CE866: Computer Vision
Assignment

Adrian F. Clark
alien@essex.ac.uk

Contents

What you should be able to do by when

week of term	what you be able to do
2	read in and display images
4	segment the map from the background
6	segment the red pointer from the map
8	work out the bearing from the pointer shape

Submission

Remember to identify your work with only your registration number, not your name. FASER allows you to upload your work as often as you like, so do keep uploading your program as you develop it.

formative feedback deadline	Friday 18th February at 11:59:59 (just before midday)
submission deadline	Tuesday 22nd March at 11:59:59 (just before midday)
what to submit	only your program (**not** the images!)
submission format	Python code only or a zip-file
marks returned	start of the summer term
marking criteria	see Section 5

Please submit your code as Python (if everything is in one file) or as a zip-file. Do not use other archive formats.

Preface

This assignment is a replacement for the examination that this module would otherwise need to have, so you should expect it to be a little challenging. What you have to do is described in some detail below but the way to solve the problem is left up to you to work out... though the lab exercises and lectures give you all the background knowledge required. Getting you to work out how to solve the problem is deliberate because it mirrors the way that tasks are often distributed to graduates in industry and I want to prepare you for it.

1 The task

A museum is developing a virtual reality (VR) reconstruction of a region of London, showing how it appeared in about 1900 — see Figure 1. They want visitors to be able to place a pointer on a map of London and use the position and orientation of the pointer to determine the location and direction of the view in the reconstruction, as shown in Figure 2. You might be interested to learn that this reconstruction can be experienced in my research lab, which has a whole-wall (4.2×2.5 m) stereoscopic display, using interaction modalities that include gestures and a bicycle!

Figure 1: VR reconstruction of Paternoster Row in London

The map to be used is Horwood's hand-drawn one from about 1792. It is actually two of his maps joined together; pretty accurate for something hand-drawn that long ago.

