used with C++ and it was the environment most familiar to me.

When the image is read in by the program, it applies color thresholding to it so that only the regions of interest remain. For example, when detecting the circles of a traffic light, one could threshold for only green colors and thus display only the green images.

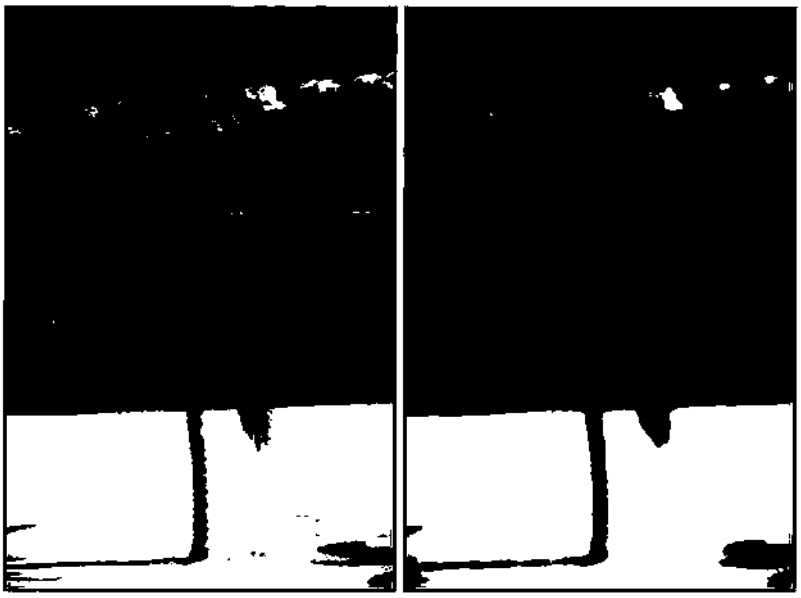


(www.societyofrobots.com)

Next a Gaussian Blur is added to the image to smooth the image of details. Since we are only interested in the overall shapes, we want to minimize interference caused by noise. The equation for a Gaussian Blur in 2 dimensions is

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where x and y are the horizontal and vertical distances from the origin respectively.



(http://pvnick.blogspot.com/2010/01/im-currently-porting-image-segmentation.html)

After that, a Hough circle transform is performed over the image. The Hough transform can find shapes within an image by employing a voting procedure. A circle can be described by the parametric equations

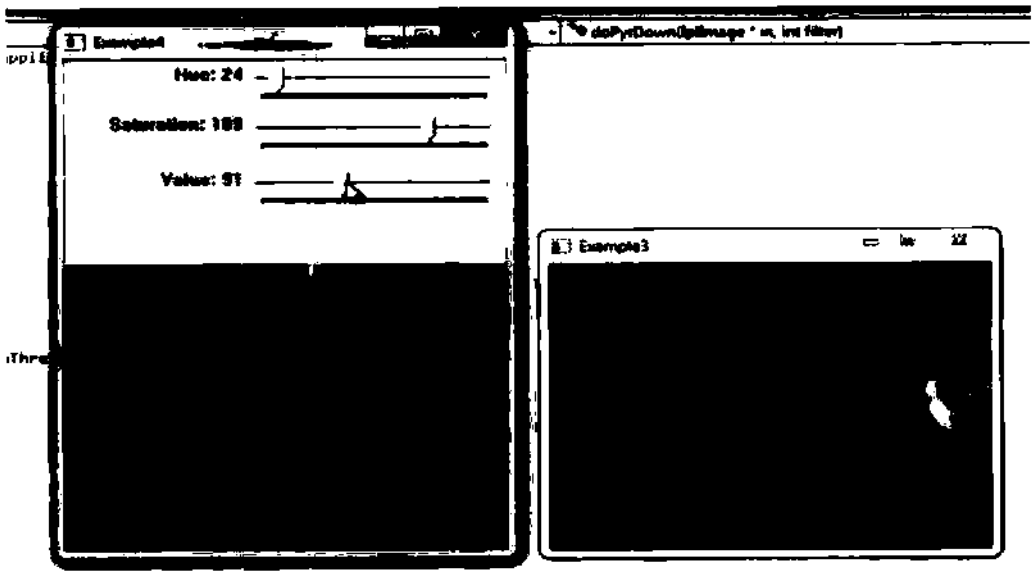
X = a + Rcos(theta)

