Introduction

This project presents an efficient method to image deblurring, making use of a secondary low-exposure (*short exposure*) image to complement and enhance the auto-exposure base image. Short exposure image is enhanced using the low rank approximation of the auto-exposure image (without any external parameters). This technique offers a much better qualitative results at a better efficiency as well. However, this technique relies on having a short-exposure image being available.

Overview

There are three parts to the pipeline:

- Image capture
- SVD/AIC of normal image
- Capture and superimposition of the brightness and contrast information on the low rank, non-blurred image
- Post-processing enhancement

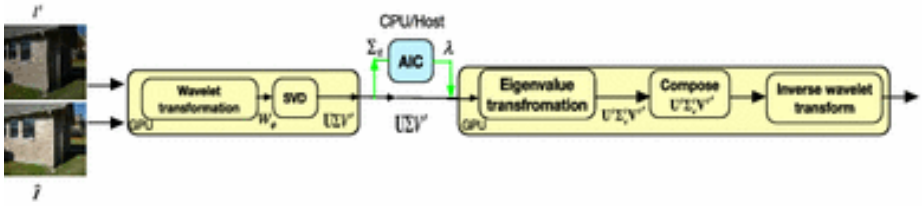

Figure 1: deblurring_pipeline

I/O Process

The input to the pipeline is two images, a short exposure image and a normal auto-exposure image. Two processess have been discussed to capture the *extra*, but **essential** short-exposure image.

To capture two images consecutively and also with different exposures or shutter speed settings, the RGB raw image data are captured noting that in a conventional image acquisition pipeline, an image goes through several stages to complete its data transfer through a buffer and an encoder. This leads to delays and such delays may cause not capturing the same scene area due to hand-shakes. Thus, in the implementation, a camera with Mobile Industry Processor Interface-Camera Serial Interface (MIPI-CSI) was used to capture images.

Figure 2: Exposure_comparison

Algorithm 1

Input: image size $m \times n$ and two exposures or shutter speeds
Output: two RGB image raw data

1. Create two memory spaces 2*3* $m*n$ bytes
2. Initialize two camera parameters
3. Capture first image and store data to the first memory space
4. Update exposure or shutter speed and connect the camera port to the second memory space.
5. Capture second image and store data to the memory space

Figure 3: sync_capture

Sequential Capture

One option is to capture the images one after the other. Now logistically speaking, this process is easy. After the subject is captured normally (*by human trigger*), a secondary image can be captured programmatically, without any human intervention.

However, this method has a couple of caveats; the scene may change between shots. This is very unlikely given the automated, almost immediate trigger of the secondary shot, but could end up reproducing an image with some reconstruction error.

Subset Capture

Another, slightly more interesting approach proposed was to have the short exposure image captured **within** the exposure bracket of the original image itself.

MIPI has become a commonly used interface protocol on mobile devices as it provides scalable serial interface for image data transfer to host/CPU processor. This way, the image raw data are directly mapped into a memory stack by enabling the camera output port. The memory size can be pre-defined based on a user-defined image size. This way delays caused by the data transfer are avoided. The pre-defined memory is also synchronized to the camera. The encoder and buffer are both deactivated. When the first image is captured, the image data are simultaneously mapped into the memory without a time delay. Next, the camera control parameters are updated using a different shutter speed. Meanwhile, the camera port is connected to a second memory space. As a result, two consecutive images get captured, each corresponding to a different exposure or shutter speed setting, while not suffering from the delay caused by the data buffer and encoder.

Method

To abstract out the process, essentially the algorithm involves an application of a low rank representation (containing information about brightness and contrast) of the original image onto the low exposure image. Given that the low-exposure image will not be blurred (*or atleast, not as significantly*), application of brightness and contrast onto this image will result in a reconstruction that is far superior (in terms of sharpness) to the original auto-exposed image, **whislt** maintaining the same ROI and frame that the original captured.

