376.054 Machine Vision and Cognitive Robotics (VU 4,0) 2018W

Exercise 1: Canny Edge Detector

The goal of this exercise is to familiarise yourself with edge detection. You will implement the *Canny edge detector*, one of the most common edge detectors. A code construct and an example image is provided in a zip file which can be downloaded here:

Download Files For Exercise 1

Use these files as templates for your program and refer to the comments in the files for details on expected inputs and outputs.

Part 1: Implementation (10 Points)

You should write code so that it can be understood by someone else (the tutors, for example). Comment sections and lines which might not be easy to understand. If you've thought about a line or section for more than a couple of minutes, it probably warrants a comment. Use descriptive variable names.

Your code should not take too long to run. For the first few exercises, anything more than a few seconds likely means you are doing something very inefficient. Points may deducted for slow code, even if your solution is correct.

Complete the function skeletons provided, implementing edge detection on a grayscale image using the Canny edge detector.

The algorithm consists of 4 main parts:

1. Image blurring (1 point)

Blur the image with a Gauss-filter. Use the Matlab commands `fspecial` and `imfilter` (with option `replicate`). The width of the square filter kernel should be calculated as follows:

```
kernel_width = 2 * round(3 * smooth_sigma) + 1
```

Complete the code in **blur_gauss.m**.

2. Edge detection (2 points)

Calculate the horizontal and vertical gradient of the image using a Sobel filter, transposing it as appropriate. Again, use the Matlab commands `fspecial` and `imfilter`.

You may find `circle.jpg` useful for testing whether your gradient and orientation results are what you expect.

Complete the code in **sobel.m**.

3. Non-maximum suppression (4 points)

Implement the non-maximum suppression algorithm along the orientation of the gradients to thin out the edges. You may wish to quantise the gradient orientations to discrete levels like "horizontal", "vertical", "diagonal-left", "diagonal-right" first and then treat each case separately. E.g. if a pixel has a "horizontal" gradient orientation, its gradient value should only be kept if its immediate left and right neighbor pixels have a lower gradient value than the center pixel.

Complete the code in **non_max.m**. Make sure you explain your approach in comments.

4. Hysteresis thresholding (1 points)

Implement hysteresis thresholding to receive connected edge segments. The Matlab commands `find` and `bwselect` will reduce your programming effort. To make parametrisation easier you should normalize the gradient values to lie within the interval [0, 1] before applying the hysteresis thresholding.

Complete the code in **hyst_thresh.m**.

5. Automatic hysteresis threshold selection (2 points)

a bar graph where each bar represents how many values in the input fall into a certain range. You can generate a histogram with the `histogram` or `histcounts` commands in Matlab.

Using the histogram, you should implement a simple automatic method for selecting the hysteresis thresholds. We will assume that some proportion of edge pixels should be above the high threshold, and some proportion above the low threshold. The cumsum function will be useful. Your computation should be precise - don't just use the bin centre or edge for the thresholds. Assume that values which fall in one bin are uniformly distributed within it: a bin with 100 points in it will have 20 points in the first fifth of its range, 40 within the first two fifths, and so on.

You should use a 50-bin histogram for a reasonable degree of granularity. You should also exclude all zero values, since they are of no interest to us.

Complete the code in **hyst_thresh_auto.m**

Forbidden and allowed Matlab commands

Matlab command `edge` must not be used. However, the commands `imfilter`, `conv` and `conv2`, along with other block-processing and sliding-window functions may be used, if needed.

Unless explicitly stated, you should not use functions which solve an exercise, or a significant part of it, in a single line of code. If you are unsure of whether you are allowed to use a function, please contact the tutors.

Advice

Since Matlab is a matrix-oriented programming language, "vectorised" processing is much more efficient than loops. Pictures should be converted to floating point format before further processing; convert your picture to a double grayscale image (function `rgb2gray`). Also make sure that your starting image only contains RGB values in the uint8 range from 0 to 255. Keep your programs as simple as possible. Catching wrong user inputs and printing error messages is not necessary.

If your code is slow, the "Run and Time" button in the MATLAB editor tab allows you to profile the code and see which lines are the most time consuming.

