EE297 Intelligent Systems Project
Final Report

Team 1

Frank Gallagher

Artur Karolewski

Adam Duke

James Oliver

TABLE OF CONTENTS

ABSTRACT

A signal is a function of one or more variables which conveys information about a physical phenomenon or system. However, signals can be corrupted by an unwanted signal (noise). Signal processing is the means by which noise is removed from signals and information is extracted. Sensors capture signals of a given physical nature and translate them into an electrical form that computers can understand and analyse. Intelligent machines often rely primarily on visual information for the perception of and interaction with their environment since it is such an information laden signal. For this reason, image processing is a primary signal processing field with an RBG camera being the principal sensor.

Given a webcam and a Raspberry Pi, the team is assigned the task of designing an application that implements audio and/or image signal processing. This report outlines the development of a vision system that analyses how many people pass a point of interest, how many of those people look towards the point of interest and how long they look for. A machine learning based approach which features person detection, facial detection and head pose estimation is proposed. The application is designed with the intention of deploying it in a consumer analytics capacity. To begin, the team's idea is introduced along with the ethical considerations associated with it and the available hardware. A brief commentary on the theoretical bases of the proposed idea is provided. Following this, the implementation of the proposed idea is discussed with regards to implementation issues, system structure and algorithm development. Finally, the experimental validation of the application is described before concluding remarks and plans for future work are offered.

1 INTRODUCTION

1.1 PROJECT BRIEF

Design an application that implements audio and/or image signal processing using a Logitech C270 webcam and a Raspberry Pi Model 3.

1.2 HARDWARE AVAILABLE

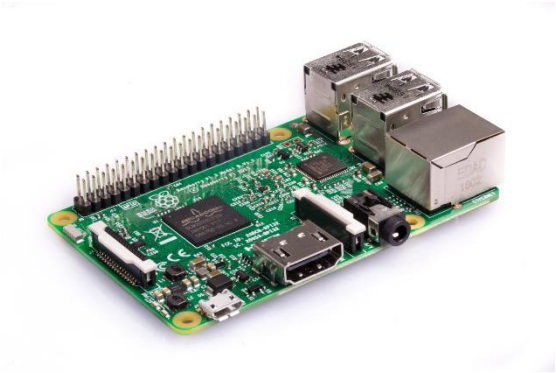
Audio and image sensing hardware and processing hardware is provided in the form of a Logitech C270 HD Webcam and a Raspberry Pi Model 3.

1.2.1 Logitech C270 HD Webcam

- **Optical Resolution:** 1.2MP (1280x960)
- **Frame Rate (max):** 30fps (640x480)
- **Field of View:** 60°
- **Focus Type:** Fixed (beyond 40cm)

1.2.2 Raspberry Pi Model 3

- **Processor:** 1.2GHz Quad Core CPU (64-bit)
- **Graphics:** VideoCore IV
- **Memory:** 1GB RAM
- **Storage:** microSD
- **OS:** Raspbian (boots from SD card)
- **Ports:** HDMI, 4xUSB 2.0, Camera Serial Interface (CSI), Display Serial Interface (DSI) and 3.5mm audio-video jack
- **Networking:** Ethernet and 2.4GHz wireless



1.3 CHOSEN PROJECT

A smart camera that checks if a person is looking in a particular direction. More specifically, a smart vision system that detects a person, locates their face and estimates their head pose, in order to determine the number of people that pass a point of interest, how many of them look towards the point of interest and for how long. This requires the implementation of person detection, face detection and head pose estimation. Person detection is the process of determining the presence and location of a person in an image. This is necessary to count the number of people that pass the point of interest. Once a person has been detected in an image, their face can be located for head pose estimation to determine which direction they are looking in. Head pose estimation is concerned with describing the orientation of a person's head in a 2D image using a 3D coordinate system.

Person detection, face detection and head pose estimation are inherently challenging. Detecting people in images is a difficult task owing to their variable appearance and the wide range of poses they can adopt [1]. The ability to cleanly discriminate the human form in cluttered backgrounds and under difficult illumination is essential for a robust person detection system. The variable appearance of people also poses challenges for face detection and head pose estimation as the system must demonstrate invariance to a range of factors, including biological appearance, facial expression and the presence of facial accessories like hats and glasses [2], in order to reliably detect faces and estimate head poses.

The aim of developing the proposed system is to combine the fields of person detection, face detection and head pose estimation into a single application designed for a specific use case, consumer analytics; analysing how many people pass a point of interest, how many of those people look towards the point of interest and how long they look for. A machine learning based approach is employed to combat the issues inherent in person detection, face detection and head pose estimation, some of which are mentioned above (the motivation for utilising machine learning is discussed in more detail in Section 2.2). Addressing these issues is especially important since the application's function is to obtain true and accurate counts of people and measurements of their gaze. Pretrained OpenCV models are used since it is impractical to build trainable structures, gather a sufficient training data set, train and validate the model in the given time frame.

1.4 ETHICAL CONSIDERATIONS

The intended use case of the system is consumer analytics which means that it will be deployed in public places. It requires video footage of people to perform its function. Such footage is sensitive personal data and could potentially be taken without the intentional participation of the individuals filmed due to the public setting. Although Irish Law permits recording in public places, as long as people are notified, it would be considered ethical to design the system in a way that all data is anonymous.

The system is not designed to identify individuals and track their interest in advertisements to build a consumer profile. The system analyses the footage to detect human shapes and determine the orientation of the head. Once the footage has been analysed and the results logged, the footage is deleted. The only data stored is an anonymous tally of the number of people that pass the point of interest, how many look at it and how long for.

1.5 Associated Challenges

The fields of person detection, face detection and head pose estimation have inherent difficulties, as mentioned previously. The specific use case of the system exacerbates these difficulties. The core problem is the variability of the operating conditions, to which the system must be invariant for accurate performance. It is designed for consumer analytics; analysing the interest of consumers in advertisements and shop displays. Therefore, it will have to operate in a range of lighting conditions. Inside light is mostly artificial so is consistent throughout the day, although, artificial light varies from building to building or even room to room. Outside lighting varies constantly depending on the weather and the time of day. There is also the possibility of sun glare.

The system's object of focus is a person in an upright pose, requiring it to be invariant to multiple factors that alter the appearance of a person such as biological appearance and clothes. Size and shape also varies from person to person as well as being affected by the distance they are from the camera; if they are up close then their legs won't be visible, if they are far away the ability of the system to detect them, their face and estimate their head pose may be limited by the resolution of the camera. Size and shape also depends on the positioning of the camera; people may be viewed from eye level or from vantage point. The system must reliably count the number of people that pass it. The accuracy of this measurement relies on the robustness of the detection algorithm. Due to the setting, multiple people could potentially be in view of the camera at a given time, causing missed detections as a result of occlusion. In order to minimise duplicate detections, people need to be tracked from frame to frame. The data collected by the system must be reliable and accurate, such accuracy is threatened by the challenges mentioned above as they may cause false positives and false negatives. The Raspberry Pi's limited hardware also poses a challenge since the adopted approach is machine learning based and running models is computationally expensive.

1.6 Literature Review

Extensive research has been carried out in the fields of Person Detection, Face Detection and Head Pose Estimation. What is considered to be the state of the art is mentioned here.

Naveet Dalal and Bill Triggs describe a linear SVM based human detector that uses grids of Histograms of Orientated Gradients (HOG) as feature descriptors. They show experimentally that 'grids of HOG descriptors significantly outperform existing feature sets for human detection' [1].

Paul Viola and Michael Jones propose the Haar Cascade approach for object detection. They use a new image representation called the 'Integral Image' along with a learning algorithm and a method for combining increasingly more complex classifiers in a 'cascade', to rapidly detect objects. They claim that 'in the domain of face detection, the system yields results comparable to the best previous systems' [3].

Erik Murphy-Chutorian and Mohan Manubhai Trivedi present a survey describing the evolution of the field of head pose estimation. They focus on the advantages and disadvantages of each approach and compare them based on their ability to estimate coarse and fine head pose, highlighting the approaches that are well suited to unconstrained environments [2]. They do not conclude that a single approach is the best. Instead they recognize the potential of many different approaches and offer design criteria for the future development of head pose estimation systems.

2 THEORETICAL BACKGROUND

In this section, certain theoretical aspects relevant to the chosen application outlined in Section 1.3 are discussed. Namely the formation of images, the application of machine learning in image processing, neural networks, using Histograms of Orientated Gradients for person detection, the Haar Cascade approach to face detection and finally head pose estimation.