Y = b + Rsin(theta)

In trying to find the center of the circle, the algorithm will actually look at the edges of the image and create multiple circles. The location where the circles intersect with each other the most, is considered to be the center of the real circle. It keeps count of the intersections through something called an accumulation array.

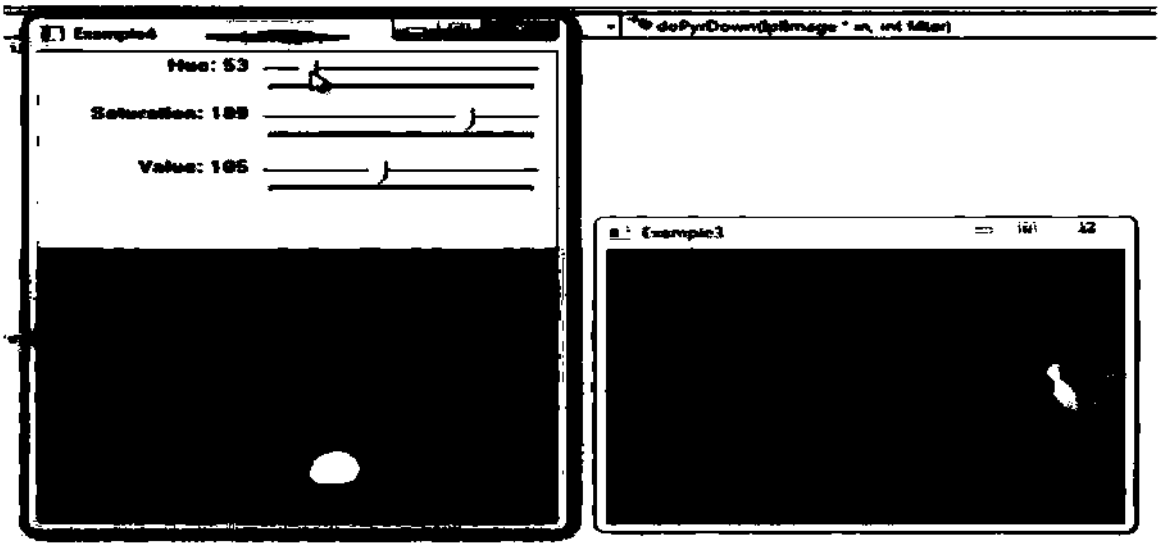
(http://hci.iwr.uni-heidelberg.de/MIP/Teaching/ip/solution06/solution6.html)

Putting all these pieces together, a software application that could take in video feed from a webcam was created that could detect circles of various colors and shapes.



