FINAL YEAR PROJECT REPORT

KALMAN TRACKING FOR IMAGE PROCESSING APPLICATIONS

Student: Julius Oyeleke
Student number: 06360289

Supervisor: Dr Martin Glavin
Date: March 2010
Declaration of Originality

I declare that this thesis is my original work except where otherwise stated

Date: ______________________________

Signature: ______________________________
ACKNOWLEDGEMENTS

I would like to thank, Dermot O’Callan and especially Ronan O’Malley for their continued assistance throughout this project.

Finally, I would also like to thank my supervisor, Dr Martin Glavin, for giving me the help I needed, and for giving me the liberty to develop this project. My gratitude also goes to Dr Fearghal Morgan, for his help with some of the finer points of the project.
ABSTRACT

Real time object tracking is arguably one of the most flexible and adaptable additions to our ever omnipotent technological advancement. 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 one or more red balls 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. The tail lamps of a distant car travelling at a great speed could say a lot about the speed at which the car is travelling, the distance between the leading and trailing cars. The current position and possible future position of the leading car could be predicted with relative ease by an observer in the trailing car with a incisive knowledge of The Kalman Filter application.

The techniques employed in this project involved; video streaming, Colour selection and extraction, object detection and trajectory estimation. 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.
# TABLE OF CONTENTS

Title Page 1  
Acknowledgements 2  
Abstract 3  
Table of Contents 4

SECTION 1: INTRODUCTION 7  
 1.1 Introduction 7  
 1.1.1 Aim of Project 7  
 1.1.2 Tasks involved 9  
 1.1.3 Problems Considered 9  
 1.2 The Human Eye 10  
 1.2.1 The Eye and Colour Sensitivity 13  
 1.2.2 Trichromatic Colour Vision 14

SECTION 2: COLOUR IMAGE PROCESSING 16  
 2.1 Definition and Applications 16  
 2.1.1 HSV (Hue Saturation and Value) 16  
 2.1.2 Colour Slicing 17  
 2.1.3 Colour Modelling 19

SECTION 3: OPENCV 20  
 3.1 OpenCV 20  
 3.1.1 How OpenCV works 20  
 3.1.2 Setting up Path Environment Variable 20  
 3.1.2 Setting up Visual C++ 22  
 3.1.3 Image Acquisition 31  
 3.1.4 OpenCV Structure and Content 32
3.1.5 HighGUI

3.1.6 to Create a Window

SECTION 4: IMAGE THRESHOLDING & CENTER POINT DETECTION

4.1 Thresholding

4.2 Finding Centre Point

SECTION 5: KALMAN FILTER

5.1 Kalman Filter

5.1.1 Example Application

5.1.2 OpenCV and the Kalman Filter

SECTION 6: PROJECT STRUCTURE & EVALUATION

6.1 Project Structure

6.2 Project Evaluation

6.2.1 General View

6.2.2 Progress Report (Refer to fig 6.0 showing project flow-chart)

7.1.3 Problems Encountered

8.1 Conclusion

BIBLIOGRAPHY
SECTION 1: INTRODUCTION

1.1 Introduction

In many image processing applications, objects must be detected and tracked over a number of frames of video. One of the most common techniques used for tracking of features is Kalman filtering, this technique was used in this project to solve real time imaging problem like tracking an object in motion under conditions of different illumination sources.

1.1.1 Aim of Project

The basic aim of this project is to track one or more moving red balls over a number of frames of video (see Fig1.0). The footage of the ball will be recorded against a highly textured background. This project required Thresholding an image, find the centre of each ball in the image, then associate the centres detected in multiple frames. The final phase of the project is to implement the Kalman filter to interpolate frames where detection may not occur, and predict that path of the ball in the future as an aid to detection. The outputs of each stage should be displayed and tested thoroughly, however once the Kalman filtering has been successfully implemented, the original source video should be displayed with key parameters overlaid on the image e.g. x marking the centre of the ball, bounding box drawn around the ball, the area of the ball indicated on screen.

This project is programmed using the C language and the OpenCV libraries.
Figure 1.0: depicts a moving red ball on a black background in a frame of video.
1.1.2 Tasks involved

The tasks involved in the process included the following:

- Commission of the 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 ball.
- Associating the detected centres.
- Implementation of Kalman Filter.
- Display of video with Overlays highlighting the position of the balls in each frame.

