UNIVERSITY OF LIVERPOOL

Department of Electrical Engineering and Electronics

# ELEC230
# Robotic Systems - Assignment 4
# Basic Image Processing using OpenCV

| | |
|---|---|
| **Module** | ELEC230 |
| **Coursework name** | Assignment 4 |
| **Component weight** | 20% |
| **Semester** | 2 |
| **HE Level** | 5 |
| **Lab location** | • Coding and simulation on and off-campus using OpenCV 4.2.0 installed on Ubuntu 20.04 (shipped with ROS Noetic).<br>• You will usually be operating on a virtual machine using:<br>  • your own PC/laptop or<br>  • a lab or library PC<br>  • from your hard drive, M: drive or removable storage<br>• If there are difficulties with the above, please contact the lecturers for support.<br><br>This assignment is mainly based on work begun in the Week 9 Lab |
| **Work** | Individual |
| **Timetabled time** | 6+ hours (Labs), 2+ hours of lecture videos, up to 2 hours of support sessions (Weeks 10 & 11) |
| **Suggested private study** | ~15 hours (in addition to above) |
| **Assessment method** | Individual, formal, word-processed reports in the format indicated in the Marking Criteria, along with other files listed in "What to hand in". |
| **Submission format** | Online via CANVAS. |
| **Plagiarism / collusion** | Standard University penalties and procedures apply for plagiarism and collusion. |
| **Submission deadline** | **Monday 9th May 2022, 23:59** |
| **Late submission** | Standard University penalties apply |
| **Resit opportunity** | August resit period (if total module failed) |
| **Marking policy** | Marked and moderated independently |
| **Anonymous marking** | Yes |
| **Feedback** | Via comments on your CANVAS submission, online |
| **Learning outcomes** | ▪ (BH1) Understanding Linux<br>▪ (BH5) Understanding the basics of Image Processing |

# Marking Criteria

| Aspect | Marks | Indicative characteristics | |
|---|---|---|---|
| | | **Adequate / pass** (40%) | **Very good / Excellent** |
| **Presentation and structure** | 10% | ▪ Submission includes Report with cover page (title, abstract, academic integrity declaration), Contents page, a background, a discussion section including focused screenshots, and an Appendix with full-screen screenshots (see below)<br>▪ Submission includes original source code .cpp files and a complete CMakeLists.txt file, according to the instructions<br>▪ Comprehensible language; punctuation, grammar and spelling are accurate<br>▪ Equations are legible, numbered and presented correctly | ▪ Appropriate use of technical, mathematic and academic terminology and conventions where relevant<br>▪ Word processed with consistent formatting<br>▪ Pages and equations are numbered; figures and tables are numbered/captioned<br>▪ Clear section headings and sub-headings where relevant<br>▪ Correct cross-referencing (of figures, tables, equations) and citations where relevant |
| ***Background, Design, Method, Discussion** | 65% | ▪ Problem background is introduced clearly<br>▪ Code is original and relates only to the assignment objectives<br>▪ Code is tidy, efficient and easy to follow / understand<br>▪ Code is clearly laid out and appropriately commented<br>▪ Clear discussion, including an explanation of key procedures undertaken | ▪ Excellent understanding of the problem background is displayed<br>▪ Excellent understanding is shown in answers to discussion questions<br>▪ Code is elegant and "DRY[1]"<br>▪ Comments show excellent understanding of syntax & semantics<br>▪ Clear explanation of all segments of code in the comments<br>▪ Clear explanation of all the testing carried out<br>▪ Discussion of what worked and what did not (and what could be improved) |
| **Results** | 25% | ▪ Focused screenshots of each program's output are presented as part of the Discussion section. In the Appendix, the full desktop version of each screenshot is given, with relevant windows and the taskbar with date and time visible, as evidence of original work<br>▪ Screenshots and code demonstrate at least partially correct or complete operation, according to the instructions | ▪ Screenshots and code demonstrate successful, correct output for every task<br>▪ Each image processing technique is clearly explained with reference to theory<br>▪ The programs only satisfy the objectives of the Assignment, with no extra unnecessary functionality |

***Note:** although 'Design', 'Method' and 'Results' are mentioned in the first column, you should **not** use these as headings to structure your report; they're just some marking categories.*

[1] Don't Repeat Yourself

# ELEC 230 Assignment 4 (2021-2022) – 20% weighting

## Background

*Adapted from https://docs.opencv.org/4.2.0/d6/d6d/tutorial_mat_the_basic_image_container.html.*
***Please visit and read the rest of the above web-page for more information about Mat objects.***

We have lots of ways to acquire digital images from the real world: digital cameras, scanners, computed tomography and magnetic resonance imaging, to name a few. What we see are images. However, when transforming this to our digital devices, what we record are numerical values for each of the points of the image.

