Progress Report

Game development project

sean khanna – Q11279516

Level 6

Contents

[Introduction 2](#_Toc2345543)

[Background 3](#_Toc2345544)

[Engine or Language to use 3](#_Toc2345545)

[Definition 4](#_Toc2345546)

[Eigen faces 4](#_Toc2345547)

[*Huu-Tuan Nguyen, 2014.* 5](#_Toc2345548)

[Steps 5](#_Toc2345549)

[*sdenton4, 2015* 7](#_Toc2345550)

[Fisherfaces 8](#_Toc2345551)

[LDA 8](#_Toc2345552)

[Local Binary patterns histograms 9](#_Toc2345553)

[Histograms of Oriented Gradients 9](#_Toc2345554)

[Step by Step Guide 10](#_Toc2345555)

[Haar cascades 13](#_Toc2345556)

[Filters 15](#_Toc2345557)

[Gaussian 16](#_Toc2345558)

[Gaussian Blur 16](#_Toc2345559)

[Morphological Transformations 16](#_Toc2345560)

[Kernel 16](#_Toc2345561)

[Dilation 16](#_Toc2345562)

[Erosion 17](#_Toc2345563)

[Initial Design 19](#_Toc2345564)

[Project Plan 19](#_Toc2345565)

[Source Control 21](#_Toc2345566)

[References 22](#_Toc2345567)

[Appendix 25](#_Toc2345568)

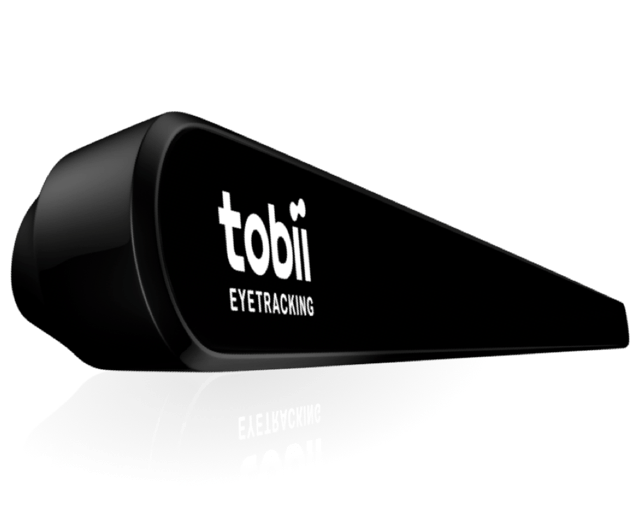
[Appendix A - Timeboxes 25](#_Toc2345569)

[Appendix B – Project Gantt Chart 27](#_Toc2345570)

[27](#_Toc2345571)

[Appendix C – User Stories 27](#_Toc2345572)

# Introduction

This project started out as a plan to replace face recognition tools were you are required to buy extra hardware for it to work. The idea was that the tool was going to replace other existing tools and make it easier for the gaming community to create games which used a pre-owned camera to control parts of the game. There was already some competition out there such as tobii (2019). tobii is one of the lead eye tracking software out now, which uses their own cameras to track eyes. When creating tools or products using their products you are required to buy one of their cameras to use when creating any tool because, it aids in tracking where the eyes are with their three-camera system. Therefore, for this project the idea was to remove the part of having to buy hardware and use what people already owned, such as web cameras or laptop cameras.

tobii, 2009

After further research I found that there are quite a few other tools out, which allows you to use any camera and create your own programs with additional features. Some of these tools are very thorough and can use over 10 different forms of AI algorithms such as neural networks, Eigenfaces, Fisher faces and more which will be talked about later. Consequently, the product will now be a program that uses one of these tools, with a technical demo demonstrating what is going on and the program recognising certain events like: winking, the mouth opening and more.

# Background

There are many games on the market that have the capability of eye tracking unfortunately, my research came up short when I couldn’t find any released games or tools which use a simple web camera, all my findings lead to games that were created with the use of tobii (2019). There is a long list of games that use tobii, for example, Assassin’s Creed Syndicate and SOMA utilise the tobii camera for aiming using your eyes.

GameCrate, 2016

As said before there is no market for any games that use a person’s web camera to control the game therefore, this project will aim to fill that void.

