1. ABSTRACT

Object tracking in real time is one of the most important topics in the field of computer Vision. Its diversified use in Sport (e.g. Olympic race timing, mobile suspension cameras in Soccer and Rugby), The Military, Police vehicles in The U.S.A and The U.K and quite recently, its increasing popularity in the automotive world cement its viability in today's ever changing world, hence, the reason why I so much relished the learning and discovery opportunity afforded by this project. The purpose of this project was to demonstrate how a real time system for image registration and moving object detection can be used to track different objects over frames of video. The algorithm was based on describing the displacement of a point as a probability distribution, over a matrix of possible displacements. Detection and tracking of moving objects in the video scenes is the first relevant step in the information extraction in many computer vision applications. This idea can also be used for the surveillance purpose, video annotation, traffic monitoring, human-computer interaction, intelligent transportation, and robotics and also in the field of medicine. The proposed approach is demonstrated for real-time object tracking system. The techniques employed in this project involved; video streaming, Colour selection and extraction and object detection. This project uses Open CV (Open Computer Vision) routines to implement the object tracking. Open CV is a C-based program repository of code to develop a distance determination algorithm. . In this section, I discuss colour object tracking using Open CV software.

2. INTRODUCTION

2.1 AIM OF PROJECT

The basic aim of this project is to track objects over a number of frames of video. This project required Thresholding an image, find the centre of each object in the image, then associate the centres detected in multiple frames. The final phase of the project is to implement the interpolation of frames and display the original source video with key parameters overlaid on the image e.g. x marking the centre of the object and drawing a crosshair around the object. This project is programmed using the C language and the OpenCV libraries.

2.2 Tasks involved

The tasks involved in the process included the following;

- Use OpenCV system to load the frames of video into memory.
- Conversion of the RGB to the HSV model.
- Thresholding of the HSV to identify the region of interest.
- Recombination of the HSV image to return to the RGB space.
- Finding the centre point of the object.
- Associating the detected centres.
- Display of video with Overlays highlighting the position of the object in each frame.

2.3 Problems Considered

The challenges for this project majorly involved areas like image acquisition, Image tracking, color extraction, image center point detection and association, filter technique application in relation to the project, deciding on the type of camera that would best suit the job in real life, as well as the position and angle of the camera's vision. The human eye is no way near perfect vision. It is subject to parallax error due to the observer's movement relative to the object being analyzed. All of the above were put into consideration during the initial course of this project. To understand how this project works. It is very important to know how the human eye works; after all, can one collate data on and assess what one cannot see?

2.4 The Human Eye

How Does The Human Eye Work?

For my project, understanding how the eye works, especially its intake, accommodation and interpretation of images is very crucial. To get a first- hand knowledge of how the eye functions, a brief insight into the components that makes the eye such a good life camera will be pivotal. The individual components of the eye work in a manner similar to a camera. Each part plays a vital role in providing clear vision. So think of the eye as a camera with the cornea, behaving much like a lens cover. As the eye's main focusing element, the cornea takes widely diverging rays of light and bends them through the pupil, the dark, round opening in the centre of the colour iris. The iris and pupil act like the aperture of a camera.Next in line is the lens which acts like the lens in a camera, helping to

focus light to the back of the eye. Note that the lens is the part which becomes cloudy and is removed during cataract surgery to be replaced by an artificial implant nowadays.

Fig 1.1(a) and Fig1.1 (b) below is showing the similarity between the eye and the camera imaging mechanisms

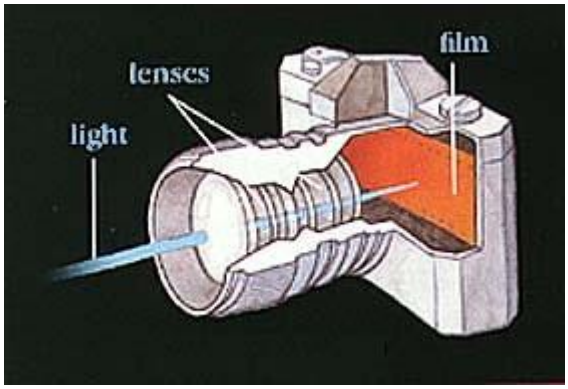


Figure 1.1(a): The Camera

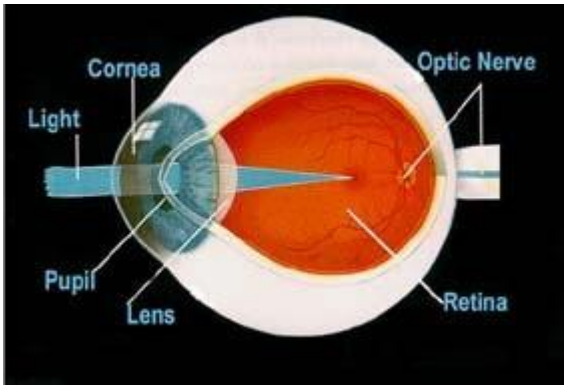


Figure 1.1(b): Human Eye

The peripheral retina is responsible for the peripheral vision, as with the camera, if the "film" is bad in the eye (i.e. the retina), no matter how good the rest of the eye is, you will not get a good picture. The human eye is remarkable. It accommodates to changing lighting conditions and focuses light rays originating from various distances from the eye. When all of the components of the eye function properly, light is converted to impulses and conveyed to the brain where an image is perceived.

