Machine Perception Assignment:
Semester 2, 2018
Detect and Recognise Hazmat Labels
Department of Computing, Curtin University
v1.01

Raymond Sheh
Raymond.Sheh@Curtin.edu.au

August 13, 2018

Abstract

You now have an open-ended opportunity to demonstrate what you have learned thus far, and in the coming weeks, about machine perception. You will need to make decisions on what to use, when and how, in order to solve a problem. You will also need to decide for yourself how to structure your solution to the problem, what techniques to use, how to bring them into your program and so-on. Feel free to use the work you have done for your tutorials, however you **MUST** self-cite any work that you are re-using. You are not allowed to use external code except for OpenCV and the standard (non image processing) libraries available in the labs. If in doubt please ask.

Note that the assignment specification is deliberately open ended and somewhat ambiguous. Making reasonable assumptions, stating them and then implementing them is part of your job in this assignment! If in doubt, please do ask but be prepared for the answer ''Well, what do you think?''.

Please start the assignment early! It has been structured in stages with most of what you need to do most of the stages already covered in the lectures so you can be doing groundwork and picking up marks right from the start. If you leave starting the assignment for when we have covered everything you will need for the whole assignment, you **will** run out of time.

A mark of 20% or more is considered a substantial attempt. This means that you must achieve at least 20% to pass the unit.

GOOD LUCK! *Not that luck should have any part in this of course! ;-D*

1 Introduction

Robots interacting in human environments will need to be able to understand signs and labels that are designed for human, rather than computer, consumption. This is a challenge because things that humans tend to find invariant, such as colour, precise shape and so-on, can be very different for a computer.

For example, we can read text in a very wide variety of fonts without much difficulty. Even if we come across a new font, we don't need to ''re-train'' to understand it, we can (usually) figure it out pretty easily. Likewise we can recognise symbols, such as emojis and icons, very easily even if we haven't seen a specific version of it before.

In this assignment, you will tackle the challenge of detecting and interpreting UN standard hazardous materials labels such as those shown in Figure 1[1]. These labels have certain attributes, the ones we care about are:

- Colour of the top half of the background.
- Colour of the bottom half of the background.
- Class number.

[1] Adapted from ''Albert, I. 2018. *Downloadable Hazmat Placards.* `http://ian-albert.com/hazmat_placards/`''.

