Using Image Analysis to Read Medical Devices – Tutorial

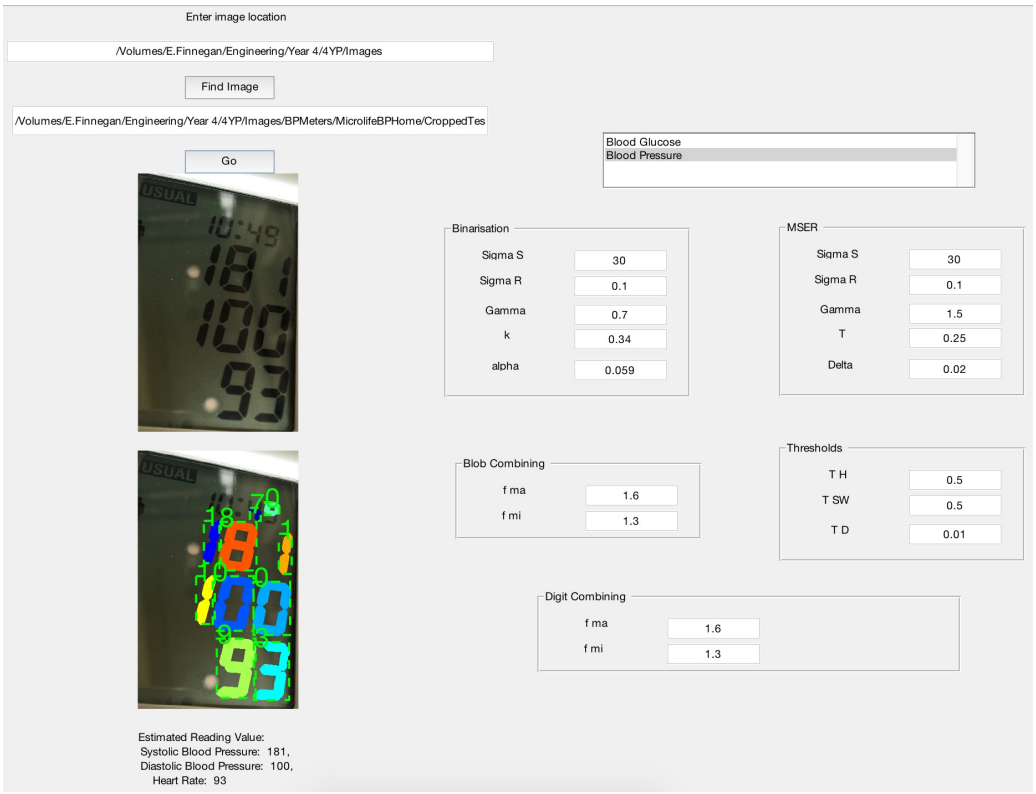


Figure 1: Example GUI with output and variables – to be changed

1. Introduction

The growing and developing market of mobile health (i.e. systems that combine modern technologies to provide innovative and pervasive approaches to healthcare) has led to an increase in smartphone applications that record, track and monitor a user's personal medical data. A user of these mobile health applications takes a measurement (such as their blood glucose level) and enters the value into the application which then adds to a time series of recorded data points that can be used to aid in the management of their condition.

This tutorial outlines the algorithms and code required to automatically read physiological values from images taken of a medical device by a smart phone camera. The algorithms are implemented in Matlab 2018a and are described in more detail in (link to paper/report). The code set for this algorithm can be downloaded from:

The data sets can be downloaded from: Medical devices [1].

This tutorial is structured as follows: chapter 2 outlines the datasets that have been developed and uploaded for this project, chapter 3 outlines the image preprocessing applied to the images.

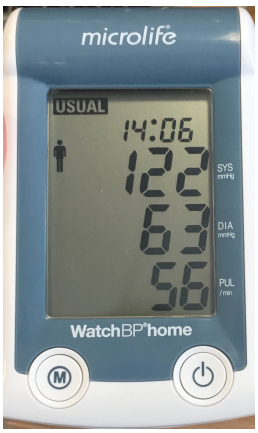
2. Datasets

2.1 Medical Devices

This tutorial focusses on blood glucose meters and blood pressure monitors. The chosen blood glucose meter was the BG One Touch Ultra Mini [**?**] (fig. 2.1a). The chosen blood pressure monitor was the Microlife BP Home [**?**] (fig. 2.1b).