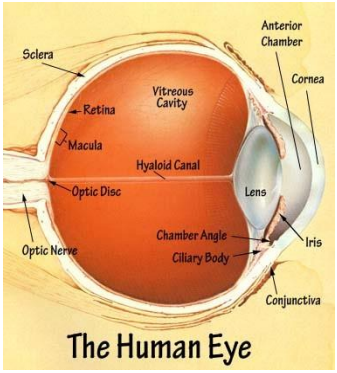


Figure 1.2: The Detailed diagram of The Human Eye displaying the focusing, transmission and accommodation peripherals of the eye.

Image processing by the eye is treated as a classical example of concatenated linear filters followed by a sampling operation. Optical imperfections and diffraction inevitably reduce image contrast in a way that may be described as low-pass spatial-filtering. If pupil diameter is less than about 2.5 mm, optical quality of the human eye for foveal vision can be nearly diffraction limited but for larger pupils, ocular aberrations limit any further improvement in retinal image quality. Light from external objects enters the eye through the pupil. The human eye has a lens and iris diaphragm, which serve similar functions to the corresponding features of a camera converts light (photons) into electro-chemical signals, which are processed by neural circuits in the retina and transmitted to the brain. The rods, located in the peripheral retina, give us our night vision, but cannot distinguish colour. Cones, located in the centre of the retina (called the macula), are not much good at night but do let us perceive colour during daylight conditions.

The links between the operational methodology employed by the eye to function so remarkably and the project are deeply entwined as in areas of motion detection (object tracking), vision perspectives (centre point association of two or more red balls), and determination of specific colours under a wide range of illumination levels (image Thresholding). Our visual system provides a motion sensor system with nearly 180 degrees horizontal coverage. The eye's peripheral vision system only supports low resolution imaging but offers an excellent ability to detect movement through a wide range of illumination levels. This motion detection has been useful to human kind for protection from aggressors and for spotting game while hunting. Likewise, the project would enhance detection of coloured objects in motion under conditions of different illumination sources which I believe have infinite real life applications.

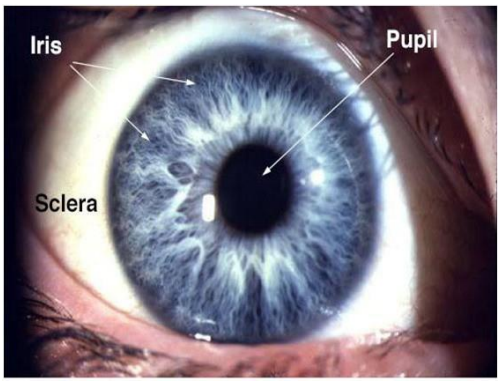


Figure 1.3: Dilated Eye, showing the Sclera, Pupil and Iris.

How do we see colour? Sensing light

We perceive colour when the different wavelengths composing white light are selectively interfered with by matter (absorbed, reflected, refracted, scattered, or diffracted) on their way to our eyes, or when a non-white distribution of light has been emitted by some system. Visible light is merely a small part of the full electromagnetic spectrum, which extends from cosmic rays at the highest energies down through gamma rays, X- rays, the ultraviolet, the visible, the infrared, and radio waves to induction-heating and electric- power-transmission frequencies at the lowest energies. Note that this is the energy per quantum (photon if in the visible range) but not the total energy; the latter is a function of the intensity in a beam. We can detect the range of light spectrum from about

400 nanometres (violet) to about 700 nanometres (red). We perceive this range of light wavelengths as a smoothly varying rainbow of colours -- the visual spectrum.

Trichromatic Colour Vision

Another topic greatly considered in this project is Trichromatic colour vision. Trichromatic colour vision is the ability of humans and some other animals to see different colours mediated by interactions among three types of colour sensing cone cells.

Each of the three types of cones in the retina of the eye contains a different type of photosensitive pigment, Each different pigment is especially sensitive to a certain wavelength of light (that is, the pigment is most likely to produce a cellular response when it is hit by a photon with the specific wavelength to which that that pigment is most sensitive). The three types of cones are L, M, and S, which have pigments that respond best of light of long (especially 560nm), medium (530nm), and short (420nm) wavelength respectively.

