DRIVER ATTENTION AND BEHAVIOUR MONITORING WITH THE MICROSOFT KINECT SENSOR

DISSERTATION

Cleshain Solomon-41278844

DRIVER ATTENTION AND BEHAVIOUR MONITORING WITH THE MICROSOFT KINECT SENSOR

by

Cleshain Theodore Solomon

submitted in accordance with the requirements for
the degree of

MAGISTER TECHNOLOGIAE

In the subject

ELECTRICAL ENGINEERING

University of South Africa

Supervisor: Professor Z. Wang

November 2015

DECLARATION

I declare that *Driver attention and behaviour monitoring with the Microsoft Kinect sensor* is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

05.11.2015

SIGNATURE
(Mr CT Solomon)

DATE

ACKNOWLEGEMENTS

I would like to thank my wife Denae Solomon for her support and patience and I am equally grateful to my supervisor Professor Wang for his input and advice throughout this study. In addition I would like to acknowledge the support from the Further Education Program at Volkswagen Group South Africa.

ABSTRACT

Modern vehicles are designed to protect occupants in the event of a crash with some vehicles better at this than others. However, passenger protection during an accident has shown to be not enough in many high impact crashes. Statistics have shown that the human error is the number one contributor to road accidents. This research study explores how driver error can be reduced through technology which observes driver behaviour and reacts when certain unwanted patterns in behaviour have been detected. Finally a system that detects driver fatigue and driver distraction has been developed using non-invasive machine vision concepts to monitor observable driver behaviour.

Key terms:

Kinect; Driver Fatigue; Driver Behaviour; Driver Distraction; Behaviour Detection; Driver Attention; Behaviour Monitoring; Feature Extraction; Face Detection

LIST OF CONTENTS **PAGE**

LIST OF FIGURES **PAGE**

LIST OF TABLES **PAGE**

LIST OF ABBREVIATIONS

ANPR – Automatic Number Plate Recognition
API – Application Program Interface
CCD – Charge-coupled Device
CMOS – Complementary Metal-oxide Semiconductor
IR – Infrared Receiver
LoG – Laplacian of Gaussian
OCR – Optical Character Recognition
MSER – Maximally Stable Extremal Regions
RANSAC – Random Sample Consensus
RGB – Red, Blue and Green colour model
ROSPA – Royal Society for the Prevention of Accidents
SDK – Software Development Kit
SIFT – Scale-invariant Feature Transform
S/N – Signal to Noise Ratio
SURF – Speeded Up Robust Features
WPF – Windows Presentation Foundation
YUV – a colour space used as part of a colour image pipeline

CHAPTER 1
INTRODUCTION

1.1 Problem statement

Various independent studies have suggested that around 25% of all road accidents are related to driver error caused by fatigue or distraction. The issue with weariness is that it influences mental readiness, diminishing a driver's capacity to operate a vehicle securely and expanding the danger of human mistake that could prompt mishaps (Kaur & Singh, 2014). Besides, it has been demonstrated that driver fatigue moderates response time, diminishes mindfulness, and disables judgment. (Barr L., 2011). A review found that around 30% of rural crashes in Western Australia could be attributed to driver fatigue. Driver fatigue in Western Australia accounts for more than 70 deaths and 500 serious injuries each year. In the UK, driver Fatigue accounts for more than 20% of traffic accidents each year (Care Drive, 2013).

Driver weariness is a noteworthy contributing variable to traffic accidents in South Africa. Goodyear's yearly 2013 street safety overview proposes that young drivers in South Africa were among the most forceful drivers on the planet. Reviews additionally uncovered that young South African drivers are significantly more inclined to be occupied by telephone calls and web use while in the driver's seat in contrasted to European youth (Goodyear , 2013). Driver error due to distraction and fatigue accounts for many road accidents in South Africa. The Royal Society for the Prevention of Accidents (ROSPA, 2001) suggests that sleep loss and sleep disruption is caused by many factors which include amongst others long working hours, family responsibilities, medication and stress.

Another independent study has additionally demonstrated that somebody who utilizes a phone while driving is four times more inclined to be in a fender bender and twice as likely if messaging while driving. Texting or e-mailing while driving is more dangerous than driving under the influence of alcohol. A late study directed in the U.S.A found that 69% of U.S. drivers between the ages 18-64 years of age reported that they had chatted on their mobile phone while driving inside of the 30 days before they were surveyed (Centers for Desease Control and Prevention, 2013).

Statistics have shown that in many cases driver error has led to fatal accidents. In 2010 there were more than one billion motor vehicles on the road (Sousanis, 2011). Fatigue has a measurable effect on driver performance. Fatigued drivers exhibit observable behaviours which include rapid eyelid blinking, sideways head movement, and facial expressions. With more vehicles on the road than ever before and with increased pressure on individuals, the demand for safer, intelligent vehicles has grown as well. The purpose of this study is to identify the possibilities of using computer vision techniques in detecting driver fatigue with face and eye gaze estimation.

1.2 Research Goals

The objective is to research the possibilities and applications of an intelligent system that monitors driver behaviour and attention in real-time with a real-time response by using computer vision techniques in detecting driver fatigue with face pose estimation. The system will continuously monitor if a driver is focussed on the road ahead. The solution is a system based on driver behavioural measurement and face pose tracking. There are numerous types of driver fatigue detection systems available on the market today. Of all the driver fatigue detection systems, none offer fatigue detection combined with detecting driver cellular phone usage even though statistics have shown that the use of a cellular phone while driving can be as fatal as driving under the influence of alcohol. A cellular phone will be detected via object recognition

1.3 Assumptions

The following assumptions will be made:

- The camera will be stationary with direct line of sight and immune to vibration.
- The camera’s line of sight is not obscured.
- There are no occlusions.
- The subject is always visible to the camera.
- The unit is maintenance free.
- User calibration is not required.
- During operation, the subject remains within the boundaries of the camera’s operating angle and workspace.

- The system in its design must be non-intrusive.
- Only one person will be detected at a time.
- The light conditions are always sufficient for tracking.
- The background is static.

1.4 Delimitations

- The computer vision programming will be based on the Microsoft Kinect sensor, the Microsoft Kinect SDK and Matlab Computer Vision Toolbox.
- The primary research is fatigue detection by means of behavioural observation.
- The secondary research is cellular telephone detection by means of computer vision object recognition.
- In-vehicle testing will be simulated in laboratory conditions and not on the road.

1.5 Limitations

- The camera could be sensitive to motion blurs, reflection and occlusion.
- Lighting conditions will not always be suitable for tracking.
- Tracking quality and accuracy is best at the optimal distance of 1.5 meters but is reduced at further distances.
- Face tracking is sensitive to faces with thick beards and eye/sunglasses. According to Microsoft, this area is open for improvements (Microsoft, 2013).
- On the road testing and in-vehicle testing will not be possible, but will be simulated.

1.6 Research methodology

As indicated by The American Heritage Dictionary of the English Language, an examination is a test under controlled conditions. It's made to show the truth and examine the validity of a hypothesis (Pickett, 2000). The research approach consists of both practical experiments and theoretical research by means of a literature review and experimental evaluation but the first and most important step was to identify and define the problem. Once the problem was defined, a hypothesis was formulated with its possible outcomes estimated. Existing literature and technologies was reviewed together with the possible effects of fatigue on human

interaction and behaviour. Sample code and new code was evaluated and the outcome manipulated and tested to get the best possible results.

1.7 Existing Fatigue detection technologies in vehicles

Many academics (Barr L., 2011) argue that an on-board computer control unit that screens the driver state continuously and in real time may have genuine worth as a safety net. Many vehicles today offer some form of fatigue detection systems. There are various types of methods and systems in use in modern vehicles for detecting driver fatigue. These different types of methods include Steering Pattern Monitoring, Vehicle Position in Lane Monitoring, Driver Eye/Face Monitoring and Physiological Measurement. Volvo's Driver Alert Control system is a mix of a Steering Pattern Monitoring system and a Vehicle Position in Lane Monitoring System that constantly tracks the vehicle's developments while on the move and identifies whether the vehicle is directed or driven in a controlled way or uncontrolled way (Volvo Cars, 2007). This system monitors the position of the vehicle on the road by looking at the lane markings. The camera is located on the windscreen and is pointed into a forward direction. The fatigue detection system combines the lane change camera with the on-board vehicle sensors to detect erratic driver behaviour. With the sensory inputs the control system can activate an audible warning and a visual warning and advises the driver to take a break. Volvo Cars also filed a patent in 2003 for a method to detect and track a driver’s head and eyes in relation to the vehicle internal space. The method quantifies the position of the driver’s head against a reference point and tracks the orientation of the driver’s head and eyes (Larsson & Trent, 2008).

The Attention Assist System developed by Mercedes-Benz works on a similar principle, but includes the monitoring and use of the turn indicators and certain control inputs. The on-board computer generates a driver profile and then evaluates whether the driver is behaving erratically (Daimler, 2008). Ford's Driver Alert is an illustration of a Vehicle Position in Lane Monitoring System and uses a forward-looking camera to screen the vehicle position in the lane and calculate a watchfulness level for the driver. On the off chance that this watchfulness level drops beneath a specific threshold, the vehicle will issue a visual and acoustic warning signal. The system issues two warning signals based on the calculated vigilance of the driver. The first warning is a soft warning with a message on the instrument panel and the second

warning is a hard-warning which is more audible and intrusive then the first warning urging the driver to take a break (Euroncap, 2013). Bosch's Driver Drowsiness Detection system is offered as a standard feature on Volkswagen's Passat Alltrack (Volkswagen AG, 2013). Fatigue is detected based on sensory input from the steering- angle sensor located in the electro-mechanical power steering system. Driver Drowsiness Detection monitors the steering behaviour of the driver. If the driver does not make steering corrections over a period of time but suddenly makes an abrupt steering manoeuvre, the control system, through its software algorithms identifies this steering input and associates this behaviour with a normally fatigued driver. The system combines the steering input with many variables such as vehicle speed, time of day, use of the turn signals and length of the trip, etc. to calculate the tiredness index of the driver. If the calculated tiredness index exceeds a specified limit, the driver is confronted with a flashing icon on the instrument panel and an audible warning buzzer. Combined with the navigational unit, the control system can also advise the driver about the next available location to stop and take a break. The driver cannot deactivate the driver fatigue detection system. Volkswagen also studied a system called Fatigue Detection System which monitors eye blinking and estimates the probability of micro-sleep-episodes (Jan, et al., 2005).

A Driver Face Monitoring System developed by Denso, an automotive component manufacturer and supplier, monitors driver fatigue by tracking the driver's facial movements over the entire facial area. The digital camera system is installed in the steering column and directed at the drivers face. Various points on the drivers face are monitored. The facial expression is monitored at 68 points on the face and a software algorithm calculates the drowsiness index. If the index is above specified limits, the system categorises driver fatigue into five levels: 1) not sleepy, 2) slightly fatigue, 3) fatigued, 4) rather fatigued, 5) very fatigued and 6) sleeping (Denso Corporation, 2012).

Care-Drive, a company that specialises in vehicle safety products has developed a Driver Fatigue Monitor called the MR688 (Care Drive, 2013). According to Care-Drive the MR688 is the only digital camera based system in world that monitors driver fatigue day and night. This system is based on a patented pupil detection technology. The main characteristics of this device are that it monitors driver fatigue all day round irrespective of the light intensity or whether the driver wears glasses or not. It is claimed by Care-Drive that the system provides

an audible warning if the driver does not focus on the road or if the driver talks to passengers, looks in the rear view mirror for an unreasonable amount of time, if the driver lowers his head and if the driver pays to much attention on navigational equipment or entertainment systems. The MR668 is small and out of sight. The system is non-invasive and is said to have a reasonable amount of accuracy (Care Drive, 2013).

An automotive supplier called Leisure Auto Security Equipment is a company that specialises in vehicle safety products. They have a product that combines a driver fatigue alarm and an alcohol tester. The system is a Physiological Measurement system and its sensors are embedded into a leather steering wheel cover which evaluates human skin biology electric signals and driver behaviour. It uses this data to calculate driver fatigue (Leisure Auto Security Equipment , 2009).

Continental Automotive, an automotive component supplier has developed a driver fatigue detection system called the Continental Driver Focus System (Continental Corporation, 2013). The system utilizes an infrared camera and it is mated with existing driver safety frameworks, for example, lane-departure cautioning and cruise control. The system scans the driver's face to detect whether the driver is paying attention. The system looks at the shape of the eyes, the position of the nose and orientation of the head and uses these points to detect whether the driver is focussed on the road (Continental Corporation, 2013).

In 2006 Toyota introduced a system called the Driver Attention Monitor in the Lexus GS. The system uses infrared sensors to monitor driver attentiveness and includes a CCD camera which is placed on the steering column. The camera is capable of tracking the eyes via six infrared LED detectors. The driver is warned with flashing lights and an audible warning tone from the instrument panel if fatigue is detected

Mazda MC developed a Vehicle Position in Lane Monitoring system called Lane Departure Warning System or LDWS. The system detects line markings and warns the driver of unintentional lane departures. Intentional lane departures are normally combined with detecting the usage of turn signals while changing lanes, but if not; the algorithm could estimate an unintentional change which triggers a sound alarm (Mazda Motor Corporation, 2015). FaceLAB 5 is an eye tracking system, designed and developed by a company called

Seeing Machines. FaceLAB 5 is said to be accurate and is capable of tracking eye movement in many different operating conditions. Their system generates data rapidly and provides rapid feedback to the user on eye movement, head position, eyelid aperture and pupil size (Seeing Machines, 2015). The University of Central Florida has developed a monitoring system, which they filed a patent for in 2002, based on digital cameras that monitor head motion and eye motion with computer vision algorithms for monitoring driver alertness and vigilance for drivers of vehicles and trucks (Smith, et al., 2000).

The systems discussed are examples of vehicle-based fatigue detection systems. The vehicle-based measurement system uses vehicle sensory technologies which include steering angle, accelerator pedal and other in-vehicle sensory input to analyse the vehicle's behaviour on the road. Fatigue can be measured, but if combined with driver behavioural measurement, the accuracy can be improved upon. Normally drowsy drivers do not fall asleep instantaneously. There is often a measurable performance drop linked with psychological signs. This research evaluates the possibilities of a driver behavioural measurement system for detecting driver fatigue and includes cellular telephone usage detection while driving.

1.8 Organisation of Dissertation

Chapter 1 - Introduction:
In Chapter 1 the problem of driver fatigue is discussed with goals and motivation for this study. This chapter provides a background as introduction and discusses existing fatigue detection technologies.

Chapter 2 - Literature review:
Chapter 2 provides a brief discussion about human vision, the history of computer vision and key concepts of computer vision systems and techniques. Image acquisition is discussed, including different types of images, photo sensors and the effects of noise in an image with noise filtering, edge detection and feature detection/extraction.

Chapter 3 - MS Kinect sensor:

In Chapter 3 the Microsoft Kinect sensor and MS Kinect SDK is introduced. The chapter describes how to open a new project within the software development kit, the colour stream, depth stream, skeleton tracking and face tracking including the benefits of the Kinect Fatigue Detection system.

Chapter 4 – Experimental design:

In Chapter 4 the experimental design is discussed. Here it is shown how the driver will be tracked and the expected performance is discussed.

Chapter 5 – Experimental evaluation:

This chapter shows the vector mathematics for calculating image geometry. The chapter shows how driver fatigue and driver distraction is monitored with Kinect and demonstrates the efficiency of this system.

Chapter 6 – Conclusion and future work:

In Chapter 6 the benefits of the Kinect Fatigue Detection system is discussed as well as future ideas and possibilities.

CHAPTER 2
LITERATURE REVIEW

2.1 Physiology of the eye

The human visual system is a sophisticated system that has evolved over millions of years and is so complex that we do not yet fully understand how it works and it this is why we cannot yet fully duplicate its function. The eye is shaped as a slightly irregular hollow sphere and is responsible for pre-processing and image formation. The cornea is a protective layer around the ocular ball which accommodates to keep the shape of the ocular perceiver. The cornea keeps dirt and irritants out and forfends the ocular perceiver from damage. Light is refracted by the cornea, passes through a gap created by the iris called the pupil size, then through the lens and is projected onto the retina. The iris gives the eye its colour and controls the amount of light entering the eye by contracting or releasing the ciliary muscle, similar to the aperture setting in a camera (Nixon & Aguado, 2012). An overview of the human eye is shown in Figure 2.1.

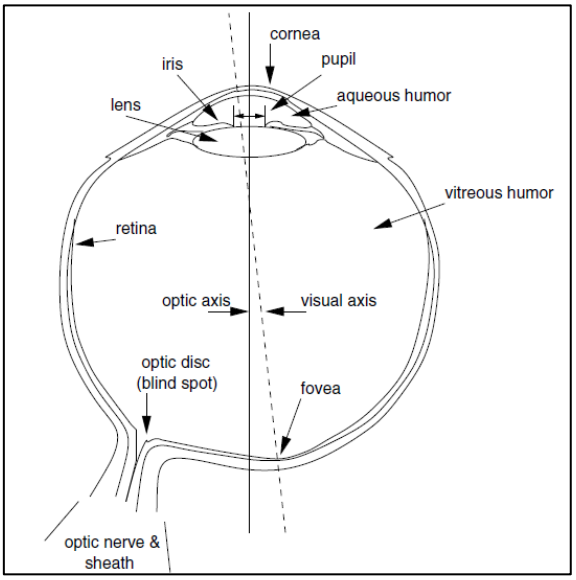


Figure 2.1: Human Eye (Duchowski, 2007)

The refraction of the light through the pupil and lens causes the image to be reversed and appears upside down on the retina. Photo receptors are located on the retina with the fovea having the largest density of photo receptors called rods and cones. Approximately 120 million rods and 7 million cones are located in the retina (Duchowski, 2007). The region on the retina that does not have any photo receptors is called the blind spot. The blind spot is

where the optical nerve is located. The optical nerve carries impulses or signals to the brain where interpretation and visual perception occurs with the correct perspective of the environment. Normally light is accurately focussed on the focal point of the eye, but this is not always the case because of imperfections with the shape of the eye or in the lens (Nixon & Aguado, 2012).

2.2 Computer Vision History and Background

Research in computer vision started in the 1960s when the first PhD thesis in this topic was published but it was only in the 2000s that significant progress was made (Aloimonos, 1990). Computer vision is in essence the science and engineering discipline concerned with making inferences about the external world from digital imagery. In simpler terms it is the ability of computer systems to interpret, process, analyse and understand the real world i.e. it's the ability of a computer system to see its environment. For instance when the human eye examines a colour image, it sees all the different colours, shapes and textures. The brain is trained to give meaning and structure to the image. To a computer this is just raw data from a photo sensitive sensor. The concept of computer vision is to give the computer system the capability to interpret the image and to make sense of it through computation. According to Vernon (Vernon, 1991) computer vision is the automatic analysis of images produced from a three-dimensional world where the image itself is two-dimensional. The third dimension is lost in the process of transforming the three-dimensional world into a two-dimensional image.

Various forms of computer vision systems exist today with a variety of applications. According to Szeliski, computer vision is used in Optical character recognition (OCR): for automatic number plate recognition (ANPR); Machine inspection: for parts analysis for quality assurance; Retail: used in automated checkout lanes; 3D model building (photogrammetry): used in 3D models from aerial photography; Medical imaging: in intra-operative imagery; Automotive safety: in obstacle detection and pedestrian safety; Surveillance: for intrusion monitoring; Fingerprint recognition and biometrics: for access authentication (Szeliski, 2010). Other applications are in biometrics (e.g. face recognition), human to computer interaction (e.g. Microsoft Kinect), robot vision systems (e.g. pick and place systems in manufacturing), image analysis (e.g. image reconstruction), vehicle safety

(e.g. front and rear view cameras), security systems (e.g. airport security), and computer graphics.

2.3 Image Acquisition and Representation

Image formation depends on three things; a light source, internal/external camera parameters and a scene with surface reflection and texture and is not immune to noise because light variations, camera electronics, surface reflection and lens quality contribute to noise in an image (Szeliski, 2010). According to Jähne nothing can be observed or processed without a radiation source that illuminates the scene or object of interest (Jähne & Haussecker, 2000). Figure 2.2 demonstrates perspective projection by means of the pinhole camera model. The coordinate system on the image plane is represented by *x* and *y* and the coordinate system of the world are represented by *X*, *Y* and *Z* where *Z* is distance between the lens and the object or depth and *f* is the focal length of the camera and is expressed by equations (2.1), (2.2) and (2.3).

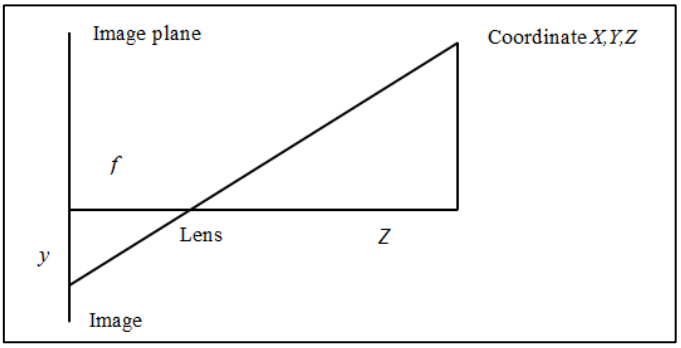


Figure 2.2: Perspective projection

$$-\mathrm{y} \div \mathrm{Y} = \mathrm{f}/\mathrm{Z} \tag{2.1}$$

Therefore:

$$\mathrm{x} = -(\mathrm{f} \times \mathrm{X})/\mathrm{Z} \tag{2.2}$$

$$\mathrm{y} = -(\mathrm{f} \times \mathrm{Y})/\mathrm{Z} \tag{2.3}$$

2.4 Images Types

There are three types of images; grey scale Figure 2.3, binary images Figure 2.4 and colour images Figure 2.5. To a computer an image, for example a greyscale image, is essentially made of 2D arrays of numbers with each element in the array representing an intensity level or grey level with the lowest value 0 representing black and the highest value of 255 representing white (Demaagd, et al., 2012). A binary image is an image that can have only two possible values for each pixel, being either black or white (0 or 1 stored as a single bit value). A colour image is made up of three 2D arrays consisting of red, green and blue (RGB) and the number of rows and columns of elements or pixels define the image's resolution. Histograms show the image intensity levels by plotting the number of pixels with a certain brightness level against another set of pixels with its brightness level. For 8 bit pixels the intensity level ranges between 0 for black to 255 for white pixels (Nixon & Aguado, 2012). Figure 2.6 shows the histogram for the image in Figure 2.5. The pixels around the black hair show low intensity levels while the pixels of the skin shows higher intensity levels.

Figure 2.3: Greyscale image

Figure 2.4: Binary image in Matlab

Figure 2.5: Colour image

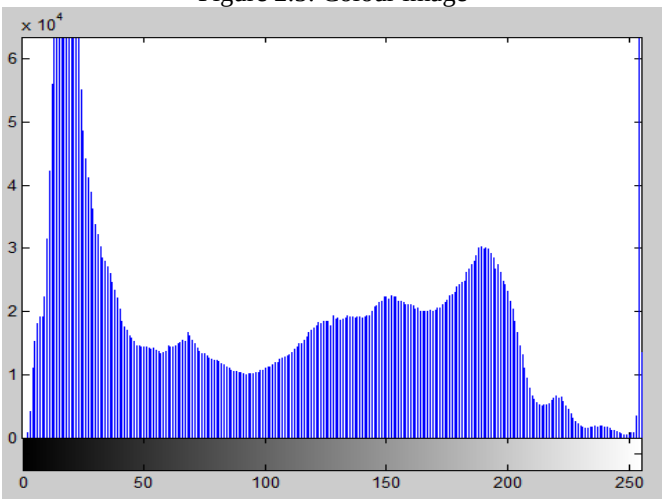


Figure 2.6: Histogram of grayscale image

When a computer examines a colour image it sees the pixels with each pixel having its own RGB value. The image at coordinate *x, y* measures how much brightness is captured at pixel coordinate *x, y* (Nixon & Aguado, 2012). The camera senses the brightness or reflectance

which is the light reflected at each visible point in the scene, than fed into an analogue to digital converter (Vernon, 1991). The data is stored as a value which is referenced to the *x, y* coordinate of the image and accordingly, the image is a matrix of pixels (Nixon & Aguado, 2012). The value of each pixel is proportional to its brightness.