[1] https://tinyurl.com/ycf8otpa

(a) BG One Touch

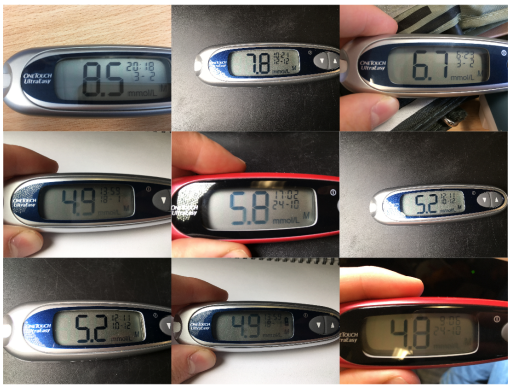


(b) Microlife BP Home

Figure 2.1: Example images of medical devices

A dataset of images of these medical devices has been developed. This dataset is representative of the different ways in which a user will be expected to take an image by considering variations such as: the distance at which the user takes the image, the angle of the camera to the device and the amount of light in the image. The images were taken with a variety of different smartphone cameras (iPhone 6S, iPhone 5, Samsung S5, HTC Desire and a Samsung Tab E).

132 example images were taken of the BG One Touch Ultra Mini and 168 example images were taken of the Microlife BP Home. Each set of images was split evenly and randomly into training and testing. The training images will be used to develop the algorithms for digit reading, whereas the testing images will be used to test its performance. Figure 2.2 shows a small subset of the images in these datasets.



(a) BG One Touch

(b) Microlife BP Home

Figure 2.2: Example images in the blood glucose meters and blood pressure monitors

As can be seen in fig. 2.2b, when taking an image of the blood pressure monitor the user is requested to focus on the whole screen with little background. This is due to the blood pressure meters containing digits on multiple lines. A more robust algorithm can be developed which automatically locates the screen of the medical device. ("Optical character recognition system for seven segment display images of measuring instruments") - via colour matching. This would reduce the complexity of the algorithm by minimising the amount of noise in the selected region and allow more use cases for the image to be taken in (i.e the image can be taken from further away). However, when implemented for this project it was found to act as a significant source of error that can be ignored by asking the user to take the image closer to the screen. This is saved as a further research area.

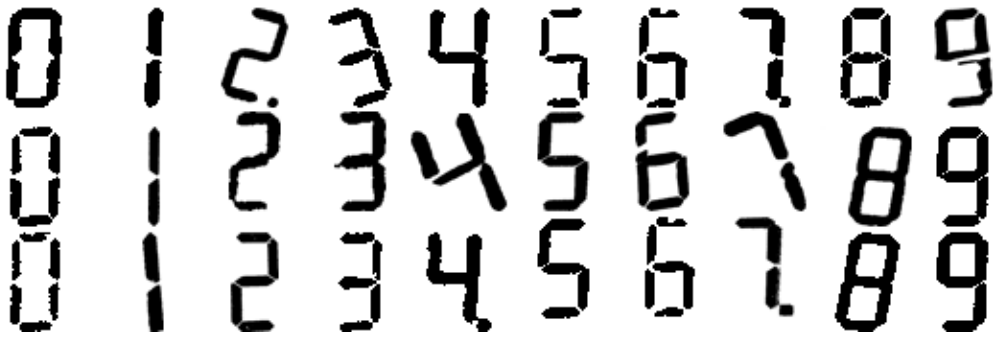


Figure 2.3: Example real digits extracted from the medical devices dataset and saved as a 52 x 52 pixel binary image.

Digit	0	1	2	3	4	5	6	7	8	9
# of occurrences	63	279	137	95	148	122	157	177	99	100

Table 2.1: Occurrences of each number in the "real" digit dataset

2.2 Segments and Noise

		Device	
		Blood Glucose	Blood Pressure
Noise	Train	2019	5818
	Test	2645	6802
Segments	Train	2954	454
	Test	3926	492

2.3 7-Segent Digits

To label the 7-segment digits found on the medical device by their value, two datasets of 7-segment digit images were generated. The first dataset is of "real" digits found on the medical devices, the second is a much larger dataset of synthetic digits automatically generated.