## Engine or Language to use

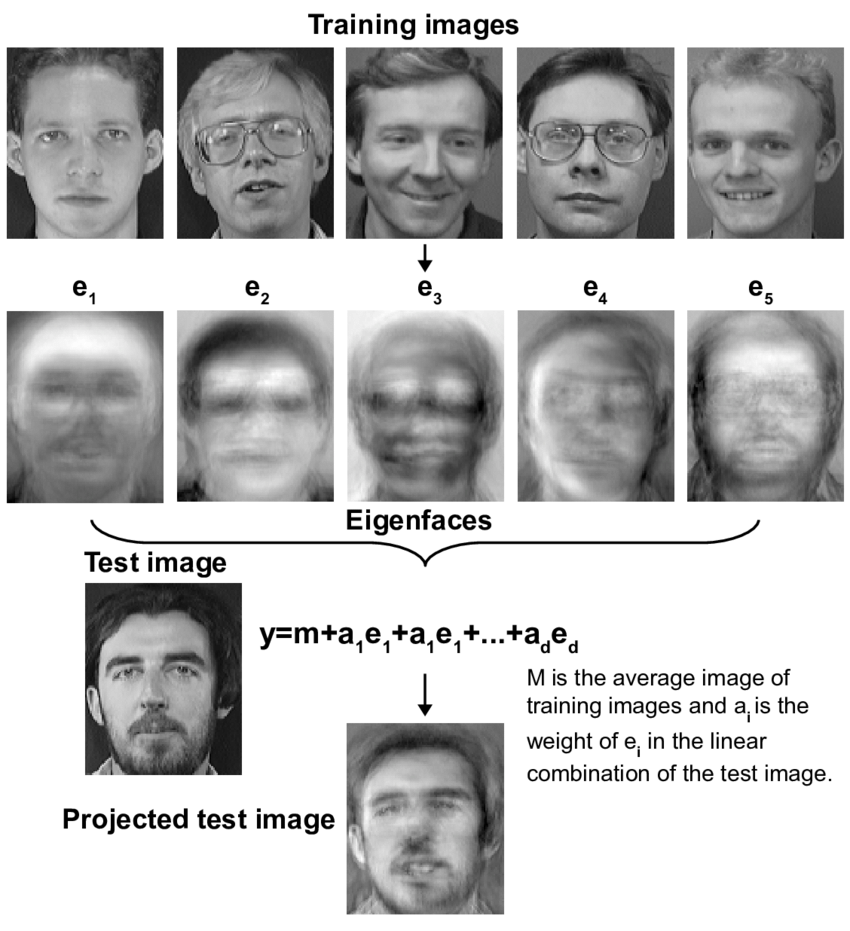
After thorough research the program is going to be written in C++ because Unity only offers two official computer vision libraries that could aid the project the first was VisionLib. VisionLib is mostly used for Augmented Reality and helps more with model tracking, in layman terms you feed it a 3D model and you can use it with AR glasses or similar, to assist tracking. The second is OpenCV, however using it through Unity meant they must put a price tag upon it, of $65. Furthermore, I looked into using Unreal Engine, unfortunately, my research came up short and could not find any libraries to use with unreal except using OpenCV and using that within my code. C# also has a library which uses OpenCV, this is called EmguCV. This tool is cross platform, and wraps OpenCV in a .NET, allowing any .NET application, that are compatible, to use it, for example C#, VB, VC++ and any IDE that can compile it. This however, is time consuming for the project having to learn a new library and how to use a .NET wrap version of OpenCV. In the end, it came down to using OpenCV but, this can be used within three languages, Java, Python and C++. Unfortunately, my knowledge in Java and Python is little or next to none therefore, I have decided to write my program in CMD in C++.

# Definition

The language chosen for this project is C++ using the OpenCV libraries, it has a wide range of functionality and the OpenCV website fully documents all of the functions, filters and other areas of its library. OpenCV is a very robust tool which uses many algorithms to help with face detection, in different ways. There are 4 main types of face detection that OpenCV offer: Eigen faces, Fisher faces, Local Binary patterns histograms and Haar cascades.

## Eigen faces

Eigen faces began out as the principal components of a distribution of faces and the idea was first created by Sirovich and Kirby in 1987. It uses a variety of face images to convert and relate individual faces in a big picture fashion.



