UNIVERSITY OF AUCKLAND
Waipapa Taumata Rau
NEW ZEALAND

CS373 Assignment - Sign Detection

2025 version 4

Tony Wang, Gibran Zazueta Cruz, Ruigeng Wang

Introduction

- In this assignment, the main component is to write a Python code pipeline to automatically detect all the road signs in the given images. We have provided a Python skeleton code package for you, so you just need to implement the necessary functions to make the pipeline working.
- It is a great way to enhance your understanding of the image processing.
- Through out the course, you will develop the following skills (your tools) to complete the pipeline for road sign detection.
  - RGB color image to greyscale conversion
  - Image contrast stretching
  - Edge detection
  - Image blurring
  - Image thresholding
  - Erosion and dilation operation
  - Connected Component Analysis
  - More…

Step 1: Convert to Greyscale and Normalize

1. Implement "`computeRGBToGreyscale()`" to convert a grey scale image: read input image using the '*readRGBImageToSeparatePixelArrays()*' helper function. Convert the RGB image to greyscale (use RGB channel ratio 0.3 x red, 0.6 x green, 0.1 x blue), and round the pixel values to the nearest integer value.

2. Implement "`scaleTo0And255And5_95Percentile()`" to stretch the image contrast: stretch the values between 0 to 255 (using the 5-95 percentile strategy) as described on lecture slides *ImagesAndHistograms*, p20-68). **Do not round your 5% and 95% cumulative histogram values.** Your output for this step should be the same as the image shown on Fig. 2.

Hint 1: see lecture slides *ImagesAndHistograms* and Coderunner Programming quiz in Week 10.
Hint 2: for our example image (Fig. 1), the 5_percentile (f_min) = 202 and the 95_percentile (f_max) = 217.

We will use this image ('sign_1') as an example on this slides

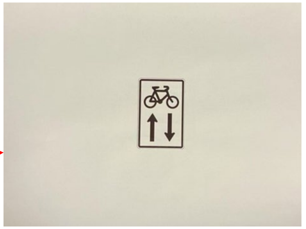


Fig. 1: input

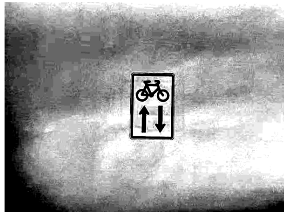


Fig. 2: Step 1 output

Step 2: Edge Detection

1. Implement "`computeVerticalEdgesSobelAbsolute(), computeHorizontalEdgesSobelAbsolute()`" to apply a 3x3 Sober filter in horizontal (x) and vertical (y) directions independently to get the edge maps, you should store the computed value for each individual pixel as Python *float*.

Hint 1: see lecture slides on edge detection and Coderunner Programming quiz in Week 11.
Hint 2: please use the 3x3 Sober filter shown below for this assignment:

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Vertical edge (g_x)

$$\frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Horizontal edge (g_y)

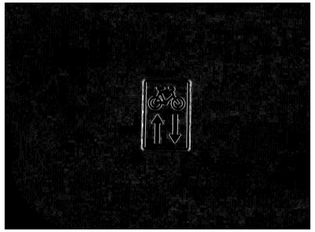


Fig. 3: Edge map (g_x) on horizontal direction

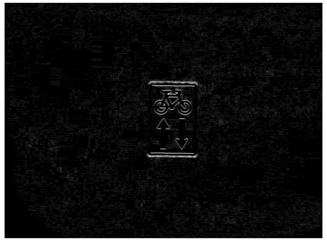


Fig. 4: Edge map (g_y) on vertical direction

Step 2: Edge Detection (Cont.)

2. Implement "`computeEdgesSobelAbsolute()`" to take the absolute value of the sum between horizontal (x) and vertical (y) direction edge maps. **Do not round the numbers in the intermediate step**. The output for this step should be similar to the image shown on Fig. 5.

Hint 3: you should use the *BorderIgnore* option and set border pixels to **zero** in output, as stated on the slide *Filtering*, p13.
Hint 4: for computing the edge strength in both x and y direction, you may use the following equation:

$$g_m(x,y) = |g_x(x,y)| + |g_y(x,y)|$$

Edge map on horizontal and vertical

Absolute grey level gradient on the horizontal direction

Absolute grey level gradient on the vertical direction

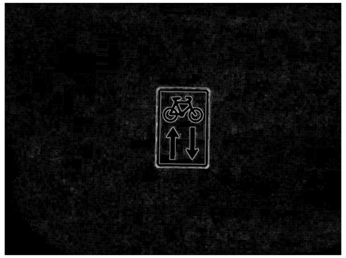
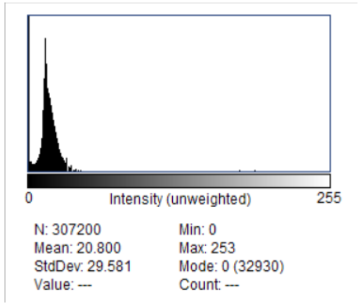


Fig. 5: Step 2 output (g_m)

Step 3: Image Blurring

Implement "`computeStandardDeviationImage7x7()`" to apply a 7x7 standard deviation filter to the image obtained from Step 2. Your output for this step should be similar to the image shown on Fig. 7.