2.1 IMAGE FORMATION

In the past images were captured using photographic film. Nowadays, digital sensors are used to artificially mimic the transduction process of a biological eye. In the human eye, rod and cone receptors work in conjunction with ganglion cells to convert photons into an electromechanical signal, which the brain then interprets as an image. In the case of digital cameras, photons are captured as charged electrons on a layer of silicon and are then converted to a voltage value through the use of capacitors and amplifiers. This voltage values are converted to digital code that a computer can understand. The two most common types of photosensors are CCD (Charged Coupled Device) and CMOS (Complementary Metal Oxide Semiconductors), which are discussed in Section 2.1.2 below. Both of these photosensors use a Bayer filter.

2.1.1 Bayer Filter

A Bayer filter is a grid of alternating red, green and blue (see Figure 1 below). Green colour filters make up half of the grid (since human eyes are most sensitive to green), with red and blue taking up a quarter of the grid each. The grid of filters sits above the photosensor, which itself is an array of sensor cells (pixels) on a silicon sheet. White light passes through each of the colour filters in the grid. The pixels in the photosensor then have a red, green or blue intensity value corresponding to the colour filter which sits above it. The photosensor now contains the same pattern of red, green and blue pixels (referred to as a mosaic) as the Bayer filter. This pattern of red, green and blue pixels is converted to a colour image with RBG values for each pixel through the process of Demosaicing or Debayering. Take a pixel with a green filter for example, the exact value for the green intensity is known, the red and blue intensity values for this pixel are interpolated from the neighbouring blue and red pixels to obtain an RGB value for the pixel. For more information regarding image formation, Bayer filters and Demosaicing algorithms see [4], [5].

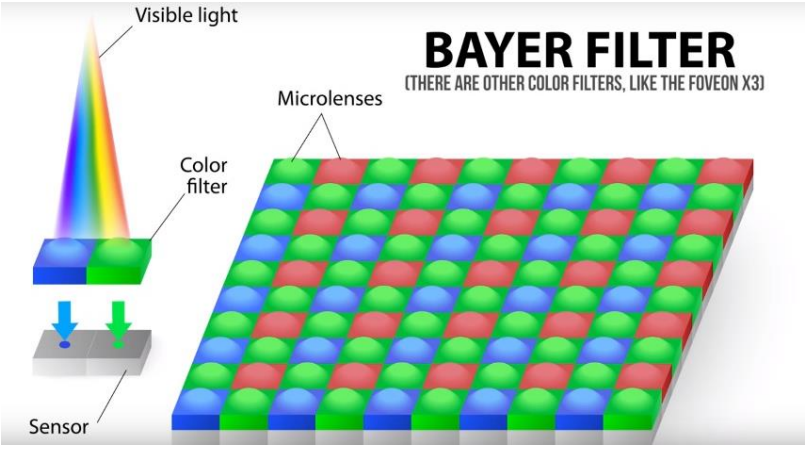


Figure 1 - Bayer Filter [6]

2.1.2 CCD vs CMOS

In a **CCD** (Charged Coupled Device), once the pixels have been captured, the grid of pixels is processed one at a time from the bottom to the top. A serial shift register is used to push each pixel out of the grid sequentially. The pixel value is converted to an analogue voltage before being processed into a digital signal using an ADC (Analogue to Digital Converter). Since each pixel is processed individually, it is slower and more power intensive than CMOS photosensors. However, it has the benefit of being less sensitive to noise. A **CMOS** (Complementary Metal Oxide Semiconductor) performs the same operations as a CCD but has circuitry for each individual pixel, allowing them to be processed simultaneously. This method achieves lower power consumption than CCD and faster processing speeds.

2.1.3 Digital Representation of Images

In computer memory each pixel of an image is represented by a binary value. The binary value represents the colour of the pixel. This binary representation of colours is called a bit plane. The number of different colours that need to be encoded determines the number of bits used. *n* bits can represent up to 2^n colours. Take Figure 2 below for example, it contains four different colours, so two bits are required to represent each pixel. Images are stored in scan lines. A single scan line encodes a row of pixels from left to right. The image is encoded from the top to the bottom. In order for the computer to interpret the image it needs to know the colour depth (how many bits represent each pixel) and resolution (width and height in pixels) of the image.

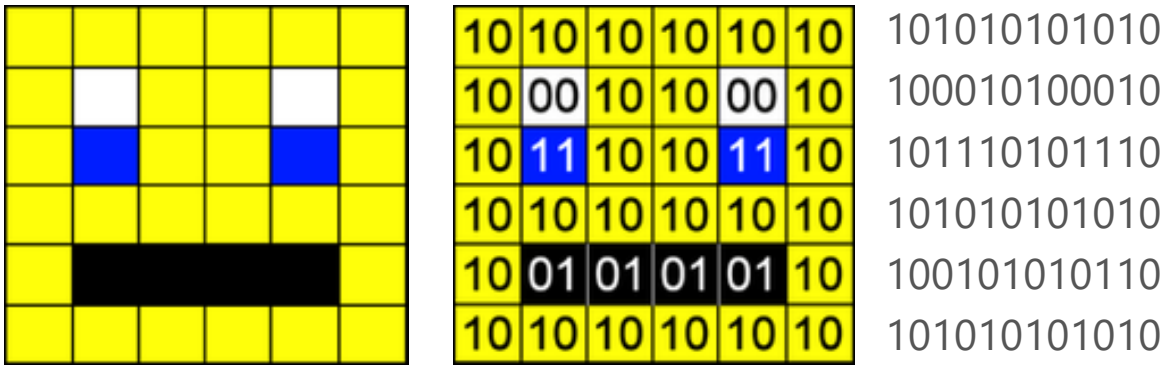


Figure 2 – Digital representation of an image

2.2 Machine Learning in Image Processing

Images exhibit tremendous variation; the same scene can be viewed from infinitely many orientations, under infinitely many lighting conditions, from up close or far away. In each case, although the specific intensity values of each pixel may change, the object of interest in the scene may be the same. For example, when detecting people in images, it is immaterial whether the image was taken at night or during the day. Classic image processing approaches to image processing consisted of some human insight appropriately coded into algorithmic form. However, these approaches did not generalise well and could not accommodate the variability present in image data.

Machine learning techniques have been successfully employed to complete image processing tasks such as image classification and object detection. Structures such as artificial neural networks and SVMs (support vector machines) are trained with a learning procedures like backpropagation of error. Instead of relying on human insight, the neural network is presented with a large number of training samples which implicitly capture the nature of the problem at hand and the variability inherent in the data. The neural network is then able to infer information when presented with new images which it has never seen. Neural networks tend to generalise far better than classical approaches and can be trained to be invariant to image transformations, such as variations in scene rotation and illumination, simply by including these transformations in the training samples (neural networks are discussed further in the next section).

One pipeline for machine learning based image processing is shown in Figure 3. A feature extraction technique (such as Histograms of Oriented Gradients) is used to identify features in the image and generate a feature vector. In computer vision and image processing, a feature is an aspect of an image that is of particular interest and relevance for solving a computational task related to a certain application. Features include structures in an image such as edges and corners. These features are described by feature vectors. A feature can describe an entire image (global feature) or a feature present at a location in the image (local feature). The classifier, which has been trained to recognise a particular class or classes of object, takes this feature vector as input and outputs the label of the object(s) that the image contains. During training the classifier is provided with the image label (supervised learning) so that it can learn which feature vectors correspond to which objects. Object detection provides the locations of the objects in the image by outputting both labels and bounding boxes. This can be achieved by sliding a detection window of multiple scales over the image and extracting features at each instance, in this way when the classifier outputs a label, the location from which the feature was extracted is known.