Real digits

Processing the dataset of medical devices using the methods later discussed in **??** provided the dataset of "real" digits. The digits found in the medical devices images were saved as a binary images of size 52 x 52 pixels. In total 1,377 digits were extracted and labelled from the medical devices dataset (434 digits from the blood glucose images, 943 digits from the blood pressure images). The number of digits that was extracted using this method was capped not only by the number of images of medical devices taken and how many digits are present in each image, but also by the accuracy of the methods to locate the digits. Figure 2.3 shows examples of the digits extracted in this manner and table 2.2 shows the distribution of digits present in the dataset.

Typical classification problems require thousands of training and testing images that are distributed evenly among classes (e.g the same number of digit 1s, 2s etc). We decided that creating a sufficiently large database of extracted digits was not feasible for the scope of this project. We therefore decided to generate a synthetic dataset of 7-segment display images, with a much larger number of images that are evenly distributed. The earlier datasets of saved digits will be used as a further test set to measure the accuracy of a classifier trained on synthetic data.

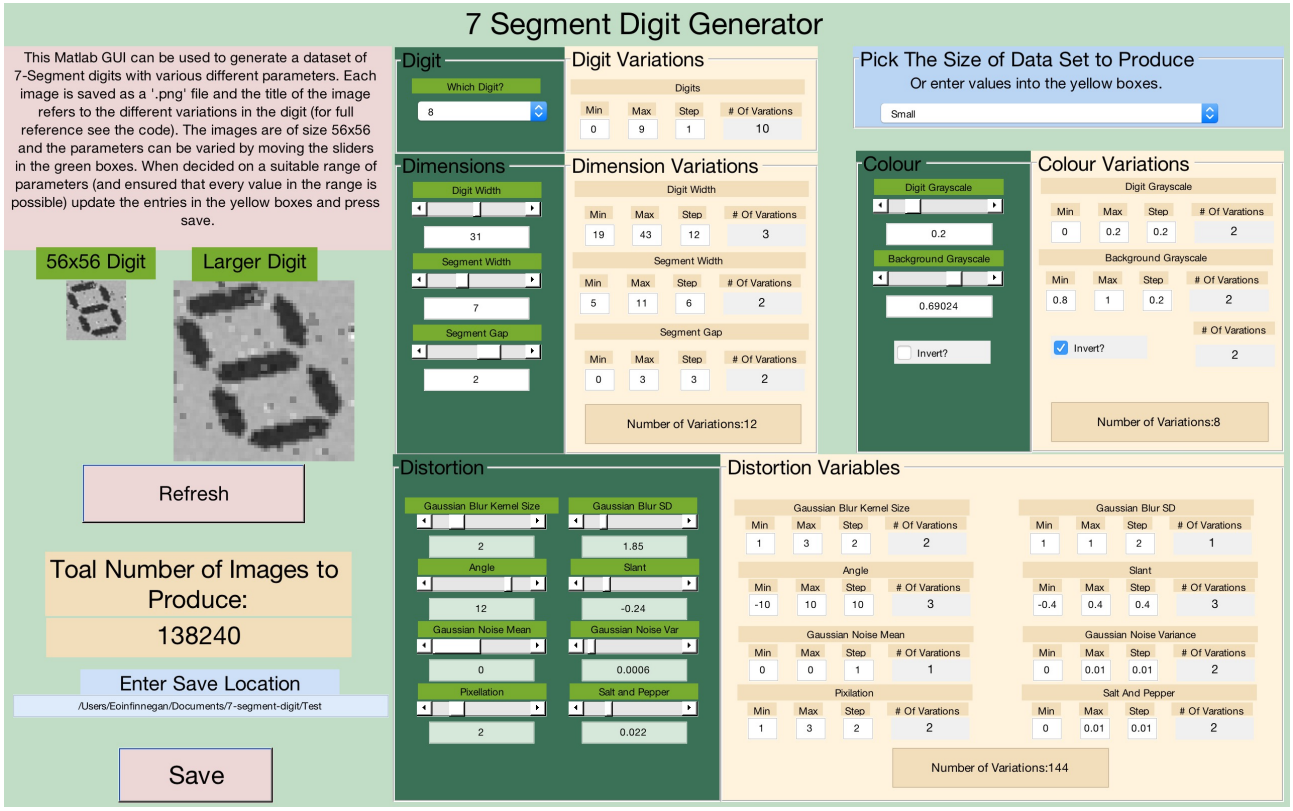


Figure 2.4: Synthetic digit generation Matlab GUI [**?**]

