|  |
| --- |
| IT Tralee |
| Implementation of Eye tracking to navigate a PC or Mobile Device |
| Supervisor – Rob Sheehy |

|  |
| --- |
| Gary O Brien  T00181299 |

# Abstract

Computer Vision is an area of Information technology that attempts to allow machines to view images and process information from these images. In modern society, the field of Computer Vision is ever growing with devices and application allowing for gesture control and different methods of control.

One such area is that of eye detection which can be used to develop heat maps of areas viewed on a screen or can be found to have uses in medical sciences.

This project is to develop an application to navigate an operating system such as windows or to navigate a mobile device using only a user’s eyes and gaze.

Table of Contents

[Abstract 1](#_Toc500881031)

[Chapter 1: Introduction 3](#_Toc500881032)

[Chapter 2: Computer vision 4](#_Toc500881033)

[2.1 Introduction 4](#_Toc500881034)

[2.2 Image Processing Operations 4](#_Toc500881035)

[2.2.1 Kernel Convolution 4](#_Toc500881036)

[2.2.2 Histograms 5](#_Toc500881037)

[2.2.3 Point Operations 6](#_Toc500881038)

[2.2.3.1 Linear Operator 6](#_Toc500881039)

[2.2.3.2 Clipping Process 6](#_Toc500881040)

[2.2.3.3 Sawtooth Operator 6](#_Toc500881041)

[2.3 Low Level Feature extraction 7](#_Toc500881042)

[2.3.1 First Order Edge Detection 7](#_Toc500881043)

[2.3.2 Prewitt Operator 7](#_Toc500881044)

[2.3.3 Sobel Operator 9](#_Toc500881045)

[2.3.4 Canny Edge Detection 10](#_Toc500881046)

[2.4 Motion Tracking 11](#_Toc500881047)

[2.4.1 Object of Interest 11](#_Toc500881048)

[2.4.2 Image Subtraction 11](#_Toc500881049)

[Chapter 3: Eye Detection 12](#_Toc500881050)

[Chapter 4: Methodology 13](#_Toc500881051)

[4.1 Research Undertaken 13](#_Toc500881052)

[4.2 Vision Document 14](#_Toc500881053)

[4.2.1 Project Scope 14](#_Toc500881054)

[4.2.2 System Architecture 14](#_Toc500881055)

[4.2.3 User Environment 14](#_Toc500881056)

[4.3 Prototype/Sprints 15](#_Toc500881057)

[4.3.1 Prototype 15](#_Toc500881058)

[4.3.2 Sprints 15](#_Toc500881059)

[4.4 Features 16](#_Toc500881060)

[References 18](#_Toc500881061)

# Chapter 1: Introduction

# Chapter 2: Computer vision

## 2.1 Introduction

Computer vision is the area of obtaining information from an image similar to the way a human brain processes visual stimulus. The idea of computer vision first arose between the 1960’s and 1970’s and when initially proposed was thought to be a relatively straightforward task to accomplish. It was perceived that, because vision is so fundamental to human life that it would translate to the areas of computer science with ease. However, this was not the case as computers can only understand numerical information. (Dawson-Howe, 2014)

## 2.2 Image Processing Operations

Image processing operations are a group of functions that can be performed on any image to either modify or obtain information about the image. These operations mostly work at the pixel level of an image by taking the number value of a pixel and performing arithmetic on it. In the case of image processing, the original image is modified to generate a new image.

### 2.2.1 Kernel Convolution

Kernel convolution or template convolution is a type of group image operation. Group image operations are image operations which are applied to more than one pixel at a time. A kernel when talking about image processing is a 3 x 3 matrix which is applied over an image. A pixel is selected from the image and this pixel constitutes the centre of the matrix. Every pixel surrounding this matrix is entered into the matrix grid. With linear convolution, assuming pixels of 8-bit colour depth, the values of the pixel are multiplied by its place in the grid. The value in the centre of the grid is then replaced in a new image with the sum of all the surrounding pixels. In average convolution, the process is similar except the value of the new in the centre is normalised by dividing the value by the number of grids in the matrix, essentially getting the average of all surrounding pixels. Kernel convolution makes up the basis for image manipulations such as blurring, embossing and many other common filter types seen in popular imaging software such as Photoshop. (Solomon & Breckon, 2010)

