UNIVERSITY OF EAST LONDON, UK

ENGINEERING PROGRAMMES

Counting people using a monocular camera for emergency building evacuation scenarios

Iason A. Tzanetatos

1342174

BEng (Hons) Electrical and Electronic Systems

EG6140: Project Report

Department of Electrical, Electronic and Mechanical Engineering

Faculty of Engineering and Architecture

Marousi Campus, Athens

2017/18

Abstract

A lot of different solutions, from simple to sophisticated, have been implemented in order to detect and classify by computational means, a variety of objects, depending on the application. This thesis examines the application of a stationary camera, responsible for counting people entering a building, in order to be utilized in evacuation scenarios.

The algorithm, developed in the MATLAB environment, tackles issues such as backgrounds subtraction, image filtering, morphological operations, contour extraction and image segmentation. The starting goal was to implement an unsupervised algorithm, specifically k-means clustering, in order to classify the present objects.

Since no such result has been achieved, hopefully this paper will provide some assistance to a similar implementation.

Acknowledgements

This thesis, while disappointingly not up to my standards, hopefully will be a symbolic way to validate the hopes and expectations that each person who supported me throughout my studies, hoped for me to achieve.

I would like first and foremost to express my deepest gratitude to my parents who made their own personal sacrifices in order to achieve this degree.

I would like to express my sincere gratitude to my supervisor, Dr. Georgios Chliveros, who throughout these years, put up with my frustrating behaviour, recognised my potential and helped me to achieve this paper.

I would like to thank each and every professor throughout this degree, each and every one of you have a unique way of influencing people in a positive way.

I would like to express my biggest gratitude and appreciation to A. Tsairi who supported me throughout the years and kept me going.

My biggest gratitude and appreciation goes to K. Tsagari for all of the support that she's provided me in order for this paper to become a reality.

To my friends who supported me in their own way.

My appreciation to my classmates who kept me company along this degree.

Table of Contents

List of Figures

List of Appendices

1 Introduction

1.1 General

The field of Computer vision has experienced an enormous exponential improvement in the latest years. The ability to detect, classify, augment reality and even, in some cases, even predict outcomes is a fascinated technology. People already benefit a lot from that field of technology, whether it is through social media, video sites, traffic analysis cameras, implementations on medical fields or even in semiautonomous self-driving cars such as the Tesla car. Most applications have in mind the detection and recognition of humans.

Nowadays with the all increasing computation power that is accessible to everyone, people find every day even more applications for this technology, hence the newfound tendency to utilize neural networks. However, some applications to this day have some room for improvement. One such complex problem is the human detection. While the field has achieved an impressive outcome with the current state-of-the-art implementations, there is still some room for improvement, for example reducing false positive detections and minimize incorrect readings, although it has reached, in some cases, the point of outperforming humans in detecting and classifying objects.

Important implementations as critical failure features still pose questions such as which methodology will be implemented, and why. One very important application is the detection and counting of people entering a building, which is extremely helpful to know the exact number of people inside the building, in the unfortunate case of emergency, demanding building evacuation.

1.2 Thesis overview

This thesis studies the implementation of a robust human detection algorithm from a still camera, in order to count the number of people present in every frame of the video feed. The algorithm is created in the MATLAB software, utilizing some of its built-in functions, present in the Image Processing Toolbox, and other functions utilized from scratch. Implementations such as Fast R-CNN (Regions with Convolutional Neural Networks), in combination with other machine learning algorithms are state of the art. In comparison, this thesis takes a more simplistic, robust approach by utilizing mathematics that have been developed since decades.

The big picture in this particular development, consist of three main topics.

The first topic is to determine and subtract the background from the foreground, in order to capture the useful information that may or may not be a human present in the product image of this procedure. This process will be achieved by utilizing fairly simple mathematical formulas and pixel manipulations.

The second process is capturing the contour of every object that is present and closing its silhouette. This is a very important process that has to be executed correctly and efficiently, because it will be the stepping stone into extracting useful information that are

present in the frame, in order to continue to the final process. Problems such as clustering of objects, meaning the algorithm of feature extraction considers two objects as one object (as shown below), will be addressed in this process. There are many approaches about feature extraction, it depends, however, on the given implementation, the methodology of choice. Features such as lines, edges, curves or texture are most of the time utilized in either production lines or in medical applications. This particular application, does not require such detailed features, it only requires the closed contour and area measurement. To achieve that, morphological operators, which its functions and mathematics behind them will be explained further in the design of the algorithm, and other properties of labeled binary objects are employed.

The final process is the classification of each present object. Since it cannot be known beforehand what kind of objects will be present in the image, the algorithm has to classify each object to a category.

1.3 Machine learning

There are three main categories to classify detected objects. One is through supervised learning, which the algorithm is fed already classified objects, and based upon the labeled object's properties, whether they may be shape, texture, surface area, the algorithm actively searches for similar patterns. One such example is decision trees, which consists a number of decision rules "as a predictive model in order to connect observations about an item to conclusions about its properties" (Mitropoulos S.). [12]

The semi supervised machine learning, is similar in concept with the supervised learning. One drawback of supervised learning, is the time-consuming task of labeling data for the algorithm to train. One way of speeding up this procedure is feeding the algorithm with both labeled and unlabeled data to train. One such example is the transductive support vector machines algorithm.

Finally, there is the unsupervised machine learning approach, which the algorithm trains on unlabeled data, in order to find a pattern to label the data. One such example is the k-means clustering algorithm. This thesis would attempt to utilize this algorithm in order to classify the unknown type of objects that may be present.

All of the above methods of classification can be being utilized either as standalone or as a part of a bigger system of classification. One such case that utilizes both supervised and unsupervised classification algorithms is the YOLO9000 algorithm, utilizing both Neural Networks and k-means clustering.

2 Literature Review

By utilizing advanced algorithms that are classified as artificial intelligence, specifically neural networks, the success rate for detecting and identifying a number of objects has been impressive, and in some cases, yields results faster and more accurate than of the human's eyes. Nowadays, the state-of-the-art in computer vision tends to lie on algorithms such as

‘Regions with Convolutional Neural Networks’ (R-CNN) and its successor algorithm, Fast R-CNN, in order to process the image and extract the features that are deemed as useful. However, despite the usage of neural networks, more robust mathematically driven implementations continue to dominate the field of Computer Vision. Neural networks have their own drawbacks such as the complexity of their design, that renders them challenging to implement, debug and fine-tune them. However, they provide an excellent real-time application, with notable examples the YOLO9000 and the Fully Convolutional Networks.

This thesis implements a more robust approach as mentioned above, therefore it only references the above solutions.

3 Algorithm Development

This thesis uses the PETS 2009 dataset in order to develop and test the performance of the proposed algorithm. There is also a traffic video that contains, besides pedestrians, vehicles and cyclists in some frames in order to have a broader review of the proposed algorithm. The PETS 2009 data has been manually labeled in order to evaluate the performance of the algorithm with that of a human eye. Since the dataset contains 97 videos, the results will be published in a separate volume. Some noteworthy results that showcase the initial stages, as well as the results of the algorithm will be presented below. This section of the paper will explain the algorithm step-by-step and also provide the necessary theoretical background and mathematical equations.

3.1 RGB Colour Format and Image Representation in MATLAB

The way that MATLAB views images in its environment as a N by M by 3, array $[(N \times M) \times 3]$, otherwise as a 3-D array. The reason behind the "by 3" multiplication, has to do with the RGB colour format, containing for each colour channel (Red, Green and Blue) intensity values that range from 0 to 255. MATLAB uses the uint8 (unsigned 8-bit integer) array type for reading images.

To further understand the RGB colour, it is necessary to note some useful information in regards to that particular colour format.