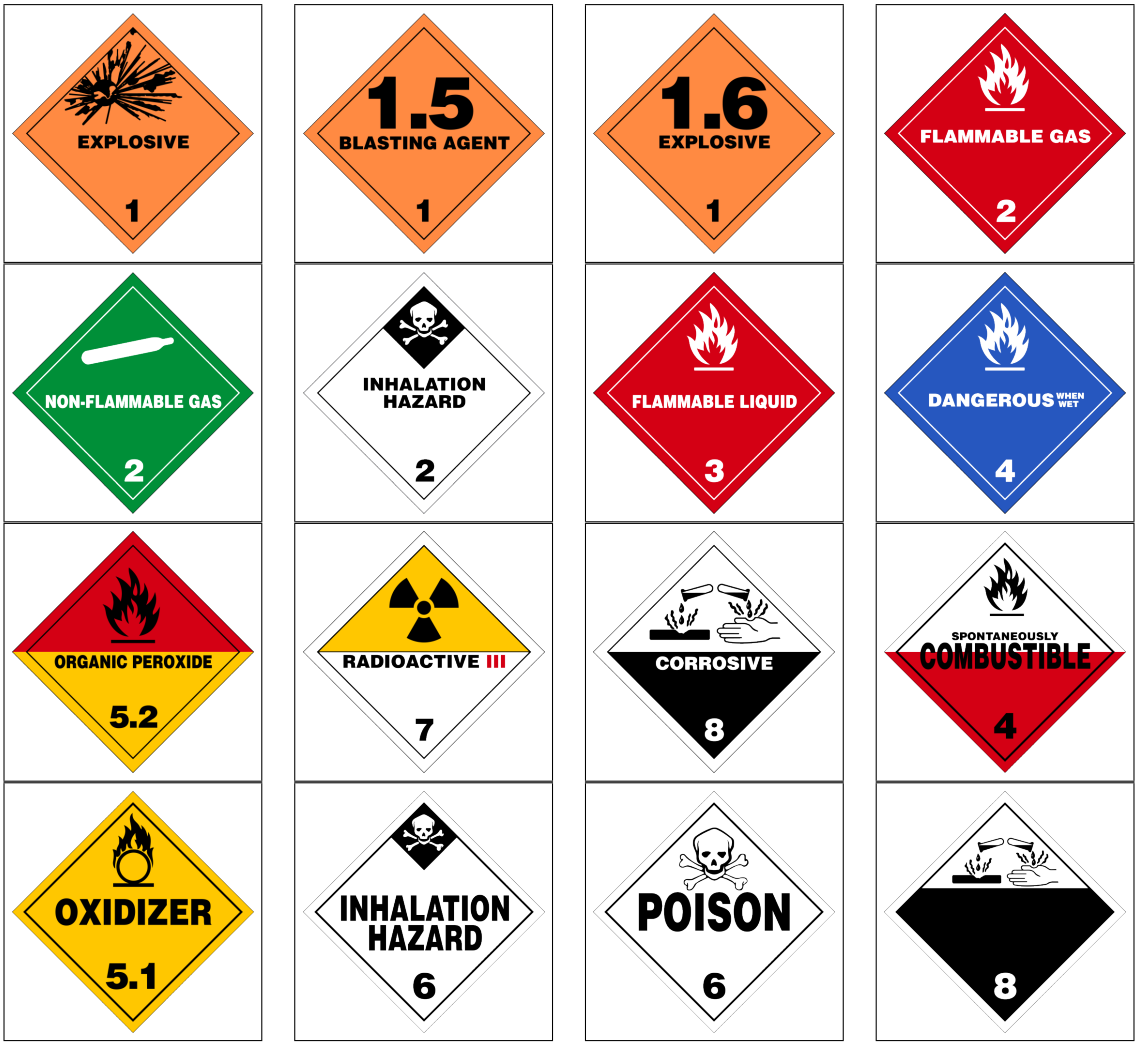

Figure 1: Examples of hazardous materials labels. Note that some of these are missing text (such as the label in the lower right corner missing the "CORROSIVE" text) so you can't rely on the text being present to recognise the symbol. These are some of the images from Set A, mentioned below.

- Other text.
- Symbol.

For the purpose of this assignment, we will ignore labels with the following characteristics.

- Labels which have a background pattern rather than a solid colour, such as "Hazardous Solid".
- Labels that have explanatory text on them. The text in the labels you will need to recognise will not be smaller or denser than those shown in Figure 1.
- Labels with multiple class numbers or class numbers that are more than just a number (with potentially a period).

Your program should start with no parameters and process all .png and .jpg images in the working directory in alphabetic order. For each image, it should report the attributes listed above for every hazmat label found. Performance marks will be awarded depending on how many of the test labels are detected, the attributes you are able to detect, for the absence of false positives or erroneous detections. You will also be awarded marks for good software engineering and coding practice, as well as efficiency and readability.

All of the test pictures we may use will be uploaded to Blackboard (although not necessarily at the same time -- some may appear a week before the due date). We will be selecting from those to test.

There are no marks for a brute force matching implementation. To make sure you implement something vaguely efficient, we will limit your program to an average of around 10 seconds of CPU time[2] per image, when run on the lab computers (with no other users present). Your program will be terminated if it takes much more than this and marked on whatever it is able to report by then. Everything must run locally on the CPU -- no using cloud computing, distributed computing or the GPU.

1.1 Colours

Each label has a top half colour and a bottom half colour (which may be identical for a solid background). We will use the following names for the colours:

- Orange
- Red
- Green
- White
- Blue
- Yellow
- Black

Note that you will need to set some thresholds between these colours as there will be some variation (the examples that will be available on Blackboard are representative of this variation). Some of the files you will get involve photographed labels where lighting will affect the appearance of the colour so in later stages you will need to do some sanity checking based on other things on the label and perhaps some white balancing based on other information you have in the label. Also note that your program should not be confused by the presence of the text or symbols.

1.2 Class number

Each hazmat label contains a number in the bottom corner, denoting the class of material. Note that this is not necessarily the largest number on the label (some labels, such as some Explosive labels, have a large number for the symbol).

Note that in the final stage (see below) the font and size may vary although you can assume that it will be a font that is designed to be easy to read.

1.3 Other text

The labels will generally have other text on them, ranging from simple (eg. ''EXPLOSIVE'') to multi-line (eg. ''NON-FLAMMABLE GAS'') to text which might be partially split onto two lines (eg. ''DANGEROUS WHEN WET''). Note that the text should NOT include the class number or any text in the symbol. The text may overlap the midline between two different coloured backgrounds (such as in the ''SPONTANEOUSLY COMBUSTIBLE'' label in Figure 1).

Again note that in the final stage, the font and size may vary although you can assume that it will be a font that is designed to be easy to read.

[2] We can re-negotiate this, particularly for more complex tasks, if many people are having trouble getting under this limit.