1.1.3 Problems Considered

The challenges for this project majorly involved areas like image acquisition, Image tracking, colour extraction, image centre point detection and association, understanding and translating Kalman 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 analysed. 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?
1.2 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](image1.png)  ![Figure 1.1(b): Human Eye](image2.png)
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.

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.

The optics of the eye forms an upside-down image of those objects on the rear, inner surface of the eyeball (the retina). There, a dense carpet of light sensitive photoreceptors
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.

![Dilated Eye, showing the Sclera, Pupil and Iris.](image)

*Figure 1.3: Dilated Eye, showing the Sclera, Pupil and Iris.*
1.2.1 The Eye and Colour Sensitivity

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.
1.2.2 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 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.
Figure 1.4: Graph of Relative Absorption of Colours against Their Wavelengths.
SECTION 2: COLOUR IMAGE PROCESSING

2.1 Definition and Applications

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. For this project, these two techniques were utilized in this project to obtain an image of a red ball in motion, in frames of video, to be sub-sequentially analyzed for position and distance of the travelling red ball.

2.1.1 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. 2.1. 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,
2.1.2 Colour Slicing

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 order 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

Certain commands are used to set up the required HSV range for configuration. For this project, the following code example (shown below) was used to detect Red colour on a Black background over frame of video in preference to the other colours present:
The code below is a representation of Table 1.0 in section 8 which shows the range of HSV value used in detecting the red colour.

//Hue is measured from 0 to 255 in OpenCV,
//so has to be scaled from 360, so hue * 255/360
//Check that hue and value and saturation are within range, if they are not red, make them white

    if (!( (ptr[3*x]<=5 || ptr[3*x]>=175) && ptr[3*x+1] >= 110 && ptr[3*x+2]>=30 )) { 
        ptr[3*x]=0;
        ptr[3*x+1]=0;
        ptr[3*x+2]=255;

        //Convert back to BGR
        cvCvtColor(src,src,CV_HSV2BGR);

    }

The detail explanation of how colour slicing was used during the execution of this project is explained further in Section 5.
2.1.3 Colour Modelling

A Colour model (also called colour space or colour system) is used to facilitate the specification of colours in some standard, generally accepted way. Quite simply, colour model is a specification of co-ordinate system and a sub-space within that system where each colour is represented by a single point. Popular examples of this system are the RGB and HSV colour models, which were used extensively during the course of this project. An illustration of how these two systems were employed in this project can be found in SECTION2.
SECTION 3: OPENCV

3.1 OpenCV

OpenCV is an open source computer vision library. It is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, MATLAB, and other languages. OpenCV is very important in that it provides an easy to use computer vision facility that helps to build sophisticated vision applications quickly. It is very efficient and has applications in numerous fields e.g. factory product inspection, medical Imaging, Security, user Interface, Camera Calibration, Stereo vision and Robotics.

3.1.1 How OpenCV works

OpenCV provides utilities for reading from a large array of image file types as well as from video and cameras. These utilities are part of the HighGUI tool kit, which is included in OpenCV packages.

3.1.2 Setting up Path Environment Variable

OpenCV was installed by running the executable file, which helped to set up path environment variable to point to the OpenCV binary directory. This allowed the programs created to use the “dll” library files in that directory. This was done by going to “Start” ->
“Settings” -> “Control Panel” and selected “System”, then selected the “Advanced” tab as shown in fig. 3.0 below.

![System Properties window](image)

*Fig 3.0: step 1*

Click on “Environment Variables”, this will bring up the following pop-up window as in fig. 3.1 below:
Under “User variables”, a variable called “Path” was created, and then appended the location of the OpenCV binary directory to the path. After ensuring that OpenCV is correctly installed, sample application can be found in the folder “samples/c” such as contours.exe. Fig. 3.2

3.1.2 Setting up Visual C++

In Microsoft Visual C++ under Tools -> Options, in the pop-up window, selected “VSC++ directories”. Under the tab “Show Directories for”, chose “Include files”, add the directories in the list box that are surrounded by a red line in fig. 3.2
Library files chosen and added “C:\Program Files\OpenCV\lib” as shown in fig.3.3:
Chose Source files and directories that were added are surrounded by a red line in fig. 3.4:

A Simple “Hello World” OpenCV Program was created. In Microsoft Visual C++, dependency information was added to VC++. Select “Project- >myproject1 Properties” Fig. 3.5. Select Configuration Properties -> Linker -> Input. Go to the tab for Additional Dependencies, and add the following: cv.lib cxcore.lib highgui.lib cvaux.lib.
In the main window, the following code example was entered which shows how to use Kalman filter to track a Red ball over a frame of video (see Fig 1.0). The code describes what each section (step 0 – step 5) of the code does.

```cpp
#include "cv.h"
#include "highgui.h"
#include "math.h"
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <conio.h>
using namespace std;
void kalmanInit(CvKalman* kalman, CvMat* x_k, CvRandState* rng);
void displayImage(IplImage *src, IplImage *gray);
void keepRedColours(IplImage *src);
int main (int argc, char** argv ){

    IplImage *src = 0;
    cvNamedWindow("src",1);
```
cvNamedWindow("gray",1);
//Open AVI file to capture frame by frame

CvCapture* capture =
cvCaptureFromAVI("C:/Users/julius/Desktop/FYP/redball short.avi");
//This is the main loop. It will keep looping while there are more
frames to grab

/*
STEP 0: Initialise Kalman filter
*/

//cvCreateKalman (#state vars, #measurement vars, #control vars)
CvKalman *kalman = cvCreateKalman( 4, 2, 0 );

//Random number generator
CvRandState rng;
cvRandInit( &rng, 0, 1, -1, CV_RAND_UNI );

//Initialise state vars as random guess
CvMat *x_k = cvCreateMat( 3, 1, CV_16UC1 );
cvRandSetRange( &rng, 0, 0.1, 0 );
rng.disttype = CV_RAND_NORMAL;
cvRand( &rng, x_k );

// Transition matrix F describes model parameters at and k and k+1
const float F[] = { 1, 0, 1, 0,
0, 1, 0, 1,
0, 0, 1, 0,
0, 0, 0, 1};
memcpy( kalman->transition_matrix->data.fl, F, sizeof(F));

const float H[] = { 1, 0, 0, 0,
0, 1, 0, 0};
memcpy( kalman->measurement_matrix->data.fl, H, sizeof(H));

// Initialize other Kalman parameters
//cvSetIdentity( kalman->measurement_matrix, cvRealScalar(1) );
const float Q[] = { 0.0005, 0, 0, 0,
0, 0.0005, 0, 0,
0, 0, 0.0005, 0,
0, 0, 0, 0.0005 };
memcpy( kalman->process_noise_cov->data.fl, Q, sizeof(Q));
//cvSetIdentity( kalman->process_noise_cov, cvRealScalar(0.0005) );

//cvSetIdentity( kalman->measurement_noise_cov, cvRealScalar(0.1) );
const float R[] = { 0.1, 0,
          0, 0.1,};
memcpy( kalman->measurement_noise_cov->data.fl, R, sizeof(R));
//cvSetIdentity( kalman->error_cov_post, cvRealScalar(0) );
const float V[] = { 0, 0, 0, 0,
          0, 0, 0, 0,
          0, 0, 0, 0,
          0, 0, 0, 0,
          0, 0, 0, 0,};
memcpy( kalman->error_cov_post->data.fl, V, sizeof(V));
// Choose random initial state
cvRand( &rng, kalman->state_post );
CvMat* w_k = cvCreateMat( 2, 1, CV_16UC1 );
CvMat* z_k = cvCreateMat( 2, 1, CV_32FC1 );
cvZero( z_k );

//In C, a return value of 0 is false, and anything else is true
while (cvGrabFrame(capture)) {
    //Load frame from video
    src = cvRetrieveFrame(capture);
    int imgWidth = src->width;
    int imgHeight = src->height;
    IplImage *gray = cvCreateImage( cvSize(imgWidth, imgHeight), 8,
                                    1 );
    
    /*
     * STEP 1: Get rid of all colours that aren't red
     */
    keepRedColours(src);
    
    /*
     * STEP 2: Create gray thesholded image
     */
    //Create gray image
cvCvtColor(src, gray, CV_BGR2GRAY);
//Threshold to make the gray black

cvThreshold(gray, gray, 150, 255, CV_THRESH_BINARY);
//smooth the image to reduce unnecessary results

cvSmooth(gray, gray, CV_GAUSSIAN, 9, 9);

/*
* STEP 3: Find circles from the gray image
*/
CvSeq* circles = cvHoughCircles(gray, cvCreateMemStorage(0),
CV_HOUGH_GRADIENT, 2, gray->height/6, 3);
//output number of the circle detected
printf("%d circles detected\n", circles->total);