Since the likelihood of response of a given cone varies not only with the wavelength of the light that hits it but also with intensity, the brain would not be able to discriminate different colours if it had input from only one type of cone. Thus, interactions between at least two types of cone are necessary to produce the ability to perceive colour with at least two types of cones, the brain can compare the signals from each type and determine both the intensity and colour of the light. Trichromatic colour vision is accomplished by using combinations of cell responses.

It is estimated that each of the three cone types in the human retina can pick up about 100 different gradations, and that the brain can combine those variations such that the average human can distinguish about one million different colours. Therefore the ability to interpret and distinguish spectrums was valuable in achieving results.

3. COLOUR IMAGE PROCESSING

Various image processing terminologies and techniques that are used frequently in image processing as well as computer vision applications and have been used in this project are described in this section.

3.1 Color image processing for object detection

Colour image processing is a powerful analysis tool that is often used to simplify object identification and extraction from a scene. It is divided into two major areas; full-colour and pseudo colour processing. In full-colour, the images in question typically are acquired with a full-colour sensor, such as colour TV camera or colour scanner. Whereas Pseudo-color processing is a technique that maps each of the grey levels of a black and white image into an assigned color. This colored image, when displayed, can make the identification of certain features easier for the observer. The mappings are computationally simple and fast. This makes pseudo-color an attractive technique for use on digital image processing systems that are designed to be used in the interactive mode. For this project, these two techniques were utilized in this project to obtain an image of an object , in frames of video, to be sub-sequentially analyzed for position and distance. Also the different color spaces, comparison between them along with some prominent terms used in image processing are discussed in this section

3.2 HSV (Hue Saturation and Value)

HSV colour model decouples the intensity component from the colour-carrying information (hue and saturation) in colour image. As a result, the HSV model is an ideal tool for development of image processing algorithm based on colour descriptions that are natural and intuitive to humans, who after all are the developers and users of these algorithms.

The purpose of these models is to aid selection, comparison, and modification of colours by organizing them into a cylindrical geometry which roughly corresponds to human perception, see fig. 2.0. Both models are derived from the Cartesian RGB cube. Both models place neutral greys (that is, those colours where R = G = B) along a central vertical axis, with black at its bottom and white at its top, and push the most colourful colours to the edge of the cylinder. The angle around the axis corresponds to “hue”, the distance from the axis corresponds to “saturation”, and the distance along the axis corresponds to “lightness”, “value” or “brightness”. HSV is developed via an inverted hexagonal pyramid or “hex cone” and so is sometimes called the hex cone model,

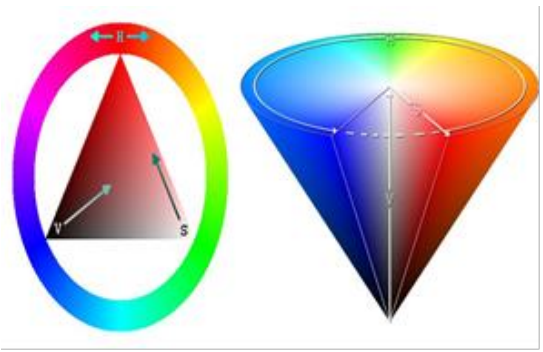


Fig. 2.0 HSV colour wheel

The conical representation (right) of the HSV model is well-suited to visualizing the entire HSV colour space as a single object. Notice that the triangle in the left image corresponds to one face of the cone cross section in the right image.

3.3 THRESHOLDING

Thresholding is defined as the simplest method of image segmentation. From the perspective of a greyscale image, Thresholding can be used to create binary images. During a Thresholding process, individual pixels in an image are marked as "object" pixels if their value is greater than some threshold value (assuming an object to be brighter than the background) and as "background" pixels otherwise. This convention is known as threshold above.

Variants include threshold below, which is opposite of threshold above; threshold inside, where a pixel is labelled "object" if its value is between two thresholds; and threshold outside, which is the opposite of threshold inside. Typically, an object pixel is given a value of "1" while a background pixel is given a value of "0". Finally, a binary image is created by colouring each pixel white or black, depending on a pixel's label.

3.4 Colour Slicing in HSV

Thresholds are chosen by highlighting a specific range of HSV values which are characteristic of the desired colour, in preference to other colours that may be present on the same frame in other to separate objects from their surroundings. The basic idea is either to

- display the colours of interest or
- use the region as a mask for further processing

3.5 Erosion and dilation in opencv

A set of operations that process images based on shapes. Morphological operations apply a *structuring element* to an input image and generate an output image.
The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e. :

- Removing noise
- Isolation of individual elements and joining disparate elements in an image.
- Finding of intensity bumps or holes in an image

Dilation and erosion is explained, using the following image as an example:

Dilation

- This operations consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle.
- The kernel B has a defined *anchor point*, usually being the center of the kernel.
- As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name *dilation*). Take as an example the image above. Applying dilation we can get:

The background (bright) dilates around the black regions of the letter.

