Lab 2

Hanz Cuevas Velásquez, Bob Fisher
Advanced Vision
School of Informatics, University of Edinburgh

Week 3, 2018

This lab will focus on learning simple image transformations and the Canny edge detector. The lab will also go through a simple image segmentation technique, thresholding.

Before anything else, we need to go inside the folder where we downloaded the lab material using the address bar of Matlab or using the terminal command `cd` in the Matlab command window. Then, we will read the image `'hand.png'` and stored in a variable called `rgb_hand`. We will transform the variable to double, transform it to the HSV colour space and store the second dimension (Saturation) in another variable:

```
rgb_hand = imread('Images/hand.png');
rgb_hand = im2double(rgb_hand);
hsv_hand = rgb2hsv(rgb_hand);
s = hsv_hand(:,:,2);
%Now we show the rgb_hand and the s colour space
subplot(1,2,1), imshow(rgb_hand)
subplot(1,2,2), imshow(s)
```

The colour space `s` will be useful for the next step of our lab.

1 Edge detection: Canny

Detecting the edges of an object is an important tool in many computer vision applications and it is used to find the boundaries of the different objects in an image. There is a variety of edge detectors which we encourage the student to learn and practice [1]. However, in this lab we will only learn to use the Canny edge detector. In Matlab, we can use the Canny detector in a gray scale image (2D intensity image) using the following command `BW = edge(I,'Canny',threshold,sigma)`. This command will return a binary image which contains the edges of the objects in the image. `I` is the image from which we want to obtain the edges, instead of `'Canny'` we can specify another edge detector, however, for this exercise we will stick with it. These two values are enough to use the edge detector, however, we can also change the threshold of the detector which should be between 0 and 1; this will return all the edges that are stronger than threshold. Finally, we can also change the standard deviation (sigma) of the Gaussian filter.

For our lab, we will observe how changing the threshold of the detector will affect the edges of the output. We will use 5 different thresholds: 0.1, 0.15, 0.2, 0.25, and 0.3. Now, you should write the code of the Canny detector using these thresholds and show them. You should use the color space `s` as input and the function `figure('Name', threshold #)` to show the thresholds in different windows, where `threshold #` should be replaced by the 5 thresholds we chose, as seen in figure 1.

[1] https://uk.mathworks.com/discovery/edge-detection.html

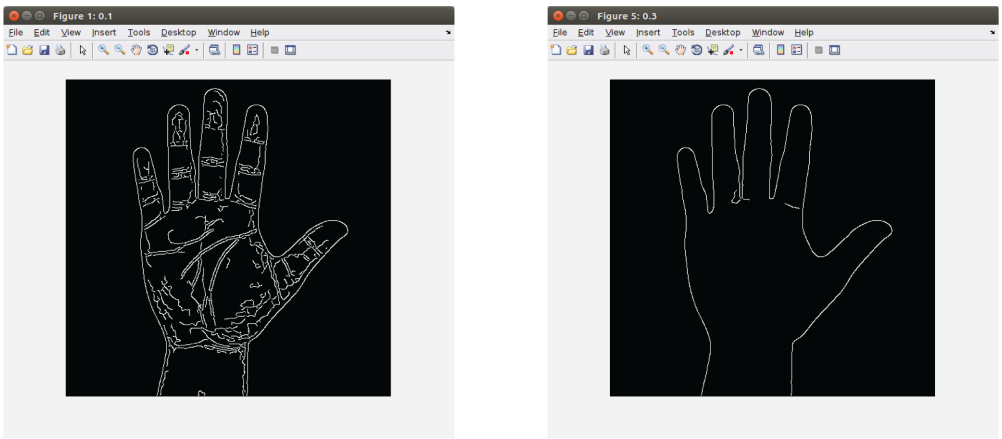

Figure 1: Canny edge detector a) threshold: 0.1, b) threshold: 0.3.