/*
* STEP 4: Draw a black circle where circle has been detected
*/
//Create arrays to hold the x and y co-ordinates of the centers
//of the circles
int *px = new int[circles->total];
int *py = new int[circles->total];
//loop through all the circles and draw a circle where circle
is detected
//for( int i = 0; i < circles->total; i++ ){
if (circles->total > 0) {
    int i = 0;
    float* p = (float*)cvGetSeqElem(circles, i);

    //px[i] is the X co-ordinate of the center
    //py[i] is the Y co-ordinate of the center
    px[i]=cvRound(p[0]);
    py[i]=cvRound(p[1]);
    int radius = cvRound(p[2]);

    // Input Measurement (z_k) (x position, y position, radius)
    int measurement[] = { px[i], py[i] };
    //cvInitMatHeader(z_k, 2, 1, CV_16UC1, measurement);
    memcpy(z_k->data.fl, measurement, sizeof(measurement));
// Correct Kalman filter state
cvKalmanCorrect( kalman, z_k );

float measured_x = z_k->data.fl[0];
float measured_y = z_k->data.fl[1];
printf ("z_k: %f, %f", measured_x, measured_y);

// Predict point position
const CvMat* y_k = cvKalmanPredict( kalman, 0 );

// Draw circle
float predicted_x = y_k->data.fl[0];
float predicted_y = y_k->data.fl[1];
cvCircle( src, cvPoint(px[i],py[i]), radius,
CV_RGB(0,0,0), 1, 8, 0 );
cvCircle( src, cvPoint(predicted_x, predicted_y), radius,
CV_RGB(0,255,0), 1, 8, 0 );
printf ("Circle %d, measured(x,y): %d, %d,
predicted(x,y): %f, %f
", i, px[i], py[i], predicted_x, predicted_y);
}

/*
* STEP 5: Display the image and the gray image
*/
// Show src
cvShowImage("src",src);
// Show gray
cvShowImage("gray",gray);

if (cvWaitKey(25) == 27){
    break;
}

delete px;
delete py;
}
// release all windows
cvDestroyAllWindows();
void keepRedColours(IplImage *src) {
    int imgWidth = src->width;
    int imgHeight = src->height;
    //Convert from BGR to HSV
    cvCvtColor(src, src, CV_BGR2HSV);

    //Set all non-red colours to white
    for (int y = 0; y < imgHeight; y++)
    {
        uchar* ptr = (uchar*) (src->imageData + y * src->widthStep);
        //printf("H: %d, S: %d, V: %d\n", ptr[0], ptr[1], ptr[2]);

        for (int x = 0; x < imgWidth; x++)
        {
            //Hue is measured from 0 to 255 in OpenCV,
            //so has to be scaled from 360, so hue * 255/360
            //Check that hue and value and saturation are within
            range, if they are not red, make them white
            if (!((ptr[3 * x] <= 5 || ptr[3 * x] >= 175) && ptr[3 * x + 1] >= 110 && ptr[3 * x + 2] >= 30))
            {
                ptr[3 * x] = 0;
                ptr[3 * x + 1] = 0;
                ptr[3 * x + 2] = 255;
            }
        }
    }
    //Convert back to BGR
    cvCvtColor(src, src, CV_HSV2BGR);
}

An image of a moving red ball on a black background in a frame of video in Fig. 1.0 was displayed.
3.1.3 Image Acquisition

Image Acquisition involves configuring OpenCV to read Images and video. Computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal. A new representation might mean turning a colour image into a greyscale image or removing camera motion from an image sequence. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. One of OpenCV goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. OpenCV is popular around the world, with large user communities in China, Japan, Russia, Europe, and Israel.
### 3.1.4 OpenCV Structure and Content

OpenCV is broadly structured into five main components, four of which are shown in Figure 4.1. The CV component contains the basic image processing and higher-level computer vision algorithms; ML is the machine learning library, which includes many statistical classifiers and clustering tools. HighGUI contains I/O routines and functions for storing and loading video and images, and Cxcore contains the basic data structures and content.

![Figure: 4.6 the basic structure of OpenCV](image)

Some of these utilities were used to create a simple program that opens and displays an image of moving red ball over frames of video on the screen. This is stated below:
IplImage *src = 0;
// this function loads the image.*

cvNamedWindow("src",1);
// opens a window on the screen that can contain and display an image

cvNamedWindow("gray",1);
// Open AVI file to capture frame by frame

CvCapture* capture = cvCaptureFromAVI("C:/redball short.avi");
    // the function takes the name of the AVI file to be loaded and then returns a pointer to a CvCapture structure

    // This is the main loop. It will keep looping while there are more frames to grab
3.1.5 HighGUI

“High-level graphical user interface” is a portable graphics tool kit and an OpenCV functions library that allow us to interact with the operating system, the file system, and hardware such as cameras. HighGUI allows us to open windows, to display images, to read and write graphics-related files (both images and video), and to handle simple mouse, pointer, and keyboard events. It can also be used to create other useful doodads like sliders and then add them to our windows. The HighGUI library in OpenCV can be divided into three parts:

- The hardware part.
- The file system part.
- The GUI part.