3.1.1 RGB Colour Format

Images that utilize the RGB colour format, can be viewed as 3-D arrays, each array containing a value for each pixel corresponding to the level of the channel colour, whether it may be Red, Green or Blue. Their combinations produce the final output that we humans perceive as colour. For example, an object that contains only blue colours, may well in fact be represented only in the array that contains the blue colour values. By applying the same logic on an object or scenery represented as a mixture of colours, it is represented as a combination of different values from the red green and blue arrays.

Each colour channel can contain values from 0 to 255, assuming it is utilizing the standard "simplified RGB colour model (based on the CIE colour standard of 1931) that is optimized and standardized towards graphical displays)" (Solomon C., Breckon T). [14]

For better understanding, one can consider a visual representation of the RGB colour as a cube, shown below. [11]

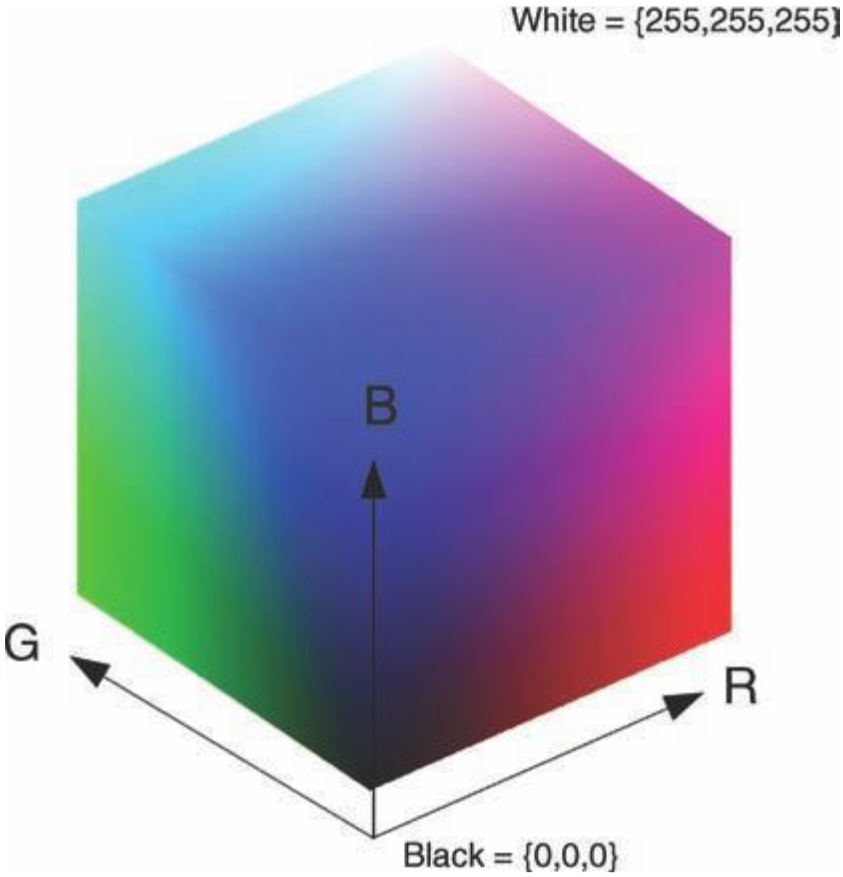

Figure 3.1 Cube representation of the RGB colour [14]

According to Shapiro, "the RGB system is an additive colour system because colours are created by adding components to black: (0,0,0)". [10] In that regard, taking the visualization of the RGB cube, it is possible, as Gonzalez R., et. Al. mention, to consider the values of the cube as Cartesian coordinates. [4]

Since the development environment of MATLAB utilizes the above information, a visualization is deemed necessary for the reader to fully comprehend the way MATLAB views RGB images.

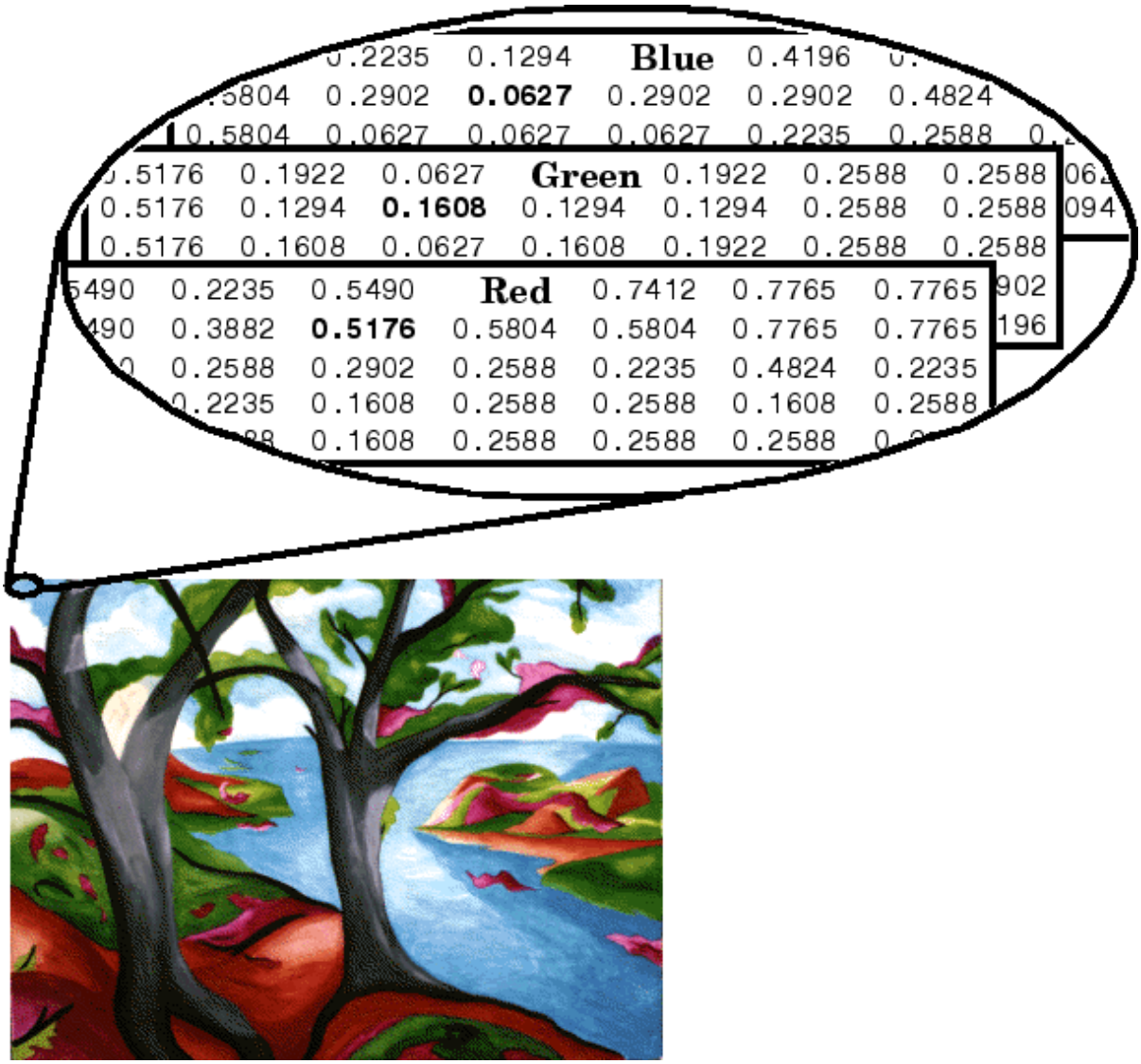

Figure 3.2 Array visualisation for each colour channel [7]

The figure above demonstrates the representation of the pixels in the image in the three colour channels.

