Lab 7:
Machine Vision and Image Processing via MATLAB

1 Objectives:

- To understand digital image acquisition.
- To understand basic image manipulation in MATLAB
- To use MATLAB's `bwlabel` function to detect coins in images that you acquire in the lab.

2 Theoretical Overview:

A *machine vision system* is an automated system that makes decisions based on the analysis of digital images. The applications of machine vision systems include routine, automatic visual inspection tasks such as counting objects on a conveyor and reading bar codes, and also embraces more sophisticated analysis of visual images for robotic systems, environmental monitoring, geographical mapping and a variety of other tasks.

In this lab, you will create a simple machine vision system by acquiring images and using MATLAB to extract information from them. Throughout the course you have gained working knowledge of the various components of an electronic system, beginning with the electronic device building blocks, the microprocessor controller/brain and most recently, moving BOE-BOTs. This lab will focus on programming the 'brain' that will take in, analyze and categorize visual data. MATLAB is an important software tool that will allow you to easily analyze your image data.

Your goal is to identify and count coins that are viewed by the camera. To accomplish your goal, you will use some common machine vision techniques, including **background subtraction** and **binary image processing**.

3 Procedure:

Please follow the steps below. Lettered points contain questions/tasks that need to be addressed in your lab write-up.

3.1 ACQUIRING IMAGES

To process your images, you will need to ”capture” an image of both the *background* as well as *the image + background*. This will allow you to subtract the background from the image + background. Acquire images using the Fire-i camera as follows:

- Start up the Fire-i application and change these settings under *Format Selection*: (note, do NOT place a tick mark in the box next to ”format selection”, as this will cause the camera to take literally hundreds of photos until you uncheck it.)

Pixel Format	RGB 24-bit
Resolution	640 × 480
Frame Rate	15.0

- Turn on the real-time preview by activating the *Preview* check box and double-clicking the camera in the right window. Adjust the camera so that it points downward toward the desk. (It is important that it points directly downward not at an angle and that it is very secure.) You may need to manually adjust the focus by rotating the lens to get a clear image.
- Place some darkly colored (ideally black) paper on the desk to completely fill the camera’s field of view, and tape it down so it does not move.

• 3.1.1 Background Capture

Click on *Frame Capture* in the *Stream Control* Make sure the captured frames are being saved onto the desktop, this can be done by clicking “Frame Capture” (under Stream Control). Click the lintextked words, not the checkbox itself. If the checkbox is checked, uncheck it. Change the file name to `background`. (The program will append the extension `.BMP` when it saves your image files.) Click *OK*.

• 3.1.2 Coin Image Capture

Now you’re ready to take pictures! Making sure that you **do not cast any shadows** on the paper, click on the camera icon in the tool bar (the 4th button from the left) to capture the image. If the camera icon is greyed out, make sure you click on the image of the camera first. Also make sure that the yellow play button (on the top next to the pause and stop buttons) is clicked.

- Scatter some coins (nickels, dimes and quarters but no pennies!) on the paper, being careful to not let them touch one another. Change your settings so that the image filename is `coins1`. Capture another image, and remember not to cast any shadows.
- Move the coins and repeat to get images `coins2`, `coins3` and at least two more (for a total of at least 5.)

3.2 PROCESSING IMAGES WITH MATLAB

3.2.1 Starting up MATLAB

Once you are satisfied with your images, move your image files into the MATLAB folder in "My Documents".

Start up MATLAB on your computer. The program will open a "Command Window." This allows you to type commands into MATLAB at the prompt. Each command will be executed after you press *Enter*.

3.2.2 Forming the Binary Image

There are two main steps that are needed to detect the coins in your images. These are: 1) computing the difference between a coin image and the background image (a process known as *background subtraction*), and 2) thresholding the difference. The desired result is a *binary image* with pixel values of `1` indicating the locations of the coins.

Open a new MATLAB script (click File > New > Script)[1], and save it with the name `detectcoins` in the same directory as your coin images (that is, the MATLAB folder with My Documents).[2] Edit your script so that it performs the following operations:

- Loads the image `background.BMP`, converts it from RGB to grayscale, and stores it in the variable `imbg`. (Hint: use `imread` and `rgb2gray`.) Remember that semicolons can help suppress output so that your MATLAB prompt doesn't get too messy!
- Loads the image `coins1.BMP`, converts it from RGB to grayscale, and stores it in the variable `imcoins1`.
- Creates a new figure with two axes (hint: use `subplot(1,2,1);`), and displays the images with `imbg` on the left and `imcoins1` on the right.

