Curtin University, Department of Computing

Machine Perception Assignment: Semester 2, 2018

Assignment Cover Sheet and Declaration of Originality

This form must be completed, signed, scanned and attached as the front page of your documentation.pdf file in your assignment submission.

Family Name: Dao

Given Name(s): Nhan Duy

Student ID: 18843905

Stage Complete (student to fill in):

Circle one of the following: 1 (10%) 2 (20%) 3 (30%) 4 (50%) 5 (70%) (6 (80%))

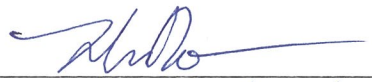
Final Stage complete? Circle one of the following: (YES (+20%)) NO (+0%)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: 

Date of Signature: 5/10/18

By submitting this form, you indicate that you agree with all the above text.

Main Pipeline

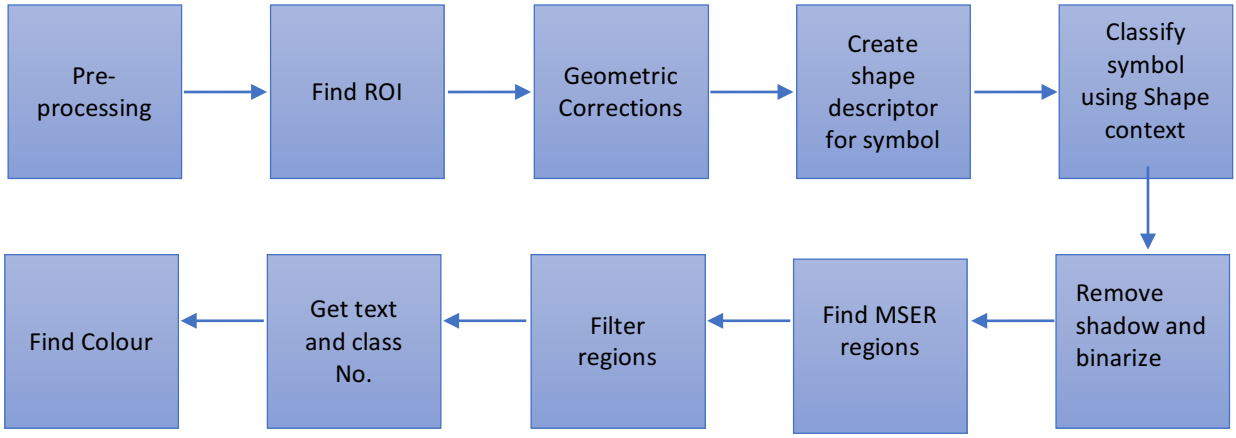


Figure 1

Pre-processing:
This step involves grey-scaling the input image applying median blur and Gaussian blur. This helps with reducing noise in the image. Applying this filter will help improve the next stage of the pipeline which is finding regions that resembles hazard labels. The window size and sigma value of the kernels are empirically determined through try and error.

Find Region of interest:
To find regions that resembles a diamond shape; the first step is finding the edges using Canny Edge algorithm, then a closing morph function is to further remove weak edges. The resulting image is then imported into OpenCV's findContour.

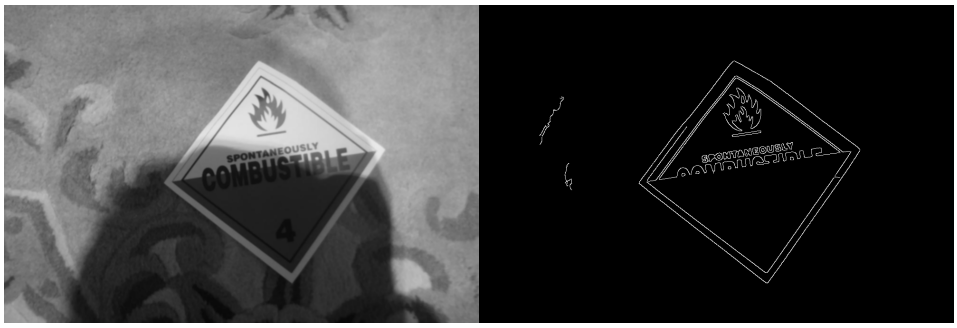


Figure 2

Then the contours are filtered based on their area; this is done to remove spurious hits that are not likely to be hazard labels. This maybe information destructive in cases where labels are small in area – but for our specific case, the labels have a defined lower bound, and its upper bound can be thought of as the maximum area the label can fit in the image without clipping on the edges.

Contours are then filter out contours that resembles a parallelogram. This is done by approximating polygonal curves of the contours within a specific precision, see (Ramer-Douglas-Peucker algorithm). Polygons that have sides of 4 are considered regions of interests, thus this stage can identify multiple hazard labels (although it doesn't classify them), making this relatively efficient.