The hardware part is primarily concerned with the operation of cameras. In most operating systems, interaction with a camera is a tedious and painful task. HighGUI allows an easy way to query a camera and retrieve the latest image from the camera. It hides all of the nasty stuff and that keeps us happy.

The file system part is concerned primarily with loading and saving images. One nice feature of the library is that it allowed me to read video using the same methods. It was used to read a camera. One can therefore abstract oneself away from the particular device one is using and get on with writing interesting code. In a similar spirit, HighGUI provides a (relatively) universal pair of functions to load and save still images. These functions simply rely on the filename extension and automatically handle all of the decoding or encoding that is necessary.
3.1.6 to Create a Window

The first is to show an image on the screen using HighGUI. The function that does this is cvNamedWindow(). The function expects a name for the new window and one flag. The name appears at the top of the window, and the name was also used as a handle for the window that can be passed to other HighGUI functions. The flag indicates if the window should auto size itself to fit an image we put into it. Here is the full prototype:

```c
int cvNamedWindow
(
const char* name,
int flags = CV_WINDOW_AUTOSIZE
);
```

When window is no longer needed, one can use cvDestroyWindow()

For this project, the following commands were used.

- CvKalman
- CvMat
- CvRandState
- IplImage
- CvNamedWindow
- CvCapture
- CvSmooth
CvCapture:

For this project, several functions were considered, including how to read and write video files, and how to actually play back much files on the screen. The first thing used was the CvCapture device. This structure contained the information needed for reading frames from a camera or video file. Depending on the source, one can use one of two different calls to create and initialize a CvCapture structure.

```c
CvCapture* cvCreateFileCapture( const char* filename );

CvCapture* cvCreateCameraCapture( int index );
```

cvCreateFileCapture() was used to simply give a filename for an AVI file and OpenCV opened the file and prepared to read it. The file opened successfully and I was able to start reading frames, a pointer to an initialized CvCapture structure was returned.

```c
IplImage *src = 0;
// determines the file format to be loaded based on the file name

cvNamedWindow("src",1);
// opens a window on the screen that can contain and display an image

cvNamedWindow("gray",1);
//Open AVI file to capture frame by frame

CvCapture*capture =
cvCaptureFromAVI("C:/Users/julius/Desktop/FYP/redball short.avi");
// takes the name of the AVI file to be loaded
//This is the main loop. It will keep looping while there are more frames to grab
```
4.1 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.

For this project the codes below were used to platform Image Thresholding (see fig 4.1 for an example of Image Thresholding).
/*
 * STEP 5: Display the image and the gray image
 */
//Show src
cvShowImage("src",src); // show a named window that already exist

//Show gray
cvShowImage("gray",gray);

if (cvWaitKey(25) == 27) // ask the program to stop and wait for a keystroke
{
    break;
}

delete px;
delete py;

//release all windows
cvDestroyAllWindows(); // closes all windows and de-allocates the associated memory
}

void keepRedColours(IplImage *src) {
    int imgWidth = src->width;
    int imgHeight = src->height;
    //Convert from BGR to HSV
    cvCvtColor(src,src,CV_BGR2HSV);
    //Set all non-red colours to white
    for (int y=0; y<imgHeight; y++) {
        uchar* ptr = (uchar*) ( src->imageData + y * src->widthStep );
        //printf("H: %d, S: %d, V: %d\n", ptr[0], ptr[1], ptr[2]);
        for (int x=0; x<imgWidth; x++) {
            //Hue is measured from 0 to 255 in OpenCV,
            //so has to be scaled from 360, so hue * 255/360
            //Check that hue and value and saturation are within range, if they are not red, make them white
            if (!( ptr[3*x]<=5 || ptr[3*x]>=175 ) && ptr[3*x+1] >= 110 && ptr[3*x+2]>=30 )) {
                ptr[3*x]=0;
                ptr[3*x+1]=0;
            }
        }
    }
}
ptr[3*x+2]=255;
}
}

// Convert back to BGR
cvCvtColor(src, src, CV_HSV2BGR);
Fig 4.0 Original Image of Red ball on Blue Background.

