Introduction to Vision and Robotics
Vision Practical: Coin Counter

Dylan Angus, Matthew Martin

October 20, 2016

1 Introduction

The purpose of this practical is to develop a program in Matlab that recognises and classifies several objects in an image. These objects can be coins or other small items, and the program must segment the image, identify each of the objects, and output the total value (in pounds and pence) of the objects in the image.

All of the images are taken from a downward facing camera viewing a scene containing the objects on a static background. We were provided with a set of 14 sample images on which to train our classifier (see Figure **??** for an example).

The following are the objects and associated values that may or may not be present in any given image:

- one and two pound pieces
- 50, 20, and 5 pence pieces
- washer with small hole (75p)
- washer with large hole (25p)
- angle bracket (2p)
- AAA battery (no value)
- nut (no value)

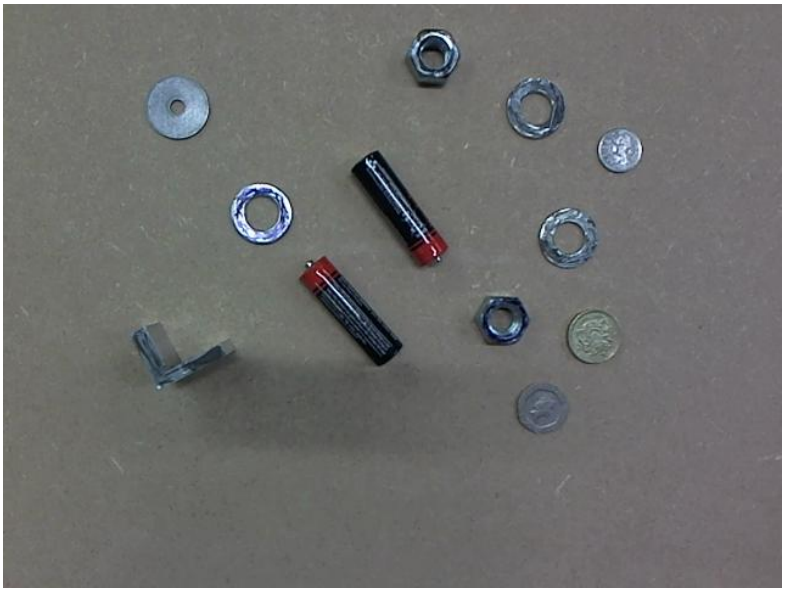


Figure 1: This is one of the test images given to train the classifier.

We approached this problem by dividing it into three distinct stages: background segmentation, object detection, and object classification.

2 Methods

2.1 Background Segmentation

Creating a reliable algorithm that would clearly segment the background from the objects of interest in the image was the most time-consuming and challenging section of this project. We tried several different methods of background segmentation, to varying degrees of success. We ended up choosing median filter thresholding as our most successful method.

2.1.1 Naive thresholding

Our algorithm for creating a naive threshold can be described by the following steps:

1. Attain the median values for each of the three color channels, r, g, b in the given image

2. For each pixel, if that pixel's color values are ± 20 from the median, label it as a background pixel. Else, label it as an object of interest.

This method has a few advantages. It is fast, as it only requires two passes over the entire image, and there are no computationally expensive operations inside of the loop. It is simple and easy to understand. However, this method fails to accommodate for shadows in the background. It also needs to be tuned specifically to the image (the range of ± 20 from the medians was chosen by trial and error). Even after careful tuning, this algorithm still miss-classifies some pixels. See Figure 2a for an example of the output of this method.

2.1.2 Adaptive thresholding

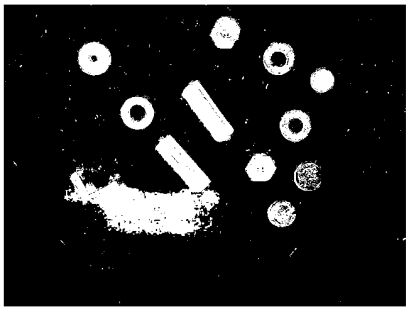
Adaptive thresholding, as opposed to naive thresholding, generates a unique threshold value for a set of sub-images inside the given image. This is meant to allow for shadows to fall on the background and still be classified as background since, the threshold is a more localized value.

This was a fairly successful method, but still had its share of disadvantages. Adaptive thresholding highlighted the edges around some of the objects rather than the objects themselves, but for others, identified the body of the object correctly. This inconsistency would cause problems when trying to classify the object. However, we never had any problems with background shadows when using this method. See the results in Figure 2b.

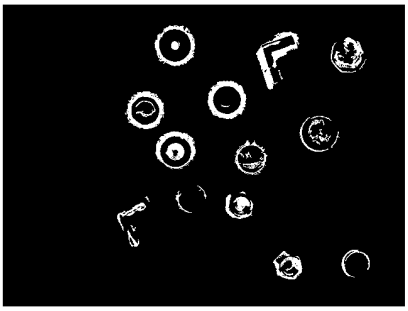
2.1.3 RGB normalization

This algorithm is meant to reduce background shadows as well by normalizing the intensity of a pixel color. It adjusts the r, g, b values based on the following division:

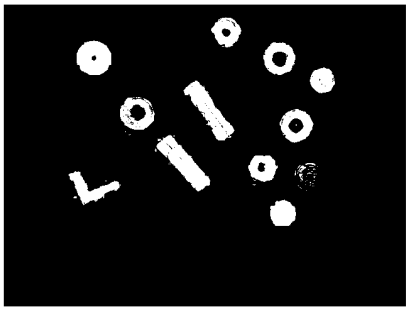
$$r = \frac{r}{r+g+b}, \quad g = \frac{g}{r+g+b}, \quad b = \frac{b}{r+g+b}.$$



(a) naive thresholding

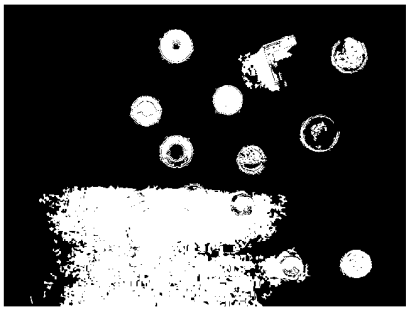


(b) adaptive thresholding

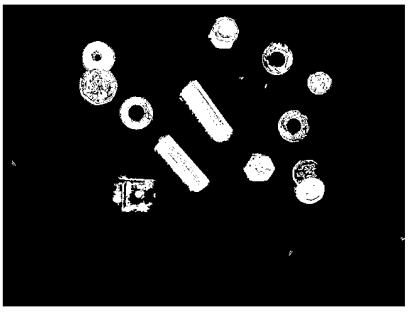


(c) RGB normalization

(d) k-means classification



(e) mean-shift segmentation



(f) median filter thresholding

Figure 2: Here are the output images for the six methods of background segmentation that we tried.

2.1.4 K-means clustering

2.1.5 Mean-shift segmentation

2.1.6 Median filter thresholding

2.2 Object Detection

2.3 Classification

3 Results

4 Discussion

Appendix

```
code can go inside these tags
\begin{verbatim}
....
\ end{verbatim} (no space though)
```