(a) To run your program, go to Debug > Save and Run, or press F5. Print `detectcoins.m` and the resulting figure, and include in your write-up.

3.2.3 Background Subtraction

Background subtraction is a simple way to detect the changes that occur during the time between capturing two images. (Of course, this assumes that the camera has not moved and that the lighting has not changed!) Here, we are only interested in the absolute value of the difference and this is computed in MATLAB using the command

```
>> imdiff = abs(imcoins1 - imbg);
```

[1] An `.m` file is simply a way of encapsulating everything we would write at the interactive MATLAB prompt in a single, reusable file.

[2] Naming the .m file this is important for reasons that will become apparent in the *Extensions* section.

(Here the absolute difference is stored in the variable `imdiff`.) Add the command for computing `imdiff` to the end of your `detectcoins` script and re-save it.

Note the presence of the semicolon at the end of this command. The semicolon indicates that the output of this operation will be suppressed (i.e. not printed out) as it is both long and uninteresting. For debugging purposes, however, it might make sense to temporarily remove some semicolons in order to see the output of your MATLAB commands. Although omitting semicolons will raise warnings from MATLAB, it will not cause an actual error at runtime.

3.2.4 Thresholding

Applying a *threshold* to the the difference image will produce the final binary image. Use the command

```
>> imdiffb = imdiff > ##;
```

where `##` is an integer of your choice between 0 and 255. Display `imdiffb` using `imshow`. Repeat this process to find a threshold that best separates the coins from the background. Add the command for computing `imdiffb` to the end of your `detectcoins` script and re-save it.

(a) Which threshold values work well? Note: there are several correct answers to this question.

(b) Print the resulting binary image to include in your write-up.

3.2.5 Eliminating Noise

In the previous question, even the best threshold value gives noisy results in the sense of having some background pixels labeled incorrectly as `1` and some coin pixels labeled incorrectly as `0`. This 'noise' in your binary image can be removed using *morphological filtering* as discussed in class. In MATLAB, this is implemented using the commands `strel`, `imdilate` and `imerode`.

The command `strel` creates a *structuring element* to be used for morphing. For example, the command

```
>> se=strel('disk',2,0);
```

creates a structuring element called `se` that approximates a disk with a radius of two pixels.[3] (Type the command without the semicolon to see what `se` looks like.)

(a) Draw the structuring elements that approximate disks of radius 3 and 4.

[3]The 0 refers to an approximation of a disk. Values of 4, 6, or 8 give increasingly bad approximations of a disk

(b) Which of these two statements best describes how MATLAB approximates a disk of radius `R`?

(i) The structuring element members comprise all pixels whose centers are no greater than `R` away from the origin.

(ii) The structuring element members comprise all pixels whose centers are no less than `R` away from the origin.

3.2.6 Dilation and Erosion

The *DILATION* AND *EROSION* operations are performed using `imdilate` and `imerode`, respectively. Try typing the commands

```
>> seo=strel('disk',3,0);
>> imer=imerode(imdiffb,seo);
>> imdi=imdilate(imdiffb,seo);
```

and use `imshow` to display the resulting images `imer` and `imdi`. Is this what you expected?

As discussed in class, you can use these two operations sequentially to fill 'holes' in the detected coins and to clean up noise in the background. These operations are referred to *CLOSING* and *OPENING*. The following code

```
>> imclean1=imerode(imdiffb,seo);
>> imclean=imdilate(imclean1,seo);
```

will "clean" your binary image in this way, by performing erosion followed by dilation (aka "opening").

(a) Try changing the radius of the disk in the structuring element `se`, running the opening operation on `imdiffb` again, and displaying the result. What is the smallest radius that successfully removes all of the false positives? Append the commands for computing `seo` and `imclean` to the end of your script `detectcoins` and re-save it.

(b) In your lab report, write down a one-line command that performs closing on `imclean` using the structuring element `sec`, and storing the result in variable `imclean2`. Use the `imclose` command for this; it gives results that are equivalent to dilation followed by erosion.

