MAY 7, 2015

ROAD AWARENESS SOLUTION

MARTIN MCKENDRY
SUPERVISOR: SHANE DOWDALL

Acknowledgements

I’d like to thank Shane Dowdall my supervisor for all of his guidance, time and effort with helping develop the concept of the project. John Loane for his advice, feedback and work as the second supervisor. My family for their constant support and devotion to me. Finally I’d like to thank all my friends for the help, company and comfort.

Declaration

I hereby declare that the work described in this project is, except where otherwise stated, entirely my own work and has not been submitted as part of any degree at this or any other Institute/University.

_____

Martin McKendry

2015

Abstract

The following paper is concerned with the development of a system that utilizes facial detection and eye tracking. This is to promote driver awareness and vigilance on the road by alerting them when they experience a lapse in attention. It is built using the Raspberry Pi micro-computer and its peripheral camera module the Pi Camera. It has been written in Python and uses Haar like features and cascade classifiers for robust rapid real time face detection and eye tracking.

Introduction

Facial recognition is a branch of image processing and analysis with the intention of detecting human faces and features from digital images and video sources. It is an area of research that has generated enormous traction in recent years. Some of the causes of this are a result of its varied and flexible application in many sectors spanning safety, security and law enforcement as well as having numerous commercial uses. In this project the focus is on the safety of drivers. The system outlined in the following paper will use rapid real time feature detection to deliver a safer and more vigilant drive. This is achieved by using Haar like feature cascade classifiers and the Raspberry Pi.

Table of Contents

Table of Figures

Aim of Project

The aim of this project is to create a lightweight, cost effective solution to help promote driver safety and awareness using facial recognition. The system will fit right into a car and will utilize a micro-computer and a camera module to deliver a machine that monitors driver attention on the road. The system will analyze the user's features and determine when their attention has begun to wane and alert them via a flashing L.E.D and audio cue that they have experienced a lapse in concentration so they can attempt to correct the behavior.

Literature Review

Hardware

The Raspberry Pi

One of the technologies I may intend to utilize throughout this project is the "Raspberry Pi" also referred to as the "RPI" or "Pi". If you are unfamiliar with this or haven't come across the Pi before a quick look on their website reveals that;

> *"The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do"* (Raspberry Pi Foundation, 2014).

I am looking at the model "B" which has twenty six GPIO pins, two USB ports, 512MB of RAM and 100MB Ethernet port. There are two different models of the RPI at time of writing with variants of each such as the model "B+" and "A+".

The RPI is a very capable machine that has garnered popularity with hobbyist and educational institutes because of its flexibility, versatility and cost effective nature. It uses an ARM processor

as opposed to an Intel or AMD processor that most people would be familiar with. An advantage to this is that it draws much less power than its aforementioned counter parts. This stems from its roots in the embedded hardware industry where a high premium is imposed on power usage and generated heat (Membry, Hows, 2013).

The Arduino

A similar technology that could have been used for this project is another micro-controller called the "Arduino". Also very popular with hobbyists and nearly anyone with an avid interest in technology. The Arduino actually pre-dates the Raspberry Pi. Possibly due to this fact, there is a much wider catalog of Arduinos that can be purchased. These different versions are a result of the micro controller's tendencies to be aimed toward very specific problem spaces. This is the reason that there is such a variety in their design.

The Arduino has its own development environment and language variant for writing its programs. Unlike most other micro controllers that only support windows, the Arduino integrates with all of the major platforms namely Linux, MAC and Windows respectively. Its advantages are its inexpensiveness and open source nature making it easily accessible.

There are so many different models of the Arduino so I have explored the "Arduino Uno" most extensively. Its website gives us a clear look at its capabilities.

> *"The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection" (Arduino.cc, 2014).*

MIPS Creator CI20

A micro-computer that seems to be suited well to the task is the MIPS Creator CI20. A future competitor to the Arduino and indeed the Raspberry Pi, this little power house boasts a 1.2Ghz dual core processor, PowerVR SGX540 graphics and 1 GB of DD3 RAM to play around with according to its specifications (Develop-online.net, 2014). Keeping with the trend set by some of the newer micro-computer/controllers this machine is relatively cheap coming in at roughly 60 euro. The disadvantage of this is that at the time of writing it is only taking pre-order and is set to release early January of 2015.

Pi Camera Module

The Raspberry Pi can be purchased with its own accompanying camera module which is capable of full HD recoding and capturing of still images.

USB Webcam

As an alternative to the Raspberry Pi's camera module a normal USB webcam could be used for this type of application. However the vast majority of these cannot record video and images in HD. The downside of this can be a slower detection within the video input which will not be ideal considering the nature of this application. However keeping in mind that they can be bought in most electrical outlets leans heavily in their favor making them worth consideration.

Software

OpenCv

OpenCv is short for "Open Source Computer Vision" This is a compilation of libraries based around computer vision. Some of the algorithms and methods that will be discussed can be used in tandem with OpenCv. According to OpenCv.org "OpenCv was designed for computational efficiency and with a strong focus on real-time applications."

It is available in many languages such as C, C++, Python and Java. The result of which is the libraries versatility across platforms such as Windows and Linux making it a desirable solution to plenty of problems and projects.

OpenBR

OpenBR according to its official description (Openbiometric.org, 2014) "A communal biometrics framework supporting the development of open algorithms and reproducible evaluations" This is a general purpose library for all manner of biometric analysis. It has a wide variety of applications from facial recognition, age estimation and even gender estimation

It contains, and to an extent builds on existing algorithms. OpenBR supports many platforms particularly the major ones like MAC and Windows and also Linux, which is what we are interested in.

LibFace

LibFace is a relatively new solution and aims to deliver facial recognition with open source software. The library relies on OpenCv 2.0 but has taken the position of removing the need to write actual code from OpenCv itself (LibFace, 2010).

Similar to other competing software libraries it supports MAC, Windows and Linux distributions. It's written in C++ but can be used for solutions and systems running other languages like Python for example.

Research

Below are the algorithms and technologies that have been deemed the most appropriate for the project requirements.

Haar feature-based cascade classifiers

One approach that has been considered is using “Haar – like” features with cascade classifiers. This algorithm, like many others is based on the original real time face detection algorithm proposed by Paul Viola and Michael Jones. The general approach taken by this system and others like these are to take in a digital image or video and bring them into a “window” from here the image is split into a grid. Each section is called a “sub-window” which is usually 24X24px. These Haar features are applied to every sub-window to find potential face candidates (Sriram, Illuri, 2014).

For each Haar-like feature, there is a threshold which indicates accept or reject. For example, the threshold may say that the difference between the dark and light areas must be greater than 10 for it to be possible that a face exists at this location.

Fig 1. Haar features being applied to human features (codeproject, 2014).

If an object, in this case a face is not detected then the sub-window is discarded. However if a face is detected within the window then it applies a stronger set of classifiers on top of this in a cascade structure until it can it is confirmed that the object is in fact a face. In order for the algorithm to be competent or anyway efficient in picking out faces it needs to be trained.

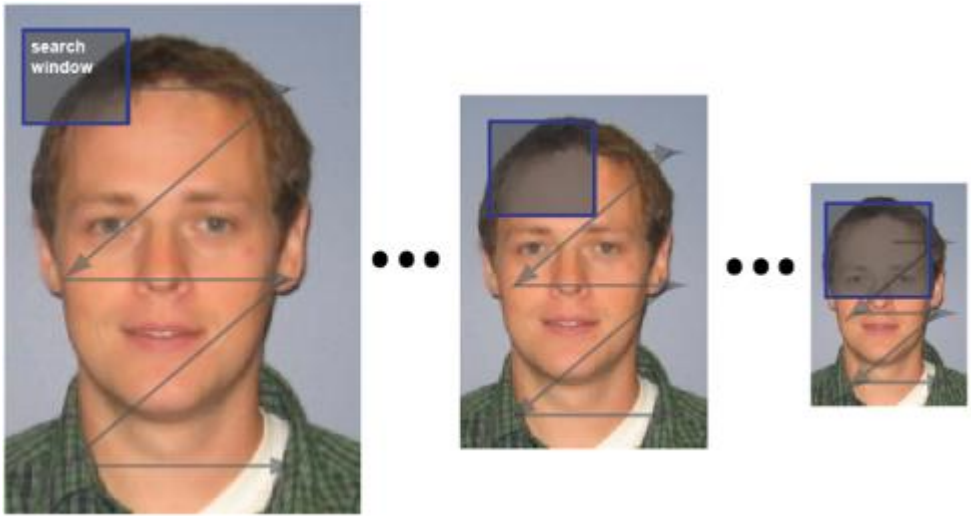

Fig 2. Image showing the cascade classifiers (Mathworks, 2015)

The computer learns what to search for by being fed a collection of positive (images with faces) and negative (images without faces) pictures which it uses as a baseline. For the purpose of this project the algorithm will be applied with "Adaboost" short for adaptive boosting which eliminates the need to train the computer myself. Like the Viola-Jones' algorithm mention below it utilizes the rectangles because of their ability to be rapidly computed in an area where performance is paramount.