Figure 2.7 represents a matrix of an image and Figure 2.8 represents its image pixel value equivalent. The image is brightest in the centre where the eye is located with higher pixel values and darkest around the eye lid and the pupil region with lower pixel values.

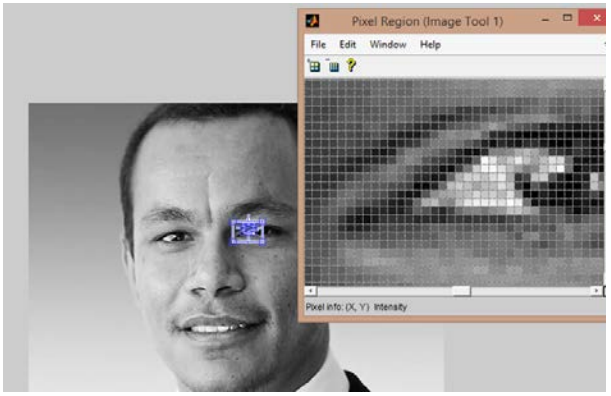


Figure 2.7: Matrix of Image in Matlab

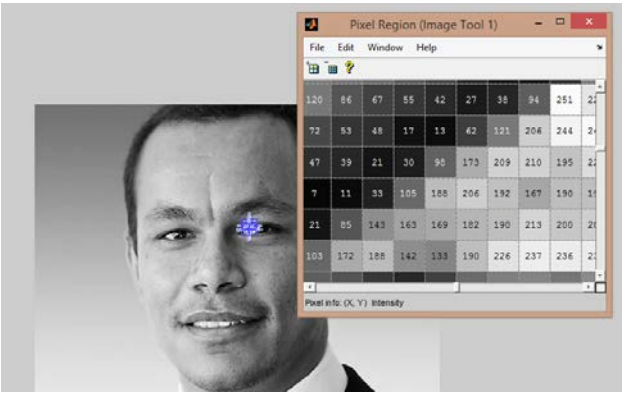


Figure 2.8: Matrix of pixel values in Matlab

2.5 Scene Illumination

In machine vision the purposes of illuminating the scene in an industrial application is to visually enhance the parts to be imaged in order to show their defects, flaws and to highlight their features (Vernon, 1991). Light is a type of electromagnetic radiation and is part of the electromagnetic spectrum along with long-waves, radio waves (commonly used in television and radio transmission), microwaves and moving into light wavelengths which include infrared, visible light, ultraviolet light and x-rays. Light is presented as both an electromagnetic wave and a particle called a photon. The energy carried by one photon traveling at the speed of light *c* is given by equation (2.4) where λ is the wavelength, *v* is the frequency and $h = 6.626 \times 10^{-34}$ is Planck's constant (Jähne & Haussecker, 2000).

$$e_p = hv = {}^{hc}/_{\lambda} \tag{2.4}$$

The amount of energy of the photon determines its wavelength and the wavelength determines colour. The number of photons corresponds to the light intensity. When photons hit a Silicon surface some electrons will get dislodged. The number of electrons depends on the light intensity and the wavelength. This is called the photoelectric effect. When an image is being captured, light passes through the lens and onto an image sensor called a pixel. A typical high end camera can have as many as 20 million pixels per image sensing device arranged in an array *n x m*. This rectangular grid is the camera's resolution. Each pixel in the rectangular grid is like a cup that captures light and converts the photons into either current or voltage depending on sensor type.

2.6 Image Sensors

The three main types of image sensing devices are charge coupled devices (CCD) plus complementary metal oxide silicon (CMOS) based on semi-conductor technology and vidicons which is based on vacuum-tube technology. The main difference between CCD and CMOS devices are in the way they transfer the charge out of the pixel and into the sensor's electrons (Nixon & Aguado, 2012).

The Charged Coupled Device was invented in 1970 by Willard Boyle and since then has found their way into many products such as fax machines, scanners, digital cameras and photocopiers (Taylor, 1998). The Charged Coupled Device is charge driven device and the quantity of the charge locales gives the resolution of the Charged Coupled Device (Nixon & Aguado, 2012). The system can be thought of in the same way as a vertical and horizontal bucket system with rows and columns of buckets stacked next each other collecting rain water and shifted out. The contents of the charge are held in each pixel and emptied into vertical transport registers and shifted towards the horizontal transport registers into a signal conditioning unit where they are amplified and converted by an analogue to digital converter (Nixon & Aguado, 2012). There are four different architectures of area array CCDs namely full frame, split frame transfer, frame transfer and interline transfer (Jähne & Haussecker, 2000).

Vidicons are photo-conductive devices that use photo-sensitive sensor layers that consist of millions of mosaic cells that are insulated from another on a transparent metal film (Vernon, 1991). They are older, analogue and cheaper technologies but are being replaced by CCD and CMOS technologies. They work like simple analogue TV, yet in converse. The picture is actuated on a screen and detected by an electron beam that is horizontally filtered. The yield is consistent and the voltage is relative to the brilliance of the scanned line (Nixon & Aguado, 2012).

CMOS sensors are voltage driven sensors. When light enters the sensor, it creates a voltage which is proportional to the light that enters the sensor. The signal is sampled individually per pixel, converted to a digital signal and emptied for the next frame to be taken. The output from the sensor is a digital output (Nixon & Aguado, 2012).

2.7 Image Noise

2.7.1 Impulsive Noise

Impulse noise, otherwise called salt and pepper noise, are irregular events of dull and white spots (Jain et al. 1995). The noise is brought about by transmission mistakes, defective pixels in a Charged Couple Device array or external noise influencing analogue to digital conversion (Trucco & Verri, 1998). Figure 2.9 (b) demonstrates the impacts of salt and pepper noise in a picture. The noise was impelled onto the first picture in Matlab with the Computer Vision Toolbox.

2.7.2 Gaussian noise

Gaussian noise are variations in intensity drawn from a Gaussian distribution (Jain et al. 1995). The Gaussian noise model is often used when the characteristic of the noise cannot be defined. Figure 2.9 (c) demonstrates the impacts of Gaussian background noise in a picture in Matlab with a mean estimation of zero and a difference of 0.01.

Figure 2.9 (a): Original image (b) Impulsive noise in image (c) Gaussian noise in an image

2.8 Image Filtering

The purpose of image filtering is to enhance images (reduce noise, reshape or resize, change contrast), extract information from images (texture and edges) and to detect patterns (template matching). In computer vision, noise can be defined as any entity in an image that is not important for the purpose of the main computation (Trucco & Verri, 1998). According to Vernon (Vernon, 1991) noise is the undesirable impedance on a video signal which is characterized quantifiably by the signal to noise proportion *(S/N)* which is the adequacy of the desired signal to the average on the signal noise. Noise can be assumed to be an additive and random signal $n(x, y)$ added to the true pixel values $I(x, y)$ which are expressed in equation (2.5). Figure 2.9 (a) illustrates an image with relatively low noise.

$$\hat{I}(x, y) = I(x, y) + n(x, y) \tag{2.5}$$

2.8.1 Types of filters

2.8.1.1 Mean filtering

There are different types of filtering used in computer vision. Mean filtering is a linear filtering method and is the simplest method of filtering used in image filtering. With mean filtering each pixel intensity value is replaced by a new value taken over the local neighbourhood of fixed size (Jain, et al., 1995). It is assumed that the neighbouring pixels are similar and that the noise is independent from pixel to pixel. Figure 2.10 illustrates the mean filter.

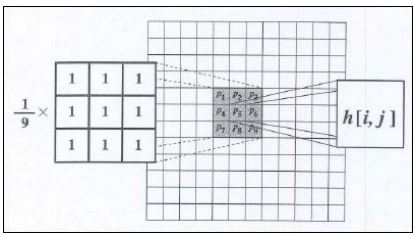


Figure 2.10: Mean filter showing 3x3 neighbourhood (Jain, et al., 1995)

2.8.1.2 Median filtering

In median filtering each pixel is replaced with the median of the grey values in the local neighbourhood. With median filtering the region centred around the pixel (i, j) is taken, the intensity values are arranged in ascending order and the middle value is taken as the new value for pixel (i, j) (Jain et al. 1995). Figure 2.11 illustrates the median filter.

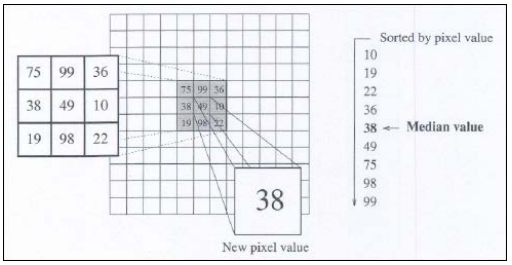


Figure 2.11: Median filter showing 3x3 neighbourhood (Jain, et al., 1995)

2.8.1.3 Gaussian filter

The Gaussian filter smoothing filter is a 2-D convolution filter that is used to obscure pictures and abstract detail and is utilized to undo Gaussian noise. It is performed by using a weighted average of the surrounding pixels based on Gaussian distribution. In mathematics, convolution is the process whereby two signals or in this case two images are combined to form a new image. Gaussian filters are the most commonly used filter (Nixon & Aguado, 2012). They are computationally efficient and they act as linear low pass filter. The Gaussian 1-D distribution is shown in equation (2.6) and illustrated in Figure 2.12.

$$f(x) = \frac{1}{(\sqrt{2\pi\sigma})} e^{-x^2/2\sigma^2} \quad (2.6)$$

In two dimensions it is the product of two 1-D Gaussians shown in equation (2.7) and in Figure 2.12, where x is the distance from the horizontal axis and y is the distance from the origin in the vertical axis. The standard deviation of the Gaussian distribution is expressed as σ.

$$f(x, y) = \frac{1}{2\pi\sigma^2} e - \frac{x^2+y^2}{2\sigma^2} \quad (2.7)$$

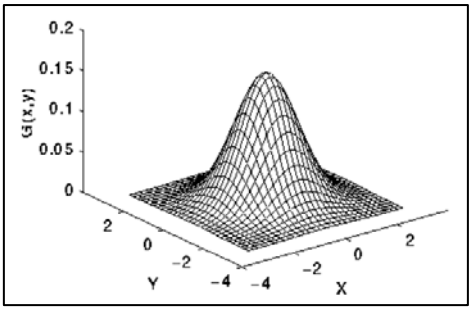


Figure 2.12: Gaussian distribution

2.9 Feature Detection, Extraction and Point Feature matching

2.9.1 Edge, corner, and feature detectors

An edge occurs when there are changes in light intensity of an image (Jain, et al., 1995). According to Verri, edge points are pixels around which the image values undergo sharp variations (Trucco & Verri, 1998). An Edge is also defined as a sharp variation of the intensity of the image. In greyscale images and edge is the brightness intensity of the pixels and in colour images an edge can also be the sharp variations in colour of the image. Edges normally indicate border regions between objects in an image. The process of detecting edges in an image is called edge detection. Edge detection looks at the relationships between neighbouring pixels and if the grey level intensity value is similar between adjacent edges there will probably not be an edge, but if the adjacent pixels vary greatly, the chance of there being an edge is higher.

Edges are important features of an image as they indicate a significant change in the image. The goal of edge detection is to produce a line drawing of a scene from an image of that scene (Jain, et al., 1995). Figure 2.13 (a) shows an image of a connecting rod. Figure 2.13 (b) shows a plot indicating changes of the image on a horizontal slice through the circular part of the connecting rod in Figure 2.13 (a). Significant changes can be seen at points 50, 60, 140 and 155 on the horizontal axis, but some of the changes are not edges because they are not significant changes and some are due to noise in the image (Jain, et al., 1995).

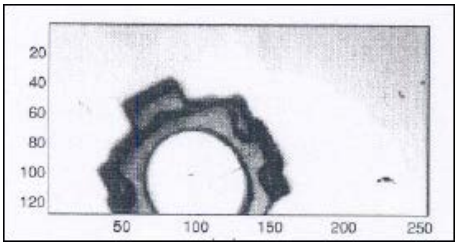
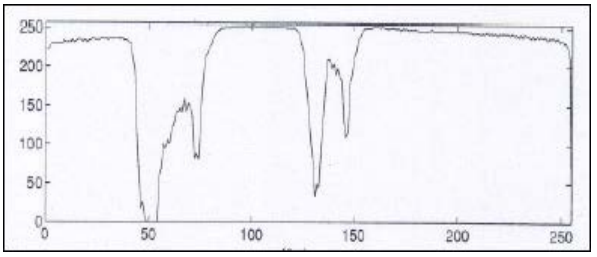


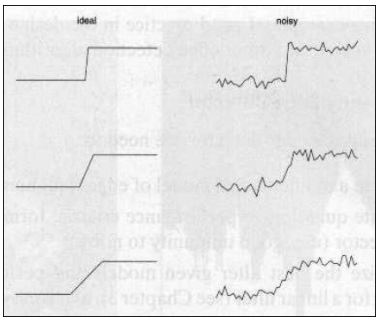
Figure 2.13 (a): Top half of connecting rod

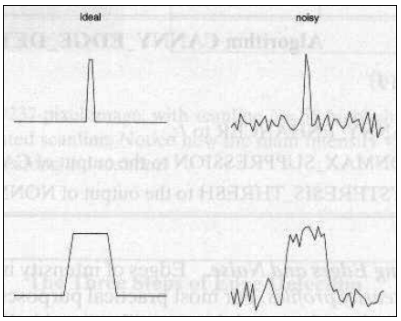


(b): Intensity plot of a step change in the image (Jain, et al., 1995).

The four steps of edge detection are noise smoothing, edge enhancement, edge detection and edge localisation. The process of noise smoothing is to eliminate as much noise as possible without altering the true edges in the image and edge enhancement is the filter that responds to edges in an image so that the output of the filter is large at the edge pixels and low elsewhere in the image. Edge detection is the process of identifying which edges are true edges and which are actually caused by noise in the image. Edge localization is used to determine the exact location of an edge (Trucco & Verri, 1998).

There are different types of edge descriptors each with their own unique properties as shown in Figure 2.14 (a), (b) and (c). Figure 2.14 (a) shows examples of step edges. A step edge is where the image intensity suddenly changes to a different value. A ramp edge is a step edge where the change is not sudden, but occurs over a certain distance. Figure 2.14 (b) shows ridge edges. Ridge edges occur when the image intensity suddenly changes value but returns to the starting value within some short distance. Lines are usually responsible for this type of edge. Figure 2.14 (c) illustrates a roof edge. Roof edges occur when the change is not sudden but happens over distance (Trucco & Verri, 1998).





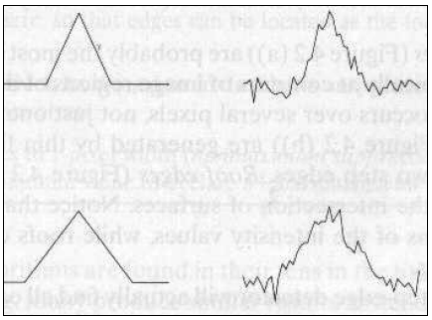


Figure 2.14 (a): Step edges (b) Ridge edge (c) Roof edge (Trucco & Verri, 1998)

There are many different types of edge detector algorithms used in computer vision including Sobel, Roberts Edge detector and Prewitt. The most commonly used edge detector was developed by John Canny in 1986 and is referred to as the Canny edge detector but is also known as the optimal detector.

2.9.1.1 Roberts Edge Detector

The Roberts operator is a differential operator and was developed by Lawrence Roberts in 1963 and was one of the first edge detectors (Roberts, 1980). Differentiation is used to estimate the image gradient by calculating the sum of the squares of differences between neighbouring pixels (Vernon, 1991). The original image is convolved with two masks *Gx* and *Gy* in equation (2.8) and equation (2.9).

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.8}$$

$$G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \tag{2.9}$$

Where $I(x, y)$ is a pixel in the original image and $G_x(x, y)$is the new pixel as a result of convolution with equation (2.8) and $G_y(x, y)$ is the result of the convolution with equation (2.9). The gradient is calculated in equation (2.10) and the direction of the gradient is calculated in equation (2.11).

$$Gradient\ |G| = \sqrt{G_x^2 + G_y^2} \tag{2.10}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.11}$$

The method is applied on the image in Figure 2.15 and the result is shown in Figure 2.16 (a).

2.9.1.2 Sobel's edge detector

The Sobel operator, named after its inventor Irwin Sobel is another edge detector often used in edge detection algorithms). The Sobel edge detector uses a pair of 3x3 convolution masks as in equations (2.12) and (2.13), to detect the absolute gradient magnitude *Gx* in a vertical direction (rows) and the absolute gradient magnitude *Gy* in the horizontal direction (columns) across an image. The convolution mask, which is smaller than the image is placed over the image to manipulate a square of pixel at a time (Green, 2002).

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad (2.12)$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (2.13)$$

The gradient is calculated by equation (2.14), the magnitude is calculated by equation (2.15) and the angle is equated by equation (2.16).

$$Gradient\ |G| = \sqrt{G_x^2 + G_y^2} \quad (2.14)$$

$$Magnitude\ |G| = |Gx| + |Gy| \quad (2.15)$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad (2.16)$$

This method is shown in Figure 2.16 (b) as applied on the original image (after filtering and conversion to greyscale) in Figure 2.15

2.9.1.3 Canny Edge Detector

The Canny edge detector is the most widely used edge detection method. This method is more stable against noise in an image than other edge detectors and better at detecting weak edges. The Canny edge detector does not make use of convolution mask-based operators. The detector

has three performance criteria which are as follows: 1) Good detection – the probability of failing to mark real edge points and the probability of false detection should be low. 2) Good localisation – the edges marked as edge points should be as close as possible to the true edge. 3) Minimal response – if an edge is detected, it should be marked only once. If another edge is detected it will be considered as false (Canny, 1986). The Canny edge detector uses a grayscale image as input and returns a dual level image in which non-zero pixels mark detected edges. This algorithm has four stages; Image smoothing by a Gaussian function, Differentiation of the smoothed image, Non-Maximum Suppression and Edge thresholding (Intel Corporation, 1999). First the Canny edge detectors eliminates the noise by smoothing then tracks along the regions to suppress pixels that are not at the maximum, thereafter the gradient array is reduced by means of hysteresis which utilises two thresholds but if the magnitude is below the first, is made zero or non-edge (Green, 2002).

After noise filtering by means of Gaussian suppression, the image is filtered by means of the Sobel operator in the horizontal direction *Gx* and in the vertical direction *Gy*. From the resultant image, the gradient and the direction can be determined in equation (2.17) and equation (2.18).

$$Gradient\ (G) = \sqrt{G_x^2 + G_x^2} \tag{2.17}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.18}$$

The method is applied on the image in Figure 2.15 and the resultant image is shown in Figure 2.16 (c).

2.9.1.4 Prewitt edge detector

With the Prewitt operator the edges are calculated by checking the gradient of the pixel intensity. By this means the direction and rate of change from a high intensity to low intensity per pixel is measured. For example the edge is determined by calculating how quickly or slowly the edge intensity changes. A higher rate of change in pixel intensity means that there is a higher probability that an edge exists and a lower rate of change means a lower chance of whether or not an edge exists in the image. The magnitude calculation derives the likelihood of

whether an edge exists or not. The Prewitt operator uses a 3x3 convolution mask that is convolved with the original image to calculate changes in the horizontal direction *Gx* and changes in the vertical direction *Gy* of the image where *A* is the original image as in equation (2.19) and equation (2.20).

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * A \tag{2.19}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * A \tag{2.20}$$

The gradient magnitude is calculated in equation (2.21) and the angle is calculated in equation (2.22).

$$Gradient\ |G| = \sqrt{G_x^2 + G_x^2} \tag{2.21}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.22}$$

The result of this method as applied on the image in Figure 2.15 is depicted in Figure 2.16 (d).

The following images show the outputs of two edge detectors in Matlab's Computer Vision Toolbox. The original RGB image is converted to grayscale and the filters are thereafter applied to the grayscale image. The results show that the Canny edge detector is the optimal edge detector, hence its popularity.

Figure 2.15: Original RGB Image

Figure 2.16 (a) Roberts edge

(b) Sobel edge detector

(c) Canny edge detector

(d) Prewitt edge detector

2.9.2 Corner detection and Blobs

2.9.2.1 Harris corner detector

A corner in an image is a position in a two dimensional array where the gradient direction of pixels changes rapidly. In a binary image a corner is the junction where two straight lines meet and is also defined as the point where two edges intersect with large variations in gradient in the neighbourhood of the point across all directions, which make them good features to match (Gonzalez, et al., 2010). Blobs are regions of similar pixels in an image which could be a group of pixels of the same colour linked in a neighbourhood of pixels which might represent skin tone, hair or shiny metal (Demaagd, et al., 2012). When a blob is identified the area within the blob can be used to find dimensions, the centre point of the region, rotation or angles and proximity to circles or other shapes.

A popular corner detector is the Harris corner detector proposed by Chris Harris and Mike Stephens in 1988 in a paper called "A combine Corner and Edge Detector". The Harris corner detector is an improvement of the Moravec detector proposed in 1980 and finds the difference in intensity values for the shift of *(u,v)* in all directions and is calculated by equation (2.23) (Gonzalez, et al., 2010).

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \quad (2.23)$$

In which $w(x,y)$ is the window function, $I(x+u,y+v)$ is the shifted function and $I(x,y)$ is the intensity.

For small shifts of $[u,v]$:

$$E(u,v) = au^2 + 2buv + cv^2 \quad (2.24)$$

Can be written as:

$$E(u,v) \cong [u,v]M\begin{bmatrix}u\\v\end{bmatrix} \quad (2.25)$$

Where M is:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (2.26)$$

Now analyse the eigenvalues λ_1, λ_2 of M which indicates the intensity change in equation (2.27). If both eigenvalues λ_1, λ_2 are small, the intensity change is constant which indicates a flat region. An edge is detected if one eigenvalue λ_1 is high and the other eigenvalue λ_2 is low. If both eigenvalues λ_1 and λ_2 are high the possibility is high that this is where a corner is located in the image.

$$\lambda_1, \lambda_2 = \frac{a+c}{2} \pm \left[\frac{4b^2+(a-c)^2}{2}\right]^{\frac{1}{2}} \quad (2.27)$$

Measure the corner response and describe the point in terms of eigenvalues:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad (2.28)$$

Where, *R* is expansive and positive for a corner, however, substantial and negative for an edge. *R* is small for a flat area in the picture (Gonzalez, et al., 2010).

2.9.2.2 Laplacian of a Gaussian (LoG) corner detector

The LoG operator combines the Laplacian with Gaussian filtering for edge detection and is known as the Laplacian of Gaussian. The Laplacian of Gaussian (LoG) edge detector can be describes in 4 steps. Step 1) the image is smoothed by means of the Gaussian function. Step 2) the image is enhanced with the second derivative. Step 3) the edge points are determined by means of finding the zero crossing of the second derivative of the image intensity. Step 4) linear interpolation is used to approximate the edge location (Jain, et al., 1995). In the first step the image is convolved with the Gaussian filter to smooth out any noise in the image. The result of filtering is smoothing so that only edges with maximum gradients are considered. The smoothing is achieved by applying the zero crossings of the second derivative and thereafter the Laplacian is used to estimate the second derivative. Non-edges are avoided by selecting only edges that are above a certain specified threshold (Jain, et al., 1995).

The image is convolved with the Gaussian smoothing filter equation (2.29)

$$L(x,y) \times G(x,y) \times f(x,y) \tag{2.29}$$

Image convolution is given in equation (2.30).

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.30}$$

The Gaussian is given by equation (2.31).