Fig 4.1 Shows extraction of the blue background and grey image of the red ball in the second frame.
4.2 Finding Centre Point

A Hough transform was used to find the centre point of the circle detected in the source image. The Hough transform technique is commonly used for detecting circles, lines and other geometric shapes; it requires features to be specified in parametric form. The main advantage of using the Hough technique in this project is that it is relatively unaffected by image noise. Given the local measurement, the Hough technique is particularly useful for computing description of features. The idea behind the use of Hough technique for finding the centre point of a circle is that each input measurement (e.g. coordinates point “x, y”) indicates its contribution to a consistent solution, Fig6.2 displays the value for the centre point of the moving red ball in Fig6.3.
SECTION 5: KALMAN FILTER

5.1 Kalman Filter

The Kalman filter possesses many applications in technology. It is an essential part of both military technology and the development of space technology.

The main function of Kalman Filter is to utilize measurements recorded over time which contain random variations and inaccuracies to generate values that tend closer to the measurement’s true values and connected values that resulted from calculations. The Kalman filter calculate associated values by predicting a value to estimate the uncertainty of that predicted value, and compute a weighted average of the predicted value and measured value.

When one is dealing with a video source, as opposed to individual still images, we often have a particular object or objects that we would like to follow through the visual field. Tracking things that we have not yet identified is a related problem. Tracking unidentified objects is important when we wish to determine what is interesting based on its motion—or when an object’s motion is precisely what makes it interesting. Techniques for tracking unidentified objects, typically involve tracking visually significant key points (more soon on what constitutes “significance”), rather than extended objects.
5.1.1 Example Application

As an example application, consider the problem of determining the precise location of a moving car with both red tail lights on, the red tail lights tracked will provide an estimate of the position of the car within a few meters to the observer. A lot of vibrations and noise will be encountered at high frequency, though always remaining relatively close to the real position. Typically, dead reckoning will provide a smooth estimate of the car’s position but will drift over time as small errors accumulate. If the car should obey the laws of physics as expected, its position should be expected to change proportionally to its velocity.

In this example, The Kalman filter can be regarded as operating in two distinct phases namely (Predict and Update). The prediction phase sees the car’s position modified in accordance to the laws of motion plus any changes the acceleration and steering produce. Not only will a new estimate be calculated, but a new covariance will also be calculated.

In the update phase, a measurement of the car’s position is taken from the car’s red rear tail lights. Together with the measurement arrive some amount of uncertainty, and its covariance relative to that of the prediction from the previous phase will determine how much effect the new measurement will have on updated prediction. If the dead reckoning drift away from the real position, the red tail lights measurements would be expected to pull the position estimate back towards the real position but have no noticeable effect to the point of changing rapidly and noisy.
5.1.2 OpenCV and the Kalman Filter

OpenCV provides four functions that are directly related to Kalman filters:

cvCreateKalman
()

int nDynamParams,
int nMeasureParams,
int nControlParams

);

cvReleaseKalman(
CvKalman**kalman

);

The first of these generates and returns to us a pointer to a CvKalman data structure, and
the second deletes that structure.

typedef struct CvKalman {
  int MP;                       // measurement vector
dimensions
  int DP;                       // state vector dimensions
  int CP;                       // control vector
dimensions
  CvMat* state_pre;            // predicted state:
  // x_k = F x_k-1 + B u_k
  CvMat* state_post;           // corrected state:
  // x_k = x_k' + K_k (z_k' - H x_k')
}
CvMat* transition_matrix; // state transition matrix
    // F
CvMat* control_matrix; // control matrix
    // B
    // (not used if there is no control)
CvMat* measurement_matrix; // measurement matrix
    // H
CvMat* process_noise_cov; // process noise covariance
    // Q
CvMat* measurement_noise_cov; // measurement noise covariance
    // R
CvMat* error_cov_pre; // prior error covariance:
    // (P_k’=F P_k-1 Ft) + Q
CvMat* gain; // Kalman gain matrix:
    // K_k = P_k’ H^T (H P_k’ H^T + R)^-1
CvMat* error_cov_post; // posteriori error covariance
    // P_k = (I - K_k H) P_k’
CvMat* temp1; // temporary matrices
CvMat* temp2;
CvMat* temp3;
CvMat* temp4;
CvMat* temp5;