Eye Tracking

Pupil tracking: one possible avenue of eye tracking is tracking the actual pupil itself. With an algorithm similar to the one described in Dr Fabian Timm's paper (Timm, Barth, 2012). The algorithm uses gradient images and an objective function to produce dot products. The algorithm would then search for the highest concentration of intersecting gradient vectors and use that as the location of the pupil.

Another possibility is finding the pupil by a collection of the darkest pixels in the image. Using Haar like features to detect the eye feature. Extract the region around the eye and apply a 'Gaussian blur' to it. This will effectively average out the pixel intensities within the area. The reason for this is because the image may contain the darkest pixel in the eyelash or perhaps eye brow. With averaging the pixel strengths within the area then the collection of darkest images should be the pupil.

Alternatively a more simplistic approach could be simply detect the eyes with the Haar like features and then split the eye into a small grid. It would allow for the pixels in the separate areas to be measured and the area with the highest concentration would be the iris. With this approach

the extent of the direction of the eye could also be measured to ensure its not sitting in a distracting position.

When checking if an eye is pointing in the wrong direction a threshold region around the eye would be set up to ensure that the eye is operating safely within the area. If it leaves the area this could trigger a function to alert the system.

The program would also need to determine the position of the eyelids and examine if they are too low which could hint at driver exhaustion. Similarly if the eye lid is closed then that would sound an alert.

Adaboost

This is short for "adaptive boosting" which is a machine learning algorithm used for "training" computer system for different types of recognition for both faces and objects.

> *"Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules" (Schapire, 2013).*

The general idea of adaptive boosting is to have a training data that it analyses and comes up with weighted "rules" to formulate an educated guess on how to proceed. In this case however it would make a rough guess as to whether it has detected faces or features.

Because of the fact that this type of rapid detection has high rates of probability between sections of images containing the objects or the sections devoid of the objet we need to find a "general" rule. This is just something that is better than random guessing. This is called a "Weak Learner" and is at the root of it just a rule of thumb. It is safe to assume that the weak learner consistently finds the weak classifiers from here the boosting can generate a single weighted classifier which accurately classifies the data. This is known as a weak learner assumption. (Emer, 2014).

Eigen Faces

Eigen Faces in reality is just the name given to a collection of Eigen Vectors that are used for recognizing a human face with computer vision. The general methodology behind the Eigen Faces algorithm is to gather together a large collection of sample images of faces and like similar approaches it uses this bank of images to train the computer what to recognize. The end result

being that the new system is able detect faces that's It has never seen before from a mathematical object called an Eigen Face.

> *"Most naturally, we think of an image as a matrix of pixel values. For simplicity, we restrict our attention to grayscale images. Recall that a pixel value in the standard grayscale model is simply an unsigned byte representing pixel intensity. In other words, each pixel is an integer ranging from 0 to 255, where 0 is black and 255 is white" (Jeremy Kun, 2011).*

The design of Eigen Faces uses the matrix of pixels and their intensities to measure the deviation between the two pictures. So boiled down to a bare minimum it acts like a sophisticated 'spot the difference'.

Viola-Jones Object Detection

The Viola – Jones detection algorithm was the precursor to a vast majority of the facial detections methods that we see today. Their algorithm was the first real time facial recognition algorithm so naturally the ones that followed have built upon the system that they have laid out.

Viola and Jones' algorithm use the Haar features and a cascade classifier structure to discern objects from digital images in real time. Using rectangular features it calculates the sum of all of the pixels contained within the black portion of the rectangle and subtracts it from the sum of all of the pixels contained within the white area of the rectangle. Some of these features can be more complex than the others. For example the most basic features have only two segments (one black and one white) and some can have up to four segments two black and two white respectively.

Each of these features have different purposes for example:

> *"The value of a two-rectangle feature is the difference between the sums of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent. A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles"* (Viola, Jones, 2001).

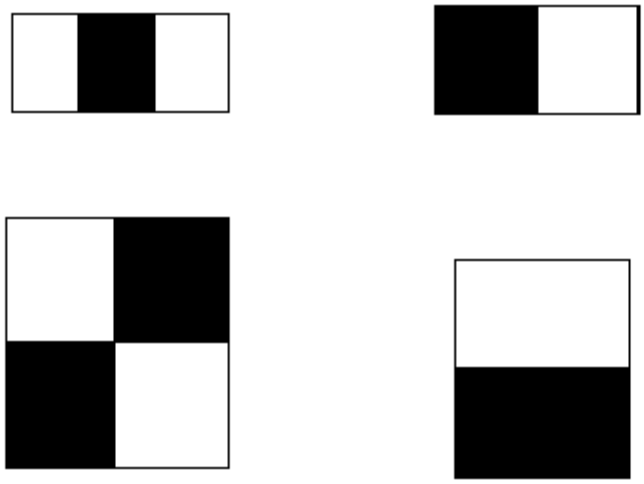

Fig 3. Examples of different Haar features

Like in a similar algorithm listed above this particular one uses rectangles sections as these can be rapidly computed. In addition the idea of making sub windows and analyzing them individually as opposed to applying the Haar features to the whole image increase the performance an exponential amount as each image an contain upwards of 16000 features.

Compare/Contrast of Algorithms

Eigen Faces

While Eigen Faces is a viable solution for face recognition in general, it is more suited to static images rather than actual real time detection itself. For the most of this paper “Facial Detection” has been synonymous with ”Facial Recognition” because of the way it has been applied. For the justification on why the project is not using Eigen Faces this difference between the two is important.

While Eigen Faces could be used for this system the way the algorithm operates it would perform better for facial recognition meaning it is efficient in detecting specific faces and wouldn’t be used to its full effect or potential. The nature of its implementation is discerning specific geometries within the images; for want of a better term “spotting the differences” and it must be trained to

look for faces as objects where it is more appropriate to train it to recognize an individual or group of faces for it to look for rather than human faces as a whole.

Viola-Jones / Haar like features and cascade classifiers

The Viola-Jones Object Detection algorithm is the precursor to most of the other algorithms discussed in the papers as it was the original 'Real time' solution for object detection therefore it can appear to have quite a lot in common with the attributes of the previous processes.

The differences between the Haar features algorithm and the Viola-Jones Object Detection method are minor they both operate with the Haar features and classifiers detecting faces in images. However the Viola-Jones algorithm is for detecting any type of object as long as the computer is trained to recognize them.

For this reason the Haar like features is a better choice as we already will have a pre-learned AdaBoost performing in concert with the algorithm optimized for picking out human features from the images. Rather than building our own AdaBoost and spend time feeding it the positive and negative images needed to come up with weak classifiers.

Chosen Approach / Proposed Solution

As a solution to the problem, I have chosen to go with the Raspberry Pi. It will use its accompanying Pi camera module. This solution will utilize the Haar like features and cascade classifiers that are present in the OpenCv libraries.

From examination of the available technologies mentioned in the report the Raspberry Pi seemed to be the most appropriate solution for this.

Through research it has been concluded that the Haar like features with cascade classifiers is the best approach to take. Its powerful and lightweight has plenty of available documentation and references online and will run natively on the RPI. Its efficiency is owed to the fact that it can calculate the features rapidly and effectively. Size is no impediment to the Haar features as it can compute objects just as quickly and efficiently as a small number of pixels (Tripathy, Daschoudhury, 2013).

Below will be a more in-depth explanation of the chosen algorithm along with a look at the steps it takes to discern faces from a digital image.

Fig 4. A visualization of the algorithm checking the sub windows of an image

Step 1) The algorithm will take in a digital image as input and transform it into greyscale. This way it can deal with pixel intensity rather than RGB values. The pixel values are now going to be either light or dark (0 or 255).

Step 2) The adaboost learning algorithm will generated the weight of the features and the size as well.

Step 3) The grey scale image will be split into numerous sub windows of 24 X 24px. This will allow the function to check an individual section at a time as opposed to all of the image at once thus reducing computational processing.

