

School of Computer Science and Informatics
Cardiff University
CM3203 – One Semester Individual Project (40 Credits)
2016 – 2017



Forensic Image Processing – Bruises

Author: Kai Wa Chan (C1531993)

Supervisor: Paul L Rosin

Moderator: David Marshall

1. Abstract

To identify a biter of a bite mark and measure the dimension of a bruise from evidential image are the complex and difficult tasks in forensic science because of the nature of a bruise and the subjectivity of human. This project aims to use image processing technique to automatically match the bite mark to the corresponding dentition cast and measure the dimension of bruises.

Automatic, consistent and identical computer method to perform matching and measuring for bruise is highly recommend to support legal system in terms of analysis and interpretation of bruise evidence to protect vulnerable victims of violent crimes.

This report covers the details of experimented methods including the results, benefits, failures as well as the improvements to support the development of computer based forensic operation.

2. Acknowledgement

I would like to give my special thanks to my Professors, Paul Rosin and David Marshal as well as Dr Xianfang Sun for their advice and supervision during this project. But also, another huge thank you to Mr. Sam Evans for making this project happen.

Contents

1. Abstract.....	2
2. Acknowledgement	3
3. Introduction	5
3.1 Goals.....	6
3.2 Project Structure	6
4. Background.....	7
4.1 Common Used Algorithms	8
5. Data Description	11
6. Dentition Cast Extraction	13
6.1 MSER – Blob Detection.....	13
6.2 Colour/Intensity Segmentation:	16
7. Skin Extraction	21
7.1 HSV & YCbCr Colour Thresholding.....	22
7.2 Probability Density Function	25
7.3 Quadratic Discriminant Analysis (1).....	28
7.4 Quadratic Discriminant Analysis (2).....	32
7.5 K-Means Clustering	36
7.6 Summary of Skin Detection	40
8. Draw Bruise Window	41
8.1 Circle feature based drawing.....	41
9. Features Extraction	47
9.1 SIFT.....	47
10. Feature Matching	50
11. Bruise Segmentation and Dimension Measurement	51
11.1 Colour Quantisation & Region Growing	52
11.2 Central Weighting & Thresholding on Saturation	60
11.3 Lighting Correction with Fourier Transform	65
11.4 Summary of Bruise Segmentation and Dimension Measurement	66
12. Future Work	67
13. Conclusion	69
14. Reflection	70
15. Reference	72

3. Introduction

One of the most important aspects of criminal justice is forensic science or the practice of scientifically examining physical evidence collected from the scene of a crime or a person of interest in a crime. Many people consider forensic science as the application of science to law enforcement. If there are no known witnesses to a crime, sometimes forensic evidence is all the prosecutors can work with. This evidence is commonly used in court to put offenders behind bars and to set innocent people free.

In this project, the evidence, bruises including bite mark is studied. The bruise is usually cast by violence. According to Refuge [1], the world's first safe house for women and children escaping domestic violence, it is reported that 1 in 4 women in England and Wales will experience domestic violence in their lifetimes and 8% will suffer domestic violence in any given year (Crime Survey of England and Wales, 2013/14). And every minute police in the UK receive a domestic assistance call – yet only 35% of domestic violence incidents are reported to the police (Stanko, 2000 & Home Office, 2002). Also, 20% of children in the UK have been exposed to domestic abuse (Radford et al. NSPCC, 2011). 62% of the children in households where domestic violence is happening are also directly harmed (SafeLives, 2015). Those crimes are not usually seen by any witnesses, but in some cases, bruises are the only available evidence. Therefore, the interpretation of the evidence plays an important role in criminal justice.

The evidence is captured by 5 groups of operators which are police forensic photographers, clinical (hospital) photographers, scene of crime officers, forensic expert (forensic Odontologists and pathologists) and clinicians (paediatrician) commonly using digital-single-lens-reflex (DSLR) or single-lens-reflex (SLR) camera equipped with a range of lenses and separate flash system.

After evidence capturing, the image is ready to be analysed. Not only observation on the image itself, forensic operators also use some photo editing software tools such as Photoshop® to extract features from evidential image or enhance contrast of images manually [2]. For bite mark analysis, the dentition cast of potential biter is also involved in the investigation. However, the extraction and judgment involve many human procedures, which also means errors and subjectivity. As evidence using in court, errors and subjectivity are the main factors to reduce the evidential value in the litigation.

Therefore, using image processing technique to automatically collect, analyse and match characteristics from the forensic image is desired. It could provide consistent and identical forensic operation. Also, computers may power up the observation of the bruise and eliminate human errors and subjectivity. it maintains the evidential value of the image to provide sufficient proof to put offenders behind bars, to set innocent people free and protect victims more effectively.

3.1 Goals

Create automatic, consistent and identical computer methods to perform bite mark and dentition cast matching and measuring bruise are the two goals of this project.

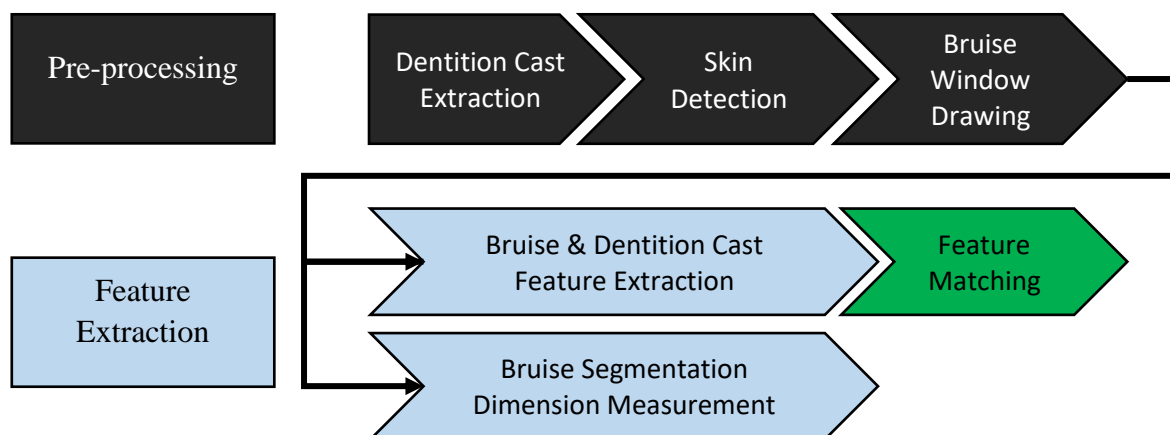
Bite mark analysis is the interpretation and comparison of two pieces of evidence, namely, the photographic record of the bite mark and the dental casts of suspect biter. There are two well recognized physical comparison methods, Feature-based Analysis and Superimposition-based Analysis. Both methods require some features including class and individual characteristics which could be found from the bite mark in the skin and the dentition cast. Class characteristic includes the shapes, sizes and arches of the teeth of the maxillary and mandibular anterior dentitions. Individual characteristics are related to the distinctive features of the incisors and canines of anterior dentitions such as displacement, rotation and incisal edges. However, those characteristics may be difficult to be observed by human due to the nature of a bruise including the leakage and spread of the blood. [3]

Moreover, collecting bruise characteristics such as the dimension is another huge challenge. Because of the colour difference of a bruise, it is hard to draw a standard to explicitly define bruise boundary on the skin. Different sensitivity of colour vision of humans is also a factor in causing subjectivity to define boundary of a bruise. Not only boundary, there is not a standard method to measure the bruise's dimension because of arbitrary shapes of bruises. Therefore, different forensic operators may have an unequal method to collect the dimension of a bruise.

3.2 Project Structure

Before identifying which cast causes the bite mark and measure the dimension of a bruise, there are some pre-processing procedures are required. This project could be roughly broken down into 3 main sections including Pre-processing, Features Extraction and Features Matching.

In general, a forensic image contains three components evidence (bruise/cast), a measurement tool and arbitrary background. A series of pre-processes are required to remove unrelated areas to let algorithms focus on features from the evidence (bruise/cast) directly rather than another unrelated object. The pre-processing should extract the evidence components by removing unrelated components in the image. After that, extracting features from the bruise and dentition cast could be performed. Finally, a feature of the matching algorithm could be designed.



4. Background

A bruise is blood leakage into tissues under the skin and causes the black-and-blue colour. As bruises (contusions) heal, they often turn colours, including purplish black, reddish blue, or yellowish green. Sometimes the area of the bruise spreads down the body in the direction of gravity. The bruises to be analysed in this project could be categorised into two groups which are bite mark and general bruise. Both the bruises share the same nature [4]. There are not many image processing literature for human bruise image produced by digital-single-lens-reflex (DSLR) or single-lens-reflex (SLR) camera. The researches [5] related to bruise analysis are majority based on hyperspectral images. However, using image processing techniques in medical fields is not a new idea. There have been some of the researches describing how image processing could identify analysis and measuring medical objects like detecting lung cancer on CT image [6], determining arthritis on MRI image [7] and classifying skin lesion [8]. Those techniques such as segmentation, feature extraction, matching and recognition may be still useful for processing the bruise image.

Several methods are experimented to reach the objective in each section. Then, one of the methods is suggested which solves the problem or perform better.

Section	Objective	Methods
Pre-processing	Dentition cast extraction	1. MSRE
		1. Colour & Intensity Segmentation
	Skin Extraction	HSV & YCbCr Colour Thresholding
		Probability Density Function
		Quadratic Discriminant Analysis (1) (2)
		K-Means Clustering
	Draw Bruise window	Hough Circle
Features Extraction	Bruise features Extraction	SIFT
	Bruise Segmentation and Dimension Measurement	1. Colour Quantisation & Region Growing 2. Thresholding & Central Weighting 3. Lighting Correction with Fourier Transform
Features Matching		
Common Used Algorithms	Fundamental image processing techniques	1. Gaussian Smoothing/Blurring 2. Contour Finding/Filling 3. Morphological Transformations

Highlights are the final selected method to address corresponding objective

4.1 Common Used Algorithms

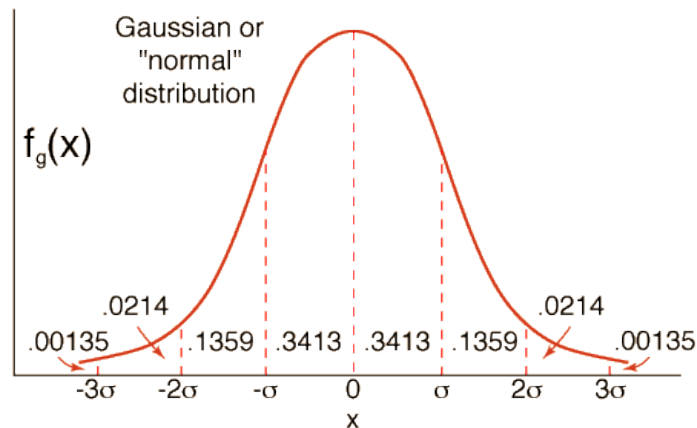
4.1.1 Gaussian Smoothing/Blurring

Gaussian Smoothing/Blurring is a popular image processing technique to reduce details of an image, which involves an image, two 1D Gaussian kernels and two convolution processes (vertical and horizontal). It is a low-pass filter, attenuating high frequency signals.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

A 1D Gaussian kernel is defined as

It looks like

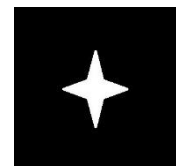


The Gaussian outputs a 'weighted average' of each pixel's neighbourhood, with the average weighted more towards the value of the central pixels. The weighting could be defined by sigma. Sigma and kernel size are the factors to control the level of blurring or smoothing. In general, larger sigma and kernel achieve higher blurring level.

4.1.2 Contour Finding/Filling

Contour finding is border following algorithm which is proposed for topological analysis of digitized binary images in 1985. The definition of border following algorithm is given by [9] "determines the surroundness relations among the borders of a binary image. Since the outer borders and the hole borders have a one-to-one correspondence to the connected components of 1-pixels and to the holes, respectively, the proposed algorithm yields a representation of a binary image, from which one can extract some sort of features without reconstructing the image". And contour filling is just a filling operation after finding, where the pixels inside borders are filled.

From non-technical speaking, a contour can be explained simply as a curve joining all the continuous points (along the boundary), having same colour or intensity. For example, consider the image on the right.



4.1.3 Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images (Mask). By default, white pixel is foreground and black pixel is background. There are required two inputs, one is the original binary image, second one is called structuring element/kernel which decides the nature of operation. In this project the kernel size I use is Ellipse, which could produce more smooth result compare to other kernels.

A 5x5 ellipse kernel:



```
[0, 0, 1, 0, 0],  
[1, 1, 1, 1, 1],  
[1, 1, 1, 1, 1],  
[1, 1, 1, 1, 1],  
[0, 0, 1, 0, 0]
```



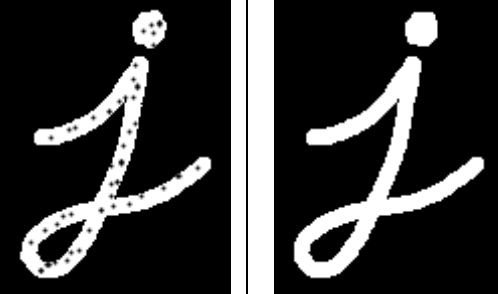
In this project, 4 morphological operations are used frequently to remove noise. These are Erosion, Dilation, Opening and Closing.

The idea of Erosion is to erode away the boundaries of foreground object. During the process in OpenCV. The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). On the other hand, Dilation is the exact opposite of Erosion. The level of Erosion and Dilation can be controlled by kernel size and the number of times of the 2D convolution.

Opening is another name of Erosion followed by Dilation. Erosion removes white noises, but it also shrinks the object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. Closing is reverse of Opening, Dilation followed by Erosion.

There are some examples obtained from OpenCV to show the effect of the transformations [10].

Transformation	Orginal Image	After Transformation	Usage
Erosion			Removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Dilation		Increase the white region in the image or size of foreground object increases.
Opening		Closing small holes outside the foreground objects
Closing		Closing small holes inside the foreground objects

5. Data Description

There are four data sets {1, 2, 3, 4} provided for this project at different time during the project schedule. All the datasets are given by Cardiff School of Dentistry and Cardiff School of Medicine.

Dataset1 image set doesn't fully fulfil the forensic science photography standard (e.g. IM1, IM2, IM3). It is because the images may not include the measurement tool and may be cropped. The 16 images include both general bruise and bite mark on human skin. The purpose of this Dataset is for initial project discussion. Dataset 1 is majority used for testing basic skin detection algorithms at the beginning of the project while waiting for targeted Datasets.

(IM1, IM2, IM3)



Datasets 2 to 4 are professional forensic photographs which fulfil the standard of forensic photographic protocols such as measurement tool, camera and lighting setting.

Dataset 2 contains 5 forensic images of bite mark on human arm in different backgrounds (e.g. IM4, IM5) and 14 sets of corresponding and potential dentition cast in lab environment (e.g. IM6, IM7). Each set of dentition cast consists of 3 images, which are the label, upper and lower cast. Also, a Control Set is attached, which contains 30 sets of professional photographs of bite mark on wax in the same lab environment (e.g. IM8) and 30 sets dentition casts with the same setting. The Control Set is not used in this project. The usage of this set is to match the dentition cast and bite-mark.

(IM4, IM5)



(IM5, IM6, IM7)



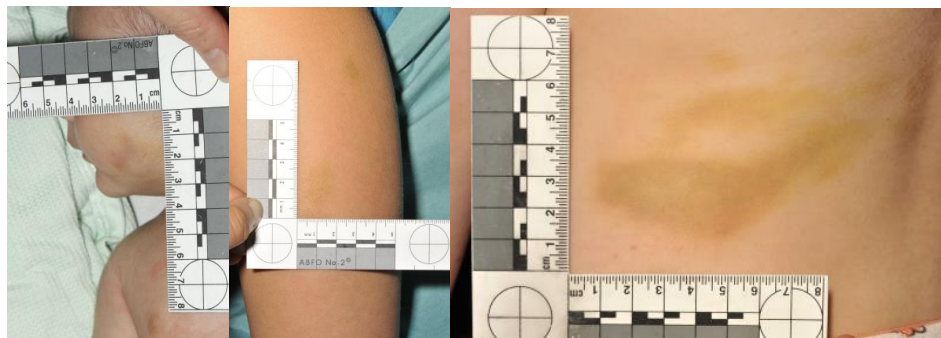
Data Set 3 contains 45 forensic photographs of general bruise on several human body parts in different backgrounds (e.g. IM8, IM9). Also, distinctly there are 3 purple dots made by marker on every image in this Dataset. This set is used to test algorithm to measure the dimensions of the bruise.

(IM8, IM9)



Data Set 4 contains fourteen images (e.g. IM10, IM11, IM12) which are similar as Data Set 3, except the 3 purple dots. This set also corresponds to a list of bruise measurement which is produced by forensic operators. This set is used to test algorithm to measure the dimensions of the bruise.

(IM10, IM11, IM12)



Data Sets 3 and 4 are majority used as bruises feature extraction.

6. Dentition Cast Extraction

The objective of this stage is to extract the dentition cast area, then we can extract features in the further process.

6.1 MSER – Blob Detection

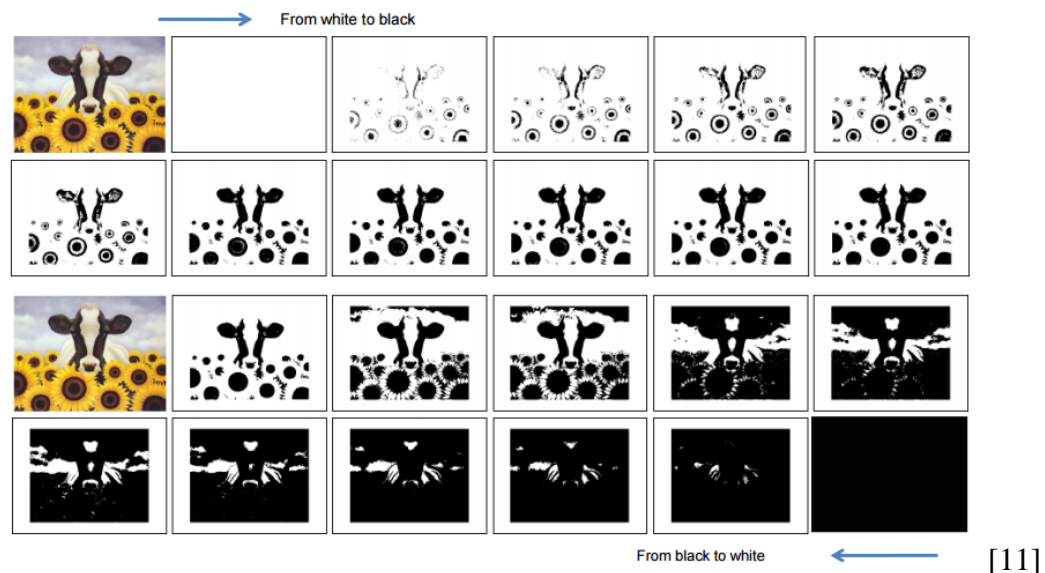
From our observation, the cast is a huge blob in the middle of the image. It is believed that the blob detection method could extract the cast region.