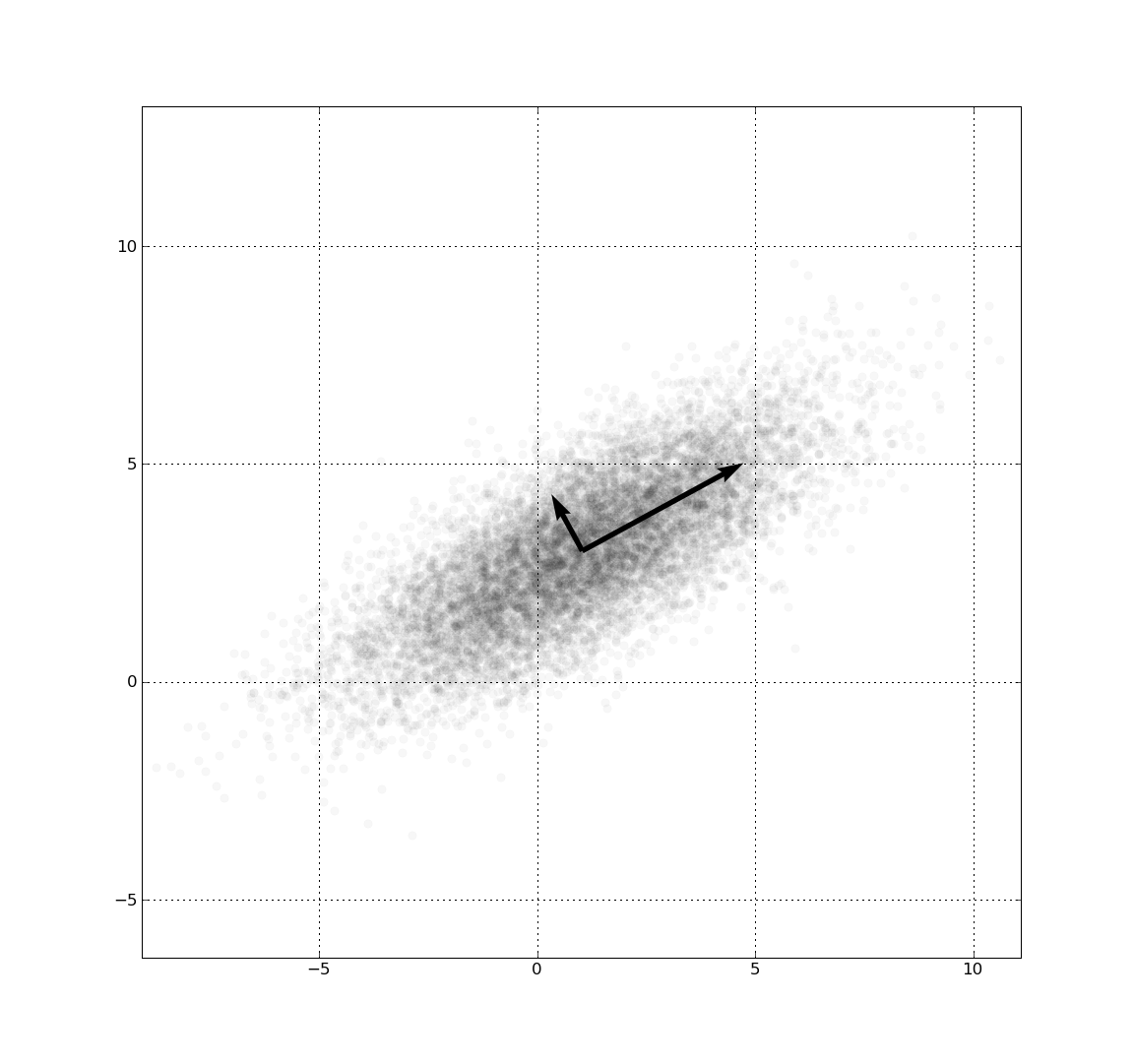
### *Huu-Tuan Nguyen, 2014.*

### Steps

Eigen faces takes in a image and inputs each pixel into a vector, in the end once you have entered all the images you wanted into vectors you then end up with multiple sets of 2 dimensional date in one vector, these are called Eigen Vectors. In turn, each of these vectors put together can be treated as a matrix. With this matrix we apply PCA (see below) to it by, subtracting the mean, the mean here would be the average vector of all the images. Next you compute the covariance matrix of each eigenvector by finding the mean between the average eigenvector created before and the original eigenvector. Currently, we have a new set of eigenvectors that we can put back into an image and it will show how each image deviates from the average picture. Now, we can use the a different image, preferably one that is similar to one that was calculated into an eigenvector for best results, and do the dot product on that with each of the eigenvectors created, add the mean image and eventually, with the more dot products created and added, you will eventually get the an image which looks very similar to the input image, and finally, we can then use this to do face similarities.

#### PCA

Principal component analysis is a way of converting data that is possibly related into a lineal unrelated set of data which is called principal components, it allows as explained by Matt Brems(2017) “…we can drop the “least important” variables while still retaining the most valuable parts of all of the variables!” . Matt further talks about how we can use this data to be able to input new variables into the graph.



## *sdenton4, 2015*

## Fisherfaces

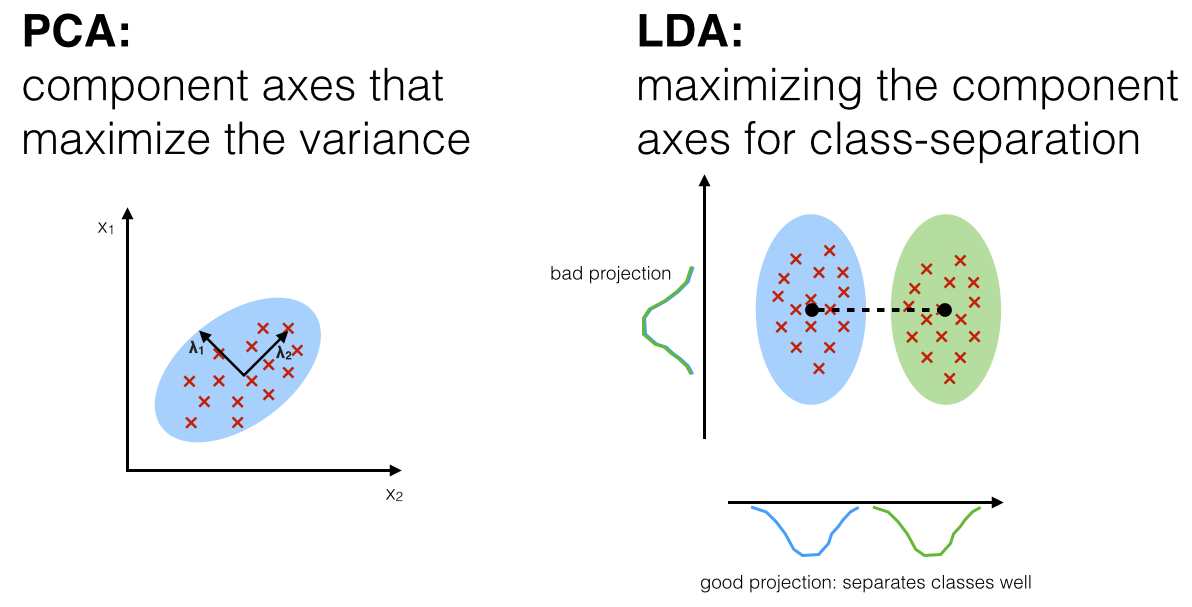
Fisherfaces is very similar to eigenfaces in how it tries to compute it however instead of using PCA it uses something called LDA (see below). This tool is more beneficial when there are more variations of illumination and facial expressions. With PCA you only get a modest model of the image whereas LDA and Fisherfaces tries to maximise the average distance of classes while lowering the variance within the class. It attempts to get better face models

that are more useful in discrimination.

### LDA

* Used for dimensionality reduction
* Multi-class classification
* Both PCA and LDA are lineal transformations

Mustamin Anggo and La Arapu, 2018

* Maximises the component axes for class separation

Sebastian Raschka, 2014

## Local Binary patterns histograms

### Histograms of Oriented Gradients

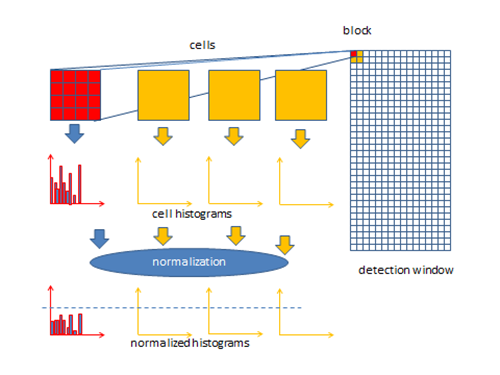
Histograms of oriented gradients is a type of descriptor that identifies objects in computer vision and image processing. This method counts occurrences of gradient orientation in localized portions of an image or region of interest (Intel, 2018).