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.31}$$

The image in Figure 2.17 is the result after the LOG corner detector has been applied to the image in Figure 2.15.

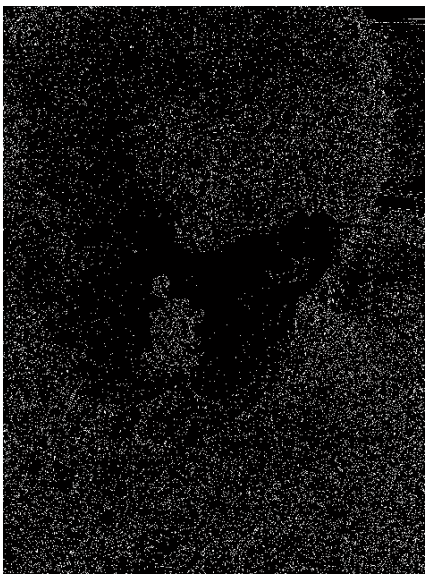


Figure 2.17: LOG

2.10 Hough transform

The Hough transform is a feature extraction technique to find shapes in images using a poling system of votes to segment out objects of which the shapes are known. The shapes to be segmented in the image can be straight lines, circles or any shapes if the parametric equation for the shape is known. The Hough Transform was first introduced in a Patent by Paul Hough in 1962 but only used in computer vision a decade later by Richard Duda and Peter Hart in 1972 to detect lines in images (Duda, et al., 2012).

An image in a 2 dimensional image plane *x, y* is made up of points and some of these points may be connected forming straight lines. The simplest form of Hough Transform is the detection of straight lines. The equation for a straight line is represented by equation (2.32), but this does not work for vertical lines therefore the equation (2.33) is used and represented in Figure 2.18.

$$y = mx + c \tag{2.32}$$

$$\rho = x\cos\theta + y\sin\theta \tag{2.33}$$

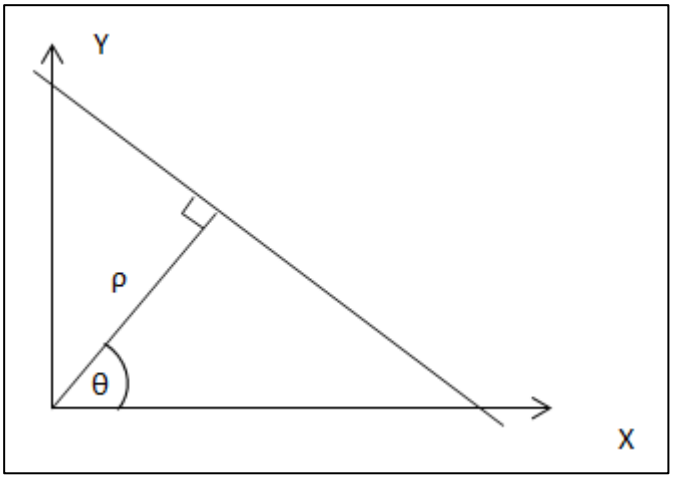


Figure 2.18: Image domain

Here rho ρ is the distance from the coordinate system's origin to the closest point on the straight line and the angle theta is the angle of the vector between the x axis and the point on the straight line (Shapiro & Stockman, 2001). Any single point can have an infinite number of straight lines going through it each with rho and theta with respect to the origin. The algorithm

uses an accumulator array to find the straight lines $y = mx + c$ where the size of the array is proportional to the number of unknown parameters for rho and theta. For each point (x,y) the Hough Transform evaluates the possibility of there being a straight line at the pixel. If a straight line is found, the rho and theta value is calculated for the straight line and quantized to correspond to values in the accumulator and incremented for the bin it corresponds to. When all the pixels have been processed, the array is searched for peaks which may indicate the parameters that are most likely to be lines in the image (Shapiro & Stockman, 2001). Therefore the elements in the accumulator with the highest number of increments or votes are the most likely to be straight lines.

Circles are found in the same way where possible circles are estimated by means of voting to select the local maxima in the accumulator array. The equation of the circle is shown in equation (2.34) where x_0 and y_0 are the centre and the radius r, are unknowns that has to be determined similar to how rho and theta is determined for the straight line, however for the straight line 2 coordinates (x,y) had to be determined but for the circle three coordinates (x_0, y_0, r) are determined (Shapiro & Stockman, 2001).

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{2.34}$$

Figure 2.19 illustrates the application of the Hough transform in MATLAB. Here circles are found and each circle is highlighted by dotted lines.

Figure 2.19: Application of Hough transform in MATLAB

2.11 Scale Invariant Feature Transform (SIFT)

Scale-invariant feature transform or SIFT is a local features descriptor algorithm first proposed by David Lowe in 1999. SIFT performs object detection by means of regional descriptors around local feature points. The SIFT operator is invariant to change in illumination orientation, scaling and distortion and can detect objects in a cluttered scene with partial occlusion (Lowe, 1999). The next image in Figure 2.20 shows how SIFT is invariant to scale, orientation and background clutter. Multiple feature points can be detected with high efficiency close to real time detection.

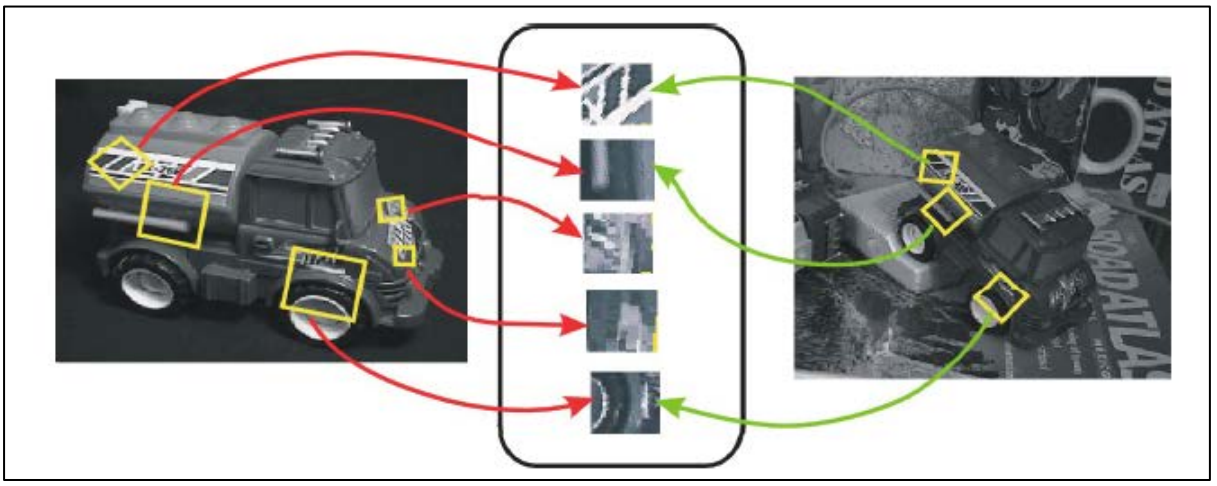


Figure 2.20: SIFT extraction

SIFT has a training phase where SIFT features are extracted from one or more training images of an object and placed into a database and a testing phase where SIFT features are removed from a test picture and these features are coordinated to the features in the database. Matches are found with one or more of the training images and a transformation is calculated from the training image to the test image. If consistencies are found, the object is positively recognised. The SIFT algorithm has various stages for detection. The stages are Scale space peak selection, Keypoint localization, Orientation assignment and Keypoint descriptor (Szeliski, 2010).

2.12 Speeded Up Robust Features Detector (SURF)

The Speeded Up Robust Features Detector (SURF) is based on the same concepts applied in the SIFT detector but has a better performance over SIFT. The other goals of SURF are to be fast, maintain comparability with other detectors and to have a high rate of repeatability under different operating conditions. The SURF algorithm consists of three steps. These steps are; 1) interest point detection, 2) local neighbourhood description and 3) matching (Davies, 2012) .

An example of the SURF detector is shown in Figure 2.21. Circles are the points of interest where the sizes of the circles represent the scale. In this image a hundred points of interest are detected.

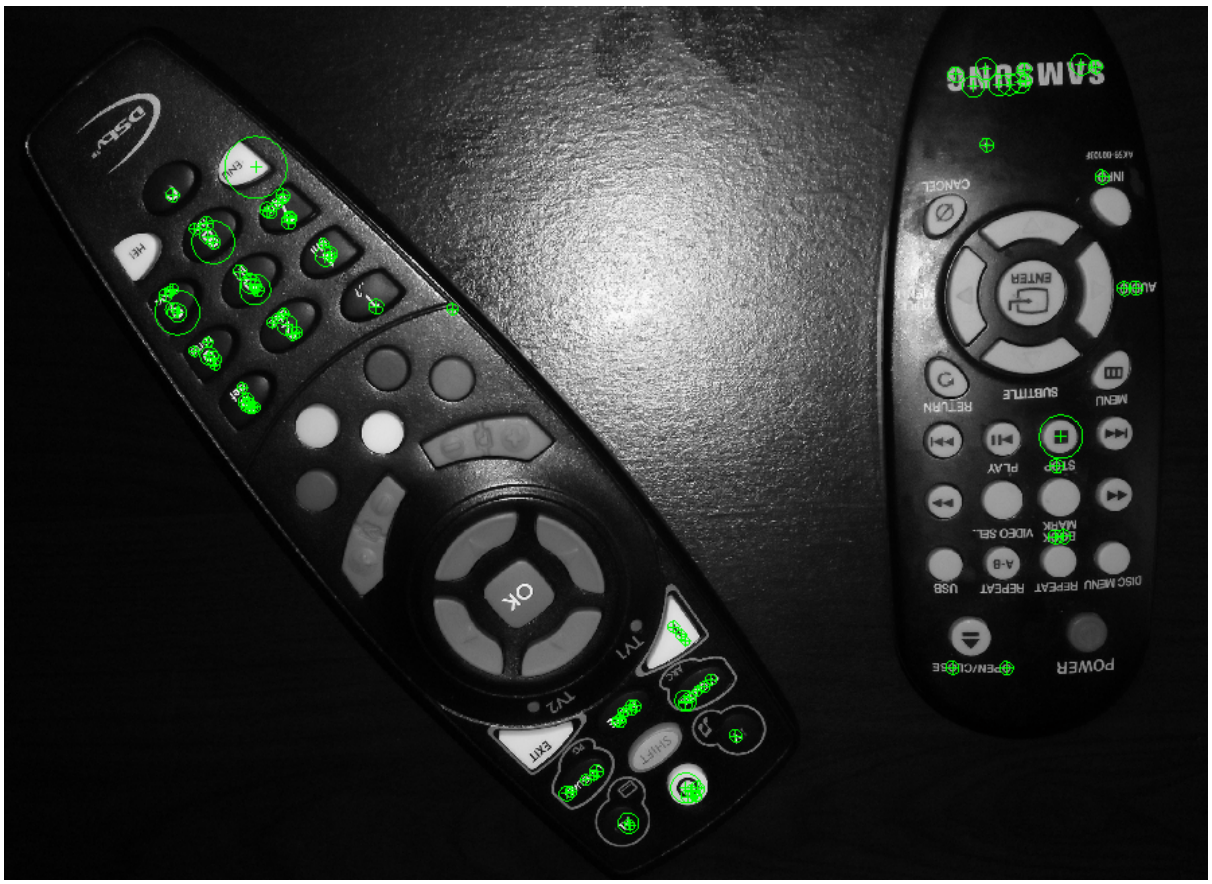


Figure 2.21: SURF detector

2.13 Maximally stable extremal regions detector (MSER)

The Maximally stable extremal regions detector (MSER) is a blob detection method that is invariant to scale and rotation. A Maximally stable extremal regions detector is a set of stably connected component of grey level images that takes regions that are nearly identical through a set of thresholds where pixels below a certain threshold are defined white and any above the threshold are defined equal or as black. So, MSERs are regions in an image that are either darker, or brighter than the surrounding regions. Ellipses can be added to regions which are kept as features in an image. The image in Figure 2.22 describes MSER regions and corresponding ellipses that have the same second moments as the regions (MATLAB, 2015).

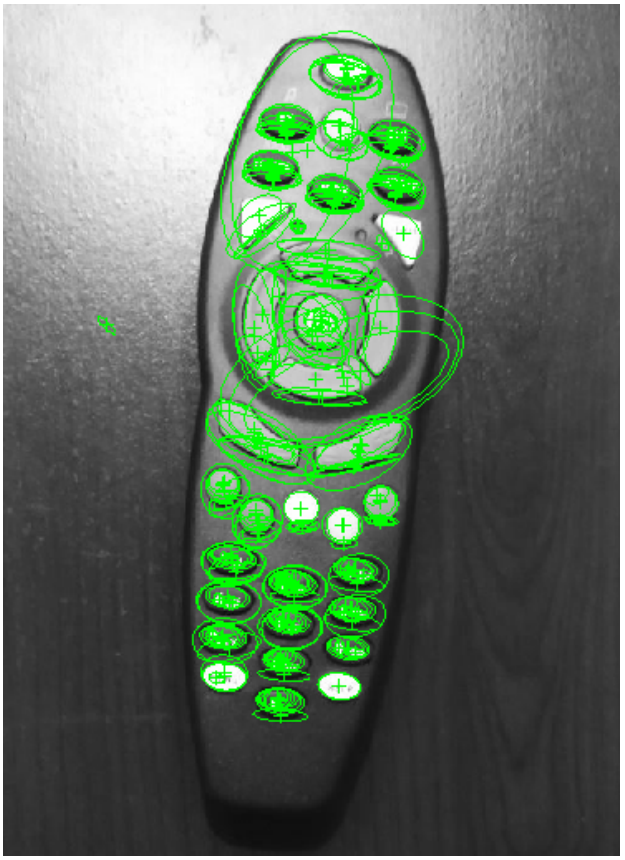


Figure 2.22: MSER detector in MATLAB

2.14 Random Sample Consensus (RANSAC) Algorithm

The RANSAC algorithm is a system for evaluating a numerical model from a data set that contains outliers and was proposed by Fischler and Bolles (Fischler & Bolles, 1981). The

calculation in RANSAC recognises outliers in a data set and evaluates the model utilizing data that does not have any outliers. The algorithm works as follows: Assume we have *M* data sets, the parameters are estimated from *N* number of data items, the likelihood of a haphazardly chosen data item being a decent model is *P* and the likelihood that the calculation will exit without discovering a solid match if one exists is *Pfail* (Fischler & Bolles, 1981).

RANSAC is performed by means of the following steps and further described in Figure 2.23:

- Randomly select points and estimate model parameters
- Solve the model parameters
- Estimate the numbers of outliers *K*
- If outliers *K* exceed the threshold, re-evaluate the parameters. If not, accept the fit and exit.
- Repeat the steps for *L* number of times where *L* is determined in equations (2.35) and (2.36).

$$Pfail = (1 - (P)^N)^L \tag{2.35}$$

$$L = \frac{\log(Pfail)}{\log(1-(Pg)^N)} \tag{2.36}$$

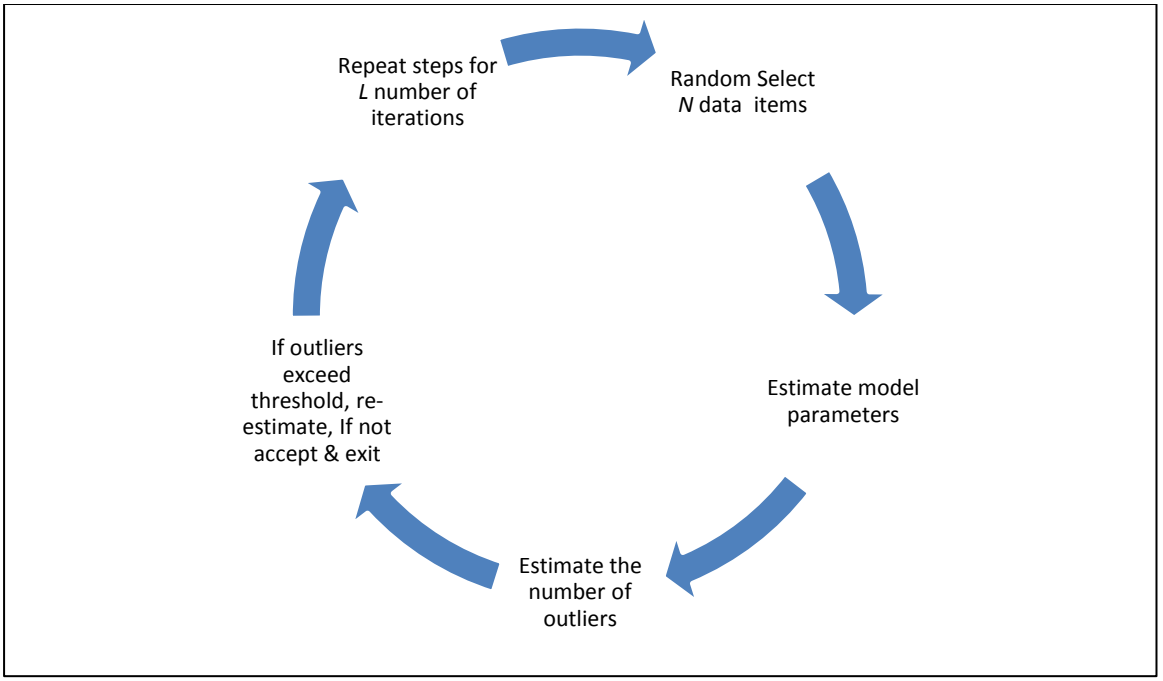


Figure 2.23: RANSAC

CHAPTER 3
MS KINECT SENSOR

3.1 Introduction

The Kinect sensor was developed by 3D sensing company PrimeSence and is licenced by Microsoft. The Kinect Sensor as shown in Figure 3.1 is a device that contains cameras, a microphone array, an accelerometer as well as software that processes colour, depth, and skeleton data (Microsoft Developer Network, 2013). The Sensor was released on November 4 2010, has 640x480 colour camera and a 640x480 infrared camera (Tuvshinjargal, et al., 2015). The RGB frame rate is 30 frames per second with an ideal range of 1.2 m to 3.5 m and can track up to six people at the same time (Microsoft, 2013).

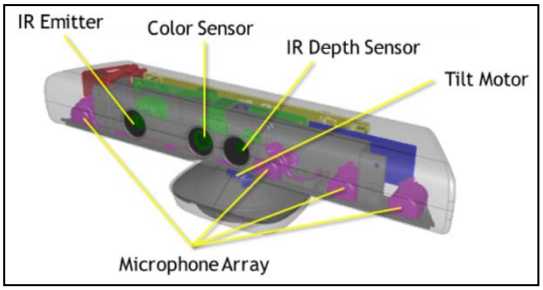


Figure 3.1: MS Kinect (Microsoft, 2013)

Because there is no CPU inside the sensor, the data processing is executed on the host device by a Kinect driver. Colour image capturing is made possible by the RGB camera that stores a 1280x960 resolution via three channel data. The IR emitter sends infrared light beams. The infrared light beams, invisible to the human eye due to its wavelength, is bounced off objects in the camera's line of sight and bounced back to the infrared sensor. The light reflected is turned into depth information by the Kinect's hardware (Jia, et al., 2015). This gives a measurement of distance between the reflected infrared pixel at the object and the sensor. The IR sensor is used for depth information and camera calibration. The Kinect sensor has a viewing angle of 43 degree vertical by 57 degree horizontal field of view and a vertical tilt angle of 27 degrees (Microsoft, 2013). The frame rate is 30 frames per second with a 16kHz, 24 bit PCM audio format. The accelerometer has a 2G range with 1 degree accuracy (Microsoft, 2013).

The Sensor has, over recent years, been adopted to a wide range of applications which include robotics and human-computer-interaction (Chávez-Aragón, et al., 2013). Examples of the types of applications that can be built using the functionality supported in the Kinect for Windows and Microsoft SDK are tracking moving people, determine distance between objects, capture audio and enable voice-activated applications, healthcare, robotics, education and training, security systems, and virtual reality (Jana, 2012). The SDK provides a software library and tools to help developers use Kinect-based input which senses and reacts to events. The Face Tracking SDK contains a couple sets of APIs (Application Program Interfaces) that can be utilized to track a face (Microsoft, 2013).

3.2 Preparing a New Project

The software is written in the C# programming language using Visual Studio 2010 with the Kinect for Windows SDK and its Natural User Interface Libraries. Figure 3.2 shows graphically how the Kinect sensor hardware streams the three types of data and interacts with an application via the NUI library.

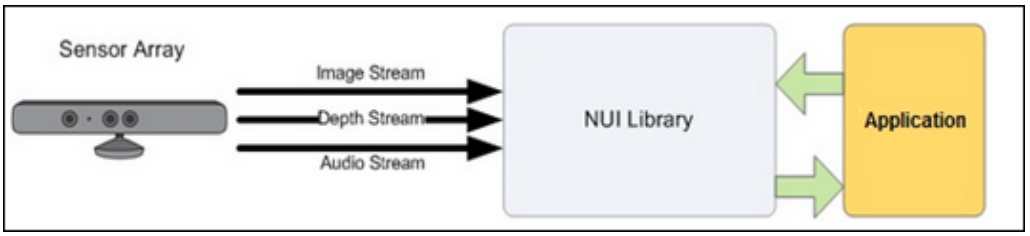


Figure 3.2: Hardware and Software Interaction with an Application

In Figure 3.3 a new project is started by selecting WPF and the project is given a name.

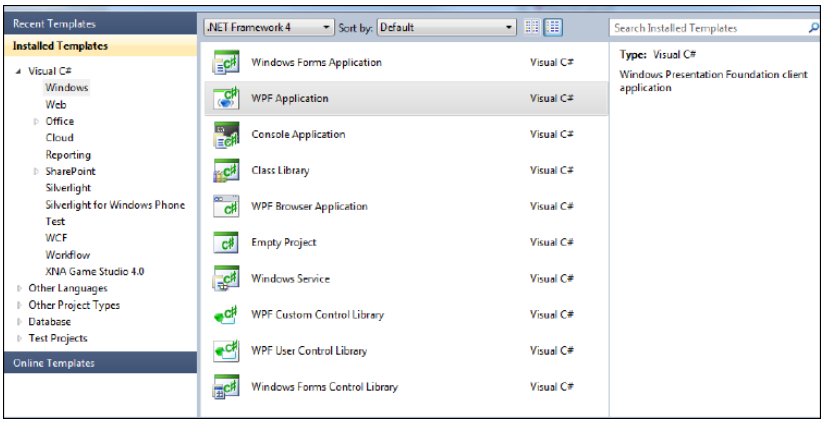


Figure 3.3: Starting a new Project

After the project is created a reference is added to the Kinect Windows Assembly as shown in Figure 3.4.

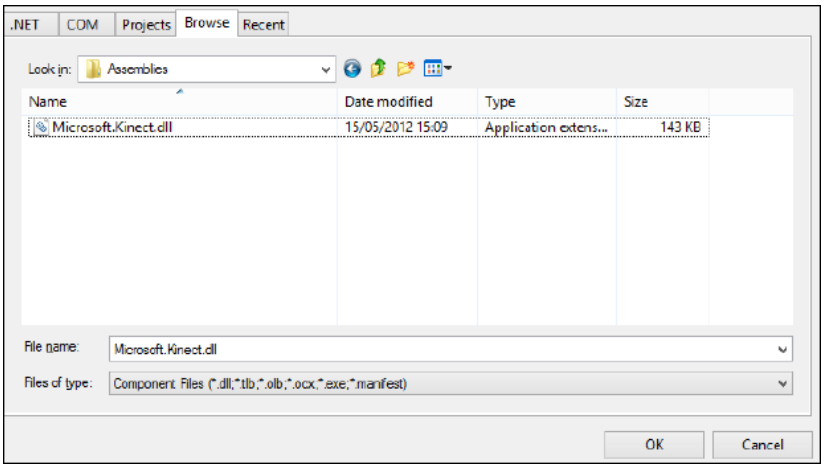


Figure 3.4: Adding a reference

3.3 Colour Stream

The video data is captured via a monochrome CMOS sensor. Different resolutions are possible in the colour image data encoded in RGB, YUV or Bayer. RGB format is possible with 640x480 resolutions at 30 FPS and 1280x960 at 12 FPS. YUV is a16-bit, gamma-redressed linear UYVY-formatted colour bitmaps, where the gamma rectification in YUV space equals sRGB gamma in RGB space and is possible at 640x480 at 15 FPS. The Bayer format, like RGB, is a 32-bit, linear X8R8G8B8-formatted colour bitmaps, in sRGB colour space. This format more proximately matches the physiology of the ocular perceiver by including more green pixels values than blue or red. It is infeasible to have a colour image stream and an infrared image stream open together because only one colour image stream is available per sensor (Microsoft, 2015).