Step 4) From here the algorithm will begin to apply the weak classifiers to each sub window searching for candidates that fulfil the criteria. The weak classifier will apply the minimum number of features to detect a face as calculated by the AdaBoost.

Step 5) If the sub window does not contain a viable option for detection then the algorithm discards it and doesn't look at it again.

Step 6) When a suitable candidate is discovered in the sub window then a weak classifier is applied. Here is where the cascade classifier structure comes in as it is at this stage that the classifiers can start rapidly discarding undesirable contenders as it applies increasingly stringent classifiers. These classifiers apply more and more features to the images the more complex the classifiers. See Fig 4.

Step 7) When a potential contender passes all of the progressively difficult classifiers then it will be allowed to exit the cascade structure. Similarly the candidate will be thrown away if it fails to meet the requirements of any of the classifiers.

Step 8) If the possible match detected in the sub window has made it all the way to this step then the system is confident that the object in question is indeed a human face.

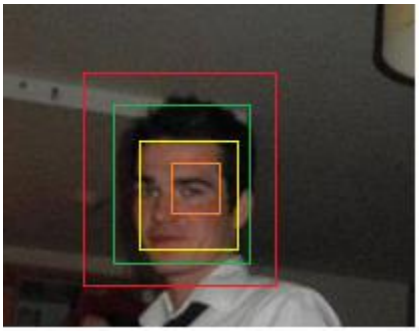

Fig 5. Image showing how classifiers are applied more rigidly for confident detection of faces.

Context

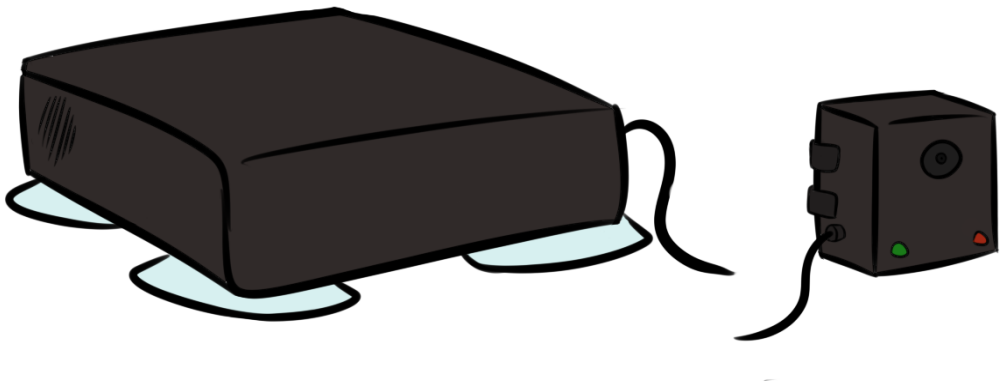

Fig 6. A mock-up image of what the system may look like.

This project intends to use some type of speakers and L.E.D lights as a form of alerting the user of attention lapses. After the 1st alert the system should flash the lights as a "silent alert" warning the user that their attention has faltered. With the second alert the lights will flash for longer and an audio alert will sound. On the third alert things are getting serious the driver has lapsed three times during the drive and the system wants to make sure that they are alert and stay vigilant. So with this third warning the lights will blink and the audio cue will sound for a longer period of time. The system may then recommend that you take a small break to rest or have a cup of coffee or an energy drink.

Speakers

Any speaker with a standard 3.5mm audio jack will work with the Pi to produce audio output so this isn't really worth delving into as any speaker will do. For the prototype a small speaker would add more benefit than a bulky one especially in a vehicle.

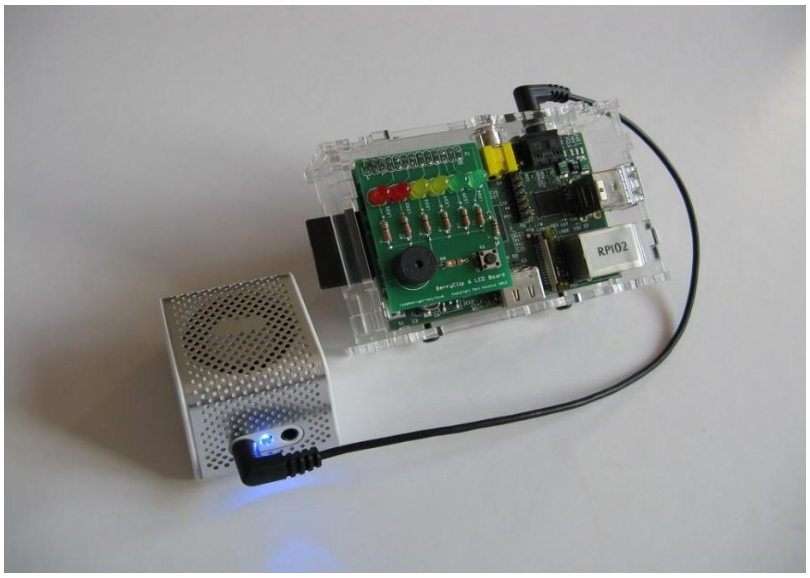

Fig 7. An example speaker that could be used (RaspberryPiSpy, 2013)

L.E.Ds

The L.E.D's will fit into the GPIO pins on the PI these are "General Purpose Input / Output" pins. These pins are programmable and can be controlled with software on the RPI. The instructions to control the lights will be written in Python which contains a library "RPi.GPIO" to aid in controlling the L.E.Ds

Fig 8. Example of L.E.D (Indestructible, 2014).

Planned Features

In addition to the system using face detection to alert the driver that their attention has lapsed, features that planned to be added after the initial solution is complete is eye tracking for greater accuracy and logs for the ability to review your progress.

The logs will provide a platform for the driver's progress to be recorded. These will contain information on the frequency of alerts trigger and the time between each alert. Hopefully armed with this information the user will be able to correct any problems in their behavior and ultimately become a safer and more vigilante driver.

Justification

Raspberry Pi

The Raspberry Pi is what I have chosen as part of my solution. While all of the systems mentions are quite powerful and very cost efficient. The RPI was the most appropriate solution for the problem space as it's completely closed boxed. With the Arduino being a micro controller its functionality is restricted in the respect that what it does it does well but it's very limited to doing that same operation until it is reprogrammed.

The Pi on the other hand can actually be used to develop the solution itself considering that it's a fully-fledged computer with the unique perspective of fitting in the palm of your hand. With this in mind there is the power to modify or create new functionality on the fly without the need to plug it into an external system for future developments. Which is quite appealing for such a small investment.

The Arduino, after much research appears to be lacking in some of the criteria that had been hoped to be used in this project. Taking into account that it is a micro-controller and not a computer it is a little lackluster. It is possible to accomplish the task with this technology however I feel that its limited flexibility means that moving forward with the Raspberry Pi is the safer alternative.

The MIPS Creator CI20 would definitely been a feasible technology to use to implement the project as it certainly is cost efficient for its size. The machine also has incredible processing power

which would have assisted in the real-time facial processing. The glaring problem with the MIPS Creator CI20 is that it's only pre-order until it is released in January of 2015.

Pi Camera Module

The Pi's peripheral camera module is a superior technology to include here instead of a standard USB webcam. While the benefits of the USB camera remain that they are cheap and can be bought in any electrical outlet. The Pi's camera module integrates seamlessly with the RPI utilizing specific built in functionality. It also supports HD video recording ensuring a high quality image to analyze whereas not all standard webcams do not.

Haar features

The reasons for choosing the Haar features as the algorithm is that it is ideal for a closed box system like the raspberry pi. The algorithm is quite robust while still maintaining consistent performance thanks to the use of the adaboost and classifiers. Due to the fact that it does not need to apply features to the whole image at once it can cut down processing requirements which in a project like this there is a high premium on the minimum amount of processing power needed.

The Haar like features builds on the ground work laid out by the Viola-Jones Object Detection methodology. The reason that the Haar features won out over Viola-Jones is that it contains the same principles as Viola and Jones work while removing some of the concepts that aren't strictly needed for this project while remaining robust and effective.

OpenCv

OpenCv was a desirable choice for many reasons, among them its price. Keeping in mind that this is to be a cost effective software/hardware solution open source software is always a favorable choice. A further advantage to OpenCv is its wide range of interoperability between numerous platforms. This feature is appreciated as the Raspberry Pi natively runs Linux which, of course is compatible. Alongside running a Linux-based operating system the Pi's native language is Python which is supported as one of OpenCv many languages on offer.

Unlike other potential candidates there is plenty of literature and documentation online from these resources I have been able to get a good grasp on the software's capabilities and now can speak from a position of confidence that this will be a valuable addition to the project.