3.2 MATLAB Implementation for Video Input

A video file, can be perceived as a collection of multiple images that are being projected at a very fast rate in order to create the 'illusion' of movement. Hence, when a video file is introduced into the MATLAB environment, it can be processed frame-by-frame, in simpler terms, by constantly feeding the input with images (video frames).

```
% Video file selection dialog box
[filename,pathname] = uigetfile({'*.avi';'*.mp4'},'Select the video file');

% Construct a multimedia reader object associated with avi files.
vidObj = VideoReader(fullfile(pathname, filename));
```

By utilizing the [filename, pathname] function, a prompt is asking the user to select aa video file. The file format is specified inside the brackets, in this specific case the algorithm will accept video files that are either .avi or .mp4 format. The *filename* variable contains the name of the video file, and the *pathname* contains the path that the selected video file is stored. Afterwards the *VideoReader* function creates a multimedia reader object that contains the video file.

In order to be able to read each frame contained in the video file, the following function is necessary.

```
% read first frame as background
bg = readFrame(vidObj);
```

The algorithm sets the variable *bg* as background, utilizing the *readFrame* function to read the first frame as the background frame.

Figure 3.3 Background frame example from PETS 2009

3.2.1 RGB2Gray MATLAB Function

In order to proceed with the algorithm, it is extremely helpful to convert the background image (variable *bg*) to greyscale. To achieve that result the MATLAB function *rgb2gray* is being utilized as shown below.

```
bg_bw = rgb2gray(bg);            % convert background to greyscale
```

The reasoning behind the conversion from RGB to greyscale is simple in concept. Although it may seem that a lot of information is lost, in actuality, as Solomon C. et. Al. note, "the majority of important, feature-related information is maintained such as edges, regions, blobs, junctions and so on." [14] It is fairly common practice for algorithms that extract or process features to operate on greyscale images and the reasoning behind it is that it simplifies and speeds up the process since the algorithm has a 2-D vector$[(N \cdot M) \cdot 2]$ to compute. The mathematical formula for converting an RGB image to greyscale as Solomon C. et. Al. notes is the following:

$$I_{grey\text{-}scale}(n, m) = \alpha I_{colour}(n, m, r) + \beta I_{colour}(n, m, g) + \gamma I_{colour}(n, m, b) \quad (3.1)$$

"Where (n, m) indexes an individual pixel within the grey-scale image and (n, m, c) the individual channel at pixel location (n, m) in the colour image for channel c in the red *r*, blue *b*, and green *g* image channel." [14] It can be concluded, as Solomon C. et. Al note, that the greyscale conversion is a weighted sum of the colour channels. However, since the human eye is more sensitive to red and green light, "the weighted coefficients (α, β and γ) are set in proportion to the perceptual response of the human eye", therefore by utilizing the NTSC television standard where α = 0.2989, β = 0.58777 and γ = 0.1440, the equation becomes:

$$I_{grey-scale} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \quad (3.2)$$

which is the same conversion equation that MATLAB utilizes in rgb2gray function. [14]

Figure 3.4 Grey-scaling of the background frame

It should be noted that the conversion from RGB to greyscale is, as Solomon C., et. Al. point out, “is a noninvertible image transform: the true colour information that is lost in the conversion cannot be readily recovered.” [14] However, in the MATLAB environment, by utilizing the *imread map* feature, a matrix that stores the intensity values for each pixel, one can revert back to RGB colour image.

3.2.2 Output Settings and Visualizing Results

It was deemed as a useful feature to include a prompt, asking the user if they prefer to start the video at a later timeframe of their choosing, if they prefer to visualize the end result of the process and how they would like to name the output video file.

```
%% dialogue box for setup variables

prompt = {'Video Feed Start Read Time (sec):',...
    'Results Visualization (no: 0 / yes: 1)',...
    'Output Video Filename:'
    };
dlg_title = 'Input';
num_lines = 1;
defaultans = {'0','1',filename '_out'};
answer = inputdlg(prompt,dlg_title,num_lines,defaultans);
```

The algorithm assumes as default values the start time as 0 seconds, that the user wants to visualize the results, and for the name of the output file, the same as the input file and adding "_out" at the end of the name.

3.2.3 Setting of Video Viewer and Frame Size Variables

In this step of the algorithm some necessary variable pre-allocations to occur.

```
%% ----------------------- set video viewer and frame size variables -------
----------------
width = vidObj.Width;
height = vidObj.Height;
fg = zeros(height, width); % Create dummy variable to store foreground
values
% if plot image is required for visualization
flag = str2num(answer{2});
if flag == 1
    fflag = figure;
    figure(fflag);
    set(fflag, 'Position', round( get(0, 'Screensize')));
end

% You can specify that reading should start at n seconds from the
% beginning using
starttime = str2num(answer{1});
vidObj.CurrentTime = starttime;


filename = answer{3};
framerate = vidObj.FrameRate;
if flag==1
        writerObj = VideoWriter([filename,'.avi']);
        writerObj.FrameRate = framerate;
        open(writerObj);
end
```

The variables *width* and *height* store a single number that is gathered from the video file properties and corresponds to the video width and height as the variable names imply. The variable *fg* is a matrix filled with zeros and its dimensions are the same with the input video's dimensions by utilizing the *height* and *width* variables. Afterwards, if the user chooses to visualize the algorithm's output, the *if loop* is executed by setting the *flag* variable value to 1 and sets the position and the size of the output figure. The *starttime* variable contains the value that the user defined in regards of the start time of the video file, to be utilized by the *VidObj.CurrentTime* variable. The variable *filename* contains the answer that the user gave in regards of the name of the output video file and the *framerate* variable, necessary for setting the video output framerate is also given by the *vidObj.FrameRate* property of the input video file. If the user chooses to record the output, by setting the *flag* variable to 1, the

if loop is executed, and the *VideoWriter* function is utilized, outputting an .avi video file with the same framerate as the input video file.

3.3 Frame Processing

Since the background image and the necessary variables and user-based settings have been established, as demonstrated and explained above, the main piece of code that is responsible for processing each frame is ready to run.

It begins by setting some variables that are being utilized as global threshold, neighborhood for frame dilation and neighborhood for the median filter. All of these functions will be further explained below.

```
thresh = 25;     % threshold for pixel value to consider as
background/foreground
ns1 = 15; % neighborhood for frame dilation (dilation not present, only
usage in imclose)
nsNN = [7, 7]; % neighborhood window for median filter
```

Afterwards, a counter is pre-allocated to verify the number of frames the algorithm has processed.

```
counter = 0; % this is a counter / useful to know when using a while-
loop
```

The algorithm in that point steps into a while loop that runs for every frame that is present in the video file by utilizing the MATLAB function:

```
while hasFrame(vidObj)
```

The program at this point reads and converts the current frame from RGB to grayscale, as explained above, by the following lines of code:

```
    fr = readFrame(vidObj);     % read current frame
    fr_bw = rgb2gray(fr);       % convert frame to grayscale
```

The *fr* variable is considered the foreground frame and the *fr_bw* variable has the foreground frame stored in the form of greyscale.