A specific type of template convolution, gaussian convolution, has practical uses in edge detection as it acts as a form of noise suppression. Gaussian filtering works mostly the same as previous filters but incorporates statistical methods. It differs from standard linear convolution by using the standard deviation as the method of controlling the smoothing as opposed absolute values in a matrix. An example of such a grid is shown below.

|  |  |  |
| --- | --- | --- |
| 8 | 12 | 8 |
| 12 | 20 | 12 |
| 8 | 12 | 8 |

The importance of this type of filtering with regards to edge detection lies in its ability to remove noise from an image but maintain the main edges that remain in the image, even if the sharpness of the edge has been reduced. (Marques, 2011)

Problems are encountered with kernel convolution in general when we deal with performing filtering on the edge of an image. There are three ways that this issue can be dealt with. We can perform all the necessary calculations as usual but leave the pixels inside the region of the image unchanged. This can result in unwanted image artefacts. We can perform the calculations for the matrix on the pixels that are only within the boundaries of the image and adjust the matrix to suit the pixel. Lastly, the pixels outside the boundary of the image may be mirrored with the values of the pixel inside the boundary and calculations can continue as normal. The latter two methods are the most common ways of dealing with the issue proposed. (Solomon & Breckon, 2010)

### 2.2.2 Histograms

Histograms can be used to map the levels of brightness of an image. In a standard image with an 8-bit colour depth, this gives us a range of colours from 0 to 255. This range correlates to the X axis of a histogram. The Y axis of an image processing histogram displays the number of pixels present containing the certain colour value. For instance, an image containing 500 pixels with an 8-bit colour representation of 204 would be mapped onto the histogram at the location (204, 500). Histograms may also be used to display brightness levels in an image. In this case the level of brightness is replaced by the colour value. This type of Histogram would mainly be used on a grayscale type image, where the only colour components are shades of grey, black and white. A histogram of this type to determine whether an image contains more dark or white colour. If the graph is skewed to the left we can determine that lighter pixels are abundant, whereas if it skewed to the right, darker pixels take up most of the image. (Nixon & S. Aguado, 2012)

### 2.2.3 Point Operations

#### 2.2.3.1 Linear Operator

Point Operators are a specific type of pixel mapping where every individual pixel value is changed based what the current value for the pixel is. In a linear point operator, multiplication/division functions are performed on an individual pixel and not a cluster of pixels. For instance, one way of increasing the brightness of any image is to multiply each pixel value by a scalar such as 2 to double the intensity of the pixel. This can be used to increase the contrast of a picture. (Marques, 2011)

#### 2.2.3.2 Clipping Process

If when applying a certain scalar to a pixel and the value exceeds the highest possible value that pixel can be (for instance 255 for 8-bit colour depth representing white), then the value for the pixel will be set to that maximum. This is known as a clipping process and is a core functionality of most point operations. (Marques, 2011)

#### 2.2.3.3 Sawtooth Operator

Another form of point operation is the sawtooth operator. The sawtooth operator works similarly to the above mentioned linear operator but instead uses a repeating form of the operator to emphasize contrast change within segments of an image and not the whole image. This has the effect of segregating parts of an image. (Nixon & S. Aguado, 2012)

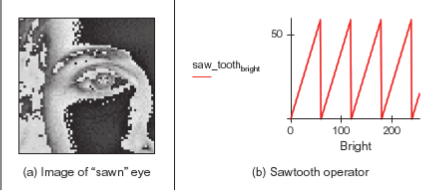


Image taken from (Nixon & S. Aguado, 2012)

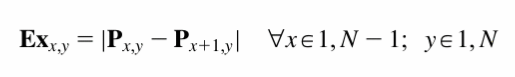
## 2.3 Low Level Feature extraction

### 2.3.1 First Order Edge Detection

Edge detection is used to determine where the boundary of an object lies. This method of feature extraction looks to draw straight lines at the edges of an object. This action occurs by looking at and measuring the contrast in an image. If there is a large difference in intensity between two parts of an image, we can classify this as an edge. In order to calculate whether a change is valid of being an edge, first order differentiation is used as this provides clearly the changes in intensity. Furthermore, we differentiate adjacent points on an image. Mark Nixon describes the difference between differentiating horizontal adjacent points and vertical adjacent points below.