3.4 Kinect Depth Sensor

The Kinect depth sensor consists of an IR emitter and an IR depth sensor and both of them collaborate to make things transpire. The IR emitter is an IR projector that interminably discharges infrared light in a discretionary spot pattern. These dots are customarily invisible to the human eye due to the wavelength. The projected dots reflect off surfaces, and the IR depth sensor receives the data from the surfaces of objects and converts it into depth data by

measuring the separation between the sensor and the item from where the IR speck was reflected. Depth data consists of pixels and is 11 bits long. The depth data is quantified in millimetres and is utilized to track kinetics or identify objects in the background at a particular (x, y) coordinate. The depth data withal contains player segmentation data where player segmentation values are integers designating the index of a unique player and is available in three resolutions: 640x480, 320x240 and 80x60. In every new frame the sensor takes a grayscale image of the scene within its viewing range and each pixel that is captured contains the Cartesian distance in millimetre at a coordinate (Microsoft, 2015).

3.5 Kinect Skeleton Tracking

Enabling skeleton tracking allows the Kinect sensor the functionality of tracking and recognising people and their movements. The Kinect uses the depth stream from the IR camera sensor to recognise up to six people in its field of view, with the possibility of recognising two people in detail. Up to 20 joints can be tracked at the same time. Skeleton tracking can track standing users or seated users facing the camera, but have some difficulty tracking users accurately with sideways poses. Skelton tracking is enabled by calling the C# method *SkeletonStream.Enable.* Two tracking modes are possible in managed code. To enable seated mode, the mode *SkeletonTrackingMode.Seated* is enbled, else the default mode is standing mode enabled with *SkeletonTrackingMode.Default*.

Joint and bone orientation is possible by means of hierarchical rotation based on the relationship defined on the joint structure or by means of absolute orientation in the coordinates of the sensor. The information on the joint orientation is provided by quaternions and rotation matrices. Joint orientation is enabled in managed code by the *BoneOrientations* property. Figure 3.5 shows an illustration of the joints that can be tracked with Kinect (Microsoft, 2015).

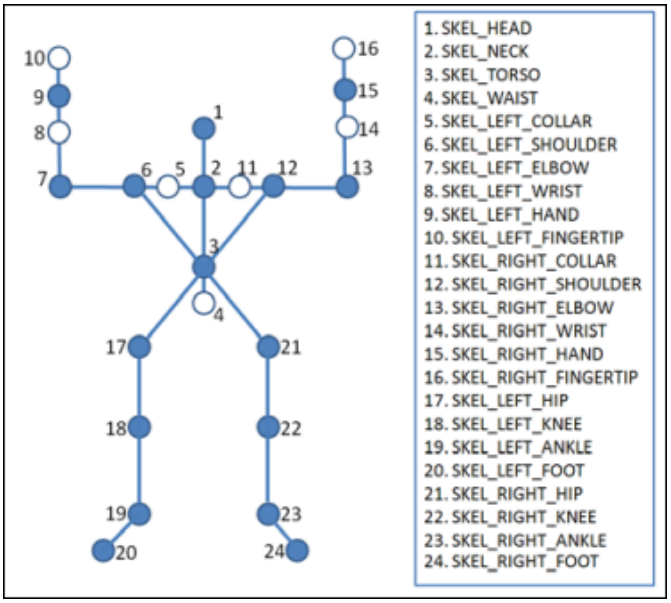


Figure 3.5: Skeleton positions relative to the human body

3.6 Face tracking with Kinect

The Face Tracking SDK uses the 3D coordinate system on the *X*, *Y* and *Z* planes as shown in Figure 3.6. The point of reference or zero point is located at the camera's optical sensor, where the *Z* axis is directed at the user, the *Y* axis is for vertical assessment and the *X* axis determines the horizontal plane.

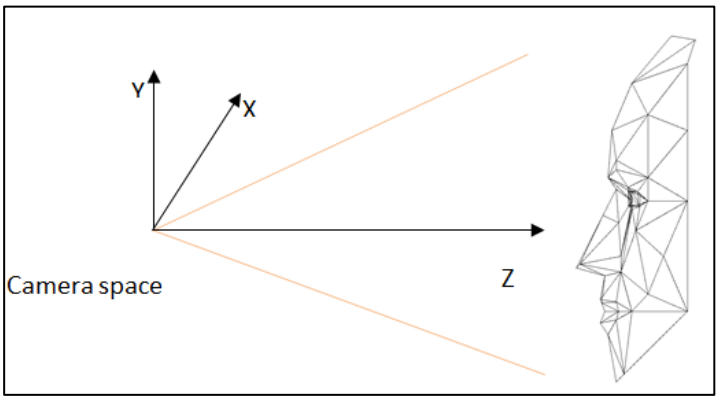


Figure 3.6: Camera Space (Microsoft, 2015)

According to Microsoft the Face Tracker Toolkit of the Kinect for Windows SDK, tracks the 2D facial points (Microsoft, 2015). Colour and depth streams are used by the Kinect Face Tracking SDK and therefore the accuracy of the results rely on overall image quality of each frame. Sharper frames track better than dark frames and better results are achieved if the face is closer to the optical sensor. The image in Figure 3.7 shows a 3D mesh of triangulated points representing numbered face units on a tracked face (Microsoft, 2015). For example feature

point number 0 represents the top of the skull known as the variable TopSkull and feature point 83 is the right top of the lower lip known as variable RightTopLowerLip. There are six Animation Unit Coefficients. The animation unit coefficient that identifies the lower jaw is known as the variable JawLower and is used to determine the opening and closing of the mouth. Head pose angles can also be determined.

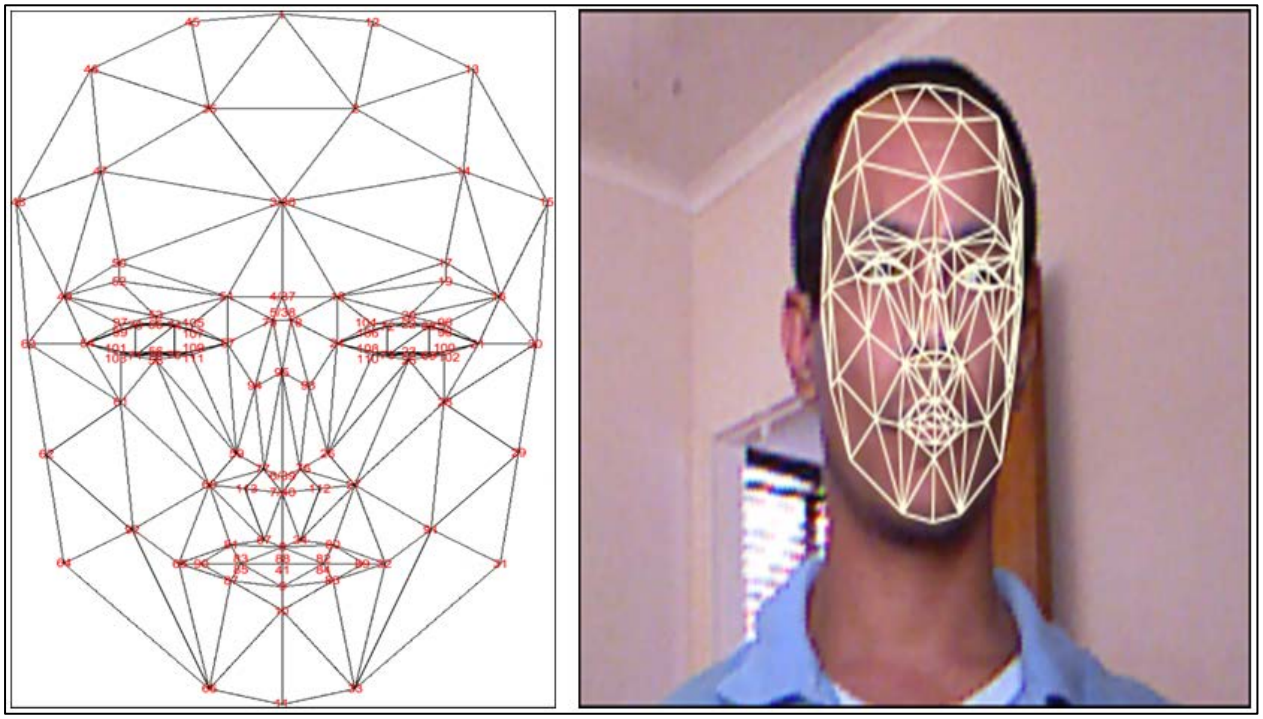


Figure 3.7: Kinect SDK Face tracking 3D mesh (Microsoft, 2015)

3.7 Benefits of Kinect Fatigue Detection System

Table 3.2: Advantages of the Kinect Fatigue detection system

Fatigue Technology	Non-Invasive	Real-time Monitor and Feedback	Vehicle-based measurement	Behavioural-based measurement	Detect Cellular Telephone
Driver Drowsiness Detection	Yes	No	Yes	No	No
Denso	Yes	Yes	No	Yes	No
Care-Drive MR668	Yes	Yes	No	Yes	No
Leisure Auto	Yes	Yes	No	Yes	No
Continental Driver Focus	Yes	Yes	Yes	Yes	No
Mercedes Benz Attention Assist	Yes	No	Yes	No	No
Toyota Driver Attention monitor	Yes	Yes	No	Yes	No
Ford Driver Alert	Yes	No	Yes	No	No
Volvo Driver Alert Control	Yes	No	Yes	No	No
FaceLAB 5	Yes	Yes	No	Yes	No
Kinect Sensor Fatigue Detection	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

As can be seen in Table 3.2 the Kinect Sensor Fatigue Detection System offers the better solution. The Kinect Fatigue detection system is non-invasive, offers real-time monitoring with feedback, and includes behavioural based measurement with cellular telephone detection.

The Kinect sensor has a high degree of accuracy and is relatively cheap. The accuracy is reduced if the Kinect sensor's distance is less than one meter from the object that is being tracked. The more readily available Kinect Version 1 for the Xbox 360 was used for this research project, but the Microsoft Kinect Version 2 for Windows is recommended. The Microsoft Kinect Version 2 for Windows includes near mode, which improves accuracy if the sensor distance is less than one meter from the object. The latest Kinect Version 2 sensor and SDK provide companies and developers a complete package to create interactive applications

that respond to gesture, voice, and movement. With the Kinect V2 sensor and SDK 2.0, the developer can create applications for commercial distribution.

For this application as in Figure 3.8, the Kinect sensor can be placed anywhere on the vehicle's dashboard that has direct line of sight to the driver and can be at any angle because of its wide viewing capability. The sensor must not obscure the driver's view. The Kinect requires a 12V, maximum 1.5A regulated external power source. For this, the vehicle's 12V accessory power socket can be used for in-vehicle testing. However, fuse protection will be required.

Figure 3.8: Kinect on dashboard

CHAPTER 4
EXPERIMENTAL DESIGN

4.1 Introduction

The purpose of experimental design is to describe how experiments will be conducted and to test the hypothesis. For pose estimation and object tracking, it is important for the tracking system to first identify the boundaries of a particular scene and the object of interest within that scene. The first order of experiments is to identify if the technology provided by the Kinect Sensor can be reasonably considered as a suitable tracking device based on the spatial limitations encountered within a vehicle's passenger compartment. For this experiment the following environmental conditions will be assumed: The camera is always stationary with direct unobscured line of sight and is immune to vibration. Lighting conditions will be sufficient for tracking and the object always remains within the boundaries of the scene.

The experiments are based on the Microsoft Kinect for Windows SDK v1.8 and MATLAB Computer Vision System Toolbox R2015a. The Computer Vision Toolbox gives algorithms, calculations, and applications for outlining and recreating computer vision and video processing frameworks. With the Computer Vision Toolbox one can create feature extraction, object tracking, motion calculation and feature matching. The Kinect for Windows Software Development Kit (SDK) is a very capable computer vision tool that empowers engineers to make applications that bolster computer vision solutions and voice acknowledgment utilizing the Kinect sensor innovation on PCs running the Windows operating system. Both Computer Vision Toolbox from MATLAB and the Kinect for Windows SDK will be used for the experimental evaluation of the Driver Attention and Behaviour Monitoring System.

4.2 Tracking with Kinect Sensor

The Kinect Sensor is ideal for computer vision applications such as the fatigue detection system. Included in the Kinect for Windows SDK is a tool for debugging and evaluation purposes. The tool is called Kinect Studio illustrated in Figure 4.1. Kinect Studio records and plays back depth and colour data streams from a Kinect Sensor and is used to create scenarios

for testing and evaluate application performance characteristics. In this chapter the results and the analysis of the results are discussed.

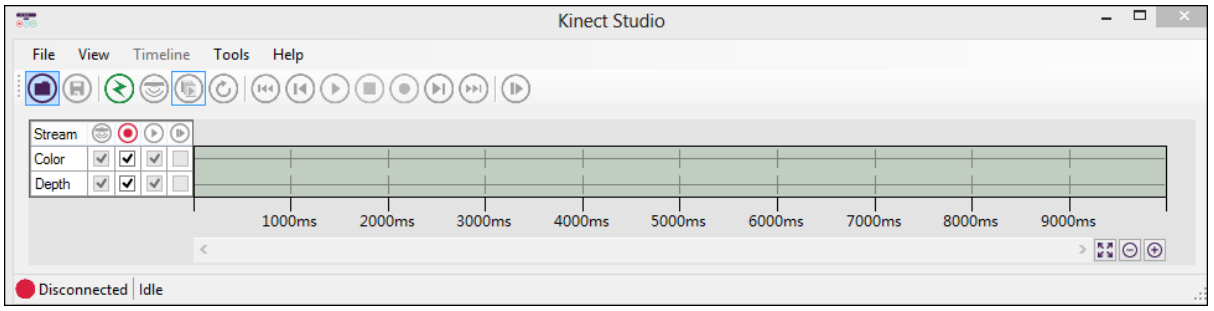


Figure 4.1: Kinect Studio

The following illustration in Figure 4.2 depicts the system design overview in a flowchart followed by the pseudo code of this design.

- On start-up the system checks the available Kinect Sensor in continues software routines.
- If a Kinect Sensor is detected, seated mode is enabled. In seated mode, and after joint filtering, the software algorithm will ignore lower limbs and joints.
- Focus is applied on the head, shoulders, arms and hands.
- The system checks for errors.
- If the Kinect Sensor is not detected or if any errors are found after start-up, an error message box is displayed.
- If no errors are found, the software routine continues with skeleton tracking, object detection and fatigue detection.
- If the use of a cellular telephone is detected, a message is displayed.
- If driver fatigue is detected a message is displayed.
- The software routine returns the loop to check for errors and continues object and fatigue tracking.

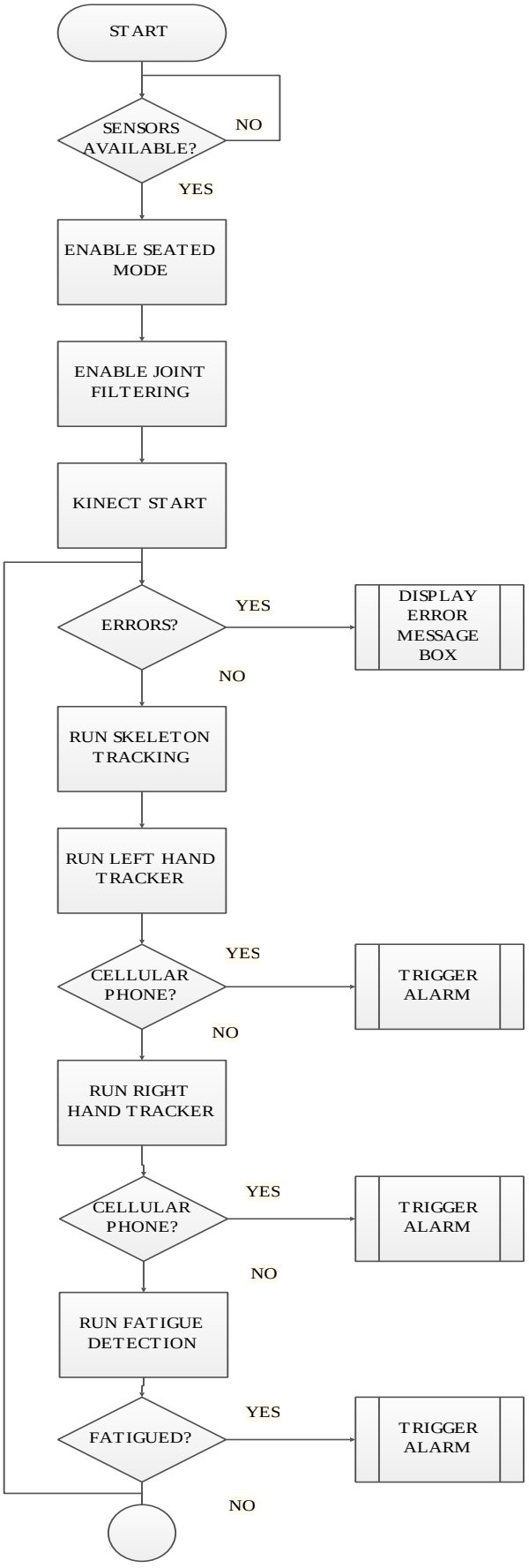


Figure 4.2: System flowchart

The following flowchart in Figure 4.3 is used to listen for any status change of the Kinect. A variable is assigned to the Kinect Sensor if the sensor is powered up and connected. Figure 4.4 is the pseudo code for this procedure.

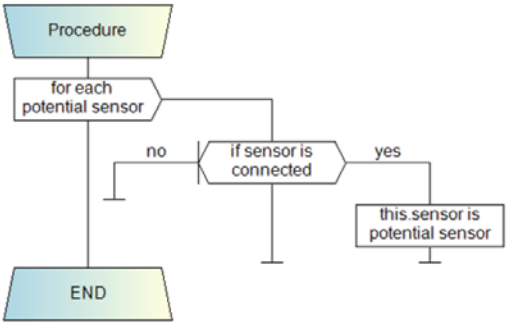


Figure 4.3: Flowchart for Kinect listen procedure

```
for each potential sensor
      if sensor is connected
            this.sensor is potential sensor
      EndIf
EndFor
```

Figure 4.4: Pseudo for Kinect listen procedure

Figure 4.5 shows the flowchart to enable the data streams and the enabling of the Kinect Sensor. The colour streaming, depth streaming and skeleton streaming are enabled. Figure 4.6 shows the pseudo code.

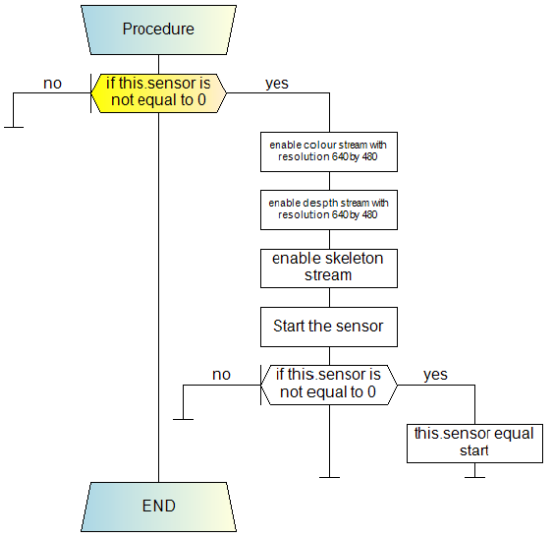


Figure 4.5: Flowchart for data streams

```
if this.sensor is not equal to 0

      enable colour stream with resolution 640 by 480
      enable despth stream with resolution 640 by 480
      enable skeleton stream
      Start the sensor
      if this.sensor is not equal to 0

            this.sensor equal start

      EndIf
EndIf
```

Figure 4.6: Pseudo for data streams

Figure 4.7 is the flowchart for joint tracking. The code is used for the head joint, the hand joint and the shoulder joint tracking. The *X*, *Y*, *Z* coordinates are displayed and the variables are used in the calculations to determine the Vector distances. The pseudo for this code is shown in Figure 4.8.

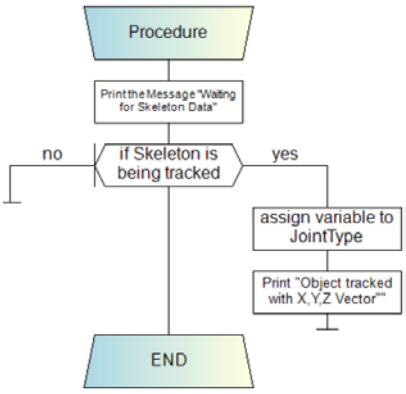


Figure 4.7: Flowchart for joint tracking

```
Print the Message "Waiting for Skeleton Data"
if Skeleton is being tracked
      assign variable to JointType
      Print "Object tracked with X,Y,Z Vector""
EndIf
```

Figure 4.8: Pseudo for joint tracking

The pseudo code for the vector mathematics is shown in Figure 4.10 and the flowchart is shown in Figure 4.9. The tracked joints have variables assigned to them e.g. *headjoint* or *handjoint*. The variables are used to determine the distances between the tracked joints.

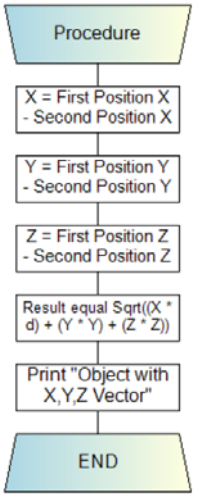


Figure 4.9: Vector math Flowchart

```
X = First Position X - Second Position X
Y = First Position Y - Second Position Y
Z = First Position Z - Second Position Z

Result equal Sqrt((X * d) + (Y * Y) + (Z * Z))

Print "Object with X,Y,Z Vector"
```

Figure 4.10: Vector math Pseudo

The first stage of the experimental evaluation is to test the Kinect Sensor's performance and its feasibility for use as the digital camera system for fatigue detection. The initial test is to enable the colour stream, showing the video being captured, the depth stream, showing depth of the objects in front of the sensor and the skeleton stream that enables skeleton joint tracking, seen here in Fig. 4.11.

Figure 4.11: The depth stream and the colour stream

The goal is to track the head joint and hand joints individually and return the 3D vector coordinates of the tracked joints. With the returned coordinates the relative head pose and hand positions are determined. The head pose is used to determine the head pitch, yaw and roll angles. These angles are used to determine whether the driver is nodding off or looking away from the direction of travel. The hand positions with respect to the driver's head position will determine whether a driver has a cellular telephone to his ear. In addition, the cellular telephone will be detected with object detection techniques and recognised.

4.3 Object Detection with Computer Vision

In the following scenes (Figure 4.13 to Figure 4.18) an image of a cellular telephone is detected using the MATLAB Computer Vision Toolbox. The first step in detecting this object is to first capture the scene; step 2 is to discover key characteristic points; step 3 remove the key attribute points and descriptors; step 4 will find characteristic point matches; step 5 localise the object within the matched points as described in Figure 4.12.