Future Work

As a roadmap for the future:

Road Map	
26th of January:	Begin implementation of the project working in tandum with the final report
23rd of March	Have the solution implemented and begin testing the project with different users
9th of April:	Implement any addition features in the remaining development time
15th of April	Submit the first draft of the final report to the supervisor
20th of April	Submit draft poster to supervisor
27th of April	Upload the final poster
30th of April	Submit the final draft of the project report
7th of May	Final submission of all of the project

Fig 9. A roadmap for future work

Design

This project was designed to increase a drivers focus and vigilance by monitoring them while they driver and alert them in the event that they losing focus or are in danger of falling asleep.

They system has a 3 distinct elements that are to be achieved and they are

- Detect the drivers face.
- Track their eyes to ensure an appropriate attention level.
- Sound an alert if they are experiencing a lapse of attention.

The system will sit on the dashboard of the car where the camera will have a clear view of the driver. All the driver will have to do is turn on the Raspberry Pi due to its portable nature. From

this point on the Pi will monitor the driver and provide feedback if it feels that the driver is losing focus. Essentially it will run in one loop, continually checking to see if the user is still watching the road and if not alerting them to the fact that they should. It repeats this process until it is shut down.

The most important element that needs to be implemented in the project will be the face detection as the other features al need to build on this to make a robust system and a successful project. This will be achieved using the Haar like cascade classifiers mentioned in the above literature review. To re-iterate the classifiers are built up of different features. An image will be captured and split into a grid containing 24x24px sub windows all of which will be subjected to the classifiers sequentially.

The classifier will be applied to the first window and checked. If the classifier fails to find the facial features it is searching for that window will be discarded and never checked again for the remainder of the operation. This process will continue until the features are found. When a potential sub window is discovered the cascade nature of the classifier is used, meaning it will employ an increasingly rigid set of criteria while determining if the features are that of a face. When there is evidence proving that beyond a doubt that it has a face the co-ordinates will be sent back noting the location of the face(s).

The co-ordinates can be used to draw a bounding box around the face to make it visible when displaying the image revealing whether or not the operation was successful.

Another aspect of the project is the alert system as it will be the feedback that the user would experience. This element along with the facial detection together make up what I have chosen to be the minimum requirements. Failing to implement these two abilities the project would not be considered a success. The alert will primarily be an audio cue which will signal that the driver is losing focus. A flashing L.E.D light will also be used as a “soft alert” which may be used for the first alert as chances are that the lapse that has occurred would only be minor.

This alert function will most likely be the least complex entry to the project in terms of code but is an imperative addition to the system. It also relies on the facial detection being successfully implemented and will need to be revised when the eye tracking has been implemented and any additions needed to be created according to development needs.

The eye tracking will be important but there isn't as high a premium put on it as it depends on the completion of the other aspects of the project. Eye tracking will be achieved by discovering the area around the eye. This will be done in a similar fashion to the facial detection. Instead the classifiers here will be searching for the specific features that make up an eye. From there the center of the eye needs to be isolated and the position of it calculated. If the iris is deviating too far in any direction deemed to be outside of the bounds of normal eyes set on the road (This threshold will become clear in testing) then the alarm will be sounded. In the same vein is the iris is found to be too small like in the event that the eyelids are drooping then in this instance the alarm will be sounded as well.

All of this works to make sure that they are not distracted by looking around them or that their eyes are not being drawn away from an external distraction like a mobile phone.

Other methods that could be employed are searching for additional facial features not just restricted the eyes but possibly also the mouth. As we know everyone's eyes are different and that has the potential to cause inaccuracy in the program. To eliminate this the mouth could be analyzed in addition to the other features. This would essentially mean that even though the eyes are narrow (which could be a sign they are closing) the mouth may be smiling meaning that they could be laughing removing the possibility that they are falling asleep. This goal would be reached by isolating the mouth, from there checking if the mouth is open as this could be a yawn. If not take the center point of the mouth and the corners and turn them into points. If the two outside points are significantly higher than the center point it's safe to assume that the person is smiling and not in danger of having a lapse in attention.

This may be an afterthought when all of the other elements in the project are implemented correctly.

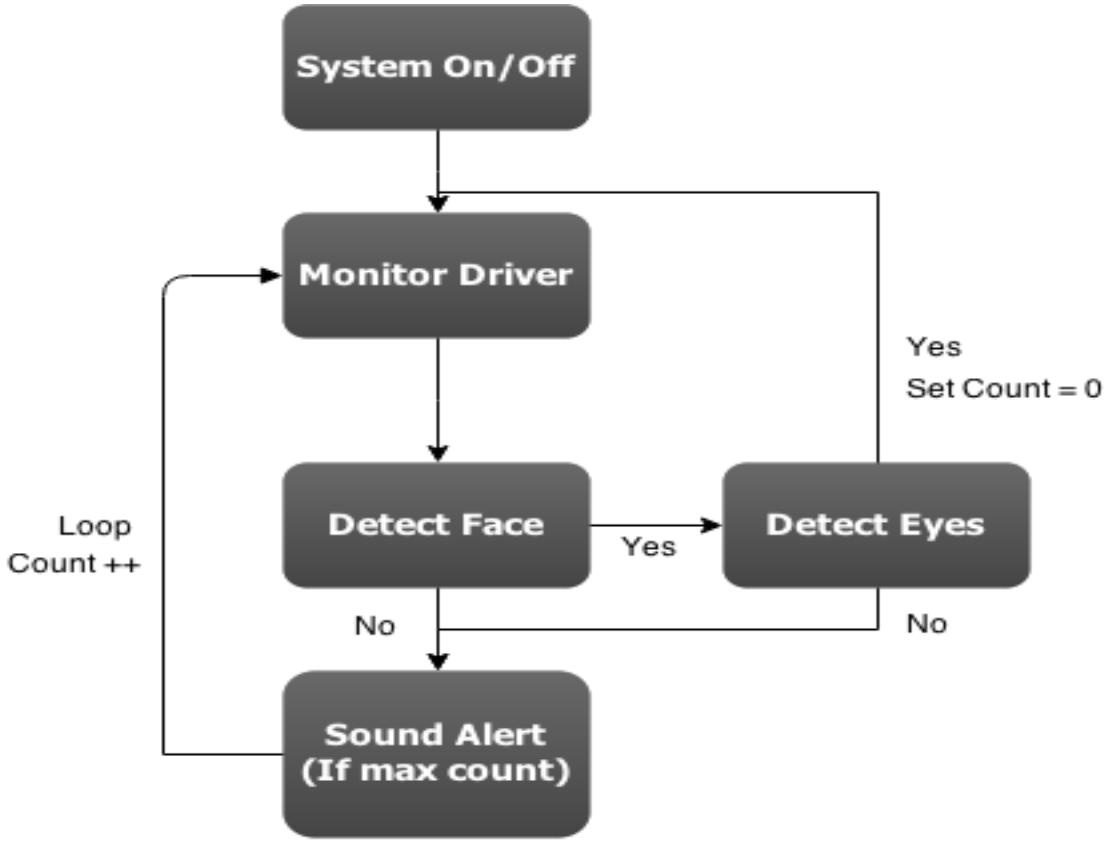

Fig 10. Overview of the system.

Technology used

OpenCv – was the computer vision libraries that worked with my python code to take advantage for processing the images and applying the classifiers. I was also able to implement smaller functions for displaying the images and extracting and drawing the regions of interest within the frames;

Raspbian – This is the operating system on the running on the Raspberry Pi. There are numerous OS's available for the micro-computer but raspbian is quite user friendly and easy to navigate while still maintaining a rich environment with a powerful level of configuration and modification options. The OS is also lightweight and doesn't take up too much real estate on the SD card.

Python – The object oriented programming language used to develop the system although I had very little previous exposure to it.

Geany – A small flexible and lightweight IDE. It was where all of the code was written. Despite its small footprint I was able to construct and keep track on the whole project. I had the benefit of syntax highlighting and even a small bit of built in code completion which is very hard to find in other text editors/IDE's that are available for the Pi.

TightVNCServer – An incredibly handy tool that is run over SSH and allows you to display the screen on another machine. This was used when I didn't have a spare keyboard and mouse available and allowed me to run the Pi in a headless portable setup while still having the ability to make big changes to the code and see the immediate results.

Git – A version control system that can be used on the Pi. This was the tool used to manage branches of code. I was able to upload a working version of the project code to remote repository and experiment and make changes without the danger of compromising the integrity of the code.