#### Requirements of HOG

* 1D centered derivative mask [-1, 0, +1]
* Detection window size is 64x128
* Cell size is 8x8
* Block size is 16x16 (2x2 cells)

Intel’s simple breakdown process is as follows:

1. Break up the picture 8x8 pixels across the whole detection window and for each cell calculate a histogram of gradient directions or edge orientations for each pixel in the cell.
2. Calculate each cell into angular bins according to the gradient orientation.
3. In each pixel in each cell within in the bin, a weighted gradient contributes towards the angular bin.
4. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping normalization of histograms.
5. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.



When combining a local binary pattern with histograms of oriented gradients, as explained by Kelvin Salton (2017), “it improves detection performance considerably”. He further explains, how you can turn the face images into simple data vector. The local binary pattern acts as a visual descriptor and can be used for face recognition tasks.

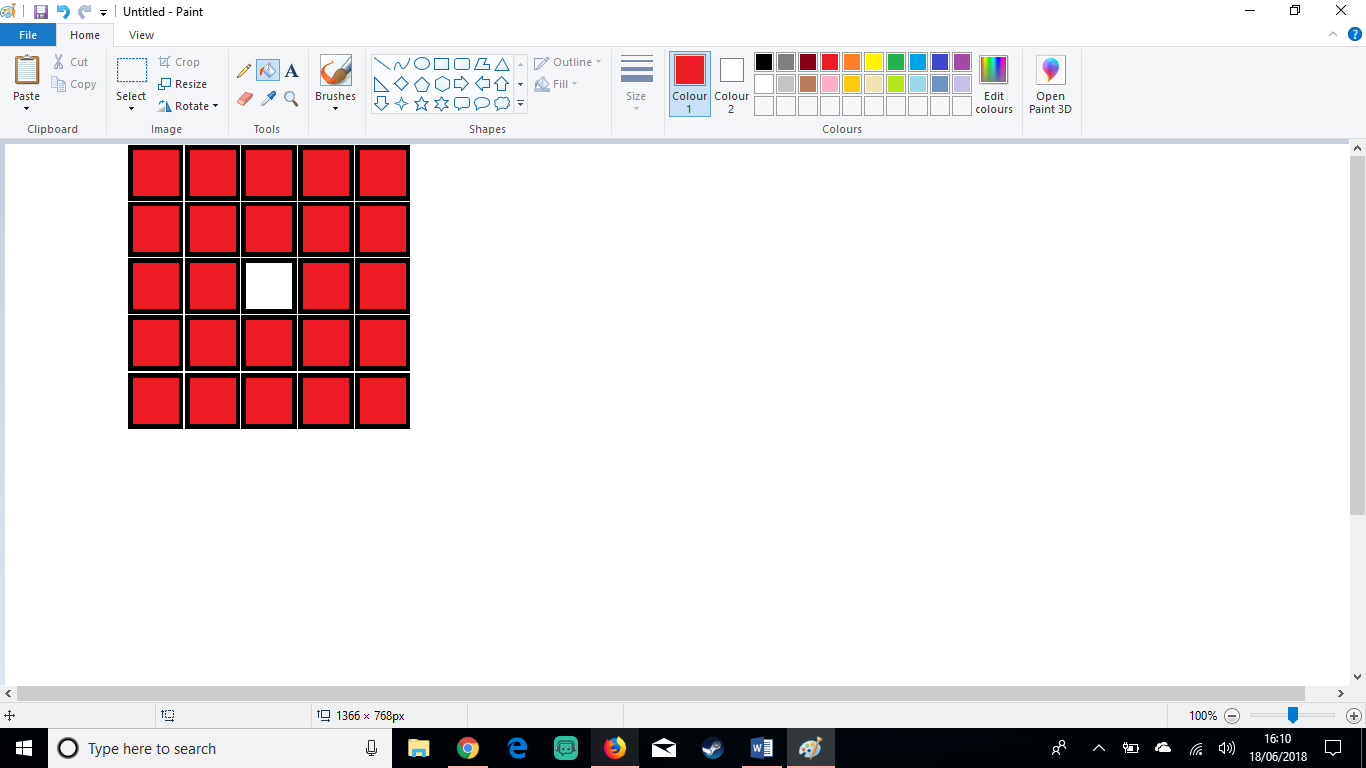
### Step by Step Guide

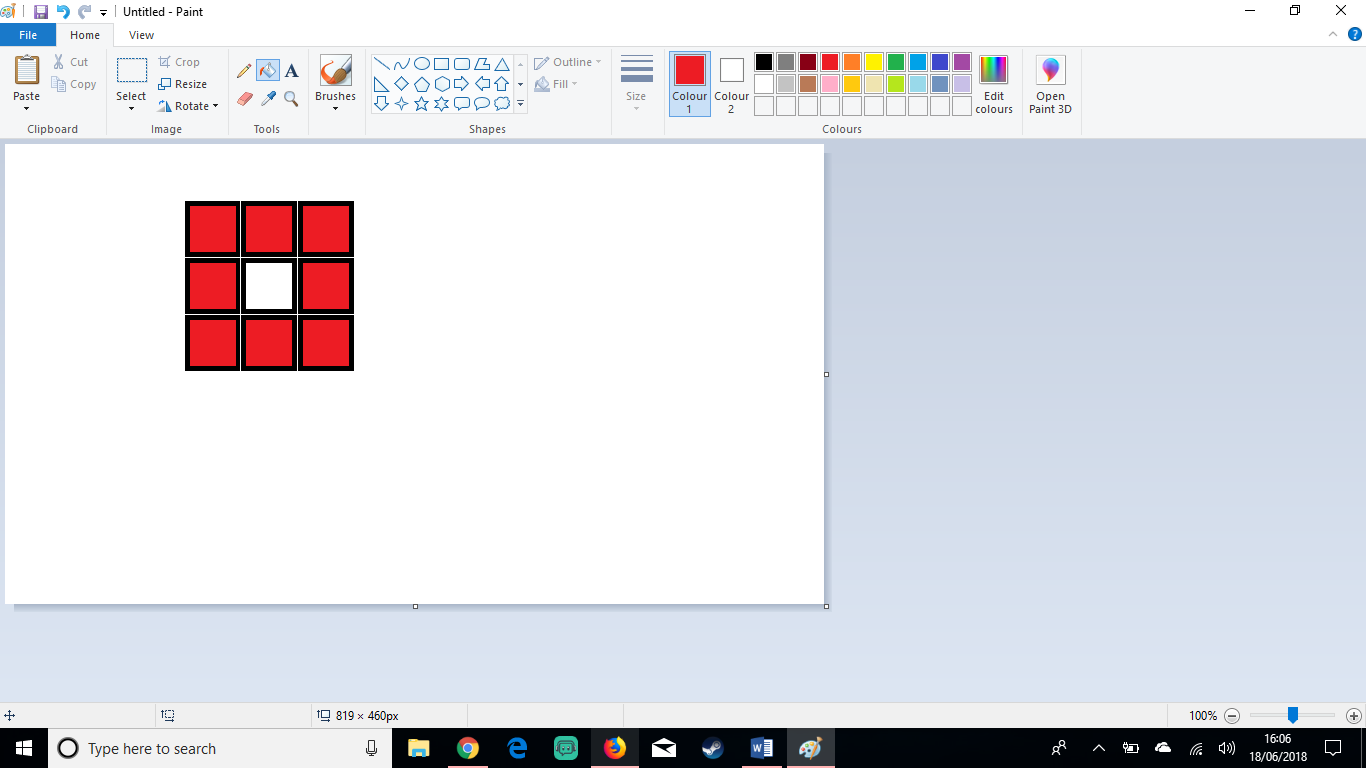
When using LBPH you use 4 parameters: Radius, Neighbours, Grid X and Grid Y. these 4 steps are used to create an algorithm for LBPH.

#### Parameters

##### Radius

We take the centre pixel of the block or region we are looking at. The radius (in red) is the area around that pixel, this builds the circular local binary pattern. Usually the radius is set to one.

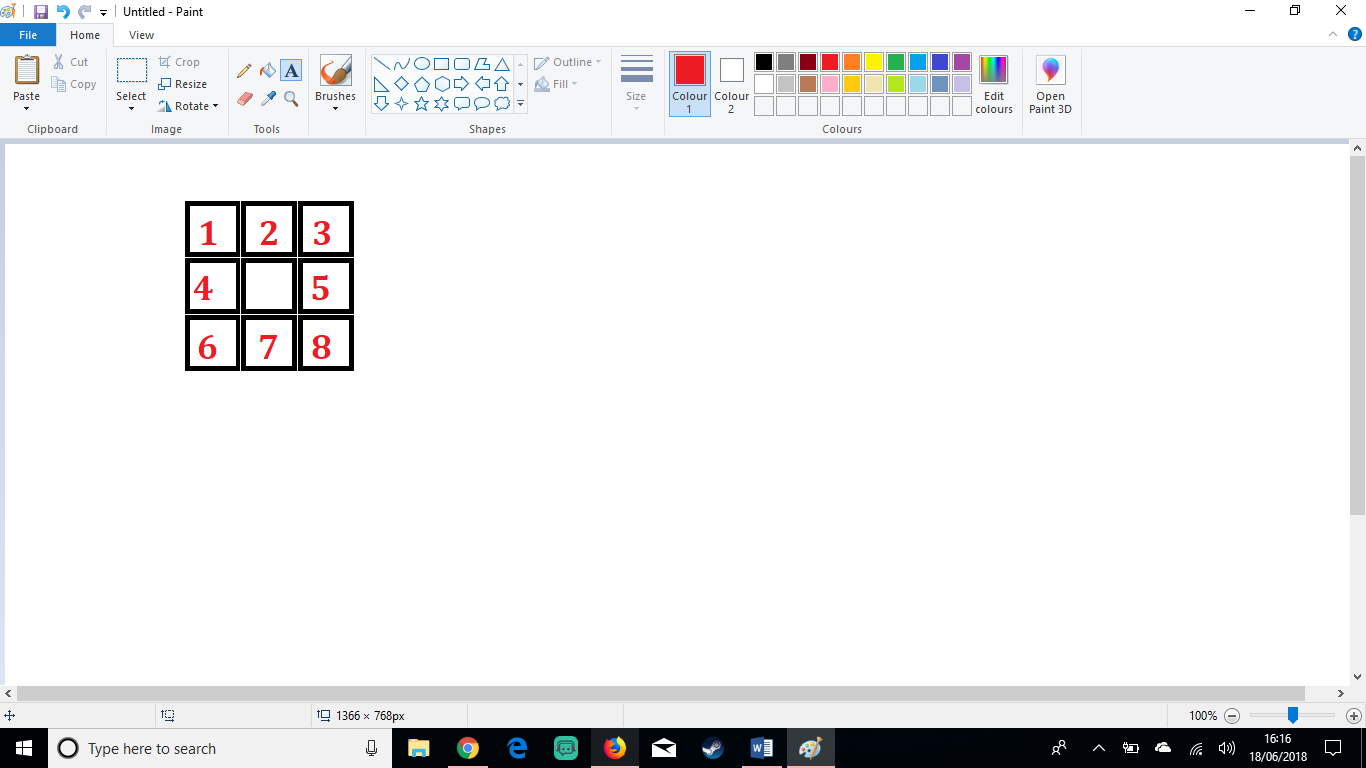




Radius = 1 Radius = 2

##### Neighbours

These are total number of sample points around the centre pixel for example, if use a radius of 1 then there will be 8 sample points. However, there is one downfall, if we use a bigger radius, this leads to an increased amount of sample points to test, increasing the computational cost to calculate each pixel.