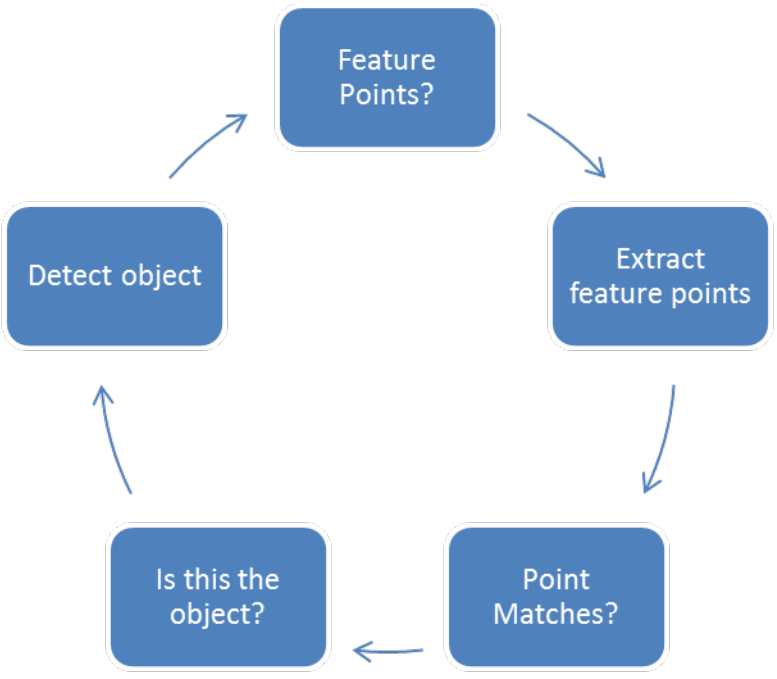


Figure 4.12: Object extraction

The image below in Figure 4.13 shows an image of a cellular telephone in MATLAB Computer Vision Toolbox and Kinect. This image represents the image of an object or reference image that will be used to detect a cellular telephone within a particular scene. Not all objects are the same colour, size or shape, but cellular telephones have a basic design and shape that has to be learned by the algorithm as the reference object required for recognition. Multiple images of cellular telephones can be learned. The scene may have random objects of similar or multiple differentiating features, but the objective of the algorithm is to detect the desired features of an object within the scene while ignoring others. Key feature points in this image are used to match similar feature points within the scene. The RANSAC algorithm is utilised to determine the matched features within the scene. The process of detecting the object is described next.

Figure 4.13: Image of cellphone

- First read the reference image of the object to be detected (Figure 4.13). In this case it's an image of a cellular telephone with a particular colour, size and shape.
- Detect the SURF feature points within the reference image. For every new image to be learned, the SURF feature points will be established.
- Extract the strongest feature points in the reference image. These feature points uniquely identify this scene. Cellular telephones have a basic design and shape.
- Read the scene and extract the strongest SURF feature points within the scene (Figure 4.14). The object within the scene is identified as having the shape of a cellular telephone. In a real world application the algorithm has to rely on the basic shape and size of a cellular telephone and if key features match will have to assume that this object within the scene can only be that cellular telephone.
- Find the Putative Point Matches between the reference image and the scene. If an apparent correlation is found between the reference image of a cellular telephone and the object in the scene.
- Display the matched features which may include outliers (Figure 4.15).
- Clean the image to only show inliers (Figure 4.16).
- Localise the object within the scene. This object can be regarded as a cellular telephone.
- Draw a boundary box around the detected object (Figure 4.17).

The following shows the steps as described above. The image in Figure 4.14 shows a scene with 3 objects of similar shape and size. The goal is to detect the object of interest, which in this example is the cellular telephone depicted in the centre of this scene.

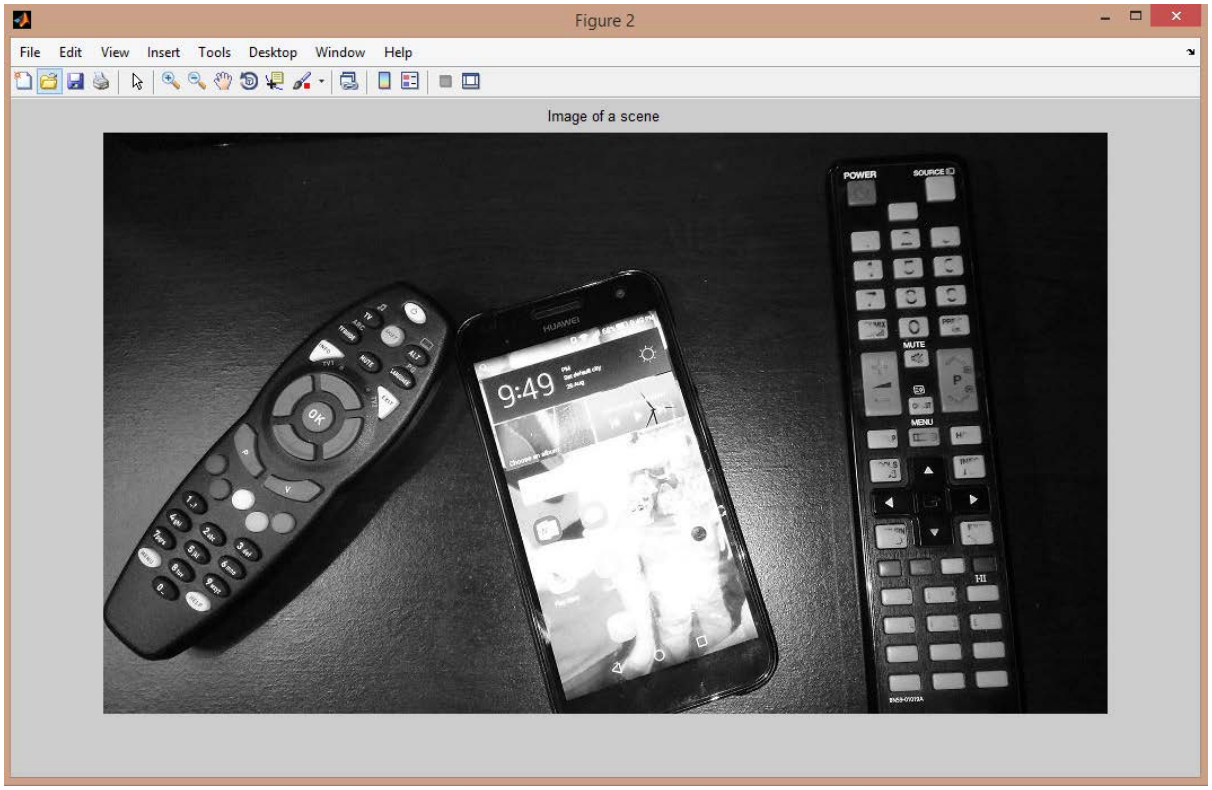


Figure 4.14: Image of a scene

In Figure 4.15 and Figure 4.16 the feature descriptors in the scene are extracted at the points of interest and matched to each other. Various points of interest were matched with only two outliers detected.

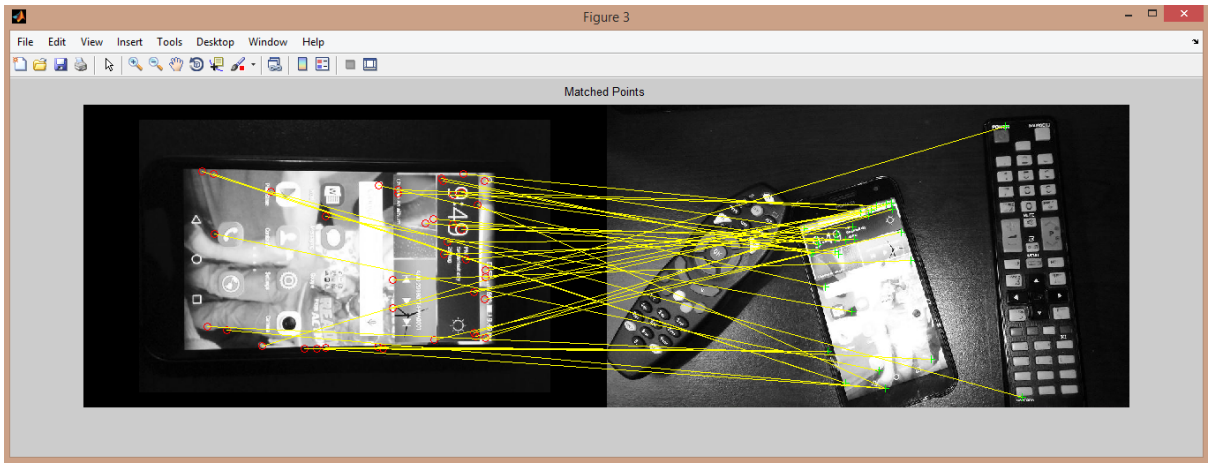


Figure 4.15: Matched points in the image with outliers

The matched features are shown and include some outliers. The image is cleaned and only inliers are shown Figure 4.16.

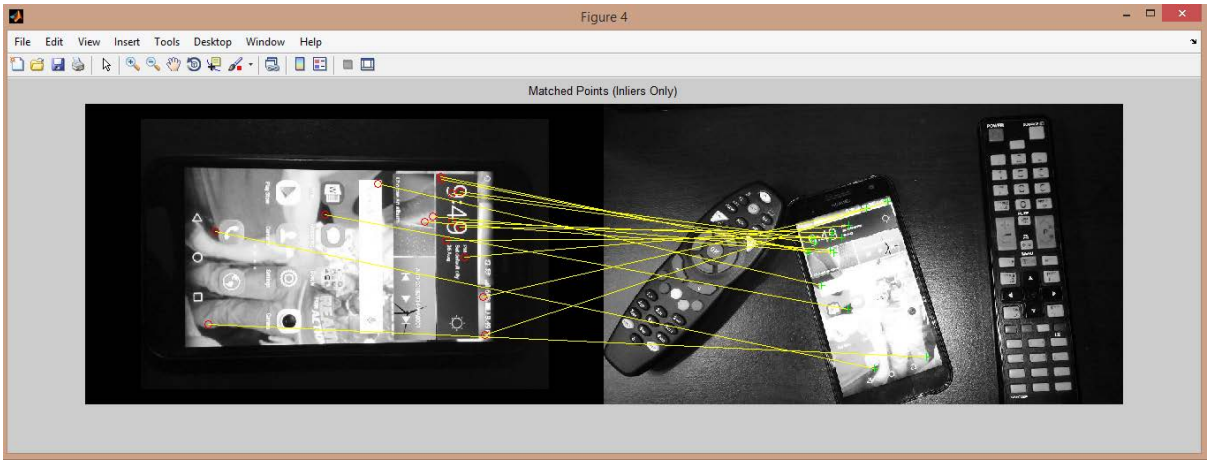


Figure 4.16: Matched points with inliers only

The cellular telephone is detected in the scene and a box is drawn around the detected object, Figure 4.17.

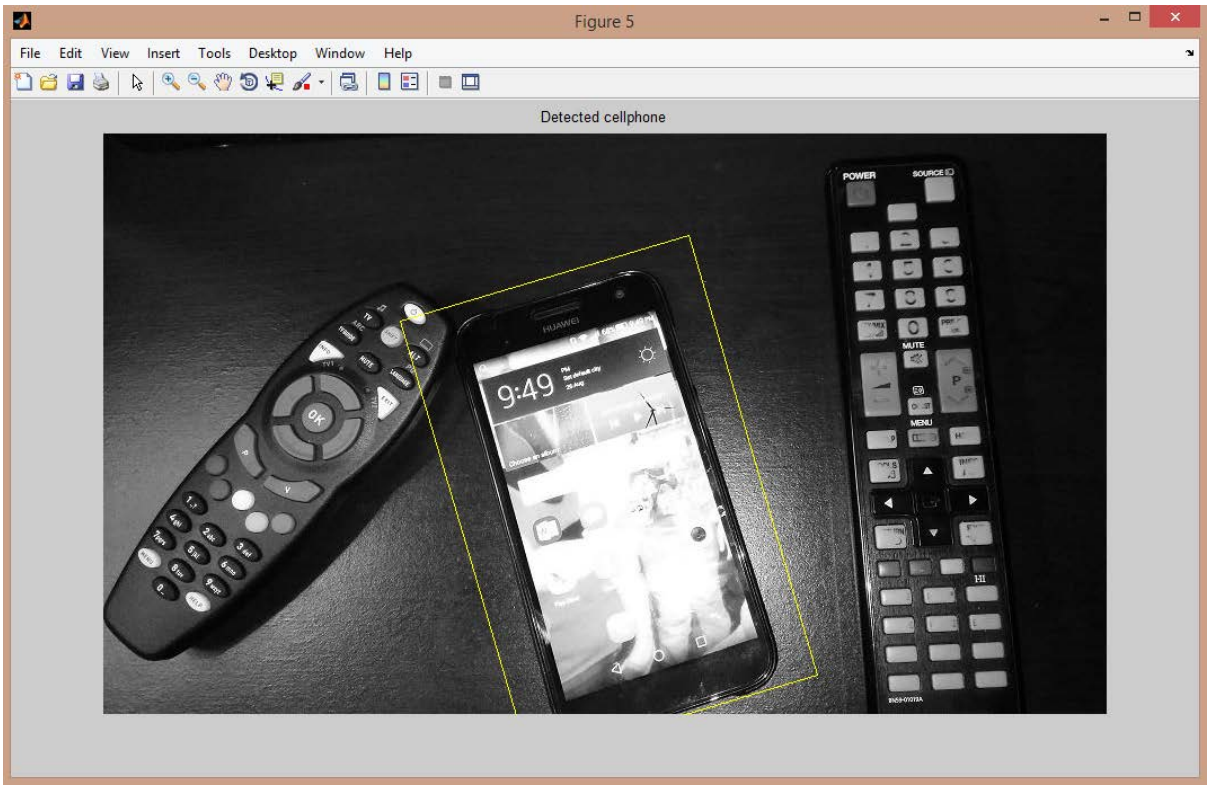


Figure 4.17: Cellular telephone detected

The following image in Figure 4.18 shows an example of how the data will be captured in an Experimental Evaluation Application User Interface. In this example the pitch, yaw and roll angles can be seen. The returned value of the lower jaw is also shown. If the lower jaw travels downwards, the returned value is increased as the mouth is opened. When the returned value

goes beyond a specified value or tolerance, the algorithm determines that the driver is yawning and the yawn frequency is increased. If the yawn frequency goes beyond a predetermined range, it is determined that the driver is probably fatigued thereby triggering the first fatigue warning. The image below shows the driver facing the direction of travel. The lower jaw vector is zero with the yawn counter and yawn frequency at zero as well. At this stage the driver is not fatigued.

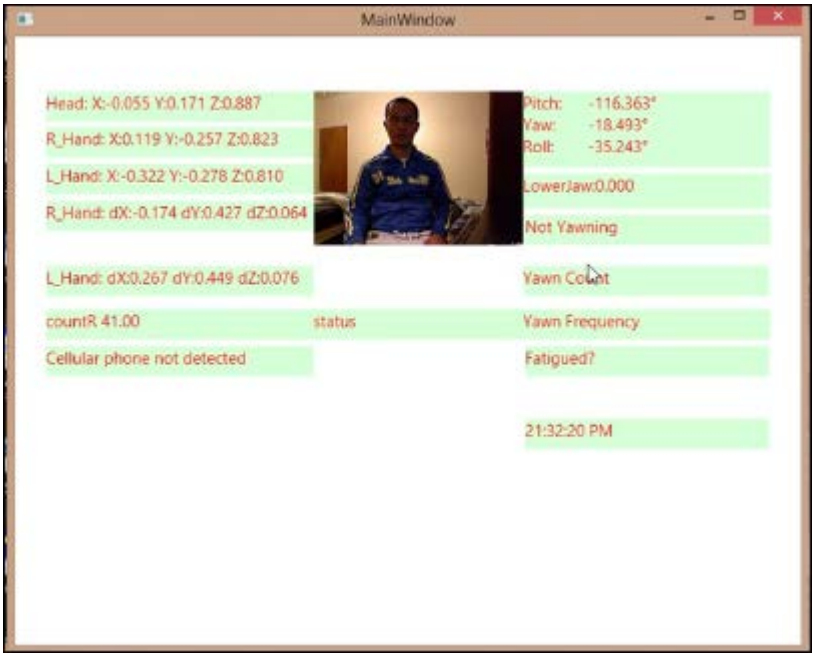


Figure 4.18: Experimental Evaluation Application User Interface

In this example, the following data is captured:

- Vector of the driver's head
- Vector of left hand
- Vector of the right hand
- The distance between the head vector and hand vectors are calculated and displayed
- The head's pitch, roll and yaw angles is calculated and displayed
- The lower jaw vector is calculated.
- Yawn frequency, e.g. yawns/minute and yawn count can be calculated

This data will be captured and used to determine whether a driver is fatigued and whether the driver is using a cellular telephone while driving.

4.4 Expected Performance and Operating Conditions

4.4.1 Expected Performance

i. The digital camera system should not obscure the drivers view.
ii. The unit should be maintenance free.
iii. The system must be easy to use.
iv. The unit must give an audible warning when driver fatigue is detected.
v. A secondary objective is for the system to give a visual warning.
vi. The system in its design must be non-intrusive.
vii. The system must be cost effective.
viii. The system must be easy to install.

4.4.2 Operating Conditions

i. The system must monitor driver attention with reasonable accuracy.
ii. The system must analyse if a driver uses a cellular telephone while driving and give a warning when cellular telephone usage is detected.
iii. The system must continuously check if the driver's head is pointed in the direction of vehicle travel.

CHAPTER 5
EXPERIMENTAL EVALUATION

5.1 Introduction

Image geometry is on a two dimensional plane with coordinates *(x,y)* that presents a point in the image. The world around us is three dimensional, so one dimension is lost. In Computer Vision an attempt is made to regain the lost dimension (Shah, 1997). In this chapter image vector translation, image rotation and a real world application is discussed.

5.2 Vector Translation

If we have the point *(X,Y,Z)* and the point is translated by *dx,dy* and *dz*, the new coordinates are given by:

$$X_1 = X + dx \tag{5.1}$$

$$Y_1 = Y + dy \tag{5.2}$$

$$Z_1 = Z + dz \tag{5.3}$$

and can be written as:

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.4}$$

with:

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = T \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.5}$$

And T is the translation matrix:

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

With the inverse of T being:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.7}$$

5.3 Rotation about an Arbitrary Axis

For vector *(XYZ)* in Figure 5.1, *R* is the length of the vector and theta is the angle of the vector around the *X*-axis. Vector *(X1,Y1,Z1)* is the new coordinates of the rotation around the *Z*-axis rotated by angle theta in a clockwise direction (Shah, 1997).

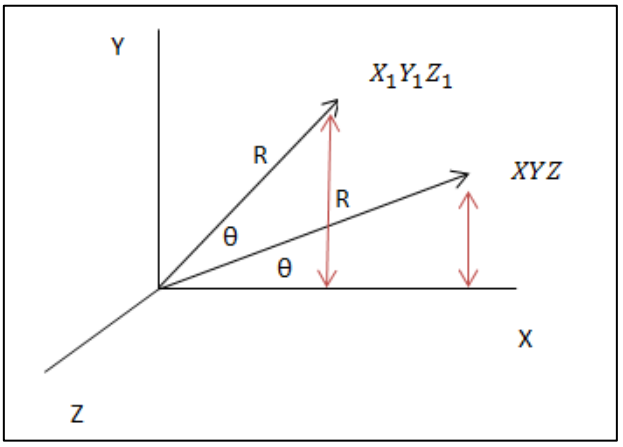


Figure 5.1: Vectors

By means of applying trigonometry the following can be derived:

$$X = R\cos\theta \tag{5.8}$$

$$Y = R\cos\theta \tag{5.9}$$

with:

$$X_1 = R\cos(\theta + \emptyset) = R\cos\theta\cos\emptyset - R\sin\theta\sin\emptyset \tag{5.10}$$

while:

$$Y_1 = R\sin(\theta + \emptyset) = R\sin\theta\cos\emptyset - R\cos\theta\sin\emptyset. \tag{5.11}$$

Therefore:

$$X_1 = X\cos\theta - Y\sin\theta \tag{5.12}$$

$$Y_1 = X\sin\theta - Y\cos\theta \tag{5.13}$$

And can be written as:

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.14}$$

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = R_\theta^z \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.15}$$

Where R_θ^z is the rotation matrix:

$$R_\theta^z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.16}$$

5.4 Application of Vector Calculus and Discussion

To generate a rotation in 3 dimensions, the axis of rotation and the amount of rotation have to be specified. With this the pitch angle β, roll angle γ and yaw angle α is determined. This can be used for tracking of the pivot angle of the driver's head about the X, Y and Z axis. The yaw angle is the pivot angle of the driver's head about the Y axis. The pitch angle is the pivot angle of the driver's head around the X axis and the roll angle is the head joint rotation around the Z axis (Figure 5.2) (LaValle, 2006).

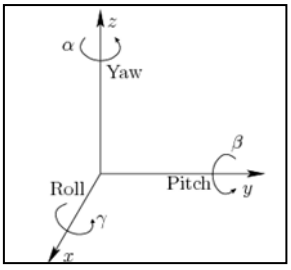


Figure 5.2: 3D rotation described as a sequence of yaw, pitch, and roll rotations

Yaw angle is a counter clockwise rotation of α about the Z-axis. The rotation matrix related to yaw angle α is given as equation (5.17) (LaValle, 2006):

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.17}$$

In Figure 5.3 Euler's rotation around Z is applied to determine the yaw angle α (Potter, 2015).

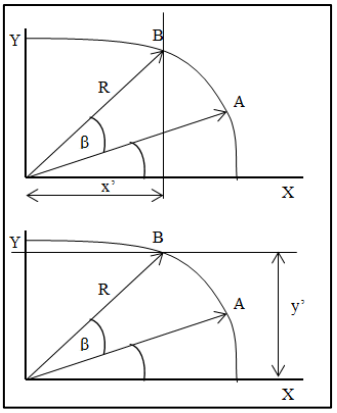


Figure 5.3: Euler's rotation around Z

Hence x' is given by:

$$x' = x\cos\beta - y\sin\beta \tag{5.18}$$

and y' is given by:

$$y' = x\sin\beta + y\cos\beta. \tag{5.19}$$

The pitch angle is the counter clockwise rotation of β about the Y-axis. The rotation matrix related to pitch angle θ is given as equation (5.20):

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ \sin\alpha & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{5.20}$$

Figure 5.4 shows Euler's rotation around Y to determine the pitch angle β (Potter, 2015).

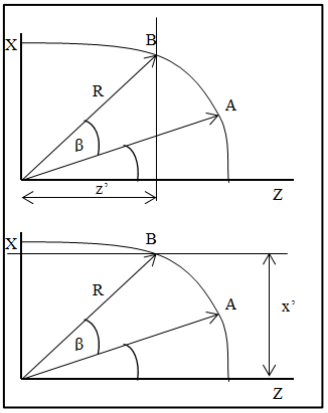


Figure 5.4: Euler's rotation around Y

Hence x' is given by:

$$x' = z\sin\beta + x\sin\beta \tag{5.21}$$

and z' is given:

$$z' = z\cos\beta - x\sin\beta. \tag{5.22}$$

The roll angle is the counter clockwise pivot of γ axis around the X-axis. The rotation matrix related to roll angle γ is given as equation (5.23):

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & 0 & \cos\gamma \end{pmatrix} \tag{5.23}$$

The following method, Figure 5.5, applies Euler's rotation around X to determine the roll angle γ(Potter, 2015).