1.4 Symbol

The labels will also have a symbol in the upper corner. This symbol will be one of the following, in black or white against the background:

- Flame.
- Gas cylinder.
- Skull and crossbones.
- Skull and crossbones on black diamond.
- Oxidiser.
- Radioactive.
- Corrosive.
- Explosion.
- A number (eg. on some Explosives labels).

I'm sure you can work out what each of these are based on Figure 1! Note that in the final stage (see below) the exact pattern/shape of the symbol may differ. For example, there are a bunch of ways of drawing the flame or the explosion or the environmental hazard and there may be a few different fonts and sizes for the number. Your implementation will receive more marks if it is able to recognise more variations.

2 Marking Breakdown

This assignment is deliberately structured in stages, allowing you to get some marks on the board quickly but also providing a challenge right through to the end of the unit. You should practice good software engineering that should allow you to add capabilities to your program without requiring extensive rewriting or reorganising.

Note that you **MUST** nominate which of the following stages you consider your submission to satisfy and we will test only on that section. Do this by circling one (and only one) of the main stage on the cover sheet, and one (and only one) of the YES/NO options of the final stage on the cover sheet. See last page of this document for details.

If you don't, we will test the datasets from Stage 1 in order and assume that you have completed the last dataset to achieve 100% success. This means you will receive no marks for partially completed stages. It will also mean your assignment will receive zero marks if it doesn't achieve 100% success on the Stage 1 dataset.

Also note that you are submitting **ONE** program. A program that satisfies a later stage automatically satisfies earlier stages.

Your mark is computed as follows:
$mark = stage\_maximum \times percent\_dataset\_success \times quality\_multiplier$
Where:

- *stage_maximum* is the maximum of your nominated stage (or the one we decide as per above if you didn't nominate one).
- *percent_dataset_success* is the percentage of the test dataset that your program succeeds on. Success in this case is computed per attribute per image so you can receive partial marks even if you consistently fail to correctly identify one attribute.
- *quality_multiplier* is as defined in Section 2.8.

image1.jpg

image2.jpg

image3.jpg

Figure 2: Example input for Stage 1, 2 and 3.

2.1 Stage 1: Up to 10% -- Identifying Background on a Computer Generated Label

Your program will be run in a directory containing one or more images from **Set A**. These are all computer generated images. Your program should report the colour of the top and bottom halves of the background. For example, when provided with images shown in Figure 2, your program should output:

```
image1.jpg
top: orange
bottom: orange

image2.jpg
top: white
bottom: red

image3.jpg
top: white
bottom: black
```

2.2 Stage 2: Up to 20% -- Identifying Text on a Computer Generated Label

Your program will be run in a directory containing one or more images from **Set A**. These are all computer generated images. In addition to the colours of the top and bottom halves of the background, your program should also identify the text on the label. If the class or other text is missing, your program should report "(none)" in the relevant place. Note that Set A does include the split multiline "Dangerous When Wet" label.

For example, when provided with images shown in Figure 2, your program should output:

```
image1.jpg
top: orange
bottom: orange
class: 1
text: EXPLOSIVE

image2.jpg
top: white
bottom: red
class: 4
text: SPONTANEOUSLY COMBUSTIBLE
```

```
image3.jpg
top: white
bottom: black
class: 8
text: (none)
```

Note that your program **must** be reading the text -- somewhere in your program you need to be reading text from the image (and possibly removing/discounting things that aren't relevant text). You will only receive half of the marks if you just template match without actually decoding the letters individually to text at some point in your system using OCR (we test your template matching in the next stage). You will receive no marks if you just use the background colour to determine the text (eg. just looking for red/yellow and immediately saying that this corresponds to "ORGANIC PEROXIDE" or white/black and saying "CORROSIVE".). To enforce this, some of the labels in Set A will have the text altered but otherwise be identical so there will be some ambiguity.

2.3 Stage 3: Up to 30% -- Fully Identifying a Computer Generated Label

Your program will be run in a directory containing one or more images from **Set A**. These are all computer generated images. In addition to the colours of the top and bottom halves of the background and the text, your program should also identify the symbol at the top of the label and report them according to the names above. For example, when provided with images shown in Figure 2, your program should output:

```
image1.jpg
top: orange
bottom: orange
class: 1
text: EXPLOSIVE
symbol: Explosion

image2.jpg
top: white
bottom: red
class: 4
text: SPONTANEOUSLY COMBUSTIBLE
symbol: Flame

image3.jpg
top: white
bottom: black
class: 8
text: (none)
symbol: Corrosive
```

You should **NOT** use the text or class to sanity check your symbols. This is because while this might be a reasonable thing to do in real life, it would also mean you could simply skip the symbol detection and just use the text. (For real labels, the symbol is redundant if you can read the text.) To enforce this, Set A contains labels that are missing their text and only contain a symbol, such as image3.jpg in Figure 2.