Geometric correction:

Next step is to perform a geometric transformation which will geometrically transform the image to the correction orientation and size (see figure 2). Now the rest of the pipeline can ignore any variance in scale or orientation, thus reducing the geometric complexity of the later stages.

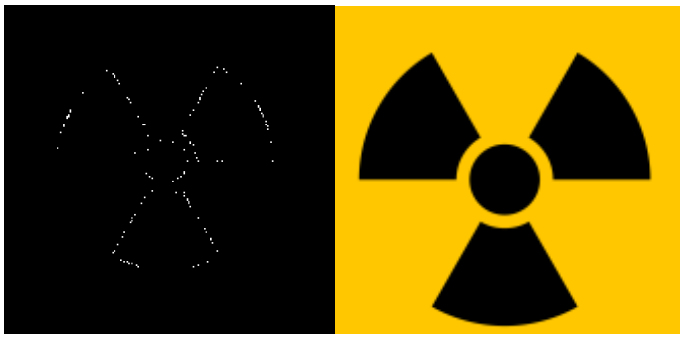
Figure 3

This makes the later stages relative easy through naïve yet effective assumptions – class number is always at the bottom of the label, texts are somewhere in the middle, and the symbol is near the top. It should be noted that the Cartesian boundary of these features aren't important, what's important is recognizing these features. After all, these labels were design by humans within specific standards, and it's important to design a pipeline with these consideration in mind.

Create shape descriptor for symbol:

This module of the pipeline is to extract points on the symbol's contours, these points are used to calculate symbol's shape context descriptor (See next stage). The perspective corrected image is first cropped, then OpenCV's findContour function on the Canny Edge image to retrieve a list of points on the symbol's contour. The sample 130 random points of the symbol for shape context descriptor.

Using this method, symbols were identified relatively easy; there were cases where the texts that appears higher up such as 'spontaneously' are picked up as being part of a symbol, which sometimes causes false symbol recognition, however this was not addressed.



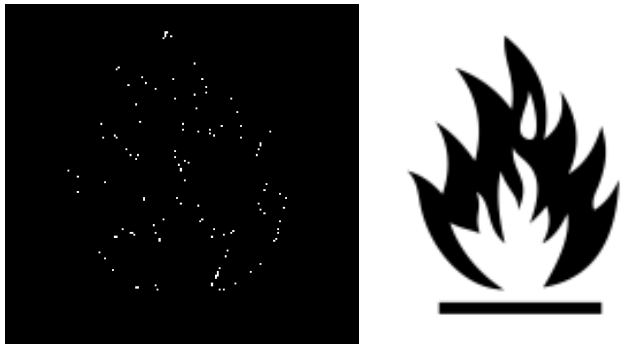


Figure 4

As described in the paper published by S. Belongie, J. Malik and J. Puzicha *"Shape context: A new descriptor for shape matching and object recognitions"*, shape context creates a rich local descriptor. Shapes are about relative positional information, the approach is to create a descriptor for each point using relative information of nearby points, namely distance and angle. Therefore, matching points will have similar relative information.

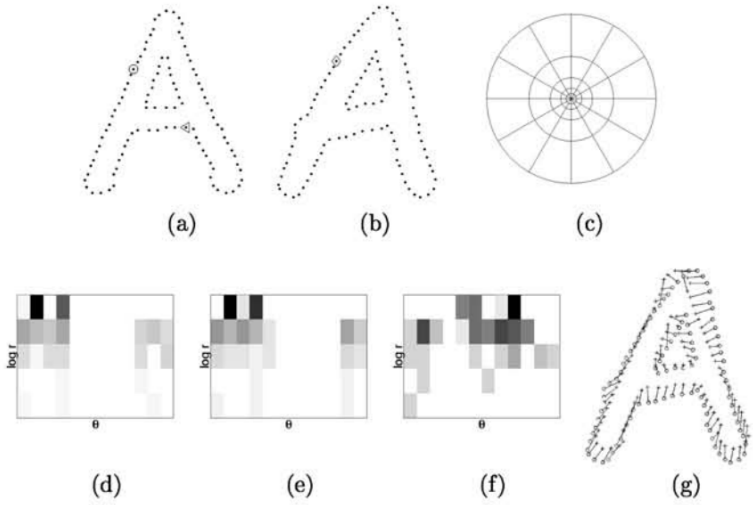


Figure 5

Figure 3, (a) and (b) are sampled points of the shape and (d), (e) and (f) are respective descriptors of each respective points circled on the shape. Points that are similar will have similar descriptor. (g) shows the correspondent points, finding these pairs of points is a bipartite matching problem.