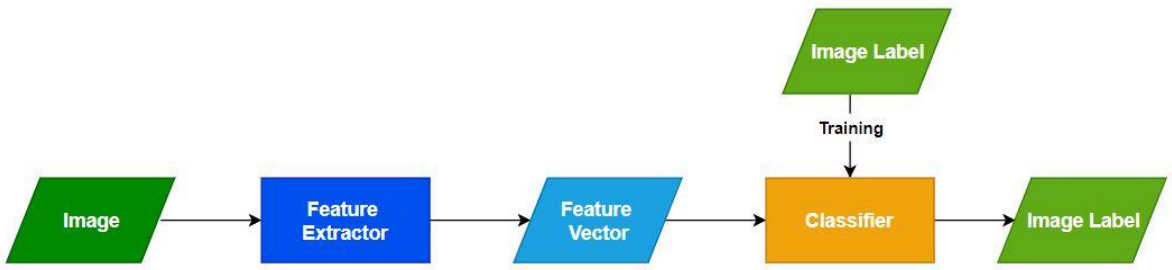


Figure 3 – Pipeline for using machine learning in image processing

2.3 Neural Networks

2.3.1 Structure

Simon Haykin defines a neural network as being a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use [7]. It resembles the brain in two respects: Knowledge is acquired by the network from its environment through a learning process. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge [7]. Put simply, a neural network is a structure used to represent a function so that the parameters of the function can be tuned automatically by a training process. The traditional structure of a neural network (shown in Figure 4 below) is called a Multi-Layer Perceptron (MLP). It consists of multiple layers of nodes called neurons. The input layer contains as many neurons as are required to accommodate the structure of the input. For example, if the input was a 28x28 pixel image, then there would be 784 input nodes, one for each pixel in the image. If the input was a feature vector, there would be as many input nodes as entries in the vector. The output layer contains as many neurons as there are possible outputs. There can be any number of hidden layers containing any number of neurons. Each neuron is connected to all of the neurons in the previous layer.

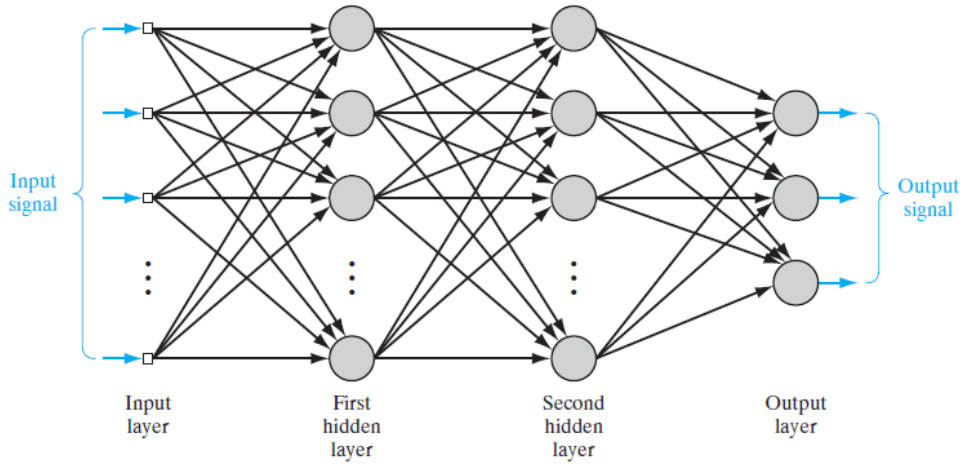
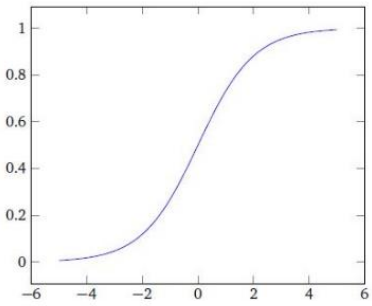


Figure 4 - Structure of a neural network [7]

Each connection has a weight associated with it and each neuron has a bias associated with it. These are the parameters that need to be tuned. The activation of a neuron is determined by computing the weighted sum of all outputs of previous layer. The bias value associated with the neuron is the threshold that must be reached for the neuron to become activated. This value is subtracted from the sum. The result of the weighted sum is then reduced to a number between zero and one using a function such as a sigmoid (see Figure 5 [8]). Other functions like ReLU (Rectified Linear Unit) may also be used. These steps define the neuron's activation function shown by equation (1) [9], where $a^{(n)}$ is the activation of a neuron in layer n, σ is the sigmoid function, W is the row vector containing the weights of the connections between the neuron in layer n and all of the neurons in layer $n-1$ and b is the bias of the neuron (a negative value). The activation of each neuron feeds forward through the network, comprising part of the input to neurons in the next layer, until the output layer is reached.

$$a^{(n)} = \sigma\left(W a^{(n-1)} + b\right) \quad (1)$$

As shown below in Figure 5, the sigmoid function produces an output close to zero when the input is very negative, an output close to 1 when the input is very positive with a steady increase for inputs around zero.



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Figure 5 - Sigmoid plot and equation [8]

2.3.2 Training

In order for neural networks to learn they are presented with a set of training data. Each training sample is labelled with the correct output. The weights and biases are set to random values initially, before being adjusted to improve the accuracy of the network on the training data. The hope is that what the network learns will generalise to new inputs that the network has never seen before. The network is validated by showing it a new set of data with labels and observing how accurately it performs.

The error of the network on a particular training example is given by summing up the squares of the differences between what the output activations are and what they should be, as shown in equation (2) below, where C is the cost, n is the number of output neurons, X'_i is the activation of output neuron i and X_i is the what the activation of output neuron i should be. This sum is small if the network produced accurate output node activations, but large if it produced incorrect output node activations. Taking the average error of the network across all training examples gives the overall cost of the network. The aim of training is to adjust the weights and biases of the network in a way that minimises the value of the cost function as quickly as possible.

$$C = \sum_{i=0}^{n-1} (X'_i - X_i)^2 \qquad (2)$$

If the neural network is a function with the input nodes as inputs, output nodes as outputs and weights and biases as parameters, then the cost function of the network is a function with weights and biases as inputs, a cost as an output and training examples as parameters. The gradient of a function describes the gradient of steepest ascent. Therefore, the negative of this gradient describes the direction of quickest descent. The procedure for minimising the overall cost of the network, is calculating the weight and bias adjustments to minimise the cost for each individual training example and averaging them together. The algorithm for determining the weight and bias adjustments to minimise the cost for a single training example is called backpropagation.

For a given training example, the negative gradient vector describes the weight/bias adjustments that must be made in order to minimise the cost for that example. It contains as many entries as there are weights and biases in the network. The sign of each entry indicates whether the corresponding weight/bias should be increased or decreased. The magnitude of each entry indicates by how much the weight/bias should be increased/decreased. Recall that the activation of a neuron is determined by the weighted sum of the activations of the previous layer plus some bias value. Consider a single output node. In order to modify its activation to match the value that it should be, its bias value along with the weights of the connections to the previous layer must be adjusted in some way. These adjustments are added to those that the other output nodes require to achieve their correct value. Moving backwards through the network, this process is applied recursively to adjust the weights and biases that determine the activation of previous layers. As mentioned above, repeating this backpropagation routine for every training example minimises the overall cost of the network, thus improving its accuracy.

2.4 Support Vector Machines

An SVM is another structure that can be used as a classifier. It is a discriminative classifier defined by a separating hyperplane. Given labelled training data (supervised learning), the learning algorithm outputs an optimal hyperplane which categorises new examples. A hyperplane of a particular dimensional space is a subspace that has one dimension less than the space in which it is contained. For example, in a 2D space the hyperplane is a line (1D) dividing a plane (2D) in two parts, where each classification is either side. The aim of training the SVM is to find the hyperplane that separates the classes with a minimal number of classifications. There are three parameters that must be tuned in the training of the SVM: Kernel, Regularisation, Gamma and Margin. The **kernel** determines the type of equation used to calculate the separating line. Kernels can be linear, polynomial or exponential. Polynomial and exponential kernels calculate the separation line in a higher dimension (see Figure 6). For more information on SVMs see [10].

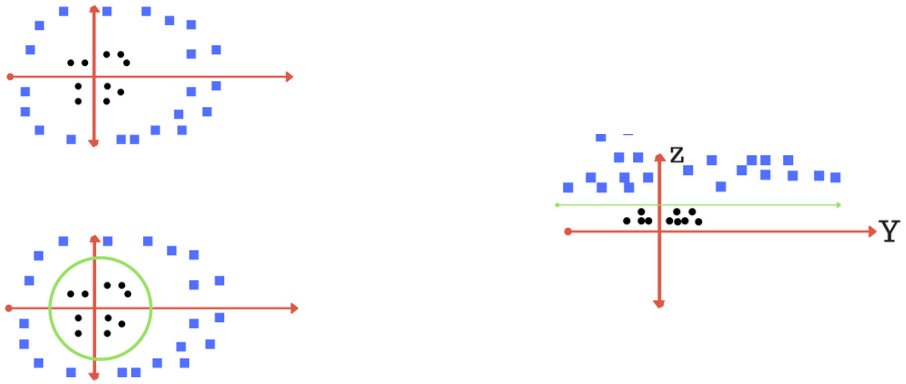


Figure 6 - (top) 2D classification space, (right) calculation of separating line in 3D space, (bottom) separation in 2D [11]

The **regularisation** parameter determines the toleration of misclassification. Large regularisation values yield a smaller margin which does a better job of classifying every training example correctly. Small regularisation values yield larger margin separating hyperplane, even if it misclassifies more points (see Figure 7).