The first approach is MSER which stands for “Maximally Stable Extremal Regions”, which is a common technique to detect blobs in computer vision. In general speaking, a MSER output is a stable connected component combined by number of co-variant regions of some grey-level sets of the image.

6.1.1 Method

MSER extraction implements following steps:

1. Sweep threshold of intensity from black to white, performing a simple luminance thresholding of the image as shown below:



2. Extract connected components
3. Find a threshold when an extremal region is “Maximally Stable”

In the above example, the cores of the sun flowers are MSER, because they are extrema and achieve “stable” properties. Over a large range of thresholds, extremal regions occur when the set is closed under continuous (and thus projective) transformation of image coordinates. And maximally stable region is defined as the regions whose support is nearly the same over a range of thresholds and is invariance to affine transformation of image intensities. [11]

6.1.2 Implementation

The implementation is defined in: `DentitionCast.py`

Function: `MSER(im)`

`im`: input image with default data type. (2D numpy array with BGR colour space)

Function returns a copy of the input image using same format with MSER output drawn

- 1) Create MSER detector given parameter
- 2) Obtain the MSER regions using `detector.detect` function
- 3) For each MSER region, approximate a convex hull using `convexHull` function
- 4) Copy the input image as `Copy`
- 5) Draw all the hulls on `Copy`
- 6) And return the `Copy`

OpenCV has defined the implementation which also support users' parameters including size pruning, region variation and diversity, threshold step and margin of edge. At the beginning of the attempt. Only the values for area pruning are modified. The 'min' and 'max' area are set to be '100000' and '5000000' to ignore too small and too big areas. Other parameters are set as default.

`MaxVariation` prune the area have similar size to its children with the default value of 0.25.

`MinDiversity` traces back to cut off MSER with `diversity < min_diversity` and with a default value of 0.2.

`MaxEvolution` is used for colour image, the evolution steps and a default value of 200.

`AreaThreshold` is the area threshold to cause re-initialize with a default value of 1.01.

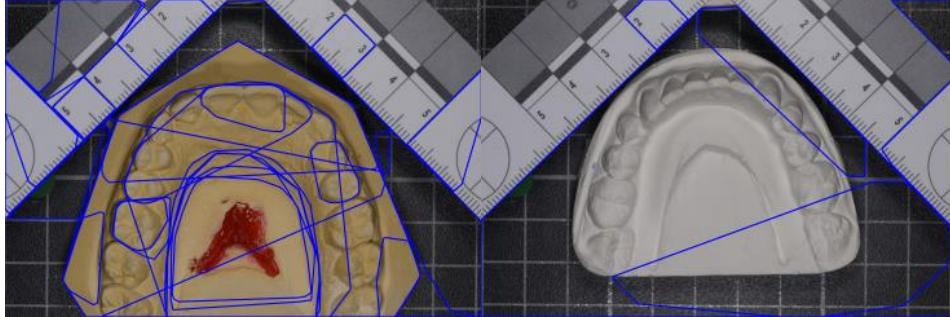
`MinMargin` ignore too small margin and the default value is 0.003.

Test Program: `TestMSER.py`

6.1.3 Results and Evaluation

The first attempt is to input original image which contains high level details to MSER. It is expected that many regions are found because of the details (IM13). There is also some missing detection (IM14).

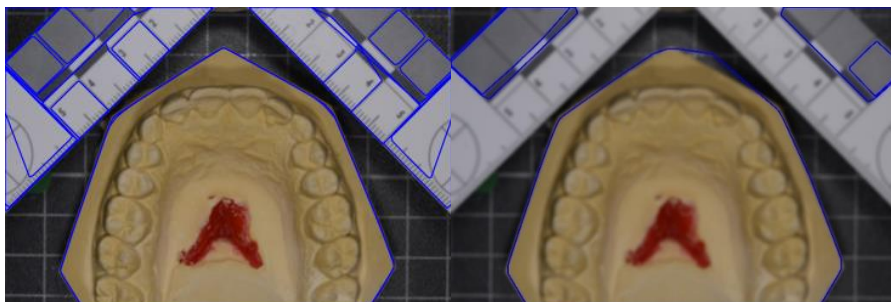
(IM13, IM14)



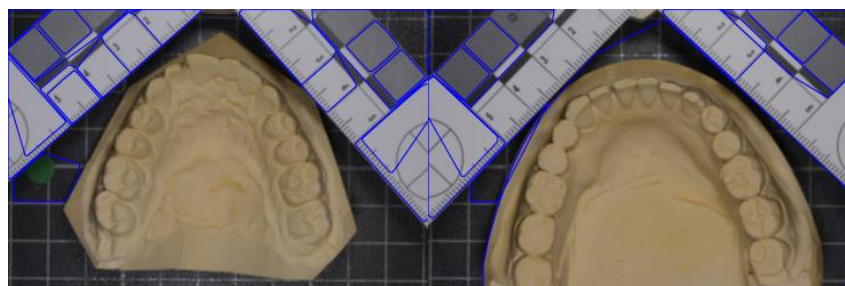
As you can see, the inside regions are detected. And if the details of the image is needed to be reduced, then blurring operation for the image will be useful.

Therefore, Gaussian Blur is applied. IM15 is the output after applying Gaussian Blur with Size 21x21 kernel with sigma 9. IM16 is the output after blurring with size 51x51 kernel with sigma 15 are experimented. And there is still some miss detected and over detected results (IM17, IM18)

(IM15, IM16)



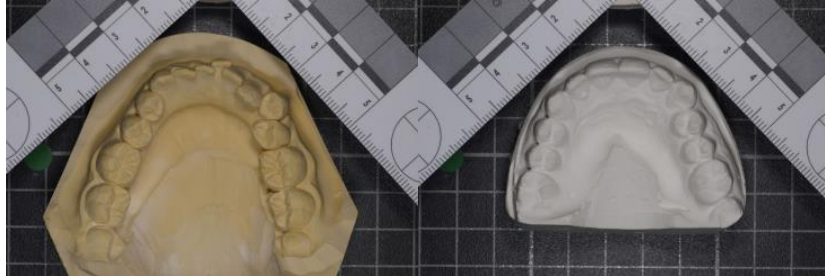
(IM17, IM18)



Although blurring operation helps to reduce the number of MSER, the operation doesn't help to detect the cast or reducing over selecting problem. To correct this algorithm, it is possible to try out some parameters such as "Evolution Steps" and "Area Threshold" values to control the thresholding results during the threshold sweeping process. The overall result is that 4 images miss detected and 3 images are over detected out of 26 images. While turning the thresholding, colour/intensity segmentation approach is introduced.

6.2 Colour/Intensity Segmentation:

As my observation, the environment setting in the cast images is highly similar including lighting, the colour and the position of the ruler and background. The only main difference is that there are two types of the casts, yellow and white coloured casts.



It is impossible to extract it directly using colour thresholding due to the colour difference, but both colours have a common property which is strong intensity value comparing to the background. It means intensity thresholding could extract the cast from the background. However, be aware that the ruler has strong intensity as well except that the ruler has a unique white colour. Therefore, it is possible to delete the top and the regions of the ruler by colour and coordination information first. Where finally, the cast could be extracted by intensity thresholding.

6.2.1 Method

Colour segmentation to extract ruler area for each image: Steps 1 to 5

1. Thresholding the image by RGB colour range from [150,150,150] to [255,255,255].
The result is represented in binary image where pixels from the range on the image turns white. Otherwise, the pixels turn black.
2. Perform dilation and contour filling method
Step 3 and 4 extends the white area in the binary image which helps to connect the fragmented white areas by minimizing the back-measurement line.
3. Find the 2 largest areas which indicate that the left and right sides of the ruler

The reason of finding the largest 2 areas is that the areas in the binary image may be fragmented. This is the result after steps 1 to 4. Blue colour represents the 1st largest area and red represents the 2nd largest area.



However, sometimes the areas are merged where the largest area contains both left and right sides like the image at the left. Therefore, comparing the size of the 1st and 2nd largest areas are necessary. A condition is set to decide to split the 1st area or not. If the 1st biggest area is bigger than the 2nd biggest area*3, it means that the 1st biggest area contains both sizes.

4. Polygon drawing to cover the unwanted areas

Based on the left and right areas, the coordinates could be organized by searching 2 points.

The first point is left bottom point (lb) in the left-side ruler, which is (0, max(y))

The second point is the right bottom point (rb) in the right-side ruler is (max(x), max(y))

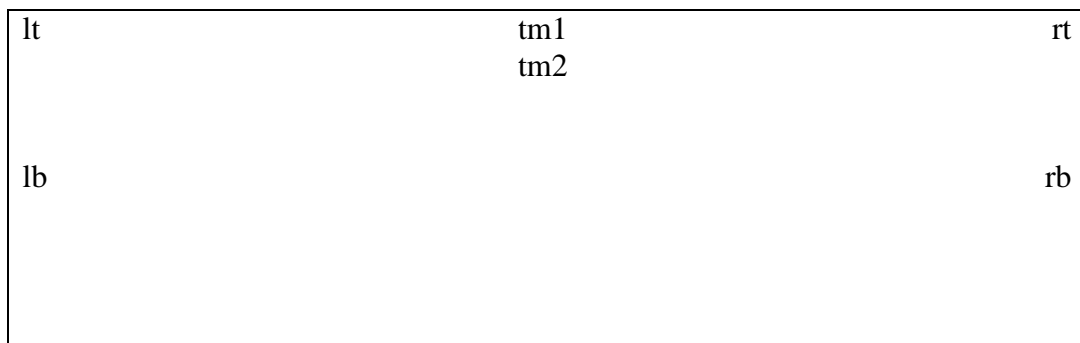
There are also 4 fixed points:

left top corner (lt) - (0,0)

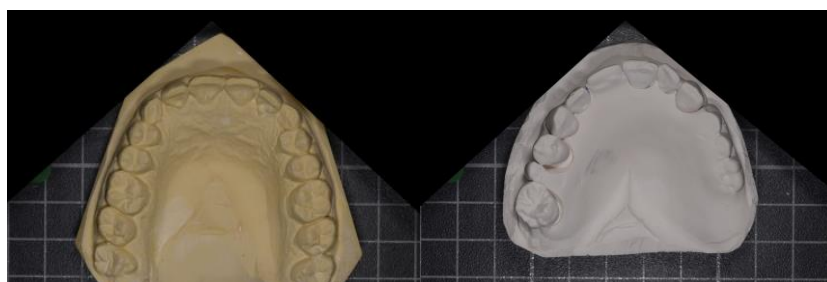
right top corner (rt) - (0, image_width)

2 points at the top middle of the image (tm1, tm2) - (image_width/2, 0), (image_width/2, 90)

The points in the cast image:



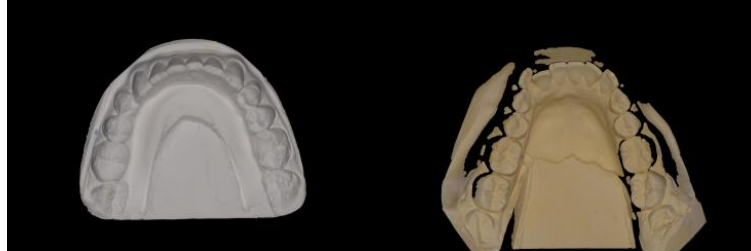
There are total 6 points of coordinates now. We can draw 2 polygons to cover the left and right top regions. Left polygon is connected by lt-tm1-tm2-lb-lt. Right polygon is connected by tm1-rt-rb-tm2-tm1. The following examples are the output after step 4.



After removing the ruler, the image is converted to greyscale. And an attempt is made to extract the cast by thresholding.

5. Thresholding by intensity between 100 and 255
6. Remove noise by morphometric opening

The following images show the result after step 6.

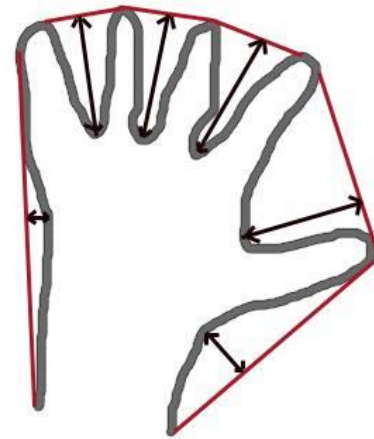


As you may see, the right hand side image is broken. It is believed that the broken parts are shadows of the teeth or the ruler where there are low intensity areas in the cast. The parts are not kept by thresholding.

7. Approximate a single convex hull to connect the broken areas

Ideally, the cast is surrounded by a convex curve. Therefore, using general drawing algorithm like Ramer–Douglas–Peucker is not idea because it doesn't maintain the convexity of the shape. It is recommend to use convex hull approximation [21] which checks the convexity defection of a curve and correct it. Considering a simple explanation, convex curves are always bulged out or at-least flat. Like the example at right hand side from OpenCV tutorial [12].

The “bulged in” areas (Red Circled areas) could be fixed by convex hull drawing.



6.2.2 Implemntation

The implementation is defined in: `DentitionCast.py`

Function: `RemoveRuler(im, lower, upper, hShift, vShift)`

`im`: original image with default data type

`hShift`: offset of point `tm1` and `tm2`

`vShift`: offset of point `tm2`

Function returns a binary image. Element with zero value means the pixel is unwanted (ruler).

- 1) Generate a mask, `Mask` where non-zeros coordinates indicate the pixel in input image is in range using `inRange` function
- 2) Apply contour filling and dilate to `Mask` using `findContours`, `drawContours` and `morphologyEx` function
- 3) Obtain contours using `findContours` function
- 4) Compute the size of each contour using `contourArea` function
- 5) Sort the contours by size
- 6) Select the first and second largest contours as `l1` and `l2`
- 7) If size of `l1` is smaller than size `l2*3`
 - a. Select all the points (x,y) in `l1` which x value is equal to or less than image width/2
 - b. Select all the points (x,y) in `l1` which x value is equal to or greater than image width/2
- 8) Find two points (x,y) in `l1` and `l2` where they are maximum y value
- 9) Check the points x value and decide they belong to left or right side of an image
- 10) Create a mask, `Result` with all zeros
- 11) Draw left and right filled polygons to 255 on `Result` using `fillConvexPoly` function
- 12) Invert `Result` and return it

Test program: `TestCastRulerRemove.py`

Function: `LocalizeCast(im, dilationKsize, dilationIteration)`

`im`: input image without ruler region

`dilationKsize`: the kernel size of dilation operation and the default is 11.

`dilationIteration`: the iteration times for dilation and default is 4.

Function returns a binary image. Element with zero value, which means the pixel is unwanted.

- 1) Convert input image to grey scale using `cvtColor` function
- 2) Generate a mask, `Mask` where non-zeros coordinates indicate the pixel in input image is in range using `inRange` function
- 3) Apply contour filling and morphometric opening to `Mask`
- 4) Obtain contours using `findContours` function
- 5) If number of contours is more than two. Concatenate all contour to one contour

- 6) Generate points of convex hull to the found contours using convexHull function
- 7) Create a mask, Result with all zeros
- 8) Draw the filled convex hull on the Result using drawContours function
- 9) Apply dilation with given kernel size and iteration to Result
- 10) Return Result

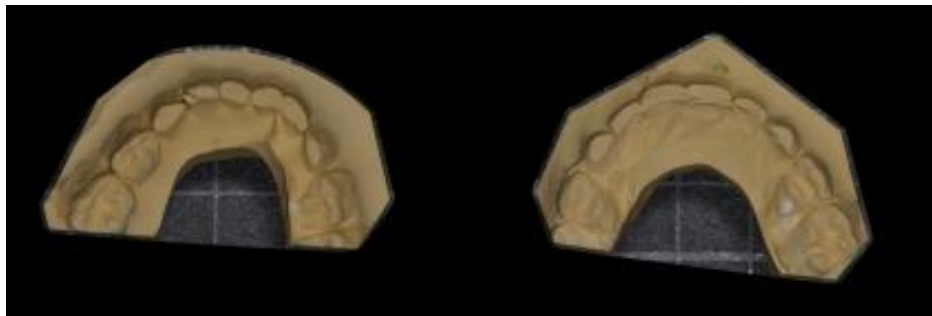
Test program: TestCastExtraction.py

6.2.3 Result and Evaluation

24 out of 26 casts are successfully segmented, which means that the cast is the only object in the image.



There are two special casts which are originally not convex shaped.



The correction is not implemented. It is believed that based on this result, colour thresholding could be re-applied to extract the background area. Then using some bitwise operations to deselect the background areas. Keeping the background is also one of the options. It is because the background might not have much features, or the features which do not correspond to the denture features.

Overall, this algorithm combining colour and intensity segmentation is successful to extract the dentition cast from the image. It is fairly simple and efficient than MSER approach because using a single threshold has basically selected the majority parts of the cast.

7. Skin Extraction

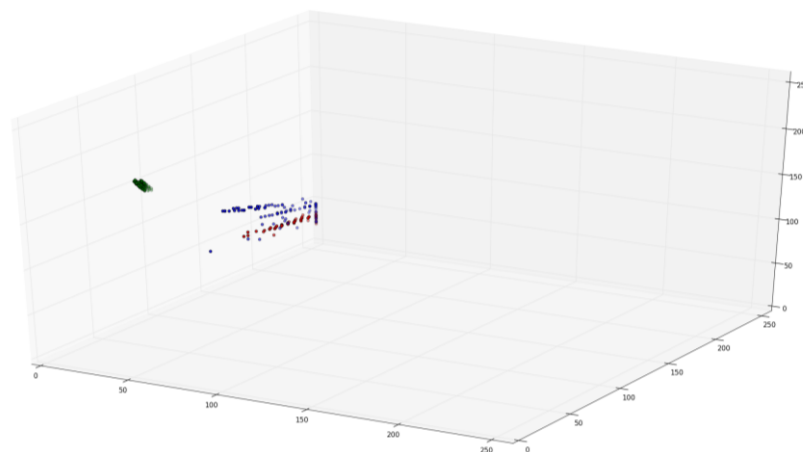
The objective of skin extraction is to reduce the unrelated areas for feature extraction algorithms in the next few steps. It aims to remove background areas for the images in Dataset 2.

As you may see, colour may be a simple and efficient way to differentiate skin and background. Therefore, the colour-based approach is the first attempt such as colour thresholding. Technically speaking, the algorithm identifies the pixel if it is skin or not.

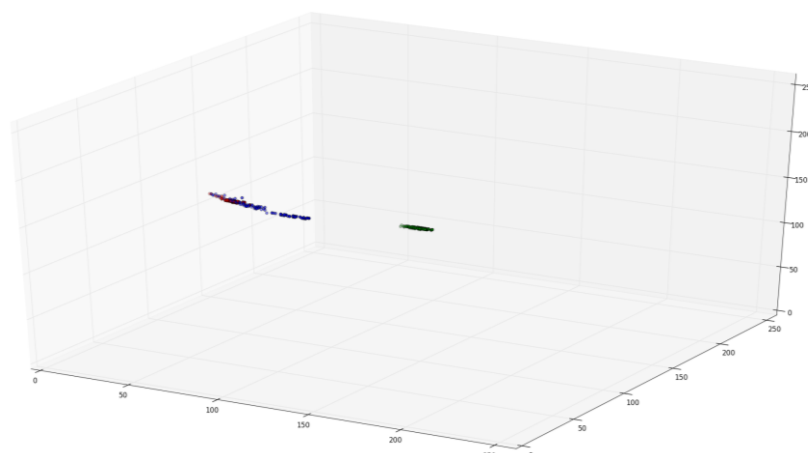
Colour space is an elaboration of the coordinate system and sub-space. Each colour in the system is represented by a single dot. To visualize the concept, skin pixels and background pixels may stand on different positions on different coordination (colour space). We may have different ways to segment the pixels. For example, there are three images cropped from the data set randomly and pixels are plotted into HSV and YCbCr colour space and rescale to the same range.