(c) Similar to (a), find the smallest radius of `sec` that successfully fills all of the holes in the detected coins. What is this radius? Is it the same value found in (a)? Why or why not? Call the final cleaned image that you want to display as `imnoholes`. Append the commands for re-computing `sec` (with the correct radius) and computing `imnoholes` to the end of your script `detectcoins`. Re- save it.[4]

[4] If your image is degraded by these transformations, try smaller values. If that does not work, swap the order of the opening and closing operations and try again.

3.2.7 Displaying Your Results

At this point, you almost have a complete script for detecting coins in an image. The last step is to add a command to display the results. For convenience, we will display the results in the same figure used to display `imbg` and `imcoin1`.

Edit your script as follows. First, change the lines that read `subplot(1,2,#)` to `subplot(1,3,#)` so that the created figure contains three axes instead of two. Next, add the following two lines to the end of your script:[5]

```
subplot(1,3,3);
imshow(~imnoholes); title
'Detected Coins';
```

and re-save it.

(a) Run your script by typing `detectcoins` in the command window and pressing *Enter*. If the result is not what you expect, correct any bugs in your script (perhaps adjusting the radii in the `se` commands of the threshold in the `imdiffb` command.) Once your script is successful, print it out, and print the resulting figure.

3.2.8 Analyzing Your Other Coin Images

So far you have done your testing on a single image, `coins1.BMP`. You can try your other coins images simply by editing the line defining `imcoin1` in your script. Edit this line to read image `coins2.BMP`, re-save your script, and run it from the command window. Do you get the expect result? If not, try adjusting the threshold and radius values.

(a) By editing and re-running your script, create and print successful output figures for the images `coins2.BMP` and `coins3.BMP`.

3.3 CLASSIFYING IMAGES

Now that you can reliably detect the presence of coins, the next step is to *classify* them as nickels, dimes and quarters. How can we tell them apart? One way is to *compute their areas* (i.e., the number of pixels they contain.) Thus, the first step toward your goal is to write commands that will compute the area of each coin.

The output of your script is a binary image `imdiffb` that provides the location of the coins. In order to distinguish the pixels belonging to one coin from pixels

[5]The tilde ~ in the second line performs a logical NOT operation, so that the coins are displayed as black circles on a white background instead of white pixels on a black background. This saves ink!

belonging to another, you will find the *connected components* using the MATLAB command `bwlabel`. Try typing the following sequence of commands

```
>> [imcc, n] = bwlabel(imnoholes); % connected components
>> figure;                         % new figure
>> imshow(imcc,[]);
>> impixelinfo                     % enable pixval (values of pixel) display
```

This computes the connected components and displays them in a new figure. The second argument in the `imshow` command simply scales the brightness of the image. The `impixelinfo` command adds an additional display to the bottom left corner of your figure. When `impixelinfo` is activated, it displays the coordinates and the intensity values of the pixel in your image according to the position of your mouse. As you move your mouse over the different coins in your image, you should see that each coin has been given a diff t integer value, from 1 to `n`, where `n` is the number of connected components (and is the second output returned by the `bwlabel` command.)

Append the line

```
[imcc, n] = bwlabel(imnoholes);
```

to your `detectcoins` script and re-save it.

3.3.1 Computing Areas

Now, computing the area of the i^{th} coin is as simple as computing the number of pixels with the value i. To do this, you can use the == operator (yes, there are two of them), which evaluates to 1 when two items are equal and evaluates to zero otherwise. To get a sense of how this operator works, try typing:

```
>> 34 == 23
>> 34 == 34
>> [23 34 52] == 23
```

Now, you can use this operator to compute the area of the first coin using:

```
>> area1 = sum(imcc(:) == 1);
```

The `sum` command computes the sum of the elements in each column of an array. The `(:)` after `imcc` reshapes it from a $H \times W$ array to an $(HW) \times 1$ array. This is necessary for the sum to be computed over all pixels in the image as opposed to being computed separately for each column.

(a) By typing commands into the command window you can compute the areas of each of the coins in the image `coins1.BMP`. What are these areas?

3.3.2 Streamlined calculation of areas: looping

Instead of having to repeatedly type similar commands, the process of computing the area of each coin can be conveniently accomplished using a *for loop*. Below is an example of a *for loop* in MATLAB:

```
>> upperlim=5;
>> for i=1:upperlim i*2
end;
```

Enter this in the command window. (Note how the prompt >> disappears until the *for loop* is complete.) Once you press *Enter* after the final `end` you will see what happens: the product `i*2` is computed 6 times, once using each value of `i` between `1` and the `5` (the value of `upperlim`), inclusively.