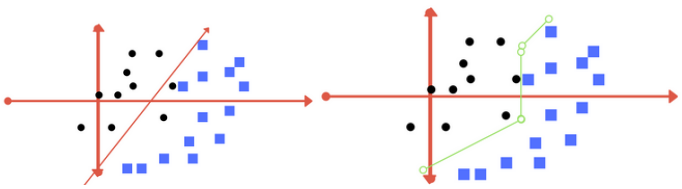


Figure 7 - (left) Low regularisation value, (right) high regularisation value [11]

The **gamma** parameter determines how far the influence of a single training example (point) reaches. Low gamma values mean that points far away from the separation are considered in the calculation of the separation. High gamma values mean that only points close to the separation are used in the calculation of the separation. The **margin** is the distance of the separation from each class. A good margin is equidistant from both classes and is as far as possible from both classes (see Figure 8).

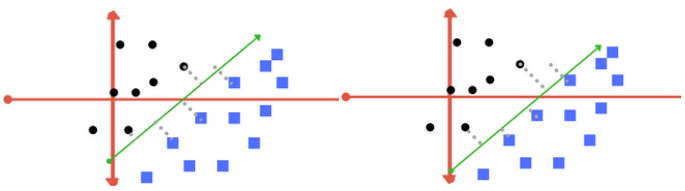


Figure 8 - (left) Good margin equidistant from both classes, (right) bad margin very close to one class [11]

2.5 Person Detection using Histograms of Oriented Gradients

Person detection is a form of object detection where the object class is 'Human'. Object detection is the process of determining both the presence of a predefined type of object and its location in an image. The machine learning pipeline for image processing outlined in Section 2.2 can be applied to person detection. Dalal and Triggs propose the Histogram of Oriented Gradients (HOG) approach to person detection [1], where Histogram of Oriented Gradients is the feature extraction technique used to generate a feature vector and an SVM is used as a classifier. The feature extraction and object detection chain for this approach is shown in Figure 9 below.

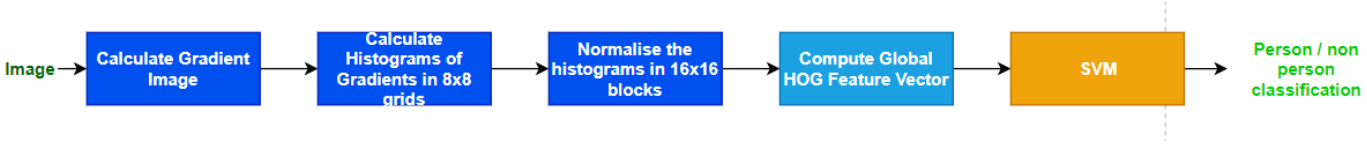


Figure 9 - Feature extraction and object detection chain for HOG person detection

A HOG is a type of feature descriptor. A feature descriptor is a representation of an image (or part of an image) that simplifies the image by extracting useful information and discarding extraneous information [12], to generalise the object of interest in such a way that it produces the same feature descriptor when viewed under different conditions. In the HOG feature descriptor, the distribution of directions of gradients are used as features. Gradients are X and Y derivatives of an image. The magnitude of gradients is large around edges and corners which describe more about an object's shape than flat regions. HOG uses a single feature vector to describe a person rather than many smaller feature vectors representing parts of a person. A sliding detection window is moved around the image. At each position of the detection window a HOG descriptor is computed which is then passed to the SVM to be classified as either 'person' or 'not person'. The image is subsampled to multiple sizes and each of these subsamples are searched to detect people at different scales.

The first step is to obtain the gradient image by calculating the horizontal (y) and vertical (x) gradients at each pixel. This is achieved by first filtering the image with the kernels shown in Figure 10 and then computing the magnitude and direction of the gradient for each pixel using equation (3) and equation (4). For colour images, the gradients for the red, green and blue channels is calculated. The magnitude of the gradient at that pixel is the largest of the magnitudes of the three channels. The angle is that of the largest magnitude. The gradient images are depicted in Figure 11.

$$g = \sqrt{{g_x}^2 + {g_y}^2} \qquad (3)$$

$$\theta = \arctan\left(\frac{g_y}{g_x}\right) \qquad (4)$$

-1	0	1

-1
0
1

Figure 10 - Filtering Kernels

Figure 11 - (left) Original image, (middle left) x gradient, (middle right) y gradient, (right) magnitude of gradient) [12]

The image is then divided up into grids of 8x8 pixels and the HOG is calculated for each grid. The histogram is a vector with 9 bins corresponding to angles 0, 20, 40, 60, ···, 160. For each pixel in the grid we look at the angle and magnitude. The angle determines which bin is added to and the magnitude determines the value that is added. If the angle is halfway between two bins then the value is split evenly between the two bins. This process is shown in Figure 12 below.

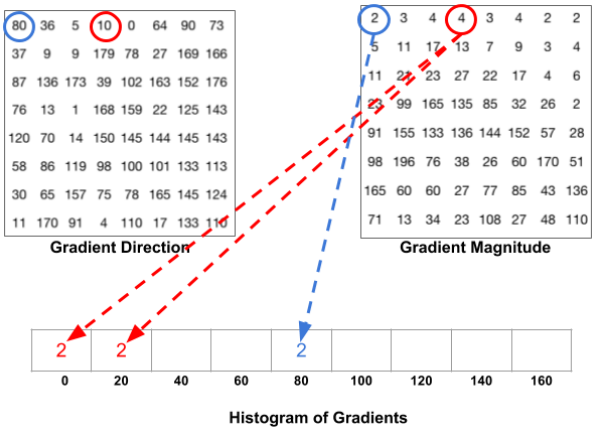


Figure 12 - Forming the histogram of gradients for a 8x8 grid of pixels [12]

The histograms are then normalised in blocks of four (16x16 pixels) to make them invariant to changing light conditions. To calculate the final HOG feature vector these blocks are concatenated. The HOG descriptor of the image is shown in Figure 13. Its visible that the dominant direction of the histogram captures the shape of the person. The final HOG feature vector is passed to the SVM to be classified. For a more in depth explanation of the HOG approach to person detection see [1].

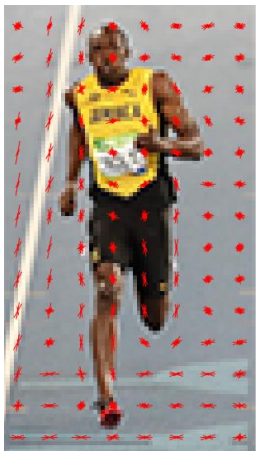


Figure 13 - Visualising the Histogram of Oriented Gradients [12]

2.6 Face Detection using Haar Cascades

Haar Cascades is a machine learning based approach to object detection, where a cascade function is trained from a set of positive and negative images and is then used to detect objects in new images. It was presented by Viola and Jones in 2001 [3]. There are three unique contributions to this approach: (i) a new image representation called the 'Integral Image', which allows the features used by their detector to be calculated quickly (ii) a learning algorithm which selects a small number of critical features from a larger set and (iii) a method for combining increasingly more complex classifiers in a 'cascade' which allows background regions of the image to be discarded quickly while spending more computation on promising object-like regions [3]. It is a popular method for face detection, since it is most clearly distinguished from previous approaches in its ability to detect faces extremely rapidly [3]. Such an attribute is desirable in our system since there will be high volumes of pedestrians in our system's use case.