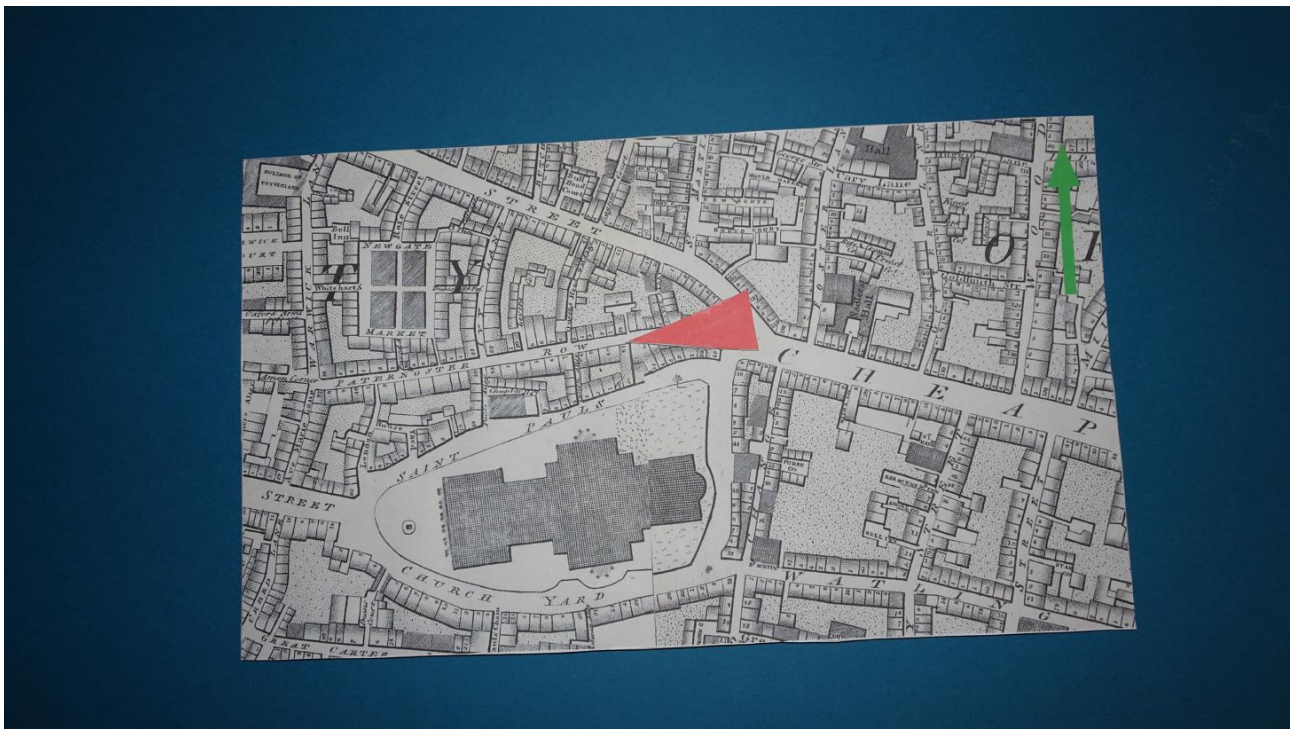

Figure 2: Horwood's map with blue background and red pointer; the view in Figure 1 corresponds to the location and direction of the pointer

In the museum, a camera is arranged above a dark blue table, looking directly downwards. The idea is that visitors put their copy of Horwood's map roughly in the middle of the table and then place the red pointer on it. The people who are developing the exhibit clearly designed things with care to make the computer vision task easier: the lighting illuminates the table fairly uniformly, the light-coloured paper on which the map is printed is a good contrast to the dark background and the red pointer is a very different colour from the paper. The map has a green arrow showing the direction of north. The tip of the red pointer determines the viewpoint and the direction in which it is pointing determines the view direction; you have to print these out from your solution as detailed in Section 2.

The general approach to the assignment should be fairly clear:

1. Segment the map from the blue background in the original image and extract it into a separate image in a way that makes the map edges match those of the extracted image.

2. Locate the green arrow and if it is not top-right of the map, rotate it through 180 degrees.

3. Segment the red pointer.

4. Locate the tip of the pointer to determine the location.

5. Determine the orientation of the pointer, convert it to a bearing and output it.

My own solution is about 150 lines of commented Python code and took about half a day to develop. Of course, I know OpenCV pretty well and knew from experience almost exactly how the program needed to work. You will take longer to develop your own solution.

All of the computer vision parts of the assignment are done using OpenCV calls so the program's run-time is well under a second. You should be able to make a start to the assignment as soon as you have done the first couple of laboratory experiments, and be able to do pretty much all of it after experiment 3. To do one step of the overall task, it might help if you review the chapter on *Vision in a 3D world* in the lecture notes.