“Differencing horizontally adjacent points will detect vertical changes in intensity and is often called a horizontal edge detector by virtue of its action. A horizontal operator will not show up horizontal changes in intensity since the difference is zero.” (Nixon & S. Aguado, 2012)

There are two formulas used to determine the horizontal and vertical edges. The first of these formulas is used to detect vertical edges Ex,



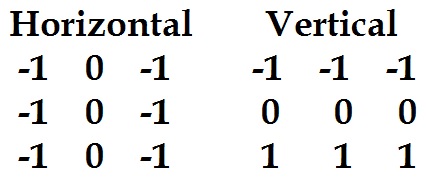
Below is the horizontal formula.



In both formulas, **P** represents the image the edge detection is being performed on. (Nixon & S. Aguado, 2012)

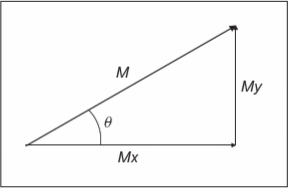
### 2.3.2 Prewitt Operator

The Prewitt operator is used to determine two types of edges. Vertical edges and horizontal edges. The key tool that the Prewitt operator uses is kernel convolution when determining whether something is a vertical or a horizontal edge. There are two different masks that are passed over every pixel on the image for the Prewitt Operator.

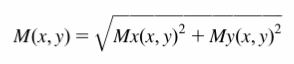


For instance, when the X filter(vertical) is passed over every pixel it will return a high value if there is a resemblance to a vertical edge present. This is the same for the Y filter(horizontal), the only difference being that this specific mask checks for horizontal edges. In the case of the Prewitt operator, it does not matter if the value received from the mask is a negative, it can be counted as if it is a positive number. (Marques, 2011)

Both of these masks generate different images based on the vertical and horizontal edges determined. The next step is to combine both of these images into one and generate the overall edges of the image. To do this the values generated for the X filter will be denoted as **Mx** and the Y filter will be denoted as **My**. This can be represented as a triangle where **M** is the magnitude of the edges.



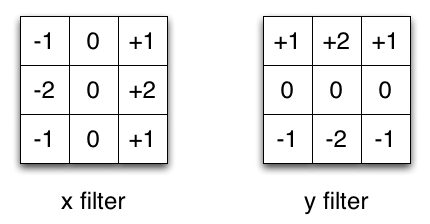
To generate the overall edges of an image these values will be passed into Pythagoras’ Theorem.



This calculates the overall gradient magnitude of the edges in the image. (ei, Dewei, Xiaoyu, Hui, & Jun, 2011)

### 2.3.3 Sobel Operator

The Sobel operator works almost identically to the way the Prewitt Operator works. To perform the Sobel operator an image must be converted to grayscale. This accentuates the difference in colour while using the X and Y masks to detect edges. As mentioned in a previous chapter, one of the main methods of reducing the noise in an image is to perform a Gaussian blur. This can be used in conjunction with the Sobel operator to accurately identify edges. Another key difference between the Sobel and the Prewitt operator is that the X and Y filters can be weighted differently. In Prewitt, the filter masks only use values of 0 and 1. Sobel allows for the value of 1 to change, as long as the weighting on both sides of the “0”’s are equal to zero.



All other processes involved in Sobel are the same as those of the Prewitt Edge detecting operator. (Yi, Xiaoyuan, Han, & Liming, 2017)

### 2.3.4 Canny Edge Detection

Canny Edge detection is more a multi stage algorithm as it is one individual process. The input for the Canny edge operator is the output of the Sobel edge operator. The main objective of Sobel is noise reduction, an issue that is encountered with many of the primary low-level edge detection techniques. With edge detectors such as Prewitt and Sobel, the edges detected can be thick and many pixels wide. The output of performing the Canny operator is that these edges are reduced to being only one pixel wide. This allows for further object detection to be performed more accurately in image post processing.

The steps of performing the Canny Edge operator are as follows:

* Apply a Gaussian filter to remove noise
* Perform Sobel or Prewitt to retrieve the gradient magnitude of the edges.
* Supress non-maximum pixels on the edges.

This step has the effect of making all the edges thin or just 1 pixel wide. This is performed by taking a pixel on an edge. If this pixels neighbours (being left and right if the edge is travelling upwards and up and down if it is a horizontal edge) are greater than the value of the current pixel then set this pixel to 0 (being black). This is the process for eliminating wide edges to provide a more accurate representation of where edges reside.