Hint 1: **do not round your output values**.
Hint 2: after computing the standard deviation filter for one 7x7 window, you should take the absolute value of your result before moving to the next window.
Hint 3: you should use the *BorderIgnore* option and set border pixels to **zero** in output, as stated on the slide *Filtering*, p13.
Hint 4: see lecture slides on image filtering and Coderunner Programming quiz in Week 11.



```
N: 307200        Min: 0
Mean: 20.800     Max: 253
StdDev: 29.581   Mode: 0 (32930)
Value: ---       Count: ---
```

Fig. 6: An example of the grayscale histogram for output from step 3. Your generated histogram may not be exactly the same as above!

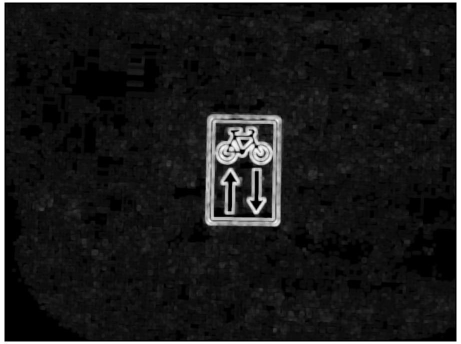
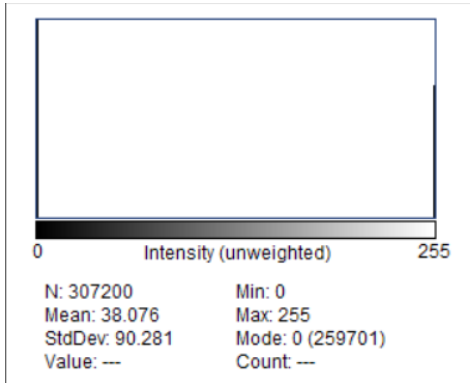


Fig. 7: Step 3 output

Step 4: Threshold the Image

Implement "`computeThresholdGE()`" to perform a simple thresholding operation to segment the sign(s) from the black background. After performing this step, you should have a binary image (see Fig. 10).

Hint 1: set the threshold value to 31 would give the best result for all the easy images we provided to you. Please set the threshold value to 31 to satisfy out auto-marking system.
Hint 2: set any pixel value smaller than 31 to 0, this represents your background (region 1) in the image, and set any pixel value bigger or equal to 31 to 255, which represents your foreground (region 2) – the sign.
Hint 3: see lecture slides on image segmentation (p7) and see Programming quiz on Coderunner on Week 10.



```
N: 307200        Min: 0
Mean: 38.076     Max: 255
StdDev: 90.281   Mode: 0 (259701)
Value: ---       Count: ---
```

Fig. 8: An example of the grayscale histogram for output from step 4

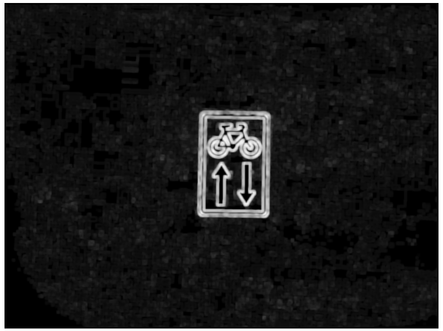


Fig. 9: Step 3 output

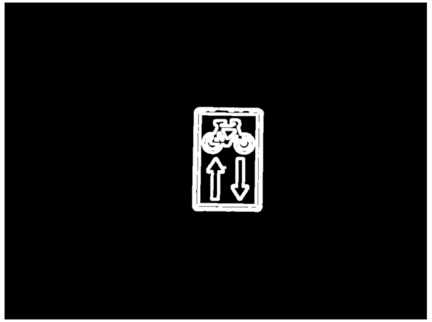


Fig. 10: Step 4 output

Step 5: Erosion and Dilation

Implement "`dilation() and erosion()`" to perform dilation and erosion. Your implemented function allow run dilation/erosion multiple times based on the function argument (*`num_of_iterations`*). Your output for this step should be similar to the image shown on Fig. 11.