Colour on Scatter	Red	Green	Blue
Image			

HSV Colour Space



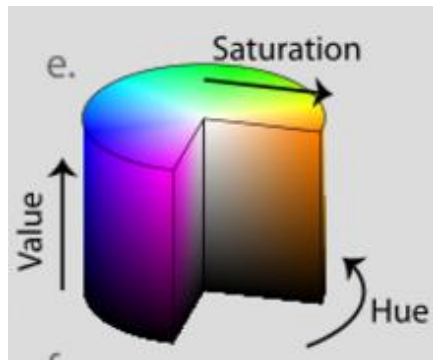
YCbCr Colour Space



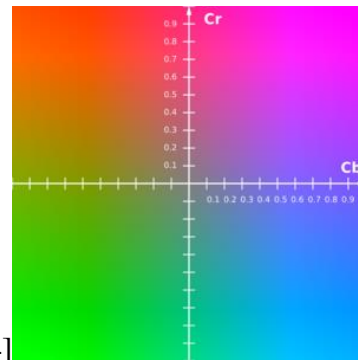
As you can see, the distance between groups blue and red in HSV colour space is more obvious than YCrCb. Therefore, thresholding different colour space may be useful.

7.1 HSV & YCbCr Colour Thresholding

HSV is one of the most intuitive colour space. It is useful to perform segmentation. HSV stands for Hue, Saturation and Value Channel. Also, low level skin detection performed by YCbCr colour space has several successful cases in image processing community [13]. Therefore, YCbCr is also a potential method to segment skin pixel. The HSV, YCbCr colour pickers can represent as follow.



[14]



[15]

7.1.1 Method

To maintain the majority skin areas to be classified, lower and upper bound are defined as follow based on numerous attempts.

1. HSV colour space: Pixels from $[0, 10, 60]$ to $[20, 150, 255]$
2. YCbCr colour space: Pixels from $[0, 133, 77]$ to $[255, 173, 127]$
3. Morphometric closing [10] and contour filling [9] for the result

7.1.2 Implementation

The implementation is defined in: `Skin.py`

Function: `SkinThresholdingV2(im)`

`im`: input image with default data type









Function returns a binary image. Element with non-zero value means the pixel is skin.

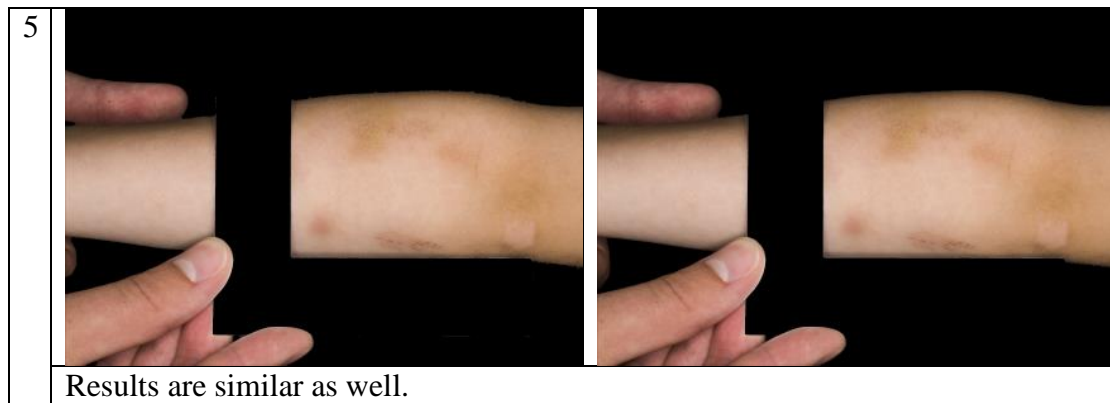
- 1) Convert input image to HSV space using `cvtColor` function
- 2) Generate a mask `hsvResult` using `inRange` function
- 3) Convert input image to YCrCb space using `cvtColor` function
- 4) Generate a mask `yccResult` using `inRange` function
- 5) Apply morphometric closing, contour finding/filling to the masks
- 6) Combine `hsvResult` and `yccResult` using `bitwise_and` operation
- 7) Return combined result

Colour conversion is performed using OpenCV function `cvtColor(im, flag)`. `flag` for BGR to HSV is `COLOR_BGR2HSV`, BGR to YCrCb is `COLOR_BGR2YCR_CB`. For HSV in OpenCV, Hue range is 0 to 179.

Test Program: `TestSkinThresholdingV2.py`

7.1.3 Result & Evaluation:

	HSV	YCbCr
1		
	The results are fairly similar in this particular image	
2		
	In this image, HSV colour space could recognize the ruler, sofa and hairs are not skin. However, the cloth is identified as skin area. Interestingly, YCbCr only identifies the cloth which is not skin area.	
3		
	Two results are similar. Both results could recognize the ruler as skin HSV result is a bit more noise than YCbCr.	
4		



To evaluate the result, it is defined that if the algorithm could only keep the front ground skin and bruise area. Result 1 and 5 are assessed as acceptable.

In image 2 3 4, some of the background areas colour are quite similar as skin colour. Current naïve method is not sensitive enough to identify them. We can conclude that both colour segmentations do not produce false negative and contribute significant true positive. It is beneficial that combine both the results by merging 2 masks using bitwise operations. E.g. image 2 to maximize non-skin areas.



Moreover, the ruler in some of the images are segmented as skin area. This problem could be fixed based on the success in segmentation of dentition cast. We can use RGB colour space to segment the partial ruler areas to contribute better true negative result. Finding ruler is one of the next step in this project. Therefore, in current stage, we leave the result as above and we go ahead to experiment more sensitive algorithms.

7.2 Probability Density Function

Based on the result from Colour Thresholding, we assume that skin and background colours are normally distributed. We could try to re-estimate the skin pixels by using the same single image as data by probability density function to filter out the less likely skin pixels.

The probability density function for multi-dimensional normal distribution is [16]:

$$(2\pi)^{-\frac{1}{2}k} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)'\Sigma^{-1}(\mathbf{x}-\mu)}$$

where μ is the mean, Σ the covariance matrix, and k is the dimension of the space where \mathbf{x} takes values.

7.2.1 Method

For each image based on result of Colour Thresholding:

1. Select the pixels where labelled as skin by Colour Thresholding
2. Skin pixels are rearranged to matrix format to calculate mean and covariance matrix

n pixels	Hue	Saturation	Value
P1	H1	S1	V1
...
Pn	Hn	Sn	Vn

The set of n observations (pixels), measuring 3 variables (colour channels), can be described by its mean vector and variance-covariance matrix.

	Size	Formula
Mean Vector	1x3	$\bar{P} = \left[\frac{1}{n} \sum_{i=0}^n (Hi) \quad \frac{1}{n} \sum_{i=0}^n (Si) \quad \frac{1}{n} \sum_{i=0}^n (Vi) \right]$
covariance matrix	3x3	$Co = \frac{1}{n-1} \sum_{i=0}^n (Pi - \bar{P})(Pi - \bar{P})'$

3. Compute probability density function, $f(\text{image_pixels})$
4. Compute the mean and standard deviation of probabilities of the skin pixels
5. Select the pixels with probability P
 $\text{Mean} - \text{Sigma} * nS \leq P \leq \text{Mean} + \text{Sigma} * nS$

7.2.2 Implementation

The implementation is defined in: `Skin.py`

Function: `PDF(im, nSigma, colourFlag):`

`im`: input image with default data type

Function returns a binary image. Element with non-zero value means the pixel is skin.

- 1) Generate a mask, Skin Mask using `SkinThresholdingV2` function
- 2) Convert input image to other colour space given by third argument using `cvtColor` function
- 3) Apply Skin Mask to input image

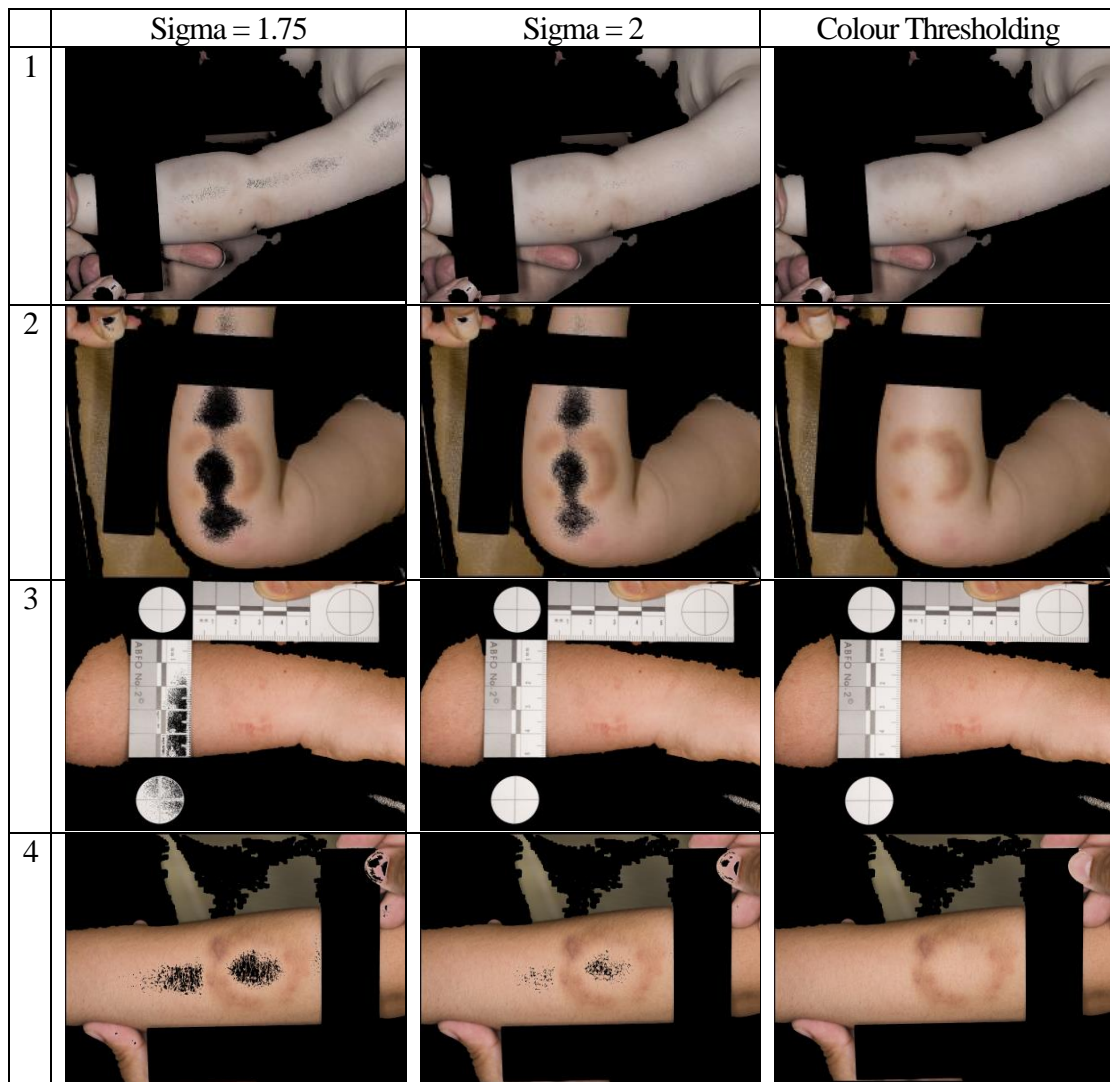
--PDF--

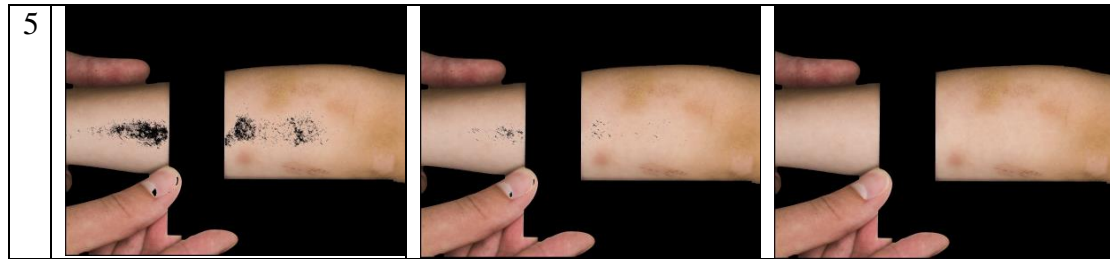
- 4) From input image, obtain all the pixels where marked as non-zero in Skin Mask
- 5) Reshape the pixel as data format
- 6) Compute mean vector
- 7) Compute covariate matrix using calcCovarrMatrix function
- 8) Compute the probability of all pixels using multivariate_normal.pdf function
- 9) Select all the probabilities where the coordiate is marked as skin in Skin Mask
- 10) Compute the standard divation of data from step 9
- 11) Create a mask, Result where the probability is within range $\text{mean} - \text{sigma} * n\text{Sigma}$ to $\text{mean} + \text{sigma} * n\text{Sigma}$
- 12) Combine Result and Skin Mask using bitwise_and operation
- 13) Return result

Test program: TestQuadraticClassifierLocal.py

7.2.3 Result and Evaluation:

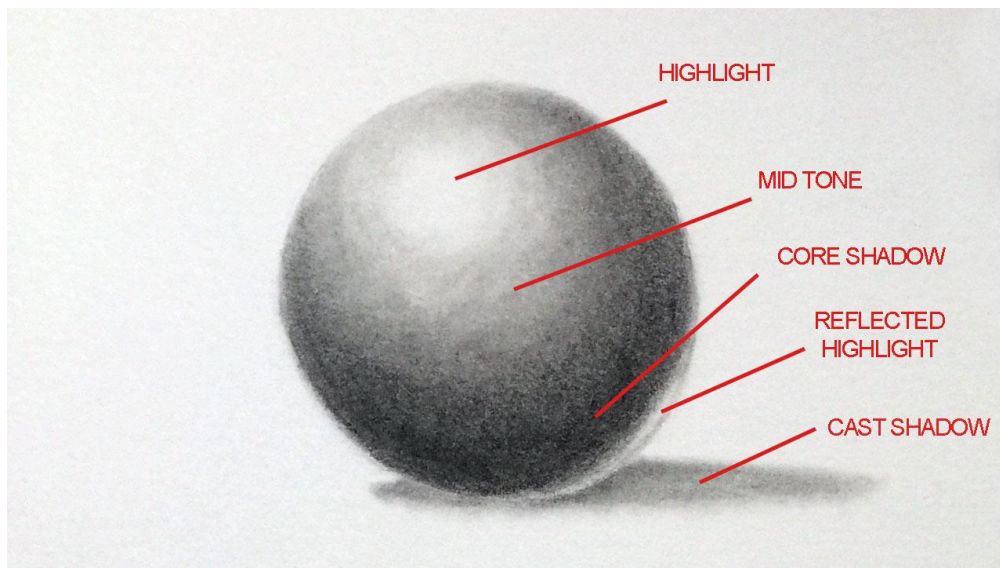
HSV





This algorithm doesn't achieve our objective to eliminate non-skin pixels. From mathematical point of view, PDF successfully removes the outliers. However, the outliers are the skin pixels on the middle of the arms and foreground ruler rather than the background pixels.

We can look back to the Colour Thresholding results, majority pixels which are labelled as skin generally have less intensity except the skin pixels on middle of the arms.



[17]

The reason of those pixels having stronger intensity is that the camera flash or light setting. When the light source hits on a bended surface, it produces some shading effects. The points on surface with same normal direction with the light source generates stronger intensity. When it is converted to HSV space, we can see whiter colour (it shows lower saturation). Therefore, skin pixels on middle of the arms are eliminated as outliers.

Using more information such as the background data may be helpful to classifier the pixels more accurately.

7.3 Quadratic Discriminant Analysis (1)

To improve the skin detection result, quadratic discriminant analysis is used. It is similar as PDA, except that QDA also takes background pixels into consideration to generate the likelihood ratio of each pixel. Then using a threshold to filter the result. It may reduce the likelihood of the non-skin pixel by contributing how the pixel is likely to background. It may produce better result.

$$\text{Likelihood ratio} = \frac{\sqrt{|2\pi\Sigma_{y=1}|}^{-1} \exp\left(-\frac{1}{2}(x - \mu_{y=1})^T \Sigma_{y=1}^{-1} (x - \mu_{y=1})\right)}{\sqrt{|2\pi\Sigma_{y=0}|}^{-1} \exp\left(-\frac{1}{2}(x - \mu_{y=0})^T \Sigma_{y=0}^{-1} (x - \mu_{y=0})\right)} < t \quad [18]$$

while class y1 is skin, y0 is background.

$\mu_{y=0}, \mu_{y=1}$ are the mean vectors of two classes.

$\Sigma_{y=0}, \Sigma_{y=1}$ are the variance-covariance matrices of two classes

t is a likelihood threshold

7.3.1 Method

For each image based on result of Colour Thresholding:

1. Separate the pixels where are labelled as skin and background into two classes
2. For each class of pixels is rearranged to matrix format to calculate mean and covariance matrix. The details of calculation of mean and covariance are same as PDA
3. Computer the likelihood ratio for all pixels in the image
4. Compute the likelihood threshold
 - Likelihood threshold $t = \text{Mean} + nS * \text{Standard Deviation}$.
 - Mean is an average likelihood value of the likelihood of the skin pixels which are labelled by colour thresholding method.
 - Standard Deviation only takes the likelihood of skin pixels.
 - nS is the number of SD for manipulating the strictness to filter the outliers
5. Apply thresholding

7.3.2 Implementation

The implementation is defined in: Skin.py

LocalQC(im, nSigma, colourFlag)

im: input image with default data type









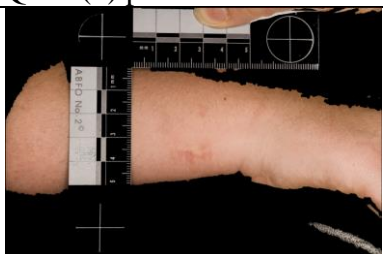
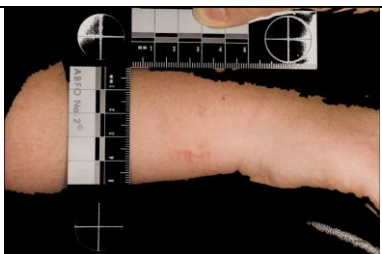
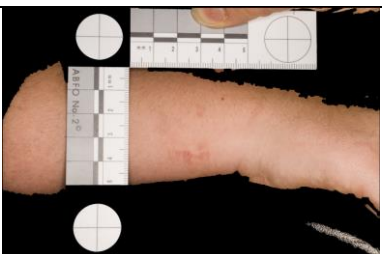
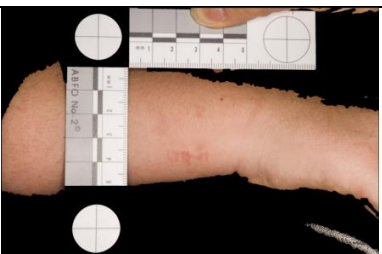
Function returns a binary image. Element with non-zero value means the pixel is skin.

