NTNU – Trondheim
Norwegian University of
Science and Technology

Continuous Autofocus for Line Scanning Hyperspectral Camera

Svein Tore Seljebotn

Master of Science in Electronics
Submission date: June 2012
Supervisor: Lise Lyngsnes Randeberg, IET
Co-supervisor: Martin Denstedt, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

The main aim of the project has been to develop and evaluate a method for continuous autofocus for a push broom line scanning hyperspectral camera.

The thesis should include:

- An overview of different focusing techniques including an evaluation of suitability for the camera system in question. A description of the camera principles, focus principles and optics should also be included. The thesis should also include a justification of the method that was chosen for implementation.

- A presentation of the experimental setup including the autofocus system and camera hardware.

- Details for the implementation of the invented solution.

- Results showing the system performance. The system performance and focusing quality should be evaluated and discussed based on the presented results.

Assignment given: 10. January 2012
Supervisor: Lise Lyngsnes Randeberg, IET

Abstract

Good focus quality is essential in imaging applications. For a hyperspectral line scanning camera, lack of continuous autofocus will quickly make the image become out of focus, given curved or sloped objects. In spite of this, continuous autofocus solutions for hyperspectral line scanning cameras seem largely absent.

In the presented work a continuous autofocus system for a HySpex VNIR-1600 hyperspectral line scanning camera is detailed. Passive autofocusing techniques were first tested, showing little potential due to failure of differentiating between contrast changes and focus quality. Moving on, an active autofocus solution was developed. Using a laser displacement sensor mounted ahead of the camera's field of view, the topography of the object is measured and focus adjusted accordingly. The system is dependent on several calibrations. Calibration procedures were invented to ease the calibration process, and the obtained calibration accuracies are discussed. Several samples were tested for focus quality using two different speed settings, and the quality and measured performance is discussed. Furthermore, working conditions for the laser sensor is also investigated.

The resulting continuous autofocus system worked as intended, adjusting for topography changes within the physical limits of the system. The obtained focus quality for slow topography changes is excellent. For large topography changes over a small distance the system struggles to follow. Furthermore, adjustment for changes of 5mm over a distance of one refocusing period (4.1-4.3mm) is not recommended due to vibrations in the camera.

The refocusing period for the system is off from the set value. While not impacting the functionality directly, it indicates an implementation error in the system. Implementing a dynamic refocusing period might improve system performance. Furthermore, "pause and adjust"-scanning is suggested as a possible improvement with potential for objects with sudden topography changes. Last, a more robust rig is advisable in order to prevent camera vibrations interfering with the image quality.

Sammendrag

God fokuskvalitet er essensielt for kameraapplikasjoner. For et hyperspektralt linjeskanningskamera vil mangel på kontinuerlig autofokus gjøre at bildet raskt blir ufokusert, dersom objektet er kurvet eller skrått. Viktigheten til tross, kontinuerlige autofokusløsninger for hyperspektrale linjeskanningskamera er vanskelige å finne.

I dette arbeidet blir et kontinuerlig autofokussystem for et HySpex VNIR-1600 hyperspektralt linjeskanningskamera presentert. Passiv autofokus ble forsøkt, men viste lite potensiale på grunn av liten evne til å skille mellom kontrastforandringer og fokuskvalitet. En aktiv autofokusløsning ble deretter utviklet. Ved å bruke en laserbasert avstandsforskjellsmåler, montert foran synsfeltet til kameraet, kunne topografien til objektet kartlegges og kompenseres for. Systemet er avhengig av forskjellige kalibreringer. For å forenkle kalibreringsprosessen ble kalibreringsprosedyrer utviklet, og nøyaktigheten til disse blir diskutert. Fokuskvaliteten for forskjellige objekter, ved to forskjellige hastigheter, ble testet og kvaliteten og den målte ytelsen til systemet blir diskutert. Omgivelsene lasermåleren virker under er også evaluert.

Det utviklete systemet justerer for endringer topografi innenfor systemets fysiske begrensinger, og virker dermed etter hensikten. Fokuskvaliteten for saktevarierende topografiforandinger er utmerket. Systemet får problemer med å følge forandringene, gitt større forandringer over korte avstander. I tillegg er det ikke anbefalt å justere for høydeforskjeller på 5mm over en refokuseringsperiode (4.1-4.3mm), fordi vibrasjoner oppstår i kameraet.

Den reelle refokuseringsperioden for systemet har et avvik i forhold til den satte verdien. Dette indikerer en implementasjonsfeil i systemet, men det har ikke direkte innvirkning på funksjonaliteten. Det kan være mulig å øke kvaliteten til systemet ved å bruke dynamiske refokuseringsperioder. Videre er stopp og justérskanning forelått som en mulig forbedring med potensiale for objekter med raske høydeforandringer. Til slutt er det anbefalt å gjøre kamerariggen mer robust, for å unngå vibrasjoner som forstyrrer bildekvaliteten.

Preface

This master thesis is a natural continuation of the previous initial focus project work, adding continuous focusing capabilities to the VNIR-1600 hyperspectral camera. It came as an initial surprise that no such feature existed for the camera, seeing the importance of good focus in images. It has been especially encouraging knowing that the fruit of the labor will be put to good use, the first image captures of wounds on patients to be performed already one week after delivery of this report, using the developed system.