MPutty – While not on the Pi itself Putty and MPutty are two windows tools that I took advantage of the SSH into the Raspberry Pi to allow for remote access to the terminal which was needed for permission privileges and proved useful when I had no screen available. It could also run the file and start the system, again to see how it performed in a headless and portable environment.

Implementation

Environments/Setup

Development began in the second semester, around the 25th of January. I had the Raspberry Pi for a while but up until that point it hadn't been used. The operating systems and everything that was needed to get started had been installed. The OS used was a *Debian* distribution called 'Raspbian'. However I was still not familiar with Linux at this point. The first step needed was installing the environments that would be used throughout the development lifecycle. Introducing the IDE 'Geany' onto the RPI presented no problems and was relatively straight forward. Installing OpenCv proved more challenging. There are many different methods which one could use to install OpenCv onto the RPI. Unfortunately this was one of the biggest pitfalls.

After a small amount of trial and error installing OpenCv there was enough of the base elements present to begin writing code. As a low barrier to entry some of the first scripts written were bash scripts. Bash scripts would be needed further down the development lifecycle in order to run the application in a 'headless' setup. Starting with bash scripts had the added advantage of exposing the need for Linux commands and provided a platform to build knowledge in that area. Concepts like 'chmod' for changing system permissions and administration privileges. The ability to manually configure permissions for files and folders as well as the creation and deletion of these components greatly speed up initial development.

One of the first scripts written was to capture an image from the Raspberry Pi camera and save it to disk. The image was manipulated with build in functionality from the Pi camera for flipping the orientation horizontal or vertical depending on needs. A typical statement took the format of "raspistill –o cam.jpg". Where 'raspistill' denotes a still image, the "-o" is for capturing the image and then the filename, format and path is specified. The orientation can be changed with '-vf' and '-hf' after the –o.

This approach was feasible for getting to grips with programming on the Pi but it became apparent that this was the not the best option available as that command always required a warm up time of about 5 seconds each time a picture is to be taken. Couple this with the fact that the system needs to take multiple picture every second and it visible that this is not the ideal solution. More important it provided little to no flexibility in the program which is when code was moved to Python scripts.

Moreover another drawback to the standard commands of the RPI camera is they could only be ran in the command line. Standard image resolution was set to 2592 x 1944 meaning that each picture captured would be about 2.4MB which can create a problem later. Now that the basics were covered development moved to writing the python scripts that would actually be used in the system. The IDE Geany is a lightweight solution for the Linux that has a very small footprint which was used for all of the development. With Python scripts it's possible to use the basic pi camera commands and in inject them into the command line via the "os.system" command. However it wasn't long before more robust code was needed.

The first script that involved image processing was when experimenting with transforming the pictures into greyscale. On the first attempt the image was simply changed to only black and white pixel values. A threshold pixel value of 50 had been used, meaning that each pixel with a value greater than 50 was converted to black and the rest made white which really served little purpose for performing operations on the image.

In order to be able to gather pixel intensity values (0, 255) the image needs to be in black and white or more accurate In this case the image needs to be greyscale. Without this the XML classifiers will have no effect when applied they won't be able to process the information they are given. With the previous error out of the way grey scaling the image was the next objective.

With the system relying on OpenCv to provide the pre-trained classifiers (Writing and training them concurrently would have been a separate project in its self.) It was imperative that these classifiers were accessible. Development halted when it was discovered that these classifiers weren't present and the cause was traced back to OpenCv. It had not installed correctly. The Pi was then updated and upgraded, and more thorough information and research was retrieved in relation to OpenCv's install on the Pi. With a lot more work and a 14 hour build it had been installed correctly without further issues and contained all the necessary libraries and dependencies.

When OpenCv had finished compiling the system was able to gain access to the classifiers so development could continue and the facial recognition and eye tracking functionality could be created. Work was completed in iterations. Whenever work was being done on a new component it would be completed in a separate script before being merged with the main file.

Face Detection

Initially for the face detection a very large image was use. This took a bit of processing time as the image was 2592 x 1944px which is the default size of the captured images from the RPI. The first detection took roughly 2 minutes. When more classifiers were added into the detection algorithm the process naturally took longer. An unforeseen issue that had crept up with the higher resolution was that along with extra processing it produces extra errors as the margin for failure had increased with the number of possible feature combination within the image. The classifiers began to pick up facial feature that weren't present such as eyes within the patterns of a shirt and mistaking nostrils as eyes also.

How it works is by taking the picture or reading in an image from a video source then converting that image into greyscale (not a threshold). Then an operation called detect multi-scale is called with the cascade classifier as one of the parameters. This splits the image into multiple windows and applies the detection rules to the all of the possible windows. Each window is checked if no face is found then that window is discarded and never checked again. If there is enough features to contain a face then the classifiers are applied in a cascade manor. The criteria get more ridged with each pass. When the image had successfully passed all of the measures then it's positive that a face has been found.

All of this data is then saved into an array. The program iterates through the array and a bounding box is then drawn over the original colour image using the x, y, h and y values obtained from the locations of the features in the grey scaled image. This was now the ROI or 'region of interest' as it's the section of the picture that contains the face. The colour image can then be displayed on the screen with the visible boundaries of the face drawn. The displaying of the image and box was only used for development, in a real world situation this is not necessary.

To combat the high error likely-hood and to improve the performance of the detection the standard Pi camera functionality was no longer used and OpenCv code was used for the programming. After importing the PiCamera as the camera it was now possible to define the resolution and the camera warm up time. Which could be set for quicker image capturing. With the lower resolution of the image, which had now been set to 640 x 400 the time it took to process the image and detect a face reduced dramatically to within 10 seconds. Even with multiple classifiers added to pick out additional facial feature the system was still performing at a competent rate.

At this point the system was capable of taking in images that resided on the SD card of the RPI and also the images that were taken with the Pi camera attachment. The next step was taking a series of images. Writing In a provisional loop to take 10 pictures and save them to disk. It was found that after moving the capture method from the terminal commands to OpenCv the images behaved different because of the fact OpenCv treats images as NumPy arrays organized as RGB pixel values e.g. [255, 0, 0]. This works fine for saving the images and then reading them back in but becomes more difficult to take a picture and display it directly from the image variable. The reason being that it expects an image object but instead receives the RGB array. The reason that OpenCv accepts its images as arrays is due to the fact it is easier to access the values and perform the various different processes on them.

Within the first loop of the code it was still simply taking the images and saving them to disk where they could be viewed through the file navigation panel to check the success of the current iteration of the program. After which the code was altered so the loop would capture an image and them display it. This is when the problem of the image object versus the expected array became an issue. Up until that point the most logical course of action was to take an image a few times every second, analyse it and then output the result but it was proving difficult to resolve the differences in datatypes. Wasting time casting the objects into different datatypes didn't make sense and also wasn't very appealing so a new route had to be found.

As an alternative avenue to single pictures there was taking in a video stream. Using a continuous capture with the Raspberry Pi camera and then simply iterating through each frame. It was possible to analysing that frame finding a face if it was present and displaying the result. This proved a much more viable solution as it would be the same object type that the system would be working with the whole time.

What was now present was the basis of the main loop of the system. It would now take in a formatted RGB video stream as opposed to single pictures and process each frame as they came. The Pi can only analyse one frame at a time and it gets through roughly 4-5 a second depending on the amount of operations it needs to run on that particular frame and the resolution of the image which had now been set to 150x150.

With the facial component in place the alert was what was needed to be done next so as the project would meet the minimum requirements; monitoring the face and sounding an alert if the driver was absence in too many consecutive frames. The alert wasn't too challenging as it is a

matter of having a frames counter variable that is added to itself every iteration when the images fails to discern any features. In the event of the features being present it would reset the frames variable to 0. When the max count of the frames counter is reached it will play the alert tone. After the variable is set back to 0 when the drivers face is back in view of the camera.

A small setback encountered with that section of the algorithm was trying to pin down if the features were not being picked up by the camera. The information gained from the classifiers being applied to the frame were stored in an array named “faces”. Unfortunately it’s impossible to try and examine this array, for example getting its size or contents as its existence is based directly on whether there are faces present at the time of capture which changes from frame to frame. Meaning the array could be measured in one iteration and throw an error in the next.

Eye Tracking

The eye tracking was the last key feature that had to be implemented. As the Raspberry Pi’s processing power is limited a lot of work has been done on the project code to try and perform only crucial operations. This was no different when it came to the eye tracking phase. Originally the idea was to keep a constant known position of the eyes in real time. However further into development it became clear that this wasn’t suitable as it would drain all of the limited resources and processing power.