Now you will edit your `detectcoins` script to use these ideas to compute coin areas. Append the following lines to the end of your script

```
coinareas=zeros(n,1);       % allocate array for areas for
i=1:n                       % repeat for each coin
  coinareas(i)=sum(****); % area of ith coin end
coinareas                   % show areas in command window
```

and replace `****` with the appropriate command to compute the area of the i^{th} coin. The first line you added simply creates a new array that is large enough to store the `n` areas and sets all of the elements of this array to an initial value of `0`. The last line, displays the areas in the command window.

Re-save your script, and run it using the image `coin1.BMP`. Does it output the correct area values?

(a) Run your script using the images `coins2.BMP` and `coins3.BMP`. In your lab report, write the areas of the coins for each image.

(b) From these values you should be able to estimate the average areas for nickels, dimes, and quarters. What is the average area for each type of coin?

4 Extension:

Completing this section is worth +1.0 Extra Credit points.

4.1 Creating a MATLAB Function

- In this lab you wrote a script, which is a list of commands to be executed in sequence. While this was useful, it meant that each time you wanted to make some changes (a different coins image, a different `strel` radius, etc.) you had to edit the file, re-save it, and then run it from the command window.

This process can be improved by making your script into a *function*. Like a script, a function is a list of MATLAB commands, but it is different because the first line in a function is special.[6] Here is an example of a simple function that has only two lines of text and adds three numbers and returns the result.

```
function S=myadder(X,Y,Z) S=X+Y+Z;
```

This function is created by typing these two lines into a new file and saving that file using the name `myadder.m`. Then, you would be able to run your new function in the command window:

```
>> myadder(3,8,2);

ans =

    13
```

The first line in a MATLAB function has the format

`function` *output = filename* `(` *input1* `,` *input2* `,` ... `)`

where the outputs and inputs are the names of variables used to perform the commands in the fi

- Make your `detectcoins` script into a function by inserting this line at the head of the file:

```
function coinareas = detectcoins(coinfilename)
```

Also, edit the command `imcoin = imread('coins1.BMP');` so that it reads `imcoin = imread(coinfilename);`, and remove the final command displaying `areas`. Re-save your file and run it in the command window using

```
>> detectcoins('coins1.BMP')
```

1. What is the output of the function?

- Edit the first line of your function to read

```
function coinareas=detectcoins(coinfilename,threshold,radius1,radius2)
```

and edit the corresponding lines in your file to make use of these inputs for the binary threshold (used in the definition of `imdiffb`), and the radii of the two structuring elements `seo` and `sec`.

[6]For those who are familiar with other programming languages, one of MATLAB's idiosyncrasies is that each `.m` script file can contain only one function

4.2 Counting loose change.

So far, your function returns the area of each coin in the image. Can you make it go one step further and add up the monetary value of the coins? To get you started, here is an excerpt that sums up only nickels and dimes

```
function howmuch=detectcoins(imname)

...

howmuch=0;                    % monetary value of coins
coinareas=zeros(n,1);         % allocate array for areas firstupperbound
= 500;                        % approx # of pixels in your smallest coin
seconduppperbound = 800;      % approx # of pixels in 2nd smallest coin
for i=1:n                     % repeat for each coin
  coinareas(i)=sum( **** ); % area of ith coin

  if coinareas(i) < firstupperbound
    howmuch = howmuch + 10; % its a dime
  elseif (coinareas(i) >= firstupperbound) & (coinareas(i) < seconduppperbound)
    howmuch = howmuch + 5;   % its a nickel
  end

end
```

Note that firstupperbound and seconduppperbound are ”thresholding” values that identify the coin types by their area. To make this work, you will need to change these numbers to reflect the numbers you discovered in the lab. Hint: we picked these numbers because they are almost certainly too small.[7] ;)

(a) Edit your function so that it sums the monetary value of the nickels, dimes and quarters in an image and returns the result. Demonstrate your working function to the TA, and print and submit your `.m` file.

[7]No it’s not an error that we have `&` instead of `&&`. Both are logical AND in MATLAB. The double ampersand is simply a short-circuiting AND (i.e. if the first argument is false, the second argument will not be evaluated)