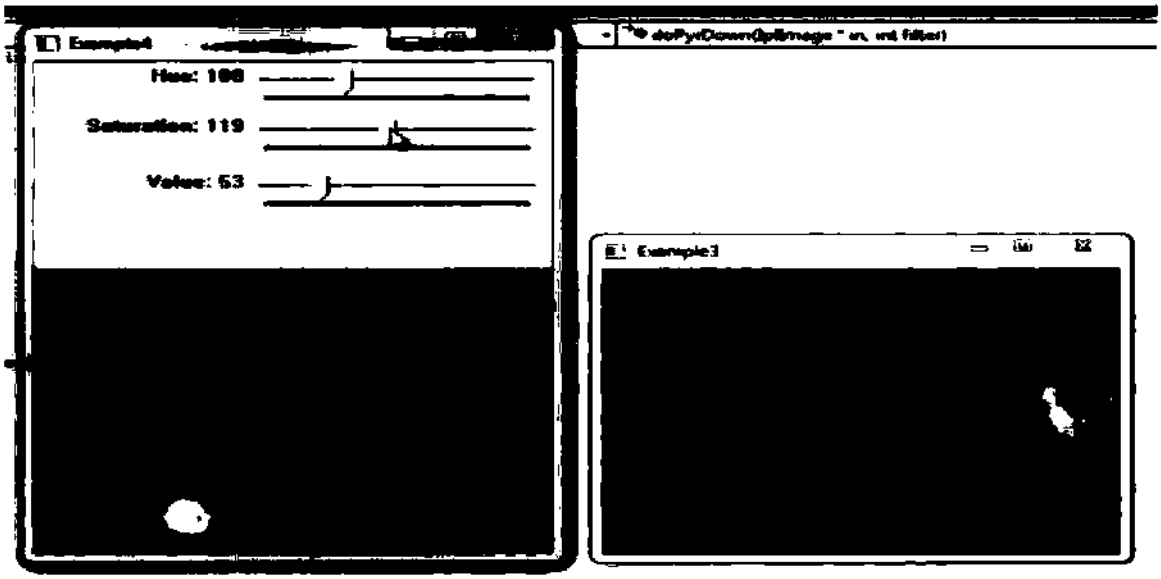
(my application running in my room, detecting red circles)

Notice the image above detected red circles using the HSV values 24, 189, 91. When the HSV values were changed to 53, 189, 105, it could detect yellow circles.



(detecting yellow circles)

When the HSV values were changed to 108, 119, and 53, it could detect green circles.

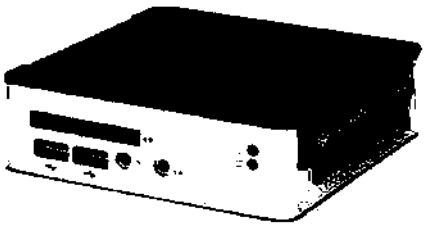


(detecting green circles)

The application shows the original image displayed on the right side. On the left side, it shows the image after applying color thresholding. The HSV values can be manipulated in real time to detect circles of varied colors. By using the program I was able to detect images that would be similar to traffic light signals. This can help serve as a proof of concept for the algorithm and its use in potentially recognizing traffic lights.

My OpenCV code:

Also, we obtained hardware to try to implement the solution with a real device. Microsoft provided us with an eBox-3300 through the Imagine Cup competition.