2.6.1 Haar Features

Haar features are rectangular features as shown in Figure 14. Each feature is a single value obtained by subtracting the sum of pixels in the white rectangle from the sum of pixels in the black rectangle. All possible sizes and locations of each kernel is used to calculate lots of features. The integral image was introduced to compute these features rapidly. Out of all the features computed only a select few are relevant. The adaboost learning algorithm is used to select the relevant features. Simply put all features are applied to every training example in an attempt to find the features that best classify the face with minimal misclassifications (see Figure 15)

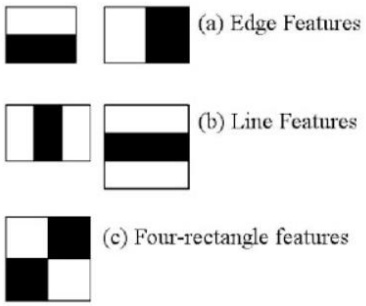


Figure 14 - Haar Features [13]

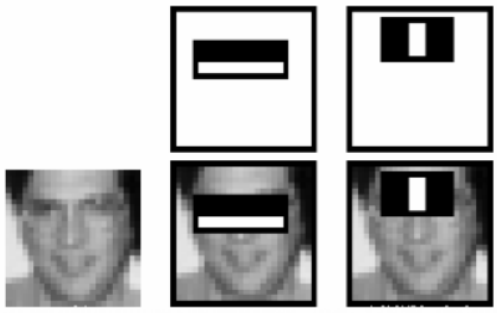


Figure 15 - Haar Features Classifying a Face [3]

2.6.2 Integral Image

The integral image reduces calculations for a given pixel to an operation involving just four pixels, no matter how large the image is (see Figure 16). The integral image *ii* at *x,y* contains the sum of the pixels above and to the left of *x,y*. *i(x,y)* is the original image.

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y') \tag{5}$$

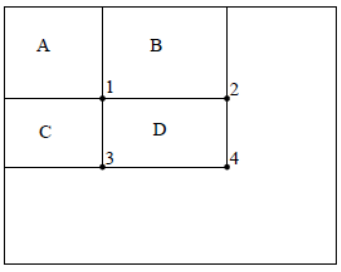


Figure 16 - Sum of pixels in D can be computed as ii(4)+ii(1) - [ii(2)+ii(3)] [3]

2.7 HEAD POSE ESTIMATION

Head pose estimation is concerned with inferring the 3D orientation of the human head, with respect to a camera, from a 2D image. Head pose can be described by specifying the three rotational angles (pitch, yaw and roll) about the three corresponding translational axes. (x, y and z), shown in Figure 17 below. This involves first determining the location of the head in the image which requires face detection. A non-pose-specific face detection system is needed, otherwise a 'chicken-egg' problem arises: the location of the head is required, while the pose is required to locate the head [14]. There are a number of approaches to estimating head pose estimation as discussed in [2]. Two such approaches are **Detector Array Methods** and **Geometric Methods**. For information on other head pose estimation methods see [2].

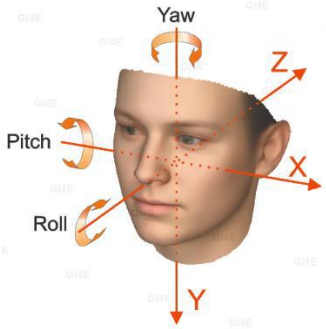


Figure 17 - Translation and rotation axes [15]

2.7.1 Detector Array Methods

A detector array consists of multiple face detectors, each one trained to a specific head pose. For binary classifiers, successfully detecting the face specifies the head pose, assuming that only one detector positively classifies the face. For detectors with continuous output, the head pose can be estimated by the detector with the most confidence (see Figure 18). An advantage of detector array methods is that a separate head detection and localization step is not required, since each detector is also capable of making the distinction between head and nonhead. Simultaneous detection and pose estimation can be performed by applying the detector to many subregions in the image.

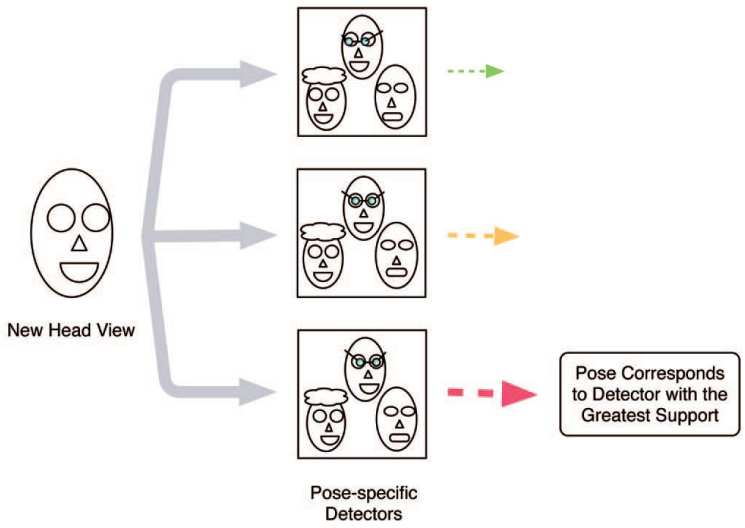


Figure 18 – Detector array for estimating head pose [2]

2.7.2 Geometric Methods

Geometric methods use the position of facial landmarks such as eyes, mouth and nose tip to determine head pose from their relative configuration. One way of using these facial features to estimate the head pose is finding the facial symmetry axis by connecting a line between the midpoint of the eyes and the midpoint of the mouth. Then assuming a fixed ratio between these points and a fixed length of the nose, the gaze direction can be determined from the 3D angle of the nose.

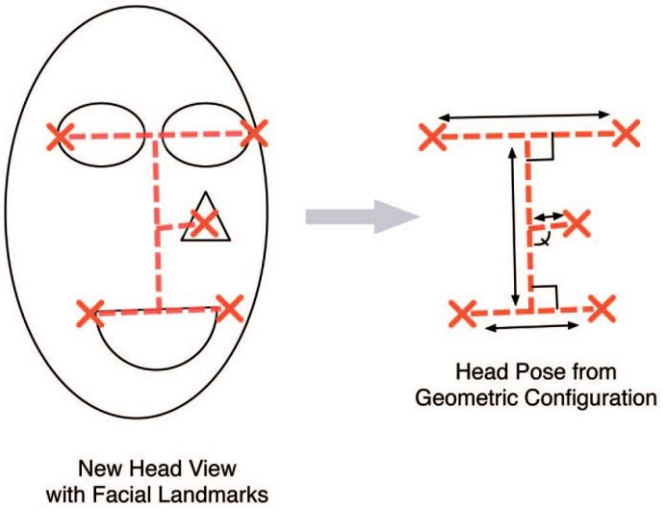


Figure 19 - Geometric head pose estimation [2]