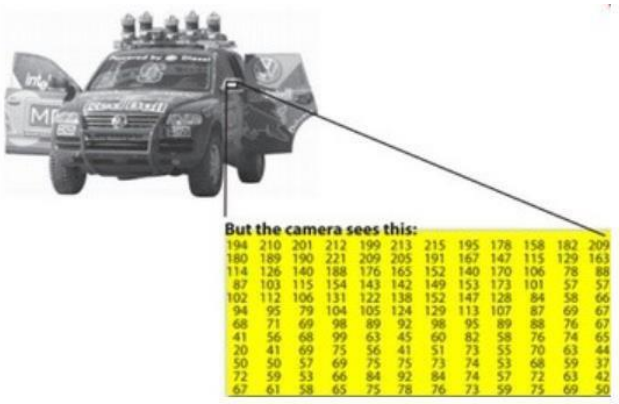


Figure 1 What the mobile robot camera sees (from OpenCV.org)

In Figure 1 you can see that the 'mirror' of the car is nothing more than a matrix containing all the intensity values of the pixel points. How we get and store the pixels values may vary according to our needs, but in the end all images inside a computer world are reduced to numerical matrices and other information describing the matrix itself. OpenCV is a computer vision library for processing and manipulating this information. The primary interface of OpenCV is in C++; it still retains a less comprehensive older C interface, so you may encounter elements of both languages in online tutorials.

## Aim of this assignment

OpenCV is like the 'image processing brain' of a modern mobile robot (although it's more than this.) This assignment aims to give you an understanding of basic image processing with OpenCV, laying some of the foundations for you to begin to understand computer vision for a mobile robot. This assignment does not ask you to interface OpenCV with ROS (connecting the robot's vision system with the rest of its brain) although this would be the logical next step and you are encouraged to explore this in your own time.

In this assignment you will generate OpenCV programs which can perform image / video processing algorithms on an image and video provided on Canvas. In Part A, the ultimate aim is to extract a door feature and similar features. In Part B, you are performing background subtraction in order to detect moving people.

## Hints & Tips

Your code must be executable in a Linux x86-64 environment without any modification. For more information on installing and working with OpenCV 4.2.0 with appropriate dependencies on Ubuntu 20.04, please have a look at the Image Processing lab script available on Canvas[2].

Before attempting the Assignment, follow all the steps in the Image Processing lab script on Canvas including working through the 7 tutorial pages recommended near the end of the script. After this, you are free to locate any **OpenCV 4.2.0 Tutorials** pages that are relevant to this assignment. **You are expected and encouraged to copy and adapt your code from the code shared on the OpenCV 4.2.0 Tutorials pages! :)** Also, go over the Image Processing lecture notes / videos, making sure you understand the image processing algorithms taught there, to help your discussion. You can also consult the relevant part of Chapter 4 of the **recommended text** for additional information.[3]

## Part A: Smooth, edge detect, feature extract (85%)

1. For your first task you are going to build a number of basic standalone image-processing programs using OpenCV and the Text Editor, where each program does the following (replace "YI" with your own 'last-letter initials'[4]):
    - Loads & displays an image '**door.jpeg**' (see Canvas) from the working directory
    - Also displays an output image, as follows:
        a) Program **YI_G** performs a Gaussian smoothing filter (4)
        b) Program **YI_S** performs a Sobel edge detection filter (4)
        c) Program **YI_C** performs a Canny edge detection filter (4)
        d) Program **YI_H** performs a Standard Hough Line transform to extract straight edges (4)
    - The bracketed ( ) maximum marks above assume the following:
        ✓ YI_S, YI_C and YI_H should incorporate conversion to grayscale, and Gaussian smoothing, before edge detection
        ✓ YI_H should use Canny edge detection before finding straight edges