3.3.1 Background Subtraction

In order to subtract the background for the foreground, the function imBackSub was written. The outputs of this function are the foreground variable *fg*, and the variable *bg_bw* that is constantly being updated, for reasons that will be further explained below. The input variables for this function are the *fr_bw*, *bg_bw* and *threshold*. The following processes occur:

```
[height, width] = size(bg_bw);

fr_diff = abs(double(fr_bw) - double(bg_bw));  % cast operands as
double to avoid negative overflow

thresh = threshold;
```

The variables *height* and *width* contain the actual height and width of the *bg_bw* frame which will be utilized below. The *fr_diff* variable performs the absolute difference between the foreground image and the background image, stored as double data types to avoid negative overflow and have corrupted areas as a result. The variable *thresh* contains the value of the variable *threshold* which is 25 and it is utilized below. The explanation of this methodology of background subtraction will be detailed below.

```
for j=1:width
        for k=1:height
            if ((fr_diff(k,j) > thresh))% if fr_diff > thresh, then
pixel in foreground
                fg(k,j) = fr_bw(k,j);
            else
                fg(k,j) = 0;
            end
        end
end

bg_bw = fr_bw;
```

Afterwards, as shown above, the function enters into a *for loop*, starting from *1* until it reaches the actual number of *width* and for each value, it enters into another nested *for loop* that runs from *1* until it reaches the actual number of *height*. At that point it enters into an *if/else loop* that compares the value the *fr_diff* variable has present in the *(k, j)* position, where *k = height loop counter* and *j = width loop counter*, with the *threshold* value *25*. If the current *fr_diff* value is above the threshold value, it is considered *foreground* and it gains the intensity values of that position *(k, j)* from the *fr_bw* variable. Else it is considered background and the intensity value for that position is set to zero. The algorithm then sets the *fr_bw*

variable as the next *bg_bw* variable, meaning that the background image is constantly updated to have the value of the previous foreground frame.

This algorithm for background subtraction utilizes a number of notable techniques. The basic principle about background subtraction is as follows. Consider an image that has a unified dark colored background, and in it there is a light-colored object that humans and machines alike can clearly distinguish since the intensity values of the background and the object are vastly different. According to Gonzalez R. and Woods R., one methodology of achieving that separation of object from background is by utilizing a threshold value. By perceiving the coordinates of a pixel in an image as $f(x, y)$, the threshold can be utilized in that specific pixel by performing an intensity comparison. [4] More precise, if at any pixel in the image the relation $f(x, y) > T$, where T the threshold value, is true, then that specific pixel is considered, as Gonzalez R. and Woods R. refer to, "an object point"; if the relation is false, then the pixel is considered a "background point". [4] To further illustrate the mathematical relation of thresholding, Gonzalez R., and Woods R. present the relation below. [4]

$$g(x, y) = \begin{cases} 1, & if\ f(x, y) > T \\ 0, & if\ f(x, y) \leq T \end{cases} \tag{3.3}$$

If the variable T has a constant value throughout the image analysis, it is referred, as Gonzalez R. and Woods R. point out, as a "global thresholding". [4] There are, however different approaches where the variable T has a non-constant value that is based on the different intensities that are present in the surrounding pixels of the selected pixel for processing (otherwise noted as properties of a neighborhood as Gonzalez R. and Woods R. note) and that type of thresholding is known as local or regional thresholding; another way of a non-constant valued threshold T, is the adaptive or dynamic thresholding as Gonzalez R. and Woods R. define, and its value has a dependency upon the "spatial coordinates (x, y) themselves". [3] There are other ways that background subtraction can be achieved without the usage of thresholding. If the background is a still image with constant intensity values, meaning the optical sensor is not in motion and the illumination is constant, then everything with different intensity value will be considered a foreground object. Mitropoulos S. has utilized this approach on his thesis, since his data samples where captured in a controlled environment, with impressive results. [12] There are however cases where background

subtraction is a complicated challenge, especially when the optical sensor is in motion, the background constantly changes and there is the need of depth in order to detect an object in the foreground. In that case, more sophisticated implementations are in order to achieve the desirable results. Such cases will not trouble this paper, however papers such as “YOLO9000: Better, Faster, Stronger” and papers that utilize Convolutional Neural Networks, as referenced below, tackle such challenges with very impressive results.

This paper, based on the above piece of code for background subtraction and based on the above considerations in regards to thresholding, utilizes a constant threshold for background subtraction; this approach is referred as global thresholding. The global threshold is applied throughout the entire image and compared with the absolute difference of the foreground/background. However, this isn’t an effective way to threshold an image, since the global threshold, while simple in principle, lacks adaptation and adaptation is an important factor to consider.

A solution for this issue is proposed by Gonzalez R. and Woods R., in order to estimate automatically the value of the global threshold. The first step is, as Gonzalez R. and Woods R. mention, to “select an initial estimate for the global threshold T.” As the authors note further below, “the average intensity of the image is a good initial choice for T”. [4]

Afterwards by utilizing the mathematical equation (something), the products are two categories of pixels, pixels that their intensity value is greater than the threshold, therefore foreground pixels, and pixels that their intensity value is less than-or equal to the threshold value, therefore background pixels. The next step is to calculate the average values of each group of pixels and calculate the threshold value using the formula above:

$$T = \frac{1}{2}(m_1 + m_2) \qquad (3.4)$$

Where m_1 and m_2 the foreground and the background mean respectively. Finally, the process of segmenting the background/foreground, calculating the mean for each pixel category and recalculate the threshold value by using the formula above shall be repeated until the differences between the newly calculated threshold and the previously calculated threshold fall into the specific difference the users deems as acceptable. In order to achieve that, the authors note that a control parameter “ΔT” is utilized to control the number of calculations for the threshold. As they note “the larger ΔT is, the fewer iterations the algorithm will perform.” [3]

In this particular implementation, a constant global threshold was defined after a trial and error procedure took place. The product of the background subtraction function can be seen in the figure below.

Figure 3.5 Background Subtraction result containing only Foreground objects

3.3.2 Contour Capture

In order to be able to detect and classify people, the feature that is needed is the contour of each object that is present in the current frame. As it can be noted by the figure above, there are several issues that pose issues with the extraction of the silhouettes. It can also be noted the binarization of the image through the background subtraction function. However, that is not the case; while the array was converted to a 1-D array and the background intensity values are 0, the intensity values for the foreground items posses values greater than 1. That does not pose any issues with the chosen implementation. In this stage there are two challenges to be addressed: the noise, present from the changes of the illumination and from the global threshold, and the closing of the silhouettes in such a way that no useful information will be lost. Objects that are deemed as “useless” will be addressed in a later solution further below.

To filter the noise from each frame, a median filter was utilized to smooth out any random noise, by utilizing the following MATLAB function

```
fg = medfilt2(fg, [7, 7]);
```

The medfilt2 function takes as an input the fg image, and a 7×7-pixel neighborhood was specified. The median filter is part of the Order-Statistic Filters, which in turn are part of spatial filters. The median filter operates, as its name suggests, by finding the median value in a predefined neighborhood of pixels. Nixon M. illustrates very clearly the process behind the median function.

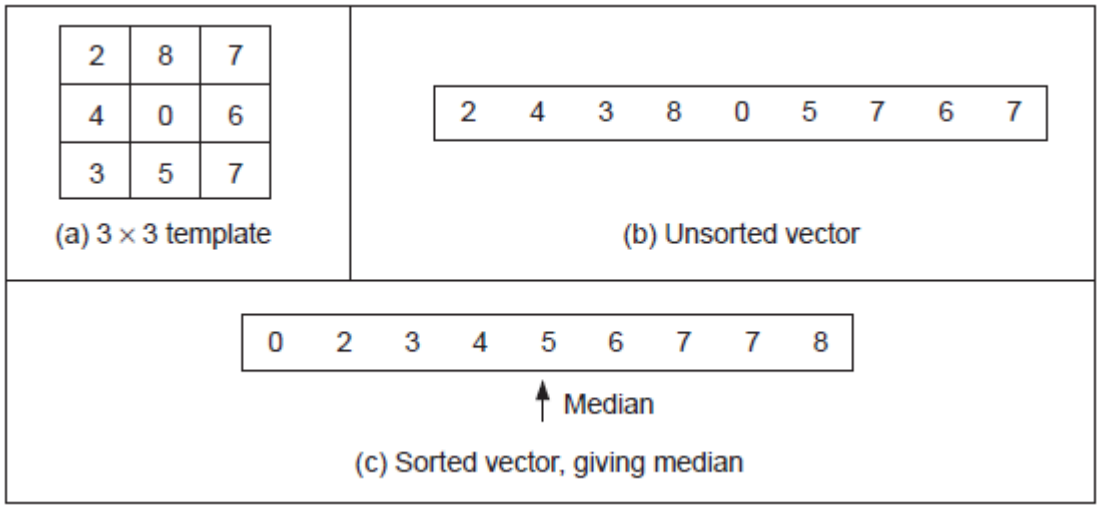

Figure 3.6 Median filter visualisation [8]