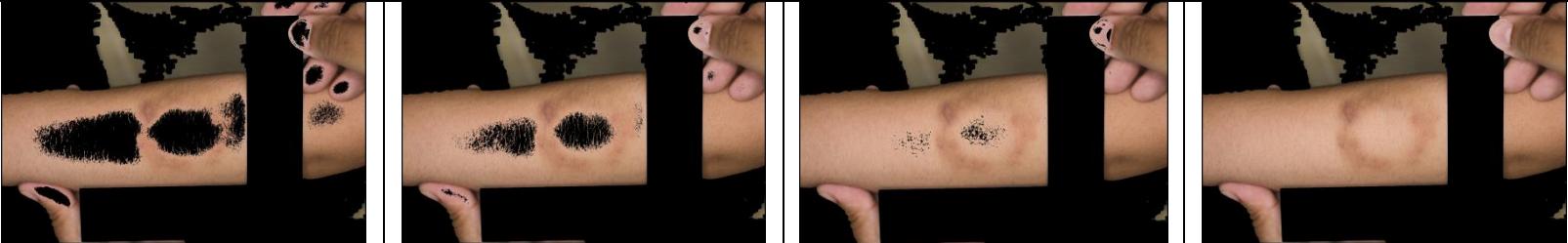

- 1) Generate a mask, Skin Mask using SkinThresholdingV2 function
- 2) Convert input image to other colour space given by third argument using cvtColor function
- 3) Apply Skin Mask to input image
- 4) From input image, obtain all the pixels where marked as non-zero in Skin Mask
- 5) Reshape the pixel as data format
- 6) Compute mean vector
- 7) Compute covariate matrix using calcCovarrMatrix function
- 8) Compute the probability of all pixels using multivariate_normal.pdf function
- 9) Select all the probabilities where the coordinate is marked as skin in Skin Mask
- 10) Compute the standard deviation of data from step 9
- 11) Repeat step 4 but using pixels marked as zero, non skin
- 12) Compute the Likelihood ratio using two sets of probability
- 13) Generate a mask Result where the probability is within range $\text{mean} - \text{sigma} * \text{nSigma}$ to $\text{mean} + \text{sigma} * \text{nSigma}$
- 14) Combine Result and Skin Mask using bitwise_and operation
- 15) Return result

Test program: TestQuadraticClassifierLocal.py

7.3.3 Result and Evaluation:

HSV

	QDA(1) nSigma = 1	QDA(1) nSigma = 1.25	PDA nSigma = 2	Colour Thresholding
1				
Results are not much difference since Colour Thresholding Result has been quite good.				
2				
QDA (1) performs better to remove the sofa brown colour compare to PDA method				
3				
QDA performs better to remove the sofa brown colour compare to PDA method				

4	
	QDA removes a large amount of skin area. The likelihood value the upper background is higher than the middle of the skin.
5	
	QDA removes a bit of skin area on the edge.

While increase the likelihood threshold to $\text{Mean} + \text{Sigma} \times 1.5$ such that no skin area is filtered out, but those images have no difference comparing to Colour Thresholding results. Running this algorithm on YCrCb colour space produce same result.

For Image 2, QDA (1) removed more brown colour pixels on the sofa instead of bright skin pixels, because this algorithm takes more information (background pixel) in the consideration. The bright part of sofa brown colour is also similar to the dark part of the sofa. Therefore, it becomes less likely to skin pixel.

For image 3, values of “Value” channel in HSV of the pixels on the ruler is lower than skin pixels. Those pixels become an outlier.

However, for image 4, the centre skin pixels have a lower saturation value by shading effect. It is very unlikely to become skin pixels.

It is because the likelihood ratio is still so close when it comes to similar background and skin pixels. To address this problem, we may provide more training samples from another dataset to increase accuracy.

QDA (1) and PDA don’t perform better than Colour Thresholding due to the false negative result.

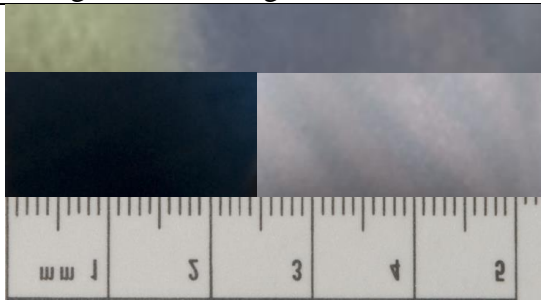

7.4 Quadratic Discriminant Analysis (2)

From machine learning's point of view, more data could achieve better prediction, because of training data coverage. And QDA (2) estimates the skin area using external skin and background data obtained from another dataset for quadratic discriminant analysis to compute how likely the pixel is skin and apply some threshold to filter out the less likely pixels. The test dataset is same as the Local QDA, Dataset2.

7.4.1 Method

Two classes of training data are cropped manually from images in Dataset 1. There are 32 background images and 39 skin images. Background images contain rulers and arbitrary background areas. Skin images contain normal skin, bruise and joint skin area.

There are several training examples.

Background Training Data	Skin Training Data
	

The method is basically same as QDA (1) except the data to computer mean and covariate matrix are from Dataset 2 in QDA (2).

7.4.2 **Implementation**

The implementation is defined in: Skin.py

Function

```
generateQuadraticClassifierParameters(colorSpaceName, imageBG,  
imageSkin)
```

colorSpaceName is String input including “hsv”, “rgb”, “ycc”

imageBG and imageSkin are list type variable, each position stores an image (default image data type) of background and skin.

Function is void type; no value is returned. The function creates two pickle files called skincolorSpaceNameData.pickle and bgcolorSpaceNameData.pickle in the working directory. They contain the covariate matrix and mean vector of their own class.
















The program firstly loops through an image list. During the iteration, image is converted to given colour space and rearrange the pixels into specific format like PDF. After the iterative process, same command is used to generate mean vector and covariate matrix. The variables are then written into corresponding .pickle file by `pickle.dump([co, mean], f)`. this process is executed 2 times for 2 lists.

```
QuadraticClassifierClassify(img, skMean, skCov, bgMean, bgCov,  
nSigma, colourFlag)
```

This function performs same as LocalQC, except that the mean vector and covariate matrix is provided by user in the argument.

Test program: TestQuadraticClassifier.py

7.4.3 Result and Evaluation

	$t = \text{Mean} + \text{Sigma} * 1.25$	$t = \text{Mean} + \text{Sigma} * 2$	Colour Thresholding
1			
2			
The bruise area is classified as outlier			
3			
4			
Result of image 3 and 4 have no difference			
5			
Some of the dark skin areas are classified as non-skin			

Using external training data produces worse result than using local data. Theoretically, using more data should produce better result in machine learning technique because of better coverage of samples. However, in this experiment, it shows that using local data has better result than using more external amount of training data.

It is because our training data only takes a narrow part of skin colour spectrum as learning process, as we have got some skin images from only few patients which may

not cover the skin colour of targeted dataset. Especially, there is no dark bruise training examples like testing image 2 in the training Dataset. It results that the bruise pixels in testing image 2 are more likely as background like dark background example even loose threshold is applied. Background's training examples are not diverse enough. "Training" data used in QDA (1) is directly from the testing image. Therefore, it provides better performance. QDA (2) method is still not ideal.

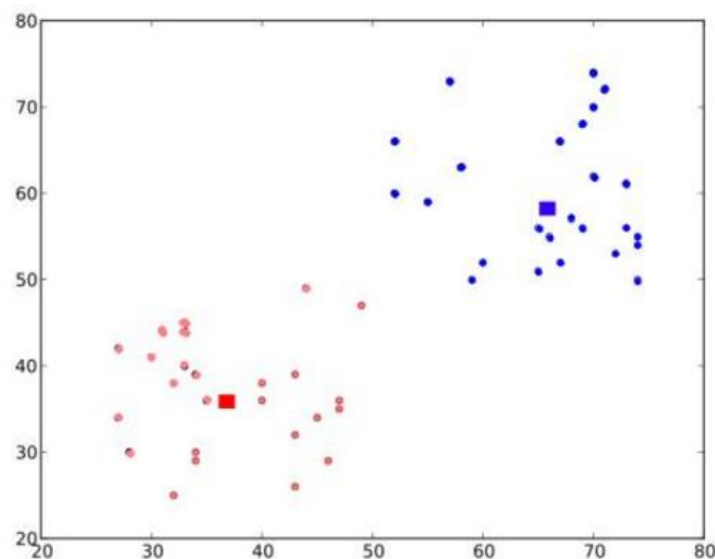
7.5 K-Means Clustering

As limited training data is provided, I go back to focus on local data. Colour Quantization is the process of reducing number of colours in an image [9]. It may be useful to extract part of the area where they have similar pixels like skin layer. This is an example for colour quantization which reduces the number of colours to four.



The theory behind it is k-means clustering [19]. In technical speaking, k-means clustering partition n pixels into k clusters. Each pixel belongs to the cluster with the nearest mean, serving as a prototype of the cluster. Here is an example from OpenCV [20] when $k = 2$ with data in 2-dimensional space.

$$\text{minimize } J = \sum_{\text{All Red_points}} \text{distance}(C1, \text{Red_Point}) + \sum_{\text{All Blue_Points}} \text{distance}(C2, \text{Blue_Point})$$



The 2 squares show the position of 2 clusters such that sum of distances between test data and their corresponding centroids are minimum.

When it is applied to bruise image, the ideal case is that the skin pixels will be clustered together. Based on the clustering result, skin segmentation may be performed to extract the skin region only. Moreover, when more clusters are used, the skin layers with different shading effect may be segmented.

7.5.1 **Method**

Before we start this algorithm, we need to specify:

1. Define number of cluster k

We run $k = \{2, 3, 4\}$ on RGB, HSV, YCrCb colour space to see if background, ruler, skin or even lighting layer could be segmented.

2. Specify termination criteria

The algorithm is terminated when the maximum number of iterations or the desired accuracy is achieved. The accuracy is specified as `criteria.epsilon`. As soon as each of the cluster centres moves by less than `criteria.epsilon` on some iteration, the algorithm stops. In this experiment, max iteration is set to be 10 and epsilon is set to be 1.0 (1 pixel).

3. Select how initial centres are taken

Random initial centres in each attempt is set.

7.5.2 **Implementation**

The implementation is defined in: `Skin.py`

Function: `kMeansClustering(img, k)`

`img`: input image with default data type with user defined colour space


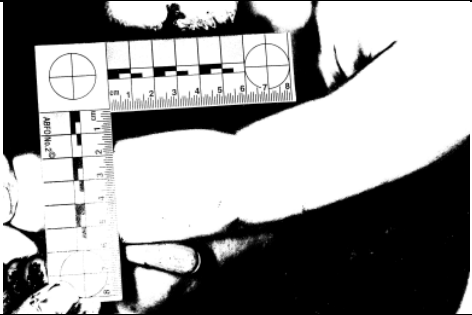
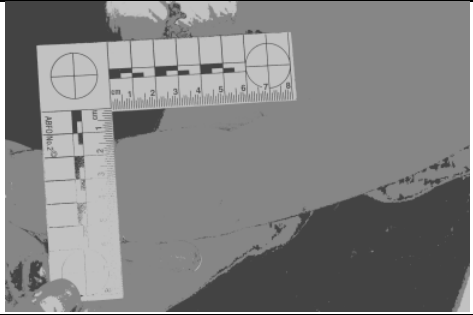
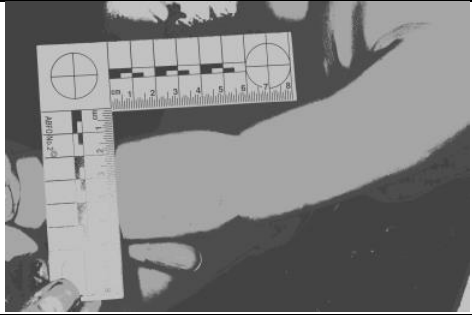



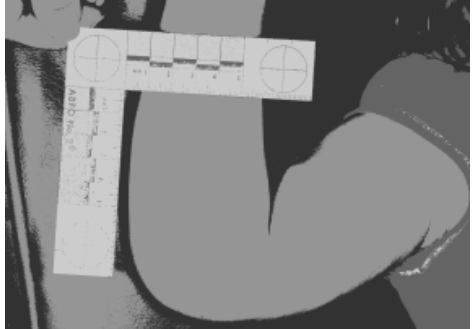



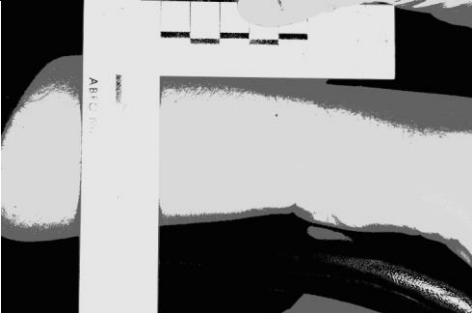
`k`: number of cluster

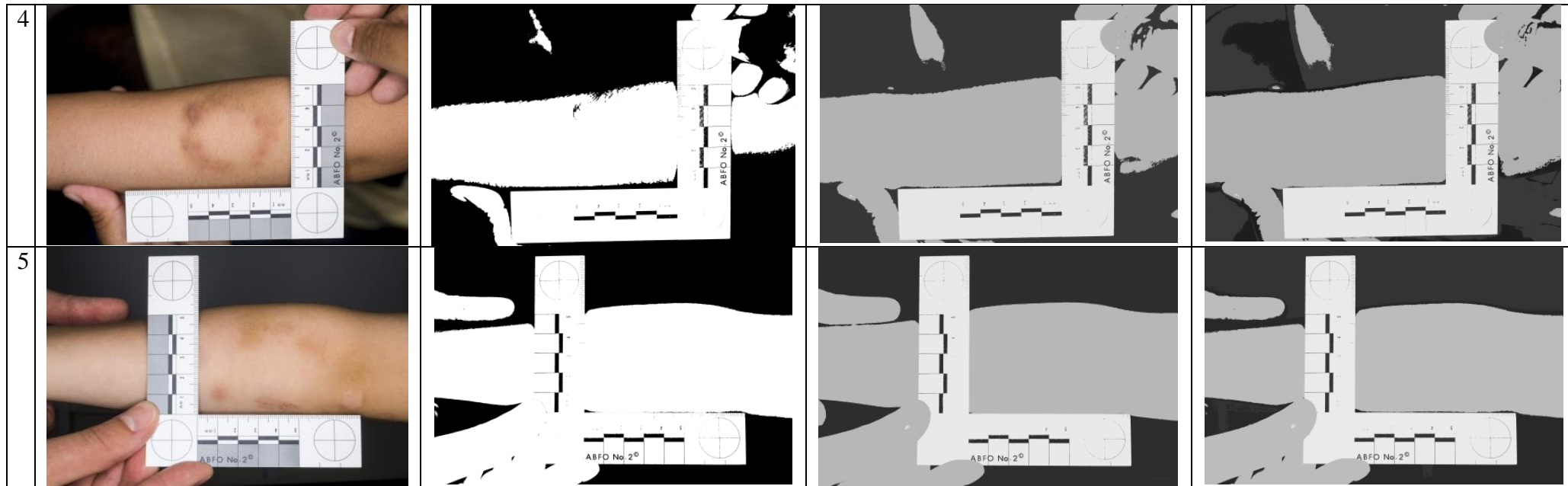
Function returns quantised image.

The implementation references the OpenCV tutorial [19].

Test program: `TestSkinCluster.py`

7.5.3 Result & Evaluation

	Original Image (HSV)	K = 2	K = 3	K = 4
1				
2				
3				



RGB and YCrCb produces similar result.

K = 4 provides most reasonable result. In general results are a bit better than Colour Thresholding, because of the completeness. The edge and area are more smooth. For image 2, sofa colour is still clustered to skin pixels. For Image 3 this algorithm segments shading effect of the skin. It may be useful for bruise feature extraction to deal with shading effect. However, the time complexity of K-means clustering is at least $O(nPixels * k * Iteration)$. It takes long time to compute all 5 images. It is not a time efficient algorithm.

7.6 Summary of Skin Detection

Apart from the above solution, SVM classification is also implemented. SVM can perform non-linear separation by maximising soft margin, where decision boundary is defined by limited number of support vectors. There are quite a lot colour classification done by SVM [21] [22]. However, the implementation doesn't work as usual. The classification turns all the pixels to same label.

To sum up, Colour Thresholding approach is the best performed so far because of low false negative and the speed.

The main problem of the skin detection on our targeted images is that some of the non-skin pixels are highly similar to actual skin pixels. Classification using local or external colour data is not sufficient. Local data could not separate similar pixels. Our external dataset as training data covers too narrow spectrum.

To solve these problems, there are some other potential solutions. It may be possible that to use pre-trained classifier from other research institution, which combines multiple colour models and covers a wide spectrum of skin colour by a huge amount of training data. However, it may not be able to cope with classifier bruises.

Apart from colour classification, some of the skin detections consider texture to give a better and more efficient recognition accuracy of skin textures. There is a study [23] shows that using feed forward neural networks to classify input textures images to be skin or non skin textures.

Nowadays, deep learning becomes one of the dominant technique in image processing community. Skin detection is not an exception. There is a study [24] shows that their deep learning skin detection method performs better compared to other methods such as rule-based, Gaussian model and feedforward neural network.

Skin detection is one of the deep topics in image processing. There are lots of other issues related to skin detection which are to be tackled including diversities of skin colour, scars, birthmark, freckles and hair to affect skin colour. Using basic and simple algorithms could not address these problems. However, the images provided for bit mark matching are white and clean skin type. A few of images for measuring dimension contain scar and hair. Due to time limitation, the proposed improvements and solutions could not be taken. Although my skin detection algorithm is not perfect, it is still acceptable for this project scope. In the coming task, a bruise window will be drawn to cover more background areas including the ruler. Also, feature matching is tolerable for some of the unrelated features. Therefore my skin detection method is sufficient to support the further operation of this project.

8. Draw Bruise Window

8.1 Circle feature based drawing

The bruise area is only a part of the skin. It is important to minimize the area to avoid extract loads of unnecessary features when doing feature extraction from the image. Therefore, boundary for skin area where only contains the bruise is important. The ruler is a good indication for bruise location on skin. Moreover, all circles at the corners of the ruler are important to be included in the image according to forensic odontology photography guidance [3]. The three circles become a main feature to draw a bruise window. To simplify this problem, it is assumed that the bite mark or the bruise is smaller than 6cm x 6cm. Meanwhile, the shortest distance between the circles are longer than 7 cm, which should cover all the bruise based on the assumption of dimension of a bruise.