The motivation to use shape context as a descriptor to classify hazard symbols is its ability to classify similar shapes. For example, consider two different manufacture creating flame symbols that are slightly different - A template matching approach would not be effective compared to shape context because it does not allow for such variations.

Classifying symbol using Shape context:

To find the matching shape, the descriptor of the symbol to match is first computed as outlined in the previous stage. Then the cost to match this descriptor with pre-calculated descriptors of each known symbols are calculated (by solving a bipartite graph problem). The symbol that matched with the lowest

cost is the correct symbol. The symbol with the lowest cost is sanity check by making sure that the cost to match that symbol is under a certain threshold, otherwise the region of interest is probably not a hazard label. Because there are no modules in Opencv to perform shape context, it is implemented using python hence it is rather slow and contributes the most to the slowness of the program.

Remove shadow and binarize the image:

Done by adaptive thresholding the V channel of the HSV colour space of the image. The rationale to use the V channel was that it was more sensitive to changes in light intensity/shadow – meaning adaptive thresholding this specific channel will make regions affected by shadows the same as regions that aren't. This binarize image is used to detect regions of text, note the artefact caused by adaptive thresholding at boundary of the shadow, this artefact did not seem to affect much in detecting the text.

Figure 6

Find MSER regions:

This stage finds the maximally stable external regions, the goal of this is to find blobs that resembles text. The MSER function is performed on the binarize threshold image to find blob. Regions are then filtered out based on their area and clustered together based on the similarity of heights of nearby regions and x/y coordinates. The depth first search algorithm was used instead of simply clustering regions by y coordinates is because sometimes the geometric correction did not work correctly and the words appear slanted, therefore clustering the regions by y coordinate would split the words into two clusters.

Figure 7

Using MSER to detect text regions is critical for the text recognition because it allows us to manipulate each individual letter for better recognition such as morphing/spacing the characters evenly out. This will be outlined in the next stage.

Get label and class number:

Tesseract was used to do this; this method was done in favour to creating an OCR for this these specific applications because it allows for text detection on the unseen label. It is also faster and is easier to implement. However, passing the original or even the threshold image into tesseract will produce a jumbled mess. Therefore, the text regions are processed, namely spacing out the characters and applying a dilating morph (Somehow that help tesseract detect the character). The output text is then sanity checked against a dictionary file using a longest common substring method.

Figure 8

Finding the colour:

To find the colour, the image was converted to HSV colour space. This is done because it's easier to determine the boundary values between one colour and the next colour based on their hue. HSV channel stands for hue, saturation and value. To determine the colour, boundaries are set between the H channel. For black and white text, however boundaries are set in the S and V channel. White are any pixel below a certain saturation value and black are any pixels below a certain V value.

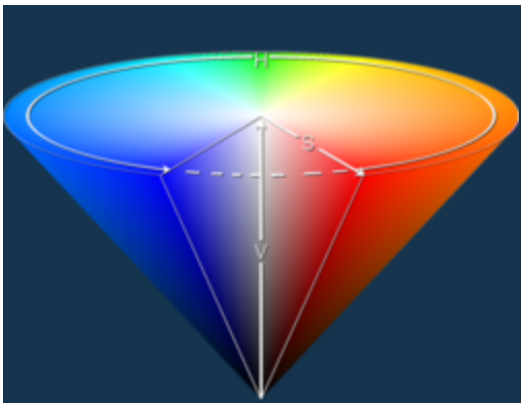


Figure 9

There was nothing done to correct the image hue, nor were there any method to determine the colour via the context of the label. But these methods were considered.

References

Belongie, S., J. Malik, and J. Puzicha. 2002. "Shape Matching And Object Recognition Using Shape Contexts". *IEEE Transactions On Pattern Analysis And Machine Intelligence* 24 (4): 509-522. doi:10.1109/34.993558.

Nikishaev, Andrey. 2018. "Shape Context Descriptor And Fast Characters Recognition". *Medium*. https://medium.com/machine-learning-world/shape-context-descriptor-and-fast-characters-recognition-c031eac726f9.

"Opencv: Contour Features". 2018. *Docs.Opencv.Org*. https://docs.opencv.org/3.3.1/dd/d49/tutorial_py_contour_features.html.

"Python Tutorial: Longest Common Substring Algorithm - 2018". 2018. *Bogotobogo.Com*. https://www.bogotobogo.com/python/python_longest_common_substring_lcs_algorithm_generalized_suffix_tree.php.

Rosebrock, Adrian. 2018. "Opencv And Python Color Detection - Pyimagesearch". *Pyimagesearch*. https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/.