The median filter, as Solomon C. and Breckon T. note "as each pixel is addressed, it is replaced by the statistical median of its N×M neighborhood", "preserving sharp high-frequency detail whilst also eliminating noise, especially isolated noise spikes (such as 'salt and pepper' noise"; in this specific implementation, 'salt and pepper' noise is present in the form of small 'objects' that remain in the foreground due to illumination changes and the implementation of the global thresholding, therefore the median filter is deemed necessary in order to remove those objects. [14]

In order to capture the contour of each object present in the frame, since the vast majority of noise has been eliminated, three morphological operators are being utilized. Morphological operators can be applied to any image type, but it is common usage to apply this set of mathematical tools to binary images. The first step of this paper's approach, is the

filling of pixels that are surrounded, in the form of 8-connection, with 1's, while themselves are zero. A visualization, for better understanding of this procedure is as follows.

The pixels that are selected for this process are in the form of:

```
1  1  1
1  0  1
1  1  1
```
(3.5)

After the process of the filling, the pixels in the center, become 1. The function for this process is as below.

```
fg = bwmorph(fg, 'fill');
```

This achieves not only the filling of the individual pixels, but also binarizes the image. However, since no thresholding solution is implemented for the binarization, the clustering issue persists.

In order to close the contours, a morphological structuring element in combination with the morphological closing of the image (erosion followed by dilation) is being implemented. A morphological disk is created by using the following function in MATLAB.

```
st = strel('disk', ns1);
```

This creates a disk shaped morphological structure with radius ns1; in this particular case the radius value is 15

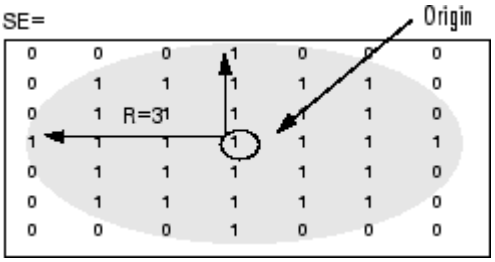

Figure 3.7 Structuring Element visualization [7]

The morphological closing of an image is the process of erosion followed by dilation, in order to eliminate small objects (erode), and then fill the entire contour with 1's so that will become one solid object. Solomon C. and Breckon T. note "To perform erosion of a binary image, we successively place the centre pixel of the structuring element on each foreground pixel (value 1). If any of the neighbourhood pixels are background pixels (value 0), then the foreground pixel is switched to background. Formally, the erosion of image A by structuring element B is denoted A⊖B".

For erosion they note “To perform dilation of a binary image, we successively place the centre pixel of the structuring element on each background pixel. If any of the neighbourhood pixels are foreground pixels (value 1), then the background pixel is switched to foreground. Formally, the dilation of image A by structuring element B is denoted A ⊕B”.

Another interesting fact they note, is that every morphological operator, essentially can be performed by the combination of erosion and dilation. [14]

By morphologically closing the image with the usage of the structuring element st, utilizing the below piece of code, the result is shown in Fig. 3.7

```
fgc = imclose(fg, st);
```

Figure 3.8 Contour capture after morphological closing of the frame

3.3.3 Image labeling and Object counter

In order to be able to differentiate between n-number of objects that are present in the frame, the function *bwlabel* is being implemented as shown below.

```
[CC,num] = bwlabel(fgc,conncomp);
```

The way that this function operates is, the connectivity of the objects is specified, in this case an 8-neighbor connectivity is utilized, and the function sets 1’s to the first object

that it encounters with an 8-neighbor connectivity, 2's to the second object, and so on, for n times, where n is the number of objects that are present in the frame. The numbering of the objects is executed from top left to bottom right. The outputs of the function are the labeled matrix CC and the number of connected components present in the current frame, num. In order to better visualize and showcase the number of objects the following algorithmic approach is utilized.

```
for i = 1:num
    [r, c] = find(CC==i);
    szr = length(r);
    if szr > minsizeofCC
            objCounter = objCounter + 1;
            CC(r,c)= objCounter;
    else %if  szr < thresh
            CC(r,c)= 0;
    end
end
```

The function, enters into a *for loop* that runs for every object present in the current frame; the first object is selected and its coordinates are stored in the r and c variables, where in r the vertical coordinates are stored (considered otherwise as y coordinates), and in c the horizontal coordinates (considered otherwise as x coordinates). An *if loop* is then utilized in order to compare the length of the object's height with the value of a threshold, in that case the minsizeofCC with value 80. If the object's height is greater than the threshold's value, then a counter is utilized, starting from 1, and the labeled matrix in its y and x coordinates, gains the current value of the counter. Otherwise, meaning if the height of the object is lower than or equal to the threshold, the loop sets zero in the object's y and x coordinates.

Finally, the centroids from the labeled objects that have been reformed, are calculated by the regionprops function

```
s = regionprops(CC,'centroid');
```

This process is helpful in order to visualize the count of the objects present in the frame. The labeled matrix has the following form.

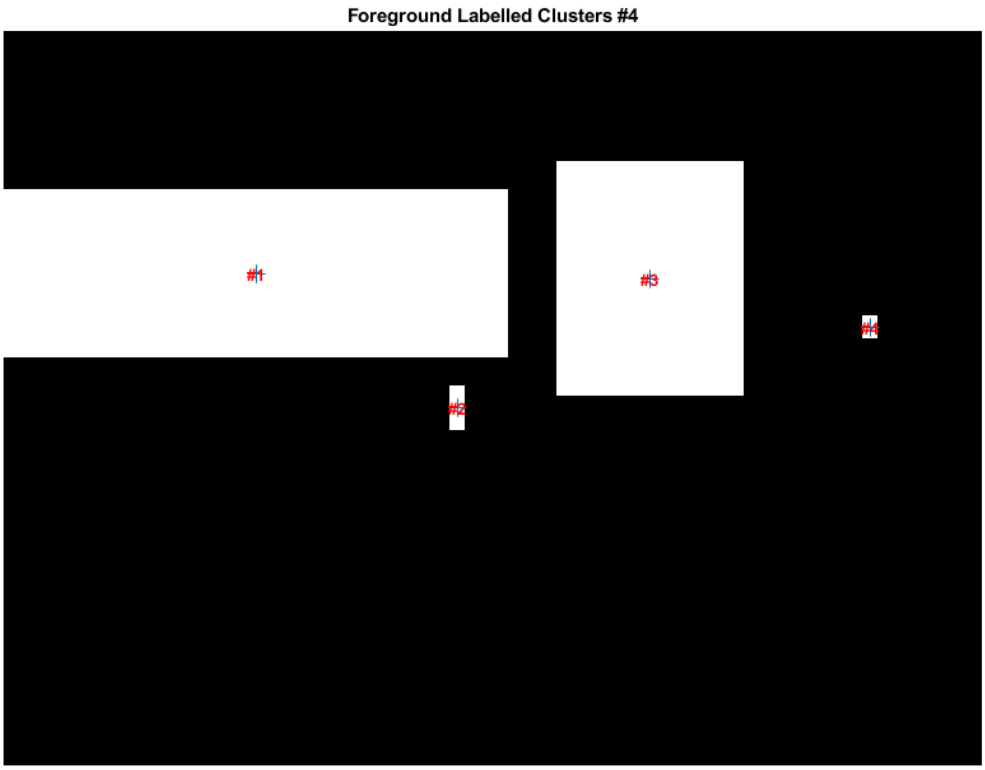

Figure 3.9 Labelled matrix CC with its centroids calculated and labelled

The reason this methodology was implemented was in order to find the centroids in more regular shapes, since the silhouettes have highly irregular shapes. Therefore, by utilizing the above centroids, the final product is as below.