Hint 1: use a 4x4 kernel (all elements equal to 1), see Fig. 12 for the kernel details.
Hint 2: the filtering process has to access pixels that are outside the input image. So, please use the *BoundaryZeroPadding* option, see lecture slides *Filtering*, p13.
Hint 3: see lecture slides on image morphology and Coderunner Programming quiz in Week 12.

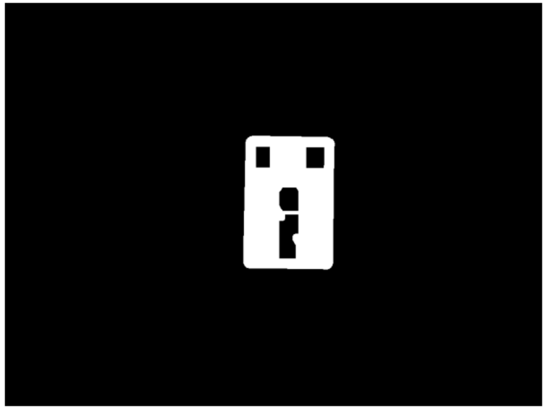


Fig. 11: Step 5 output

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

Fig. 12: 4x4 kernel for dilation and erosion

Step 6: Connected Component Analysis

Implement "`computeConnectedComponentLabeling()`" to perform a connected component analysis to find **all** connected components. Your output for this step should be similar to the image shown on Fig. 13.

After erosion and dilation, you may find there are still some holes in the binary image. That is fine, as long as it is one connected component.

Hint 1: see lecture slides on *Segmentation_II*, p4-6, and Coderunner Programming quiz in Week 12.

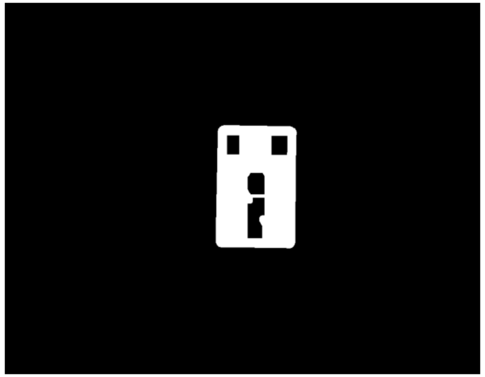


Fig. 13: Step 6 output

Step 6: Draw Bounding Box

Implement “`returnBBoxCoords()`” to extract the bounding box(es) around all regions that your pipeline has found by looping over the image and looking for the minimum and maximum x and y coordinates of the pixels in the previously determined connected components. Your output for this step should be the same as the image shown on Fig. 14.

Make sure you record the bounding box locations for each of the connected components your pipeline has found.

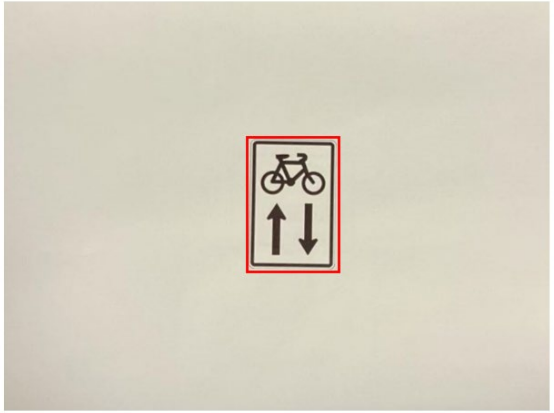


Fig. 14: Step 7 output

Step 6: Draw Bounding Box (Cont.)

We will provide code for drawing the bounding box(es) in the image, so please store all the bounding box locations in a Python list called ‘*bounding_box_list*’, so our program can loop through all the bounding boxes and draw them on the output image.

Below is an example of the *‘bounding_box_list’* for our example image on the right.

```
Bounding_box_list = [[451, 251, 623, 503]]
```