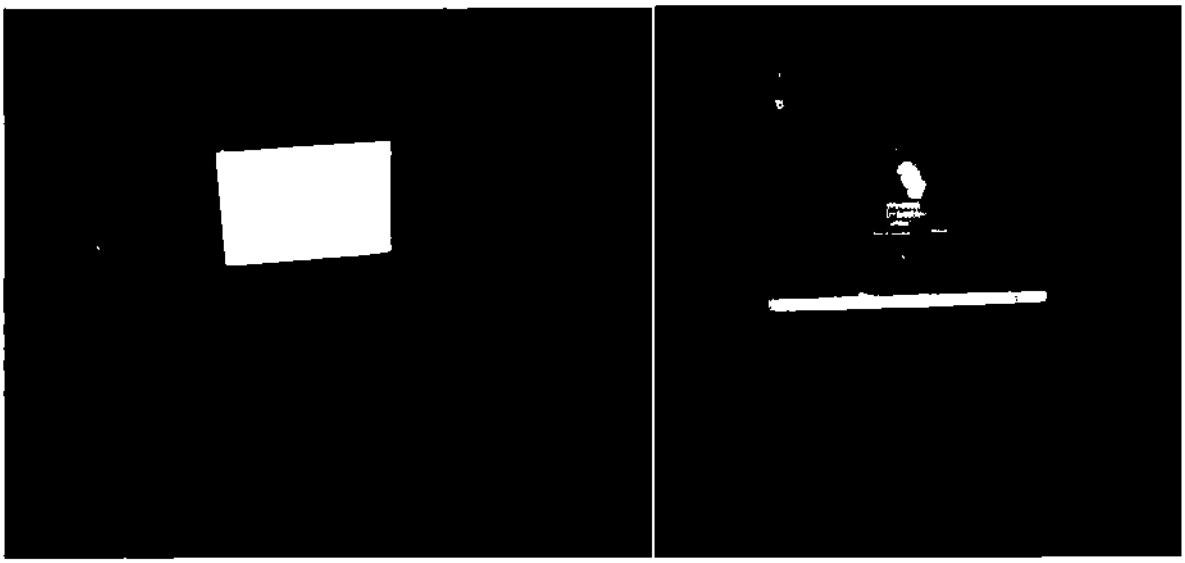


(http://www.gprs-controllers.com/ebox-3300-p-5288.html)

(http://store.mp3car.com/Lilliput_629GL_70NP_C_T_7_VGA_Touchscreen_w_Aut_p/MON-016-0002.htm)

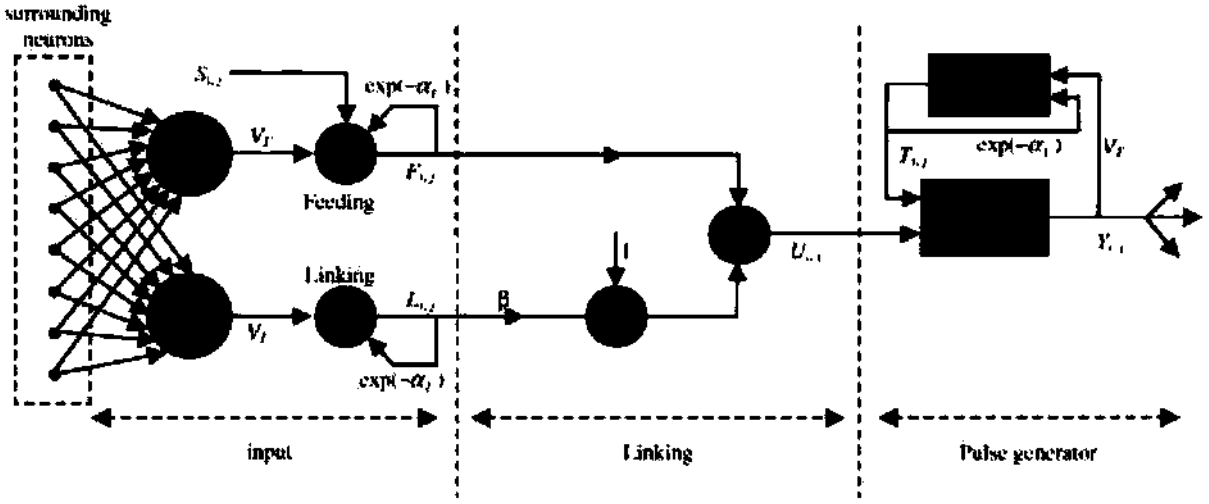
(http://reviews.cnet.com/webcams/logitech-quickcam-pro-9000/4505-6502_7-32509550.html)

The touchscreen and webcam were provided by a UROP grant. I tried to utilize the Windows Embedded Compact 7 to get the devices to work. I had to create the operating system and configure and build it in visual studio. Then I had to generate a bootloader for it and put it on a compact flash so that the ebox could run off it. Unfortunately, the new operating system did not yet support drivers for the webcam so it was not compatible to run. Future development could potentially use a different operating system such as Linux which has more drivers and support than Embedded Windows.



(Set up in my dorm room)

Another idea that was explored was the use of pulse coupled neural networks. These are neural models that are based on a cat's visual cortex and can be used in image processing. It involves the use of a simulated 2d array of neurons which are composed of two primary parts, the feeding component and the linking component. It possesses synaptic weights M and W respectively and has a decay factor. It receives the image and interprets it as an input stimulus S. It is described by 5 main equations corresponding to the feedback of the neuron, the linking item, the internal activity of the neuron, the dynamic threshold, and the pulse output of the neuron. The internal activation function will increase until it exceeds the dynamic threshold, at which point it will fire an output signal. Then the dynamic threshold will increase and the output will be fed back into the feedback function. This model does not require any training and has been proven effective in functions such as image segmentation, image denoising, and object detection among others. People have created many variations of pulse coupled neural networks such as the Eckhorn model, Intersecting Cortical model, and Parodi model; each with their own advantages and disadvantages.