The unique white colour is reasonably easy to be extracted which produce a binary image where ruler white area is white and lines on ruler and background are black. Also, there are three obvious circles (targeted circles) on the ruler's corner which indicate 3 points of a square window. Therefore, Colour Thresholding and Hough Circle Transform should be useful to find out the coordinate of the window.

8.1.1 Method

1. Gaussian Smoothing
2. Apply RGB Thresholding range from [180,180,180] to [255,255,255]
In the binary image, white represents the white coloured ruler areas and black represents others including the measurement lines on the ruler.
3. Erode the binary image to thicken all the black feature lines
4. Canny Edge Detection to create edge image of the binary image and gradients of the edges

- 1) Gaussian Smoothing
- 2) Use Sobel Edge Detector to find out gradient strength and direction of edges
- 3) Non-maximum Suppression

The edge image is scanned along the image gradient direction, and if pixels are not part of the local maxima they are set to zero. This has the effect of suppressing all image information that is not part of local maxima.

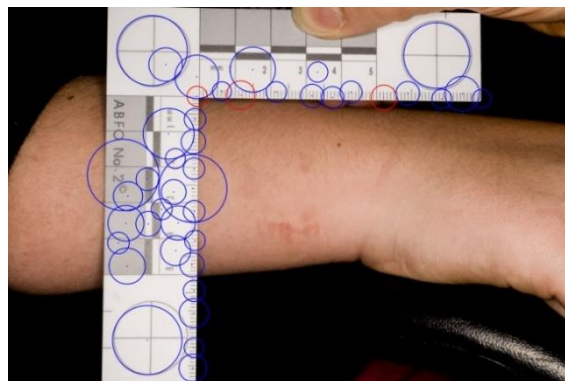
4) Hysteresis range between 30 and 90 [25]

Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit it is immediately accepted. If the value lies below the low threshold it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response. The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur. Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

5. Circle Detection

- 1) Find gradient pairs which the two direction of gradient vectors have nearly 180 degrees' difference by sorting the gradient [26]
- 2) Check the distance of two points of the pairs if they are in the range of the expected $2 \times \text{radius}$. We set the valid radius between 120 and 480-pixel unit
- 3) Record the midpoint of the gradient pairs and its radius in the accumulator if they are in range
- 4) Filter the low response centre in accumulator. Low response is set to be 20 times.
- 5) Merge the closed centre if 2 centres are closer than 250
- 6) Return centre and radius corresponding to the larger accumulator values, will be returned first.

The 3 targeted circles do not always have strongest response in the accumulation space because of the weak gradient. Also, some false circles are also detected. We need more steps to locate the targeted circles at the corners of the ruler. The following image show the failure. Red circles are the three most accumulated circles.



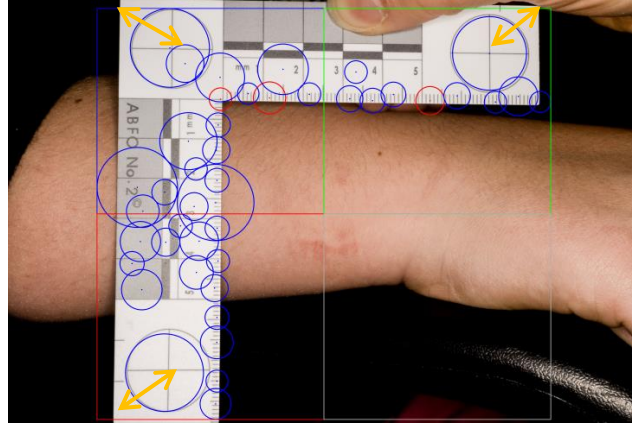
6. Find Circles at corners

- 1) Draw a non-rotated bounding box for all circles & split the bounding box into 4 equal sub-boxes

By my observation, the false circles are always found on the measurement lines. If a bounding box is divided into 4 equal sized square sub-boxes, there should be an invalid sub-box which contains fewer circle centres. Then the three targeted circles could be identified in other three sub-boxes. Each valid sub-box should have one targeted circle.

2) Find targeted circle in a sub-box

All the centres of the circle are labelled to a sub-box by polygon inside test. Each centre of targeted circle in a sub-box should be the closest to the corresponding corner of the bounding box. After that we have three targeted centres in three sub-boxes.



7. Draw window

Now, there are three valid sub-boxes, each of which has a centre of the targeted circle. Drawing a square window requires 2 locations, left top ($P1x$, $P1y$) and right bottom ($P2x$, $P2y$) points. The ruler is not always placed to show us that 2 points. So, there are 4 possible cases. The black square in the 2x2 grid indicates an invalid sub-box.

Case 1: $P1x$ = x value of the centre in grid 3 $P1y$ = y value of the centre in grid 2 $P2$ = the centre in grid 4		1	2	
		3	4	
Case 2: $P1$ = the centre in grid 1 $P2$ = the centre in grid 4		1	2	
		3	4	
Case 3: Same as case 2		1	2	
		3	4	
Case 4: $P1$ = the centre in grid 1 $P2x$ = x value of the centre in grid 2 $P2y$ = y value of the centre in grid 3		1	2	
		3	4	

8.1.3 Implementation

The implementation is defined in: `Skin.py`

Function:

`setUpWindow(im)`

`im`: input image with default data type

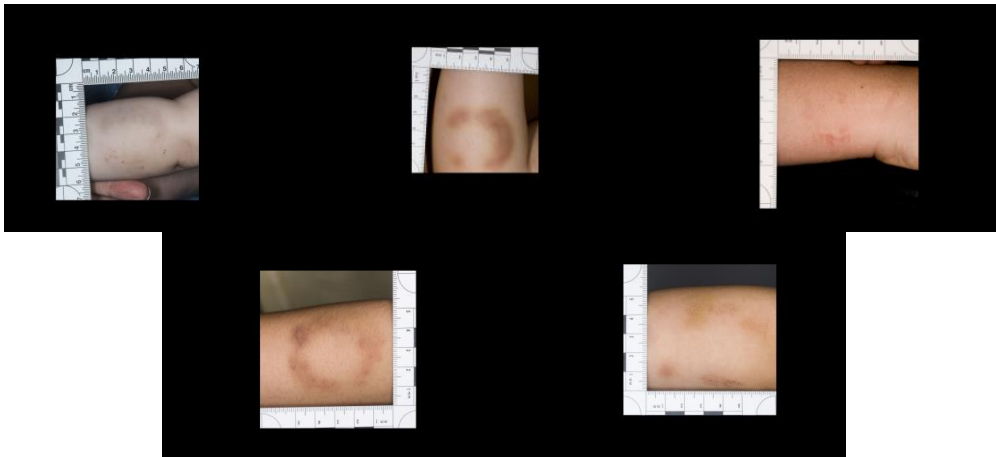
Function returns a binary image. Element with zero value means the pixel is outside of the window.

- 1) Generate a mask, Ruler Mask where non-zero elements are in range
- 2) Extend zero region using Erode function
- 3) Retrieve circles found from Ruler Mask using HoughCircles function
`HoughCircles(mask, cv.CV_HOUGH_GRADIENT, dp=1, minDist=250, param1=90, param2=20, minRadius=120, maxRadius=440)`
- 4) Create an array Temporary Space with all zero elements
- 5) Draw all the circles in the Temporary Space using circle function
- 6) Find all the coordinate with non-zero elements from Temporary Space using `findNonZero` function
- 7) Apply bounding box to all the circles using `boundingRect` function
- 8) Retrieve the bounding box coordinate, width and height
- 9) Compute all the sub-boxes' coordinate, width and height
- 10) Create a 2-dimension list where the row indicates a sub-box
Sub-box/list index 0 to 3. Sub-box 0 is top left, 1 is top right, 2 is bottom left and 3 is bottom right.
- 11) For each circle
 - a. Test the centre in which sub-boxes using `pointPolygonTest` function
 - b. Put it into corresponding list
 - c. Count which list has the fewest centres
- 12) For each list, find out the nearest point to the given corner point using Euclidean distance
- 13) Create an empty mask, Window Mask with all zeros
- 14) Based on three cases uses different points to draw a rectangle on Window Mask using `rectangle` function
- 15) Repeat step1 to obtain a new Ruler Mask
- 16) Extend the non-zero elements using `dilation` function
- 17) Invert Ruler Mask
- 18) Combine Ruler Mask and Window Mask using `bitwise_and` function
- 19) Return combined mask

Test program: `TestBruiseWindow.py`

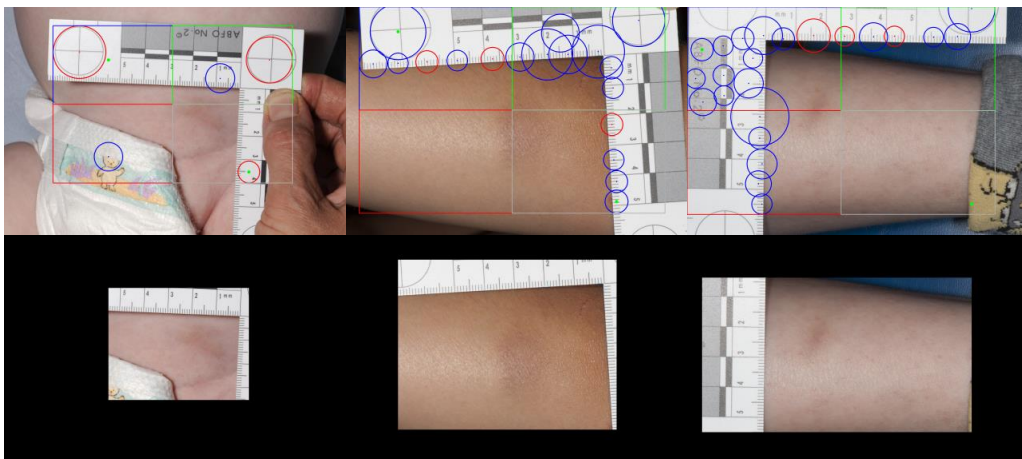
8.1.4 Result & Evaluation

For Dataset 2, all the targeted circles in the images could be located.



The advantage of using circle rather than only thresholding result is that the capability to cope with white background. Imagine the image has some white background. Then a bounding box is drawn on the thresholding result directly. The bounding box may be much bigger than the ruler area. During the process of circle finding, the white background will be ignored to avoid the oversized window, because arbitrary white background may not contain edge and circle features.

In addition, I also received Dataset 3 and 4 after this algorithm was finished. However, the colour of the measurement lines on ruler are faded, the targeted circles are not fully or are not included in some of the images. This algorithm still produces fair results. There are 65 images in total. 53 results have reasonable window which covers majority areas outside the ruler and not covers the bruise, even the targeted circles are not in some of the images or white background.



12 images are failed to draw a valid bruise window. Where 7 of them have no complete targeted circles in the image, which results in small window to cover the big part of the bruise. The ruler in 1 of them could not provide a threshold due to unusual colour of light. 4 of them with unusual illumination, the line of a circle is detected as the ruler because of strong intensity.

To address unusual illumination problems, I have tried skip the first step to not perform a Gaussian smoothing which ensures that the detail is kept. However, it produces more worse results for other images. The targeted circle is broken down into multiple circles. Also, a lot of false circles on the edge of the ruler not only on the measurement lines.

Another method of avoiding using the circle feature on the ruler with unusual illumination is using the same method like extracting the ruler in dentition cast images and check the shape including symmetricity, regularity and size, but it may not be capable for white background.

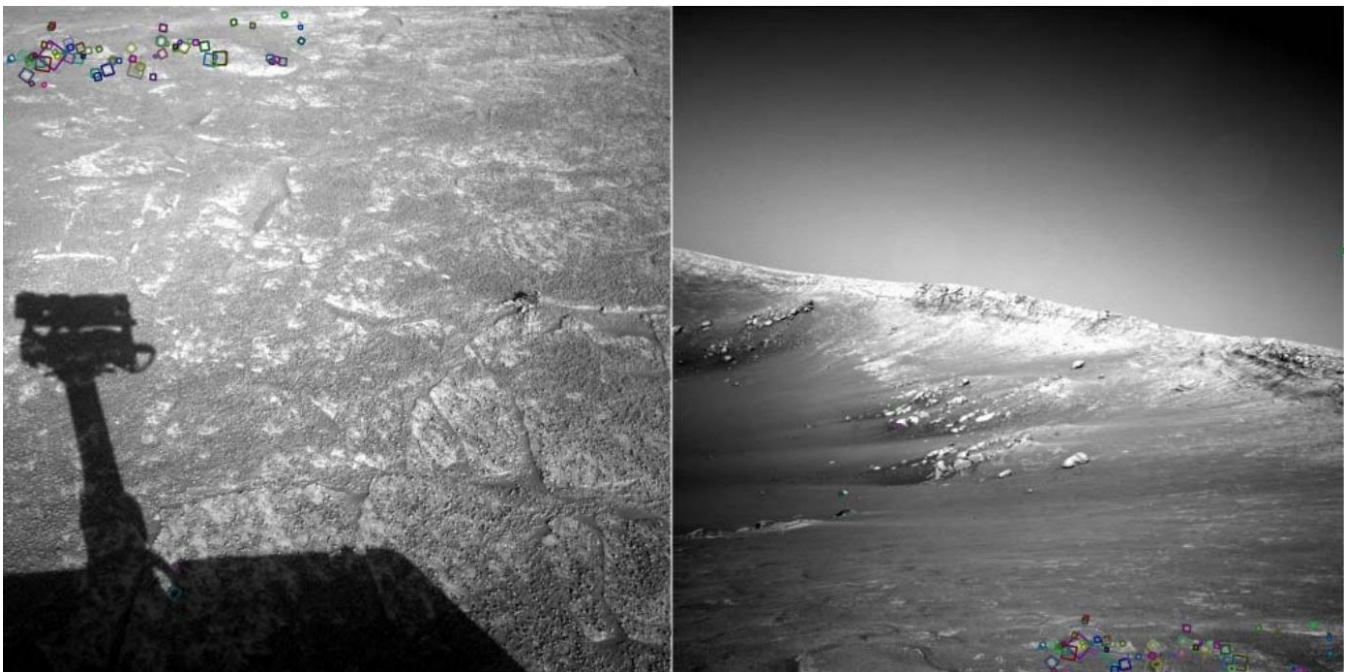
Therefore, that 3 images with unusual illumination are not solved due to time limitation. Overall, this algorithm could draw a bruise window to minimize the unrelated area.

9. Features Extraction

From machine aspect, object matching or mapping is based on image local features. For any object in an image, interest point on the object can be extracted to provide a "feature description" of the object. The feature description may be the orientation of edges, corner.

9.1 SIFT

SIFT, Scale-invariant Feature Transform is one of the most common used method to perform features finding. It is robust to support changes in location, scale and rotation, illumination, small changes of viewpoint, noise, blur. It basically records multi-scale blobs as key points and their spatial gradient distribution as description. This is one of the example from NASA Mars Rover images [27]. SIFT matching is used to recognize a location from different point of views. The coloured squares and circles are the key points.



9.1.1 Method

1. Combine skin extraction & bruise window result
2. Convert image to grayscale
3. SIFT Features

1) Key point detection

Using Different of Gaussian to detect extrema in different scale space. The extremas are the blobs at characteristic scale. Low contrast candidate points and edge response points along an edge are discarded.

2) Compute the dominant orientation of each key point

In region around the key point, the gradients in Gaussian-blurred image are computed. Based on the result, we can generate an orientation histogram and apply weighted gradient magnitude and smoothing to histogram to find out the peak response bin as dominant orientation.

4. Draw circle on location of key point and the circle size is based on scale space

9.1.2 Implementation

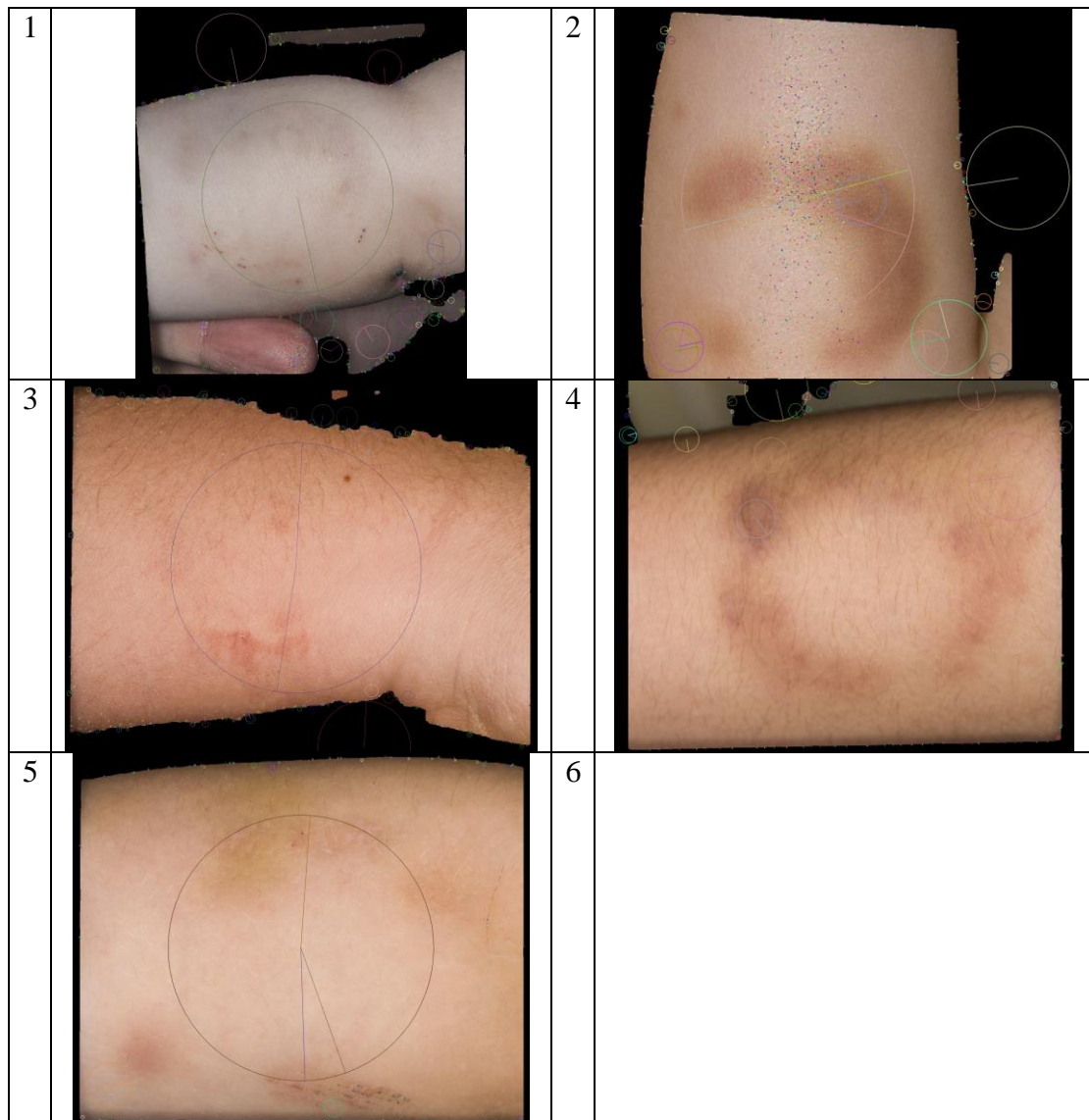
The implementation is defined in: `TestSIFT.py`

Note that, SIFT in OpenCV version 3 is removed from default package. SIFT is packed into the `opencv_contrib` package. The `opencv_contrib` packages contains implementations of algorithms that are either patented or in experimental development. It is recommended to down grade to OpenCV version 2 or install external packages following OpenCV community guidance.