A list of list

Bounding_box_min_x

Bounding_box_min_y

Bounding_box_max_x

Bounding_box_max_y

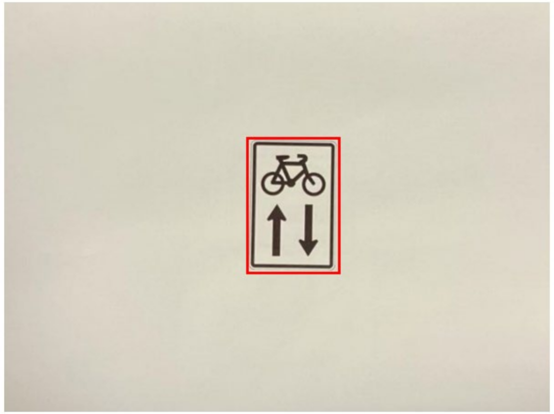
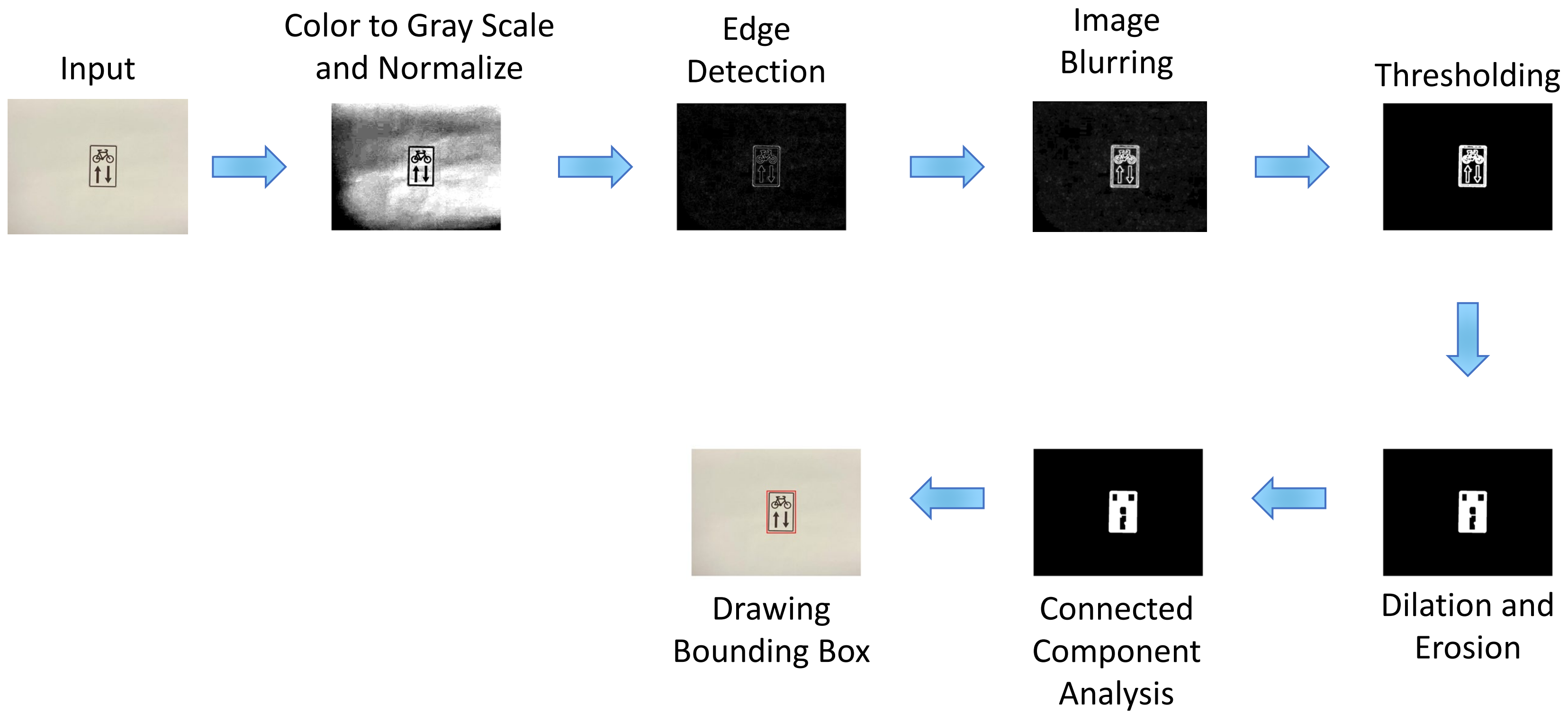


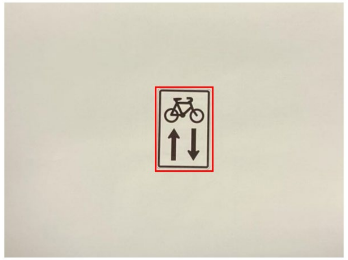
Fig. 14: Step 7 output

Sign Detection Full Pipeline

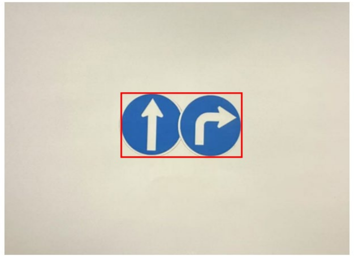


Assignment Sample Outputs

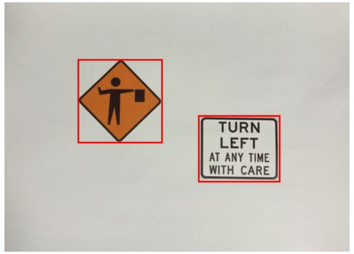
Important: treat as only one bounding box!