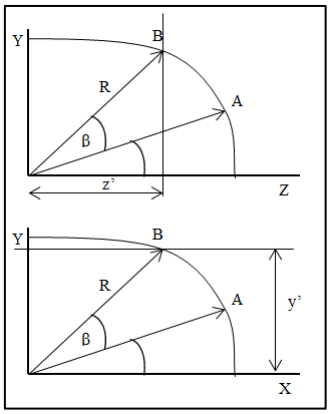


Figure 5.5: Euler's rotation around X

Hence y' is given by:

$$y' = y\cos\beta - z\sin\beta \tag{5.24}$$

with z' given by:

$$z' = y\sin\beta + z\cos\beta. \tag{5.25}$$

According to LaValle a singular rotary motion matrix can be accomplished by multiplying the yaw, pitch, and roll rotation matrices to ascertain equation (5.26) (LaValle, 2006):

$$R_{(\alpha,\beta,\gamma)} = R_z(\alpha)R_y(\beta)R_z(\gamma)$$

$$\begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix} \tag{5.26}$$

The yaw angle α, pitch angle β and roll angle γ can be determined directly from a given rotation matrix. If the next rotary motion matrix is given or known and represented by equation (5.27), the pitch, yaw and roll angles are calculated in equations (5.28), (5.29) and (5.30) respectively (LaValle, 2006).

$$x = \begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix} \tag{5.27}$$

$$\alpha = \tan^{-1}(r21 \div r11) \tag{5.28}$$

$$\beta = \tan^{-1}\left(-r31 \div \sqrt{r_{32}^2 + r_{33}^2}\right) \tag{5.29}$$

$$\gamma = \tan^{-1}(r32 \div r33) \tag{5.30}$$

Below, in Figure 5.6 and in equation (5.31) is our application of the vector calculus, using some of the principles above, for Vectors A and B with angle θ. Here we calculate the distance between two joints. These are the distances between the head joint and hand joint or left shoulder joint. This procedure is for detecting whether the driver is using a cellular telephone while driving. The angle between the Vectors can be determined in equation (5.31).

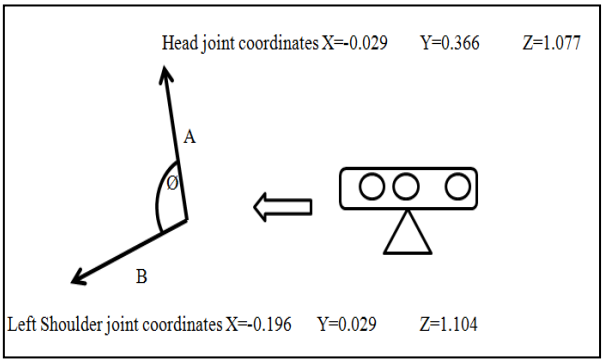


Figure 5.6: Illustration of projected vectors

The vector and angle is calculated using the projected variables in Figure 5.6 and Figure 5.7 as following:

$$A \cdot B = (A_x \times B_x) + (A_y \times B_y) + (A_Z + B_z) \tag{5.31}$$

Therefore:

$$|A| = \sqrt{(0.029^2 + 0.366^2 + 1.077^2)}$$

$$\therefore |A| = 1.1378$$

$$|B| = \sqrt{0.196^2+0.079^2+1.104^2}$$

$$\therefore |B| = 1.1235$$

$$1.2236 = \sqrt{(1.2097)} \times \sqrt{(1.1235)} \times \cos\theta$$

$$\therefore \cos\theta = 0.99$$

$$\text{and} \quad \theta = 8°$$

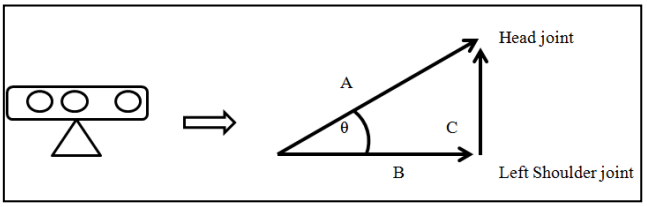


Figure 5.7: Determining distance between two joints

Hence we have:

$$C^2 = A^2+B^2 - 2AB \times \cos\theta \qquad (5.32)$$

$$= 1.1378^2+1.1235^2 - 2(1.1378 \times 1.1235) \times \cos 8$$

$$C = 0.15838$$

Figure 5.8 demonstrates how the joints are tracked. Here it can be seen that the tracked joints are used to calculate the vector distances between joints. For example in this demonstration the calculated distance between the left hand joint and the head joint on the *X* axis is 0.057m, on the *Y* axis the distance is 0.097m and on the *Z* axis the distance is 0.132m. The algorithm, in addition to cellular telephone object recognition, can be used to determine whether the driver has a cellular telephone in hand.

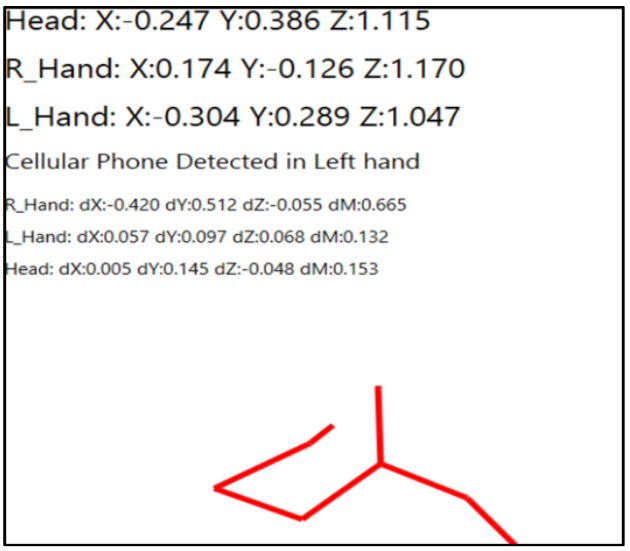


Figure 5.8: Skeleton tracking with coordinates

The distance between two joints is determined by:

$$Joint\_Distance = \sqrt{(X^2 + Y^2 + Z^2)} \tag{5.33}$$

Below in Figure 5.9 and Figure 5.10 the returned vector of the right and left hands are shown.

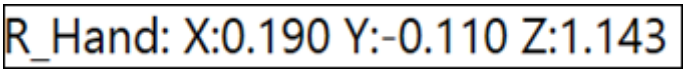


Figure 5.9: Right Hand Vector

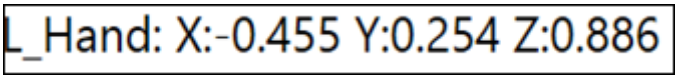


Figure 5.10: Left Hand Vector

Figure 5.11 is a message displayed in the Experimental Evaluation Application's User Interface after the control system sensed that a cellular telephone has been detected in the left hand.

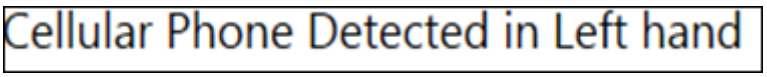


Figure 5.11: Text displaying when a cellular phone is in the left hand

Figure 5.12 shows the calculated distance of the vector between the head joint and the left shoulder joint using the above mentioned formula. The angle between the head joint and the shoulder joint can also be determined by using the calculated distances between the vectors. This method is used to determine the position and direction of the head joint.

Head_LShldr: dX:0.147 dY:0.255 dZ:-0.005 dM:0.294

Figure 5.12: Shoulder distance

To generate a rotation in 3D, the axis of rotation and the amount of rotation have to be specified. This is important for tracking of the rotation of the head about the *X*, *Y* and *Z* axis. With this the pitch angle, roll angle and yaw angle is determined. The yaw angle is the rotation around the *Y* axis. The pitch angle is the rotation of the head joint around the *X* axis and the roll angle is the head joint rotation around the *Z* axis as illustrated in Figure 5.13.

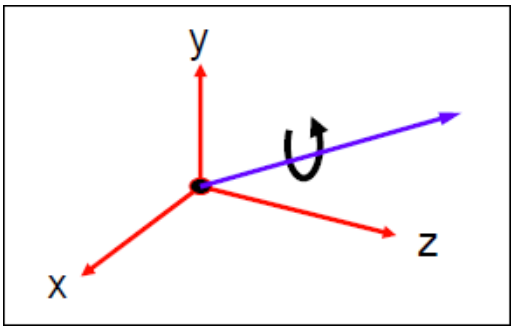


Figure 5.13: Rotation in 3D

A tolerance -30 degrees to +30 degrees is used for the pitch, roll and yaw angles. In Table 5.1 the tolerance of the variables are described. The returned pitch, yaw and roll angles are used to determine whether the driver is looking away from the direction of travel. If the driver looks away for a predefined period e.g. 5 seconds, the warning is triggered to remind the driver to attend to the road ahead. The returned angles are also used to determine whether the driver is nodding off due to fatigue. The driver's mouth position is also calculated. This variable determines whether the driver is yawning or not yawning. Figure 5.14 illustrates the head movement around each axis.

Table 5-10 Head pose angles

Angle	Acceptable tolerance value range in degrees
Pitch angle	-30 = looking down towards the floor +30 = looking up towards the ceiling
Roll angle	-30 = horizontal parallel with right shoulder of subject +30 = horizontal parallel with left shoulder of the subject
Yaw angle	-30 = turned towards the right shoulder of the subject +30 = turned towards the left shoulder of the subject

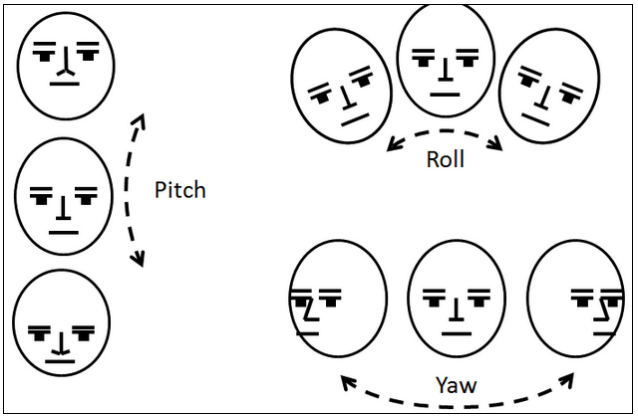


Figure 5.14: Head pose (Microsoft, 2013)

5.5 Experimental Evaluation and Analysis

5.5.1 Experimental Evaluation

The experimental evaluation was performed with the Kinect Sensor and Microsoft Visual Studio Development Environment for C#. Figure 5.15 shows the user application interface for the Fatigue Detection System's experimental evaluation. The user interface displays the head rotation, status of left hand, status of right hand, face direction, face direction after 2 seconds, cellular telephone status after 2 seconds, jawbone travel in mm, yawn status and yawn counter.

If the driver's face is not in the direction of travel for more than 2 seconds, the face direction status will change to "Driver is NOT looking in direction of travel". In the real-world application the direction of travel is determined via the steering angle/moment sensor and the compass of the satellite navigation system.

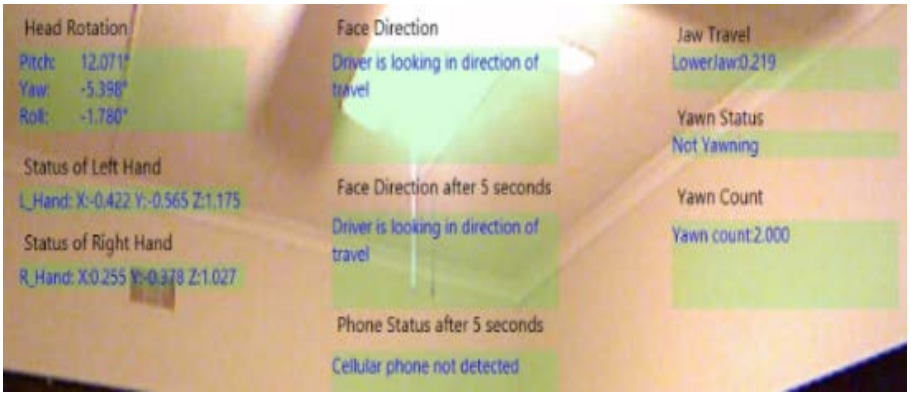


Figure 5.15: Evaluation User interface

In Figure 5.16 it can be seen that the driver is looking to one side. This is a sign of driver distraction and inattentiveness. This pose has a 2 second threshold. If the driver holds this position for 2 seconds or more, the warning will be triggered.

Figure 5.16: Driver looking to one side

After 2 seconds of inattentiveness, the warning is triggered as can be seen in Figure5.17. The face direction status has changed to show that the driver is not looking in the direction of travel.

Figure 5.17: Driver's face direction status after 2 seconds

In Figure 5.18 the driver is nodding his head up and down. This is one of the symptoms of a fatigued driver. If the driver keeps this pose for more than 2 seconds, the warning will be triggered.

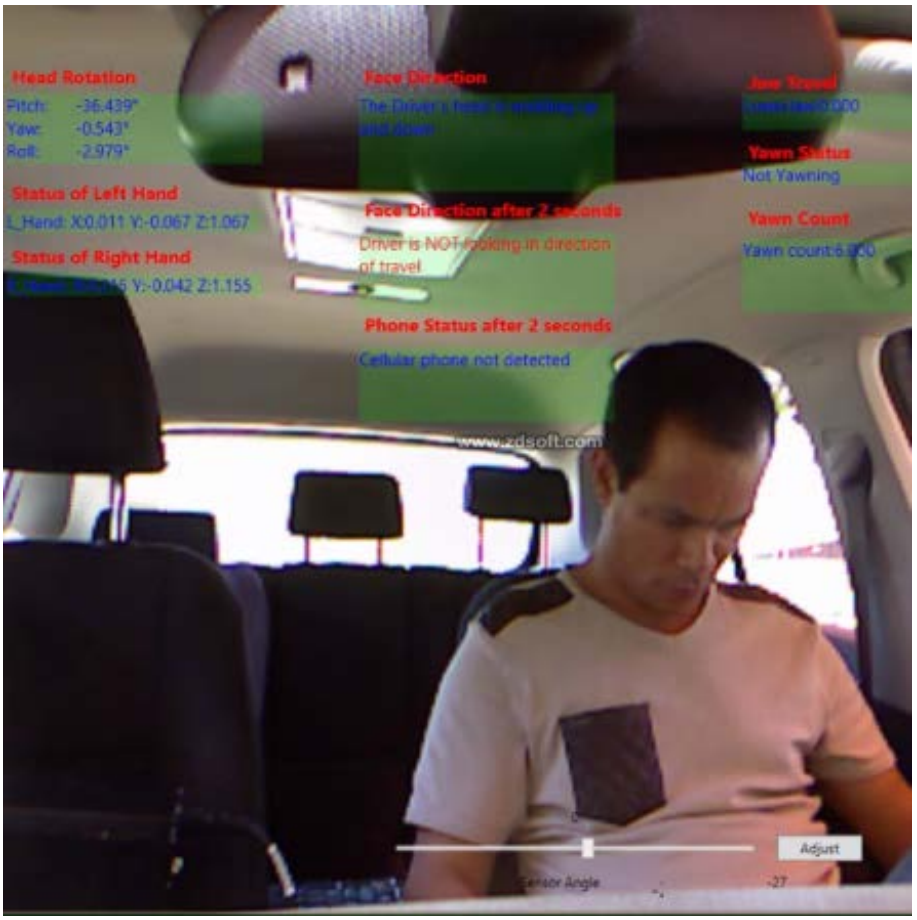


Figure 5.18: Driver nodding head up and down

In Figure 5.19 and Figure 5.20 the driver has a cellular telephone is his right hand (this is an inverted left/right image). The warning is triggered if the driver has the cellular telephone for more than 2 seconds. The warning can be a message displayed in the instrumentation panel that advises the driver to make use of the hands free telephone while driving. Figure 5.21 show that the system has detected yawning. The yawn counter is incremented.

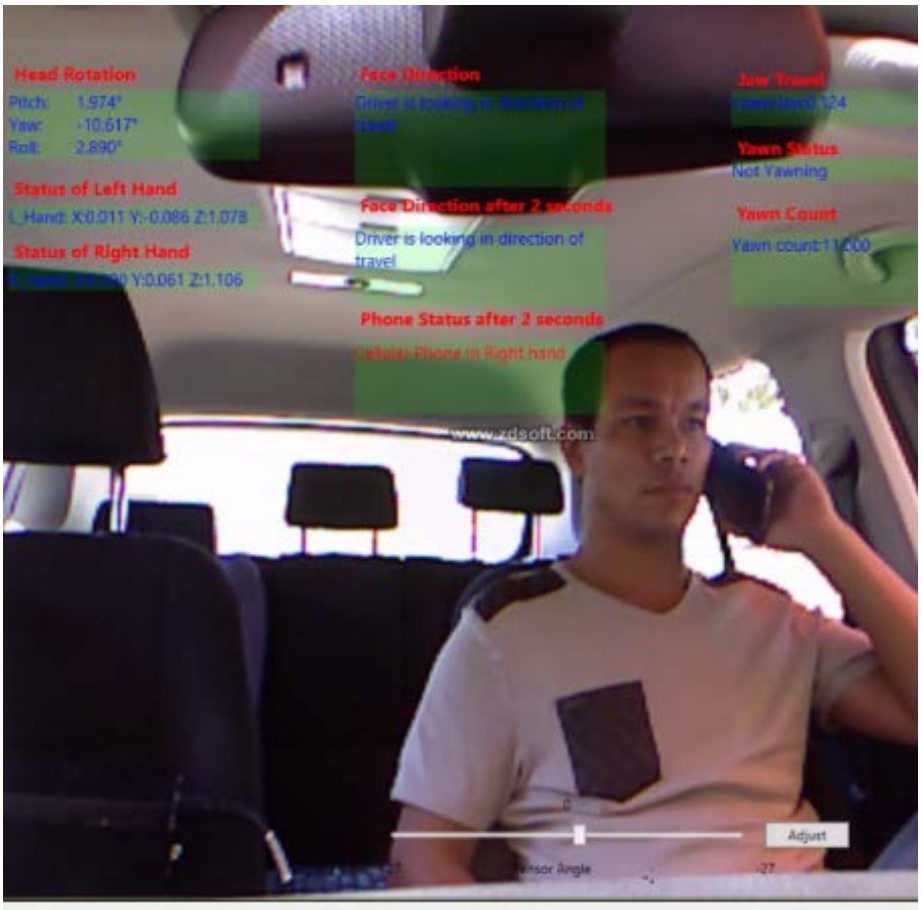


Figure 5.19: Driver has cellular telephone to his right ear

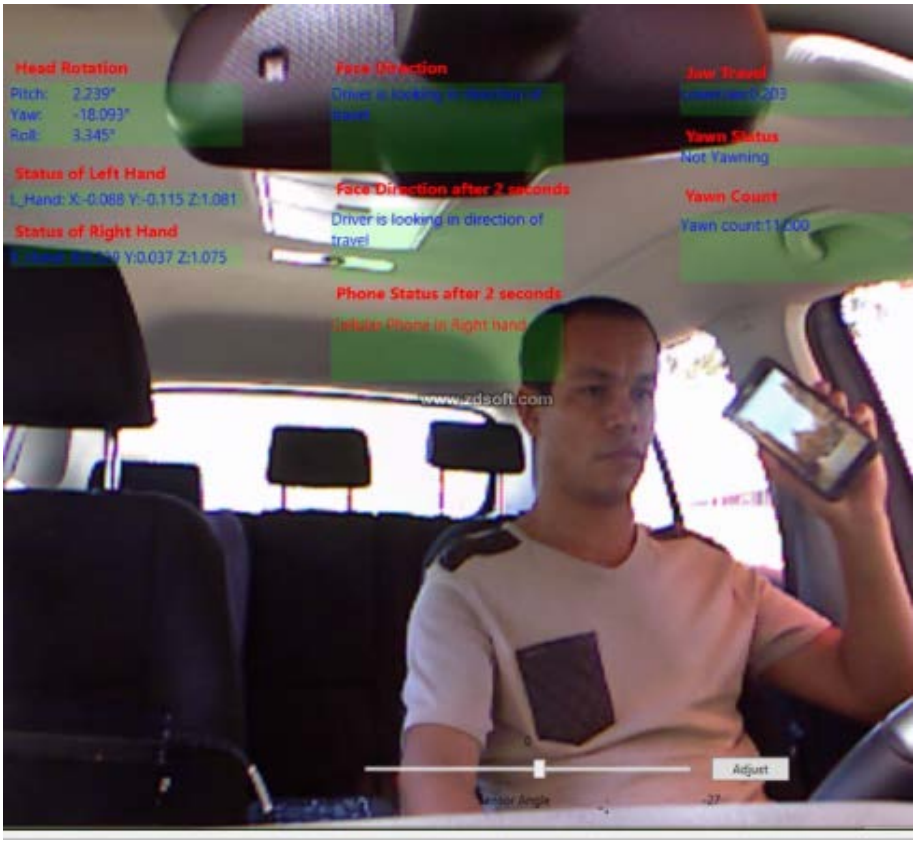


Figure 5.20: Driver has cellular telephone in his right hand

In Figure 5.21 the driver is yawning. The yawn status changed to Yawing and the yawing counter is incremented.