Instead to save processing power and increase performance the eyes would only be searched for if the face had been found to begin with. For instance after it checks for a face it will discard the frame when no face was found but will perform the additional eye tracking if it passes leading to slightly longer processing times. While this process is in effect any additional frames that the video feed submits are discarded.

This meant that the application would not waste time applying a separate classifier in vain. For this to be achieved the “faces” array that held the feature information would have to be exist. To get past the previous issue the array had to not be null and also have a length of 0. Then the x, y, h and w co-ordinates would be pulled out to create the region of interest. This ROI square would be used as the new image. This section would then be cropped to the upper left region where you would typically find the left eye. This further reduces the search window and protects against the chances of a false positive. In turn all of this should optimise the system and improve the overall accuracy.

With the above components implemented the next step was the testing. Much of the program code was configurable so the testing phase would be used to find the optimum values and settings

to ensure the best performance and accuracy while also having the opportunity to see where it could be improved.

Fig 11 Finished Prototype (excluding power supply), (Mckendry, 2015)

Testing

Much of the testing for the project was done in an “ad hoc” manner and was concurrent through most of the development. Each time work began on a new component or there was a concept needing to be tested a new file would be created for it. This meant that the new code wouldn’t conflict the working version of the system. There was a more structured set of tests performed after development was complete to gain the optimum configuration of the system and get some information in what settings it would perform the best.

One aspect of the project that has always been a paramount concern was the performance given the RPI’s small stature. It was important to get the correct resolution for the captured images. If they were too large in scale the processing power wouldn’t be sufficient to detect faces in a fast enough time scale. Also there is a direct correlation between the size of the image and the number of false positives. An image of a resolution of 640x400 still produces excess false features to be considered accurate enough. While if the image was too small, the system gets faster but the accuracy also drops to the level of not picking up any features at all.

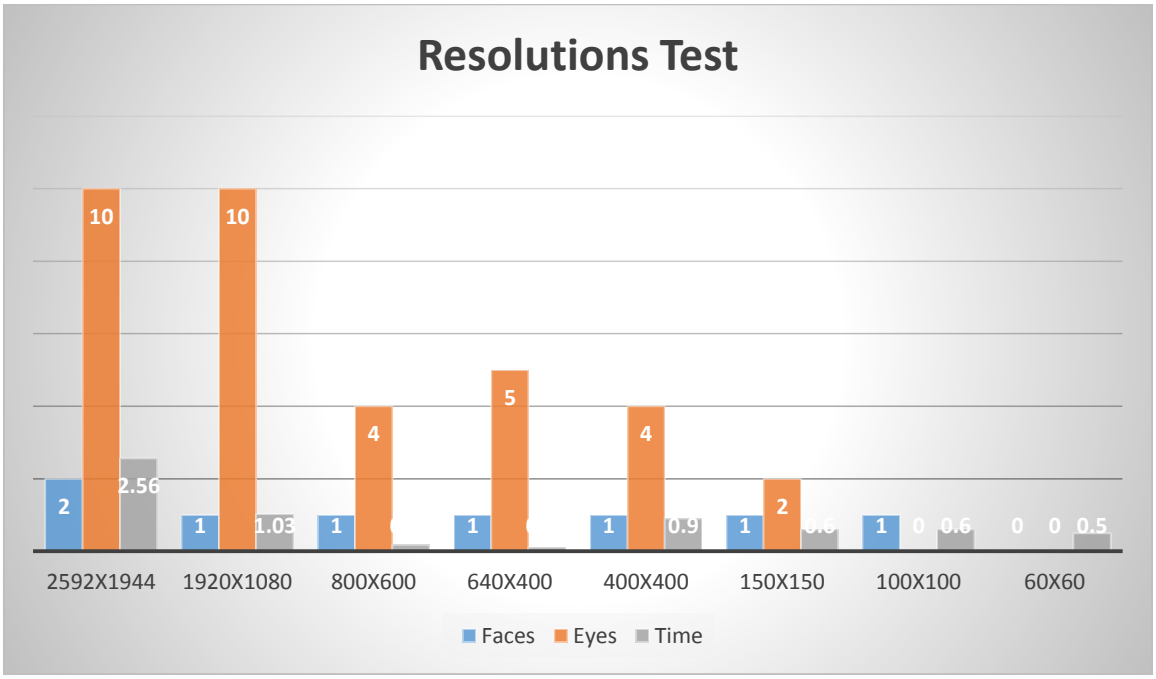

Fig 11. Resolution testing chart

As can be seen from above the optimum resolution is in the 150 to 100 region. It yielded the most accurate results and balanced well with performance. As we can see the false positives with higher resolutions but dramatically decreased whereas facial detection was consistent throughout. It's important to note that the whole image was scanned with the classifiers not a cropped section.

Distance is almost as important as the resolution as it can have similar effects on the accuracy of detection. However keeping in mind that the product systems needs to be placed on a dashboard of a car the distance consistently remains the same. During ad hoc testing experiments were done with different distances and an effective rule of thumb is having the device an arm's length away.

Lighting was a factor that hadn't been considered until after some ad hoc testing and it was noticed that the light levels provided different degrees of effectiveness in discerning features. The eyes seemed to be the feature that suffered the most severe changes when light levels were low. So it was felt that a lighting tests should be carried out to provide a more well-rounded set of test cases. Below are the results from the tests.

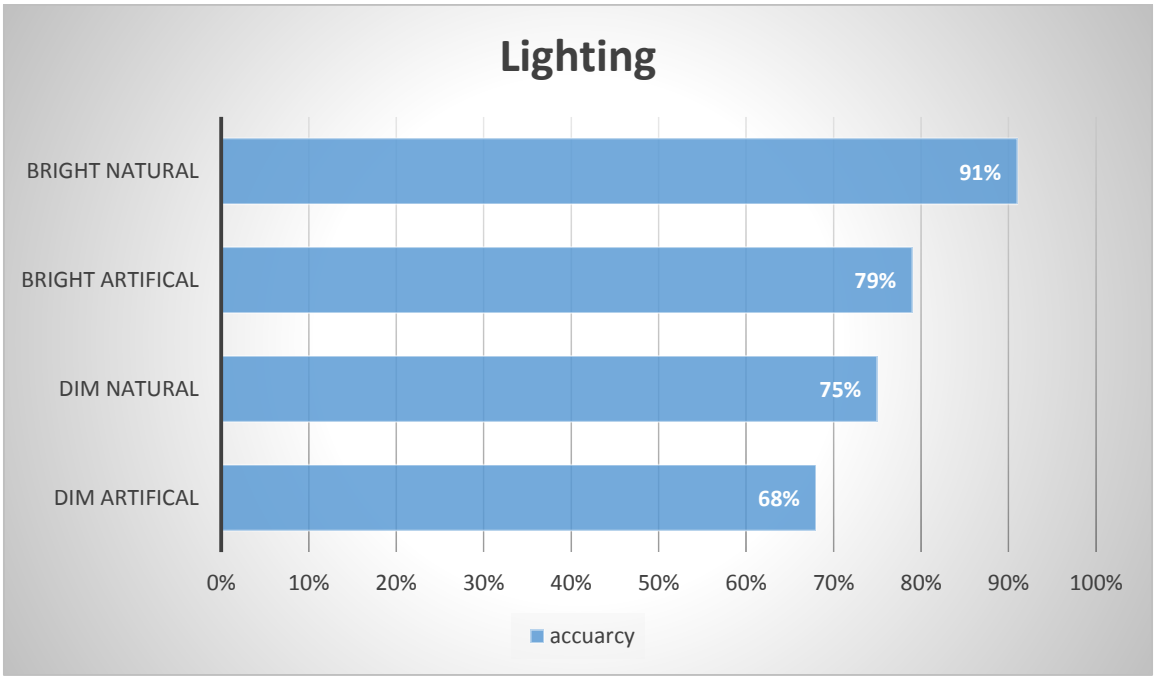

Fig 12. Lighting accuracy chart

The tests was performed by taking a number of 30 second clips and counting the number of missed detections and getting the averages. The tests were only concerning the facial detection as the eye tracking is still unreliable at this phase.

Lighting played a larger factor than anticipated and it's evident that the detections suffer under low light conditions. This would weaken the system in darker settings such as a night drive. One possible solution to this is the Pi Noir camera module which is specifically designed with low light levels in mind.

Challenges encountered

There has been numerous challenges that have been encountered during the development of the project. Most of these haven been overcome and the system has differed from the original design for the better.