For OpenCV2 users, using `SIFT()` function to create object then using `detect(greyImage)` function to obtain all the key points.

9.1.3 Result and Evaluation

The display result is cropped manually.



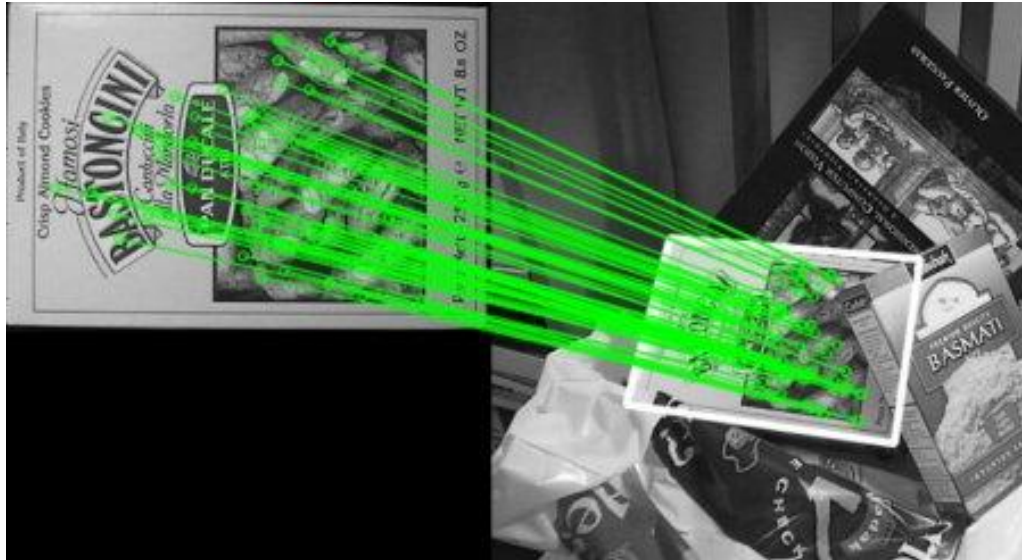
There are only one or two feature points on the bruise. It is because the bruise has no edges, corners features. Blob detection using Difference of Gaussian could not find any extrema. Therefore, SIFT is not able to extract any features from the given bruise.

For the dentition cast features. There are some of the detected features with orientation on the edge of the teeth. However, bruise features are not found.

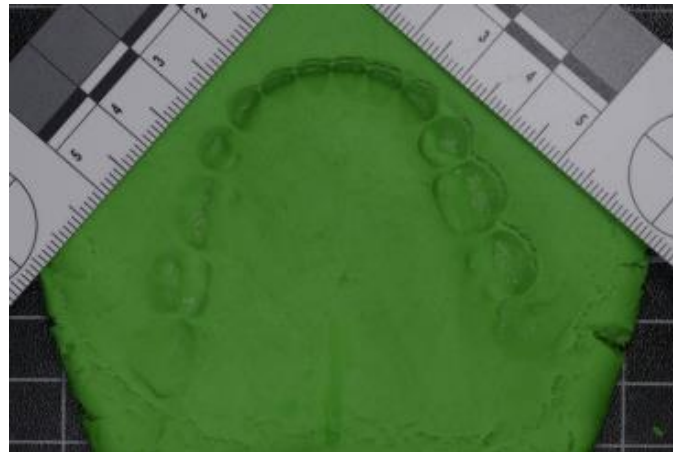


10. Feature Matching

Due to lack of features from the bruise, currently there is no way to match the feature. From the initial plan, if features were found. Feature matching could be performed. The following example shows that how to use feature to match object.



[28]



Let say the bite mark is more obvious like the picture above, which contains edges or lines. Using features found from dentition cast could be possible to perform matching by RANSAC method, where Random sample consensus (RANSAC) is that to find a good subset of these matches that provides a consistent transformation such as an affine transformation. It would finally output the number of matched features, or how likely the feature is matched.

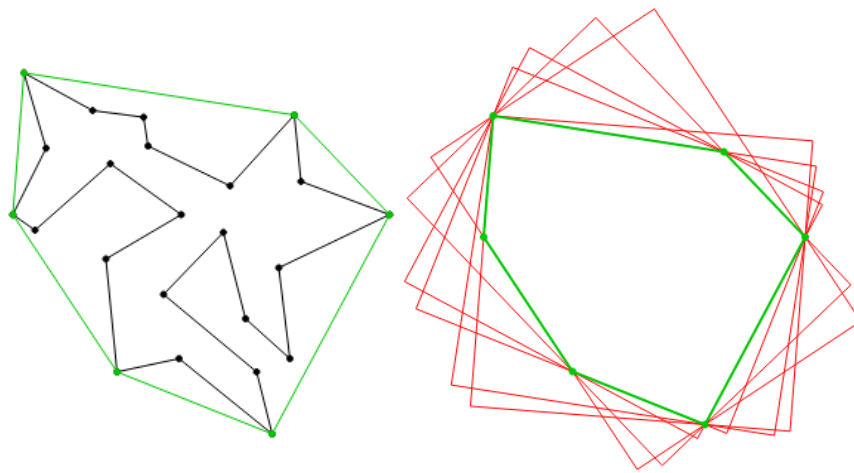
However, the feature is not found from the given bite mark. No match could be performed.

11. Bruise Segmentation and Dimension Measurement

In this section, it is aimed to segment the bruise from the skin. After the segmentation, the boundary of a bruise could be defined. Then, a minimum bounding box is drawn on the bruise and measure its dimension in pixel units. Pixel unit could be converted to centimetre after further process. Also, it is assumed that every bruise image only contains a single bruise.

For the segmentation, colour quantisation, thresholding with central weighting region growing methods are experimented and evaluated.

For dimension measurement, minimum-perimeter bounding box is used to define the dimension. Minimum bounding box means an oriented minimum bounding box (OMBB) which is the smallest-area enclosing rectangle of a polygon has a side collinear with one of the edges of its convex hull.

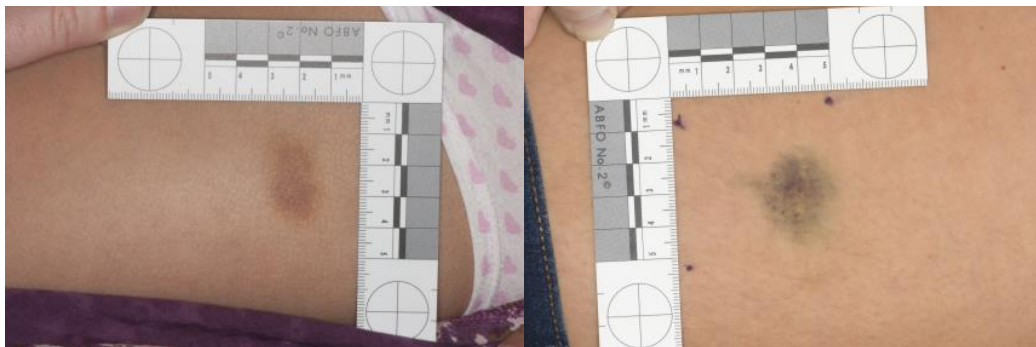


[29]

It provides a consistent and identical method to define the dimension of a set of points. It is significant to eliminate the human subjectivity from forensic science point of view.

Bruise images are different in terms of the shading effect which is mentioned in colour thresholding section or noise level including skin detection failure. An image with less shading effect and noise is more easy to segment. Therefore, the initial stage, some of the “easy” images are selected to be processed.

The following images are examples of few noise and less shading effect images. They are selected to study for our initial algorithm.



11.1 Colour Quantisation & Region Growing

Colour Quantisation would be useful to classifier skin and bruise pixels in 3-dimensional colour space. The cut off is dynamic, especially the classification of faded bruise area depends on both bruise itself and the skin which results a fair classification.

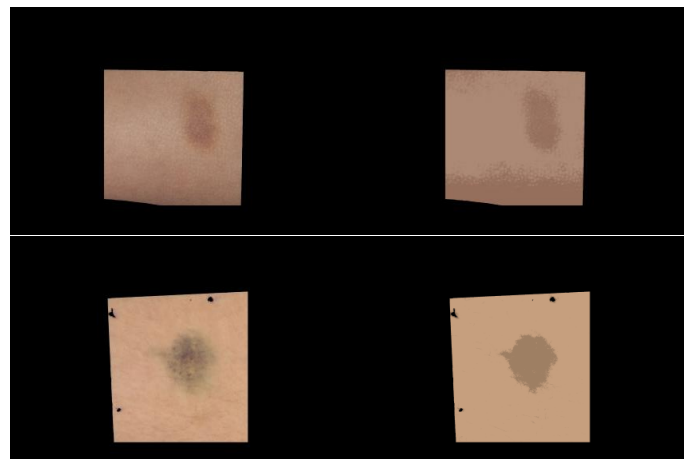
Moreover, there are still some pixels with shading effect on the skin which are classified as bruise. They need to be segmented by location information given that a bruise region is always located near the middle of the image. It may be possible to select bruise pixels which are nearer to image centre as seeds to grow the bruise region.

11.1.1 Method

1. Using thresholding to remove purple dots and mole.

2. Colour Quantisation in HSV and RGB space

The process is same as K-Means Clustering in Skin Extraction section. This time we use $k = 3$. They are bruise, skin and the black pixels. The map of labels is also stored as grey image. In the result of Colour Quantisation, you can assume that one colour belongs to one label. Also, we need to find out which label belongs to bruise by computing mean intensity of pixels grouped by labels. The ascending order of mean intensity should be background, bruise and skin. It is because of the nature of bruise.



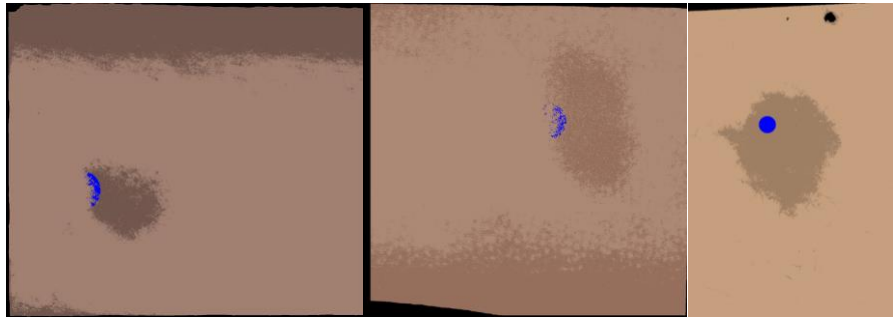
3. Seed selection for region growing

If you look at the quantisation result detailly, you may see there are some noise that the bruise labels are distributed on different part of skin. If a seed in the noise area is selected, the seed cannot grow and the result will be failed.



As bruise region is always located close to the middle of the image, seed selection starts from near centre of an image. Initially, a circle with radius 50-pixel unit is drawn in the centre of image to indicate seed searching area. Then, all the bruise labels inside the search area become potential seeds. If there are less than 500 seeds, the searching area is extended by radius +50 and check the number of bruise labels, till the there is more than 500 potential seeds in the searching area or the searching area is bigger than the skin area. This step could avoid getting noise labels.

They are examples that bruise labels in searching area are blue coloured.



10 actual seeds are randomly selected from the potential seed pool to grow region independently. If the seed grows less than 2500 pixels, result will be rejected. Random seed selection is better than distance based selection because random selection may avoid choosing the fragmented labels which cannot grow.

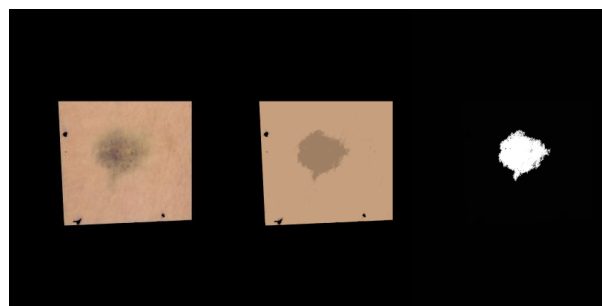
4. Region growing

Growing the seed is simple, just check its 8 directions to see if any neighbour pixels have same label, then neighbour become a seed as well till all the seed cannot grow. This example shows that the middle is seed which has label 0 and grows to a neighbour with same label.

1	1	0
1	0	2
1	2	2

5. Combine results

Combine those results from 10 seeds (or less) to avoid getting result of using fragmented seed.



6. Minimum bounding box

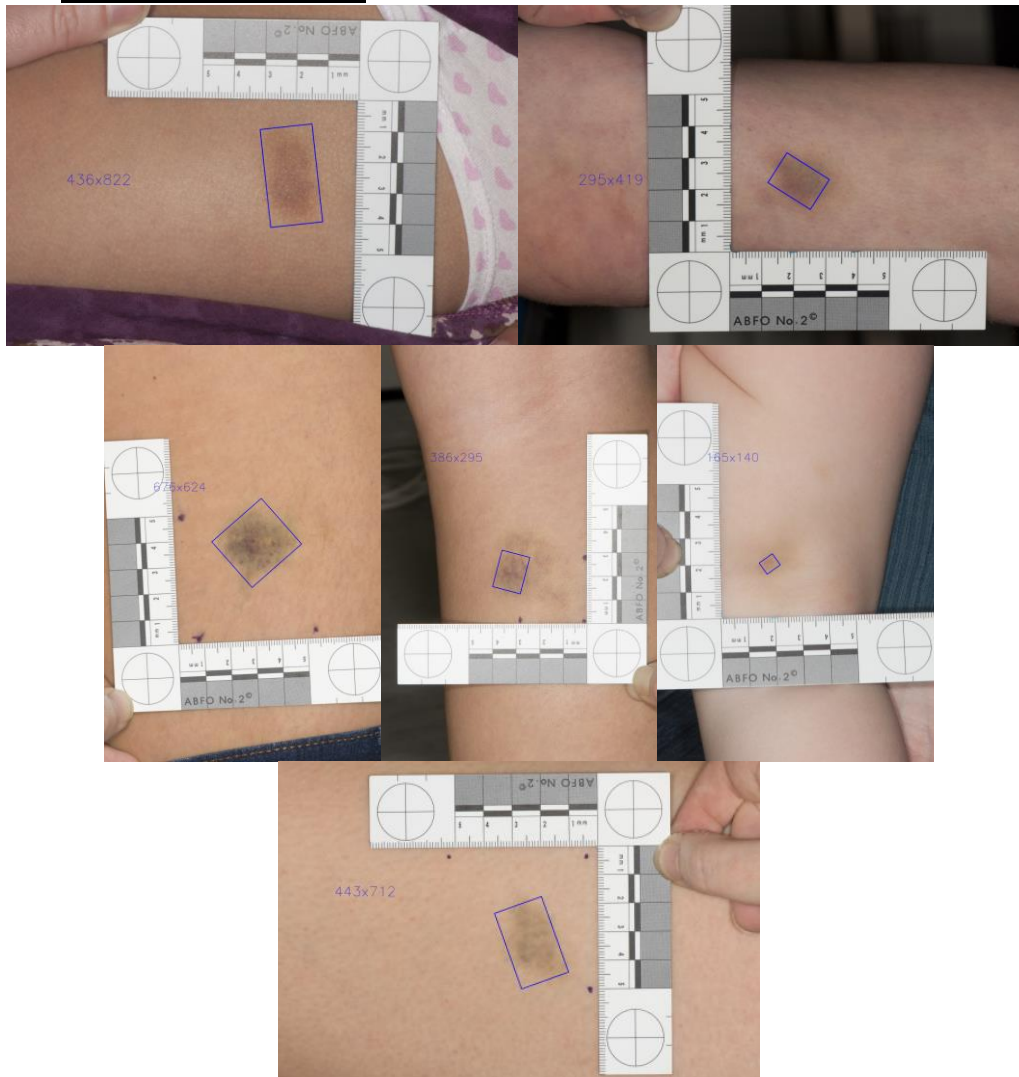
11.1.2 Implementation

The implementation is defined in: TestKmeansRegionGrowing.py

- 1) Read images from directory
- 2) For each image
 - Pre-processing--
 - a) Apply skin detection on image -> skinMask function
 - b) Generate Window Mask using setUpWindow function
 - c) Generate Dot Mask using RemovePurpleDot function
 - d) Generate birthMask using RemoveBirthMark function
 - e) Combine 4 masks to Window Mask
 - f) Generate Window Image by applying Window Mask to the image
 - Colour Quantisation--
 - g) Convert Window Image to HSV space
 - h) Generate Cluster Image, Label Map using K-means clustering to Window Image
Each value in Cluster Image is the colour corresponding to the cluster. Each value in Label Map is the corresponding label.
 - i) Convert Window Image to Grey Scale
 - j) Compute mean intensity of Window Image where the pixels grouped by labels
 - k) Find the medium intensity and corresponding label
 - Region Growing--
 - l) Generate a mask, Flag Map where non-zero element indicates bruise label
 - m) While loop:
 - i) Set radius = 50
 - ii) Create a mask, Circle Mask with a filled circle with initial radius in centre
 - iii) Apply Circle Mask to Flag Map
 - iv) Store all the coordinates of non-zero elements of Flag Map to Point Candidates
 - v) If size of Point Candidates ≥ 500 break the loop. Otherwise, radius + 50
 - n) For 10 times:
 - i) Repeat step l and invert it to Mask Grow
 - ii) Extend the 4 borders from 4 sides by 1 pixel from Mask Grow for region growing function
In OpenCV the mask with non-zero indicates a border than the seed cannot grow. And border + 1 is requirement for the method.
 - iii) Use a random index to select a Seed from Point Candidates
 - iv) Region growing using floodFill function given
 - v) Append the result to a list
 - o) Use bitwise or operation to combine all results in the list
 - p) Store all the non-zero
 - q) Coordinates in Bruise Mask to Cnt
 - r) Find minimum bounding box given Cnt using minAreaRect, BoxPoints function
 - s) Draw the box on original input image using drawContours function

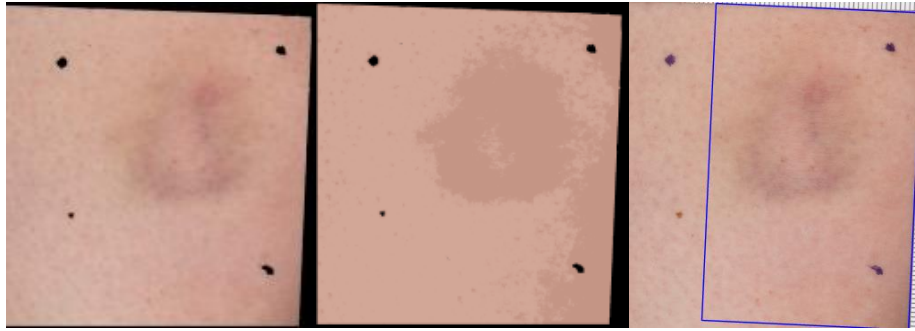
- t) Draw the text of the box dimension on original input image using putText function
- u) Write output image using imwrite function

11.1.3 Result and Evaluation

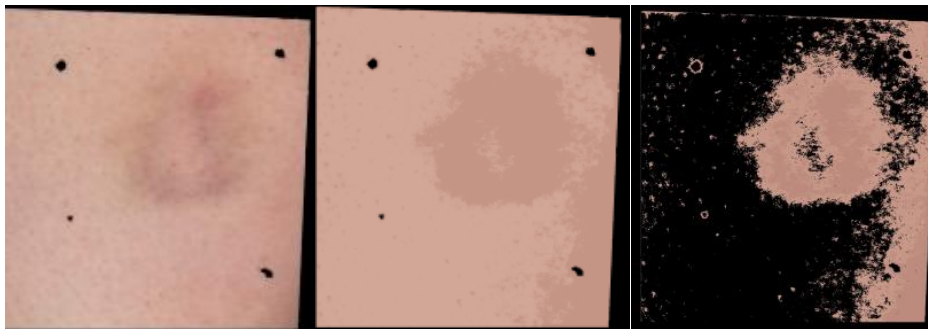


While using HSV and RGB colour space, 7 of 53 results show that the algorithm works. All that 7 images are only contain clear and obvious bruise as well as less and segmented shading effect. For these 7 images, this algorithm successfully labelled bruise plus shading areas as a group and skin as another group. Also, region growing can select the bruise area rather than shading area. Minimum bounding box estimation could produce the dimension of the bruise in pixel unit.