Erosion

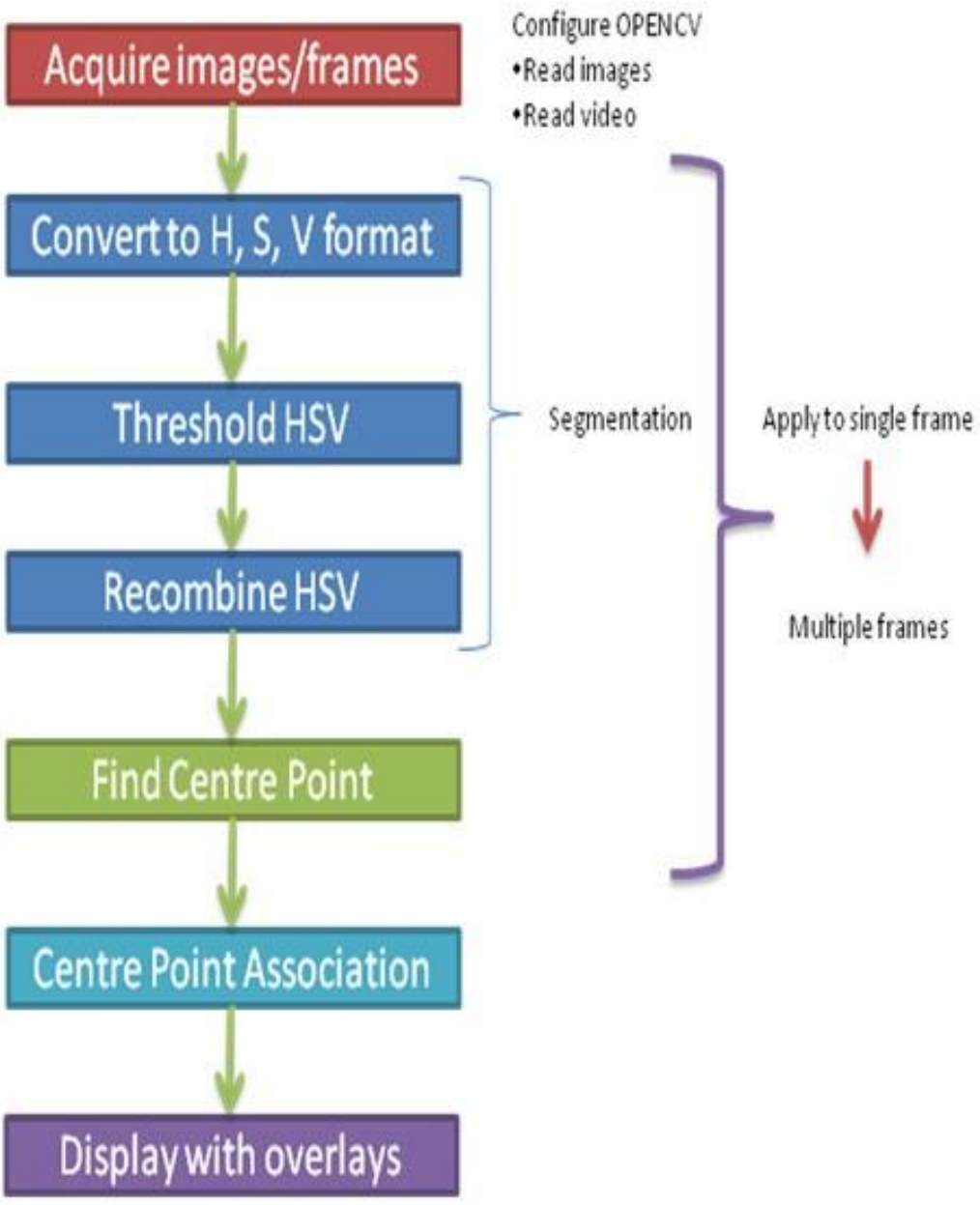
- This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel.
- As the kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.
- Analagously to the example for dilation, we can apply the erosion operator to the original image (shown above). You can see in the result below that the bright areas of the image (the background, apparently), get thinner, whereas the dark zones (the “writing”(gets bigger.

