Future Autonomous Vehicles: Better Quality Data through Image Pre-Processing

1. Background

Autonomous road vehicles and advanced driver assistance systems are fast becoming a reality. Images are the primary input used for such systems, and certain autonomous system manufacturers are aiming to enable the understanding of the environment solely based on imagery from a single on-board forward-facing camera. Image quality is thus highly important in system performance.

In this assignment, we are dealing with the image stream captured from a forward-facing camera mounted on a vehicle driving around Durham. The captured data is used for research in autonomous driving applications, such automatic detection of objects, at Durham University.

The objective of the assignment is to improve the quality of the images using image processing techniques in order to extract the best performance out of the downstream object detection system. In this scenario, the dataset we are going to be working with is made up of low-quality images captured via a broken camera (not really, the images have actually been corrupted artificially). Examples of these low-quality images can be seen below in Figure 1.

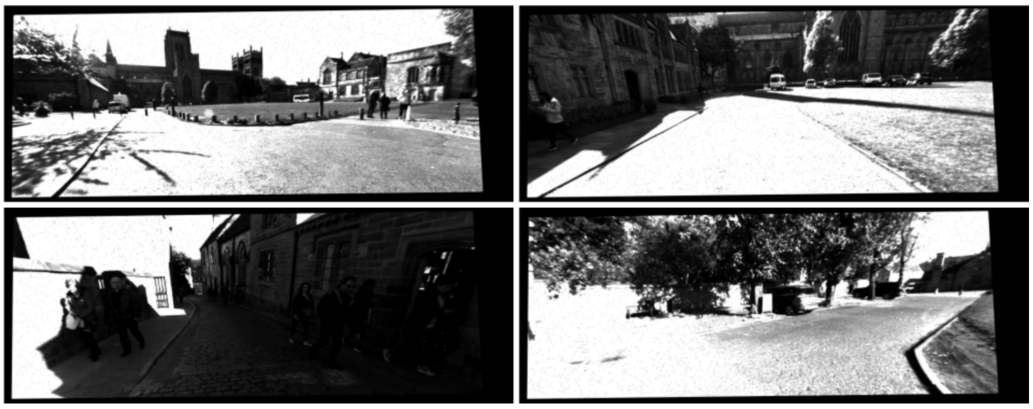

Figure 1: Examples of low-quality images that require enhancement.

This assignment requires you to improve the quality of the images, which translates as a number of distinct tasks, that will work in tandem with each other:

A. **Noise Removal:** The images contain significant amounts of noise, which hinders the performance of the downstream object detection model. Using strong filtering techniques (or any other approach, novel or existing), the noise can be removed.

B. **Adjusting Brightness and Contrast:** The images are very dark and suffer from low contrast. The images can be transformed so the brightness and contrast are adjusted and details within the image are more visible.

C. **Dewarping:** The images are distorted. Using projective transformations, the distortions within the images can be removed so that objects within the images look as they should.

D. **Measuring Object Detection Performance:** Since the end goal is for the images to be passed through an object detection system, it is important to objectively measure the effect of the image processing techniques on the performance of the object detector. It is expected that *the better the quality of the images, the better the object detection method will perform*.

This is a real-world task, comprising a real-world image set. As such, this is an open-ended challenge type task to which a perfect solution that works perfectly over all the images in the provided data set may not be possible.

In constructing your solution, you may wish to consider the following aspects of your design:

- **combining the different image processing approaches** covered in the lectures for the enhancement of the quality of the images.
- exploring **the use of filtering techniques** to improve the quality of the images.
- **the order** in which the **image processing** techniques can be applied to the images for maximum effect.

Your solution must perform well both subjectively (in most data science applications, it is expected that the input data is interpretable by a human) and objectively (your results will be numerically compared against high-quality ground truth and must improve the performance of the downstream object detection system).

2. Task Data and Example Software

You will receive two image directories, one containing a set of **validation** images and the other containing a **test** set. All images will be in grayscale. The validation set will contain both the corrupted images and the original high-quality images. This will enable you to check how your processing has improved the quality of the images, since you will have the ground truth for comparison. The test set will be used for marking and only contains corrupted images. You will only submit your results on the test data.