Synthetic dataset

In order to achieve a reliable classification, it was important that the datasets generated fully represent the test cases (i.e. the data set of saved digits). Although online, open source automatic 7-segment digit generation software exists, [**?**], [**?**], these software do not provide the same parameter variations which are seen in the displays of medical devices. We therefore developed a new software toolbox to automatically generate a set of digits that reliably represents real-case images and release this code for public use so that future projects requiring a synthetic dataset of 7-segment digits can easily generate samples.

To ensure accurate detection of digits on medical devices, the dataset needed to contain the same parameters and variations as seen in real test cases. We split the digit parameters into 3 main sections:

- **Dimensions** - Variation of size and shape: digit width, segment width and segment gap.
- **Distortion** Geometric distortions and noise that distort an image: Gaussian blur size, Gaussian blur variance, pixellation, Gaussian noise variance, salt and pepper density, angle and slant.
- **Colour** Changes in grayscale intensity of the digit and background: digit greyscale, segment greyscale and invert. This is to simulate a low-cost LCD screen with a low contrast between background and foreground.

To assist in the visualisation of the effects that these parameters have on a digit, and to generate a dataset to be used in classification, we generated and uploaded a GUI (shown in fig. 2.4) to Bitbucket for public use [**?**]. Details of these parameters and the justification for including them are available in the README file. Users can vary all parameters using the sliders in the green boxes. The resulting digits are shown to the left of the user interface. In addition, the yellow boxes allow users to input the minimum, maximum and step values (the step sizes for the parameter to take between min and max) to generate a synthetic dataset of 7-segment digits. The generated digits are grayscale images of size 52 x 52 pixels.

We investigated different variations of these parameters in order to make a resulting synthetic dataset that contained as many of the realistic variations seen in the real-world data as possible. When deciding the parameter

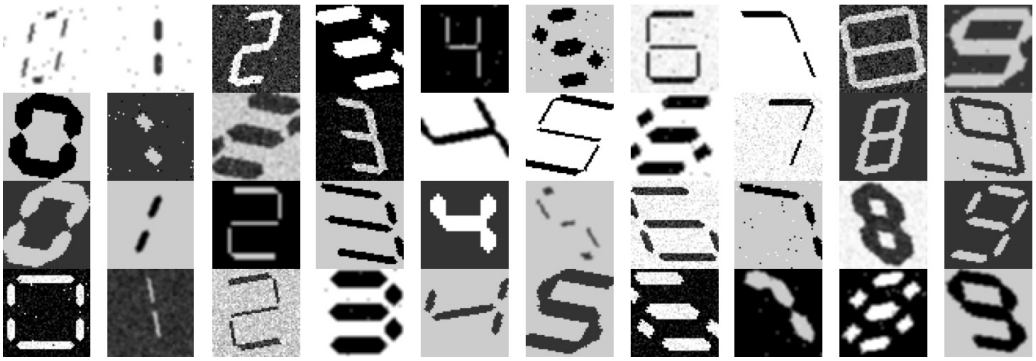


Figure 2.5: Examples of synthetic digits generated and saved as grayscale 52x 52 pixel images.

variations, it is important to include the extreme values that will be seen in the real digit test set. Learning algorithms are able to interpolate from training data; however it is difficult to extrapolate beyond the range covered in the training set.

The final parameter choices resulted in 13,824 images per digit, thus a total of 138,240 images were generated for the 10 digits (0-9). The parameter variations can be seen in table 2.2. We decided not to vary the Gaussian blur variance, as significant blurring can occur by simply varying the kernel size with a unity variance. The synthetic digits will be used to train and test a digit classifier in **??**. Figure 2.5 shows examples of the synthetic digits generated.