(Model of pulse coupled neural network, from Zhaobin Wang, *Review of pulse-coupled neural networks*)

It was hoped that the algorithm could later be modified for use in detecting traffic signals although it has not progressed that far yet. So far the basic raw algorithm has been implemented in OpenCV. Here is the code that was done in C++.

```
CvMat* pcnn(IplImage* src){

        IplImage* img32 = cvCreateImage(cvGetSize(src), IPL_DEPTH_32F, src->nChannels);
        cvConvertScale(src, img32,1, 0);

        IplImage* img2 = cvCreateImage(cvGetSize(src), IPL_DEPTH_32F, 1);
        cvCvtColor(img32, img2, CV_BGR2GRAY);

        cvNormalize(img2, img2, 0,1, CV_MINMAX);
        CvMat matrix, *S = cvGetMat(img2, &matrix);

        float alpha_f   0.1;
        float alpha_l = 1.0;
        float alpha_t = 1.0;

        float v_f   .5,
        float v_l= 0.2;
        float v_t   20.0;
        float beta = 0.1;
        float num  100,

        float w[3,[3]   = {0.5,  1,  0.5},
                          {1 ,   0,   1},
                          {0.5,  1, 0.5},  };
        cvMat w = cvMat( 3, 3, CV_32FC1, &w );

        float m[3,][3] = {0.5,  1,  0.5},
                          {1,   0,   1},
                          {0.5,  1, 0 5 ,,
        CvMat M = cvMat( 3, 3, CV_32FC1, &m );


        CvMat* F = cvCreateMat( src->height, src->width, CV_32FC1);
```

```
for(int i=0; i < src.height; i++){
        for(int j=0; j < src.width; j++)
                cvmSet(I, i, j, 0);
        }
}

CvMat *V = cvCloneMat(I);
CvMat *U = cvCloneMat(I);

CvMat *F = cvCreateMat( src.height, src.width, cv_64FC1);
for(int i=0; i < src.height; i++)
        for(int j=0; j < src.width; j++)
                cvmSet(F, i, j, 1);
}

CvMat *Mi = cvInitMat(1);
float expf = exp(-alpha_1);
float expt = exp( alpha_1);
float expi = exp( alpha_t);


for(int i=1; i < num; i++){
        cvMat* temp = conv2(Y, Mi);
        cvConvertScale(F, F, expf);
        cvConvertScale(temp, temp, -1);
        cvAdd(I, temp, I);
        cvAdd(I, Y, F);

        cvMat* temp2 = conv2(Y, Mi);
        cvConvertScale(I, I, expt);
        cvConvertScale(temp2, temp2, -1);
        cvAdd(I, temp2, I);

        cvConvertScale(I, temp2, beta);
        cvAnd(temp2, one, temp2);
        cvMul(F, temp2, U);

        for(int m=0; m < U.height; m++)
                for(int n=0; n < U.width; n++)
                        if(cvmGet(U, m, n) > cvmGet(I, m,n)){
                                cvmSet(Y,m,n, -1);
                        }
                        else{
                                cvmSet(Y,m,n, 0);
                        }
                }
        }

        cvConvertScale(I, I, expi);
        cvConvertScale(Y, temp2, -1);
        cvAdd(I, temp2, Y);
}

return Y;
}
```

For my second project, I also had to figure out how to convert a spectral color distribution to the CIE XYZ color space. This involved first calculating the tristimulus values through