The validation set will contain 100 images and the test set is made up of 200. All images are in sequence from a video. The set of all images is available in a directory called “l2-ip-images” along with example videos and code within a single ZIP file from Ultra as follows:

l2-ip-assignment.zip *(~440MB)*

Videos (compressed – `DIVX` - https://www.fourcc.org/divx/) of the three sets of images (corrupted validation images, ground truth validation images and corrupted test images) are also made available on Ultra within the same ZIP file. An example video of the output of the objection method on the validation ground truth images is also provided for your reference.

The videos are created using OpenCV at 3 frames per second (the frequency at which the images are sampled). The videos are easily playable using the VLC media player – www.videolan.org. Note that a part of your submission will include submitting a video of your results as well. Your video should also be in the same format.

Two Python scripts are also included in the same ZIP file on Ultra. One contains a pre-trained implementation of YOLO (https://arxiv.org/abs/1804.02767), which is a commonly-used object detection method and the other script will output a few simple metrics to compare two images:

`yolo.py` – an example object detection approach which you can use *"out of the box"* for the purposes of this assignment (this can be treated as a black box detection component, although you will come across this method again in L3). Your use for YOLO is only to see how it works with the input images, pre-processed or otherwise. This script will read a video as its input and produce a video as its output. You can modify the script to read images from a directory. The script requires network weights ("`yolov3.weights`"), config file ("`yolov3.cfg`") and class names ("`coco.names`"). All the required files are provided with the ZIP file on Ultra.

YOLO will detect objects within the scene and this script will draw bounding boxes around the detected objects. The name of the object class and the "*confidence*" with which YOLO has made its prediction for any given object is also part of the output. For the purposes of this assignment, a simple metric, henceforth referred to as the "*quality score*", is used to measure how YOLO operates with the images it is receiving as its input.

This quality score is dependent on the **number of objects detected** by YOLO as well as the **average confidence** of YOLO across all images in the entire dataset:

$$Score = Mean[\frac{\#\ detected\ objects}{\#\ total\ objects}, average\ confidence]$$

This is not a perfect metric as low-quality images may lead to erroneous confident detections by the object detection method, but this is a natural part of real-world research applications.

For reference, the number of detected objects in the grayscale ground truth validation set is 391 with an average confidence of 0.606. The numbers are significantly worse for corrupted images. Note that it is possible for you to get better numbers than what is reported above for the ground truth images if the quality of your results exceeds what the raw camera feed provides [*this is the point of image processing and happens often!*], so you may be able to beat the numbers for the ground truth validation set (391 objects with a confidence of 0.606).

The ground truth information for the test set will not be made available to you. The *`yolo.py`* script will partially be used in the marking of your work.

`compare_images.py` – a script that will read two sets of images and compare them using common image comparison metrics, namely

- SSIM (https://en.wikipedia.org/wiki/Structural_similarity),
- Mean Squared Error (https://en.wikipedia.org/wiki/Mean_squared_error) and
- Mean Absolute Error (https://en.wikipedia.org/wiki/Mean_absolute_error).

You can use this script to compare the results of your image quality enhancement methods against the ground truth high-quality images in the validation set. Note that the corruptions will not be completely solvable since information is lost when the images are corrupted. These image comparison metrics (especially SSIM) can only be used for reference to show how well some of your processing methods (like the dewarping) are doing. They are not an absolute measure of the performance of image enhancement techniques. The *`compare_images.py`* script will partially be used in the marking of your work.

3. Hints

Here are a few hints to help get you started:

- You will notice that during the warping process, only the **perspective** has been affected.
- You can see that **multiple types of noise** are added to the image at the same time (Gaussian, Salt and Pepper, etc.)
- If you carefully inspect the images, you will notice that both their **contrast** and their **brightness** have been affected by the corruptions.
- Though using the techniques covered in the lectures will get the task done, you are not limited to what was already covered. You will even get extra credit for finding more obscure techniques or proposing new processing methods that perform the required tasks.