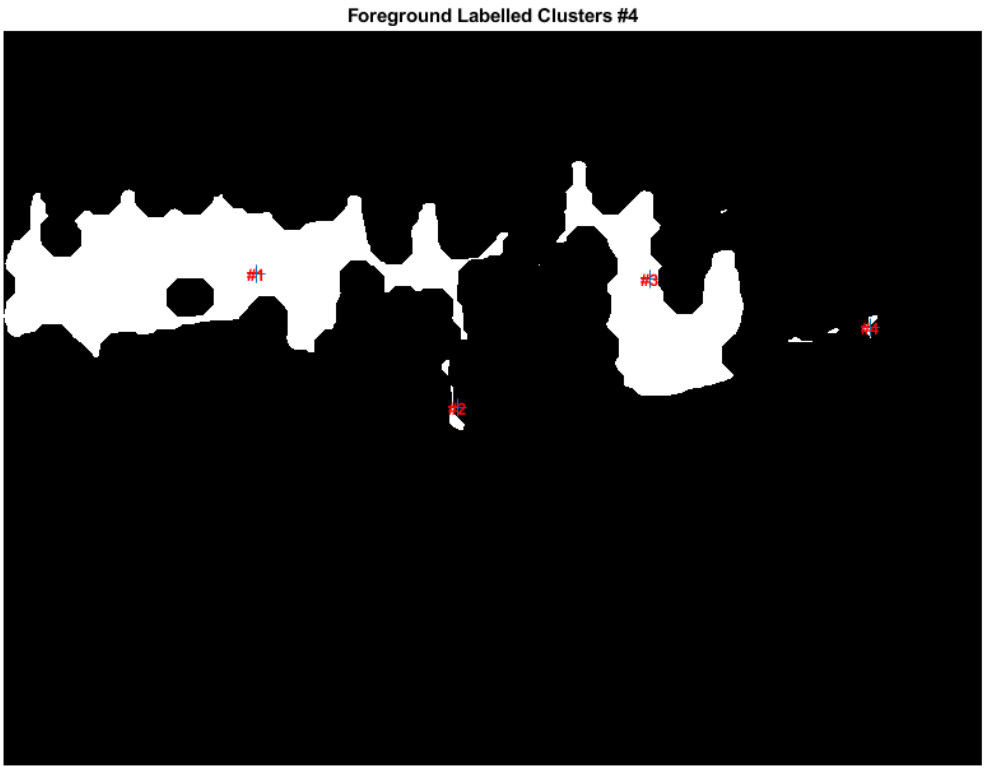

Figure 3.10 Counting of Present Objects

3.3.4 Cluster isolation and further processing

As noted above, while isolated people are detected, as shown below, there is a significant issue that needs to be addressed, and that is the clustering of two or more people together as one object.

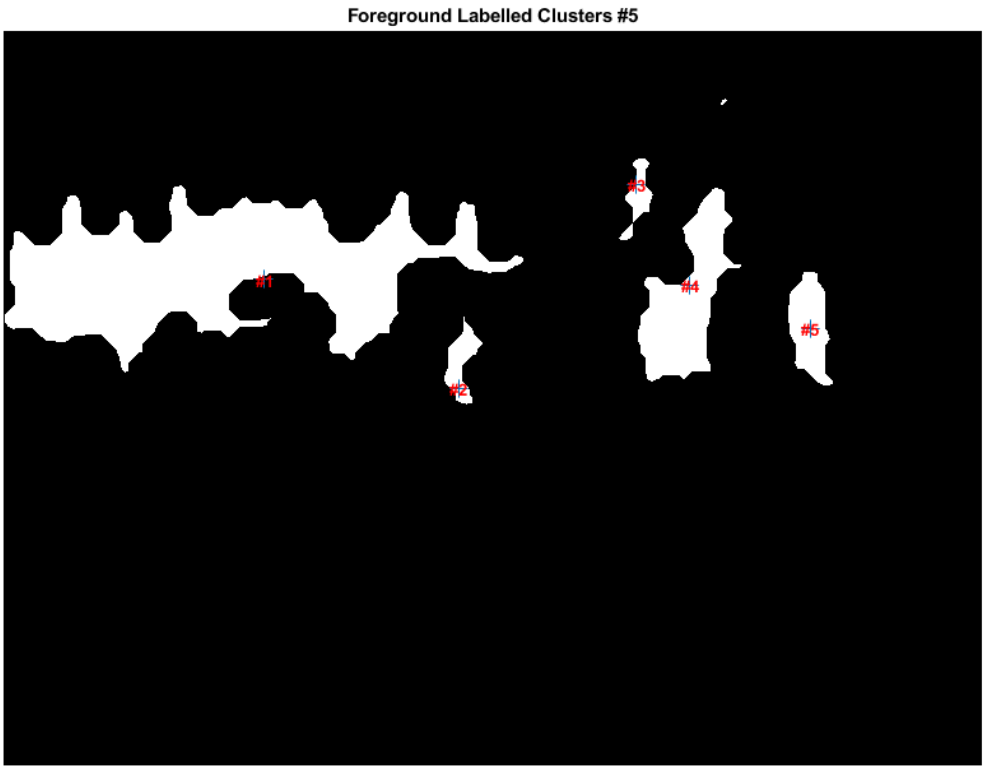

Figure 3.11 Example of a single person's contour (Object 5)

In order to tackle this issue, in principle, the clusters that are larger than one person have to be isolated and further processed. While the processing was not achieved, the separation of the clusters was.

The two main problems in this process is the identification of a cluster that contains two or more people, and its isolation.

In order to identify the clusters, the regionprops function was utilized in order to gather all the area measurements of all the objects. From there, the mean value, the most frequent value and the distance from the mean value were calculated by issuing the following piece of code.

```
allAreas = [getarea.Area]; % Store all area values into an array

meanvar = mean(allAreas); % Find the average/mean value
freqvar = mode(allAreas,2); % Find the most frequent value
distance = (allAreas - meanvar) / (num-1); % Calculate the distance
from the mean value
```

The functions utilized to calculate the mean and the most frequent value are *mean* and *mode* respectively. The reason that the function mode requires the argument '2' is in order to perform the calculation horizontally.

By hypothesizing two things, that the mean value is the same with the most frequent value, and that the most frequent value is the area of a single person, the following algorithm has been developed.

An algorithm that runs for every detected object, selects all the area values for each cluster, one cluster at every repetition of the loop; then the distance from the mean value is compared with the most frequent value, since in reality the mean value and the most frequent value are not the same, and if the distance measurement is greater than the most frequent value, it means that this cluster is larger than a single person's cluster. The area measurements and coordinates are temporarily stored into a variable that stores each cluster that fulfills the above criteria, and then a new variable sorts the measurements from largest to smallest. Finally, by utilizing another for loop, that runs for every cluster of interest that was detected. By utilizing a counter every time, the if statement was true, the clusters of interest are selected, one at a time.

The developed algorithm is the following:

```
k = 0; % If loop counter, useful to know how many times the if loop was
used
for a=1:num
    thisCentroid = [getarea(a).Area]; % Variable to select the current
cluster
        if distance(a) > freqvar
            k = k +1;
            clusterTemp(a) = thisCentroid; % Variable to select the
desirable clusters
            [sortedAreas, sortIndexes] = sort(clusterTemp, 'descend'); %
Store and index the selected clusters
            for j = 1:k
                biggestCluster = ismember(fgc, sortIndexes(k));
            end
        end
end
```

Since no further processing is being done on the clusters, there are no measurements to be shown.