Parameter	**Min**	**Step**	**Max**	**Variations**
Digit width (pixels)	19	12	43	3
Segment width (pixels)	5	6	11	3
Segment gap (pixels)	0	3	3	2
Gaussian blur size (pixels)	1	2	3	2
Gaussian blur variance	1	~	1	1
Pixellation (block size)	1	2	3	2
Gaussian noise variance	0	0.01	0.01	2
Salt and pepper density	0	0.01	0.01	2
Angle (°)	-10	10	-10	3
Slant	-0.4	0.4	0.4	3
Digit Grayscale	0	0.2	0.2	2
Background Grayscale	0.8	0.2	1	2
Invert	0	1	1	2
Total per digit				**13,824**

Table 2.2: Variations of parameters found in the synthetic dataset

3. Image Preprocessing

3.1 HSV Colour Space

The initial RBG image should be converted to the HSV (Hue, Saturation and Value) colour space. The Hue component of an image corresponds to which colour it resembles, the saturation component corresponds to how white the colour is and the Value component corresponds to how dark the colour is. The following code shows how to extract the Hue and Value component of an image. The Value component is used in next part and the Hue component is saved till later.

```
IMG_Original = imread('BPMeter.png'); %Read in the image

%Resize the image for processing speed -- can change to be anything
global HEIGHT; HEIGHT = 500;
global NewWidth;
NewWidth = ceil(HEIGHT / size(IMG_Original,1) * size(IMG_Original,2));
IMG_Original= imresize(IMG_Original, [HEIGHT, NewWidth]);
```

```
hsv = rgb2hsv(IMG_Original); %Convert RGB to HSV

hue = hsv(:,:,1); value = hsv(:,:,3); %Save the hue and value component
```

3.2 Retinex Using Two Bilateral Filters

Each image is first preprocessed by Retinex using two bilateral filters [?]. This filter removes the impact of the illumination in which an image was taken. Retinex using two bilateral filters makes an estimate of the illumination to then derive an estimate of the reflectance (the illumination independent image). This estimate is made using the edge-preserving bilateral filter as a measure of the illumination variation across the image.

Retinex using two bilateral filters has three main inputs: the variance of the bilateral filter in the spatial domain σ_S, the variance in the intensity domain σ_R and γ. γ is used after the illumination component of the image has been filtered out as:

$$I_{out} = I_{in}^{\frac{1}{\gamma}} \tag{3.1}$$

```
%First define the relevant parameters
sigmaSpatial = 30; sigmaRange = 0.1; samplingSpatial = sigmaSpatial;
    samplingRange = sigmaRange;
gamma = 1.5;

%Apply retinex using two bilateral filters from https://github.com/KirillLykov/
    cvision-algorithms.git
image_retinex = retinexFilter(value, sigmaSpatial, sigmaRange, samplingSpatial,
    samplingRange, gamma, 0);

%Show the results
figure
multi = cat(3, value, image_retinex);
montage(multi)
title('Applying Retinex')
```

Figure 3.1 shows the result of applying retinex using two bilateral filters to an example image.

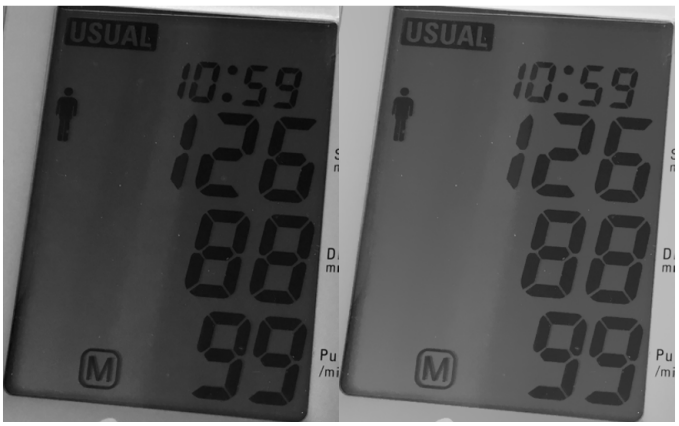


Figure 3.1: Output of applying retinex using two bilateral filters

3.3 Adaptive Histogram Equalisation

The segments of a digit can be detected by using the property that segments on the screen are darker than the screen itself. Contrast enhancement was used to increase this contrast and to aid in segment detection. This was performed using adaptive histogram equalisation (AHE) [**?**] with windows of size 8x8.

```
%Apply adaptive histogram equalisation to the retinex image
image_ahe = adapthisteq(image_retinex, 'NumTiles', [8 8]);

%Display results
figure
multi = cat(2, image_retinex, image_ahe);
montage(multi)
title('Applying AHE')
```

Figure 3.2 shows the result of applying AHE to the output of retinex.