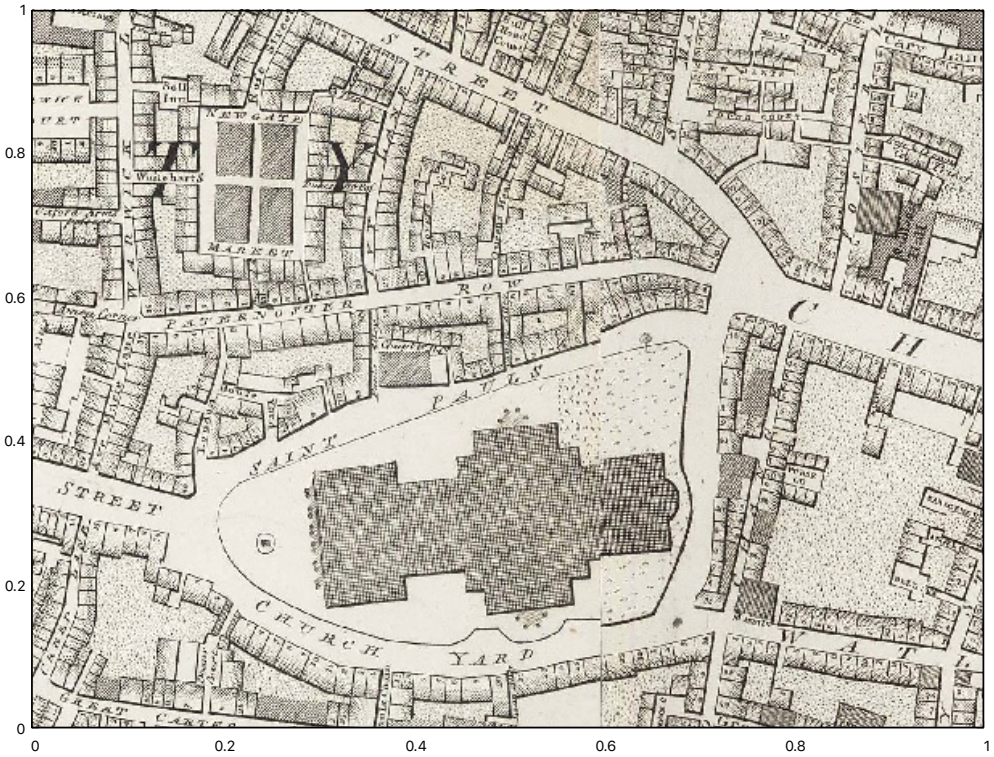

Figure 3: The coordinate system for the map

2 Operation and evaluation

Your solution should be written in Python. You are free to use OpenCV and/or numpy functionality in your solution, which must run on CSEE's Horizon server under Linux or in CSEE's Software Labs, also under Linux. You should not use any modules that are not already installed on these systems.

Your program must accept precisely one argument from the command line, the name of the image to be processed, as in:

```
python3 mapreader.py develop/develop-001.png
```

It should output two lines in the following format:

```
POSITION 0.673 0.212
BEARING 316.4
```

Any other output it generates is ignored. Your submitted program must not display any images. A template version of `mapreader.py` is supplied, which does nothing more than handle the command line and print output in the required format; you are advised to use this as a starting point for your own solution.

When the map is oriented correctly with north pointing upwards, the two numbers following `POSITION` represent the location of the point of the red pointer, being respectively the distance along the bottom of the map and the distance up its side, with the origin at the bottom left-hand (south west) corner. These numbers should both be in the range 0–1, as illustrated in Figure 3. The number following `BEARING` should be the angle in which the red pointer is pointing, given in degrees measured clockwise from north.

In accordance with good testing in computer vision, the imagery I shall test your program with is different from that you are using to develop it. It does however have identical characteristics, so if your program works on the supplied imagery it should also work on my unseen test imagery.

The reason for being so prescriptive about the way your program is executed and the appearance of its output is because its functionality will be checked using a *test harness*, a piece of software that runs it and analyses its output. You will lose marks if your submission doesn't work with the harness. There is nothing secret about the test harness, it is included as part of the assignment's zip file. You run it as follows:

```
python3 harness.py
```

If your program is called anything other than `mapreader.py`, you have to give its name on the command line when you run `harness.py`.

3 Incorporating others' work

Almost everyone looks for inspiration when writing software: a web search is very quick and easy to do and can often help with something that has you stumped. This is absolutely fine and you are encouraged to do it — but do be careful how you use the information you find to avoid being accused of cheating. This is perhaps best illustrated by an example.

Let's say you cannot remember how to work out the roots of the quadratic $ax^2 + bx + c = 0$ and you do a web search for:

```
python roots of a quadratic equation
```

One of the top links gives you the two lines of Python you need. If you incorporate this directly into your code, *you must acknowledge the source* as shown below:

```
# The following two lines are taken from
# https://www.w3resource.com/python-exercises/math/python-math-exercise-30.php
x1 = (((-b) + sqrt(r))/(2*a))
x2 = (((-b) - sqrt(r))/(2*a))
```

Studying the code you might see that, at least in Python 2, this could give the wrong answer if `a`, `b` and `c` were all integers; and there are far too many parentheses. If you take that code but correct it and change it to fit into your algorithm, you should write in your program:

```
# The following two lines are adapted from
# https://www.w3resource.com/python-exercises/math/python-math-exercise-30.php
x1 = (-b + math.sqrt(r)) / 2.0 / a
x2 = (-b + math.sqrt(r)) / 2.0 / a
```

You don't lose any credit for doing so.