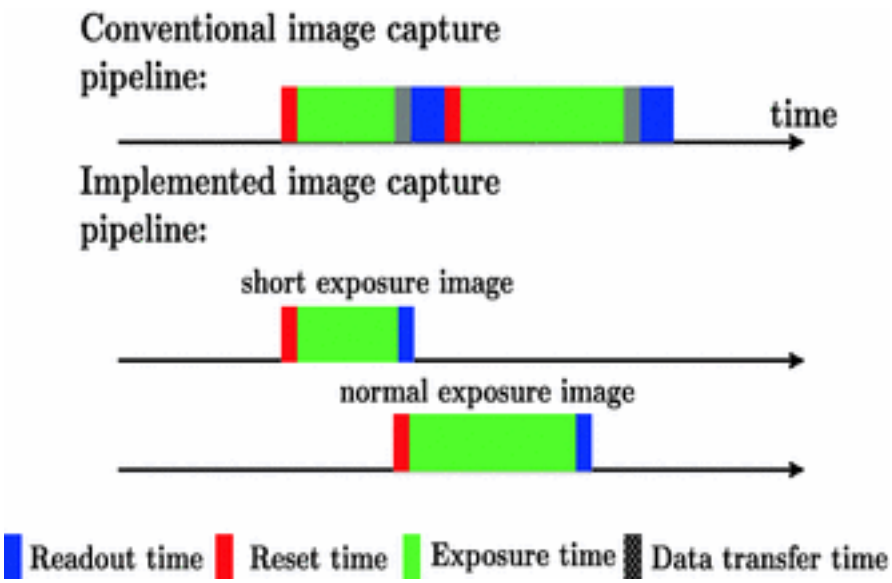

Figure 4: timing_diagram

Algorithm 3. Deblurring part

Input: two ($m \times n$) images in YCbCr format
Output: one RGB image

1. Convert the two images to the RGB format
2. Repeat steps *a* to *g* for each of the channels R, G, and B
   a. Apply wavelet transform to both of the images to lower their resolution
   b. Apply SVD to the blurred image without generating eigenvectors
   c. Calculate the AIC value from the eigenvalues of the blurred image
   d. Apply SVD to the short-exposure image
   e. Enhance the eigenvalues using the AIC value
   f. Generate a deblurred image using the enhanced eigenvalues
   g. Apply inverse wavelet transform to return to the original size
3. Combine the R, G, and B channels to generate an original size deblurred image

Figure 5: method_algorithm

SVD & Low rank representation

- Low rank matrix approximation using SVD (support vector decomposition)
- Find appropriate number of eigenvectors using AIC (Akaike Information Criterion) from the normal, auto-exposure blurred image.
- This approximation image remains undistorted and carries only the brightness and contrast information.

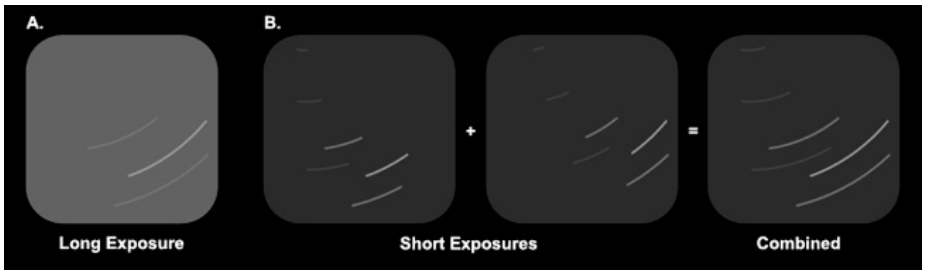

Figure 6: similar_approach

The low rank approximation can be simplified as follows:

$$\widehat{I} = E + R + Z$$

where E represents a low rank approximation image, R the detail content of the image and Z the blurring effect. In general, E does not suffer from the blurring effect as it is rank deficient:

$$rank(E) = p < m$$

Using the AIC, a proper estimate for the number of eigenvalues required is made, without any prior estimation or external help.

Once the appropriate `p` is calculated, the low rank approximation E is obtained.

Eigenvalue Transformation

After obtaining the number of eigenvalues of the low rank approximation image, the next step consists of adjusting the eigenvalues to enhance the brightness and contrast of the short-exposure image. It is important to keep the original singular value ratio of the short-exposure image to avoid introducing distortions. This is achieved by considering this enhancement ratio based on the eigenvalues of $Iˆs.