##### Grid X

This is the number of blocks or cells in a horizontal direction. The more you have the finer the grid and the higher the dimensionality of the resulting feature vector. Normally has a value of 8.

##### Grid Y

Very similar to Grid X except in a vertical direction.

#### Employing the Local Binary Pattern

The first step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. The algorithm then uses the radius and neighbour parameters, to create a “sliding window”.

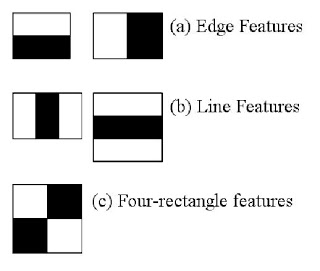
We take the original image, remove all colour as to leave it greyscale. From this we can then extract any part of the new image as a window of 3x3 pixels. This 3x3 matrix can be displayed as value of intensity (0-255). We use the centre point of the matrix as the threshold value.

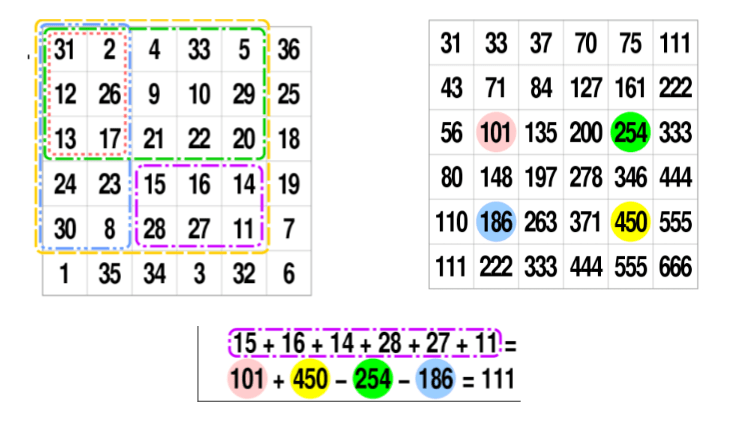
We then turn these new values into a binary number. Taking a look at the neighbours of the centre pixel and comparing them to the threshold will leave us with two values, one or zero. These are worked out as follows: if the neighbour value is equal or higher than the threshold then it becomes a one and if it is lower, then it is zero.

Now that we only have binary values we can then link these into a binary value e.g. 10101110. With this new binary value, we then turn it into a decimal, using the previous example the decimal value is 174. With this new decimal value, you set the centre pixels value to it.

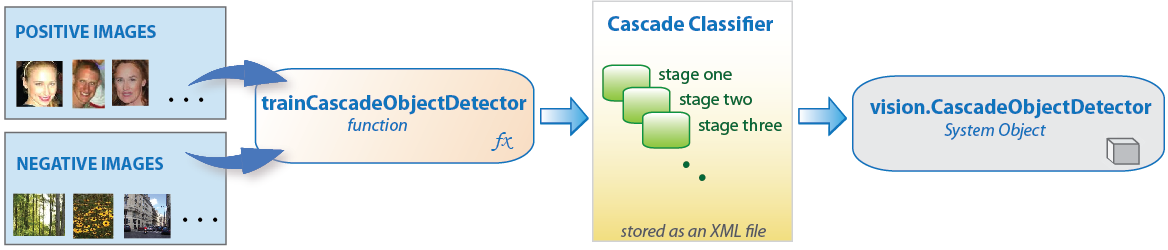
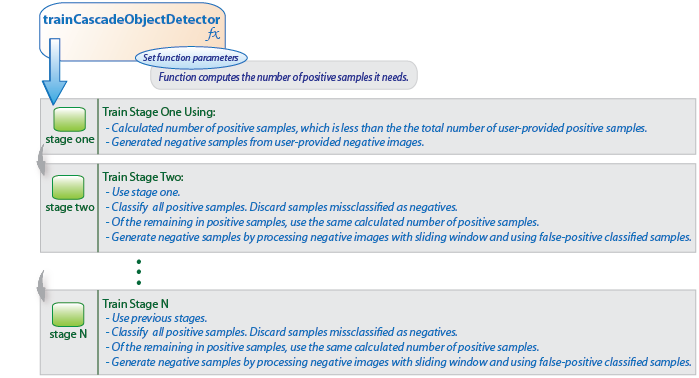
## Haar cascades

Haar Cascades is a machine learning detection algorithm which recognises certain objects within an image built by feeding it data of other similar objects.

* A cascade function is trained through enter lots of positive and negative images
* Four steps to the algorithm:
  + Assemble Haar features
    - Within a detection window in a specific region and all adjacent regions, it adds up all the pixel intensities of each region and then calculates the difference between them
    - There are three type of features possible edge features, line features and four–rectangle features. 
    - Create Integral Images
    - Take the image as a data structure or “Summed-area table” (see below), it is used to swiftly and inexpensively generate a sum of values in a division of the grid



* + Once created it needs to go through a process called adaboost training
    - Simply it goes through all the haar features and selects only the features which improve the analytical capability of the algorithm and like the other algorithms reduces dimensionality.
  + Finally, you enter all the positive and negative images into a cascade classifier which trains it and then stores it in an XML file



Berger, W., 2018.