* To provide an even more accurate determination of the edges on an image, a threshold of what an edge is must be applied.

This consists of two threshold limits. An upper and a lower limit. If a pixel is above the high limit it is considered a strong edge. If it is between the lower and upper limit it is considered a weak edge and edges that fall under the lower limit are not counted as edges.

* Final step is to perform hysteresis on the image. This has the effect of keeping weak edges only if they are attacked to a strong edge.

Below is an example of an image before and after the Canny operator has been applied. (Parker, 2010)



## 2.4 Motion Tracking

### 2.4.1 Object of Interest

When talking about motion tracking, defining the Object of Interest (OOI) is the first step. The OOI is an area which is moving relative to a still background. A key factor in determining OOI’s is velocity and acceleration. If a cluster of pixels moves from one side of the viewport to the opposite side, then it would not be beneficial to consider this an OOI. A maximum velocity must be set alongside having an increase in acceleration being constant. Frame rate must be taken into consideration when determining the maximum velocity for an OOI. To avoid tracking every moving pixel in a video, OOI’s are only identified if they contain a minimum number of pixels. There are many issues affecting this form of motion tracking. Sudden or incremental changes of lighting can interrupt the tracking process such as the sun becoming blocked by clouds in the sky. Shadows mimic the movement and shape of objects which can confuse the system into representing these as OOI’s. Weather conditions scramble the image of an area causing indistinguishable noise over a recording. (Dawson-Howe, 2014)

### 2.4.2 Image Subtraction

# Chapter 3: Eye Detection Tools

This chapter will look at some of the tools that currently exist which relate to the field of computer vision.

## 3.1 OpenCV

OpenCV is a Computer Vision library for Java, C++ and Python. For this project, the java driver will be used. This is because later in the project the application will be ported to an Android device using Android studio. OpenCV provides most to all of the tools necessary for this project including image manipulation and feature extraction through the use of Haar-Cascades.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

# Chapter 4: Methodology

## 4.1 Research Undertaken

The research conducted for this project was conducted under the impression that it will assist with the overall production of the application during the development phase. This segment details the key findings in this research report and the findings during the production of the prototype.

The main research chapter of this report outlines the basics of what computer vision is. As the purpose of this project is to recognise and locate the eyes, this chapter details some basic image processing techniques. While implementing the prototype it was found that the eye detection on raw images was not accurate. Basic image processing is used widely in the areas of computer vision to increase feature extraction precision such as using the Canny edge detector process. The next section will outline some of the software used in the prototype.

## 4.2 Research Question

## 4.3 Project Proposal

The purpose of this project is to develop an application that allows users to navigate an OS by means of controlling a mouse on a computer screen or tracking the user gaze on a mobile device. The user’s eyes will be isolated and tracked using a webcam or front facing camera. The image will be processed using the Java implementation of OpenCV. An algorithm will determine where on the screen the user is looking.

* OpenCV provides a built in eye detection algorithm through the use of Haar Cascades.
* Blob detection can be used on the isolated eye region to determine where the pupil is.
* The centroid of the pupil blob can be determined and used as the primary centre of focus for the eye.

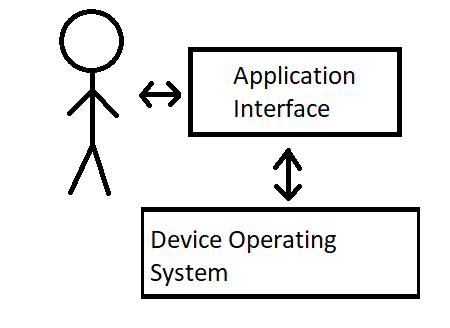
For this algorithm to work, a calibration system must determine the range of movement the user has by getting them to look at the centre of the screen, then all corners of the screen. Another algorithm will move the mouse to the location of the user’s gaze. This will be implemented using Javas Robot class.

* The ratio of the pupil to the bounding box of the eye will be determined and this will be translated to the screen of the user’s device.
* The robot class’s mouse movement method will run in a loop, moving the mouse to the location of the user’s gaze.

Simple gesture control such as a wink will emulate a mouse click. This will implemented using OpenCV.

This application will be ported to mobile devices also, taking advantage of the front facing camera. Android Studio will be used for the port.