Another way to solve problems is to talk about them with your friends. Feel free to do this while thinking about algorithms and approaches... but stop when you reach the point of talking about code. To continue the above example, you might say to your friend that you need to intersect a line with a circle in your program. Your friend may then explain how the problem turns out to be a quadratic equation, a term you remember vaguely only from school. This is fine... but if she gives you code to put into your program, you are both guilty of cheating. (Incidentally, this equation is a remarkably poor numerical solution to a quadratic equation.)

Be warned that I am pretty rigorous in checking for both kinds of cheating, using a combination of tools such as `turnitin` and some of my own *and* identifying chunks of code in submissions that look similar as I work through them. In general, you lose no marks by incorporating modest amounts of external code (when done correctly), so cheating is a really, really stupid thing to do. People found guilty of cheating may well receive a mark of zero for the assignment — just think what that would do to your overall module mark.

4 Formative feedback

As the mark for your assignment is such a large proportion of the total module mark, we are going to trial an approach used elsewhere in the University to give you useful feedback on it as you develop it. There is a submission deadline mid-way through the term (see the first page for when it is) for you to make an initial submission of your solution. I will look at what you have submitted and suggest ways in which you can improve it, helping you eradicate any bugs I spot and suggesting ways of improving your solution. The intention is for me to get the feedback back to you quickly, in plenty of time for you to use it to improve your final submission. It does mean that the amount of feedback you'll receive on your final submission will be less, but in any case that comes too late to be of any use in subsequent assignments before your exams.

You are also welcome to ask for help during the laboratory sessions too — but please don't use them too much to work on the assignment as the experiments assessed in the progress tests. Note that there is a FAQ on the module Moodle site; please read that before asking for help as it will usually be much quicker.

5 Marking criteria

I shall assess your submissions to the assignment under four broad headings, *algorithms*, *style*, *results* and *presentation*. Each of these is marked out of 25, so they add up to give an overall percentage mark. What I look for under each of these headings is explained below. After that, the feedback from a pair of programs achieving substantially different marks is presented to help you gauge how to prepare your own submission.

Algorithms. Underlying every program is an algorithm or, more commonly, several algorithms. An algorithm is a series of steps that transform some input to an output. The questions I ask when marking submissions are:

- When there are several alternative algorithms, has the most suitable one been chosen? Are there comments that justify the choice?
- When an algorithm has had to be developed from scratch, does it do the job it is intended to do? Has it been implemented correctly?
- A program that runs in one second is better than an equivalent program that takes ten seconds to run, and that is rewarded under this heading. Conversely, attempting to squeeze every nanosecond of performance from a single line of code that is executed only once and takes little time to run is pointless and will be penalised. It is important to know *when* to improve performance as well as *how* and *where* to improve it.

Coding style. Although an algorithm is the core of a piece of code, the way in which it is implemented is important. When reviewing submissions, I ask the following questions:

- Is your code easy for someone to read? You might know that, say, a particular numpy (numerical Python) expression can be made to execute very quickly; but if it is hard to understand and not in the most time-critical part of a program, it is probably better written to be easier to understand, or at least well-explained in a comment.
- Does the way the program is structured help its maintenance? In the Real World[1], maintaining and extending existing code is much more common than writing code from scratch, so a good programmer will make sure there is enough explanation for someone coming to the program can maintain it.
- Does your program structure aid re-use of parts of the code? This could be by creating useful classes, modules or procedures that are general enough to be used elsewhere, for example. Similarly, do you use standard procedures rather than writing your own?

[1] This is where people work for a living and time costs money.