## Filters

There are certain parts to OpenCV that allow me to do specific tasks such as reduce noise. To do this I will need to use some filters to help smooth out my image and make the information on screen more readable to the program, allowing for better results when tracking.

## Gaussian

Probability theory is a part of mathematics that deals with random events within quantities. Within probability theory the “normal distribution” or Gaussian is when there is a “random” anomaly. However, a guy called Galileo Galilei, questioned it and found that this phenomenon was symmetrical around a central value.

### Gaussian Blur

Gaussian blur is a way of processing an image and blurring out any part of an image, usually to reduce the noise.



OpenCV, 2019

## Morphological Transformations

These transformations usually are applied to black and white images only.

### Kernel

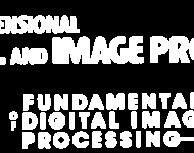
Both Dilation and Erosion use a kernel which will go over the whole image and can be any size or shape, usually a square or circle.

### Dilation

As the kernel traverses over the image it will add a layer of pixels to the inner and outer boundaries of regions making them foreground pixels.



Original image

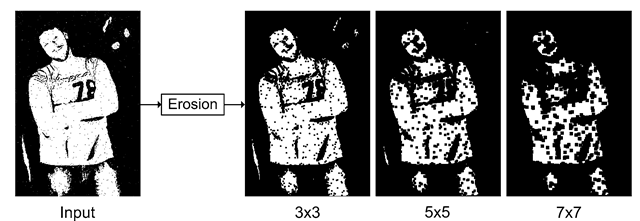


Dilated image

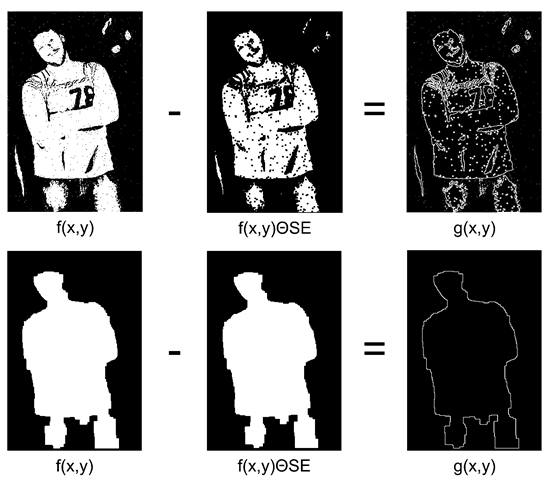
### Erosion

Erosion has the opposite effect and removes the foreground pixels from both regions.

These can be used within images to reduce things such as reflections or bright lights in the background. While using erosion on a black and white image you can begin to define outlines of certain things see image below.



What-when-how, Unknown date.

Later we can use this to start doing edge detection called ‘boundary detection’ for example take the previous image and minus it from the input and we have our boundaries.

What-when-how, Unknown date.

# Initial Design

Unfortunately, I was unable to design any flow charts, as my knowledge base for the filters was not sufficient enough. However, through a bit more research and testing I will be able to figure out what filters I need to apply to images before any algorithm is applied. The knowledge base known already gives enough insight and ground to begin that the rest will fall into place.

# Project Plan

As Ben Aston (2017) says “Project management is important because it ensures what is being delivered, is right”. Therefore, as part of my rationale, I decided to write up user stories to help me build my project plan (see [Appendix C](#_Appendix_C_–)). User stories are a phenomenal tool which helps define what your product is going to achieve by stating what the user, programmer or even the client might want. One of the advantages of user stories explained by Kamlesh Ravlani (2017) is that “User Stories allow for easy addition and removal of features from the Product.” and with each user story completed the overall value of the product then rises.

From these user stories now created, we can begin defining the tasks and build a work break down structure (see [Appendix A](#_Appendix_A_-)) which displays each task and any sub tasks they may have and how long roughly, they might take to complete and their dependencies if any. Obviously, there are some limitations which I will further explain in my risk analysis however, all tasks should be achievable.

Once each sub task has been defined we can then put all of them into a Gantt Chart (see [Appendix B](#_Appendix_B_–)) were you can clearly see the critical path and how long I have, to do this project and how much slack time I have.

|  |  |  |  |
| --- | --- | --- | --- |
| Severity | Risk | Solution | Mitigation Steps |
| 2 | Losing work | Use Source Control | * There are many other tools for me to reduce this risk including USB, Google Docs and Dropbox. |
| 3 | Not understanding how the library works | Research | * Look in to how the library works * What algorithms are involved * What other tools it has to offer |
| 1 | Choosing a language | Research | * Find out what engines, languages and libraries are available * Deduce advantages and disadvantages * Based on these results decide on a language |
| 3 | Fall behind in other classes | Manage my time well | * Follow a well-structured Gantt chart allowing for lots of slack |
| 2 | My health | Look after one’s self | * Eat healthy * Exercise * Don’t over work myself * Spread work load over time |
| 1 | Program updates | Don’t update the program | * Take a note of which version of everything I’m using including Visual Studio and OpenCV Library |

## Source Control

One of the major risks involved when creating any program is losing your work. It can happen very easily and might not be anybody’s fault but if people don’t back up their work it can be lost. However, using source control allows you to have a backup were ever you go, as Kemper (p.75, 2012) explains it “No matter what you’re working on, or what you’re using, you always want to keep some kind of backup”. In conclusion to this, GitHub will be used to fully backup this project furthermore, because of the large file size of the OpenCV library, GitLFS is needed on any computer that tries to pull these files from GitHub.

