Calibration of beam size monitors

Scott Williams

June 18, 2020

Introduction

- Responses to previous comments
- Software status
- Hardware status

Previous comments

Previous comments

Some questions regarding hardware performance

- How fast is the communication, and how quickly can we move the lens?
- How stable is it?

Regarding software

- What are the software goals/performance criteria/requirements?
- What does it need to do?
- How do you know when you've reached your goal?

Previous Qs - How fast is the communication, and how quickly can we move the lens?

- When the lens is attached to a proper camera, autofocusing should be able to track fast moving objects. Eg., according to[1], lenses from approx. 10 years ago can track moving objects 10m away moving at approx 70km/hr
- Currently, it takes around 70 microseconds to send a byte using the current code. Time taken for the lens to respond varies and depends on command, but ranges from 10 microseconds to 100 microseconds. In the current code sending lots of commands quickly wasn't required, so a lot of 'safety' wait times are inserted between commands, which could be removed or finetuned if required.
- Ultimately, we'd like to be able to focus the camera in a similar time compared to how long it takes to move the test USAF1951 calibration chart into position
- Currently working on benchmarking common use cases.

[1]`https://web.archive.org/web/20131014113454/http://www.canon.com/camera-museum/tech/report/200304/200304.html`

Previous Qs - How stable is it?

Stability varies between lenses.

- The Tamron lens currently used at ThomX is very temperamental, and will often become unresponsive if a malformed command is sent, requiring a power cycle.
- The Canon pancake lens appears to ignore malformed commands, and processes the next command
- The Tamron aspherical lens used for testing in Australia appears to get stuck on malformed commands, but will respond after sending the lens sync command a few times

Overall, when sending reasonable commands to the lenses, the interface is stable. However, for ease of use, we'll include a relay to make it easy to power cycle the lens.

Previous Qs - Software goals and requirements

Hard requirements:

- Find and report the group and element number of the smallest element(s) visible
- For the largest element(s) visible, measure their dimenions and find the group and element number
- Write the calibration information from the previous points in an easily parsed file format

Soft requirements:

- Script completes in 'reasonable' time
- Accurate results
- Consistent results
- Results clearly presented

Hard requirements are simple pass/fail, while soft requirements are more debatable.

Software

Software - Measuring clarity of image

As part of implementing a software based autofocusing method, we'll have to be able to numerically or comparitively describe how in or out of focus an image is. Unfortunately, we only have a few images where the focal length of the lens has been varied, so we've had to improvise by comparing different parts of an image using our current datasets.

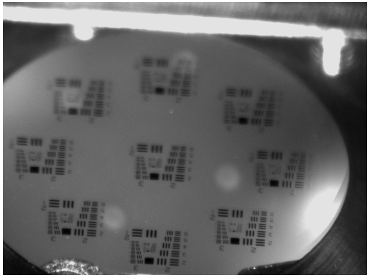

Figure 1: Example image - blurriness of target patterns varies with vertical coordinate

Software - Measuring clarity of an image

Two methods for computing focus score investigated:

- Method developed by undergradutes under guidance of Nicolas based on high frequency discrete FT components. Take 2D DFT of image, then take a subset of the DFT (indexes 1:15), and return the sum of the normed elements, ie.
  $$\sum_{u=1,v=1}^{u=15,v=15} |\mathcal{F}_{u,v}| \div \sum_{u,v} |\mathcal{F}_{u,v}|$$
- My readaptation of the above method, but with dynamically set boundaries to capture the top 25% of frequencies

Software - Measuring clarity of an image

Method:

1. Extend analysis software to also compute focus score on detected regions of interest
2. Save focus score to results files
3. Look at how focus score varies depending on vertical coordinate, and whether this relates to successful or unsuccessful analysis.

Key questions: Is focus score/clarity linked to successfully analysed images, and does focus score increase with vertical coordinate?