2 Morphological operations

In this section we will learn how to use two morphological operations: erosion and dilation. Dilation is used to gradually enlarge the boundaries of regions of foreground pixels (white pixels); erosion does the opposite[2]. In matlab these two operations have two inputs, a binary image, and a structuring element object (or an array of structuring element objects). This structuring element can have different shapes, like a line, square, or disk and it represents the number of pixels that will be expanded or shrunk. To understand it better, we will do one experiment taken from the Matlab documentation[3]:

First, we will create a binary image:

```
close all;
BW = zeros(9,10);
BW(4:6,4:7) = 1
```

$$BW = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This will give us a square of ones of 3×4. Now, we will create a square structuring element of size 3×3. Finally we will use `imdilate` to dilate the image using the created structuring element.

```
SE = strel('square',3);
BW2 = imdilate(BW,SE)
```

[2]More information about morphological operations in the webpage: https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm
[3]https://uk.mathworks.com/help/images/dilate-an-image.html

$$BW2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We can observe that our square of ones has increased its size to 5×6. This shows that the center pixel of the structuring element is placed on the pixel of the image that is being processed and the rest of the pixels of the structuring element are used in the morphological computation. The student should do the same experiment using `strel('diamond',2)` and analize the output.

We can use the same principle in our image of the hand. First, we will use the Canny edge detector to obtain the edges of the hand, then, we will use dilation and erosion and observe the changes.

```
ed_canny = edge(s, 'Canny', 0.2);

%Structuring element
SE = strel('square', 3);
SE2 = strel('square', 8);

%Morphological operation
erosion = imerode(ed_canny, SE);
dilation = imdilate(ed_canny, SE2);
subplot(1,3,1), imshow(ed_canny)
subplot(1,3,2), imshow(erosion)
subplot(1,3,3), imshow(dilation)
```

You will see a window similar to figure 2

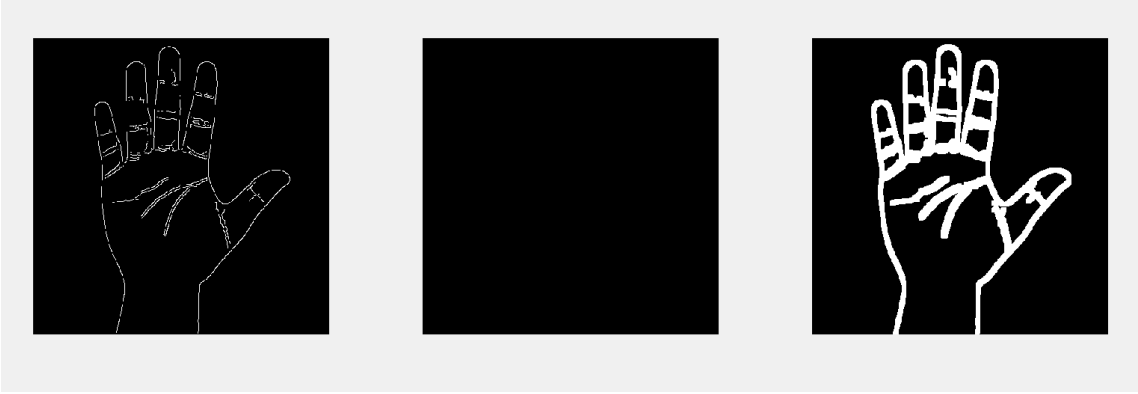

Figure 2: From left to right, Canny edge detector binary image, binary image after erosion, and binary image after dilation.

We can observe that, because the lines of the edges are too thin, applying erosion to the image makes the edges disappear. Unlike erosion, dilation makes the edges thick.

