Samuel Petit
Student no: 17333946
Exam no: 46429

Computer Vision CSU44053 - Final Examination

Question 2a - Application

The first thing to consider here is that we notice the cones need to be detected in different environments, that is by day or night. In order to solve this problem, the first operation I will execute on images is a histogram equalisation such as to make the lighting consistent for all images we will be dealing with, this should enable us to be able to process both daytime and night time images.

Histogram equalisation is done by taking the luminance channel from a coloured image, computing it's histogram. Then equalising that histogram. We can then use the values in the equalised histogram as the new luminance by mapping each pixel's luminance to its a value within the new histogram (this works by using a lookup table).

Now that we have an equalised image, let's start with detecting cones. To do so we can use chamfer matching using a cone template. Chamfer matching requires a template which needs to be a binary image. We can obtain such an image by outlining by hand the shape of a cone. Including the colour separation on the cone (from orange to white). From this we obtain our template which should be roughly a triangular shape (only the edges - not filled), includes edges where the colour changes and should align with an actual cone when layering one over the other.

Chamfer matching takes as input this template as well as an edge image. Hence we need to compute the edges of the scene. To do so we can use once again Canny edge detection as we did in the first question. Canny edge detection takes as input a grayscale image so we need to first convert our image to grayscale. This is fairly straightforward, going from RGB to grayscale we can map every pixel in the colour image to a grayscale value using the following formula:

- $Y = 0.2126R + 0.7152G + 0.0722B$

We now have our image as grayscale. We can give it as input to the canny edge detection technique. The output of this will be a binary edge image. There are 2 thresholds that this method uses as parameters, a high and low threshold. These are used by setting anything below the low threshold to a non edge, anything above the high threshold to an edge and values lying within the thresholds range will depend on how connected they are to other edges. In this case we are looking for a good amount of details in edges hence I believe we could start with a relatively low value for the low threshold (such as 100). In terms of the high threshold we could set it to a relatively high threshold such as 200 for example which should give us a fairly detailed edge image. Once again the selection of these parameters should be optimized through cross validation thanks to comparing confusion matrices and their related metrics (f1 score, accuracy…) over a range of threshold values.

Now that we have our edge image and the binary template image, we can use those in Chamfer matching. The chamfer matching algorithm will start by computing a “Chamfer image”. For each point of the provided edges image, the algorithm will compute the distance of the closest pixel that is also an edge. The chamfer algorithm will consider the template in every possible position in the chamfer image, it will compute the sum of the chamfer image values that correspond to object points in the template. The output of the chamfer matching algorithm is a grid (or array) of weights which corresponds to the degree of match of our template within the original image.

We can now extract the local minimas from the grid such as to obtain the positions of cones within our original image. Since we have the positions of the cones in our image, we have solved this problem.

Question 2b - Compare & Contrast

To make it easy for us, let’s give code names to 2 of these methods:

- Robust object detection using a cascade of Haar classifiers: Haar detection
- Principal component analysis: PCA

Let’s start with something all these techniques have in common: they can be used to detect objects within scenes, a common use case might be to detect faces.
While Template matching does not require training in advance of performing object detection, both hair detection and PCA require training as they share a statistical approach to detection. This training for both haar and PCA requires a set of image labels and known outputs in order to be able to train those models.

Once these models have been trained though, we can get to the detection phase. Let’s say we wanted to detect a face within a scene, the template matching approach will require a template to look for within the image, meaning we need to know in advance what or who we are looking for. On the other hand for both PCA and haar detection are more general object detection systems. With the case of faces, PCA and haar detection are able to detect any face with reasonable accuracy in an image while template matching will only be able to detect faces which match (up to a certain degree) a face that we have the image (a template) of. This is also true for these techniques inputs, template matching requires an image and a template to try to match, PCA and haar detection will only require the image to attempt to detect the object they have been trained to find inside of.

Following up on this fact, it is also worth mentioning that template matching is able to detect any object within a scene as long as you have a template for that object. Unlike PCA and Haar detection which can technically be trained to detect most objects, however in practise it is more complicated than that as training such models requires a set of images with known outputs.

Another key difference between these techniques is that while Haar detection and template matching can be used to detect objects within images. PCA is also able to detect objects within scenes but is part of a broader technique which includes data compression and data analysis.

Continuing this comparison, we find that take a somewhat similar approach, template matching will attempt to place the template in every possible position within the queried image, if the size or rotation can vary will make for very expensive searches as we need to consider all the possibilities there too. Haar detection will also locate each of its kernels in all possible sizes and locations in order to retrieve many features, thankfully the haar detection approach is efficient to compute. This is not the case with PCA which works very much differently by analysing data covariance by using eigenvalues and eigenvectors.

Another difference is that with template matching, we need to select a match criterion (some common of these can be the cross correlation or the square difference), we also need to threshold the output such as to obtain the matches by extracting the local maximas (this is done using dilation and looking for unchanged values). This is not the case with haar detection which will handle the thresholding step with its cascade of classifiers as only the accepted matches move on to the next step of the cascade. PCA behaves slightly differently in that we will select a set of eigenvectors (the ones with the highest eigenvalues) and use those to locate the closest match in PCA space.