The next two functions implement the Kalman Filtering. Once the data is in the structure, one can compute the prediction for the next time step by calling cvKalmanPredict() and then integrate the new measurement by calling cvkalmanCorrect(). After running each of these routines, one can read the state of the system being tracked. The result of cvKalmanCorrect() is in state_post, and the result cvKalmanPredict() is in state_pre.
5.1.3 Corner Finding

There are numerous kinds of features that can be tracked. In the case of this project, it involved the tracking of one or more red balls over frames of video. If a red ball is highlighted over other surrounding colours and a point that is unique on the red ball is marked, it will be easy to find that point again in the next and subsequent frames. In this project, the red colour that was selected is unique and parametrizable, such that it is comparable to other points in a different image.
SECTION 6: PROJECT STRUCTURE & EVALUATION

6.1 Project Structure

This project was laid out as according to flow-chart (fig 6.0) below;

![Flow-chart showing the order in which project was carried out.](image)

*Fig 6.0 Flow-chart showing the order in which project was carried out.*
The Project Phases were carried out as follows (in order of flow-chart): sample code (step 0 – step5) in section5 describes each section in fig 6.0 above.

- Loading of video from file (Image Acquisition with the aid of OpenCV)
- Image Conversion (Hue Control and colour extraction).
- Thresholding for object detection.
- Object Co-ordinates.
- Kalman Filtering for Trajectory Estimation.
- Streamed Image Co-ordinates (Centre Point Association)
- Differentiating between different objects on the same Image.
6.2 Project Evaluation

This section describes the outcome of each section of the project.

6.2.1 General View

Taking on this project has increased my insight into functions of OpenCV with C-programming.

6.2.2 Progress Report (Refer to fig 6.0 showing project flow-chart)

✓ I successfully commissioned the OpenCV system to load frame of video into memory.

   (Completed).

The following code shows how an image is shown from a frame of video and displays it on the screen see Fig 4.0

```c
int main( int argc, char** argv )
{

   IplImage* img = cvLoadImage( argv[1] );
   // determines the file format to be loaded based on the file name

   cvNamedWindow(“C:/Users/julius/Desktop/FYP/redblue.bmp”);
   // opens a window on the screen that can contain and display an image

   cvShowImage( “redblue.bmp”, img );
   // show a named window that already exist
```
Conversions from RBG to HSV, RGB to HSV to RGB were completed.

- RGB $\rightarrow$ HSV
- RGB $\rightarrow$ HSV $\rightarrow$ RGB (Completed)

Conversions from RGB to HSV to Threshold to RGB and output to screen were also completed.

- RGB $\rightarrow$ HSV $\rightarrow$ Threshold $\rightarrow$ RGB $\rightarrow$ RGB (Completed)

Step 5 of the sample code in section 3 shows how the above can be achieved, Fig 4.1 and Fig 6.4 Shows extraction of coloured background and grey image of a Red ball.

- Centre point detection (Completed)
  - Fig 6.1 below shows the amount of circles detected in the frame and the centre point detection of the Red ball.

- Kalman filtering application (Partially completed – see section 6.1.3)
  - Fig 6.1 also shows the result of the predicted path of the moving object using Kalman tracking; Fig 6.4 shows the direction at which the ball should be moving.
Fig 6.2 Centre point detection & the use of the Kalman filter to track a Red ball in a frame of video.
Kalman Tracking: Predicting the path of the Red ball

Display with overlays (Completed)

The black circle drawn on the screen around the detected object in Fig 6.4 above is an overlay
7.1.3 Problems Encountered

Initially I was unable to commission OpenCV to read images. This problem resulted from my initial installation of OpenCV 2.0 instead of OpenCV 1.0 which contained all the sample codes and set-up for commissioning OpenCV to read images and video (see 4.1.1, section 4).

Path environment variable set up for OpenCV 1.0 and OpenCV 2.0 are similar (see section 4). However, OpenCV 2.0 was installed for the following reasons stated below

- OpenCV 2.0 has a better algorithm for circle detection works which works better than OpenCV 1.0.
- OpenCV 1.0 takes hue value to be 0-255 instead of 0-180 which works better for the red colour detection, table 1.0 shows HSV min-max ranges for GIMP and OpenCV. A practical example is shown in (fig 7.1 and fig 7.2)

Getting OpenCV 2.0 to work perfectly, OpenCV 1.0 was uninstalled completely.

C-make was also needed.

- C-make helped in compiling OpenCV from the source code.
- OpenCV 2.0 needs different files for different versions of studio.
- One will need to complete OpenCV 2.0 for visual studio 2008.