2.4 Stage 4: Up to 50% -- Fully Identifying a Single Photographed Label on a Plain Background

Your program will be run in a directory containing one or more images from **Set B**. These are the Set A labels printed out, stuck to plain backgrounds and photographed. The labels will overlap the centre of the image, not be clipped against the edges of the image, will be within 30° of upright and will be at

image1.jpg

image2.jpg

image3.jpg

Figure 3: Example input for Stage 4.

least 20% of the image, and at least 100 pixels, in width and height but can otherwise be any size and orientation. In particular the label may be photographed at an angle. The labels may have a white border that may or may not be cut perfectly straight. Lighting will be reasonably smooth but not guaranteed to be consistent.

You will be expected to report as per Stage 3. For example, when provided with images shown in Figure 3, your program should output the same as Stage 3:

```
image1.jpg
top: orange
bottom: orange
class: 1
text: EXPLOSIVE
symbol: Explosion

image2.jpg
top: white
bottom: red
class: 4
text: SPONTANEOUSLY COMBUSTIBLE
symbol: Flame

image3.jpg
top: white
bottom: black
class: 8
text: (none)
symbol: Corrosive
```

2.5 Stage 5: Up to 70% -- Fully Identifying a Single Photographed Label with Complex Background and Lighting

Your program will be run in a directory containing one or more images from **Set C**. These are the Set A labels printed out, stuck to different backgrounds and photographed. Some will be on complex backgrounds, some will have shadows and some will have both but will otherwise be similar to Stage 4.

You will be expected to report as per Stage 3. For example, when provided with images shown in Figure 4, your program should output the same as Stage 3:

```
image1.jpg
top: orange
bottom: orange
class: 1
text: EXPLOSIVE
```

image1.jpg

image2.jpg

image3.jpg

Figure 4: Example input for Stage 5.

```
symbol: Explosion

image2.jpg
top: white
bottom: red
class: 4
text: SPONTANEOUSLY COMBUSTIBLE
symbol: Flame

image3.jpg
top: white
bottom: black
class: 8
text: (none)
symbol: Corrosive
```

2.6 Stage 6: Up to 80% -- Fully Identifying Multiple Photographed Labels with Complex Background and Lighting

Your program will be run in a directory containing one or more images from **Set D**. These each contain two or more Set A labels printed out, stuck to different backgrounds and photographed. Some will be on complex backgrounds, some will have shadows and some will have both but will otherwise be similar to Stage 4. Each label will be at least 20% of the image, and at least 100 pixels, in width and height.

You should report each label as per Task 3, in the following order:

- Top colour name in alphabetic order.
- Bottom colour name in alphabetic order.
- Class number in numeric order.
- Text in alphabetic order.
- Symbol name in alphabetic order.

For example, when provided with image shown in Figure 5, your program should output the following:

```
image1.jpg
top: orange
bottom: orange
class: 1
text: EXPLOSIVE
symbol: Explosion
```

image1.jpg

Figure 5: Example input for Stage 6.

```
top: white
bottom: black
class: 8
text: (none)
symbol: Corrosive

top: white
bottom: red
class: 4
text: SPONTANEOUSLY COMBUSTIBLE
symbol: Flame
```

2.7 Final Stage: Additional 20% to Stage 2, 3, 4, 5 or 6 -- Alternative and Unseen Labels

In the introduction to this assignment, we talked about how we need to be careful that our perceiving machines are able to recognise things that are equivalent in the real world even though they may not look identical. However, up until now, all of our hazmat labels have been based on the Set A labels so you only had to deal with the one set of fonts and symbols. It's still not trivial but realistically you could just template match after doing various corrections.

In this final round, you will receive up to an additional 20% for your program succeeding in also completing your nominated stage on a set of labels that you have not seen but are also compliant with the UN standard and have the same meaning as those in Set A. Like Set A, we will also be deleting the text from some of the labels so your program will need to be able to detect the symbol without the aid of the text.

Basically, we'll re-run your program with sets of labels from different manufacturers. The percentage of attributes/labels that you get right there will be multiplied by 20% and added to the stage maximum of whichever stage you declare, Stage 2 or beyond (Stage 1 doesn't require you to deal with fonts or symbols). If you have completed Stage 6, this final stage will bring you up to a maximum of 100%. If

you're only able to run on the well centred, computer generated images (Stage 3), this will bring you up to 50%.