However, there are other problems such that the segmentation fails when the bruise and shading areas are connected, because the seeds grow all the connected neighbours like the following example.

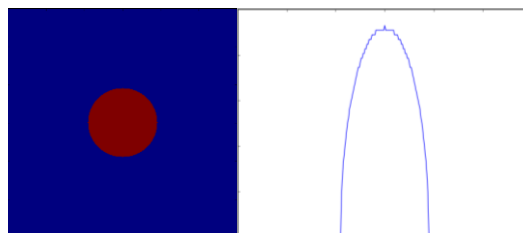


Therefore, re-classification is performed again based on the result above to see if shading and bruise could be segmented. And here is the result.



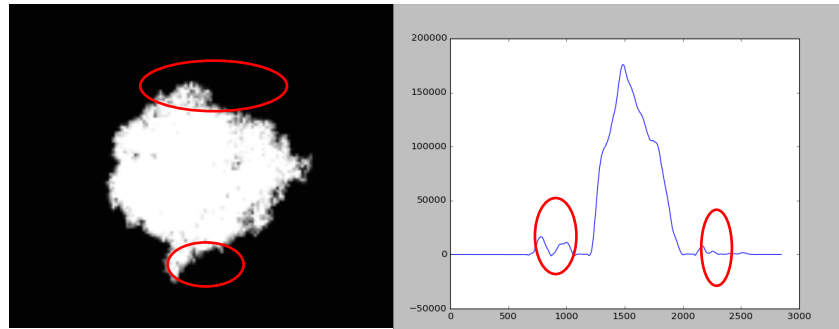
As you can see the bruise is segmented into two layers, the darkest part and the faded part. And the shading is segmented as well. Ultimately, clustering cannot segment bruise and shading.

Label counting method is also considered. For example, there is a bruise image. The bruise labels are vertically and holistically accumulated column by column and row by row. The accumulated series is plotted on 2D space, x and y axis represent the row/column index and number of bruise labels accumulated respectively. Left image indicates bruise image where red is the bruise labels. The right plot is the vertical accumulation result.



As you may see, a bell shape curve is in the plot. The minimum extrema are the clear cut off point of the object. However, the bruise in life is not always circular as the above example. To smooth the accumulation series, a Gaussian weighting may be useful to generate bell shaped like curve to find out the potential cut off given that the bruise is always in centre of image.

Let's say the below example is after a perfect Gaussian weighting and smoothing for vertical accumulation which flattened all the shading votes. As you can see the outer peaks are also part of the bruise. Minimum extrema are not able to identify the cut off in this situation.



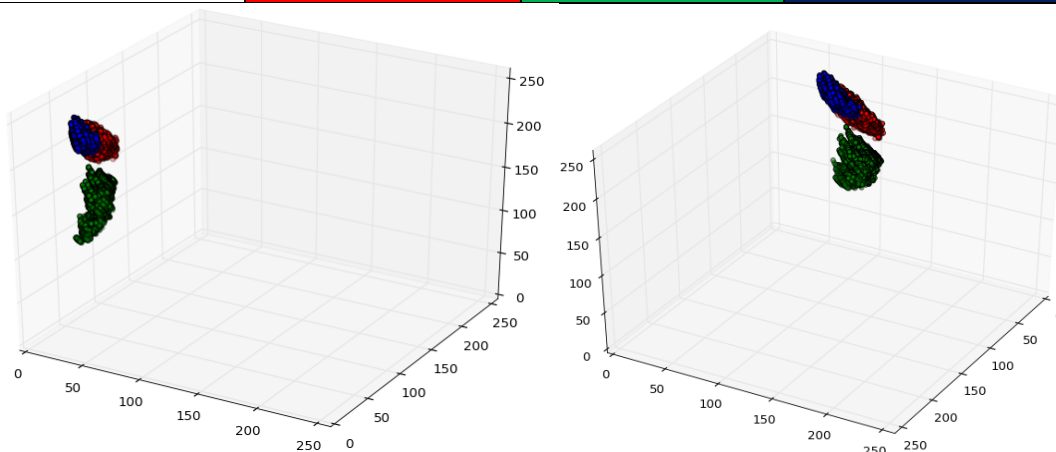
If the small peaks are ignored, the bruise will be cut. If the peaks are selected, for some shading connected image, the bruise will be over selected. There is a problem to how to segment the bruise and shading area.

Moreover, the colour quantisation clusters the skin and bruise pixels as a group, when it comes to the situations that the shading pixels are much darker than the bruise or the bruise is not too obvious. It means the distance of skin and bruise pixels are closer than the distance of shading and bruise pixels. The following examples show that strong shading effect on the right side.



This method is arguable that from forensic science point of view, although this method successfully draws bounding box on the bruise. The size of a bruise especially classification of the faded bruise pixels in k-means clustering highly depends on the amount of and the level of the shading effect. To explain this concept a diagram is produced. There are 3 small images cropped from example image in HSV colour scheme. And plot it in the 3-dimentional space.

Example image	Bruise	Shading area	Skin
	Red dots	Green dots	Blue dots



(Same plot with views from two angles)

When $k = 2$, 2 clusters will be roughly located in inside of green group and between blue and red groups. It results that bruise and skin become a group.

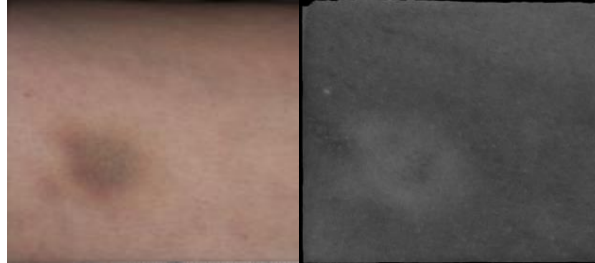
Imagine 2 cases. First case is that there only few green dots (less shading area). The seoncd case is that there are more few green dots (but not many as the above example).

For case 1, the bruise cluster is shifted to more central to the red group comparing to case 2. It results more “marginal”/”faded” bruise pixels will be classified in case 1 which is more reasonable from scientific aspest.

This algorithm is passively affected by the level of shading. Therefore, it is considered to experiment on some other methods to suppress the effect of shading. The next algorithms are required to add some methods to increase the chance that the bruise to be selected or reduce the chance that the shading to be selected.

11.2 Central Weighting & Thresholding on Saturation

In this section, central weighting is given to the bruise image to avoid domination of shading area. In HSV colour space, Saturation is one of the main components to segment skin and bruise/shading. The following example images shows the saturation response of an image.



Although the saturation between bruise and shading are similar, the saturation can be decreased from centre such that, the centre area takes the more weight compare to the far location by applying a grid with weighting, shading in the edge become less response in terms of saturation compare to the bruise.

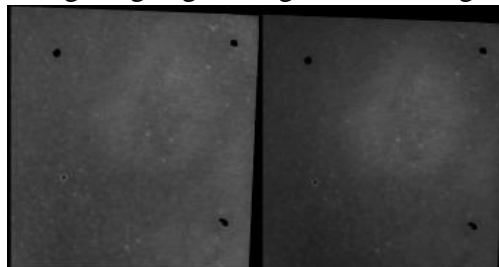
11.2.1 Method

1. Generate a grid with weighting

Generate a Gaussian kernel with sigma 6 and size $N \times N$ where N is max of the image width or height. And crop it to image size. Then rescale all the values in range from 0.6 to 1.

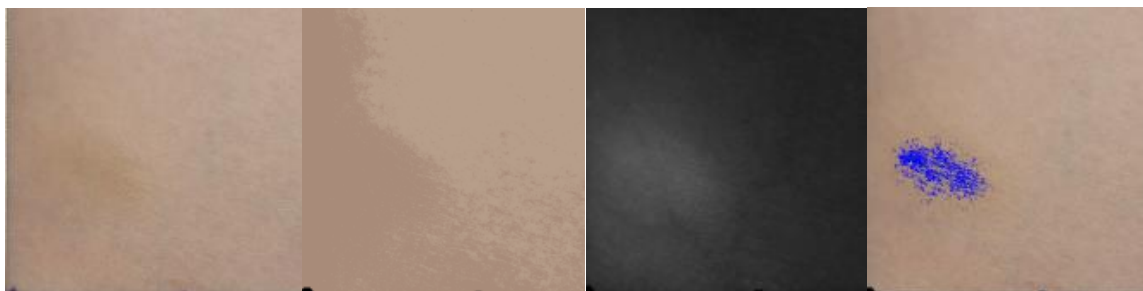
2. Apply the weighting to the saturation of the image

Left image is before weighting, right image is after weighting



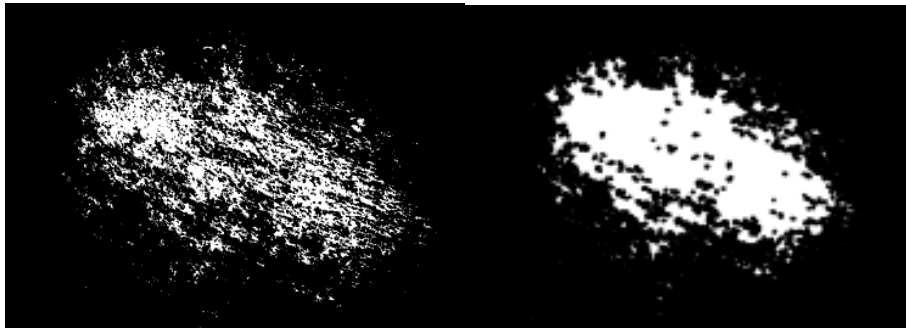
3. Choosing Threshold

It is reasonable that the maximum response from the saturation channel is the darkest part of a bruise. A lower bound is required. Initially threshold $t = \max * 0.85$ is set. If less than 5000 pixels are selected, a new threshold $t = t * 0.9$ will be applied till more than 50000 pixels are selected. It ensures that the selected pixels are not only arbitrary extrema. Images from left to right are original, k-means, saturation after central weighting, and the threshold result.



4. Morphometric Closing

Closing with 9x9 kernel is applied to connect near pixels to ensure the thresholding result is not over dispersed.



5. Region Growing to remove unconnected region and noise

6. Minimum bounding box

11.2.2 Implementation

The implementation is defined in: `TestCentralSaturation.py`

- 1) Read images from directory
- 2) For each image
 - Pre-process--
 - Same processes as Colour Quantisation implementation a to f
 - Central Weighting--
 - a) Convert Window Image to HSV space
 - b) Obtain Satiation Image by keeping Saturation Channel from Window Image
 - c) Obtain Gaussian Weighting Kernel using `GetCentralPriorKernel` function
 - d) Rescale the values of Weighting Kernel from 0.6 to 1
 - e) Update Saturation Image by applying Weighting Kernel
 - Thresholding--
 - f) Find the max saturation value and set threshold $t = \text{max} * 0.85$
 - g) Generate a mask, Result where non-zero pixels indicate the coordinate that value is $> t$ in Saturation Image
 - h) While loop
 - i) Count the number of non-zero elements in Result
 - ii) If the number is small than 50000, set $t = t * 0.9$ and repeat step m
 - i) Apply morphometric closing to Result using `morphologyEx` function
 - Region Growing--

Same processes as Colour Quantisation implementation l to t

Function `GetCentralPriorKernel(imSize, sigma)`

`imSize` is [`imHeight imWidth`] like array. Sigma is value for Gaussian Distribution

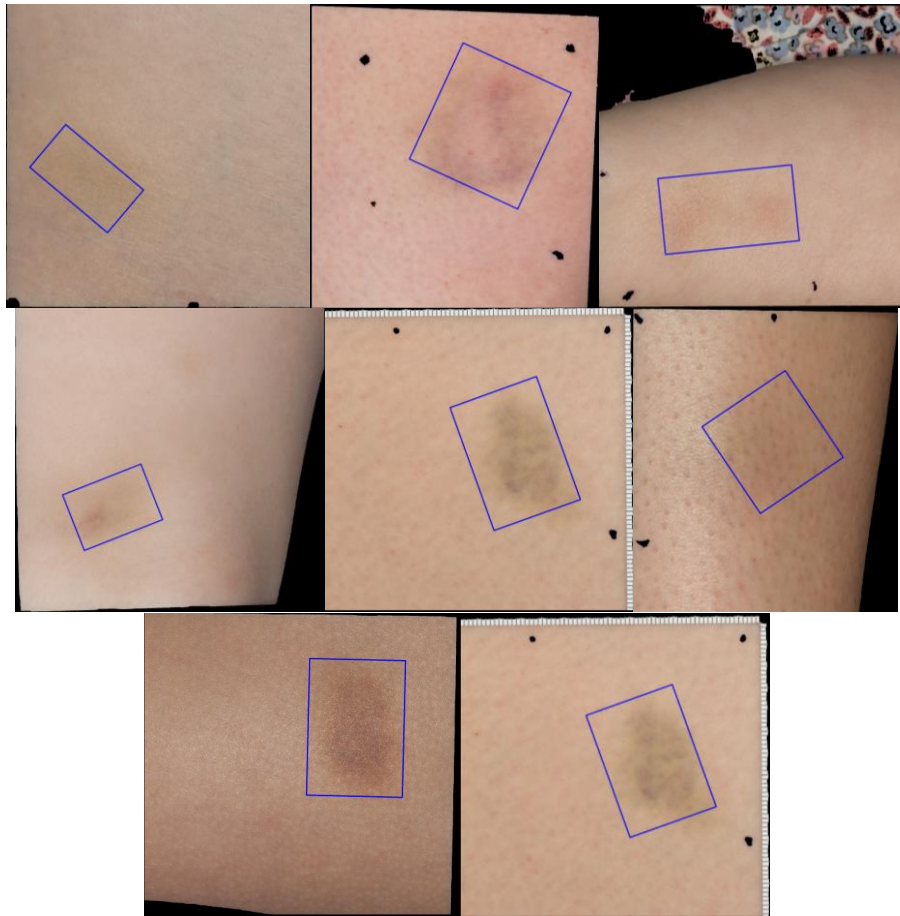
Function returns Gaussian kernel with same size as `imSize`

- 1) $kSize = \max(imSize)$
- 2) Generate 2-dimentional Gaussian Kernel with size $kSize * kSize$ using two 1-dimentional Gaussian kernels
- 3) If the height and width of 2d Gaussian Kernel and `imSize` are not equal
 - a. If $kSize > imWidth$
 - i. Compute the offset for unwanted part of the kernel by $(kSize - imWidth)/2$
 - ii. Select all the row from kernel with column index between offset to $kSize - offset$
 - b. Else
 - i. Compute the offset for unwanted part of the kernel by $(kSize - imHeight)/2$
 - ii. Select all the column from kernel with row index between offset to $kSize - offset$

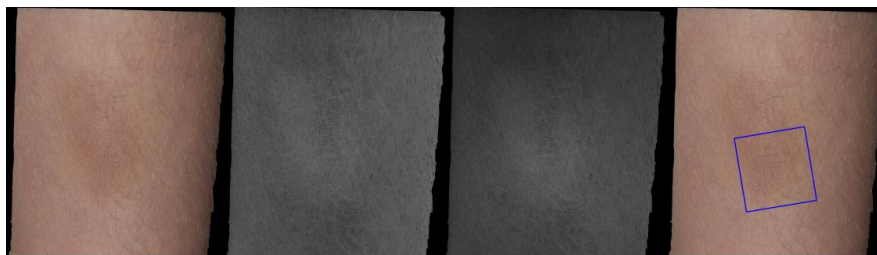
Return cropped kernel

11.2.3 Result & Evaluation

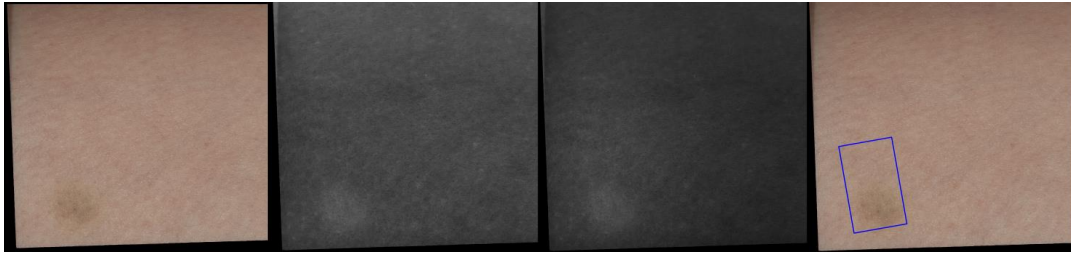
8 out of 51 results are acceptable as my observation.



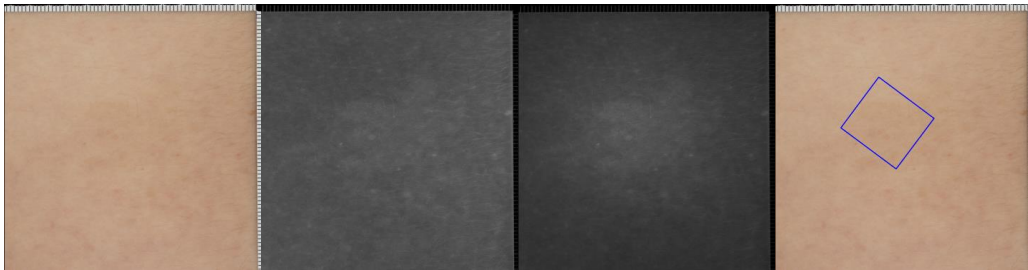
The weight may be incorrect for some images. The first reason is that the weight is not based on any features or environment from the image itself. If image contains a big blob of bruise, the response of the far bruise area will be decreased. The size of a bruise becomes smaller. The left to right image shows HSV original bruise, Saturation before weighting, Saturation after weighting, and the result.



