School of Computer Science and informatics
Coursework Assessment Proforma

Module & Lecturer: CM3113: Computer Vision, P. L. Rosin

Date Set: 19th October 2015

Submission Date: 19th November 2015

Submission Arrangements:
Your coursework program – a Java application program whose main method is in the file named `SquareHough.java` – should be submitted by logging into the coursework testbed at: `https://egeria.cs.cf.ac.uk/` before 9am on the submission date.

Do not include the image I/O code from my web page – this will be automatically linked when your program is compiled.

Make sure to include (as comments) your student number (but *not* your name) at the top of your program file (`SquareHough.java`), Also, follow this by any notes (as comments) regarding your submission. For instance, specify here if your program does not generate the proper output or does not do it in the correct manner.

Title: Computer Vision Coursework

This coursework is worth 30% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks. You are reminded of the need to comply with Cardiff University's Student Guide to Academic Integrity. Your work should be submitted using the official Coursework Submission Cover sheet.

INSTRUCTIONS

Write a Java application program whose main method is in the file named `SquareHough.java` that detects squares in an input image. The basic idea is to first convert the input image into a line map using the Difference of Gaussian filter, and then apply the Hough Transform to detect straight lines. Squares are identified as specific patterns in the Hough space. Further details are now given:

- The Difference of Gaussian filter (with $\sigma = \{1, 2\}$) is applied to the input image to find dark lines. Truncate negative responses of the DoG to zero.
- Use the $\rho\theta$ parameterisation of a line for the Hough Transform, and set its origin to be the centre of the image. The bin sizes are 1 pixel for ρ and 1° for θ. Set the range of θ to $[0°, 180°]$, and ρ can be both negative and positive.
- Use orientation to reduce the number of votes cast in the Hough space. Estimate local orientation θ_{xy} using gray level moments in local 7×7 windows of the DoG image. The votes should be cast in the range $\theta_{xy} \pm \Delta\theta$ and weighted according to their DoG response.
- For visualisation purposes only, find the peaks in Hough space using a 19×19 window and plot in an output image the lines whose accumulator values are greater than a factor f_1 of the maximum value in the Hough space.
- A square will be modelled by four points which specify how the four lines making up the square will appear in Hough space. There are three degrees of freedom for the square (assuming fixed size) – compute a response map over these parameters based on the four accumulator values. Rather than *add* the four accumulator values use the *minimum* value, as this will ensure that a shape consisting of only three strong lines is penalised, i.e. given a low score.
- Find peaks in this three dimensional parameter space using a $19 \times 19 \times 19$ window, and retain peaks whose response values are greater than a factor f_2 of the maximum value in the response map. Plot in red in an output image the squares corresponding to these peaks and overlay them on the input image.
- Validate these selected squares by performing back-projection on the DoG image. Retain squares whose back-projection values are greater than a factor f_3 of the maximum back-projection value of the initially selected squares. Add to the previous overlay image the retained squares, plotted in blue.
- As an alternative option to your program allow the DoG response to be replaced by the inverted Sobel edge magnitude and use the Sobel orientation.

`SquareHough.java` must take seven command line arguments (in the given order):

- filename of the input PGM image
- length of the target square
- $\Delta\theta$
- f_1
- f_2
- f_3
- L or E (to indicate lines (DoG) or edges (Sobel) to be used)

`SquareHough.java` outputs the following images with the following filenames:

- **DoG.pgm** or **Sobel.pgm** – the DoG response or Sobel edge magnitude
- **accumulator.pgm** – the rescaled Hough space
- **lines.ppm** – the input image overlaid with green lines detected by the Hough transform
- **squares.ppm** – the input image overlaid with initial red squares
- **backproject.ppm** – as above but also overlaid with blue squares which were validated by back-projection

See the last page for examples of applying `SquareHough.java`

Some test images are provided for you on Learning Central to help develop and test your program.

A testbed is provided at `https://egeria.cs.cf.ac.uk/` which will help you test your program by running it on sample input images and comparing the results to my own solution. To use the testbed just login and follow the instructions.

Apart from the Image I/O code that I provide (`Image.java`) you should write all the code yourself. E.g. do not use Java's built-in image processing classes.

NOTE: I will perform extensive tests on your program, using more than just the test data I have provided.

Submission Instructions

Description		Type	Name
Cover Sheet	**Compulsory**	One PDF (.pdf) file	[student number].pdf
Solution	**Compulsory**	One or more Java source files (.java)	SquareHough.java plus unrestricted

Note: both the cover sheet and the source code should be submitted by accessing `https://egeria.cs.cf.ac.uk/` as detailed in Submission Arrangements.