3.5 Why HSV is better than RGB and is used for object tracking?

- Color tracking and slicing is useful when you can control the color of the tracked-object or it already sticks out from the background.
- A "color space" describes how we represent a single color as a mix of different elements.
- The most common color space is RGB.
- RGB is the default color space in Processing.
- RGB is flawed for color tracking because changes in color that look small to the eye effect all three channels significantly.
- HSV is a more useful color space for tracking.
- HSV stands for Hue, Saturation, and Value.
- Hue is color. Saturation is intensity. Value is brightness.
- For color tracking we can just use the Hue channel.
- OpenCV provides functions that lets us filter an image based on a particular range of values.
- Filtering a colorful image by a range of hues reveals which hue corresponds to which color.
- Hue 0 is red and then it ranges up through orange, yellow, green, white (around 90), and up to blue.
- In OpenCV, hue ranges from 0-180.
- To track an object in live video we apply a range filter to the hue, selecting the color range of our object.
- Then we find contours in that filtered image.
- The contour with the largest area will be the object we want to track
- We can draw the bounding box of that contour.
- The center of the contour's bounding box will be the object's location.
- Thus we can successfully track an object using a HSV color space.

4. Project algorithm

From the above it was clear what needed to be done an algorithm was made and code was made according to the algorithm shown below.



5. The code

```
#include <sstream>
#include <string>
#include <iostream>
#include <opencv\highgui.h>
#include <opencv\cv.h>

using namespace cv;
//initial min and max HSV filter values.
//these will be changed using trackbars
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
//default capture width and height
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
//max number of objects to be detected in frame
const int MAX_NUM_OBJECTS=50;
//minimum and maximum object area
const int MIN_OBJECT_AREA = 20*20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;
//names that will appear at the top of each window
const string windowName = "Original Image";
const string windowName1 = "HSV Image";
const string windowName2 = "Thresholded Image";
const string windowName3 = "After Morphological Operations";
const string trackbarWindowName = "Trackbars";
void on_trackbar( int, void* )
{//This function gets called whenever a
    // trackbar position is changed
}
string intToString(int number){


    std::stringstream ss;
    ss << number;
    return ss.str();
}
void createTrackbars(){
    //create window for trackbars


    namedWindow(trackbarWindowName,0);
    //create memory to store trackbar name on window
    char TrackbarName[50];
    sprintf( TrackbarName, "H_MIN", H_MIN);
    sprintf( TrackbarName, "H_MAX", H_MAX);
    sprintf( TrackbarName, "S_MIN", S_MIN);
    sprintf( TrackbarName, "S_MAX", S_MAX);
    sprintf( TrackbarName, "V_MIN", V_MIN);
    sprintf( TrackbarName, "V_MAX", V_MAX);
    //create trackbars and insert them into window
```

```
60      createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
61      createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
62      createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
63      createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
64      createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
65      createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );
66
67
68  }
69  void drawObject(int x, int y,Mat &frame){
70
71      //use some of the openCV drawing functions to draw crosshairs
72
73
74
75      //added 'if' and 'else' statements to prevent memory errors from writing off the screen
76      circle(frame,Point(x,y),20,Scalar(0,255,0),2);
77      if(y-25>0)
78      line(frame,Point(x,y),Point(x,y-25),Scalar(0,255,0),2);
79      else line(frame,Point(x,y),Point(x,0),Scalar(0,255,0),2);
80      if(y+25<FRAME_HEIGHT)
81      line(frame,Point(x,y),Point(x,y+25),Scalar(0,255,0),2);
82      else line(frame,Point(x,y),Point(x,FRAME_HEIGHT),Scalar(0,255,0),2);
83      if(x-25>0)
84      line(frame,Point(x,y),Point(x-25,y),Scalar(0,255,0),2);
85      else line(frame,Point(x,y),Point(0,y),Scalar(0,255,0),2);
86      if(x+25<FRAME_WIDTH)
87      line(frame,Point(x,y),Point(x+25,y),Scalar(0,255,0),2);
88      else line(frame,Point(x,y),Point(FRAME_WIDTH,y),Scalar(0,255,0),2);
89
90      putText(frame,intToString(x)+","+intToString(y),Point(x,y+30),1,1,Scalar(0,255,0),2);
91
92  }
93  void morphOps(Mat &thresh){
94
95      //create structuring element that will be used to "dilate" and "erode" image.
96
```