3 Colour thresholding

Colour thresholding is a basic operation that allows us to use the intensity values of our image to create binary images or masks. We will use colour thresholding in the following task. Our goal is to segment the five fingers of the hand. As a first step, we will use `imfill(Img, 'holes')` to fill

the holes inside the boundaries created by our dilated version of the Canny detector, where `Img` is a binary image. Then, we will use erosion to reduce the thickness of the fingers. The student should write both functions in the code below:

```
filled_fingers = %(Code: Fill the wholes inside the dilation image)
SE = strel('square', 15);
erode_fingers = %(Code: Use erotion on the variable filled_fingers) We will
    consider this as our mask

%We put the mask on the rgb image using the bsxfun command
masked_hand = bsxfun(@times, rgb_hand, cast(erode_fingers, 'like', rgb_hand));
subplot(1,2,1), imshow(erode_fingers);
subplot(1,2,2), imshow(masked_hand);
```

You will obtain a similar window to figure 3.

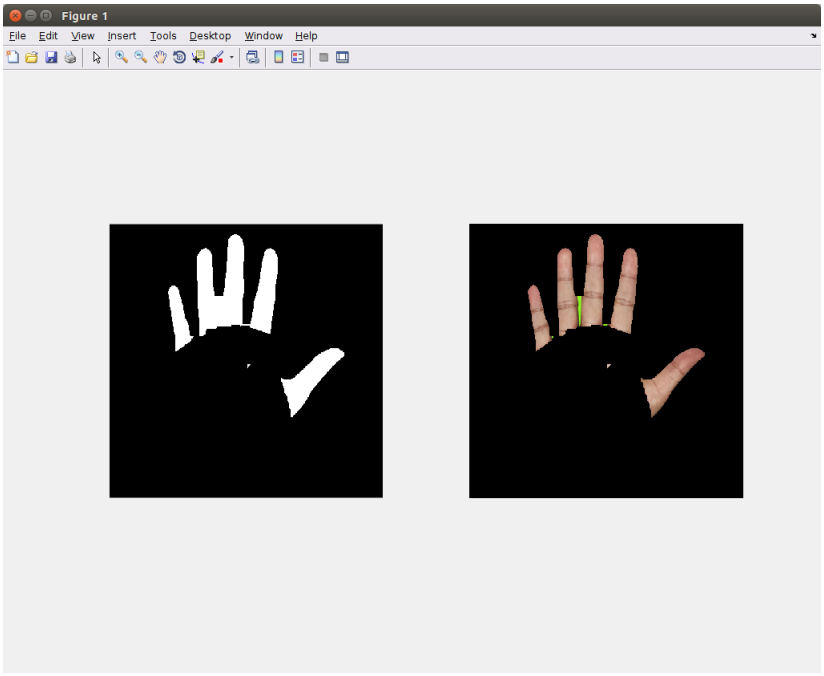

Figure 3: On the left, mask created after the morphological operations. On the right colour image with the mask.

In the figure, we can observe that our mask has segmented most of the fingers correctly. However, there is a green region from the background that was also segmented. If we click on the green region using the data cursor tool and then on any part of the fingers, we will see that the intensity of the pixels in the green colour space (G) is higher in the green region than in the fingers. We can use this value to get rid of the green region. Your task is to find a sensible threshold in the green colour space to eliminate the green region without affecting the colour of the fingers. The chosen threshold should be able to separate the fingers similar to figure 4.

```
%Given that the green background has high intensity values in the green space, we
    will threshold that space.
G = rgb_hand(:,:,2);
imshow(G); %Show the green space.
mask_G = G.*erode_fingers; %create a mask based on the green colour space and the
    previous mask.
imshow(mask_G)%Show the binary mask.

%(Code: Use the image mask_G to find the value of the threshold. Once you find it,
    replace the comment in "tr" with it.)
tr = %Threshold;
%We threshold the intensity value of our new mask. All values above "tr" will
    become 0.
```

```
mask_G(mask_G>tr)=0;
mask_G(mask_G>0)=1; %we make our mask a binary image

%check if your threshold separated the fingers successfully
new_masked_hand = bsxfun(@times, rgb_hand, cast(mask_G, 'like', rgb_hand));
subplot(1,2,1), imshow(masked_hand)
subplot(1,2,2), imshow(new_masked_hand)
```