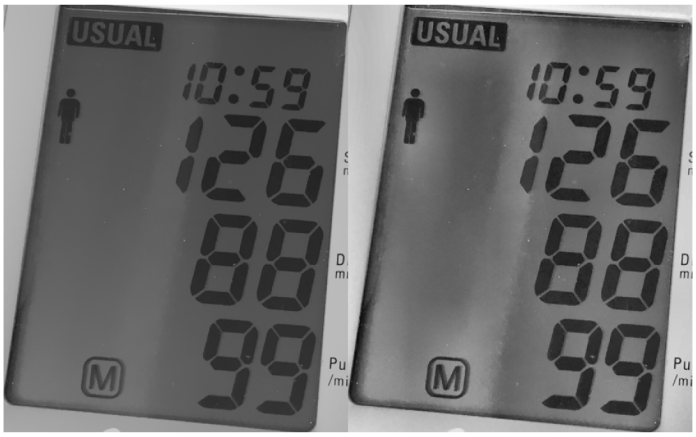


Figure 3.2: Output of applying AHE

4. Digit Location

Figure 4.1 outlines the algorithm for locating 7-segment digits in images of medical devices. The preprocessing algorithm first enhances the image by reducing noise and illumination effects. Potential segments of the 7-segment display are located and stored as binary large objects (known as blobs). This is performed using two methods in parallel. The first method uses the property that the pixel intensities inside the area defining a segment have very little variation, and the second method uses the property that the contrast between the intensities of the background of the display and the 7-segment digit is large. The blobs found by these two strategies are then classified as either noise or digit segments using a logistic classifier trained on several extracted features. As one digit is made up of several segments, the segments are then clustered to form a single digit.Finally the digits found are passed to the digit classifier.

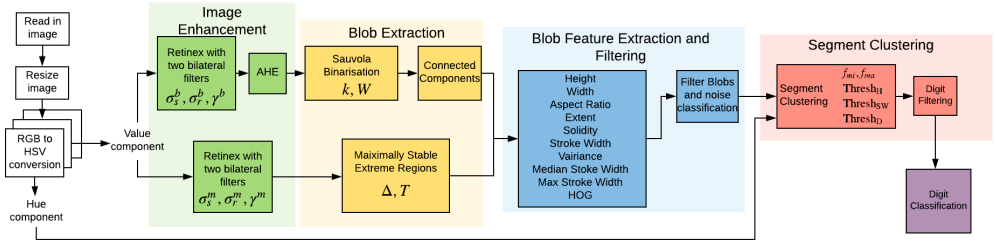


Figure 4.1: Digit location process

4.1 Maximally Stable Extremal Regions

The Maximally Stable Extremal Regions (MSER) feature detector works well for finding digit regions [?]. MSER detected potential digits by finding regions in which the intensity values of a region vary minimally.

As AHE added noise to an image and MSER is dependent on consistent segment intensities rather than large contrasts between background and foreground, the MSER was operated on the output of retinex by two bilateral filters.