3 IMPLEMENTATION

3.1 HARDWARE IMPLEMENTATION

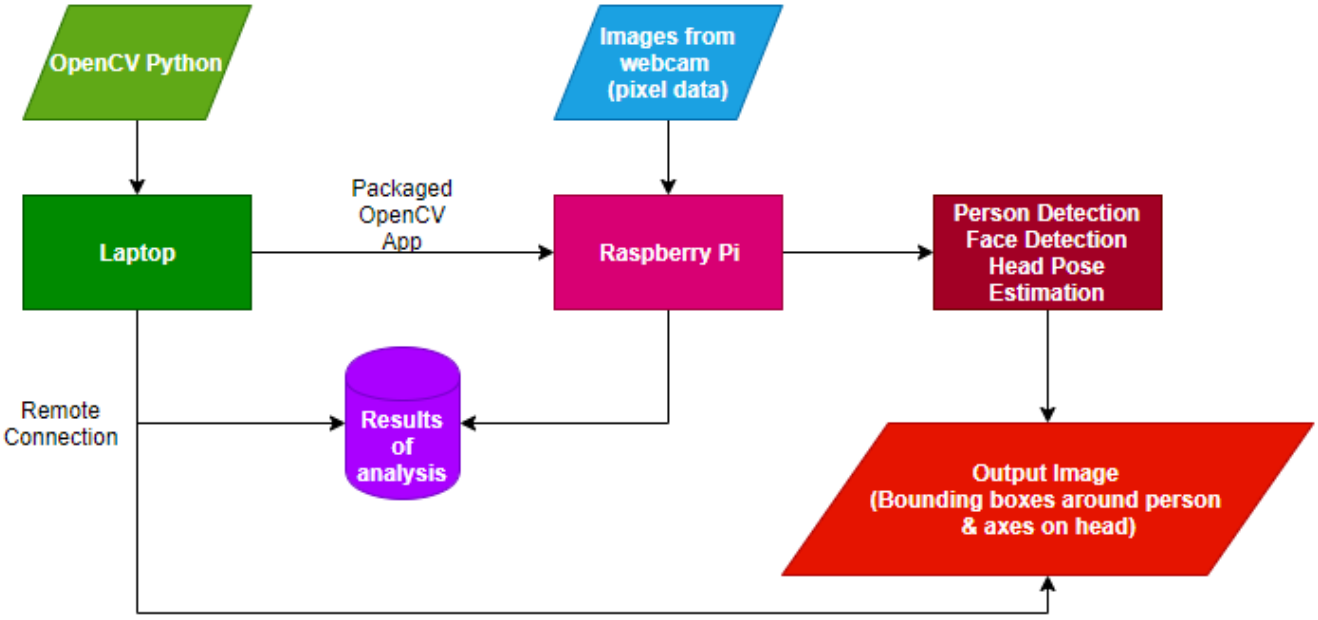


Figure 20 - System design

The system as depicted in Figure 20 above consists of a laptop, webcam and Raspberry Pi. The software for the system is developed on the laptop using the python version of the OpenCV computer vision library. The software consists of python scripts which use pretrained models from OpenCV to perform person detection, face detection and head pose estimation. These scripts are then packaged into an app so that the software can be deployed to the Raspberry Pi. The app runs on the Raspberry Pi where it receives images from the webcam as input. The pixel data from the webcam is passed to a HOG person detector to determine if a person is present in frame and their location (see Section 2.5 for an explanation of how HOG person detection works). If a person is present a counter is incremented and the region of the image which contains the person is passed to a Haar Cascade face detector which is used as a coarse head pose estimator and also locates the head for the fine head pose estimation. The system then checks if the detected person was looking towards the point of interest by comparing the head pose angles against the predefined angle ranges. The time duration that the head pose angles lie within these ranges is considered the length of time that the person was looking at the point of interest. The output of this process is an image with a bounding box around the person and axes on the detected person's head to describe their head pose. The results of the analysis are sent to a log file. These results are the number of people that passed the point of interest, how many of them looked towards the point of interest and how long they looked for. The laptop is remotely connected to the Raspberry Pi so the user can view the output images and analysis results without removing the system from its operating point.

3.2 Algorithm Design

A fully developed software solution for the system must be capable of counting the number of *different* people that pass the point of interest and estimate their head pose to determine the direction of their gaze. Due to the challenges discussed previously in relation to duplicate detections (see Section 1.5), a ground up approach was applied to developing an algorithm. The initial algorithm was designed to be applied to a simplified version of the problem. This could then be modified into a more complex and robust algorithm. Both algorithms use a frontal face detector as a means of coarse head pose estimation. This can be replaced by a head pose estimation technique for a more accurate estimation.

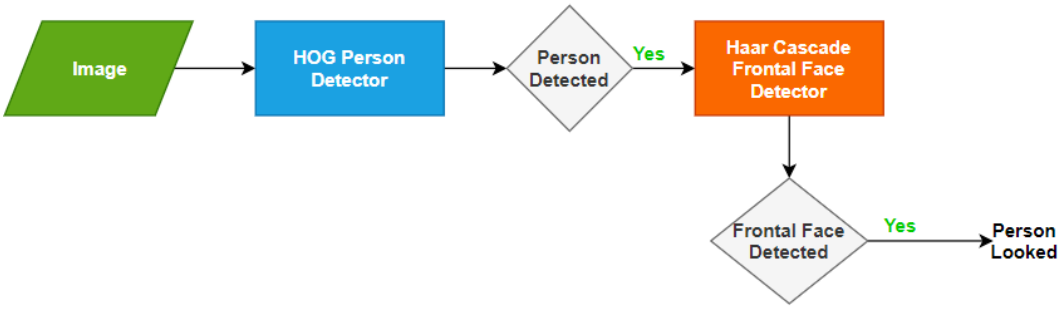


Figure 21 - Simplified Algorithm

The initial algorithm, shown above in Figure 21, determines how many times the point of interest has been looked at. It takes in an image as input. A HOG person detector is then used to check whether a person is present in the image and if so, determine their location. If a person is detected, the part of the image where they were detected is passed to a Haar Cascade frontal face detector. If a frontal face is detected, that person is considered to have looked. Since the number of different people detected is not being recorded, there is no requirement to track detected people from frame to frame.

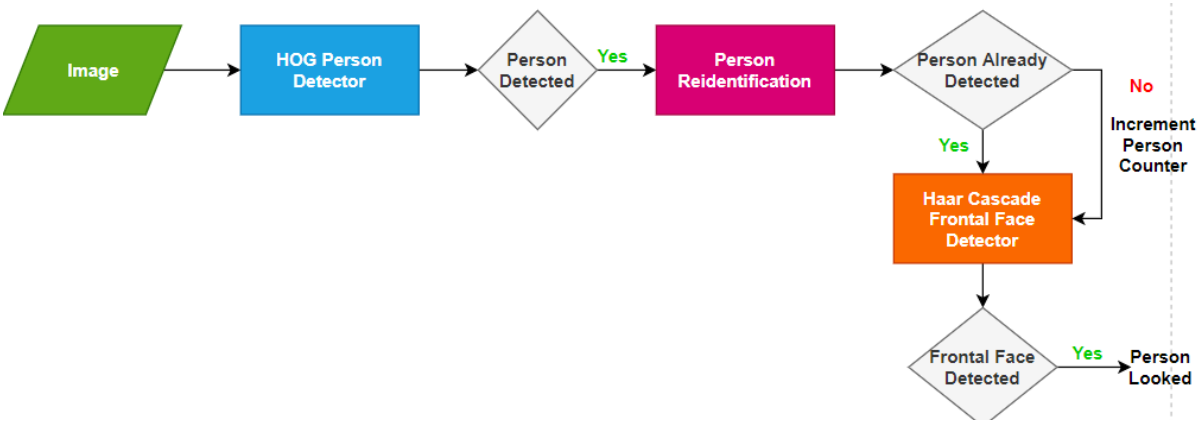


Figure 22 - Modified Algorithm

The modified algorithm, shown in Figure 22, expands on the initial algorithm. It still uses a HOG person detector, however, when a person is detected a person reidentification routine is invoked. The already detected person is compared to the detected persons in the last frame to determine if the detection is a new person. If so, the person counter is incremented and then the frontal face detection routine performed. Since this algorithm can distinguish detected people, it can count the number of different people that passed the point of interest. Using the time stamps on the frames it can also determine how long each person looked by calculating the duration that the person first started looking to when they stopped.

3.3 SOFTWARE IMPLEMENTATION

3.3.1 OpenVINO

OpenVINO is a computer vision library focussed around neural networks that is optimised to run on Intel hardware. It was used to implement the person detection and tracking as well as head pose estimation. The **pedestrian tracker** (see Figure 23) performs inference on the pretrained pedestrian detector network shipped with the OpenVINO toolkit. A tracker class then matches pedestrians with already tracked persons from frame to frame to prevent duplicate detections. The locations of the detected pedestrians are shown by bounding boxes. The trajectories of the detected people are shown as well as the total count of people.

Figure 23 - OpenVINO Pedestrian Tracker

The OpenVINO **face detector** (see Figure 24) runs four neural networks simultaneously: facial landmarks detection, head pose estimation, emotions recognition and age/gender recognition.

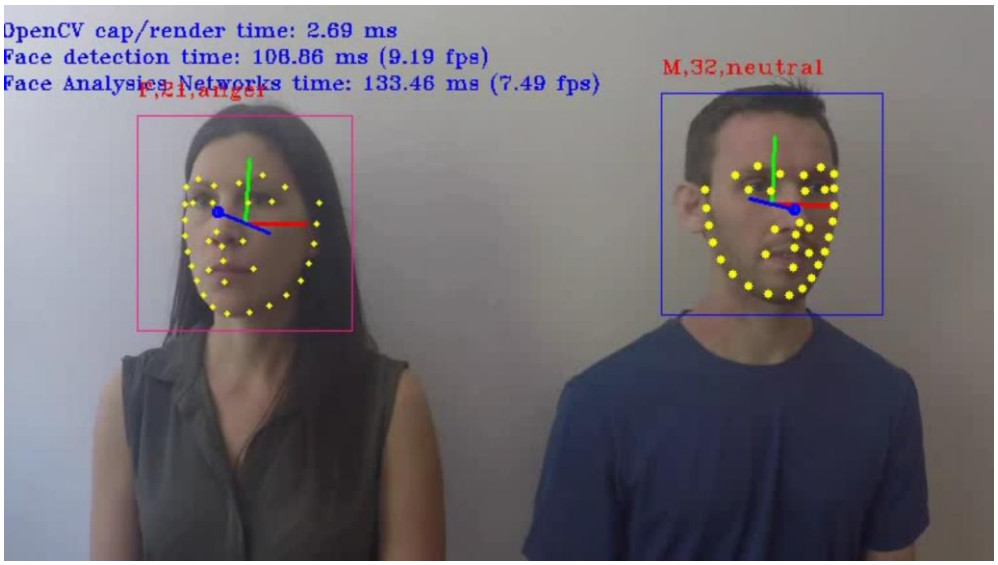


Figure 24 - OpenVINO Face Detector

The requirement of Intel hardware to run software developed with the OpenVINO toolkit as well as its resource heavy nature, made it impractical to run on the limited hardware of the Raspberry Pi.