4 Conclusions and future work

This paper approaches the human detection challenge by developing algorithms to subtract the background, filtering, contour capturing and blob separation. The remaining objects that where initially set, could not be pursued at this time, since the blob de clustering could not be achieved.

The results are promising for further development and probably, finalization of the initial goals. In summary the paper has achieved:

1) Background subtraction by utilizing a global threshold and binarizing the foreground image.
2) Utilizing a median filter that efficiently removes the vast majority of noise.
3) Morphological operators to close the silhouette and clusters.
4) Identification of surface area for a single human, and segmentation of clusters that contain more than one human.

In order to attempt to further segment the clusters that contain more than one person, a number of possible solutions were pursued with, unfortunately, no results. A further pursue of the possible solutions is highly recommended. The following proposals that have been attempted to a small extend and failed due to time restrictions, might contain the solution for that challenge:

1) Watershed morphological operation
2) Reverting the clusters to RGB arrays and applying spatial segmentation to a set of pixel neighborhoods
3) Erosion followed by dilation morphological operators in an if loop in order to verify the operators will take place on one object and will not remove any useful information.
4) A split-and-merge algorithm approach

If said problem is solved by any of the following proposals, the algorithm then can begin the k-means clustering implementation. Some important improvements are, the introduction of an adaptive threshold over the global threshold, for background subtraction, and the development of a function that preserves the identified objects, in this case people,

since the objects will be subtracted and considered background, should they remain still. If said implementations and improvements take place, the algorithm can be tested over real time applications and with the introduction of additional objects to classify.

Since no reliable results were produced, unfortunately, there are no measurements of detection presented in the appendix of this paper.

5 Project Management and Expenses

The total cost for implementing this project, not taking into account the computers and the development software, cost nothing. The data for implementation and testing were free of charge and are intellectual property of the University of Reading that generously has granted free access for the purposes of academic research.

The overall progress of the project did follow the Gantt chart in the early stages, in some cases arriving at the milestones earlier than anticipated, however there was a small delay in the completion of the final milestone due to the challenge that the nature of the final milestone possesses.

In the initial stages of the project, most of the time was spent familiarizing with the development platform and its functions in regards of the intended application. After that, a much-needed basic theoretical study of the technology and the necessary communication with the professor supervising the project, were required in order to continue to further develop the project.

The first milestone, Image Pre-Processing, was completed on time, utilizing most of the time in code development and communication with the professor, ensuring the smooth progress of the project.

The second milestone, Contour Capture, was also completed on time, utilizing most of the time experimenting with different approaches utilizing morphological operators and image filtering.

The third and final milestone, Human Detection, was not completed on time, due to the complexity of the problem, failure to separate the clustering phenomenon that has been presented above, and lack of experience in this technological field and algorithms.

References

1. Arif M., et al. Counting of People in the Extremely Dense Crowd using Genetic Algorithm and Blobs Counting, IAES International Journal of Artificial Intelligence, 2012
2. Chan A., et al. Analysis of Crowded Scenes using Holistic Properties, 11th IEEE Intl. Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2009), 2009
3. Fradi H., New insights into crowd density analysis in video surveillance systems, Télécom ParisTech, 2014
4. Fu Z., et al., Particle PHD Filter Based Multiple Human Tracking Using Online Group-Structured Dictionary Learning, IEEE, 2018
5. Liang M., Hu X., Recurrent Convolutional Neural Network for Object Recognition, Computer Vision Foundation, 2015
6. Marsden M., et al., Fully Convolutional Crowd Counting on Highly Congested Scenes, 2017
7. Mitropoulos S., Human Detection in CCTV, 2016
8. Petrushin V., Roqueiro D. Counting People using Video Cameras
9. Redman J., Farhadi A. YOLO9000: Better, Faster, Stronger., 2016
10. Ren S., et al., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016
11. Roqueiro D., Petrushin V., Counting People using Video Cameras, International Journal of Parallel Emergent and Distributed Systems, 2007

Bibliography

1. Ballard D., Computer Vision, 2nd Edition, Prentice-Hall, 1982
2. Bigun J., Vision with Direction, Springer, 2006
3. Forsyth D., Ponce J., Computer Vision A Modern Approach, Prentice Hall Professional Technical Reference, 2002
4. Gonzalez R., Woods R., Digital Image Processing Third Edition, Prentice-Hall, 2006
5. Jähne B. Haussecker H., Computer Vision and Applications A guide for Students and Practitioners, Academic Press, 2000
6. Maragos P., Computer Vision, NTUA, 2005
7. MATLAB Image Processing Toolbox Documentation [Last Accessed: 8/05/2018] https://www.mathworks.com/products/image.html
8. Nixon M., Aguado A., Feature Extraction and Image Processing, Butterworth-Heinemann, 2002
9. Paragios N., Chen Y., Faugeras O., Handbook of Mathematical Models in Computer Vision, Springer, 2006
10. Ritter G., Wilson J., Handbook of Computer Vision Algorithms in Image Algebra Second Edition, CRC Press, 2001
11. Russ J., Neal F., The Image Processing Handbook Seventh Edition, CRC Press, 2016
12. Shapiro L., Stockman G., Computer Vision [Last Accessed: 28/04/2018] http://nana.lecturer.pens.ac.id/index_files/referensi/computer_vision/Computer%20Vision.pdf
13. Shah M., Fundamentals of Computer Vision, Mubarak Shah, 1992
14. Solomon C., Breckon T., Fundamentals of Digital Image Processing A Practical Approach with Examples In MATLAB, John Wiley & Sons, 2011
15. Szeliski R., Computer Vision: Algorithms and Applications, Springer, 2010
16. University of Reading., PETS 2009 Benchmark Data [Last Accessed: 25/04/2018] http://www.cvg.reading.ac.uk/PETS2009/a.html

Appendices

Appendix A: MATLAB CODE

Appendix A1: Background subtraction function

Function to remove the background from the image.

```
function [ fg, bg_bw ] = imBackSub( fr_bw , bg_bw , threshold )
%% Function to subtract background from foreground
%   ------------------------------------------------------------ %
%   Function that subtracts the background from the foreground
%   The function calculates the absolute difference between the
%   foreground
%   and the background, background being the first frame which
%   can be
%    calibrated, and foreground being the next frame. Afterwards
%   the two for
%   loops are utilized to scan top to bottom vertically the frame
%   and
%    compare the values of each coordinate with the threshold. %
%   If the
%   values are lower than the threshold, the function sets said
%   coordinates
%   to zero.

[height, width] = size(bg_bw);

fr_diff = abs(double(fr_bw) - double(bg_bw));  % cast operands
%as double to avoid negative overflow

thresh = threshold;
for j=1:width
        for k=1:height
            if ((fr_diff(k,j) > thresh)) % if fr_diff > thresh,
then pixel in foreground

                fg(k,j) = fr_bw(k,j);
            else
                fg(k,j) = 0;
            end
        end
end

bg_bw = fr_bw;

end
```

Appendix A2: Contour Extraction Function

Function to capture the contour of an object using morphological operators.

```
function fgc = contour(fg, ns1)

%% Function to capture the sillouete of an object using
morphological operators
%   ------------------------------------------------------------  %
%
%   The following function is utilized to capture the sillouete of
%any
%   present objects.
%   The function firstly, creates a morphological structuring
%element,
%   specifically, a disk where nsl is the radius of the disk.
%Afterwards it
%   converts the foreground image into a gpuArray element for
%improved
%   speed processing. Then it implements a 2-D median filter to
%remove any
%   noise from the image. A morphological operator is used to fill
%isolated
%   interior pixels. Finally, it utilizes the imclose morphological
%   operator, to morphologically close the image

st = strel('disk', ns1);
fg = gpuArray(fg); % Uncomment if the system's GPU is supported
fg = medfilt2(fg, [7, 7]);
fg = bwmorph(fg, 'fill');
fgc = imclose(fg, st);
end
```

