EE551 Mini-Project
Detection and Classification of Speed Limit Signs

Background

Traffic sign detection and interpretation is an important element of advanced driver assistance systems in vehicles, and is integral to the development of fully autonomous vehicles. An important subset of traffic signs are signs that indicate that a speed limit is in operation, and for which correct detection is particularly critical. Camera-based technologies are an obvious way to achieve this goal and can be used in both autonomous vehicles and in vehicles with drivers as an aid to the driver. The objective of this mini-project is to develop a system to detect speed limit signs, and to determine the applicable speed limit, by processing a database of test images that will be provided.

The overall functionality of the system can be divided into two main components:

1. Detection of a traffic sign/speed limit sign within the scene; this represents a "region of interest" within which other processing needs to be carried out;
2. Analyse the content of the speed limit sign, and determine the speed limit.

Within a given jurisdiction, speed limit signs typically follow a particular layout. For example, signs in the UK (and Ireland) typically look something like Figure 1:

Figure 1: "Template" speed limit sign

i.e. consisting of black digits on a white background, enclosed in a red circle. A typical scenario in which a speed limit sign would appear would be the following example:

Figure 2: Example speed limit sign (in rainy conditions).

To detect such a sign in an image of a road scene, you will e.g. need to detect circles, then red circles, and then isolate the white region with black digits within. There are several ways in which this could be done. For example, a useful and widely used technique to detect circles is the Hough Transform, which uses edge detection (Chapter 4 of the textbook) as a component of its processing (see the Matlab function "*hough*" for an example implementation). However, you may use any other method you like as well.

Once you have detected the circles, you will need to process the "contents" of the circle to check what digits they correspond to (if indeed they are digits). This is a simple form of "classification", where typically, features are extracted from the image and compared to a library of pre-calculated "templates". Whichever template is closest to the features extracted from the unknown data is the one that is chosen. First of all, the different "digits" need to be isolated (e.g. this could be done by processing a binary image with the function "*bwlabel*"; several examples of its usage have been used in the textbook and associated Matlab examples), and then features extracted from the image. Feature extraction is the subject of Chapter 9 of the textbook, which covers a variety of features, including simple shape-based features (simple features, as well as moment-based features), through to more sophisticated features. In essence, this produces a set of numbers that "capture the characteristics" of the object. You can use some of the features covered in Chapter 9, but you're not restricted to this; if you're feeling adventurous (and would like to earn extra marks ☺) you can investigate other features that we didn't cover in the lectures.

Databases

Three data sets have been placed on Blackboard (in the "Mini-Project" folder under "Assignments"). One of them comprises perfect "Gold Standard" templates (like Figure 1) for some of the speed limit signs. Another of them is the "Development Dataset" and contains a number of zip files, each of which contains many examples of speed limit signs corresponding to different speed limits (20, 30, 50, 80 and 100 kph). In this data set, the speed limit signs occupy a large percentage of the image area, so represents an "easier" scenario than the example in Figure 2 above. However, this data set does cover a wide range of conditions: lighting, blurriness etc. so has its own challenges. This data set is a good one can be used for software development, and you are expected to present performance results from your system for the images in this data set e.g. percentage correct detection of the "speed limits" (of which there are 5 classes). Depending on how you build your system, you can use this data set as you see fit. For example, if you decide to also look at some sort of machine learning/classification system, you might decide to use half the images for training, and the other half for testing; an 80%/20% split between training and testing is also a common approach.

The third data set is called the "Stress Dataset" and contains a small number of images of more challenging scenarios (like Figure 2). You are also asked to test your system in these more challenging cases and report on performance.

Points to keep in mind

- Shape detection is often done with a grayscale image so you can convert colour images to grayscale first. At the same time, don't forget that the red circle around the outside of the sign has considerable significance in this application, so you should still do some of your processing on the RGB image.
- At various points in the processing, you may need to "fix" small imperfections in the images, or objects contained therein, e.g. you may have white blotches in a digit that need to

be removed or filled in. You can use the morphological operators covered in Section 6 to do this.