- Are you careful in your use of resources, especially imaged and large arrays? Programs that are wasteful in terms of memory usage are difficult to port to small computers or embedded platforms.
- Code that carefully arranges data to be organized so that a series of processing steps can be performed quickly and easily gains more credit than brute-force processing. It is not always straightforward to make code elegant in this way but it is easy to recognize it when you see it.

Clearly, there can be trade-offs between computing things several times and storing the results of calculations for subsequent use, and a good programmer will discuss this topic briefly in the code.

Results. This section catches all the other aspects of the program — principally whether it actually is a bug-free solution to the assignment, but also other factors such as whether or not the code provides useful output when it encounters problems. When marking submissions, I look for answers to the following questions:

- Is the program invoked as per the specification? Failure of a program to meet its specification is a problem in the Real World as it will often end up as a component of a larger system. By the same token, writing a program that does what you want rather than what was specified will yield a substantially lower mark than one that does meet the specification.
- Is the program buggy? All bugs that you can find should be exterminated before submitting your program.
- Does the program describe what should be produced from a specific input? How would a person receiving your program know that it works? The usual way is to include example inputs and the corresponding outputs in a block of comments at the top of the program. Python provides really cute doctest functionality, for example.
- Does the program produce helpful output if invoked incorrectly? If a program requires an argument on the command line, it should not crash but should instead output (on the *standard error* stream) a message that explains how the program should be used. Similarly, if the program reads files with specific names as part of its operation, you need to produce a useful message if they are not present.
- Are the results as expected? Does the program produce results that are consistent with what you expect? If not, *say so* in your program's comments as I can award you some credit for having spotted that your program is not giving the correct answers.

You might be interested to learn that some of my former students who work in the software development industry have to write a series of tests, with expected outputs, *before* writing the actual code.

Presentation. In the Real World, the software industry you may well encounter after you graduate, the presentation of code is more important than you might think. Many software companies establish guidelines for the presentation of source code, often called a *house style*, that developers and maintainers are expected to use. Maintenance tends to be a more expensive activity than development, so it is important that people reading someone else's code are able to understand quickly what it is doing and how. When marking submissions, questions I ask myself are:

- Is the code well commented? This means that the comments explain what the program is trying to do rather than interpreting individual lines of code in English. Well-commented programs usually have blocks of comments interspersed between sections of code, not a comment for each individual line.
- Is there a block of comments at the top of the program that describes its purpose and shows how it is to be used?
- Does the indentation of the program indicate its structure? Note that the indentation seen in an editor or IDE does not necessarily match what will be seen outside that editor.

- Does the program source code conform to the house style? For this assignment, you need to ensure that *no comment or line of code is longer than 80 characters*. The underlying reason for this is that humans find lines containing more than 60–80 characters difficult to read. If your statement is longer than 80 characters in length, end all lines but the last with a backslash character, which is Python's notation for continued logical lines. The only exception to this rule for comments containing long URLs, which can appear on a single line rather than being arbitrarily split into 80-column chunks.
- Are there blank lines between sections of code to aid the reader's understanding?
- Is there white space within lines to make them easier to read?

A good way to check that the code is OK in terms of line length is to use the program called `longlines` included in the assignment zip-file:

```
./longlines mapreader.py
```

to report the lines in `mapreader.py` with over 80 characters in them. Note that there is no trade-off between categories here: a program which is beautifully written but totally devoid of comments will not score well.

Examples To give you an idea of how programs are actually marked, here are two fictitious examples. The first is a bare pass: it works and produces correct results but omits all the niceties of good programming. The second is as close to a perfect submission as I think is possible to expect: it *does* include all the information a maintainer might reasonably want when seeing the code for the first time (or an academic when marking a submission). It is easy to read and coded in a way that encourages re-use.

- a program receiving 50%
- a program receiving 100%

In both cases, the document starts with cover sheet much like the one that you will receive for your own submission. That is followed by all the source code files of your submission that I marked, partly because I may refer to specific line numbers in my feedback and partly so that you can check there were no problems with FASER (which has happened occasionally in the past).