Added code for perimeter extraction for each contour captured:

```
fgc = bwperim(fgc,8);
```

Appendix A3: Label Clusters Function

Code to identify and label the objects for every frame.

```
function [ CC, centroids, objCounter, num ] = imlabel( fgc,
conncomp, minsizeofCC )
%% Identify connected components / label the objects
%   ------------------------------------------------------  %
%   The function utilizes the blwabel command to identify any number
%of
%   objects that have an 8-pixel connection. Afterwards if the
%vertical
%   length of the object is lower that the threshold, the %functions
%sets
%   said object's coordinates to 0, in order to remove it %since it
%isn't a
%   useful object for this implementation. Finally, it calculates
%the
%   centroid for each squared-object in order to display a label to
%number the
%   object

objCounter = 0; % counter for num of objects in scene
[CC,num] = bwlabel(fgc,conncomp);
for i = 1:num
    [r, c] = find(CC==i);
    szr = length(r);
    if szr > minsizeofCC
            objCounter = objCounter + 1;
            CC = gather(CC); % Copy the array back to MATLAB
%workspace.
            %Uncomment if the system's GPU is supported
            CC(r,c)= objCounter;
    else %if  szr < thresh
            CC = gather(CC); % Uncomment if the system's GPU is
%supported
            CC(r,c)= 0;
    end
end
s = regionprops(CC,'centroid');
centroids = cat(1, s.Centroid);

end
```

Appendix A4: Cluster Extraction Function

Function to extract the present clusters that contain more than one person.

```
function [ cluster] = imcluster(num, fgc)
%%  Function to identify and eliminate clusters
%   ------------------------------------------------------------------  %
%   The function calculates the total area for every object in the current
%   frame. It calculates the average and frequent value for the allAreas
%   matrix. Afterwards it calculates the standard deviation. The function
%   then loops for every object present in the current frame, storing the
%   current object's area and comparing the distance of the current object
%   with the most frequent value.

fgc = gather(fgc); % Uncomment if the system's GPU is supported
fgtemp = size(fgc); % Create dummy matrix to combine with fgc image
cluster = size(fgc); % Preallocate cluster variable
getarea = regionprops(CC,'Area'); % Get the area of all the objects in the
frame

allAreas = [getarea.Area]; % Store all area values into an array

meanvar = mean(allAreas); % Find the average/mean value
freqvar = mode(allAreas,2); % Find the most frequent value
medianvar = median(allAreas,2); % Find the median value

distance = (allAreas - meanvar) / (num-1); % Calculate the distance from the
mean value

k = 0; % If loop counter, useful to know how many times the if loop was used
for a=1:num
    thisCentroid = [getarea(a).Area]; % Variable to select the current cluster
        if distance(a) > freqvar
            k = k +1;
            clusterTemp(a) = thisCentroid; % Variable to select the desirable
clusters
            [sortedAreas, sortIndexes] = sort(clusterTemp, 'descend'); % Store
and index the selected clusters
            for j = 1:k
                biggestCluster = ismember(CC, sortIndexes(k));
            end
        end
end
```

Appendix A4: Main Program Function

The main function that combines all of the above functions with plotting features and recording the produced output. This part of the main code is responsible for the selection of the input video file, and the prompt for the option of saving the output file and visualizing it.

```
% This m-file implements the frame difference algorithm for background
% subtraction.  It may be used free of charge for any purpose

clear all
close all hidden

% Video file selection dialog box
[filename,pathname] = uigetfile({'*.avi';'*.mp4'},'Select the video
file');

% Construct a multimedia reader object associated with avi files.
vidObj = VideoReader(fullfile(pathname, filename));

% read first frame as background
bg = readFrame(vidObj);
bg_bw = rgb2gray(bg);            % convert background to greyscale


%% dialogue box for setup variables

prompt = {'Video Feed Start Read Time (sec):',...
    'Results Visualisation (no: 0 / yes: 1)',...
    'Output Video Filename:'
    };
dlg_title = 'Input';
num_lines = 1;
defaultans = {'0','1',filename '_out'};
answer = inputdlg(prompt,dlg_title,num_lines,defaultans);
```

This part of the main function is responsible for setting the visualization parameters and setting the parameters of the recording of the output file.

```
%% ---------------------- set video viewer and frame size variables --
---------------------
width = vidObj.Width;
height = vidObj.Height;
fg = zeros(height, width); % Create dummy variable to store foreground
values

% if plot image is required for visualization
flag = str2num(answer{2});
if flag == 1
    fflag = figure;
    figure(fflag);
    set(fflag, 'Position', round( get(0, 'Screensize')));
end

% You can specify that reading should start at n seconds from the
% beginning using
starttime = str2num(answer{2});
vidObj.CurrentTime = starttime;


filename = answer{3};
framerate = vidObj.FrameRate;
if flag==1
        writerObj = VideoWriter([filename,'.avi']);
        writerObj.FrameRate = framerate;
        open(writerObj);
end
```

The final piece of the main function, is responsible for calling the previous functions, visualizes the results and records them for further analysis.

```
%% --------------------- get and process frames for n times-----------------
disp('processing initiated...')
thresh = 25;    % threshold for pixel value to consider as
background/foreground
ns1 = 15; % neighborhood for frame dilation (dilation not present, only usage
in imclose)
nsNN = [7, 7]; % neighborhood window for median filter

counter = 0; % this is a counter / useful to know when using a while-loop
while hasFrame(vidObj)
    fr = readFrame(vidObj);     % read current frame
    fr_bw = rgb2gray(fr);       % convert frame to grayscale
    [ fg, bg_bw ] = imBackSub( fr_bw , bg_bw, thresh ); % Background
Subtraction function
    fgc = contour(fg, ns1); % Contour extraction function
    [ CC, centroids, objCounter, num] = imlabel( fgc, 8, 80 ); % Object
labeling function
    [cluster] = imcluster(objCounter, fgc); % Shadow removal & crowd analysis
function
    % plotting some results
    counter=counter+1; % increase counter
    % interactively plot if needed
    if flag == 1
        figure(fflag)
        drawnow % ensure figure window is updated
        subplot(1,2,1);imshow(fr);
        title(['Original Frame #',num2str(counter)]);
        % make the colour image clusteres visible
        fgcRGB(:,:,1) = (uint8(fgc).*uint8(fr(:,:,1)));
        fgcRGB(:,:,2) = (uint8(fgc).*uint8(fr(:,:,2)));
        fgcRGB(:,:,3) = (uint8(fgc).*uint8(fr(:,:,3)));
        % make visible estimated labelled objects
        subplot(1,2,2);
        imshow(fgcRGB);
        hold on
        if objCounter~=0
            %plot(centroids(:,1),centroids(:,2), '+', 'MarkerSize', 10);
            for i=1:objCounter
                strObj = ['#',num2str(i)];
                text(centroids(i,1),centroids(i,2), strObj,...
                'HorizontalAlignment','center',...
                'FontWeight','bold','FontSize',10,...
                'Color','red')
            end
        end
        hold off
        title(['Foreground Labelled Clusters #',num2str(objCounter)]);
        frame = getframe(gcf); % 'gcf' can handle if you zoom in to take a
movie.
        writeVideo(writerObj, frame);
    end
end
disp('Background subtraction, foreground labelling: DONE...')
close(writerObj);  %stop(vidObj);
```