Software - Measuring clarity of image

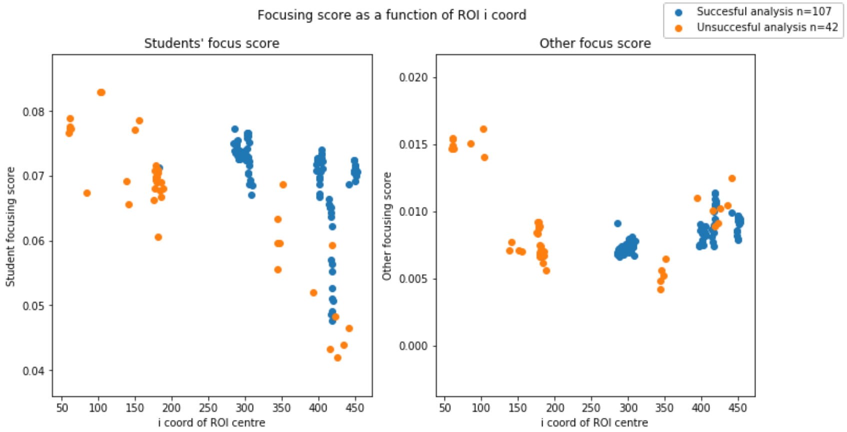

Figure 2: Scatter plot of focusing score as function of vertical coordinate. Relationship between vertical coordinate and focus score does not seem clear

Software - Measuring clarity of image

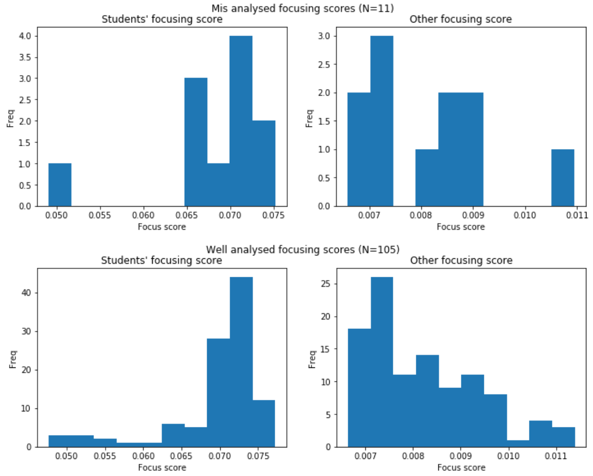

Figure 3: Histograms of focusing scores. Low statistics, but no clear graphical relationship between focus score and analysis success

Software - Measuring clarity of image

At this stage it isn't obvious whether we can compare the different levels of focus throughout the image. A lot of variation being introduced by using dissimilar images not normalising ROIs before attempting to quantify focus quality.
At this stage we could try to refine the algorithms more, particularly by doing some ROI normalisation before running the focus score algorithms, but it would probably be more prudent to wait until we have a dataset where the focus has been moved.

Software

Status:

- Implemented and tested some algorithms to try to give a numerical quantity describing how in focus the image is
- Found that these algorithms weren't particularly suited for within image comparison at this stage.

Goals:

- Get a dataset of images where the focus has been shifted, and test on that
- Implement autofocusing with lens

Hardware

Hardware - status

Since last presentation:

- Can now control aperture, though issue with apparently high current draw
- Finalised Arduino/lens interface and tested commands with lens
- Created two separate web interfaces for controlling lens via Arduino, one using ethernet adapter and one using separate computer

Hardware - HTTP interface for lens

We decided to use a HTTP for the following reasons:

- We can expose a straightforward and discoverable interface for users
- We can expose a consistent and easily parsed interface for machines
- It allows us to abstract away low level details (instead of sending byte 0x0a, issue a GET to `http://192.168.0.1/lens_sync`)
- It can be extended or wrapped later on

Hardware - HTTP interface for lens

We created two web interfaces, one using a separate single board computer (Raspberry Pi) and one running on the Arduino using an ethernet adapter, with similar functionality.
The pros and cons of each approach were approximately the inverse of the other:
Separate web server:

- Easy to extend and reprogram, separation of responsibilities
- Few hardware/processing restrictions
- Requires forwarding on commands to Arduino
- Extra infrastructure requirements

Hardware - HTTP interface for lens

Web server on Arduino:

- Reprogramming requires reflashing Arduino, Arduino now providing interface and hardware control
- Limited memory (2kb RAM, 32kb flash) and processing speed. SD card can be used to extend memory, but requires manipulating file system.
- Extra ethernet board required, but doesn't require extra power sources etc

At this stage due to reduced infrastructure requirements we will probably base the web server on the Arduino.

Conclusion

Status:

- Attempting to quantify whether image or parts of are in focus.
- Tested two approaches for HTTP interface to lens.
- Controlled the lens via HTTP interface, browser and programmatically.

Goals:

- Set up lens and camera for focusing demonstration
- Finish board

End

Backup

Hardware - benchmarking

Using python and the requests library I'll run some quick benchmarks on how long it takes to do certain operations. Currently need to rework some solder joins, so results are not available in time for presentation. With the current http interface:

- Time to get a blank page: approximately hundreds of ms
- Time to get a lens sync response: approximately hundreds of ms
- Time to do focus range sweep: couple of seconds, with generous wait times at ends of focus