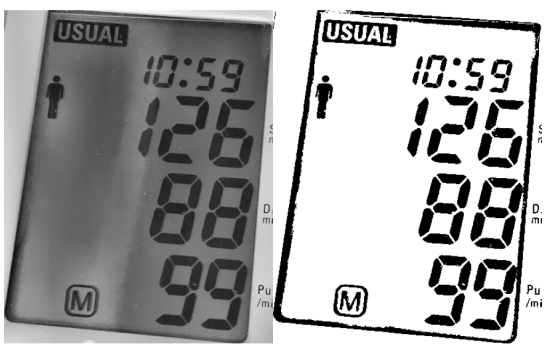
```
%Define params
Delta = 0.02; T = 0.25;

%Detect MSER regions
[regions_MSER, mserConnComp] = detectMSERFeatures(image_retinex, 'ThresholdDelta
    ', Delta*100, 'MaxAreaVariation', T);

%Use regioprops to get features of each region -- to be used later
regionstats_MSER = regionprops(mserConnComp,'PixelIdxList','Centroid','
    MajorAxisLength', 'MinorAxisLength','Orientation', 'Area', 'BoundingBox', '
    Euler', 'Solidity', 'Extent', 'Image');

% Get Hue Stat
for i= 1:length(regions_MSER)
   regionstats_MSER(i).Hue = hue([regionstats_MSER(i).PixelIdxList ]);
end

figure
imshow(image_retinex)
hold on
plot(regions_MSER, 'showPixelList', true,'showEllipses',false)
hold off
title('Regions located by MSER')
```

Figure 4.2 shows the regions detected by applying MSER to the output of retinex.

Figure 4.2: Output of applying MSER to Retinex

4.2 Connected Components of a binarised image

To binarise the image apply Sauvola's binarisation [?]. Sauvola's binarisation is an adaptive technique based on the contrast between two regions in an image. The binarisation threshold is given by:



(a) Output of applying Sauvola to AHE

(b) Regions detected by Connected Components of a Sauvola binarisation

Figure 4.3: Labels from CC of binarisation

$$T(x,y) = \mu(x,y)(1 + k(\frac{\sigma(x,y)}{R} - 1)) \tag{4.1}$$

where μ is the mean intensity of the window surrounding a pixel, σ is the standard deviation of the window, k is a fixed variable and R is the maximum variance of the image. Thus the parameters defining the binarisation are the window size, W, and k. The optimum size of the window is dependent on the size of the input image. The size of one side of a square window is defined as $W = \text{round}(\alpha H)$, where H is the height of the image, $0 < \alpha < 1$ and round(x) rounds x to its nearest integer. As blob extraction is now dependent on the contrast difference between the segments and the background, binarisation is performed on the output of the contrast enhanced image.

```
%Define params
alpha = 0.059; k = 0.34;

%Get binary CC
image_binarised = sauvola(image_ahe, floor(HEIGHT*alpha),k);

%Display the results of the binarisation
multi = cat(3, image_ahe, image_binarised);
figure, montage(multi)
title('Result of applying Sauvola binarisation to output of AHE')

%So that the regions detected by MSER and Binarisation and handled in the same
    way
[regions_binary, ConnComp_binary] = detectMSERFeatures(image_binarised);

%Use regioprops to get features of each region -- to be used later
regionstats_binary = regionprops(ConnComp_binary,'PixelIdxList','Centroid','
    MajorAxisLength', 'MinorAxisLength','Orientation', 'Area', 'BoundingBox', '
    Euler', 'Solidity', 'Extent', 'Image');

% Get Hue Stat
for i= 1:length(regionstats_binary)
   regionstats_binary(i).Hue = hue([regionstats_binary(i).PixelIdxList ]);
end

figure
imshow(image_retinex)
hold on
plot(regions_binary, 'showPixelList', true,'showEllipses',false)
hold off
title('Regions located by CC of Binarised Image')
```

(a) (b)

Figure 4.4: Example blobs from the blob extraction algorithm. a) Noisy blobs. b) Segments

4.3 Combine Regions

The blobs extracted using the two methods are combined together and handled in the same way.

```
regions_combined = regions_MSER; %Assign the new regions to be the same type as
    the regions located by MSER

%Concatenate the regions by MSER and CC of binary image (Only need to
%combine the PixelList field and the other fields are combined
%automatically)
regions_combined.PixelList = [regions_MSER.PixelList; regions_binary.PixelList];

%Combine the stats of the regions
regionstats_combined = [regionstats_MSER; regionstats_binary];
```

4.4 Filter Regions

In both blob extraction techniques, a significant number of the blobs detected were noise. Figure 4.4 shows examples of the detected blobs where fig. 4.4a are noise and fig. 4.4b are digit segments (or a combination of digit segments). In order to achieve accurate reading of the digits on the medical device, the noisy blobs were filtered from the segments by a combination of rule-based filtering and classification on extracted features.