The first major barrier was the installation of OpenCv. My biggest error during the whole project was in assuming that because the area was so new and small that all of the literature online would be accurate. This was not the case as it soon became clear that there was a plethora of different install methods for the various libraries I would use over the course of development. Many of these were quite basic and didn't involve or include all the elements that were needed for my project.

The task of getting OpenCv onto the PI seemed relatively straight forward in the beginning and was done without much challenge. It was only when trying to get access to the basic elements like the classifiers that the issues arose. It became apparent quickly that the guides online were wrong or at least didn't provide enough information to make an informed decision as to whether or not it suited the needs of the project. This shortcoming had not turned up in the original research.

It was at this point that more rigorous research and searching was necessary. This took a while and plenty of failed attempts before the correct solution was found. Unfortunately this was time consuming and ultimately delayed development time. Eventually however after a 14 hour build it was installed corrects and all the assets needed were accessible and real development could begin.

Unlike information on the install which was bloated with extraneous and often incorrect material, documentation for the Python distribution of OpenCv was scarce. Even the official documents online didn't provide much inside into the API resulting in a lot of trial and error. For example it wasn't clear that OpenCv handled normal images fine but any picture taken with their functionality were treated as arrays of RGB values. This was something that had to be learn during development.

This manifested later on when writing the code as it moved away from the basic RPI methods of capturing images to the more robust OpenCv features. Then it became an issue again when

capturing multiple images in a loop and displaying them within the loop. However this was overcome by using a continuous capture from the camera which behaved similar to a video stream. With pulling a frame out and analyzing that frame the system didn't have to deal with multiple data types and it was a much more logical approach to take.

Inexperience with the Raspberry Pi and Python provided a steep learning curve but it was a somewhat anticipated obstacle compared to the other major ones. It took a lot of extra learning to be able to do basic functionality and coupled with the fact that this project was undertaken with no experience with OpenCv either the workload of new material never diminished and ongoing research was essential right up to the end.

Future Work

In terms of progression beyond the original scope of this project there is a huge potential for future work. What I would focus on first however is:

Eye Tracking: Given more time eye tracking is an important feature that should receive more attention. The areas in particular would be its accuracy, consistency and position. Gaining the position of the eye itself would be a huge benefit as instead of checking the level of the lid and so on the pupil would be a more valuable asset to analyze. An acceptable operating area for the pupil could be defined and measured. So that if the view of the eye has left this area for an extended period of time a more precise alert could be provided.

Enhanced Testing: Because of the length of the development lifecycle for this project, focus shifted from testing to finishing the basic eye tracking feature. This meant that the test that were performed were usually I an ad hoc manner meaning testing as development. The overall accuracy of the system would have been enhanced by a more structured and rich suite of testing. Also user testing had it been available would have provided invaluable insights in an 'in the wild' setting

Robust Alerts: An improved alert system would enhance the versatility of the application. 'Soft' alerts could be produced for minor offences like a glance in the wrong direction. This alert could be in the form of flashing L.E.D to gain the attention of the driver like previously theorized. After which an increasingly urgent alert with sound. This would possibly be in response to a drivers repeated offences such as persistent distraction. Finally if the system calculates that the driver is in danger of experiencing a serious lapse of attention then a combination of flashing lights and repeating alarm could be sounded until the driver performs an action alleviate or improve their condition.

More complex features: Another aspect that would improve the systems accuracy and precision would be the addition of more complex facial features possible eye brows and the mouth like briefly discussed above. The implementation of this would help distinguish between emotions and provide a better calculation of when the driver is experiencing exhaustion from more clues in their face and eyes.

Accompanying app: Possibly out of reach for a project of this scope, an accompanying phone application could be developed to work with the system. This app could provide feedback to the driver with statistics such as number of attention lapses, frequency of lapses and also the areas where these infractions occurred the most. This would present them with usable information on where they lack focus on the road. It would also give them an opportunity to alter their driving habits to accommodate for a more vigilant journey.

Conclusion

Overall the system is quite accurate when it comes to facial detection. This of course is largely down to choosing the optimum resolution and setup. The program runs at roughly 5-6 frames per second which is a competent rate for device of its limitations.

Eye tracking seems to be the weakest of all of the areas surrounding the project. It was difficult to strike the balance between accuracy and performance in relation to detecting the eyes. If the image was too small the feature wouldn't be detected by the classifier. If it was too large the performance suffers as a result also it runs the risk of increasing the number of false positive within the image.

These issues could be resolved with more processing power which would be available with the Raspberry Pi 2. The RPI 2 was released during the development phase of this project. The system would perform faster and eye detection could be improved with the extra power.

In reflection of the project overall it was a success. All of the minimum requirements have been met and it performs as advertised. Unfortunately the eye tracking component suffers from inaccuracy but could be improved given more time. The most difficult element of the project was effectively learning 3 new technologies or areas of study.

In doing a project in a similar area it would be advised to possibly only take on one new technology as opposed to multiple as the experience can be slightly overwhelming. Being previously unfamiliar with the Raspberry Pi, Python and OpenCv meant that the initial research was enormous and development was slow. It took quite a while to build up confidence in the integrity of the code and learning the system. Having no familiar territory in the project had the effect of demotivation as the research and learning material was always quite large. However there is a great reward felt after achieving what you set out to do after a long development cycle.

Appendices

Source Code

Main program roadAware.py

```
#!/usr/bin/env Python

# All of the imports used

import cv2

import math

import time

import pygame

from picamera.array import PiRGBArray

from picamera import PiCamera

import numpy as np


#Definition of the classifiers and their locations

face_cascade =
cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-
2.4.9/data/haarcascades/haarcascade_frontalface_default.xml")

eye_cascade = cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-
2.4.9/data/haarcascades/haarcascade_mcs_righteye.xml")


#These are used for playing the sound

pygame.mixer.init()

pygame.mixer.music.load("/home/pi/Downloads/beep.mp3")


#Initialising the camera, defining resolution and framerate

camera = PiCamera()

camera.resolution = (150, 150)

camera.framerate = 24

rawCapture = PiRGBArray(camera, size=(150, 150))

time.sleep(0)


#The frames variable is the counter and the maxFrames is the
threshold before the alert goes off

frames = 0

maxFrames = 10



#reduces the size of the image for faster processing

def shrinkImg(img):
```

```
        dst = cv2.resize(img, None, fx=0.80, fy=0.80, interpolation =
cv2.INTER_LINEAR)
        return dst
#enlarges image for clearer display
def enlargeImg(img):
        dst = cv2.resize(img, None, fx=2.00, fy=2.00, interpolation =
cv2.INTER_LINEAR)
        return dst


#Function for facial detection and eye tracking
def detectFace(image):
        #iterate through and draw a bounding box over face
        for (x, y, w, h) in faces:
                cv2.rectangle(image, (x, y), (x+w, y+w), (255, 0, 0),
2)
                roi_color = image[y:y+h, x:x+w]


                #take the face and make it the region of interest
                roi = image[y:y+h, x:x+w]
                #crop the region of interest to the right eye
                #limits search window of the right eye
                cropped = roi[10:80, 30:100]
                cv2.imshow("Eye", cropped)


                #Detect eye in the new cropped window
                eye = eye_cascade.detectMultiScale(cropped, 1.3,
5)
                for (x, y, w, h) in eye:
                        #iterate through and draw bounding box
                        cv2.rectangle(cropped, (x, y), (x+w, y+w), (0,
0, 155), 2)
                        roi_color = cropped[y:y+h, x:x+w]
                        #if eye is found reset the count and return it
                        if(eye is not None and len(eye) > 0):
                                frames = 0

        return image


# This loops takes a video feed and pulls out each frame and
analyzes it
```

```
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):

        image = frame.array
        image = shrinkImg(image)
        #converting the image to grey scale to apply classifiers
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        #Apply the face classifier to the image
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)



        #If there is an face detected reset the frame counter
        if(faces is not None and len(faces) > 0):
                #frames = 0
                frames += frames
                detectFace(image)

        #Else add to frames, if the count reaches the max count
then sound alert
else:
        frames += 1
        if (frames == maxFrames):
                pygame.mixer.music.play()
                frames = 0
        print frames

resImage = enlargeImg(image)
#display the image (for testing and demonstration)
cv2.imshow("image", resImage)

#waiting to detect key press
key = cv2.waitKey(1) & 0xFF



rawCapture.truncate(0)

#Loop plays until the 'q' key is pressed
if key == ord("q"):
        break
```