<table>
<thead>
<tr>
<th></th>
<th>GIMP</th>
<th>MAX</th>
<th>OPENCV</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>10&lt;H&lt;350</td>
<td>360</td>
<td>5&lt;H&lt;175</td>
<td>180</td>
</tr>
<tr>
<td>S</td>
<td>75</td>
<td>100</td>
<td>191</td>
<td>255</td>
</tr>
<tr>
<td>V</td>
<td>30</td>
<td>100</td>
<td>76</td>
<td>255</td>
</tr>
</tbody>
</table>

*Table 1.0 showing HSV min-max ranges for GIMP and OpenCV.*
Fig. 7.1 showing the problems encountered in Thresholding with OpenCV 1.0.
RESULTANT IMAGE FROM THRESHOLDING WITH OPENCV 2.0.

Fig: 7.2 showing resultant image after the Installation of OpenCV 2.0.
Another problem which ensued in the course of doing this project was in Kalman filtration. The example in OpenCV only has a 1-D example while a 2-D was needed for this project. Also the program crashed at the line

```
// Correct Kalman filter state
cvKalmanCorrect( kalman, z_k );
```

which is supposed to update the filter state with the values that was measured from the circle detection. However, the crashing problem was fixed. The problem was that if a circle wasn’t detected in one frame of the video it would still go ahead and do all that Kalman functions.

I just added "if (circles->total > 0) {" so that if there are no circles it won't go doing all the Kalman functions.

```
//loop through all the circles and draw a circle where circle is detected
//for( int i = 0; i < circles->total; i++ ){
   if (circles->total > 0) {
      int i = 0;
      float* p = (float*)cvGetSeqElem( circles, i );
```

It still didn’t predict properly, the predicted values were above zero but very small which was strange. Values were printed to the command line for debugging.

More functionality could be implemented in C-program in getting the Kalman filter to track the red balls to a reasonable accuracy before display with overlays.
8.1 Conclusion

As can be seen from the above report, the project was successful. There were problems with Thresholding, but this was eventually implemented successfully, however, the penultimate part of the project couldn’t be fully implemented because of certain hitches in Kalman filter. Since Kalman filter couldn’t be fully implemented to predict the path of the red ball correctly in a video.

A valuable lesson learned from this project is to not start with such massively high ambitions, as was done at the beginning of this project. More realistic goals should have been set earlier, and a simpler image of a red ball on a blue background like the image in fig. 6.1 rather than the more complex and more demanding image in fig. 7.1 This could have resulted in a realisation of the original specification ()

For example, attempting to start by implementing OpenCV2.0, this should have been preceded with a better version in OpenCV1.0. OpenCV 1.0 contained all the sample codes and set-up for commissioning OpenCV to read images and video. Also, a prototype path environment set up should have first been developed, before starting on C++ development.

Overall, many of the project goals were fulfilled, with the fundamentals of the specification realised.
BIBLIOGRAPHY

Web-sites:
http://www.nunosantos.net/?p=114

http://www.emgu.com/wiki/files/1.3.0.0/html/0ac8f298-48bc-3eee-88b7-d2deed2a285d.htm

http://en.wikipedia.org/wiki/Trichromacy


http://opencv.willowgarage.com/wiki/

http://opencv.jp/opencv-1.0.0_org/docs/ref/opencvref_highgui.htm

Image Processing by the human eye
http://research.opt.indiana.edu/Library/imageProcessing/imageProcessing.pdf

JPEG Compression (accessed September 2009)
http://images.google.ie/images

JPEG Compression the Human Eye (accessed January 2010)
http://www.pasadenaeye.com/faq/faq15/eyelabeled.jpg

The Function of Human Eye (how it works)
www.pasadenaeye.com/faq/faq15/faq15_text.html

Image Compression (accessed September 2009)
http://images.google.ie/images?um=1&hl=en&tbs=isch:1&q=red+ball&sa=N&start=756&ndsp=21

Image Thresholding (accessed September 2009)
http://en.wikipedia.org/wiki/Thresholding_(image_processing)

Trichromatic Colour Vision
http://en.wikipedia.org/wiki/Trichromacy

The view of Human eye - webvision.med.utah.edu/imageswv/pupil.jpeg

http://en.wikipedia.org/wiki/HSL_and_HSV

**Books**

**O’Reilly** *(Gary Bradski and Adrian Kaehler)* – learning OpenCV computer vision with the OpenCV vision library

**Project Spec** – flow chart