Stroke Width (SW) [**?**] is a common feature used in letter recognition in natural images. The stroke of a blob is defined as the length of a straight line from one edge pixel to another along its gradient direction. Figure 4.4 shows that segments tend to have a more consistent SW than noise. Therefore, we used the variability of blob strokes to classify if a blob was a digit segment or not.

Rule-based filtering is first performed to remove overly large blobs that cannot be segments. Blobs were rejected if they had properties outside of the following bounds:

- $(A \times 0.0001) \leq \text{Area} \leq (A \times 0.4)$
- $0.15 \leq \frac{\text{Height}}{\text{Width}} \leq 6$
- $(H \times 0.015) \leq \text{Height} \leq (H \times 0.4)$
- $(W \times 0.01) \leq \text{Width} \leq (W \times 0.3)$
- SW variance ratio ≥ 5
- SW diameter median ≥ 6

where H and W are the height and width respectively of the image in pixels, and $A = H \times W$. The area of a blob was considered the total number of pixels that make up the blob. SW variance ratio and SW diameter median are defined below.

4.4.1 Classification

A database of blobs was created by running the blob extraction algorithm on all images in the training and test sets for each medical device separately. The blobs were saved as a 52 x 52 pixel binary image and manually labelled as either a segment or noise. Each medical device, therefore, had a training and test database of blobs labelled as noise or segments.

Further blob filtering was performed by classifying based on a certain set of features.

The features extracted were commonly used features for locating text in natural images [**?**, **?**]:

1. Aspect ratio = $\frac{\text{Height}}{\text{Width}}$
2. Occupancy rate = $\frac{\text{Area}}{\text{Height x Width}}$
3. Solidity = $\frac{\text{Area}}{\text{Convex Area}}$
4. Euler number = The total number of objects in the image, minus the total number of holes in those objects
5. SW variance ratio = $\frac{\text{Stroke Width Variance of Blob}}{\text{Mean Stroke Width}}$
6. SW diameter median = $\frac{\sqrt{(\text{Height}^2+\text{Width}^2)}}{\text{Median Stroke Width}}$
7. SW length max = $\frac{\text{Length of blob}}{\text{Max Stroke Width}}$
8. Height ratio = $\frac{\text{Height}}{\text{Height of image}}$
9. Width ratio = $\frac{\text{Width}}{\text{Width of image}}$
10. HOG feature vector with cell sizes 8x8

A logistic classifier was trained on the feature vectors of blobs extracted from the training images and was tested on the blobs extracted from the test images.

```
%First define the rule based values:

%Toal size of the image
TotalArea = HEIGHT*NewWidth;

%Filteiring is done twice this is round 1
Filtering.round = 1;


%Area
Filtering.MaxArea = TotalArea * 0.4; Filtering.MinArea = TotalArea*0.0001;

%Height
Filtering.MaxHeight = HEIGHT*0.7; Filtering.MinHeight = HEIGHT * 0.01;

%Width
Filtering.MaxWidth = NewWidth * 0.5; Filtering.MinWidth = NewWidth * 0.001;

%Ratio
Filtering.RatioMin = 0.1; Filtering.RatioMax = 8;

%Stroke Width
Filtering.NormalisedVariance = 5; Filtering.DiameterMedian = 6;

%Filter the regions
[regions_filtered, regionstats_filteres] = Filter(regions_combined,
    regionstats_combined, Filtering);

%Display the results:
figure
imshow(image_retinex)
hold on
plot(regions_combined, 'showPixelList', true,'showEllipses',false)
hold off
title('Combined Regions')
```

```
figure
imshow(image_retinex)
hold on
plot(regions_filtered, 'showPixelList', true,'showEllipses',false)
hold off
title('Filtered Regions')
```

5. Overall Script

```
clear
close all
clc
%Step 1 -- Convert to HSV colour space
HSV
%Step 2 -- Retinex
Retinex
%Step 3 -- Adaptive Histogram Equalisation
AHE
%Step 4 -- Maximally Stable Extremal Regions
MSER
%Step 5 -- CC of binarise image
BinarisationAndCC
%Step 6 -- Combine
Combine
%Step 7 -- Filter
Filter_Script
%Step 8 -- Classify Blobs
```