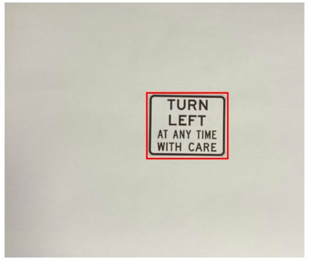
sign_1 final output



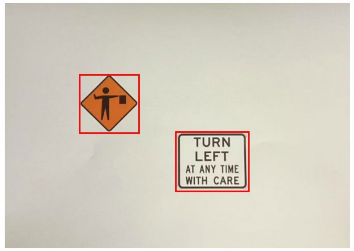
sign_2 final output



sign_3 final output



sign_4 final output



sign_5 final output

Important: Put the right bounding box first in the bounding box list!

Assignment Details

- First, join the GitHub classroom using the link below:

  https://classroom.github.com/a/Hr9UEGX0

- The main component of the assignment ("CS373_sign_detection.py") must not use any non-built-in Python packages (e.g., *PIL*, *OpenCV*, *NumPy*, etc.,) except for Matplotlib. Ensure your IDE hasn't added any of these packages to your imports.
- This is an individual assignment, so every student has to submit this assignment!
- We have provided you with some road sign images for testing your pipeline (you can find the images in the 'Images/easy' folder).
- This assignment is worth **15 marks**.
  - Your pipeline should be able to detect all the signs in the image labelled with easy-level (5 images). This will reward you with up to **10 marks**.
  - You will need to come up some extensions to award **5 marks**. For example, try images labelled as hard-level images in the "Images/hard" folder.
  - Write a short reflective report about your extension. (Using Latex/Word)

Assignment Details (Cont.)

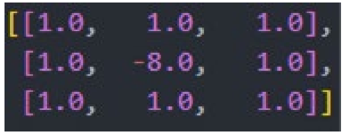
- Your code for executing the main sign detection algorithm has to be located in the provided "CS373_sign_detection.py" file!
- For the extensions, please create a new Python source file called 'CS373_sign_detection_extension.py'; this will ensure your extension part doesn't mix up with the main component of the assignment. Remember, your algorithm has to pass the main component first!

- Important: Use a lab computer to test if your code works on a different Windows machine (There are over 300 students, we cannot debug code for you if it doesn't work!)

Extensions

For this extension (worth 5 marks), you are expected to alter some parts of the pipeline. In extension part, you are allowed to use any python libraries to implement your ideas. There are some samples of extension ideas:

- Using other filter (e.g., Laplacian filter) for image edge detection
  - Please use the Laplacian filter kernel on the right (see Fig. 15).
  - You may need to change subsequent steps as well, if you decide to use Laplacian filter.
- Output number of signs your pipeline has detected.
- Testing your pipeline on the hard-level images we provided.
  - For some hard-level images, you may need to try different edge filter, threshold, image blurring and erosion/dilation values for sign detection. You need to look at the size of the connected components to decide which component is the sign.
- Identify the type of signs (whether it is a stop sign, “pass with care” sign and etc.).
  - Since different type of signs have different sizes, you may want to compute the area of the bounding box in the image to identify them.
- Using a deep learning approach to detect signs.
- etc.



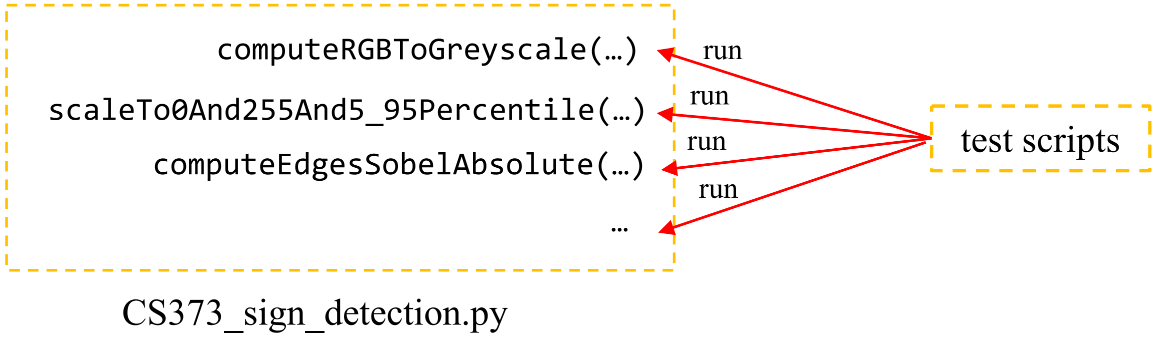
```
[[1.0,   1.0,   1.0],
 [1.0,  -8.0,   1.0],
 [1.0,   1.0,   1.0]]
```

Fig. 15: Laplacian filter kernel

Submissions that make the most impressive contributions will get full marks. Please create a new Python source file called ‘*CS373_sign_detection_extension.py*’ for your extension part, and include a short PDF report about your extension. The creative ideas worth 2 marks, the implementation worth 2 marks and the report worth 1 marks, Try to be creative!