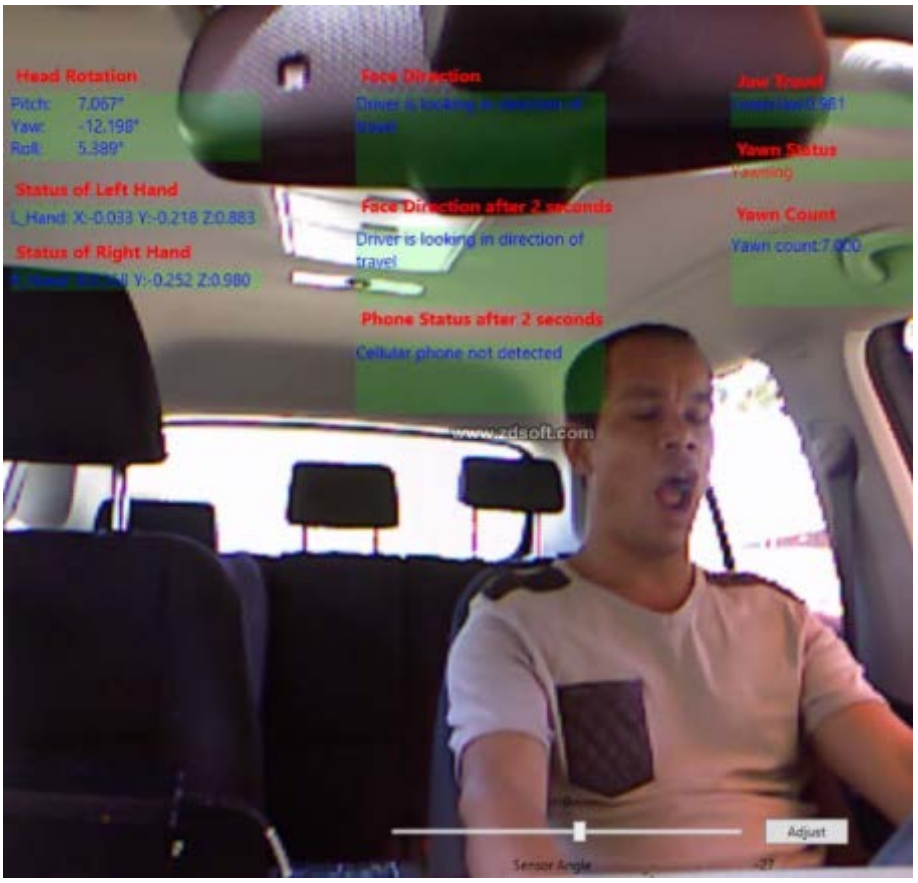
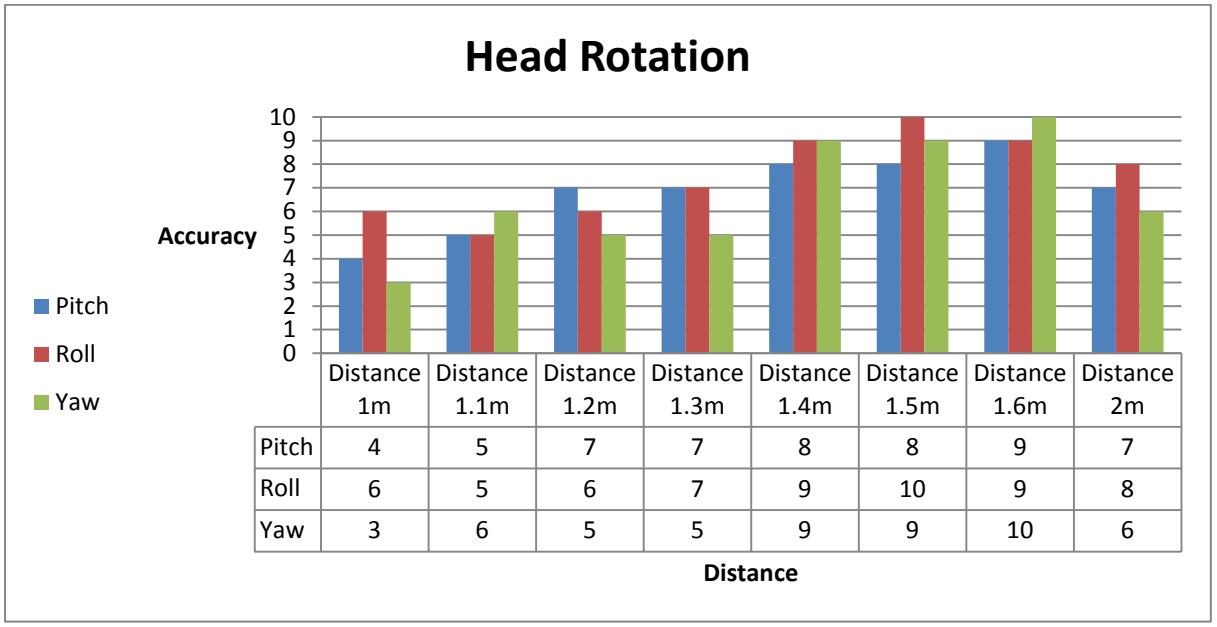


Figure 5.21: Driver is yawning

5.5.2 Analysis of results

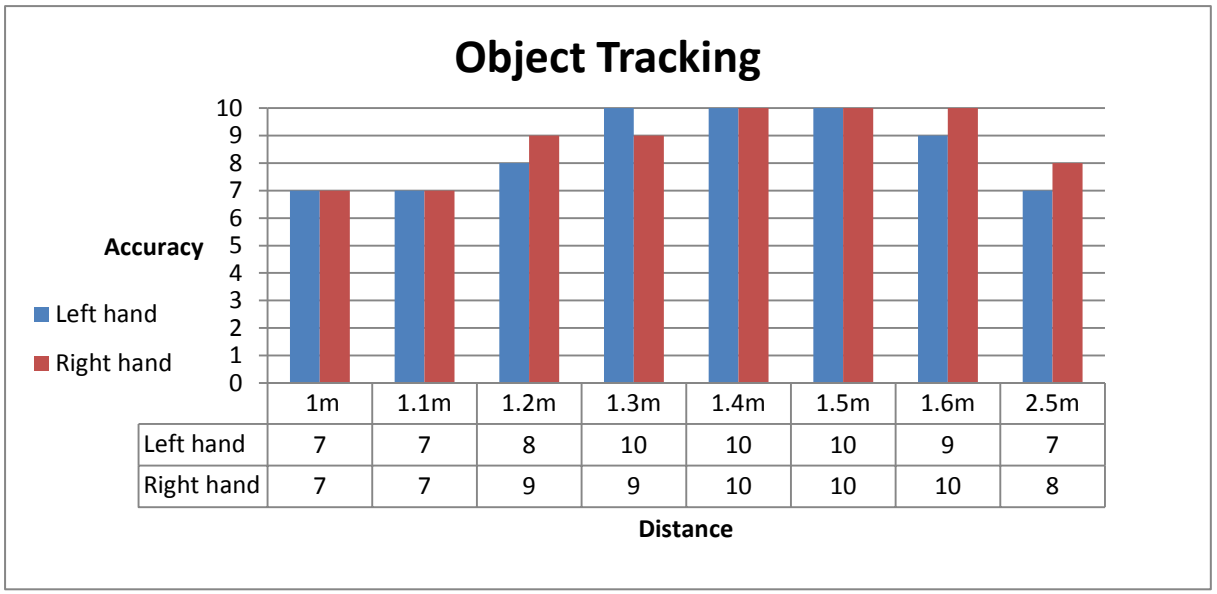
The Kinect for the Xbox 360 was used for the evaluation. The depth sensor range is between minimum 800mm to maximum 4000mm; however the Kinect for Windows SDK supports Near Mode whereas the Kinect for the Xbox 360 does not. With Near Mode in the Kinect for Windows SDK, the range of the Kinect is at minimum 500mm to a maximum of 3000mm. The chart in Figure 1 shows the relative accuracy of the head rotation angles measured over a distance ranging between 1m to 2m. Ten samples were taken for each rotation. At a distance of 1m the pitch angle was accurately measured only 4 times out of 10 test runs while the roll angle was accurate 6 times. The yaw angle was accurate only 3 times out of 10. The sensor accuracy at 1.6m is 90%. The desired depth as illustrated in Figure 5.22 is between 1.5m to 1.6 meters.



	Distance 1m	Distance 1.1m	Distance 1.2m	Distance 1.3m	Distance 1.4m	Distance 1.5m	Distance 1.6m	Distance 2m
Pitch	4	5	7	7	8	8	9	7
Roll	6	5	6	7	9	10	9	8
Yaw	3	6	5	5	9	9	10	6

Figure 5.22: Head rotation

Object tracking is reasonably accurate between 1.4m and 1.6m, Figure 5.23. At 1m and 1.1m depth the success rate is as low as 7 out of 10 successful object tracking or 70%. At 1.5m the success rate is 100%. The success rate drops however after 1.9m and is as low as 70% to 80% at 2.5m.



	1m	1.1m	1.2m	1.3m	1.4m	1.5m	1.6m	2.5m
Left hand	7	7	8	10	10	10	9	7
Right hand	7	7	9	9	10	10	10	8

Figure 5.23: Object tracking

Yawn detection was measured at a depth of 1.6m at various pitch, yaw and roll angles. The success rate is best between 0 degrees and 10 degrees but lowest at 30 degrees. The success rate at 30 degrees is as low as 60% at times. A successful yawn could only be measured 6 out of 10 times at angles greater than 30 degrees. Figure 5.24 illustrates the success rate of the Yawn detection system.

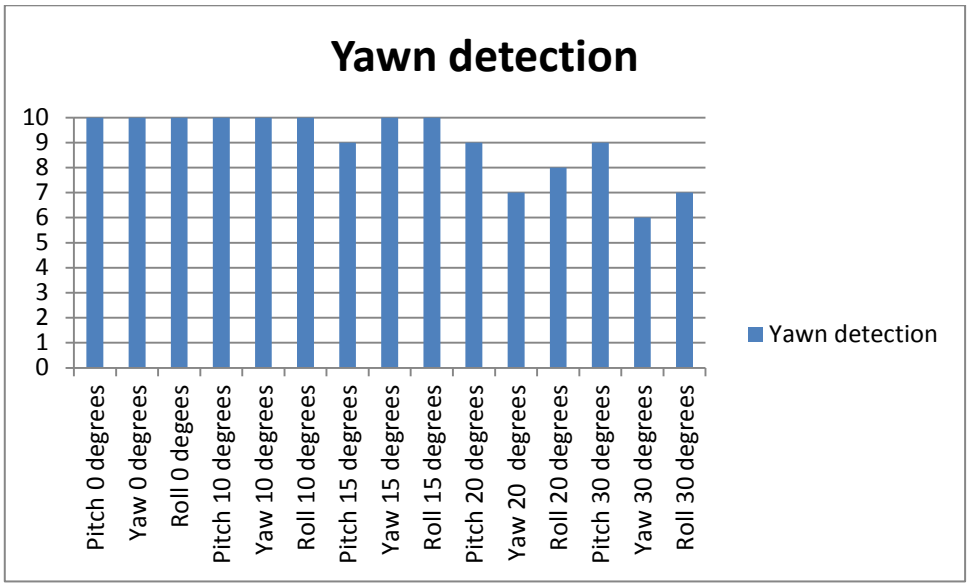


Figure 5.24: Yawn detection

5.5.3 Discussion

For distraction detection the head pose and or face direction was used to derive if the driver was looking on or away from the road ahead. For fatigue detection the drivers head pose angles were used. The detection and control system monitors pitch angle, roll angle and yaw angle. If any of the angles exceed a predefined value for a certain period of time, a warning is triggered, to remind the driver to be attentive. The angles were also used to determine if the driver was nodding off or whether he has fallen asleep. If a driver uses a cellular telephone while driving, the system uses the *X*, *Y* and *Z* coordinates of the driver's wrist joint and compares this coordinate and vector to the driver's head position together with object recognition. The warning is triggered if this pose is held for more than 5 seconds. Experiments have showed the efficiency of the proposed system. The system excludes involuntary touching of the face, hair and ears and will consider this type of disturbances in the future.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Discussion

A driver fatigue detection system which monitors observable behaviours plays an important role in monitoring driver actions and deciding whether the driver behaves in manner that is acceptable and safe before activating a visual or acoustic warning signal. Research has found that most accidents are caused by human error. In the United States alone, over half a million injuries and thousands of fatalities are caused by distracted driving every year. In most vehicle accidents passive safety devices are not enough to ensure passenger safety. The best protection is avoidance. Over the last decade motor vehicle manufacturers have invested in active safety devices such as advanced lane departure warning systems, automatic braking systems and fatigue detection systems. However, active safety devices in modern vehicles do not include cellular telephone detection when research has shown conclusively the dangers of cellular telephone usage while driving. Although many different types of fatigue technologies exist in most high end vehicles, none researched by the student provided a complete solution to driver fatigue and driver distraction detection.

6.2 Future work

The tracking of both eyes are possible with the Kinect V2. The eyes can be tracked and their 2D coordinates can be displayed for evaluation. Eyelid movement and the opening and closing of the eyes can be tracked. First the camera creates an image; the software detects the head, crops the edges of the face, locates the region around the eyes and monitors fatigue by calculating the eye lid opening and closing frequency. However, it was decided to exclude eye tracking due to the time required for measurement and refinement.

The Kinect includes voice and speech recognition capabilities. It is possible to develop a system that locates stress and fatigue in a voice by measuring the acoustic properties of the voice being analysed. If fatigue has been detected through a camera based fatigue detection system, the driver is prompted to repeat certain words or phrases randomly. The voice

analysis provides further fatigue detection while at the same time helps the driver with staying active and alert, especially when driving long distances.

7. LIST OF PUBLICATIONS

Cleshain Solomon and Zenghui Wang, “Driver Attention and Behavior Detection with Kinect,” Journal of Image and Graphics, Vol. 3, No. 2, pp.84-89, December 2015.

Cleshain Solomon, Zenghui Wang, Microsoft Kinect sensor based driver fatigue and distraction detection system, submitted to Journal of Sensors.

8. BIBLIOGRAPHY

Aloimonos, J., 1990. *Purposive and qualitative active vision.* Atlantic City, NJ, s.n., pp. 346-360.

Barr L., H. H. P. S., 2011. *A review and evaluation of emerging driver fatigue,* Washington, DC: U.S. Department of Transportation.

Canny, J., 1986. A Computational Approach to Edge Detection. *The IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI),* 8(6), pp. 679 - 698.

Care Drive, 2013. *Driver Fatigue is an important cause of road crashes.* [Online]
Available at: http://www.care-drive.com/theme_detail.php?id=6
[Accessed 04 05 2013].

Centers for Desease Control and Prevention, 2013. *Distracted Driving in the United States and Europe.* [Online]
Available at: http://www.cdc.gov/features/dsdistracteddriving/
[Accessed 04 05 2013].

Chávez-Aragón, A., Macknojia, R., Payeur, P. & Laganière, R., 2013. Rapid 3D Modeling and Parts Recognition on Automotive Vehicles Using a Network of RGB-D Sensors for Robot Guidance. *Journal of Sensors,* Volume 2013, p. 16.

Continental Corporation, 2013. *Continental Counts on LEDs as Co-pilot.* [Online]
Available at: http://www.continental-corporation.com/www/pressportal_com_en/themes/press_releases/3_automotive_group/interior/press_releases/pr_2013_02_07_driver_focus_en.html
[Accessed 01 07 2015].

Daimler, 2008. *TecDay Real Life Safety.* [Online]
Available at: http://media.daimler.com/
[Accessed 06 05 2013].

Davies, E. R., 2012. *Computer and Machine Vision: Theory,Algorithms,Practicalities.* 4th ed. Oxford: Elsevier Inc..

Demaagd, K., Oliver, A., Oostendorp, N. & Scott, K., 2012. *Practical Computer Vision with SimpleCV.* 1st ed. California: O'Reilly Media, Inc..

Denso Corporation, 2012. *Don't Get Hurt, Stay Alert with DENSO's Driver Status Monitor.* [Online]
Available at: http://www.densodynamics.com/dont-get-hurt-stay-alert-with-densos-driver-status-monitor/
[Accessed 01 07 2015].

Duchowski, A., 2007. *Eye Tracking Methodology: Theory and Practice.* 2nd ed. London: Springer-Verlag London Limited.

Duda, R., Hart, P. & Stork, D., 2012. *Pattern Classification.* 2nd ed. s.l.:John Wiley & Sons.

Euroncap, 2013. *Ford Driver Alert.* [Online]
Available at: http://www.euroncap.com/
[Accessed 08 05 2013].

Fischler, M. A. & Bolles, R. C., 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM,* 24(6), pp. 381-395.

Gonzalez, R., Woods, R. & Eddins, S., 2010. *Digital Image Processing Using MATLAB.* 2nd ed. s.l.:Tata McGraw Hill Education.

Goodyear , 2013. *Goodyear News.* [Online]
Available at: http://www.goodyear.eu/za_en/news/139257-goodyear_survey_anxiety/index.htm?to=2013&from=2003
[Accessed 13 04 2013].

Green, B., 2002. *Canny Edge Detection Tutorial.* [Online]
Available at: http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html
[Accessed 01 07 2015].

Green, B., 2002. *Edge Detection Tutorial.* [Online]
Available at: http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/edge.html
[Accessed 01 07 2015].

Intel Corporation, 1999. *intel open source computer vision library: Reference manual.* 1 ed. s.l.:Intel Corporation.

Jähne, B. & Haussecker, H., 2000. *Computer Vision and Applications: A Guide for Students and Practitioners.* illustrated ed. San Diego: Academic Press.

Jain, R., Kasturi, R. & Schunck, B., 1995. *Machine Vision.* illustrated ed. s.l.:McGraw-Hill.

Jana, A., 2012. *Kinect for Windows SDK Programming Guide.* 1st ed. Mumbai: Packt Publishing.

Jan, T. V., Karnahl, T., Hilgenstock, J. & Zobel, R., 2005. *Don't sleep and drive-VW's fatigue detection technology.* Washington, DC, s.n.

Jia, S. M., Wang, L. J., Li, X. Z. & Wen, L. F., 2015. Person Tracking System by Fusing Multicues Based on Patches. *Journal of Sensors,* Volume 2015, p. 12.

Kaur, T. & Singh, S. K., 2014. Enhance the Driver Drowsiness Detection System Using EDAMAS and Region Prop Techniques. *International Journal of Engineering Research and Applications,* 4(3), pp. 707-711.

Larsson, P. & Trent, V., 2008. *Method and arrangement for interpreting a subjects head and eye activity.* USA, Patent No. US7460940.

LaValle, S., 2006. *Planning Algorithms.* s.l.:Cambridge University Press.

Leisure Auto Security Equipment , 2009. *Keep away from fatigue driving.* [Online]
Available at: http://www.leisureauto.com/bbx/884888-884888.html?id=15025&newsid=439388
[Accessed 01 07 2015].

Lowe, D. G., 1999. *Object Recognition from Local Scale-Invariant Features.* Kerkyra, IEEE.

MATLAB, 2015. *MSERRegions class.* [Online]
Available at: http://www.mathworks.com/help/vision/ref/mserregions-class.html
[Accessed 21 07 2015].

Mazda Motor Corporation, 2015. *Safety Technology.* [Online]
Available at: http://www2.mazda.com/en/technology/safety/active_safety/ldws.html
[Accessed 01 07 2015].

Microsoft Developer Network, 2013. *Kinect for Windows Sensor.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/hh855355.aspx
[Accessed 03 05 2013].

Microsoft, 2013. *Kinect for Windows Architecture.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/jj131023.aspx
[Accessed 04 05 2013].

Microsoft, 2013. *Kinect for Windows Programming Guide.* [Online]
Available at: http://msdn.microsoft.com/en-us/library/hh855348.aspx
[Accessed 04 05 2013].

Microsoft, 2015. *Kinect for Windows SDK.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/hh855347.aspx
[Accessed 04 07 2015].

Nixon, M. & Aguado, A., 2012. *Feature Extraction & Image Processing for Computer Vision.* 1st ed. s.l.:Elsevier Ltd..

Nixon, M. & Aguado, A., 2012. *Feature Extraction & Image Processing for Computer Vision.* 3rd ed. London: Elsevier Ltd..

Pickett, J. P., 2000. *The American Heritage Dictionary of the English Language.* 4th ed. s.l.:Houghton Mifflin Harcourt Publishing Company.

Potter, R., 2015. *A Vector Type for C#.* [Online]
Available at: http://www.codeproject.com/Articles/17425/A-Vector-Type-for-C#_rating
[Accessed 10 02 2015].

Roberts, L., 1980. *Machine Perception of Three-Dimensional Solids.* s.l.:Garland Pub..

ROSPA, 2001. *The Royal Society for the Prevention of Accidents.* [Online]
Available at: http://www.rospa.com/
[Accessed 06 05 2013].

Seeing Machines, 2015. *Advanced Driver Fatigue & Distraction Detection.* [Online]
Available at: http://www.seeingmachines.com/
[Accessed 01 07 2015].

Shah, M., 1997. *Fundamentals of computer vision.* 1st ed. Florida: University of Central Florida.

Shapiro, L. & Stockman, G., 2001. *Computer Vision.* illustrated ed. California: Prentice Hall.

Smith, P. S., Shah, M. & da Vitoria Lobo, N., 2000. *Algorithm for monitoring head/eye motion for driver alertness with one camera.* s.l., s.n.

Sousanis, J., 2011. *World Vehicle Population Tops 1 Billion Units.* [Online]
Available at: http://wardsauto.com/ar/world_vehicle_population_110815
[Accessed 10 05 2013].

Szeliski, R., 2010. *Computer Vision: Algorithms and Applications.* 1st ed. Lake Wenatchee: Springer.

Taylor, S. A., 1998. CCD and CMOS imaging array technologies: technology review. *UK: Xerox Research Centre Europe.*

Trucco, E. & Verri, A., 1998. *Introductory Techniques for 3-D Computer Vision.* illustrated ed. New Jersey: Prentice Hall.

Tuvshinjargal, D., Byambaa, D. & Lee, D. J., 2015. Hybrid Motion Planning Method for Autonomous Robots Using Kinect Based Sensor Fusion and Virtual Plane Approach in Dynamic Environments. *Journal of Sensors,* Volume 2015, p. 13.

Vernon, D., 1991. *Machine Vision.* illustrated ed. Hertfordshire: Pearson Education, Limited.

Volkswagen AG, 2013. *Innovation and Technology.* [Online]
Available at: http://en.volkswagen.com/en/innovation-and-technology/innovative-technologies.html
[Accessed 05 05 2013].

Volvo Cars, 2007. *Volvo Cars introduces new systems for alerting tired and distracted drivers.* [Online]
Available at: https://www.media.volvocars.com/global/enhanced/en-gb/Media/Preview.aspx?mediaid=12130
[Accessed 06 05 2013].

