1. Introduction to Computer Vision

1.1 Goals

The goals of this exercise are:

- getting started with Matlab
- getting familiar with the basic image manipulation functions
- implementing some simple real-world Computer Vision algorithms

1.2 Instructions

- To get started, first **download** the corresponding images and code from Moodle.
- Make sure you have the **Matlab Image Processing toolbox** installed.

1.3 Matlab tutorial

Go through Matlab tutorial available at http://cs.brown.edu/courses/cs143/docs/matlab-tutorial/.
If you are already familiar with Matlab and matrix operations, you can go directly to Section 8 of the tutorial ("Working with gray level images"). For those of you beginning with Matlab, we recommend that you spend enough time to make sure you understand all the concepts introduced in the tutorial. Try to copy-paste the code and run it step by step.

1.4 Background subtraction

Background subtraction is an important preprocessing step of many algorithms, e.g. object detection. In the following exercises we will try to subtract the scene background using multiple images.

Figure 1.1: Input “street” images

Exercise 1.1 Extracting a moving object.

- Load the "street1.gif" and the "street2.gif" images using `imread()` function.
- Transform the 8-bit images into double images. Don't forget to check variable types and cast them if necessary when manipulating images in Matlab. Why is it important?
- Subtract the second image from the first one using basic matrix arithmetic operations. What is the result? Why? Save this result for future work.
- Use the `imsubtract()` function of Matlab to check your result.

■

Exercise 1.2 For this exercise, you are given a sequence of images that contains pedestrians we wish to segment with a background subtraction algorithm.

- Uncompress the sequence1.zip file
- Load and create a stack of images. Build a "background model" by averaging out the set of given images seperately for each color channel. Detect pedestrians by subtracting the background model from the original images and applying the right threshold. *Hint:* use `dir()` function to get the list of files.
- Create a more sophisticated background model, where each pixel can be modeled with a Gaussian distribution. We can classify a pixel as background if its current intensity (I_t) lies within some confidence interval of its distribution's mean (μ_t):

$$\frac{|(I_t - \mu_t)|}{\sigma_t} > t \rightarrow Foreground$$
$$\frac{|(I_t - \mu_t)|}{\sigma_t} < t \rightarrow Background$$

  where t is some threshold value and σ_t is the standard deviation of the pixel.
- What difference do you notice between the two approaches? How does changing the threshold affect them?

■

1.5 Image Segmentation

In many vision applications, it is useful to separate out the regions of the image corresponding to objects in which we are interested in the regions of the image that correspond to the background. Thresholding often provides an easy and convenient way to perform this segmentation on the basis of the different intensities or colours in the foreground and background regions of an image.

The input to a thresholding operation is typically a grayscale or colour image. In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or vice versa). Multiple thresholds can be specified, so that a band of intensity values can be set to white while everything else is set to black.

If it is possible to separate out the foreground of an image on the basis of pixel intensity, then the intensity of pixels within foreground objects must be distinctly different from the intensity of pixels within the background. In this case, we expect to see a distinct

peak in the histogram corresponding to foreground objects such that thresholds can be chosen to isolate this peak accordingly. If such a peak does not exist, then it is unlikely that simple thresholding will produce a good segmentation.

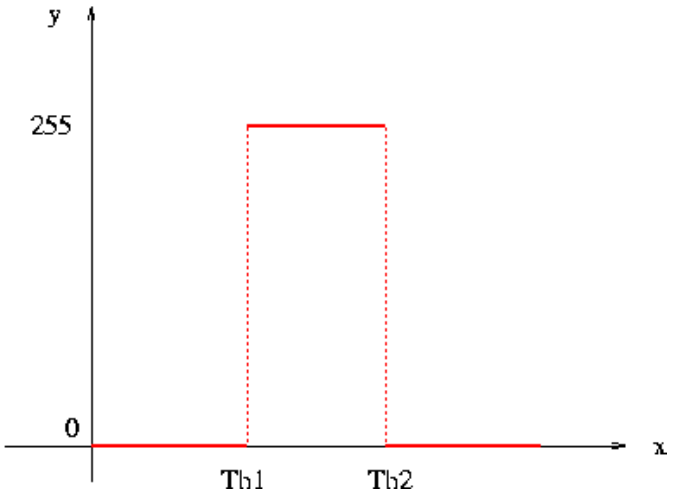

Figure 1.2: An example of thresholoding the image histogram.

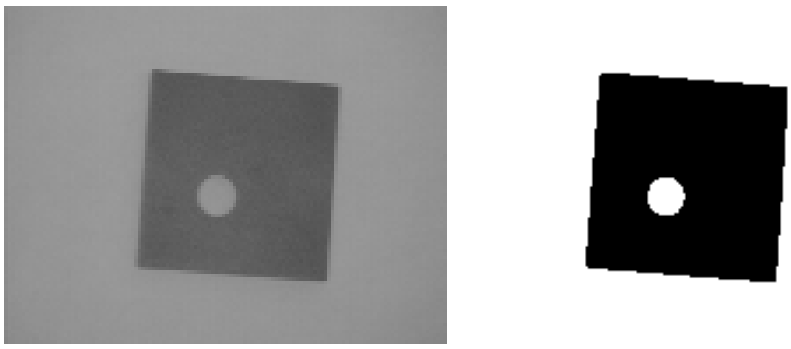

Figure 1.3: Input image (left), segmented image (right)

Exercise 1.3 Image segmentation

- Load the image "wdg.png"
- Write a method to threshold a gray scale image by using two threshold values as shown above. The values must satisfy the following conditions: Th1<Th2, Th1>0, Th2<max.
- Take a look at the histogram using function `imhist()` and choose the best threshold values and segment the image.
- Repeat the same steps for images "brain.gif" and "shading.gif". What do you notice? What are the drawbacks of this segmentation method?

■

Segmentation can be also done for colour images. Here our goal is to count the number of apples in the picture below.

Figure 1.4: An original input image.

Figure 1.5: Apple tree image decomposed into three channels.

Exercise 1.4 Color image segmentation

- Download the "Apples.zip" package, containing the image and the `CountConnectedComponents()` function.
- Read the 3-channel image and visualize the 3 channels (red, green, blue) separately, as it is shown in the figure below.
- Try to obtain a binary image such that binary image == 1 for pixels representing apples and 0 otherwise using the function implemented in the previous question. Which channel(s) would you use for that?
- Count the number of connected components in your binary image (here corresponding to apples). For this, you can use the provided function

```
[imageColor, nConnectedComponents] = ...
   CountConnectedComponents(binaryImage, ...
   elementsThreshold)
```

- Discuss with your neighbors and answer the following questions:
  - Could your algorithm count the correct number of apples?
  - What are the main limitations of this approach?
  - How could you improve your method?
  - Why is it so easy for a human to perform this task?

■