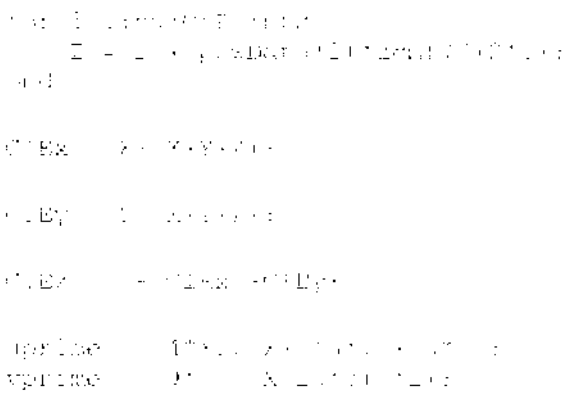
$$X = \int_0^\infty I(\lambda)\,\overline{x}(\lambda)\,d\lambda$$
$$Y = \int_0^\infty I(\lambda)\,\overline{y}(\lambda)\,d\lambda$$
$$Z = \int_0^\infty I(\lambda)\,\overline{z}(\lambda)\,d\lambda$$

Then transforming them into the xyY color space through

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z}$$
$$z = \frac{Z}{X+Y+Z} = 1 - x - y$$

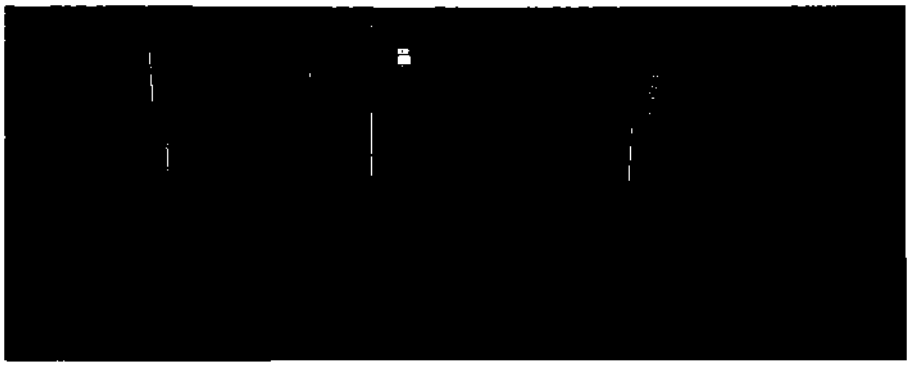
This application was created in Matlab. Here is the code.

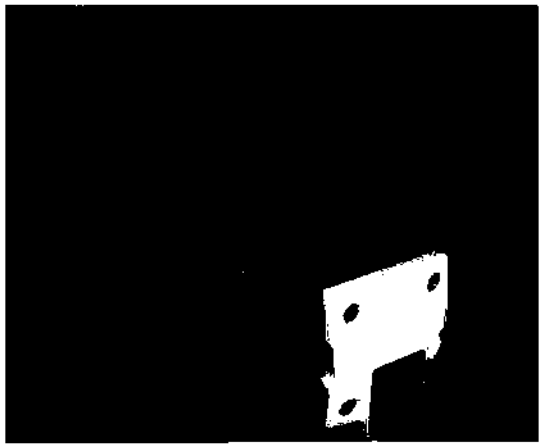
```
% Matlab code
% Jonathan Chu
clear
clc
start = [illegible];
last = [illegible];
steps = 10;
numOfPoints = [illegible];

% given points from [illegible] in steps of [illegible]
xcmf = csvread('[illegible]', [illegible]);
ycmf = csvread('[illegible]', [illegible]);
zcmf = csvread('[illegible]', [illegible]);

% reads spectral data from 4[illegible] to [illegible] in steps of [illegible]
newData = xlsread('[illegible]', [illegible]);
X = 0;
for i=1:numOfPoints
    X = X + newData([illegible])*xcmf([illegible]);
end

Y = 0;
for i=1:numOfPoints
    Y = Y + newData([illegible])*ycmf([illegible]);
end

Z = 0;
```