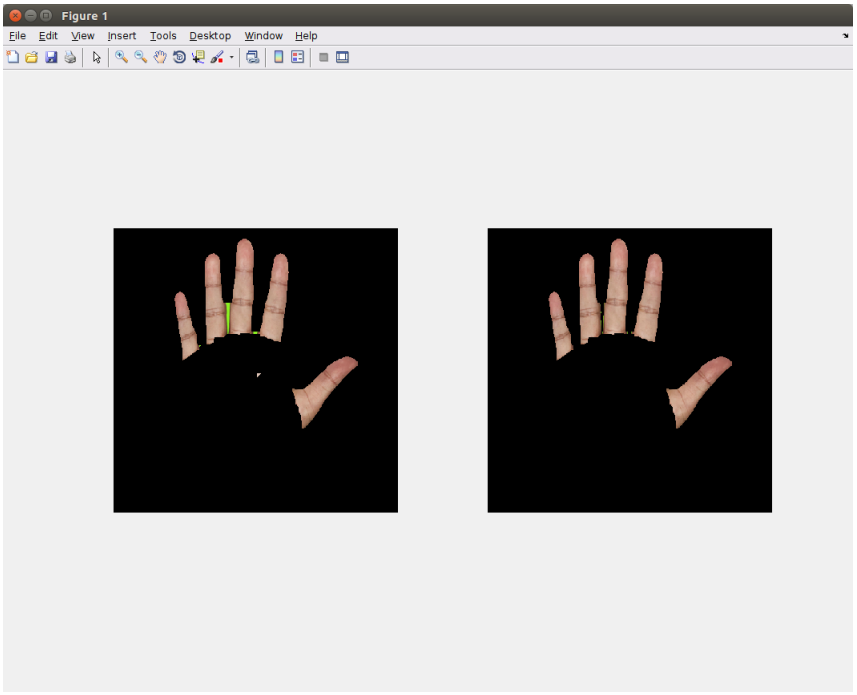

Figure 4: On the left, the colour image with the first mask. On the right, the colour image after the colour thresholding.

4 Labeling and bounding box

We are almost done with our objective, our next step is to create a bounding box around our five fingers. Before doing that, we need to label them. In Matlab, we use the function `bwlabel(BW)` to label all the separated objects in a binary image, this function will return an image where the values of the pixels belonging to a connected region have the same number. You should write the code to label the fingers, show the output of the function `bwlabel` and show each finger separately using the code below.

```
%First, if there is a small area near the palm of the hand, we can eliminate it
    using bwareopen.
mask_G = bwareaopen(mask_G, 50); % Only the objects with area (pixels) >50 will
    remain.

lbl = %(Code: label the fingers using mask_G)
%(Code: Show the output of the bwlabel function and observe the values of the
    fingers)
close all;
%(Code: Show each finger separately. HINT: You can use (lbl==i) which will output a
     binary image were the pixels with value i will become 1 and the others 0)
```

To draw a bounding box we will use the following function `RGB = insertObjectAnnotation(I, shape, position, label)`, where `I` is the image where we want to draw the bounding box, `shape` is to select the type of bounding box, rectangle or circle. `position` is the location and size of the bounding box, and `label` is the text that will show. We will use a rectangle as a shape, so we need

to provide four values, x, y, width, and height, to the function. The elements, x and y, indicate the upper-left corner of the rectangle, and the width and height specify the size.

Before going any further, we will make a small comment on how are the pixel indices in Matlab. First of all, we have to know that each image is divided in rows (y) and columns (x). They are used to access to the values of each pixel in the image, as observed in 5. Unlike other programming languages like Python, Matlab uses the number 1 as the first value of its arrays and matrices, so, if we want to read the value of a pixel that is in the 2nd row and 3rd column, we will write `Image(2, 3)`.