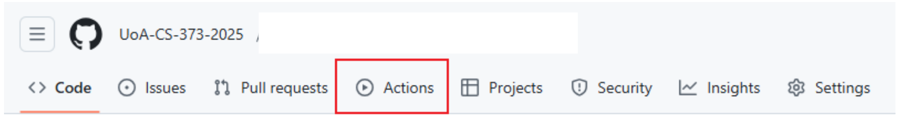
Auto-marking System

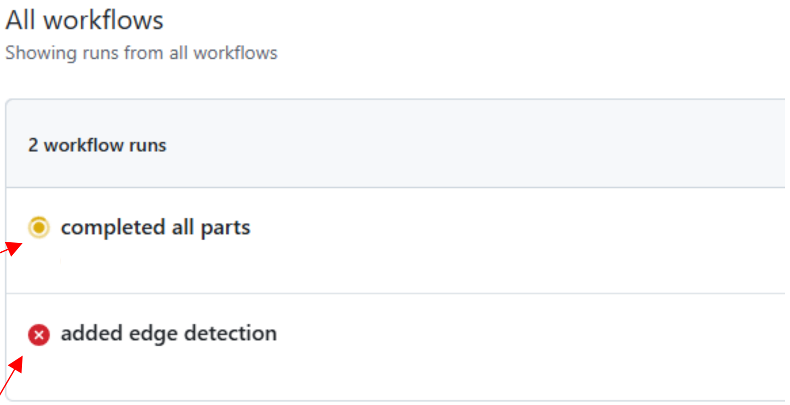
- We implemented an auto-marking system to give you a fast feedback, and also help you to debug your code.
- The auto-marking system is based on GitHub classroom auto-grading system. After each "git push" to your remote *master* branch repository, the GitHub Actions will automatically run the test scripts on your main component. We will mark the extension part manually (so not included in the auto-marking system).
- The test scripts will **only** run your implemented functions in "CS373_sign_detection.py" and compare with our results. Thus, it is essential to **not change the Python file name and the function name**, otherwise, the test cases may fail even you implementations are correct.



Assignment Submission and Grading

- All submissions have to be submitted using GitHub Classroom before **Wednesday 11th of June, 17:00 (5:00 pm)**.
- After you submit your code to the designated GitHub repository, and the auto-marking is completed, a result will show in your GitHub Action tab.