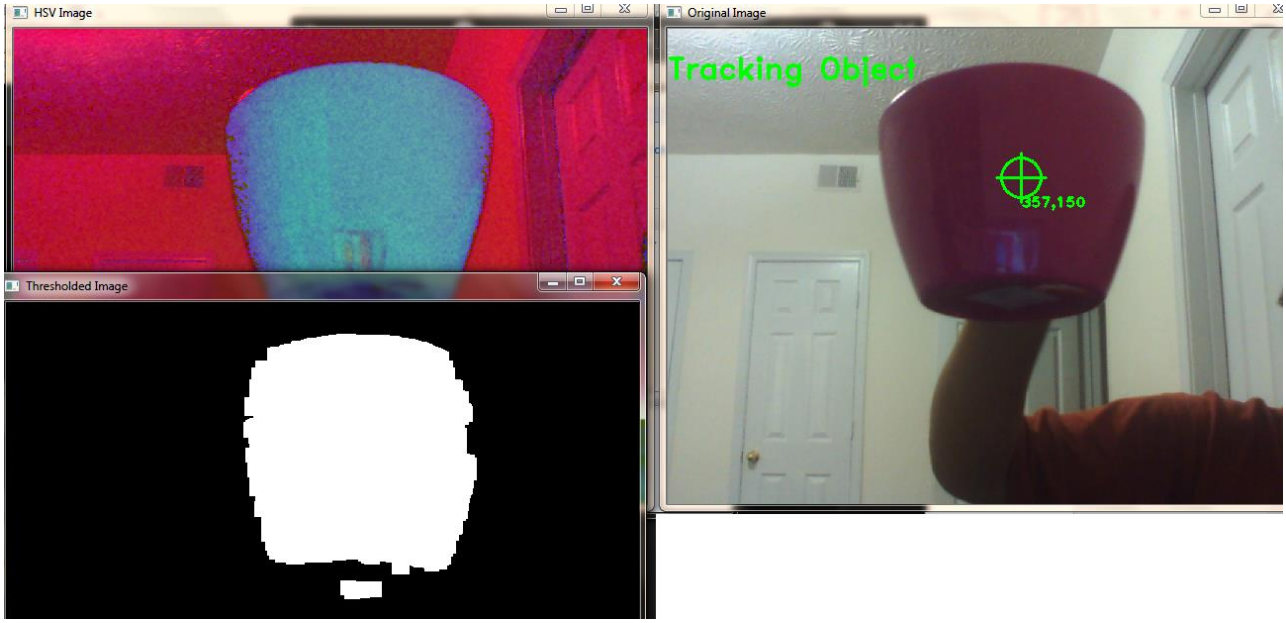
```
98      Mat erodeElement = getStructuringElement( MORPH_RECT,Size(3,3));
99      //dilate with larger element so make sure object is nicely visible
100     Mat dilateElement = getStructuringElement( MORPH_RECT,Size(8,8));
101
102     erode(thresh,thresh,erodeElement);
103     erode(thresh,thresh,erodeElement);
104
105
106     dilate(thresh,thresh,dilateElement);
107     dilate(thresh,thresh,dilateElement);
108
109
110
111 }
112 void trackFilteredObject(int &x, int &y, Mat threshold, Mat &cameraFeed){
113
114     Mat temp;
115     threshold.copyTo(temp);
116     //these two vectors needed for output of findContours
117     vector< vector<Point> > contours;
118     vector<Vec4i> hierarchy;
119     //find contours of filtered image using openCV findContours function
120     findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );
121     //use moments method to find our filtered object
```