3.3.2 OpenCV

The OpenCV computer vision library was used to implement separate python programs for person detection, face detection and head pose estimation. The code for these programs is included in the appendix.

Figure 25 - OpenCV Person Detection using HOG and SVM

The **person detection** script uses a pretrained HOG and Linear SVM model to detect people. It takes in a list of images as input. For each image it draws a bounding box around detected people. It then uses non maxima suppression to remove multiple bounding boxes around the same person (see Figure 25).

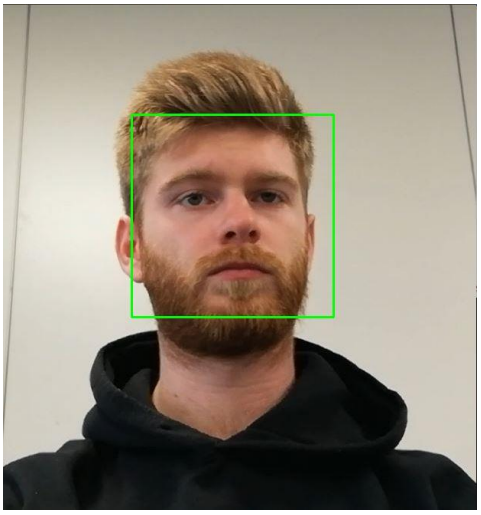


Figure 26 - OpenCV Face Detection using Haar Cascades

The **face detection** script takes in a single image and uses a pretrained Haar Cascade frontal face detector. It outputs an image with a bounding box around the detected face(s) (see Figure 26).

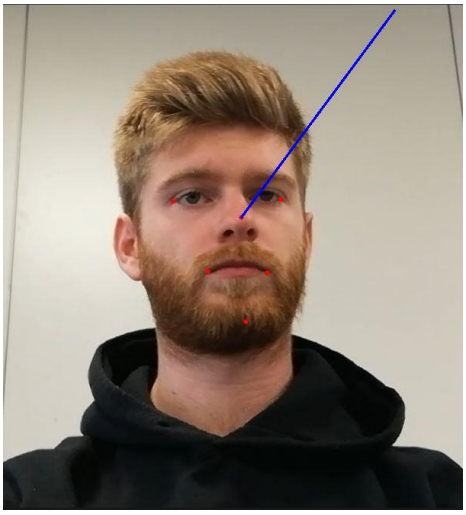


Figure 27 - OpenCV Head Pose Estimation

The **Head Pose Estimation** script is a primitive example of head pose estimation. It takes a single image as input and uses the locations of facial features to estimate the pose of the head and outputs an image with a line pointing from the nose in the direction of the estimated pose. These features are the nose tip, chin, the left and right corners of the mouth and the corners of both eyes. However, the locations of these features must be manually entered into the script prior to running it.

4 EXPERIMENTAL VALIDATION

As mentioned previously, the variable appearance of people poses significant challenges for person and face detection. The HOG person detector was tested by showing it three images of the same person viewed from different angles (see Figure 28). It successfully detected the person in the image when viewed from the back, the side and the front.

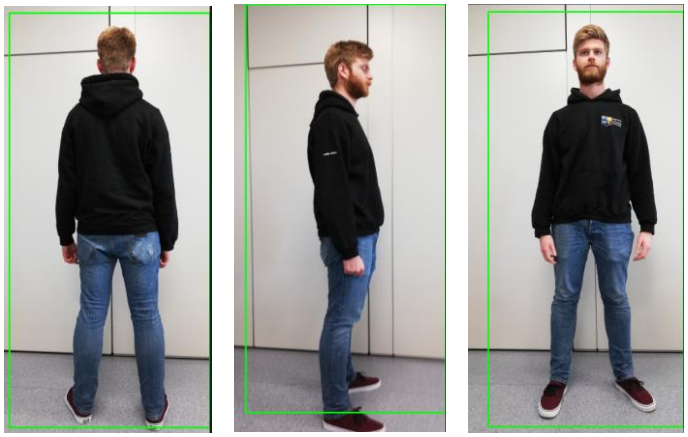


Figure 28 - Detecting a person from different angles

The robustness of the Haar Cascade face detector was tested in multiple ways. First by showing it three images of the same person viewed from the front the back and the side. Then by showing it images of a similar nature, instead with the person wearing facial accessories (glasses, sunglasses, hat and hood). It correctly detects the face when viewing it from a frontal view. It correctly detects nothing when viewing the head from the back. However, it incorrectly locates the face in the image from viewing it from the side (see Figure 29). In the presence of facial accessories, the detector successfully detects the face in each case (see Figure 30).

Figure 29 - Detecting faces from different angles

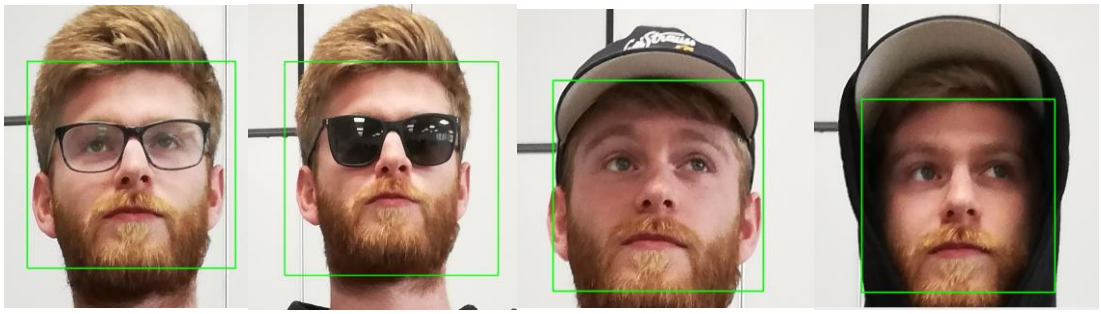


Figure 30 - Detecting faces with facial accessories

5 CONCLUSIONS AND FUTURE WORK

This report outlines the development of a consumer analytics smart camera that analyses how many people pass a point of interest, how many of those people look towards the point of interest and how long they look for. This requires the implementation of person detection, face detection and head pose estimation. Detecting people and faces in images is a difficult task owing to the variable appearance of a person. A machine learning based approach was applied to implement these procedures. The pipeline for applying machine learning in image processing involves extracting features from the image (Histograms of oriented gradients in the case of person detection and Haar features in the case of face detection) and passing these features to a classifier (based on a neural networks or SVM). The system design consists of a laptop, a Raspberry Pi and a webcam. The software is developed on the laptop using OpenCV. It is then packaged into an app and deployed to the Raspberry Pi. The software runs on the Raspberry Pi and performs person detection, face detection and head pose estimation on the images it receives from the webcam. The laptop is remotely connected to the Raspberry Pi, enabling the user to view the results of the analysis without removing the system from its operating location. Due to the challenge of tracking detected persons from frame to frame to avoid duplicate detections, the first algorithm design served only to detect people and their face, as a means of coarse head pose estimation. This algorithm was then expanded to track detected people from frame to frame, allowing the system to count the number of different people that pass by and calculate how long they look at the point of interest for. Person detection and head pose estimation were initially implemented using the OpenVINO toolkit. However, due to hardware limitations, it was impractical to run software developed with this toolkit on the Raspberry Pi. Instead person detection, face detection and head pose estimation scripts were written using OpenCV. The person detection script used a pretrained HOG and linear SVM detector. The face detection script used a pretrained Haar Cascade frontal face detector. The head pose estimation script used the location of facial features to estimate gaze direction. The person detector is capable of detecting a person from the front, side and back. The face detector can detect a frontal face, even in the presence of facial accessories such as glasses and hats.