4. Code Specifications

Additionally, to facilitate easy testing, your prototype program must meet the following **functional requirements**:

- Your program must operate with **OpenCV 4.1.x** on the lab PCs.

- Your program must contain an obvious variable setting (or an argument parser) at the top of the main script that allows a directory containing images to be specified. e.g.

  ```
  path_to_dataset = "l2-ip-images/test/corrupted"
  ```

  from which it will cycle through the images in the specified directory, perform all the processing and save the images **without changing the filenames** in a directory called "**<u>results</u>**". The contents of this directory will also be a part of your submission.

- Your program should save the images in the same resolution as they are originally provided.

- You are not allowed to use any object detection methods other than the YOLO implementation provided for this assignment to measure the performance your work. Only this script will be used during the marking.

- The three main components of your image processing program – i.e. noise removal, contrast adjustment and dewarping must be within their own separate functions within the code. This is to enable reading the code for these components more easily.

5. Submission

You must submit the following:

- Full program **source code together with any required additional files** for your final solution to the above task as a **working python script,** meeting the above ***"code specifications"*** for testing. Include all supporting python files and clear instructions (*e.g. in a README.txt)* on how to run it on the test dataset.

- A **directory called `"Results"`** which contains the results of your image enhancement techniques on the corrupted test images. Do **not** submit your results on the validation set.

- **Video file** showing the performance of YOLO (from the provided script) on your test results. This can be constructed using OpenCV directly or any tool of your choice. Video must be made at 3 FPS. File size of the video must be less than 25MB in size, video format in use must playback using VLC – https://www.videolan.org/]. An example of such a video is provided in the hand-out ("YOLO-validation.avi").

- **Report (max. 750 words)** detailing your approach to the problem and the success of your solution in the tasks specified. Provide any illustrative images (as many as you feel necessary) of the intermediate results of the system you produce *(overlays, results of processing stages, etc.) Remember that any images, titles, captions, tables, references, and graphs do not count towards the total word count of the report.*

  Summarise the success of your system in enhancing the quality of images captured in an urban driving scenario and the effects of your image processing techniques on improving the performance of a downstream object detection method (YOLO) on the test data set. Submit this report as a **PDF** (not in any other format).

6. Marks

The marks for this assignment will be awarded as follows:

• Overall design and implementation of your solutions for:	
○ Noise removal from the corrupted images	20%
○ Improving the contrast and brightness of the images	15%
○ Dewarping the images	15%
• Performance of the object detection system (YOLO) on the resulting images based on the total number of objects detected and the confidence of the model	10%
• Clear, well documented and presented program source code	5%
• Report:	
○ Discussion/detail of solution design and choices made	10%
○ *Qualitative* and/or *quantitative* evidence of performance	10%

- Additional credit will be given for one or more of the following:
  - the design and use of an alternative or novel image processing methods
  - the use of heuristics or advanced processing to improve performance
  - significant improvements in the performance of the object detector
    (for any of the above, up to a maximum, dependent on quality) 15%

Total: 100%

Plagiarism: You must not plagiarise your work. You may use program source code from the provided course examples, the OpenCV library itself or any other source BUT this usage must be acknowledged in the comments of your submitted file. Automated software tools (e.g. https://theory.stanford.edu/~aiken/moss/) may be used to initially detect cases of potential source code plagiarism in this practical assignment which will include automatic comparison against code from previous year groups. Attempts to hide plagiarism by simply changing comments/variable names will be detected.

You should have been made aware of the Durham University policy on plagiarism. Anyone unclear on this must consult the course lecturer prior to submission of this practical.

To submit your work, create a directory named as your username (e.g. *cxfh123*). Place all required files in this directory using, **ZIP** compress/archive this entire directory structure (not .rar or .z7 or anything else - as this breaks the automated extract/test tools) and submit it via Ultra *(late submissions will be penalised following departmental policy).*

***Submission Deadline: 2pm (UK time) on 3rd February 2022* (03/02/2022)**