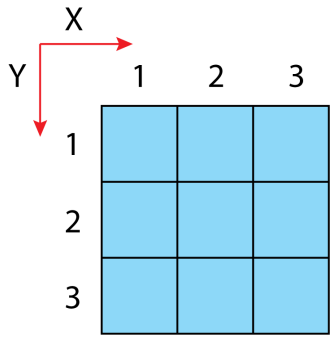

Figure 5: Pixel indices in Matlab using the Cartesian coordinates.

To obtain the dimensions of our bounding boxes, we need to obtain the minimum and the maximum row and column indices where there is a pixel with value higher than 0. We do that with the following code:

```
close all;
%We will concatenate the positions of our 5 fingers, so we initialize the position
    as empty.
position=[];
%Names for our fingers.
label_str ={['Pinky'],['Ring Finger'],['Middle Finger'],['Index Finger'],['Thumb'
    ]};

%Go through all the 5 labels to find the maximum and minimum row (r) and column (c)
for i=1:5
    [r,c]=find(lbl==i);
    max_r=max(r);
    min_r=min(r);
    max_c=max(c);
    min_c=min(c);
    width = max_c-min_c+1; %obtaining the width of the bounding box
    height = max_r-min_r+1; %obtaining the height of the bounding box
    position = [position; [min_c, min_r, width, height]]; %concatenating the
        positions of the fingers
end
rgb=insertObjectAnnotation(rgb_hand, 'rectangle', position, label_str, 'Color', '
    red', 'FontSize', 10);
    imshow(rgb);
```

You should have the following image as a result, see figure 6.

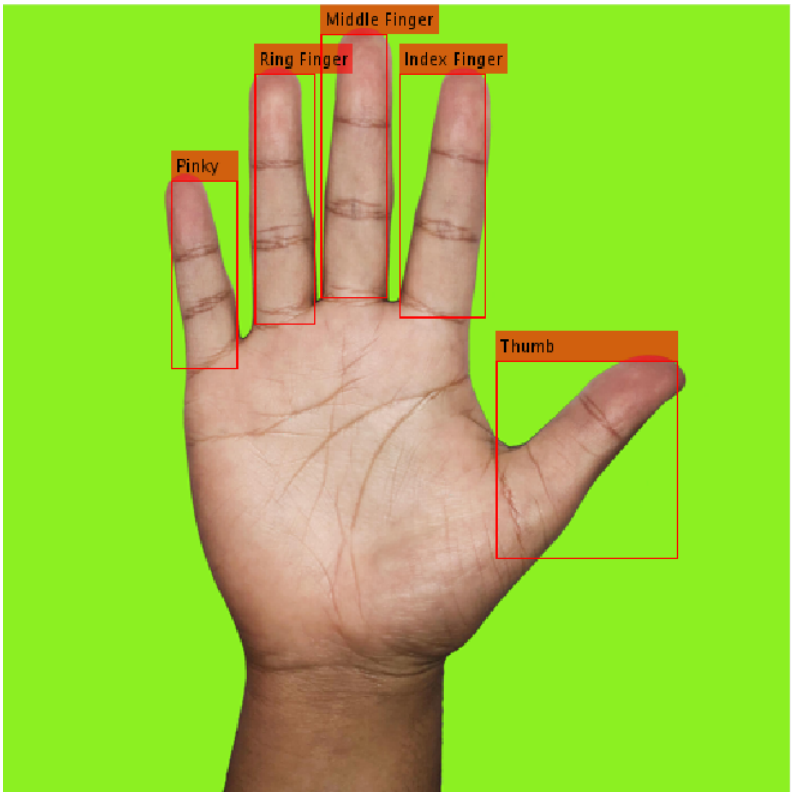

Figure 6: RGB image with bounding boxes in each finger.