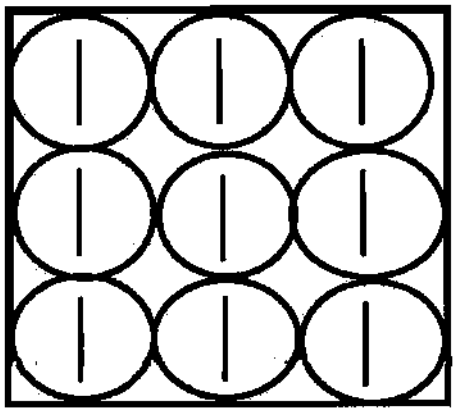
This code could take in a table of spectral data and convert it into coordinates in the CIE colorspace. This is useful so that colors can be represented in a uniform manner. There are many different types of color formats depending on what device you are using to capture or display an image. Examples are RGB, sRGB, HSV, CMYK, CIE XYZ. This color space was first created back in 1931 from experiments on the human eye run by David Wright and John Guild. Using this color space, one can represent all the values that the human eye can detect within its gamut. It provides a standard from which to represent colors and one can convert from CIE XYZ to other colorspaces.

There was a hardware component set up involving a microlens array combined with a diffraction grating. This was the prototype that was originally created by graduate student Maxim Lazarov.



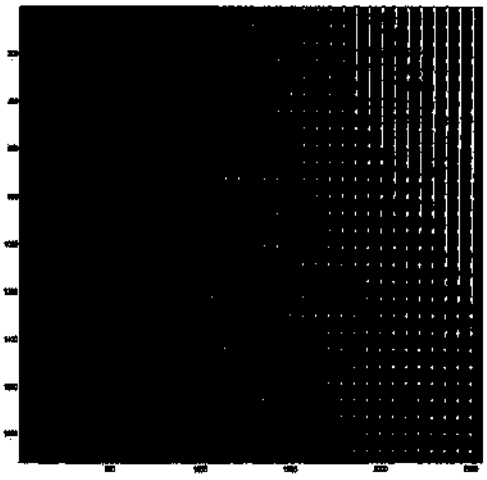


(Camera Setup)

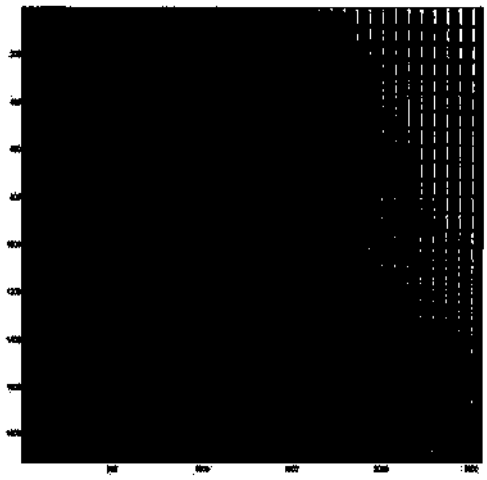


(Microlens Diagram and the spectral band for each lens)

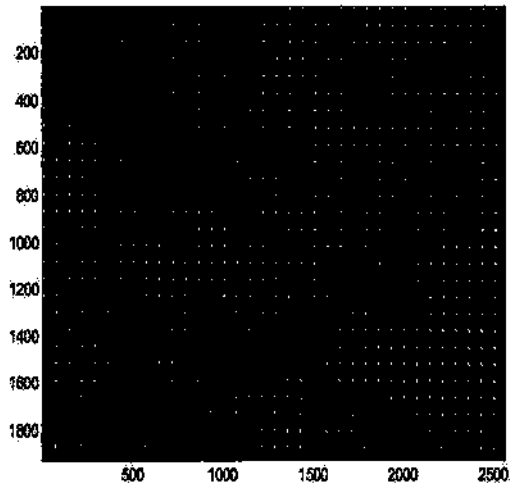
To try to calibrate the camera, we tried to examine the picture when a pure red, green, or blue laser hit the diffraction grating. This type of picture could be theoretically run through the Matlab code to determine the calibration.