Useful commands

`imfilter` - Filtering with linear filter kernel
`fspecial` - Generation of filter kernels
`imread` - Load an image
`imshow` - Display an image (for RGB image, value format is uint8 with range [0, 255], for grayscale, value format is double with range [0, 1])
`rgb2gray` - Image conversion to grayscale
`double` - Converts to double-precision floating point format
`uint8` - Converts to 8-bit unsigned integer format
`imresize` - Resizes an image
`edge` - Edge detection
`bwselect` - Selects objects in binary images
`histogram` - Histogram plot
`max` - Finds maximum

Part 2: Documentation (6 Points)

Perform the following experiments and answer the questions in a few sentences. Only change the parameters in the file main_ex1.m in the marked places. The `imrow` function provided in the exercise files may be useful for supporting your answers.

- Answer each numbered question separately, with the answer labelled according to the experiment and question number. For example, experiment 1 question 2 should be numbered 1.2. You do not need to include the question text.
- Include images to support your answers

Maximum 1500 words, arbitrary number of pages for result images (referenced in the text). Violating the text limit will lead to point deductions!

Experiment 1 (2 points)

Investigate the effect of `sigma`

1. Why is the kernel width defined as a function of sigma? Try various width and sigma inputs for the `fspecial` function and examine the results. Does an incorrect width have an effect in practice?
2. What effect does blurring have on pixel intensities in the image? Use `imrow` here.
3. What is its effect on the edges you receive after non-max suppression?
4. How does it affect the edges you are left with after the hysteresis thresholding, assuming thresholds are kept constant?

Experiment 2 (2 points)

Investigate the effect of thresholds used during hysteresis thresholding.

1. Change the parameters `thresh_low` and `thresh_high`. What effects do these values have on the result?

3. Does the automatic selection help with getting reasonable edges across various images? Why?
4. Is there a situation where the selection method you implemented doesn't work well? How might one mitigate this issue?

Experiment 3 (1 points)

Investigate the effect of noise on the results. You can add noise to an image with the function `imnoise`. Add gaussian noise with zero mean and try variances in the range [0.001, 0.25].

1. What effect does the noise have on the resulting edges?
2. Is it possible to set the parameters to get reasonable edges? How does changing `sigma` and the hysteresis threshold values affect the results?

Experiment 4 (1 points)

The *Marr-Hildreth Operator*, also called *Laplacian of Gaussian Operator*, uses the second derivative of the gradients instead of the first to localize the edges. Compare this operator - Matlab function `edge(img, 'log')` - with your implementation of the Canny algorithm. What differences can you see?

Assistance

Please keep to the following hierarchy whenever there are questions or problems:

1. Forum: Use the TUWEL forum to discuss your problems.
2. Email for questions or in-person help requests: machinevision@acin.tuwien.ac.at

Upload

Please upload the Matlab files **main_ex1.m**, **blur_gauss.m**, **non_max.m**, **hyst_thresh.m**, **hyst_thresh_auto.m**, **sobel.m** and your documentation (pdf-format) **as one ZIP-file** via TUWEL.

Submission status

Submission status	Submitted for grading
Grading status	Released
Due date	Monday, 22 October 2018, 8:00 AM
Time remaining	Assignment was submitted 13 hours 18 mins early
Last modified	Sunday, 21 October 2018, 6:41 PM
File submissions	MinhNhatVu_Exercises_submit.zip 21 October 2018, 6:41 PM
Submission comments	Comments (0)

Edit submission

You can still make changes to your submission

Feedback

Grade	12.0 / 16.0
Graded on	Monday, 5 November 2018, 1:01 PM

Feedback comments

Protocol:

1.2) (-0.5 points) This is not correct. Blurring doesn't increase or decrease the pixel intensities directly. A pixel becomes ...

◀ Vision-based Autonomous Landing in C… Jump to... Files for Exercise 1 ▶

My courses
All my courses
Course overview
Search courses
New course
Help
TUWEL Tutorials (Students)
TUWEL Tutorials (Teachers)
TUWEL scenarios
E-Mail Ticket Support
General Information
Terms of Use
Teaching Support Center