Currently the person detection, face detection and head pose estimation are implemented in separate scripts that process a single image (or a list of images in the case of the person detection script). These need to be incorporated into a single script that can process a video. The frame to frame tracking included in the second algorithm design must also be implemented. To date the experimental validation of the system has been limited. The completed software needs to be tested more thoroughly under various operating conditions. For example, dim and bright lighting conditions and multiple people in view at a time occluding each other. Finally, the software needs to be deployed to the Raspberry Pi so its performance can be tested.

APPENDICES

OPENCV PERSON DETECTION [12]

```python
#import the necessary packages
from __future__ import print_function
from imutils.object_detection import non_max_suppression
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2

#construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--images", required=True, help="path to images directory")
args = vars(ap.parse_args())

#initialise the HOG descriptor/person detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

#loop over the image paths
for imagePath in paths.list_images(args["images"]):
    #load the image and resize it to reduce detection time and improve detection accuracy
    image = cv2.imread(imagePath)
    image = imutils.resize(image, width=min(400, image.shape[1]))
    orig = image.copy()

    #detect people in the image
    (rects, weights) = hog.detectMultiScale(image, winStride=(4, 4), padding=(8, 8), scale=1.05)

    #draw the original bounding boxes
    for (x, y, w, h) in rects:
        cv2.rectangle(orig, (x, y), (x+w, y+h), (0,0,255), 2)

    #apply non-maxima suppression to the bounding boxes using a fairly large overlap threshold
    #to try to maintain overlapping boxes that are still people
    rects = np.array([[x,y,x+w,y+h] for (x,y,w,h) in rects])
    pick = non_max_suppression(rects, probs=None, overlapThresh=0.65)

    #draw the final bounding boxes
    for (xA, yA, xB, yB) in pick:
        cv2.rectangle(image, (xA, yA), (xB, yB), (0,255,0), 2)

    #show some information on the number of bounding boxes
    filename = imagePath[imagePath.rfind("/") +1:]
    print("[INFO] {}: {} original boxes, {} after suppression".format(filename, len(rects), len(pick)))

    #show the output images
    cv2.imshow("Before NMS", orig)
    cv2.imshow("After NMS", image)
    cv2.waitKey(0)
```


OpenCV Face Detection [16]

```
# load the library using the import keyword
# OpenCV must be properly installed for this to work. If not, then the module will not load with an
# error message.

import cv2
import sys

# Gets the name of the image file (filename) from sys.argv

imagePath = sys.argv[1]
cascPath = "haarcascade_frontalface_default.xml"

# This creates the cascade classifcation from file

faceCascade = cv2.CascadeClassifier(cascPath)

# The image is read and converted to grayscale

image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# The face or faces in an image are detected
# This section requires the most adjustments to get accuracy on face being detected.


faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(1,1),
    flags = cv2.CASCADE_SCALE_IMAGE
)

print("Detected {0} faces!".format(len(faces)))

# This draws a green rectangle around the faces detected


for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow("Faces Detected", image)
cv2.waitKey(0)
```


OpenCV Head Pose Estimation [17]

```python
#!/usr/bin/env python
import cv2
import numpy as np

# Read Image
im = cv2.imread("headPose2a.JPG")
size = im.shape
#2D image points. If you change the image, you need to change vector
image_points = np.array([
                            (314, 279),#(359, 391),     # Nose tip
                            (320, 416),#(399, 561),     # Chin
                            (224, 258),#(337, 297),     # Left eye left corner
                            (366, 255),#(513, 301),     # Right eye right corne
                            (270, 350),#(345, 465),     # Left Mouth corner
                            (348, 352)#(453, 469)      # Right mouth corner
                        ], dtype="double")
# 3D model points.
model_points = np.array([
                            (0.0, 0.0, 0.0),             # Nose tip
                            (0.0, -330.0, -65.0),        # Chin
                            (-225.0, 170.0, -135.0),     # Left eye left corner
                            (225.0, 170.0, -135.0),      # Right eye right corne
                            (-150.0, -150.0, -125.0),    # Left Mouth corner
                            (150.0, -150.0, -125.0)      # Right mouth corner

                        ])
# Camera internals
focal_length = size[1]
center = (size[1]/2, size[0]/2)
camera_matrix = np.array(
                         [[focal_length, 0, center[0]],
                         [0, focal_length, center[1]],
                         [0, 0, 1]], dtype = "double"
                         )
dist_coeffs = np.zeros((4,1)) # Assuming no lens distortion
(success, rotation_vector, translation_vector) = cv2.solvePnP(model_points, image_points, camera_matrix, dist_coeffs)
# We use this to draw a line sticking out of the nose
(nose_end_point2D, jacobian) = cv2.projectPoints(np.array([(0.0, 0.0, 1000.0)]), rotation_vector, translation_vector, camera_matrix, dist_coeffs)
for p in image_points:
    cv2.circle(im, (int(p[0]), int(p[1])), 3, (0,0,255), -1)

p1 = ( int(image_points[0][0]), int(image_points[0][1]))
p2 = ( int(nose_end_point2D[0][0][0]), int(nose_end_point2D[0][0][1]))

cv2.line(im, p1, p2, (255,0,0), 2)

# Display image
cv2.imshow("Output", im)
cv2.waitKey(0)
```

BIBLIOGRAPHY

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005, vol. I, no. 3, pp. 886–893.

[2] E. Murphy-Chutorian and M. M. Trivedi, "Head pose estimation in computer vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 4, pp. 607–626, 2009.

[3] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," in *CVPR*, 2001.

[4] R. A. Maschal, S. S. Young, J. Reynolds, K. Krapels, J. Fanning, and T. Corbin, "Review of Bayer Pattern Color Filter Array (CFA) Demosaicing with New Quality Assessment Algorithms," 2010.

[5] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[6] Techquickie, "Image Sensors as Fast As Possible," *youtube.com*, 2014. [Online]. Available: https://www.youtube.com/watch?v=2ZXamWYdUgQ. [Accessed: 08-May-2019].

[7] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Pearson Prentice Hall, 2009.

[8] M. Nielsen, *Neural Networks and Deep Learning*. Determinaion Press, 2015.

[9] 3Blue1Brown, "But what *is* a Neural Network? | Chapter 1, deep learning," *Youtube*, 2017. [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk. [Accessed: 05-May-2019].

[10] K. P. Bennett and C. Campbell, "Support Vector Machines : Hype or Hallelujah," *SIGKDD Explor.*, vol. 2, no. 2, pp. 1–13, 2000.

[11] S. Patel, "Machine Learning 101 Chapter 2 : SVM (Support Vector Machine) — Theory – Machine Learning 101 – Medium," *medium.com*, 2017. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. [Accessed: 09-May-2019].

[12] S. Mallick, "Histogram of Oriented Gradients," *Learn OpenCV*, 2016. [Online]. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/. [Accessed: 09-May-2019].

[13] doxygen, "OpenCV: Face Detection using Haar Cascades," *opencv.org*, 2018. [Online]. Available: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html. [Accessed: 09-May-2019].

[14] J. Sherrah, J. Ong, and S. Gong, "Estimation of Human Head Pose using Similarity Measures," *Queen Mary University of London, School of Electronic Engineering and Computer Science*, 2000. [Online]. Available: http://www.eecs.qmul.ac.uk/~sgg/pose/. [Accessed: 13-Mar-2019].

[15] M. Ariz, J. J. Bengoechea, A. Villanueva, and R. Cabeza, "A novel 2D/3D database with automatic face annotation for head tracking and pose estimation," *Comput. Vis. Image Underst.*, vol. 148, pp. 201–210, Jul. 2016.

[16] V. Tabora, "Face Detection Using OpenCV With Haar Cascade Classifiers," *becominghuman.ai*, 2019. [Online]. Available: https://becominghuman.ai/face-detection-using-opencv-with-haar-cascade-classifiers-941dbb25177. [Accessed: 11-May-2019].

[17] S. Mallick, "Head Pose Estimation using OpenCV and Dlib | Learn OpenCV," *Learn OpenCV*, 2016. [Online]. Available: https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/. [Accessed: 11-May-2019].