All workflows

Showing runs from all workflows

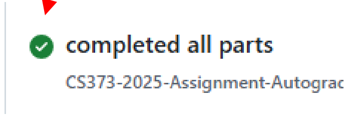
2 workflow runs

completed all parts

added edge detection

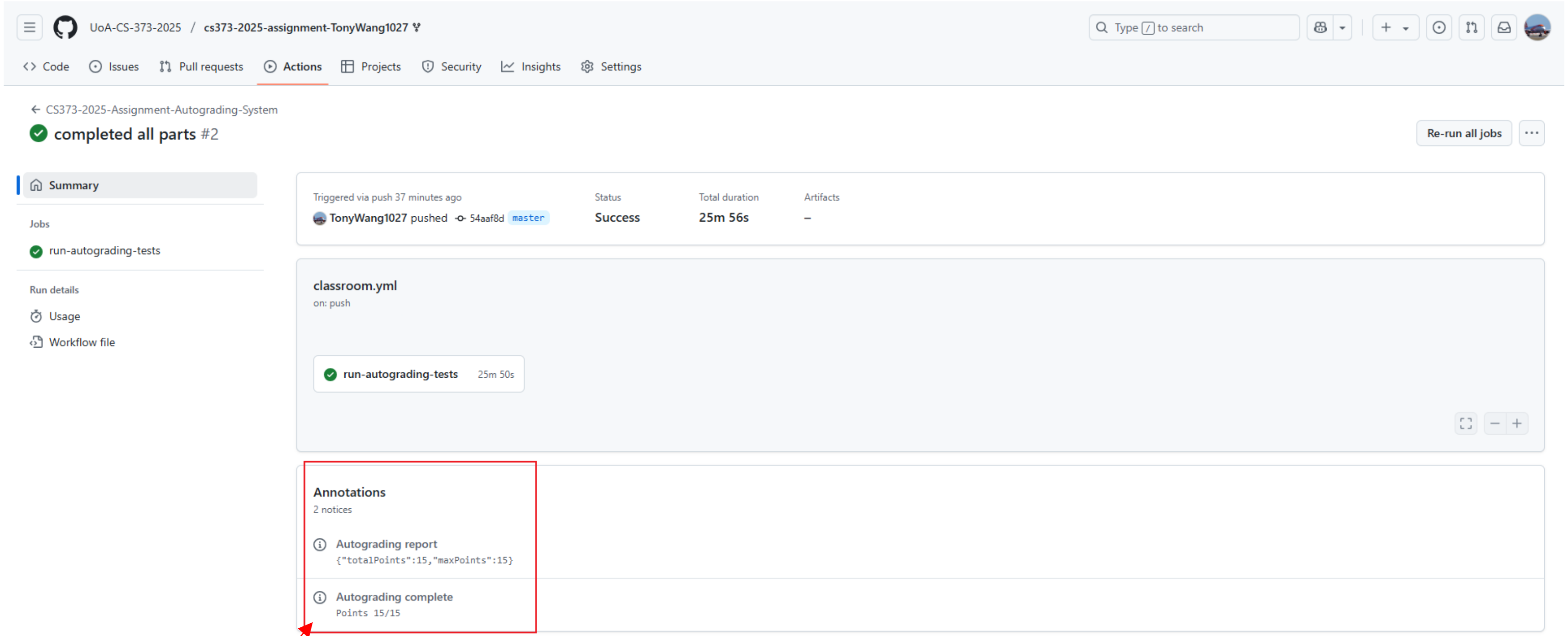
Green: well done! Full marks for the main component!

Yellow: Running the tests


completed all parts

CS373-2025-Assignment-Autograc

Red: Don’t mean your assignment is wrong, check the passed test cases!


UoA-CS-373-2025 / cs373-2025-assignment-TonyWang1027

Code | Issues | Pull requests | Actions | Projects | Security | Insights | Settings

← CS373-2025-Assignment-Autograding-System

completed all parts #2

Re-run all jobs

Summary

Jobs

run-autograding-tests

Run details

Usage

Workflow file

Triggered via push 37 minutes ago	Status	Total duration	Artifacts
TonyWang1027 pushed 54aaf8d master	Success	25m 56s	–

classroom.yml
on: push

run-autograding-tests 25m 50s

Annotations
2 notices

Autograding report
`{"totalPoints":15,"maxPoints":15}`

Autograding complete
`Points 15/15`

Points you awarded

Assignment Submission and Grading

- Grading for the main component:

1. The auto-marking system first checks your bounding box(es) by running 5 tests, one test per image. If your pipeline passed 3 out of 5 test cases, you will award 10 points (max marks for the main component). Max running time: 30 minutes.

2. If the above test cases failed, it automatically checks each step of your pipeline, if some of the following steps are correct, then you will award partial marks (1 mark per each step).
   - Edge detection (max 10 minutes)
   - Image blurring (max 15 minutes)
   - Image thresholding (max 15 minutes)
   - Erosion and dilation (max 20 minutes)
   - Connected component analysis (max 30 minutes)

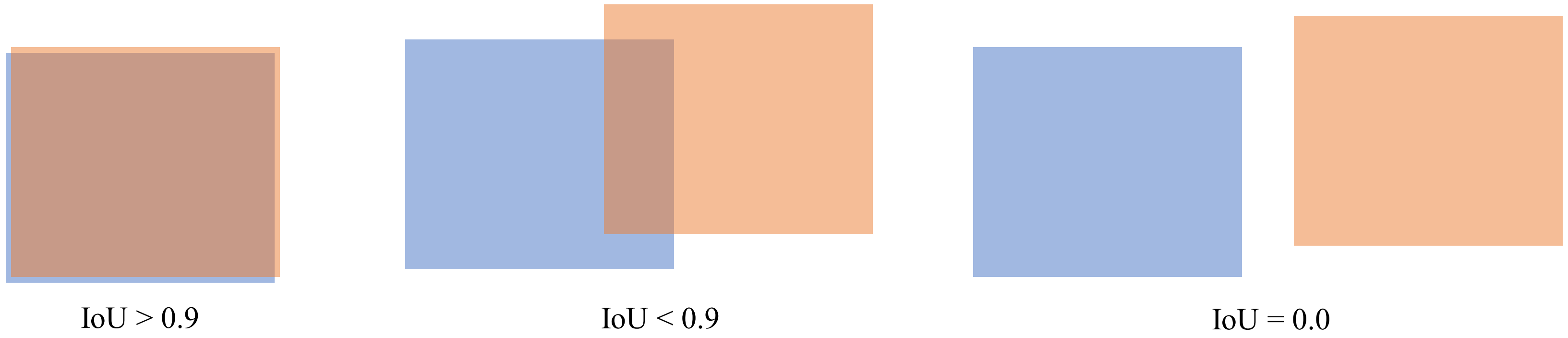
3. The max point you can get is 15 in the GitHub, but, the main component only worth 10 marks.