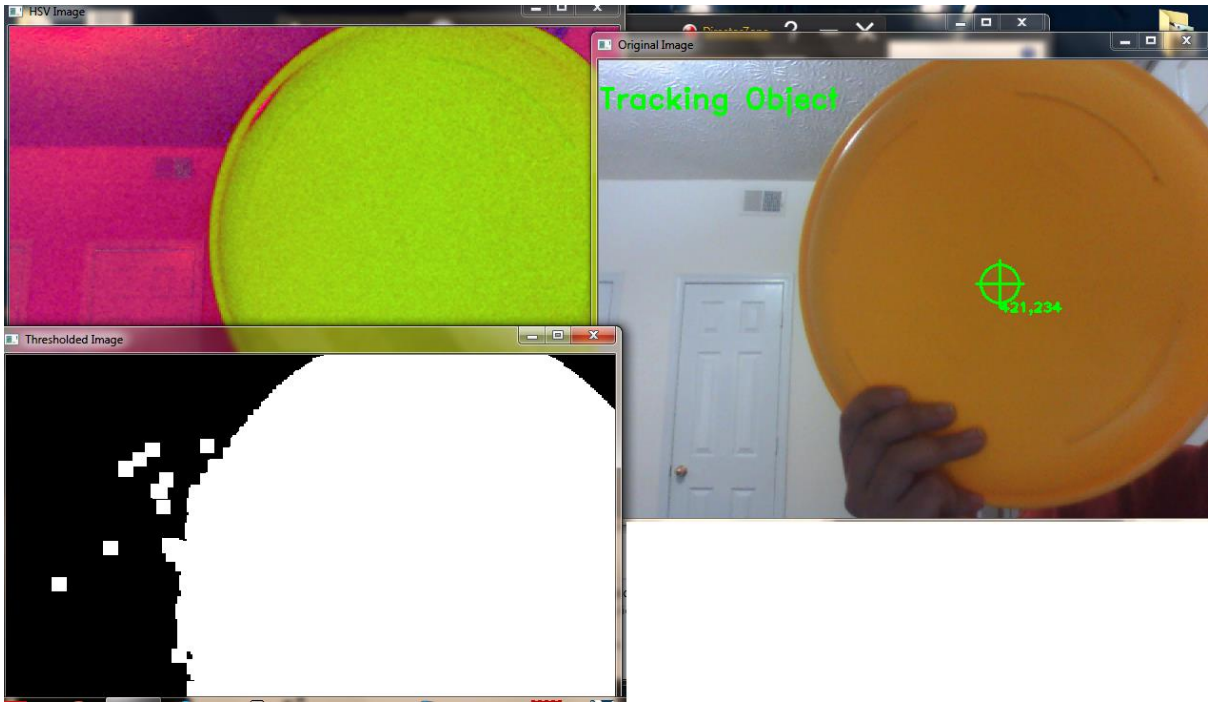
```
122         double refArea = 0;
123         bool objectFound = false;
124         if (hierarchy.size() > 0) {
125             int numObjects = hierarchy.size();
126             //if number of objects greater than MAX_NUM_OBJECTS we have a noisy filter
127             if(numObjects<MAX_NUM_OBJECTS){
128                 for (int index = 0; index >= 0; index = hierarchy[index][0]) {
129
130                     Moments moment = moments((cv::Mat)contours[index]);
131                     double area = moment.m00;
132
133                     //if the area is less than 20 px by 20px then it is probably just noise
134                     //if the area is the same as the 3/2 of the image size, probably just a bad filter
135                     //we only want the object with the largest area so we safe a reference area each
136                     //iteration and compare it to the area in the next iteration.
137                     if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea){
138                         x = moment.m10/area;
139                         y = moment.m01/area;
140                         objectFound = true;
141                         refArea = area;
142                     }else objectFound = false;
143
144
145                 }
146                 //let user know you found an object
147                 if(objectFound ==true){
148                     putText(cameraFeed,"Tracking Object",Point(0,50),2,1,Scalar(0,255,0),2);
149                     //draw object location on screen
150                     drawObject(x,y,cameraFeed);}
151
152             }else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);
153         }
154     }
155     int main(int argc, char* argv[])
156     {
157         //some boolean variables for different functionality within this program
158         bool trackObjects = true;
159         bool useMorphOps = true;
```