# References

Anggo, M. & Arapu, L., 2018. Face Recognition Using Fisherface Method [viewed 01/03/2019]. Available at: <https://iopscience.iop.org/article/10.1088/1742-6596/1028/1/012119/pdf>

Aston, B., 2017. Why Is Project Management Important? [viewed 27/02/2019]. Available at: <https://thedigitalprojectmanager.com/why-is-project-management-important/>

Berger, W., 2018. DEEP LEARNING HAAR CASCADE EXPLAINED [viewed 28/02/2019]. Available at: <http://www.willberger.org/cascade-haar-explained/>

Brems, M., 2017. A One-Stop Shop for Principal Component Analysis [viewed 27/02/2019]. Available at: <https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c>

EmguCV, 2019 [viewed 19/01/2019]. Available at:<http://www.emgu.com/wiki/index.php/Main_Page>

Gale, T., 2008. Normal Distribution [viewed 19/02/2019]. Available at: <https://www.encyclopedia.com/science-and-technology/mathematics/mathematics/normal-distribution#3>

GameCrate, 2016. CES 2016: Tobii Eye Tracking on Assassin's Creed Syndicate [viewed 25/02/2019]. Available at: <https://www.youtube.com/watch?v=6HHvCd_GPek>

Intel, 2018. Histogram of Oriented Gradients (HOG) Descriptor [viewed 14/06/18]. Available at: <https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor>

K, Salton, 2017. Face Recognition: Understanding LBPH Algorithm[ [viewed 14/06/2018]. Available at: <https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b>

Kemper, C., 2012. Foundation Version Control for Web Developers. Publisher: Apress

Nguyen, H., 2014. Eigenfaces scheme [viewed 01/03/2019]. Available at: <https://www.researchgate.net/figure/Eigenfaces-scheme_fig4_280596094>

OpenCV, 2019. Image Blurring (Image Smoothing) [viewed 25/02/2019]. Available at: <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html>

OpenCV, 2019. Open Source Computer Vision Library [viewed 19/01/2019]. Available at: <https://opencv.org/>

Pandey, P., 2018. Face Detection with Python using OpenCV [viewed 28/02/2019]. Available at: <https://www.datacamp.com/community/tutorials/face-detection-python-opencv>

Raschka, S., 2014. Linear Discriminant Analysis [viewed 28/02/2019].Available at: <https://sebastianraschka.com/Articles/2014_python_lda.html#principal-component-analysis-vs-linear-discriminant-analysis>

Ravlani, K., 2017. 6 Benefits of Good User Stories [viewed 27/02/2019]. Available at: <http://agileforgrowth.com/blog/userstory-benefits/>

sdenton4, 2015. Principal Component Analysis via Similarity [ viewed 01/03/2019]. Available at: <https://inventingsituations.net/2015/02/21/svdpca-via-similarity/>

Siegmund, D.O., 2019. Probability theory [viewed 19/02/2019]. Available at: <https://www.britannica.com/science/probability-theory>

SOD, 2019. An Embedded Computer Vision & Machine Learning Library [viewed 19/01/2019]. Available at: <https://sod.pixlab.io/>

Tobii, 2019. Tobii [viewed 24/02/2019]. Available at: <https://www.tobii.com/>

VisionLib, 2019. Augmented Reality Tracking Library for industries [viewed 19/01/2019]. Available at: <https://visionlib.com/>

What-when-how, Unknown. Morphology [viewed 19/02/2019]. Available at: <http://what-when-how.com/introduction-to-video-and-image-processing/morphology-introduction-to-video-and-image-processing-part-2/>

Zhang, S., Turk, M., 2008. Eigenfaces [viewed 26/02/2019]. Available at: <http://www.scholarpedia.org/article/Eigenfaces>

# Appendix

## Appendix A - Timeboxes

Note: All documentation includes evidence of testing and dependencies refer to the numbering of all other time boxes within this appendix

1. Setting up camera including initialisation of OpenCV - 6 hours
   1. Initialise OpenCV - 4 hours
      * Test it works – 3 hours 30 mins
      * Document – 30 mins
   2. Initialise Camera – 2 hours
      * Test – 1 hour 30 mins
      * Document - 30 mins
      * Dependencies: 1.1
2. Display camera – 1 hour
   1. Show camera on screen – 1 hour
      * Test - 45 mins
      * Document – 15 mins
      * Dependencies: 1.2.
3. Getting camera to recognise where the face is – 35 hours
   1. Locate the face – 16 hours
      * Dependencies: 1.2.
   2. Track the face position – 16 hours
      * Dependencies: 3.1.
   3. Testing – 2 hours
   4. Document – 1 hour
4. Getting the camera to recognise where the eyes, mouth and ears are – 26 hours
   1. Eyes - 10 hours
      * Test - 9 hours
      * Document – 1 hour
      * Dependencies: 1.2.
   2. Mouth – 8 hours
      * Test – 7 hours
      * Document – 1 hour
      * Dependencies: 1.2.
   3. Ears – 8 hours
      * Test – 7 hours
      * Document – 1 hour
      * Dependencies: 1.2.
5. Use this to calculate if a player is winking, blinking, turning their head etc – 20 hours
   1. Winking – 2 hours
      * Test – 1 hour 30 mins
      * Document – 30 mins
      * Dependencies: 4.1.
   2. Blinking – 2 hours
      * Test – 1 hour 30 mins
      * Document – 30 mins
      * Dependencies: 4.1.
   3. Turning head - 6 hours
      * Test – 5 hours
      * Document – 1 hour
      * Dependencies: 3.2.
   4. Mouth open – 10 hours
      * Test – 9 hours
      * Document – 1 hour
      * Dependencies: 4.2.

**Grand total:** 88 hours.

## Appendix B – Project Gantt Chart

## 

## Appendix C – User Stories

* As a player I want to be able to turn my head to make my character move because I want to explore.
* As a player I want to be able to use my eyes by winking or blinking to do certain actions because I want to be able to communicate with NPCs.