How the auto-marking system check your bounding box?

- We use a evaluation metric called Intersection-over-Union (IoU):

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- Area of Overleap: the overleaping area between your bounding box and the expected bounding box
- Area of Union: the total area of your bounding box plus the area of the expected bounding box.
- When comparing your bounding box with the expected bounding box, the IoU score should no less than 0.9.



Other things you need to know

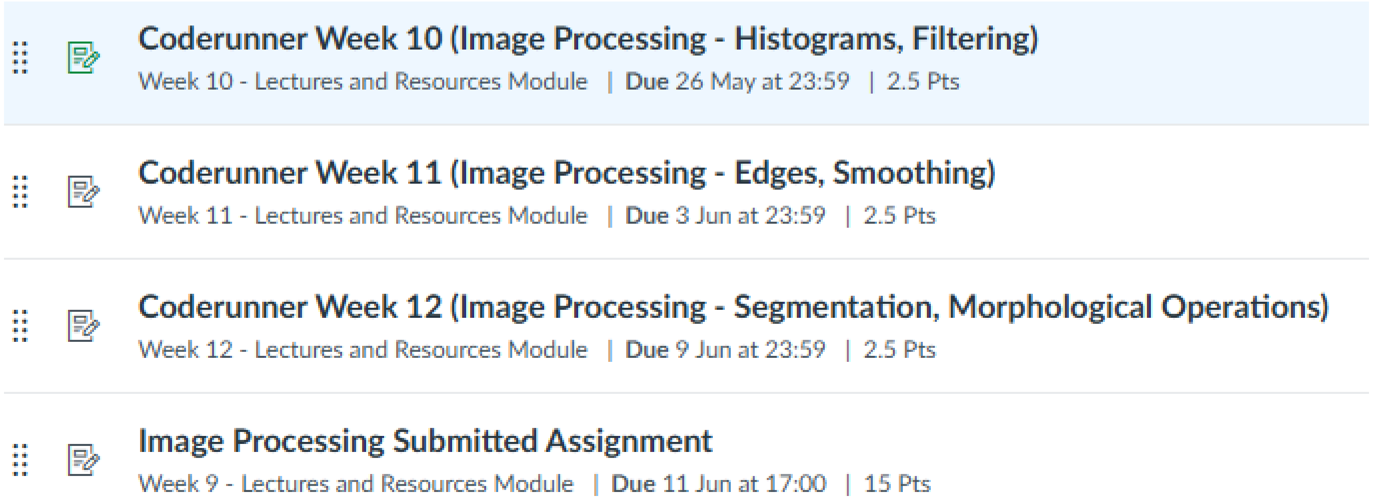
- Do not try to trick the auto-marking system by hard coded a bounding box locations in your code. We have scripts to detect that. If detected, marks will be removed!
- We will also do an offline testing, where we will run your "CS373_sign_detection.py" manually to check the results.
- The marking of your assignment extension is based on your extension implementation, your creative idea, and your report.
  - Do not use GenAI to generate your report, we will also detect that!
- To run the auto-testing locally on your machine, you need to install "*Pytest*" Python package, and open your Windows command prompt under the root direction of your assignment package, then type "*Pytest*" to run the tests.
  - You can also run a specific test by specify the path of the test file (Pytest ./tests/<test_file>).
  - The test files are located inside "tests" folder.

Some useful materials you can use

W3 Schools provides nice Python and Git/GitHub tutorials:

- To learning Python programming language, go to this link: https://www.w3schools.com/python/default.asp
- To learning Git/GitHub, go to this link: https://www.w3schools.com/git/default.asp?remote=github

Due Date for the Quiz and Assignment



Coderunner Week 10 (Image Processing - Histograms, Filtering)
Week 10 - Lectures and Resources Module | **Due** 26 May at 23:59 | 2.5 Pts

Coderunner Week 11 (Image Processing - Edges, Smoothing)
Week 11 - Lectures and Resources Module | **Due** 3 Jun at 23:59 | 2.5 Pts

Coderunner Week 12 (Image Processing - Segmentation, Morphological Operations)
Week 12 - Lectures and Resources Module | **Due** 9 Jun at 23:59 | 2.5 Pts

Image Processing Submitted Assignment
Week 9 - Lectures and Resources Module | **Due** 11 Jun at 17:00 | 15 Pts