- You will need to calculate "templates" for each digit that is likely to appear, so that unknown data can be compared with these templates (and the closest one chosen as the recognised digit). The template feature sets can be derived from "ideal" representations of the speed limit signs that you will be given along with a set of real road scenes that include speed limit signs.
- You will also need some way of "comparing" the features from unknown data with the digit templates. A **very** simple way of doing this is to calculate the Euclidean distance between the templates and the test features (as you may already know, the Euclidean distance is calculated from the square of the differences between the template and test features); the template with the smallest "distance" to the test features is the one selected. However, you can "stretch" yourself a bit here and investigate more sophisticated feature and classifiers. Even better, a comparison between a simple Euclidean-based system and something more sophisticated would also be good.
- If you are using e.g. simple shape-based features to represent the "digits", it is likely that more than one feature will be needed to adequately distinguish between different digits.
- Use *a priori* knowledge of the application to place some constraints on the processing, for example, the second digit in a speed limit is normally a "round number" like either 0 or 5. This restricts the possibilities you have to search for. Likewise, the first digit in a speed limit is normally one of a finite number of possibilities.

Marks

System for detection of circles and pre-processing to isolate digits representing speed limit, classification of detected objects and extraction of the speed limit from the sign: **55%**
Discussion of results, and suggestions for enhancements etc.: **20%**
General presentation quality of report, code etc.: **10%**

The remaining 1**5%** of the marks will be used to reward particular innovation in the implementation of your system. For example, while you can implement a very "basic" classifier with a simple set of e.g. shape-based features and the Euclidean distance, you will gain extra credit if you look at some alternatives – particularly if you can demonstrate superior performance with an alternative approach. A "comparison" of more than one approach would be particularly nice, along with some critique of the strengths and weaknesses of different approaches.

NB. For this project, don't feel constrained to use the approaches suggested above – feel free to investigate any other technique you like. At the same time, there is a wide range of software available to implement high level operations and these can make building a system like this one quite easy. However, while you're not restricted to use any particular features or approaches, you are discouraged from using "black box" approaches like what you'll find on the internet, and which can (often) be implemented with a few lines of code; instead, you're expected to demonstrate that you have an understanding of what's happening at a "low level" (hence the allocation of marks above).

Submission

You must submit the following:

- Working Matlab .m script for implementing the system.
- Report detailing your work, including a comprehensive set of results, graphs of performance, and your analysis and critique of the results. Provide any illustrative images (as many as you feel necessary) to illustrate what's happening. Reports should be as long as they need to be and no longer (i.e. no "padding"!). Submit your report as a PDF or a Microsoft Word document.

Details on submission, deadlines etc. will be communicated to you via e-mail.

Notes

1. Marks are awarded for the quality and performance of your algorithm as well as for "working code", for each fault case. Where appropriate, your report should include a description of your algorithm that is, in a sense, "software independent", so use flow charts, pseudo-code, equations etc. as you need to. Marks will also be awarded for the "critique" of your system and its performance.
2. You should structure your source code so that all that needs to be done is to "point" the software at a directory that contains a set of test images to be processed – these could be directories that contain only one type of fault, or it could be a directory that contains all of the randomised images. It **must** be possible to run your software with minimal editing i.e. only directory names should need to be changed to reflect differences in the file structure on your own laptop and those who will be assessing your submissions. For submission, you can have all of your code ("main" program and any functions) in one source M-file; however, if you do this, remember that the functions will have limited scope and will not be visible outside this source file.
3. For submission, you can have all of your code ("main" program and any functions) in one source program; however, if you do this, remember that the functions will not be visible outside this source file. Also, for submission, if TurnItIn has difficulties with the .m file format (it also can have difficulties with Zip files) you can improvise and save your Matlab script as a .TXT document or MS Word document and upload that – as long as it's uploaded in some "text" format, that's fine. You should also e-mail your report and source code (.m file) to edward.jones@nuigalway.ie as well.

Plagiarism

You will no doubt be aware that the University treats plagiarism very seriously. Students are expected to work independently and to submit their own original report, without copying material from another person, or from another source. Submissions will be processed by Turnitin to detect plagiarism. Further information relating to plagiarism may be found at: http://www.nuigalway.ie/plagiarism/

Also, while you may discuss and debate elements of the assignment with your classmates, **each student must write their own code, and complete and submit their report independently of all other students** (TurnItIn will be used to check for plagiarism) – in other words, you can "collaborate, but don't copy".