Experimental Setup

Five sets of 15 images were captured with different image resolutions:

- Set A (`800x600`)
- Set B (`960x720`)
- Set C (`1024x768`)
- Set D (`1296x864`)
- Set E (`2592x1936`)

The developed technique was implemented on both a CPU and a GPU. The most of memory consumption is from the image storage on GPU and CPU, and memory usage takes (2*3*m*n) bytes on both GPU and CPU sides.

Three images were consecutively captured from 60 different scenes. The first image was captured with a user-defined exposure or shutter speed with no handshake, while the second and the third images were captured with a short and a user-defined exposure or shutter speed in the presence of handshake movements. The first image was used as the reference. The resolution of the captured images was `1024*768`, and the two short shutter speeds were `1/100s` and `1/200s`.

Results

As can be seen, this approach does not require any search iterations for finding the enhancement parameters as done in the ATC technique. In other words, the information of the blurred image is directly used to enhance the short-exposure image.

Having gone through the problem statement and overall algorithm and procedure, one of the main focus was an efficient GPU implementation of the algorithm to highlight it's efficiency and improvements over traditional algorithms. For this purpose, I have written two codes:

- One Python base code for comparison (`opencv`)
- One GPU based CUDA-powered C++ code. (`opencv-GPU`)

Dependencies

For the most part, the code is a *vanilla* implementation using standard image processing and machine learning libraries.

Python

For the Python based implementation, the following libraries have been used:

- `numpy`: Basic matrix computations and `SVD`
- `opencv`: Image based IO and processing
- `matplotlib`: Plotting results and graphs

C++

For the C++, GPU based approach, two approaches were considered. One was an STL based non-openCV implementation and another openCV, CUDA based implementation.

- `CUDA`: Basic library for GPU interaction (version 9.0)
- `cuDNN`: Supplementary library for CUDA support (version 9.0)
- `openCV`: for image reading and other basic operations
- `thrust`: For high level abstraction on GPU based operations

Image tests

Timings tests

Conclusion

As seen from the results above, the results in the paper were verified, within experimental constraints and error. This method, as compared to the other deblurring techniques is efficient, both in the data acquisition process and computation. Because most computation is done on the GPU, further room for advancement could be in the buffer transfer, which isn't an issue if specialized hardware (*as mentioned earlier*) is used.

The findings from this paper could be used to generate a model that *simulates* the low exposure image, given a `RAW` image. Extracting information from the same, on the basis of say, exposure, shutter speed, etc, could lead to the elimination of the secondary photo requirement.

Acknowledgments

Firstly, I am thankful to the course instructor, Dr Ravi Kiran, and the TAs for giving me this opportunity to work on a new vision problem, and expand my knowledge in GPU/CUDA programming and image processing techniques. I have used the following resources (*some of which have been bundled in the repo*):