### 4.3.2 Proposed System Architecture



**Figure 1 Overview of System Architecture**

### 4.3.3 User Environment

The user will access the application locally from there device. In the case of a computer this will be a program the user will have to start up. With a phone or mobile device the user will start an app. This application will only run on windows computers and mobile devices that support the android OS. Lighting will have to be of a sufficient level to recognise the eyes.

## 4.4 Features

To display the features and their priority in this project, the Moscow method will be used. This is a management tool used to break up a project into the features that it Must have, Should have, Could have and Won’t have.

|  |  |
| --- | --- |
| Must Have | 1 |
| Should Have | 2 |
| Could Have | 3 |
| Won’t Have | 4 |

|  |  |  |
| --- | --- | --- |
| **Feature ID** | **Feature Detail** | **MoSCoW Prioritisation** |
| 01 | Camera live feed capture using a java driver | 1 |
| 02 | Eye detection and isolation from face | 1 |
| 03 | High frame rate video capture | 2 |
| 04 | Pupil isolation from detected eyes | 1 |
| 05 | Gaze heatmaps for video input | 3 |
| 06 | Manual recalibration functionality | 2 |
| 07 | Android mobile device compatibility | 2 |
| 08 | iOS mobile device compatibility | 3 |
| 09 | User defined settings such as sensitivity | 3 |

# Chapter 5: Implementation

## 5.1 Prototype

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Install OpenCV and complete online tutorials to become comfortable with the fundamentals | Complete |
| 2 | Create an application that captures a live feed from the webcam in OpenCV | Complete |
| 3 | Use OpenCV’s feature recognition to identify facial features | Complete |
| 4 | Identify rough circles where the pupil might reside using OpenCV | Complete |
| 5 | Create an application in FFmpeg to test if captured framerate is higher than that of OpenCV | Complete |
| 6 | Merge the functionality of FFmpeg and OpenCV to capture footage from webcam at a higher frame rate and process it using OpenCV | In Progress |

## 5.2 Sprints

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint Number** | **Start Date** | **End Date** | **Tasks** |
| 1 | 24th November, 2017 | 8th December, 2017 | OpenCV Research, Basic feature recognition, Project prototype |
| 2 |  |  | Develop application to allow for highest frame rate capture |
| 3 |  |  | Accurately isolate the eyes and pupils from the rest of the face |
| 4 |  |  | Write logic to move a mouse pointer using eyes |
| 5 |  |  | Port the logic used on the PC version of the application to an Android device |

## 5.3 Sprint Plan

### 5.3.1 Sprint 1

**Start Date:** 24th November, 2017

**End Date:** 8th December, 2017

|  |  |  |
| --- | --- | --- |
| **Task No.** | **End Date** | **Status** |
| 1 | 8th December, 2017 | OpenCV Research, Basic feature recognition, Project prototype |

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

**FFmpeg** It was found during the development of the prototype for this project that OpenCV captures live video from the webcam at a low framerate (5-7 frames per second). It was found that FFmpeg can be used in conjunction with Java to capture the live feed at a much higher and consistent speed (30 fps). This then allows for OpenCV to perform functions to the live video at the higher capture rate adding to overall application performance and smoothness.

# References

Bunke, H. (2014). *Progress in Computer Vision and Image Analysis.* World Scientific Publishing Co Pte Ltd.

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV.* John Wiley & Sons, Incorporated.

ei, Y., Dewei, Z., Xiaoyu, W., Hui, L., & Jun, Z. (2011). An improved Prewitt algorithm for edge detection based on noised image. *Image and Signal Processing (CISP), 2011 4th International Congress on.* Shanghai: IEEE.

Marques, O. (2011). *Practical Image and Video Processing Using MATLAB.* Wiley.

Nixon, M., & S. Aguado, A. (2012). *Feature Extraction and Image Processing for Computer Vision.* Elsevier Science.

Parker, J. R. (2010). *Algorithms for Image Processing and Computer Vision.* John Wiley & Sons, Incorporated.

Solomon, C., & Breckon, T. (2010). *Fundamentals of Digital Image Processing : A Practical Approach with Examples in Matlab.* Wiley.

Yi, Z., Xiaoyuan, H., Han, Z., & Liming, Z. (2017). Edge detection algorithm of image fusion based on improved Sobel operator. *Information Technology and Mechatronics Engineering Conference (ITOEC), 2017 IEEE 3rd.* Chongqing: IEEE.