Original Face Detection faceDetection.py

```
import numpy as np

import cv2


def resizeImg(img):

        dst = cv2.resize(img, None, fx=0.30, fy=0.30, interpolation =
cv2.INTER_LINEAR)

        return dst


face_cascade =
cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-
2.4.9/data/haarcascades/haarcascade_frontalface_default.xml")

eye_cascade = cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-
2.4.9/data/haarcascades/haarcascade_mcs_lefteye.xml")

mouth_cascade =
cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-
2.4.9/data/haarcascades/haarcascade_mcs_mouth.xml")

nose_cascade =
cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-
2.4.9/data/haarcascades/haarcascade_mcs_nose.xml")

#these classifiers are needed for the program to run


path = "/home/pi/EyeInThePi/"

img = cv2.imread(path + "greyScale.jpg")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


faces = face_cascade.detectMultiScale(gray, 1.3, 5)

#eyes = eye_cascade.detectMultiScale(gray)

for (x, y, w, h) in faces:

        cv2.rectangle(img, (x, y), (x+w, y+w), (0, 255, 0), 2)
```

```
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
eyes = eye_cascade.detectMultiScale(roi_gray)

for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (255, 255,
0), 2)
        mouth = mouth_cascade.detectMultiScale(roi_gray)
        for (mx, my, mw, mh) in mouth:
                cv2.rectangle(roi_color, (mx, my), (mx+mw,
my+mh), (0, 255, 255), 2)

        nose = nose_cascade.detectMultiScale(roi_gray)
        for (nx, ny, nw, nh) in nose:
                cv2.rectangle(roi_color, (nx, ny), (nx+nw, ny+nh),
(255, 255, 0), 2)
cv2.imwrite("leftEye.png", img)
cv2.imshow("img", img)
cv2.imshow("gray", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

First iteration initScript.py

```
import os

import cv2

import math

import time

import picamera


def resizeImg(img):

        dst = cv2.resize(img, None, fx=0.30, fy=0.30, interpolation = cv2.INTER_LINEAR)

        return dst
```

```
#display images

cv2.imshow("img", img)
cv2.imshow("greyScale", grey)
cv2.waitKey(0)
```

Testing a loop loop.py

```
import cv2
import math
import time
import picamera

img = "foo.jpg"
with picamera.PiCamera() as camera:
        camera.resolution = (400, 400)
        camera.start_preview()
        time.sleep(2)
        for i in range (0, 10):
                camera.capture(img)
                img = str(i) + (img)

#cv2.imshow("img", img)
```

Testing the region or interest ROI.py

```
#taking a picture using the pi camera
with picamera.PiCamera() as camera:
        camera.resolution = (640, 400)
        camera.start_preview()

        time.sleep(10)
        camera.capture('foo.jpg')

#os.system("raspistill -vf -o test.jpg")

path = "/home/pi/EyeInThePi/"
```

```
#loading the image
img = cv2.imread(path + "foo.jpg")
grey = cv2.imread(path + "foo.jpg", 0) #the 0 is for greyscale


#resize image
img = resizeImg(img)
grey = resizeImg(grey)

cv2.imwrite(path + "greyScale.jpg", grey)
cv2.imwrite(path + "imagecolor.jpg", img)
```

```
#!/usr/bin/env Python
import numpy as np
import cv2
import cv2.cv
import math
import time
import picamera
from PIL import Image
```

```
def resizeImg(img):
        dst = cv2.resize(img, None, fx=1.0, fy=1.0, interpolation = cv2.INTER_LINEAR)
        return dst

face_cascade = cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-2.4.9/data/haarcascades/haarcascade_frontalface_default.xml")
eye_cascade = cv2.CascadeClassifier("/home/pi/EyeInThePi/opencv-2.4.9/data/haarcascades/haarcascade_mcs_lefteye.xml")

path = "/home/pi/EyeInThePi/"
img = cv2.imread(path + "foo.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = resizeImg(gray)
face = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x, y, w, h) in face:
        cv2.rectangle(img, (x, y), (x+w, y+w), (0, 255, 155), 2)
        roi_color = img[y:y+h, x:x+w]
        roi = img[y:y+h, x:x+w]

cropped = roi[10:120, 20:90]
eye = eye_cascade.detectMultiScale(cropped, 1.3, 5)
for (x, y, w, h) in eye:
        cv2.rectangle(cropped, (x, y), (x+w, y+w), (0, 0, 155), 2)
        roi_cropped = cropped[y:y+h, x:x+w]

cv2.imshow("blownUpImg.jpg", roi)
#cv2.imshow("wreked.jpg", img)
```

```
cv2.imshow("cropped",cropped)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Testing capturing in a loop capAndDis.py

```
import numpy as np
import cv2
import math
import time
import picamera

name = "w.png"
path = "/home/pi/EyeInThePi/pictures/"
with picamera.PiCamera() as camera:
        camera.resolution = (400, 400)
        camera.start_preview()
        time.sleep(0.5)
        for i in range (0, 11):
                camera.capture(path + name)
                name = str(i) + (name)

                #img = camera.capture('pictures/w.png')

img = cv2.imread(path + name)
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.imshow("img", img)
cv2.imshow("gray", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

References

Raspberry Pi Foundation. (2014). “What is a Raspberry Pi”. [Online] available: http://www.raspberrypi.org/help/what-is-a-raspberry-pi/ [accessed 3/11/2014]

Arduino.cc. (2014) “Arduino Uno” [Online] available: http://arduino.cc/en/Main/ArduinoBoardUno [accessed 5/11/2014]

Membrey P, Hows D. (2013). Learn Raspberry Pi with Linux. Place of Publication. Apress

Sriram S, Illuri B. (2014) Real-time Face Detection and Tracking Using Haar Classifier. International Journal of Computer Applications, Volume 104 – Number 10

Timm F, Barth E. (2012) Accurate Eye Center Localization By Means of Gradients. [Online] available at: http://www.inb.uni-luebeck.de/publikationen/pdfs/TiBa11b.pdf [accessed 15/04/15]

Schapire, R. (2013). Explaining adaboost. Department of Computer Science [Online], Available: https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf [accessed 20/ 11/ 2014]

Viola, P. Jones, M. (2001) Rapid Object Detection using a Boosted Cascade of Simple Features. COMPUTER VISION AND PATTERN RECOGNITION 2001[Online] available: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf [accessed 5/11/2014]

Emer, E. (2014). Boosting (AdaBoosting Algorithm) [Online], available: http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Eric-Boosting304FinalRpdf.pdf [accessed 20/11/2014]

OpenCv.org (2014) OpenCv – Computer Vision [Online] available: opencv.org [accessed 15/11/2014]

Tripathy, R, Daschoudhury, R.N. (2013) Real-time Face Detection and Tracking Using Haar Classifier on SoC. Department of Electronics and instrumentation Engineering. [Online] Volume 3 Number 2, Available: www.ijecse.org [accessed 15/11/2014]

Kun, J. (2011). EigenFaces, for Facial Recognition. [Online] available: www.jeremykun.com/2011/07/27/eigenfaces/ [accessed 3/12/14]

Chapple, C. (2014) Imagination gets into micro-computing with Raspberry Pi rival. [Online] available: http://www.develop-online.net/news/imagination-gets-into-micro-computing-with-raspberry-pi-rival/0200945 [Accessed 29/ 11/ 2014]

OpenBR. (2014). Open Source Biometric Recognition [Online] available: http://openbiometrics.org/ [accessed 3/12/2014]

LibFace. (2010). Libface – Face Recognition Library. [Online] available: http://libface.sourceforge.net/file/Home.html [accessed 3/12/2014]

Fig 1 CodeProject (2014). Haar Features [Online] available: http://www.codeproject.com/KB/graphics/441226/haar-working.png [accessed 15/11/2014]

Fig 2 Mathworks, (2015). Cascade Object Detector Scale Factor [Online] available at: http://www.mathworks.com/help/vision/ref/cascade_object_detector_scale_factor.png [accessed 20/ 11/ 2014]

Fig 7 RaspberryPi Spy, (2013). Raspberry Pi powered speaker [Online] available: http://www.raspberrypi-spy.co.uk/wp-content/uploads/2013/06/raspberry_pi_powered_speaker_03.jpg [accessed 16/11/2014]

Fig 8 Indestructibles, (2014) L.E.D [Online] available: http://cdn.instructables.com/FJ6/X9Z5/H8CVKODY/FJ6X9Z5H8CVKODY.MEDIUM.jpg [accessed 16/11/2014]

Fig 11 McKendry M, (2015). RoadAware Prototype [Photograph] (From own private collection)