- Original Paper
- Deblurring Article
- CUDA (*Thrust*) user guide

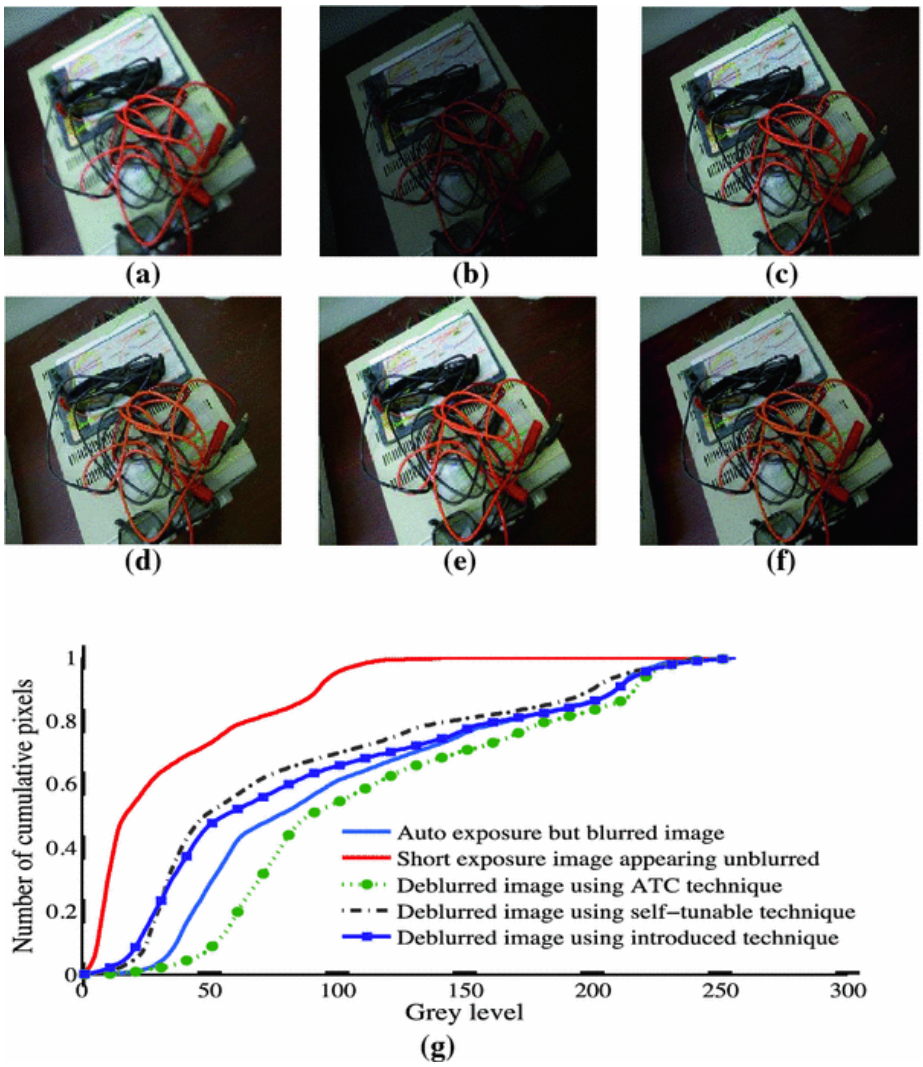

Figure 7: image_results

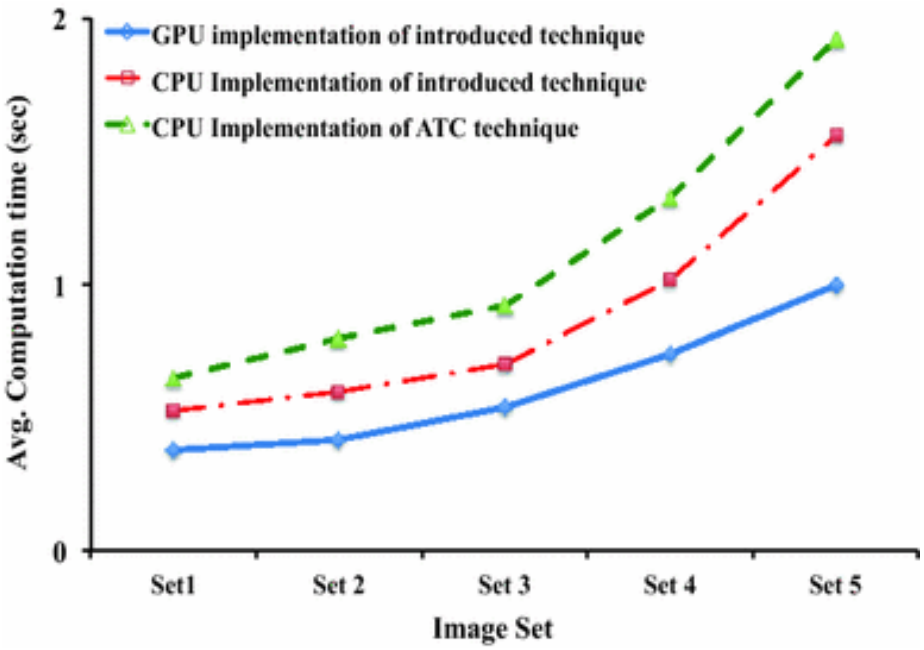

Figure 8: timing_tests

- OpenCV GPU guide