9. APPENDICES

9.1 Appendix 1

MainWindow.XAML.cs

```
/// Fatigue Detection System////
/// Created By CT Solomon//////
/// for UNISA MTech>//////////
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Diagnostics;
using System.Timers;
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit.FaceTracking;
using System.Media;
using System.Windows.Controls;


namespace Project
{



    public partial class MainWindow : Window
    {

        int count_R = 0;
        int count_L = 0;
        int countz = 0;
        bool Y; bool Z; bool X; bool R; bool L;

        FaceTracker faceTracker;

        static System.Windows.Forms.Timer timer1 = new System.Windows.Forms.Timer();
        Stopwatch stopWatch = Stopwatch.StartNew();

        private byte[] colorPixelData;
        private short[] depthPixelData;
        private Skeleton[] skeletonData;

        private EnumIndexableCollection<FeaturePoint, PointF> facePoints;

        public MainWindow()
        {
            InitializeComponent();

        }

        KinectSensor myKinect;
        //
```

```
        SoundPlayer dingPlayer;


        private void Window_Loaded(object sender, RoutedEventArgs e)
        {

            // Check to see if a Kinect is available
            if (KinectSensor.KinectSensors.Count == 0)
            {
                MessageBox.Show("Zero Kinects attached", "Face viewer");
                Application.Current.Shutdown();
                return;
            }

            // Get the first Kinect on the computer
            myKinect = KinectSensor.KinectSensors[0];

            // Start the Kinect running and select the skeleton stream and color
stream
            try
            {
                myKinect.SkeletonStream.Enable();
                myKinect.ColorStream.Enable();

                myKinect.DepthStream.Enable(DepthImageFormat.Resolution80x60Fps30);
                myKinect.SkeletonStream.EnableTrackingInNearRange = false;

                myKinect.SkeletonStream.TrackingMode = SkeletonTrackingMode.Seated;
                myKinect.SkeletonStream.Enable(new TransformSmoothParameters
                {
                    Smoothing = 0.8f,
                    Correction = 0.05f,
                    Prediction = 0.5f,
                    JitterRadius = 0.05f,
                    MaxDeviationRadius = 0.04f
                });

                // Initialize data arrays
                colorPixelData = new byte[myKinect.ColorStream.FramePixelDataLength];
                depthPixelData = new short[myKinect.DepthStream.FramePixelDataLength];
                skeletonData = new Skeleton[6];

                //Time.Text = string.Format("{0:HH:mm:ss tt}", DateTime.Now);
                myKinect.Start();


            }
            //error handling
            catch
            {
                MessageBox.Show("Kinect not started", "Face Viewer");
                Application.Current.Shutdown();
            }
            // Initialize a new FaceTracker with the KinectSensor
            faceTracker = new FaceTracker(myKinect);
            // connect a handler to the event that fires when new frames are available
            myKinect.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(myKinect_SkeletonFrameReady);
            //colour stream
            myKinect.ColorFrameReady += new
EventHandler<ColorImageFrameReadyEventArgs>(myKinect_ColorFrameReady);
```

```
            // Listen to the AllFramesReady event to receive KinectSensor's data.
            myKinect.AllFramesReady += new
EventHandler<AllFramesReadyEventArgs>(myKinect_CCSkeletonFrameReady);
            // Load the sound
            dingPlayer = new SoundPlayer();

            // Use Seated Mode
            myKinect.SkeletonStream.TrackingMode = SkeletonTrackingMode.Seated;
            sliderAngle.Value = myKinect.ElevationAngle;
        }

        // set kinect sensor angle
        private void sliderAngle_ValueChanged_1(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            lblSliderValue.Content = (int)sliderAngle.Value;
        }
        private void btnSetAngle_Click_1(object sender, RoutedEventArgs e)
        {
            myKinect.ElevationAngle = (int)sliderAngle.Value;
        }

        //Kinect body Tracker
        void myKinect_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
        {
            string message1 = "Not started-Waiting";

            string message2 = "Not started-Waiting";

            string message3 = "Not started-Waiting";

            Skeleton[] skeletons = null;



            using (SkeletonFrame frame = e.OpenSkeletonFrame())
            {
                if (frame != null)
                {
                    skeletons = new Skeleton[frame.SkeletonArrayLength];
                    frame.CopySkeletonDataTo(skeletons);
                }
            }

            if (skeletons == null) return;

            foreach (Skeleton skeleton in skeletons)
            {

                if (skeleton.TrackingState == SkeletonTrackingState.Tracked)
                {
                    Joint headJoint = skeleton.Joints[JointType.Head];
                    SkeletonPoint headPosition = headJoint.Position;

                    Joint handJoint_R = skeleton.Joints[JointType.HandRight];
                    SkeletonPoint handPosition_R = handJoint_R.Position;

                    Joint handJoint_L = skeleton.Joints[JointType.HandLeft];
                    SkeletonPoint handPosition_L = handJoint_L.Position;

                    message1 = string.Format("Head: X:{0:0.000} Y:{1:0.000}
Z:{2:0.000}",
```

```
                    headPosition.X,
                    headPosition.Y,
                    headPosition.Z);

                    message2 = string.Format("R_Hand: X:{0:0.000} Y:{1:0.000}
Z:{2:0.000}",
                    handPosition_R.X,
                    handPosition_R.Y,
                    handPosition_R.Z);

                    message3 = string.Format("L_Hand: X:{0:0.000} Y:{1:0.000}
Z:{2:0.000}",
                    handPosition_L.X,
                    handPosition_L.Y,
                    handPosition_L.Z);

                    // cellphone in right hand
                    float distance_R =
jointDistance_R(skeleton.Joints[JointType.Head],
skeleton.Joints[JointType.HandRight]);

                    if (distance_R < 0.25f)
                    {
                        R = true;
                        int counted_R = count_R++;
                        counted_R = counted_R / 10;
                        //test.Text = string.Format("TimerR {0:0.00}", counted_R);
                        if (counted_R >= 5 && oldDistance_R > 0.2f && distance_R <
0.25f)
                        {
                            //dingPlayer.Play();
                            Phone_Detected_TextBlock.Text = "Cellular Phone in Right
hand";
                            Phone_Detected_TextBlock.Foreground = Brushes.Red;
                            timer1.Tick += blinkTextbox;
                            timer1.Interval = 250;
                            timer1.Enabled = true;
                            count_R = 0;
                        }
                    }

                    if (distance_R > 0.4f)
                    {
                        R = false;
                        if (L == false)
                        {
                            count_R = 0;
                            Phone_Detected_TextBlock.Text = "Cellular phone not
detected";
                            Phone_Detected_TextBlock.Foreground = Brushes.Blue;
                            timer1.Stop();
                            textBox1.Background = new
System.Windows.Media.SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 10,
40));

                        }
                    }

                    distance_R = oldDistance_R;

                    // cellphone in Left hand
```

```
                    float distance_L =
jointDistance_L(skeleton.Joints[JointType.Head], skeleton.Joints[JointType.HandLeft]);

                    if (distance_L < 0.25f)
                    {
                        L = true;
                        int counted_L = count_L++;
                        counted_L = counted_L / 10;
                        //test.Text = string.Format("TimerL {0:0.00}", counted_L);
                        if (counted_L >= 5 && oldDistance_L > 0.2f && distance_L <
0.25f)
                        {
                            dingPlayer.Play();
                            Phone_Detected_TextBlock.Text = "Cellular Phone in Left
hand";
                            Phone_Detected_TextBlock.Foreground = Brushes.Red;
                            timer1.Tick += blinkTextbox;
                            timer1.Interval = 250;
                            timer1.Enabled = true;
                            count_L = 0;

                        }
                    }

                    if (distance_L > 0.4f)
                    {
                        L = false;
                        if (R == false)
                        {
                            count_L = 0;

                            timer1.Stop();
                            textBox1.Background = new
System.Windows.Media.SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 10,
40));

                            Phone_Detected_TextBlock.Text = "Cellular phone not
detected";
                        }
                    }
                    distance_L=oldDistance_L;
                }

            }
            HeadStatusTextBlock.FontSize = 15;
            HeadStatusTextBlock.Text = message1;

            R_HandStatusTextBlock.FontSize = 15;
            R_HandStatusTextBlock.Text = message2;

            L_HandStatusTextBlock.FontSize = 15;
            L_HandStatusTextBlock.Text = message3;
        }

        //cell in right hand flasher
        private void blinkTextbox(object sender, EventArgs e)
        {

            if (textBox1.Background == Brushes.Red) textBox1.Background =
Brushes.White;
            else textBox1.Background = Brushes.Red;
```

```
        }

        int Count = 0;
        //AllframesReady event
        void myKinect_CCSkeletonFrameReady(object sender, AllFramesReadyEventArgs e)
        {
            // Retrieve each single frame and copy the data
            using (ColorImageFrame colorImageFrame = e.OpenColorImageFrame())
            {
                if (colorImageFrame == null)
                    return;
                colorImageFrame.CopyPixelDataTo(colorPixelData);
            }

            using (DepthImageFrame depthImageFrame = e.OpenDepthImageFrame())
            {
                if (depthImageFrame == null)
                    return;
                depthImageFrame.CopyPixelDataTo(depthPixelData);
            }

            using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
            {
                if (skeletonFrame == null)
                    return;
                skeletonFrame.CopySkeletonDataTo(skeletonData);
            }

            // Retrieve the first tracked skeleton if any. Otherwise, do nothing.
            var skeleton = skeletonData.FirstOrDefault(s => s.TrackingState ==
SkeletonTrackingState.Tracked);
            if (skeleton == null)
                return;


            // Make the faceTracker processing the data.
            FaceTrackFrame faceFrame = faceTracker.Track(myKinect.ColorStream.Format,
colorPixelData, myKinect.DepthStream.Format, depthPixelData, skeleton);

            // If a face is tracked, then we can use it.
            if (faceFrame.TrackSuccessful)
            {
                // Retrieve only the Animation Units coeffs.
                var AUCoeff = faceFrame.GetAnimationUnitCoefficients();

                var jawLowerer = AUCoeff[AnimationUnit.JawLower];
                var lipstretch = AUCoeff[AnimationUnit.LipStretcher];
                var lefteye_top = FeaturePoint.MiddleTopLeftEyelid;//added 30.04.2015
                var lefteye_bottom = FeaturePoint.MiddleBottomLeftEyelid;//added
30.04.2015
                //Debug3DShape("OuterCornerOfRightEye",
faceTrackFrame.Get3DShape()[FeaturePoint.OuterCornerOfRightEye]);
                var shape = faceFrame.Get3DShape();
                var X_22 = shape[FeaturePoint.MiddleTopLeftEyelid].X;//added
30.04.2015
                var Y_22 = shape[FeaturePoint.MiddleTopLeftEyelid].Y;//added
30.04.2015
                var Z_22 = shape[FeaturePoint.MiddleTopLeftEyelid].Z;//added
30.04.2015
```

```
                var X_23 = shape[FeaturePoint.MiddleBottomLeftEyelid].X;//added
30.04.2015
                var Y_23 = shape[FeaturePoint.MiddleBottomLeftEyelid].Y;//added
30.04.2015
                var Z_23 = shape[FeaturePoint.MiddleBottomLeftEyelid].Z;//added
30.04.2015

                textblockeye.Text = string.Format("toplfteye: X:{0:0.000} Y:{1:0.0000}
Z:{2:0.000}", X_22, Y_22, Z_22);//added 30.04.2015
                textblockeye2.Text = string.Format("btmlfteye: X:{0:0.000}
Y:{1:0.0000} Z:{2:0.000}", X_23, Y_23, Z_23);//added 30.04.2015

                var X_72 = shape[FeaturePoint.InnerTopLeftPupil].X;//added 30.04.2015
                var Y_72 = shape[FeaturePoint.InnerTopLeftPupil].Y;//added 30.04.2015
                var Z_72 = shape[FeaturePoint.InnerTopLeftPupil].Z;//added 30.04.2015



                textblockeye3.Text = string.Format("InTLPupil: X:{0:0.000}
Y:{1:0.0000} Z:{2:0.000}", X_72, Y_72, Z_72);

                jawLowerer = jawLowerer < 0 ? 0 : jawLowerer;
                //lefteye_top = lefteye_top < 0 ? 0 : lefteye_top;// added 30.04.2015
                //var a = (lefteye).ToString();// added 30.04.2015

                textBlock4.Text = string.Format("LowerJaw:{0:0.000}", jawLowerer);
                //textblockeye.Text =
string.Format("lefteye:{0:0.000}",lefteye_top);//added 30.04.2015


                //system time
                //Time.Text = string.Format("{0:HH:mm:ss tt}", DateTime.Now);


                if (jawLowerer >= 0.5)
                {
                    textBlock5.Text = string.Format("Yawning");
                    textBlock5.Foreground = Brushes.Red;

                    if (jawLowerer >= 0.5 && oldjawLowerer < 0.5f)
                    {

                        int Yawn_count = Count++;
                        textblock_Yawn_Count.Text = string.Format("Yawn
count:{0:0.000}", Yawn_count);

                        TimeSpan ts = stopWatch.Elapsed;

                        //textBlock2.Text = string.Format("Elapsed: {0} s",
ts.TotalMinutes);

                        double yawn_frequency = (Yawn_count / ts.TotalMinutes);

                       // textBlock3.Text = string.Format("Yawn_freq: {0}",
yawn_frequency);

                        if (yawn_frequency >= 3f && Yawn_count > 3f)
                        {

                            Count = 0;
```

```
                            //  textBlock_Fatigue.Text = string.Format("Driver is
Tired");
                             // textBlock_Fatigue.Foreground = Brushes.Red;

                        }
                        else
                        {
                            // textBlock_Fatigue.Text = string.Format("Driver is NOT
tired");
                            // textBlock_Fatigue.Foreground = Brushes.Blue;
                        }
                        old_frequency = yawn_frequency;

                    }
                    else
                    {
                        //textBlock5.Text = string.Format("Not Yawning");
                    }


                }
                else if (jawLowerer < 0.5)
                {
                    textBlock5.Text = string.Format("Not Yawning");
                    textBlock5.Foreground = Brushes.Blue;
                    //Count = 0;
                }

                oldjawLowerer = jawLowerer;

                // Head Pose Angles
                Vector3DF faceRotation = faceFrame.Rotation;


                // X - Pitch
                // Y - Yaw
                // Z - Roll
                textBlock6.Text =
string.Format("Pitch:\t{0:0.000}°\nYaw:\t{1:0.000}°\nRoll:\t{2:0.000}°",
faceRotation.X, faceRotation.Y, faceRotation.Z);

                //Yaw
                if (faceFrame.TrackSuccessful)
                {
                    if (faceRotation.Y >= 30f || faceRotation.Y <= -30f)
                    {
                        Y = true;
                        FaceDirection.Text = string.Format("Driver is looking to the
side");
                        if (faceFrame.TrackSuccessful)
                        {
                            int counts = countz++;
                            counts = counts / 10;
                            //elapsed.Text = string.Format("Timer_Y {0:0.00}",
counts);
                            if (counts >= 10)
                            {
                                countz = 0;
                                if (faceFrame.TrackSuccessful)
                                {
```

```
                                    if (faceRotation.Y >= 20f || faceRotation.Y <= -
20f)
                                    {

                                        Status_textBlock.Text = string.Format("Driver
is NOT looking in direction of travel");

                                        Status_textBlock.Foreground = Brushes.Red;
                                        dingPlayer.Play();

                                    }
                                }

                            }
                        }
                    }
                    else
                    {
                        Y = false;
                        if((Z != true) && (X != true))
                        {
                        countz = 0;
                        }
                    }
                }
                //Roll
                if (faceFrame.TrackSuccessful)
                {
                    if (faceRotation.Z >= 30f || faceRotation.Z <= -30f)
                    {
                        Z = true;
                        FaceDirection.Text = string.Format("Driver's head is tilted to
one side ");
                        if (faceFrame.TrackSuccessful)
                        {
                            int counts = countz++;
                            counts = counts / 10;
                            //elapsed.Text = string.Format("Timer_Z {0:0.00}",
counts);
                            if (counts >= 10)
                            {
                                countz = 0;
                                if (faceFrame.TrackSuccessful)
                                {

                                    if (faceRotation.Z >= 30f || faceRotation.Z <= -
30f)
                                    {

                                        Status_textBlock.Text = string.Format("Driver
is NOT looking in the direction of travel");

                                        Status_textBlock.Foreground = Brushes.Red;
                                        dingPlayer.Play();
                                    }
                                }

                            }
                        }
                    }
                    else
                    {
```

```
                        Z = false;
                        if(( Y != true) && (X != true))
                        {
                        countz = 0;
                        }
                    }
                }

                //Pitch
                if (faceFrame.TrackSuccessful)
                {


                    if (faceRotation.X >= 30f || faceRotation.X <= -30f)
                    {

                        X = true;
                        FaceDirection.Text = string.Format("The Driver's head is
nodding up and down");
                        if (faceFrame.TrackSuccessful)
                        {
                            int counts = countz++;
                            counts = counts / 10;
                            //elapsed.Text = string.Format("Timer_X {0:0.00}",
counts);
                            if (counts >= 10)
                            {
                                countz = 0;
                                if (faceFrame.TrackSuccessful)
                                {
                                    if (faceRotation.X >= 20f || faceRotation.X <= -
20f)
                                    {

                                        Status_textBlock.Text = string.Format("Driver
is NOT looking in direction of travel");

                                        Status_textBlock.Foreground = Brushes.Red;
                                        dingPlayer.Play();
                                    }
                                }
                            }
                        }
                    }
                    else
                    {
                        X = false;
                        if ((Y != true) && (Z!=true))
                        {
                            countz = 0;
                        }
                    }
                }
                if(Y == false)
                {
                    if(Z==false)
                    {
                        if(X==false)
                        {
                            Status_textBlock.Text = string.Format("Driver is looking
in direction of travel");
```

```
                            FaceDirection.Text = string.Format("Driver is looking in
direction of travel");
                            Status_textBlock.Foreground = Brushes.Blue;
                        }
                    }
                }

                this.facePoints = faceFrame.GetProjected3DShape();
                //
                EnumIndexableCollection<FeaturePoint, Vector3DF> facePoints3D =
faceFrame.Get3DShape();
                EnumIndexableCollection<FeaturePoint, PointF> facePoints =
faceFrame.GetProjected3DShape();
                //

            }
        }

        ///hand distance Math
        private float oldDistance_R = 100.0f;
        private float oldDistance_L = 100.0f;
        private float oldjawLowerer = 100.0f;
        private double old_frequency = 100.0f;

        private float jointDistance_R(Joint first, Joint second)
        {
            float dX = first.Position.X - second.Position.X;
            float dY = first.Position.Y - second.Position.Y;
            float dZ = first.Position.Z - second.Position.Z;

           // Math_TextBlockR.FontSize = 15;
            //Math_TextBlockR.Text = string.Format("R_Hand: dX:{0:0.000} dY:{1:0.000}
dZ:{2:0.000}", dX, dY, dZ);

            return (float)Math.Sqrt((dX * dX) + (dY * dY) + (dZ * dZ));
        }

        private float jointDistance_L(Joint first, Joint second)
        {
            float dX = first.Position.X - second.Position.X;
            float dY = first.Position.Y - second.Position.Y;
            float dZ = first.Position.Z - second.Position.Z;

            //Math_TextBlockL.FontSize = 15;
            //Math_TextBlockL.Text = string.Format("L_Hand: dX:{0:0.000} dY:{1:0.000}
dZ:{2:0.000}", dX, dY, dZ);

            return (float)Math.Sqrt((dX * dX) + (dY * dY) + (dZ * dZ));
        }

        //Camera Capture
        void myKinect_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
        {
            using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
            {
                if (colorFrame == null) return;

                byte[] colorData = new byte[colorFrame.PixelDataLength];

                colorFrame.CopyPixelDataTo(colorData);

                kinectVideo.Source = BitmapSource.Create(
```

```
                                    colorFrame.Width, colorFrame.Height, // image
dimensions
                                    96, 96,  // resolution - 96 dpi for video frames
                                    PixelFormats.Bgr32, // video format
                                    null,               // palette - none
                                    colorData,          // video data
                                    colorFrame.Width * colorFrame.BytesPerPixel //
stride
                                    );
            }
        }
    }

}
```

9.2 Appendix 2

MainWindow.XAML

```
<Window x:Class="Project.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Fatigue Detection System" Loaded="Window_Loaded" mc:Ignorable="d"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" Height="966"
Width="1071">
    <Grid Margin="149,178,28,-74">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="219" />
            <RowDefinition Height="99*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="0" />
            <ColumnDefinition Width="247"/>
            <ColumnDefinition Width="123" />
            <ColumnDefinition Width="261"/>
            <ColumnDefinition Width="230" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Image Height="812" HorizontalAlignment="Left" x:Name="kinectVideo"
Stretch="Fill" Width="1013" SnapsToDevicePixels="False" VerticalAlignment="Top"
Grid.ColumnSpan="6" Margin="-127,-172,0,0" Grid.RowSpan="2" />
        <TextBlock Foreground="Blue" Height="32" HorizontalAlignment="Left"
Margin="198,-93,0,0" x:Name="textBlock4" Text="LowerJaw" VerticalAlignment="Top"
Width="224" Opacity="5" TextWrapping="Wrap" TextTrimming="WordEllipsis" FontSize="15"
Background="#2800FF0A" Grid.ColumnSpan="2" Grid.Column="3" />
        <TextBlock Foreground="Blue" Margin="-77,-93,100,479" x:Name="textBlock6"
Text="Head Angles" Opacity="5" TextWrapping="Wrap" TextTrimming="WordEllipsis"
FontSize="15" Background="#2800FF0A" Grid.ColumnSpan="2" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="198,-32,69,460" x:Name="textBlock5" Opacity="5" Text="Yawn Status"
TextTrimming="WordEllipsis" TextWrapping="Wrap" Grid.ColumnSpan="2" Grid.Column="3" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="198,34,69,349" x:Name="textblock_Yawn_Count" Opacity="5" Text="Yawn Count"
TextTrimming="WordEllipsis" TextWrapping="Wrap" Grid.ColumnSpan="2" Grid.Column="3" />
```

```
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue" Margin="-
77,9,100,419" x:Name="L_HandStatusTextBlock" Opacity="5" Text="L_Hand"
TextTrimming="WordEllipsis" TextWrapping="Wrap" Grid.ColumnSpan="2" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue" Margin="-
77,65,100,363" x:Name="R_HandStatusTextBlock" Opacity="5" Text="R_Hand"
TextTrimming="WordEllipsis" TextWrapping="Wrap" Grid.ColumnSpan="2" />
        <Label Content="Sensor Angle" Grid.Column="2" Height="28"
HorizontalAlignment="Left" Margin="121,138,0,0" x:Name="label1"
VerticalAlignment="Top" Width="109" Grid.Row="1" Grid.ColumnSpan="2" />
        <Slider Grid.RowSpan="3" Height="323" HorizontalAlignment="Left"
Margin="41,407,0,0" Maximum="27" Minimum="-27" x:Name="sliderAngle"
Orientation="Vertical" VerticalAlignment="Top" Width="20" Grid.Column="3"
ValueChanged="sliderAngle_ValueChanged_1" RenderTransformOrigin="0.5,0.5" >
            <Slider.RenderTransform>
                <TransformGroup>
                    <ScaleTransform/>
                    <SkewTransform/>
                    <RotateTransform Angle="-89.955"/>
                    <TranslateTransform/>
                </TransformGroup>
            </Slider.RenderTransform>
        </Slider>
        <Label Content="+27" Grid.Column="2" Height="28" HorizontalAlignment="Left"
Margin="0,138,0,0" x:Name="label2" VerticalAlignment="Top" Width="31" Grid.Row="1" />
        <Label Content="-27" Height="28" HorizontalAlignment="Left"
Margin="214,138,0,0" x:Name="label3" VerticalAlignment="Top" Grid.Column="3"
Grid.Row="1" Width="28" />
        <Label Content="0" Height="28" HorizontalAlignment="Left" Margin="43,81,0,0"
x:Name="label4" VerticalAlignment="Top" Grid.Column="3" Width="16" Grid.Row="1" />
        <Button Content="Adjust" Height="23" HorizontalAlignment="Left"
Margin="228,110,0,0" x:Name="btnSetAngle" VerticalAlignment="Top" Width="75"
Grid.Column="3" Grid.Row="1" Click="btnSetAngle_Click_1" Grid.ColumnSpan="2" />
        <Label Content="0" FontSize="20" Height="34" HorizontalAlignment="Left"
Margin="112,138,0,0" x:Name="lblSliderValue" VerticalAlignment="Top" Grid.Column="3"
Grid.Row="1" Width="21" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
HorizontalAlignment="Left" Margin="232,28,0,349" x:Name="Status_textBlock" Opacity="5"
Text="Driver is looking in Direction of travel" TextTrimming="WordEllipsis"
TextWrapping="Wrap" Width="224" Grid.Column="1" Grid.ColumnSpan="3" />
        <TextBlock Height="42" HorizontalAlignment="Left" Margin="197,237,0,0"
x:Name="textBox1" Text="" VerticalAlignment="Top" Width="50" Grid.ColumnSpan="2" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="232,-93,175,455" x:Name="FaceDirection" Opacity="5" Text="Face Direction"
TextTrimming="WordEllipsis" TextWrapping="Wrap" Grid.ColumnSpan="4" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="254,267,0,102" x:Name="textblockeye" Opacity="5" TextTrimming="WordEllipsis"
TextWrapping="Wrap" Grid.ColumnSpan="2" Text="eye1" Grid.Column="3" IsEnabled="False"
Visibility="Collapsed" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="254,366,0,35" x:Name="textblockeye2" Opacity="5" TextTrimming="WordEllipsis"
TextWrapping="Wrap" Grid.ColumnSpan="2" Text="eye2" Grid.Column="3" IsEnabled="False"
Visibility="Collapsed" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="254,0,0,154" x:Name="textblockeye3" Opacity="5" TextTrimming="WordEllipsis"
TextWrapping="Wrap" Grid.ColumnSpan="2" Text="eye3" Grid.Row="1" Grid.Column="3"
IsEnabled="False" Visibility="Collapsed" />
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="0,263,0,165" x:Name="HeadStatusTextBlock" Opacity="5" Text="Head"
TextTrimming="WordEllipsis" TextWrapping="Wrap" HorizontalAlignment="Right"
Width="224" Grid.ColumnSpan="3" Visibility="Collapsed" />
        <Label Grid.ColumnSpan="4" Content="Face Direction" HorizontalAlignment="Left"
Height="31" Margin="232,-124,0,0" VerticalAlignment="Top" Width="224" FontSize="16"/>
```

```
        <Label Content="Face Direction after 5 seconds" HorizontalAlignment="Left"
Margin="232,-7,0,0" VerticalAlignment="Top" Grid.ColumnSpan="4" Width="236"
FontSize="16"/>
        <Label Content="Head Rotation" HorizontalAlignment="Left" Margin="-77,-
124,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="224" FontSize="16"/>
        <Label Content="Jaw Travel" Grid.Column="3" HorizontalAlignment="Left"
Margin="198,-119,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="224"
FontSize="16"/>
        <Label Content="Yawn Status" Grid.Column="3" HorizontalAlignment="Left"
Margin="198,-58,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="224"
FontSize="16"/>
        <Label Content="Yawn Count" Grid.Column="3" HorizontalAlignment="Left"
Margin="198,0,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="224"
FontSize="16"/>
        <Label Content="Status of Left Hand" HorizontalAlignment="Left" Margin="-77,-
22,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="224" FontSize="16"/>
        <Label Content="Status of Right Hand" HorizontalAlignment="Left" Margin="-
77,34,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="224" FontSize="16"/>
        <TextBlock Background="#2800FF0A" FontSize="15" Foreground="Blue"
Margin="232,130,175,253" x:Name="Phone_Detected_TextBlock" Opacity="5" Text="Cellular
Telephone Status" TextTrimming="WordEllipsis" TextWrapping="Wrap" Grid.ColumnSpan="3"
Grid.Column="1" />
        <Label Content="Phone Status after 5 seconds" HorizontalAlignment="Left"
Margin="232,94,0,0" VerticalAlignment="Top" Grid.ColumnSpan="4" Width="236"
FontSize="16"/>
    </Grid>
</Window>
```