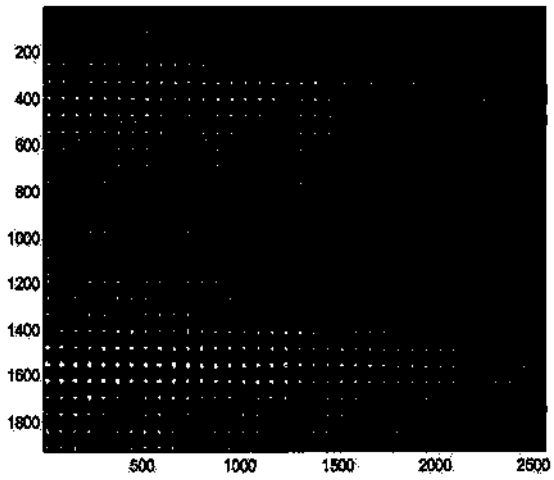
(Raw image of white halogen light)



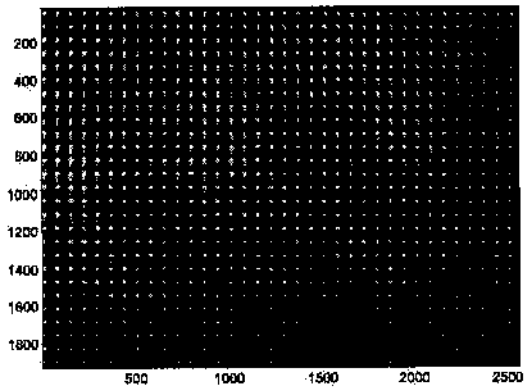
(after locating center of mass of blobs)



(Preliminary red laser image)



(Preliminary green laser image)



(Preliminary blue laser image)

The pictures did not turn out like we had wanted due to the imperfections in the lens so the next phase was not continued on.

Results:

Using the Hough color transformation combined with color thresholding was an effective way to identify circles. I could detect circles of virtually any color by setting the HSV value to it. This is a step towards an eventual system that could potentially detect traffic lights. The circular shapes of traffic signals are similar to those in the video so one would imagine that the algorithm would be effective in that domain as well if the hardware system had been completely set up. Also, the pulse coupled neural network was created and the raw algorithm works, which could also later be modified for future use. In the end parts of systems worked, although not the whole thing was put together and implemented.

Acknowledgements:

This project was funded by a UCI UROP grant. Microsoft also provided an eBox 3300 through the Imagine Cup. I thank the support of Kitty Ho and the UCI Computer Graphics group.

Bibliography:

Wang, Zhaobin, Yide Ma, Feiyan Cheng, and Lizhen Yang. Review of pulse-coupled neural networks. *Image and Vision Computing* Vol 28, Issue 1, January 2010.

Lindblad, Thomas and Jason Kinser. Image Processing using Pulse-Coupled Neural Networks. Springer, 2005.

Rhody, Harvey. (2005, Octobert 11) Lecture 10: Hough Circle Transform. Powerpoint Lecture for SIMG 782 at Rochester Institute of Technology.

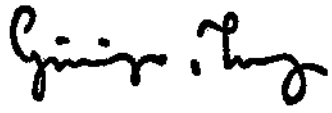
Kaster, Frederik. "Image Processing 2008/09 - Solution 6". Heidelberg Collaboratory for Image Processing. May 10 2011. http://hci.iwr.uni-heidelberg.de/MIP/Teaching/ip/solution06/solution6.html.

Bradski, Gary and Adrian Kachler. Learning OpenCV O'Reilly Media, 2008.

Walker, John. "Colour Rendering of Spectra". Fourmilab Switzerland. February 5, 2010. http://www.fourmilab.ch/documents/specrend/.

Nickerson, Paul. "Pretty fast Gaussian blur in Javascript". May 23, 2011. http://pvnick.blogspot.com/2010/01/im-currently-porting-image-segmentation.html

Palmisano, John S. "Programming-Computer Vision Tutorial, Part 3: Computer Vision Algorithms." Society of Robots. December 16, 2010. http://www.societyofrobots.com/programming_computer_vision_tutorial_pt3.shtml.



Tony Givargis