[2] The work should be possible with earlier versions back to OpenCV 2.0 (contact Dave if unsure).
[3] Siegwart et al (2011) Introduction to Autonomous Mobile Robots, 2nd ed. Select UoL and enter MWS credentials to read the e-Book via this link. You can also find the e-Book yourself in the Library catalogue (https://library.liv.ac.uk/). There are options to download individual chapters as PDFs.
[4] **Y = last letter of your first name, I = last letter of your surname – this preserves anonymity**

- ✓ In any 'Canny' function arguments (YI_C and YI_H), the high threshold should be set to be automatically three times the lower threshold
- ✓ Each program, should show both the original and final image until the user presses any key while a display window is selected (which should terminate the program)
- ✓ All relevant filters should initially (and in the submitted .cpp files) use a 3x3 mask
- ✓ **No other functionality which is unnecessary for this assignment is included**

2. Importantly, the code in each program's .cpp file should be commented extensively BY YOU (delete the default comments present on the OpenCV code, and replace with your own, explaining each significant segment of code. (8)

3. Demonstrate that each program in 1 (a) to (d) works by running it and taking focused screenshots of the original and final images. Include these in your report, with a brief explanation about the theory of each filter/transform (18).

4. Continue your report for Part A, considering the following questions. Discuss each question with the help of reading and experimentation *(i.e. playing with the code!)* and support your discussion with focused screenshots of the associated outputs:
    a) What effect, and why, does increasing the kernel size have on the Gaussian smoothing filter (assume the kernel is a square)? (3)
    b) What effect does entering 0.2, instead of -1, for the standard deviations in x and y, have on the Gaussian smoothing filter when the kernel size is 9x9? Explain the effect. (5)
    c) With a smoothing kernel size of 7x7, what difference does it make if you change the smoothing filter used in the Sobel edge detector from Gaussian smoothing to a normalized box filter (simple average filter)? Why is this? Which do you think is better and why? (4)
    d) At the moment, YI_C is probably detecting far too many edges. Change the smoothing filter kernel to 7x7, and adjust the hysteresis minimum threshold in the Canny algorithm from 0 to 100 in steps of 25. Document & explain the differences you see. (4)
    e) What difference does it make if you effectively remove the smoothing stage from Canny edge detection by setting the smoothing kernel size to 1x1? Explain what you observe. (5)
    f) Choose an 'optimal' Canny threshold from 4(d) and compare the Sobel and Canny edge detection algorithms (both with 7x7 smoothing filter kernel). Explain what difference(s) you see and why they may occur. (4)

g) For the Hough transform program, adjust the smoothing kernel size (assume it's square) and the Canny 'minimum threshold' value, to try and obtain a better extraction of the door-frame. Comment on the quality of the results, including the relevance of your efforts to mobile robotics. Discuss how this relates to (e.g.) the need for dynamic thresholding. (10)

## Part B: Some fun video background subtraction (15%)

1. Your second task is to copy, save and build **this video background subtraction program** on OpenCV.org, which uses the BackgroundSubtractor class. Initially save and build the copied program "as it is" without any alterations. You do not need to replace the tutorial comments with your own, for this program only. Name the program **YI_backsub** (replace YI with your own last-letter initials as described in Footnote 4 above on p.4). This program does the following:
   - Loads a video
   - Plays it
   - Performs a background subtraction filter in another window

   Edit and re-make the code so that the program uses the "K-nearest neighbours" (KNN) algorithm and is applied to "**walking2.mp4**" found on Canvas. (2)

2. Verify that the program works, by running it and taking a full-screen screenshot of the functionality showing the original video side by side with the subtracted background video. Include this in your report (no explanation needed.) (3)

3. Continue your report for Part B, considering the following questions:
   - Why do you think the background-subtracted moving people sometimes look semi-transparent? (3)
   - Suggest a robotics application that background subtraction might be useful for, and explain your answer carefully. (5)

*The marks distributions above in brackets are an approximate guide and are subject to reasonable, slight adjustments at the point of grading. The bracketed marks add up to 90 and cover 'Background, Design, Method, Discussion' and 'Results' in the appropriate proportions. 10 marks are left over for 'Presentation and Structure' as described in the Marking Criteria.*