Although the bruise is always close to the image centre, there are some cases that the response of the main part of the bruise is decreased. Then the response of non-bruise pixel in centre is increased. The area of a bruise becomes over estimated. The left to right image shows HSV original bruise, saturation before weighting, saturation after weighting, and the result.

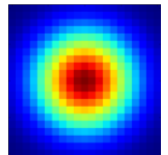


Also, there are some ambiguous situations. From my personal observation, I do not discover the bruise on the unweighted image. After the weighting, a slightly bright blob appears. It is not convincible to say the blob is made my weighting or the original bruise itself.

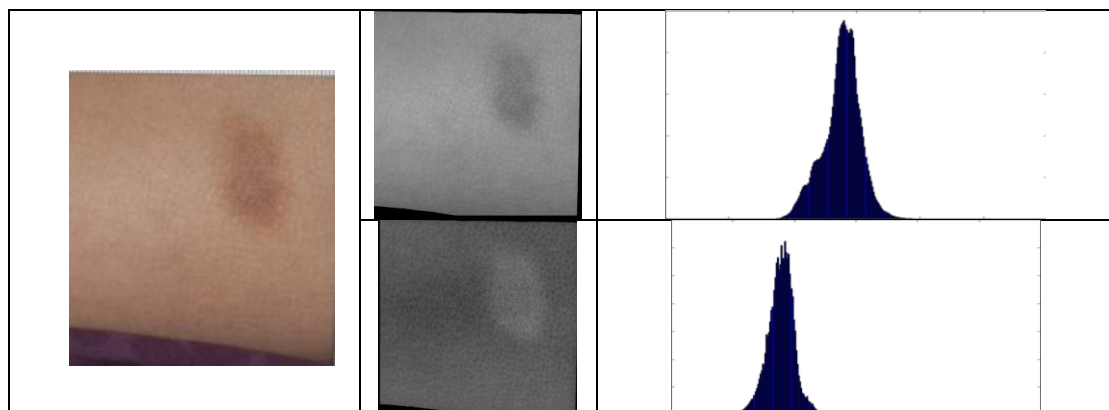


The current threshold selection is another problem. Threshold will get relaxed when the number of pixels in result is smaller than 50000. These criteria control the size of the detected bruise incorrectly. When there is a big and partially faded bruise, the faded part will not be included. Because the initial threshold may already contain more than 50000 pixels, algorithm then stops.

If the threshold is determined by checking the size growth of centre blob, it may over select the bruise area. After apply centre weighting, the saturation response decreases from centre, lower the threshold will include more pixels where just far from centre.



Apart from the above thresholding approach, k-means 1D clustering and Otsu's thresholding are also experimented, but they perform badly. The idea of Otsu's thresholding is to look for two peaks then approximately take a value in the middle of those peaks as threshold value. However, the bruise response on saturation or Grey Intensity are not strong as to be segmented. The following example shows the Greyscale and saturation of an original obvious dark bruise without shading effect and its histogram.



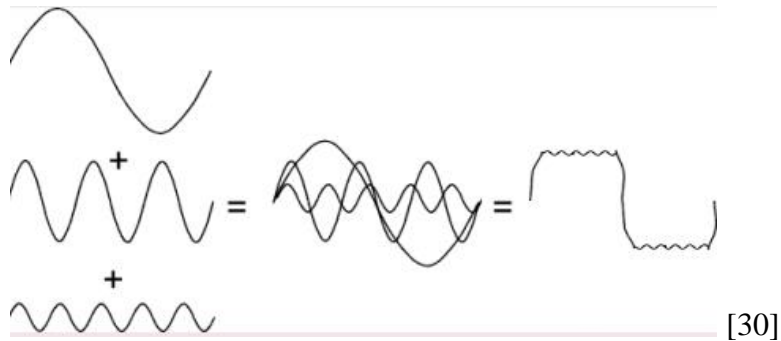
11.3 Lighting Correction with Fourier Transform

When the flash light hits on the blended surface, some of the surface received less amount of light. The area becomes dark where colour is similar to the bruise. Especially, the cylinder shape like limbs. Lighting artifact attenuation could be approximated and corrected.



To flatten the lighting effect, the image can be converted to grey scale where the pixel value is presented as intensity. The dark pixel contains low intensity. Then the intensity is accumulated row by row horizontally and column by column vertically to form two signals. It is then normalised by dividing its highest value. After doing so, we would use the Fourier transform to obtain magnitude information to decide which direction such as vertical or horizontal to be correct.

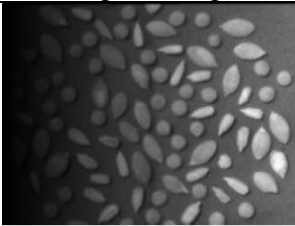
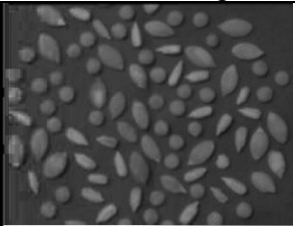
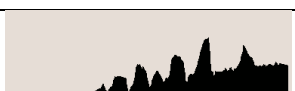


The idea of Fourier Transform is that any digital signal can be decomposed into purely sinusoidal components. [30]



For the left image, the signal accumulated horizontally will form an up-side-down cosine-like curve which indicates the lighting effect. And the signal accumulated vertically will form relatively flat with small peaks shape. After normalisation, they are passed to Fourier Transform. The strongest magnitude in low-frequency range could be found in the signal with lighting effect. The up-side-down cosine shape signal is contributed by the dominant peak in low-frequency range in Fourier Space. Then the direction of lighting effect to be corrected is identified.

Based on the accumulation signal S , some weighting could be assigned to grey scale image from the same direction to flatten the lighting difference. S is upside down cosine like curve. The new intensity is estimated as $I = I_{old} * 1 + (1 - S(n) + c)$ where c is a constant to control the level of correction.

The following example shows that the intensity is accumulated column by column horizontally. Then using linear weighting to correct the lighting on greyscale image [31].

Original Image		Corrected Image
		
The intensity of a row	Flattened Linear signal S	The intensity of a row after correction
		

Due to the time limitation, the above method has not been implemented. However, this current method could only deal with the limbs which are similar to cylinder. There are number of bruises on other body part like face and back which are bulged in on the skin surface rather than cylinder like.

11.4 Summary of Bruise Segmentation and Dimension Measurement

K-means clustering method seems to be most reasonable in the terms of personal observation. However, it is highly affected by the shading area. Also, segmentation method is not working to segment bruise and shading area. Lighting effect is also a major factor to block the bruise extraction or turns the bruise invisible in saturation channel.

12. Future Work

Dentition cast extraction:

There are still two non-convex shape casts which the backgrounds have not fully removed. It may be useful to reapply colour segmentation method to eliminate the background.

Skin detection:

Although the current thresholding skin detection method is acceptable under this project scope, it is still recommended to use the state of art skin detection algorithm to achieve a better result. It is because the future datasets may potentially contain more challenging backgrounds or ambiguous object behind. Therefore, it is beneficial to enhance the skin detection performance to eliminate non-skin areas as many as possible. Since neural network and deep learning techniques on skin detection has performed much better than simple thresholding solutions, it may be useful to build from the recommendation rather than experimenting new methods.

Bruise window drawing:

There are some drawing failures due to ruler circle feature is not included or only partially included. Although including the ruler circle feature in the forensic images is recommended, there are some forensic images do not contain the or partially contain the feature only. It may be necessary to create non-circle feature based drawing method to cope minority cases.

Be aware that an assumption is defined in this project which the bruise could be bounded by the circles, but the real-life bruise does not always fit into this assumption. If the project scope is extended, this algorithm must be re-designed.

Features Extraction:

The current received dataset is challenging, due to no features on the bruise at all. However, image processing technique is still useful for bite mark feature extraction. According to the Forensic Science International Journal, there are still a lot of bite mark matching problems that have not been scientifically or equally solved, in which contains a lot of obvious line/edge features of teeth on the skin. It is useful lower the difficulty and starts from those images such as the control set in dataset 2 as initial feature extraction and matching algorithm testing.

Bruise Segmentation and dimension Measurement:

The current bruise segmentation is majority affected by the shading effect. It is important to balance/flatten the illumination effect first. Then performs segmentation. Also, the hair or scar removal algorithm should be implemented to reduce noise which is not related to bruise. Retinex [32] lighting correction method is worthy to try out. Retinex Image Enhancement is convert the input image to the enhanced image which is independent of the illumination source.

The current result is only based on pixel unit. It must be converted to other measurement unit for forensic purpose like centimetre. However, the ruler in different images has different scale. It is necessary to find out the ratio between pixel unit and centimetre or millimetre.

13. Conclusion

The goals of this project are that to use image processing technique to match the bite mark to the corresponding dentition cast and measure the dimension of a bruise. Although both goals have not been reached, all most all the pre-processing tasks have been completed. Those tasks are necessary and significant gateways to reach the goal.

There are series pre-processing tasks have been done. For bite mark matching, there are 2 methods have been introduced which are MSER blob detection approach and Colour + Intensity thresholding approach. Using Colour + Intensity approach successfully segments the almost all the dentition cast is successfully extracted, except 2 of the unusual convex shape cast.

Then 5 skin detection approaches were experimented including HSV & YCbCr Colour Thresholding, Probability Density Function, 2 versions of Quadratic Discriminant Analysis and K-Means Clustering. The colour thresholding approach is rated to be the best under our scope. The flaws of each algorithm are also included in the corresponding part.

Circle feature based window drawing using Colour Thresholding and Hough Circle with geometric selection has demonstrated the significant success. Potential problems and solutions are also reported and proposed.

SIFT feature detection has been applied on bite mark, but there are only few features which are found. The reason of the extraction failure has explained. It is a bottleneck that the first goal cannot be reached.

Regarding to bruise dimension measurement, rotated minimum bounding box to measure dimension has been proposed and tested. Segmentation methods including Colour Quantisation, Region Growing and Thresholding have been tested. Shading effect has been identified as bottleneck for bruise segmentation. Also, a potential shading effect correction has been proposed.

Finally, future work corresponding to each task has been outlined to assist further development on bite mark matching and bruise segmentation.

14. Reflection

Having come towards the end of this module, I shall come to the reflection on what I have learnt over this project. In this project, I have applied and experimented a lot of fundamental image processing techniques range from feature extraction, recognition and segmentation that mentioned in Computer Vision, Graphic, Scientific Computing modules and Data Processing and Visualisation. Not only image processing technique, some mathematical concepts behind the image processing theories are practically explored. Also, I understand how Software Engineer or Scientist use their knowledge to solve real-life problems, including linking the solution to requirement, improving solution from base scenario. This experience is valuable.

Understanding the failure and turning failure as lesson are important in every development. During this project, I had a period feeling frustrated, because solution didn't go well. I was so caught up to reach the best solution. I sometimes dropped the solution directly without carefully improving it. During the meetings with supervisors, I was always advised to keep the solution and to develop more from the bugs or errors. Then, I found the iterative processes useful. It is normal that solutions do not always perform as good as I think. The best solution does not come out directly. The best solution is always iterated through many versions like extracting the circle feature from the measurement tool. Each iteration solves a single bug, improve the result. When the algorithm or idea does not work as expected, do not throw it. Keeps it and analyses it then improves it to achieve the best one.

Moreover, there are several skills could be improved. Especially, time management. The time spent on skin detection function was too much, it consequently reduced the time spending on core problem which extracts bruises from skins. When I was doing skin detection function, I was pursuing perfect result. However, all my proposed methods were not perfect. I kept trying new methods which they didn't seem improved. I didn't realise the imperfectness of the function is acceptable and fulfils the expectation under the project scope. At that moment, I lost the sense of sticking to schedule. By this experience, I learnt the new way to assess a result. Assessing a result not only takes the performance of accuracy or efficiency as considerations. The expectation of outcome, the resource such as time given are also need to consider. Luckily, the tasks after skin detection including dentition cast extraction, window drawing were efficient enough to balance the majority time loss. Luck is not guaranteed in developments. Understanding how to assess the result if it is sufficient or not is important to keep development smoothly.

Planning too tight schedule should be avoided. In the initial plan, I planned the schedule quite tight, the academic holiday was filled by different further tasks. The ideal holiday plan was supposed to refine and keep missing tasks back to track. Therefore, there were no buffers for any delay or accidents. Unfortunately, I was invited to few technical interviews with Amazon Alexa Team for graduate position. As Amazon interview is well known as tough, I spent total 2 weeks during holiday to manage the interviews. It could have predicted that job interview or any delay and refinement would occur in project development. It is not necessary pushing tasks together and expecting smooth development. Assigning time slot for buffering for any projects is important.

I truly enjoy this project which has helped me to identify my strength and improvement for my future career life.

15. Reference

- [1] Refuge, “Domestic violence – the facts,” Refuge, [Online]. Available: <http://www.refuge.org.uk/get-help-now/what-is-domestic-violence/domestic-violence-the-facts/>. [Accessed 05 May 2017].
- [2] Sam Evans, Suzanne Noorbhai, Zoe Lawson, Seren Stacey-Jones, Romina Carabott, “Contrast Enhancement of Bite Mark Images Using the Grayscale Mixer in ACR in Photoshop®,” *Journal of Forensic Science*, vol. Volume 58, no. 3, p. 804–810, 2013.
- [3] Catherine Adams, Romina Carabott, Sam Evans, Forensic Odontology, UK: Wiley Blackwell, 2014.
- [4] “Bruises,” Forensic Medicine For Medical Student, 2015. [Online]. Available: <http://www.forensicmed.co.uk/wounds/blunt-force-trauma/bruises/>. [Accessed 3 May 2017].
- [5] Claire G. Ross, Carley Chwal, Jeffrey A. Beckstead, Roger W. Byard, Neil E.I. , “Hyperspectral imaging of bruises,” *Pathology*, vol. Volume 46, p. S88, 2014.
- [6] Anita Chaudhary, Sonit Sukhraj Singh, “Lung Cancer Detection on CT Images by Using Image Processing,” in *Computing Sciences (ICCS) 2012 International Conference*, Phagwara, India, 2012 International Conference on.
- [7] BHAVYASHREE K G, SHEELA RAO N, “Determination and Analysis of Arthritis Using,” in *IRF International Conference*, Bengaluru, India, 2014.
- [8] R. Sumithra, Mahamad Suhil, D.S. Guru, “Segmentation and Classification of Skin Lesions for Disease Diagnosis,” *Procedia Computer Science*, vol. Volume 45, pp. 76-85, 2015.
- [9] S. SUZUKI, “Topological Structural Analysis of Digitized Binary Images by Border Following,” *Computer Vision, Graphics, and Image Processing*, vol. Volume 30, no. Issue 1, pp. 32-46, 1985.
- [10] “Morphological Transformations,” OpenCV, 2 May 2017. [Online]. Available: http://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html. [Accessed 2 May 2017].
- [11] A. D. Bimbo, “MSER,” The Media Integration and Communication Center (MICC), 2011. [Online]. Available: <https://docs.google.com/document/d/1A1XkWJzs1H0DpzDE05qXQkCA2tor1mzo5fZC5QxVm40/edit>. [Accessed 03 May 2017].

- [12] "Contour Features," OpenCV, 18 December 2015. [Online]. Available: http://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html. [Accessed 03 May 2017].
- [13] K. B. Shaik, "Comparative Study of Skin Color Detection and Segmentation in HSV and YCbCr Color Space," *Procedia Computer Science*, vol. 57, pp. 41-48, 2015.
- [14] "HSL_and_HSV," Wikipedia , 3 May 2017. [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:Hsl-hsv_models.svg. [Accessed 3 May 2017].
- [15] "YCbCr-CbCr," Wikipedia , 02 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/File:YCbCr-CbCr_Scaled_Y50.png. [Accessed 03 May 2017].
- [16] "Probability Density Function," Wolfram Math Word, 2016. [Online]. Available: <http://mathworld.wolfram.com/ProbabilityDensityFunction.html>. [Accessed 2 May 2017].
- [17] "The Anatomy of Shading," The Virtual Instructor , 2016. [Online]. Available: <http://thevirtualinstructor.com/images/theanatomyofshading.jpg>. [Accessed 2 May 2017].
- [18] "1.2. Linear and Quadratic Discriminant Analysis," Scikik Learn, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/lda_qda.html. [Accessed 02 May 2017].
- [19] "K-Means Clustering in OpenCV," OpenCV 3.0, 10 November 2014. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_kmeans/py_kmeans_opencv/py_kmeans_opencv.html . [Accessed 2017 May 03].
- [20] "Understanding K-Means Clustering," OpenCV, 10 November 2014. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_kmeans/py_kmeans_understanding/py_kmeans_understanding.html. [Accessed 3 May 2017].
- [21] Michael J. Quinlan, Stephan K. Chalup, Richard H. Middleton, "Application of SVMs for Colour Classification," School of Electrical Engineering & Computer Science, The University of Newcastle, Australia.
- [22] Nakhoon, BaekSun-Mi, ParkKu-Jin, KimSeong-Bae Park, "Vehicle Color Classification Based on the Support Vector Machine Method," in *International Conference on Intelligent Computing*, Berlin, 2007.
- [23] N. K. E. abadi, N. Dahir, Z. Alyasseri and A. Alkareem, "An algorithm of skin detection based on texture," in *4th International Congress*, Iraq, 2011 .

- [24] A. A. Mohammadreza Hajiarbabi, "Human Skin Detection in Color Images Using Deep Learning," *International Journal of Computer Vision and Image Processing*, p. 13, 2015.
- [25] S. Price, "Edges: The Canny Edge Detector," Institute for Computer Based Learning, 4 July 1996. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/canny.htm. [Accessed 03 May 2017].
- [26] Ali Ajdari Rad, Karim Faez, Navid Qaragozlou, "Fast Circle Detection Using Gradient Pair Vectors," in *Proceedings of the VIIth Biennial Australian Pattern Recognition Society Conference*, Sydney, 2013.
- [27] R. Rao, "Features and Image Matching," 2009. [Online]. Available: <https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf>. [Accessed 2017 05 03].
- [28] Alexander Mordvintsev, Abid K. Revision, "Feature Homography," OpenCV, 2013. [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html. [Accessed 2017 May 05].
- [29] D. Geier, "Computing oriented minimum bounding boxes in 2D," THE INFINITE LOOP, 23 January 2014. [Online]. Available: <https://geidav.wordpress.com/tag/rotating-calipers/>. [Accessed 3 May 2017].
- [30] P. D. Marshall, "CM2208 Fourier Transforms 01 Theory," [Online]. Available: http://users.cs.cf.ac.uk/Dave.Marshall/CM2208/LECTURES/CM2208_Fourier_Transforms_01_Theory.pdf. [Accessed 05 May 2017].
- [31] R. CLOUARD, "TUTORIAL: ILLUMINATION CORRECTION," GREYC, 08 July 2011. [Online]. Available: <https://clouard.users.greyc.fr/Pantheon/experiments/illumination-correction/index-en.html>. [Accessed 2017 May 05].
- [32] Glenn A. Woodell, Daniel J. Jobson, "Retinex Image Enhancement: Application to Medical Images," NASA Langley Research Center, Greenbelt, 2001.