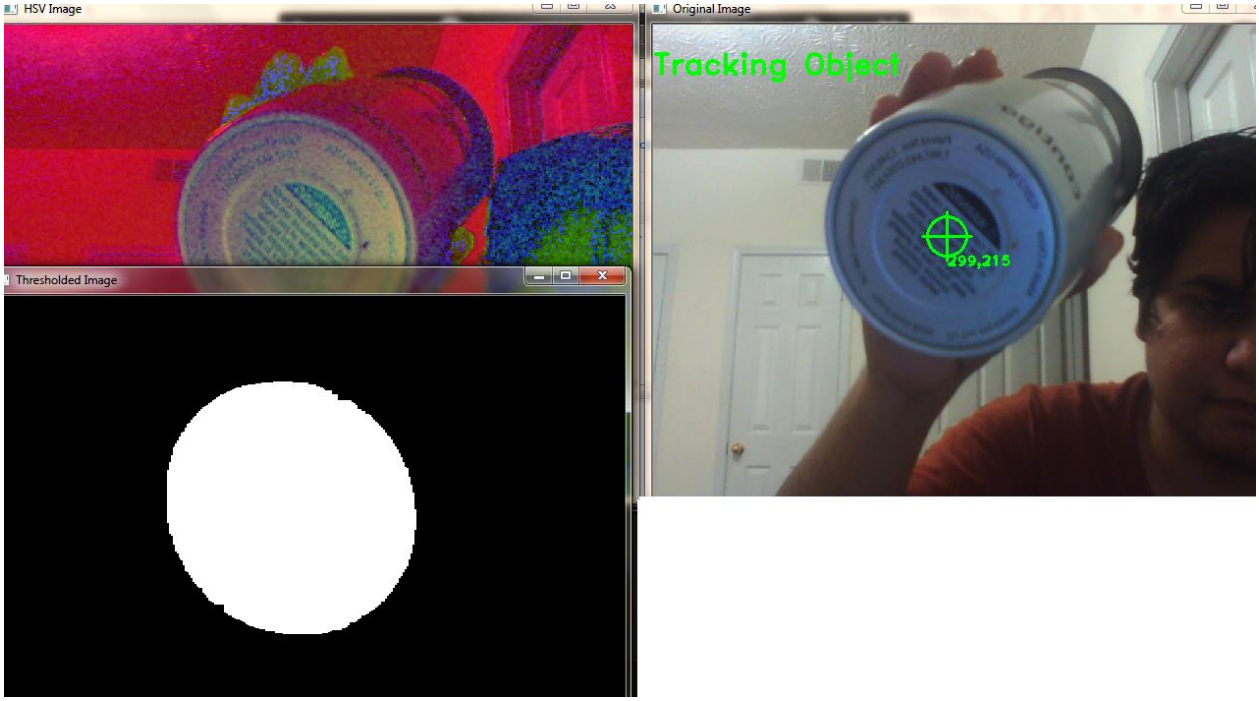
```
122         double refArea = 0;
123         bool objectFound = false;
124         if (hierarchy.size() > 0) {
125             int numObjects = hierarchy.size();
126             //if number of objects greater than MAX_NUM_OBJECTS we have a noisy filter
127             if(numObjects<MAX_NUM_OBJECTS){
128                 for (int index = 0; index >= 0; index = hierarchy[index][0]) {
129
130                     Moments moment = moments((cv::Mat)contours[index]);
131                     double area = moment.m00;
132
133                     //if the area is less than 20 px by 20px then it is probably just noise
134                     //if the area is the same as the 3/2 of the image size, probably just a bad filter
135                     //we only want the object with the largest area so we safe a reference area each
136                     //iteration and compare it to the area in the next iteration.
137                     if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea){
138                         x = moment.m10/area;
139                         y = moment.m01/area;
140                         objectFound = true;
141                         refArea = area;
142                     }else objectFound = false;
143
144
145                 }
146                 //let user know you found an object
147                 if(objectFound ==true){
148                     putText(cameraFeed,"Tracking Object",Point(0,50),2,1,Scalar(0,255,0),2);
149                     //draw object location on screen
150                     drawObject(x,y,cameraFeed);}
151
152             }else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);
153         }
154     }
155     int main(int argc, char* argv[])
156     {
157         //some boolean variables for different functionality within this program
158         bool trackObjects = true;
159         bool useMorphOps = true;
```

```
    //Matrix to store each frame of the webcam feed
    Mat cameraFeed;
    //matrix storage for HSV image
    Mat HSV;
    //matrix storage for binary threshold image
    Mat threshold;
    //x and y values for the location of the object
    int x=0, y=0;
    //create slider bars for HSV filtering
    createTrackbars();
    //video capture object to acquire webcam feed
    VideoCapture capture;
    //open capture object at location zero (default location for webcam)
    capture.open(0);
    //set height and width of capture frame
    capture.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
    capture.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
    while(1){//store image to matrix
        capture.read(cameraFeed);
        //convert frame from BGR to HSV colorspace
        cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
        //filter HSV image between values and store filtered image to threshold matrix
        inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold);
        //perform morphological operations on thresholded image to
        //eliminate noise and emphasize the filtered object(s)
        if(useMorphOps)
        morphOps(threshold);
        //pass in thresholded frame to our object tracking function
        if(trackObjects)
            trackFilteredObject(x,y,threshold,cameraFeed);
        //show frames
        imshow(windowName2,threshold);imshow(windowName,cameraFeed);
        imshow(windowName1,HSV);
        //delay 30ms so that screen can refresh.
        //image will not appear without this waitKey() command
        waitKey(30);
    }
    return 0;}
```

6. The Results







As you can see from above the pictures objects of different colours and sizes are tracked effectively using the code shown above. Also, as can be seen a crosshair with coordinates has been drawn overlaying the image of the object being tracked.

7. Porting the program on to the Ion Board.

Once the Demo Scenario was successfully implemented using OpenCV library on a CPU, the next step was to Port the scenario on to the Ion Board.

This was a challenging task as the ION Board is a standalone Motherboard with no Input or Output Devices. To overcome this challenge of porting the code on to the ION Board, it was decided to assemble the motherboard on to an existing CPU cabinet. We used DDR 3 RAMs and connected a Hard Disk and CD Rom Drive to install the drivers. The OS was loaded on to the Hard disk. We then used the OS on the hard disk and installed OpenCV and Visual Basic to run the demo scenario. The ported code ran as expected and the desired output was obtained.

The NVIDIA Ion Board 2 has a NVIDIA GeForce GPU and an Intel Celeron CPU. Thus, it is suitable for small level systems which require Graphics Processing.

The outputs obtained were consistent with the ones obtained using a CPU.

Thus the Porting of the program onto the NVIDIA Ion Board 2 was successfully completed.