We will **not** provide you with these labels beforehand but you are welcome to train and develop on other labels you find elsewhere (doing a search online should provide you with plenty of examples of alternative labels). You will also need to declare that you wish to have your program tested for the bonus stage or we won't run your program over the bonus dataset. The bonus stage is a "bolt on" to whichever main stage you nominate, you can't have your program marked for one stage and then the bonus for a different stage.

2.8 Quality Multiplier

Your code is marked not just on if it performs the task but also on the quality of your submission. It is divided as follows:

30% Code testing, fixed at 30%. So if everything else is terrible but your program works, you get a 30% quality multiplier.

40% Design decisions. Did you pick sensible ways of doing things, balancing efficiency, readability, ease of implementation and so-on. Note that there are many ways of doing things so you will be expected to justify your choices.

30% Coding quality. How well is your code actually written and structured, do you have a sensible amount of commenting, are you using a reasonable coding style and layout (hint, it's OK to use an automatic code formatter). Is your directory structure sensible. Do you have your data or models stored in a reasonable way. Have you removed all un-necessary files (including those left by your operating system, version control system or IDE).

3 Submission

You will be required to submit to Blackboard a single archive, either a .zip file or a .tar.gz file. This file should have the name `MP_Assignment_99999999.zip` or `MP_Assignment_99999999.tar.gz` where 99999999 is your student number.

We will decompress this into a directory on a clean Machine Perception 2018 virtual machine and then, in that directory, do the following:

- Attempt to open a file called `documentation.pdf`. We expect this file to contain your cover sheet (print the final page of this document) with signed declaration and your nomination of which stage you have completed, as well as a few pages of documentation outlining your design decisions.
- Attempt to find a file called `readme.txt`. This file is optional and should only be used to specify the command that will compile and run your program (if it isn't the defaults specified below).
- Figure out if your program is Python or Java. If it contains any .java files we'll assume it's Java (regardless of if there are .py files as well -- but then we'll consider these un-necessary files and you'll lose code quality marks).
- Look at your code and other files and give you a score for coding quality.
- Compile your program (if it's a Java program) using `javac *.java` or by copying and pasting the line you specified in your readme.txt file.
- Copy one or more .jpg and/or .png files (the test files for the stage you nominated) into the directory.
- Execute `java MPAssignment` (for a Java program) or `python MPAssignment.py` (for a Python program) or by copying and pasting the line you specified in your readme.txt file.
- Terminate your program after a total of $(10 \times number\_of\_test\_files)$ seconds of CPU time (if it hasn't finished by then).

- Compare your output to the expected answer and give you a score for code testing.

We highly recommend you download your submission after you submit it, perform these steps and make sure it yields the result you expect. Any component that doesn't work will not receive any marks, even if it's a minor oversight (such as incorrect filename or file format). At this stage in your studies, you should not be making those.

If this procedure does not work, we may spend a little bit of time investigating why but in the general case, we will return the assignment to you with a zero for code testing. If this happens, you will have an opportunity to demonstrate your program working (most likely during the practical). If it can be done following the above procedure you will receive the relevant marks, if not you will be given an opportunity to correct your program or the commands to compile/run but a penalty will be applied as if your submission was late.

4 Final notes

- Start early! The semester will creep up on you quick. The complexity of later stages of this assignment won't be obvious when you start. This is due week 10, aim to be largely done by week 8. We're speeding up the lecture schedule so this should be doable (with the possible exception of the final stage which we'll cover in week 8).
- You have 8 weeks to do this assignment so are expected to plan for potential unforseen disruption. Any unforseen disruption (health for instance) will need to be significant for extensions to be granted. As a general guide, any unforseen disruption of less than a week will not be considered.
- I will be regularly uploading new sample files to Blackboard for you to test with. I suggest you keep an eye on this. I will send out announcements when this happens.
- I will also be uploading amendments to this assignment specification, again keep an eye on this to avoid doing more work than you need to or wasting effort on things that don't answer the question. I will also send announcements when this happens.
- Ask questions! If something is uncertain, either send me an email or ask me during the lecture. I don't want you doing a bunch of work that might not be required. If it's a general issue I may make amendments to address them.
- Please don't try and loophole the assignment. You know what is expected and yes there are deviations from what might be done in the real world, they're there to exercise different capabilities.
- Make backups, make sure you have alternative ways of getting your work done. IT issues are not an excuse for late submission. In particular, upload your submission early. You can always upload a new version.

Good luck!

5 Change Log:

5.1 V1.0, 2018-08-11

Initial release.

5.2 V1.01, 2018-08-13

Loosened up the requirements for compiling and running the program.

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:

Given Name(s):

Student ID:

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:

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