## The Rules

1. Your code should be tidy and well-commented. 'Tidy' means indentations are correct, etc.[5] Write your own comments so that the purpose of each distinct part of the code is explained (see the example on Canvas, mentioned at the end of this document). All coding comments must be in your own words.
2. While you are expected and encouraged to adapt code from the OpenCV.org tutorial pages, it will harm your marks if you include unnecessary copied code that does not relate to the assignment instructions. Similarly, **include only headers that are needed to satisfy the objectives**. (Do not include <opencv2/opencv.hpp>, for example.) In Part A, you must not include any third-party coding comments, e.g. comments in example code found on the OpenCV.org tutorial pages; write your own comments in your own words.
3. This goes without saying: you should do this work on your own. All the work you turn in should be yours, and not done in collaboration with anyone else (or copied from them, with or without their knowledge).
    - If you use any external sources of coding inspiration, other than OpenCV 4.2.0 Tutorials,then let us know in a README file.
    - Any and all sources of inspiration for your understanding and explanations of the theory and operations of the various algorithms should be credited with a reference (and placed within quotation marks if quoted verbatim.)

[5] Although you are not coding in the CLion IDE for this assignment, you can make your code tidier by copying and pasting it there, using the Reformat Code option, then copying and pasting it back to your Text Editor file.

## What to Hand In

Hand in everything that someone else needs to build and run your programs on Linux / OpenCV 4.2.0. For this assignment, this means:

- Your source code (.cpp files), well-commented **in your own words**
- CMakeLists.txt

You should also submit a Word document including a 1-page cover sheet (with title, Abstract and academic integrity declaration), a Contents page, section headings and page numbers etc. (see Marking Criteria.) This document should also incorporate:

- Background (around one paragraph)
- Discussion of no more than 7 pages (suggest 6 pages for Part A), Calibri font size 11, single-spaced, 'normal' margins, including:
    - Part A q.3: Demonstration of the functionality of your programs supported by:
        - **focused** screenshots of initial output (see Marking Criteria)[6]
        - some explanation of the theory of the image processing algorithms used (see lectures and the recommended text)
    - Part B q.2: Full-screen side-by-side screenshot as described in B q.2
    - Each of the above should be followed by a discussion of the questions in the respective Part (for Part A, this is q.4, and for Part B, q.3), backed up by more **focused** output screenshots.
- Appendix with all the full-screen versions of your focused screenshots with full desktop showing date/time (see Marking Criteria). These do not have to be numbered. The Appendix is not included in the 7-page limit above.
- Do **NOT** copy and paste your entire .cpp code in this document!

Therefore, if you have completed everything you should be submitting 1 Word document, 5 .cpp files and a CMakeLists.txt file. **Submit 7 individual files, not a zip file.** You will only be graded on the files you submit, and points will be deducted if you break the submission instructions.

Do *not* hand in executable files. **Each program should run after copying your files to the working directory and typing the following commands in a terminal:**

```
$ cmake .
$ make name
$ ./name
```

…where "name" is replaced by the name of the program.

[6] Present your focused screenshots as numbered figures, referred to in your discussion.

## Helpful Further Resources

Please make sure to start by following the Image Processing lecture videos/PDFs and the Week 9 lab script, as this will give you the background you need to carry out this assignment. If you have any questions, where a requirement is unclear or there are some concepts for which you need additional information, you can use the Canvas "Assignment 4" discussion board (DB) to ask questions.

Helpful insights into how OpenCV works (including the tutorials and other resources too such as descriptions of OpenCV modules (functions)) can be found at OpenCV.org.

If you read more widely to develop your understanding or support your arguments, make sure to attribute such resources with references.

*For further interest and development, see the tutorials on the wiki.ROS.org pages about interfacing OpenCV with ROS.*

## Space problems?

You can make your focused screenshots quite small if you wish to for reasons of space, and tile several together in a single figure (e.g. Figure 1(a), (b), etc.) Just make sure that at normal size (100% zoom) we can see enough detail to confirm the functionality and/or what you are discussing or demonstrating, etc.

## Example of (overly) well-commented code

Please see Canvas for an example of very well-commented code; you should aim to emulate this in your .cpp files, if to a slightly less extreme extent.