Criteria for Assessment

Credit will be awarded against the following criteria.

Your program must run on the School's computers – not just on your computer at home. To verify this, you should use the testbed at `https://egeria.cs.cf.ac.uk/` which will allow you to upload your program and a sample input image, and will then run both your program and mine, and compare their results. The testbed will also indicate roughly where and/or how much the results differ.

I highly recommend that you make extensive use of the testbed to check and refine your program – you can upload your program as many times as you like. The last uploaded version will be automatically taken as the version to be marked.

The breakdown of marks will be for correctly computing:

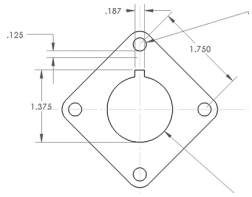
1. 5% – difference of Gaussian filtering (basic)
2. 5% – extra marks are given if DoG is implemented efficiently
3. 5% – estimate orientation using moments
4. 15% – generate Hough space
5. 20% – detect and plot lines
6. 25% – detect and plot squares
7. 15% – validate squares using back-projection
8. 10% – other (coding style, in-line comments)

Feedback on your performance will address each of these criteria.

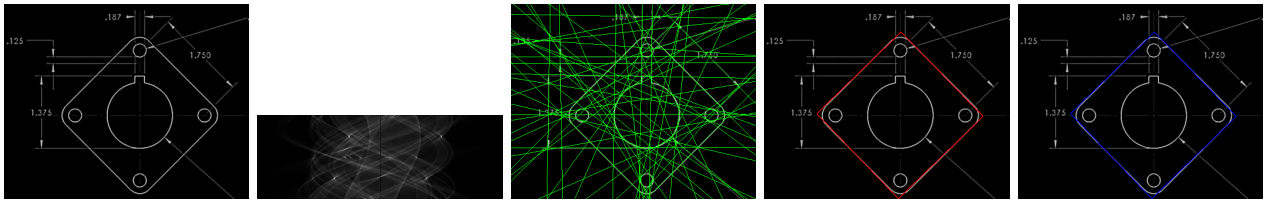
Further Details

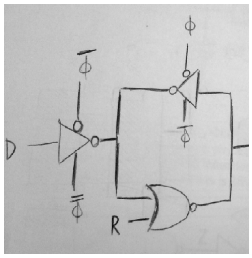
Feedback on your coursework will address the above criteria and will be returned in approximately three weeks. This will be supplemented with oral feedback in lectures and tutorials. If you have any questions relating to your individual solutions talk to the lecturer.

Sample Outputs

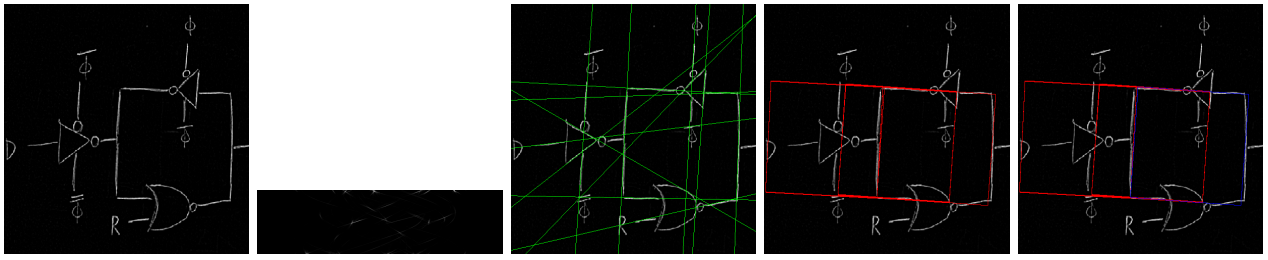


Given the input image on the left, my output of:
`% java SquareHough im1.pgm 200 90 0.5 0.5 0.75 L`
is (DoG.pgm accumulator.pgm, lines.ppm, squares.ppm, backproject.ppm):





Given the input image on the left, my output of:
`% java SquareHough im13.pgm 220 5 0.5 0.5 0.75 L`
is (DoG.pgm accumulator.pgm, lines.ppm, squares.ppm, backproject.ppm):



Given the input image on the left, my output of:
`% java SquareHough im50.pgm 124 5 0.25 0.5 0.75 E`
is (Sobel.pgm accumulator.pgm, lines.ppm, squares.ppm, backproject.ppm):