I'd like to extend a warm thanks to my supervisor Lise Lyngsnes Randeberg for all the advice and guidance in both the master and project work. Further thanks goes to co-supervisor Martin Denstedt for all advice, insight and the time spent assisting me in the lab, you've been of great help. Additional thanks goes to Lukaz Paluchowski for letting me use him as test subject, and for advice. Furthermore, Norsk Elektro Optikk has also kindly provided me code samples and brought me up to speed with the camera system.

Last I'd like to thank Hanna for all your support and encouragement, and for assisting me in the lab as test subject.

Some parts of the introduction, theory and camera setup sections are reused from the project work, either verbatim or adapted.

Title image: Demonstration of camera setup, showing initial adjustment of camera position using laser positioners.

Contents

1 Introduction

1.1 Background and motivation

Hyperspectral imaging is today used in several important areas, including military, medical and industrial applications. Many of these put high demands on the stability and quality of the image data, and this makes good focus throughout the image a crucial component - a component the camera was lacking prior to this work. For instance, in medical research soon to be conducted at Norwegian University of Science and Technology, trials on patients with chronic wounds are to be conducted. The body, with all it's curves and slopes, alongside with deformations in wounds, make it a challenging object for focusing. Without continuous autofocus, imaging can quickly be limited to very small areas to ensure that the entire area stays within focus. Moreover, it can sometimes be challenging to identify the parts being in optimal focus and the parts of the image needing a new scan. More importantly, however, is that the extra time required could cause patients unnecessary stress. The presented work has taken place as part of a biomedical optics group and the main target when working on the system has therefore been scanning patients.

It requires some training, experience and an understanding of focus to be able to adjust focus in an efficient manner. In addition, capturing several images and combining them is time consuming. A working continuous autofocus solution would alleviate these problems and can open up for new applications for the camera.

1.2 Key features of hyperspectral imaging

Hyperspectral cameras captures image data in a large number of spectral components (wavelengths), referred to as bands, separating them from DSLR cameras or other conventional cameras. A hyperspectral camera can capture up to several hundreds bands, also extending into the non-visible infrared, whereas a conventional camera captures only three spectral components (wavelengths corresponding to red, green and blue light). Different materials exhibit different reflection intensities over the different bands. By making a profile of the reflection spectra, one can recognize different materials in an image based on their profile. In addition the infrared light will penetrate deeper into some materials as skin and paint before being reflected, increasing the possibilities in the field of medical diagnostics and painting conservation.

The camera used for this work is a HySpex VNIR-1600, developed and manufactured by Norsk Elektro Optikk. It is a line scanning camera (see section 2.1), scanning the scene by capturing one line at a time. This implies different working conditions for a continuous autofocus solution, compared to full-scene

cameras. A more thorough treatment of hyperspectral imaging and the camera is given in section 2.1.

1.3 Summary of previous work

In the work leading up to this project, as a part of the required project course at NTNU, an initial focusing solution was developed using passive autofocusing for the VNIR-1600 camera. Several focusing algorithms were tested and a recommendation given. The report can be found in [20]. The system was successful, and the knowledge and experience gathered from that work is carried on to this work.

1.4 The autofocus problem

Focus is achieved by moving a lens' focal plane to the same position as the object one want to image. This can either be accomplished by moving the lens itself, if the lens' focal length is fixed, or by changing the focal length of the lens, if we can alter the shape of the lens.

The need to focus is present in many things, from cameras to animals and humans. Our eyes perform autofocus procedures every time we move them, a process seldom given much thought. The eye can quickly respond to changes in distance to the observed object by adjusting the curvature of the lens which is located behind the cornea. This process involves a series of steps. First, starting from an arbitrary position, the brain has to do a measure of the focus. If the image is not in focus, e.g. it is blurred, it has to readjust the curvature of the lens. Given a new focal length, the brain repeats the focus evaluation and either consider the image in focus, or adjusts the curvature more and does it all over again. At one point the image will be optimal, and moving it further will give a result worse than the previous. This quick and fine-tuned process is very similar to autofocusing procedures in cameras referred to as *passive autofocusing*. Bats have the ability to use echolocation to navigate and hunt in the dark[1]. By emitting ultrasonic waves and measuring the response from the environment, the bats can measure the distance to the surrounding objects and prey. This method of distance measurement can also be utilized in focusing, in a process referred to as *active autofocusing*. There exists a wealth of methods, which can be grouped together, as discussed shortly.

Extracting as much information from an image as possible is important in hyperspectral imaging, and a defocused image (image out of focus) will not contain as much information as the focused one. Therefore, it is important to construct a method to find the optimal lens position throughout the scanning process. Additionally, it is important to say something about the working conditions for said methods, including scanning speed and noise and contrast tolerance.

Generally, we can define the following desired qualities for a continuous autofocus solution for hyperspectral cameras:

- Low to no requirement for operator interaction and training.
- Rapid focus recognition.
- High reliability, regardless of object to be focused and lighting conditions.
- High accuracy.
- Low cost.
- If external components are required, these should have low weight and volume.

1.4.1 Methods overview

In order to limit the scope for the project, an evaluation of the different methods must be done. All the methods presented here are well tested in literature. There are two commonly used categories, in which we can group different focusing techniques, often referred to as *active autofocus* and *passive autofocus*. Passive autofocus can be further divided into *phase detection autofocus* and *contrast measurement autofocus*.

Active autofocus systems rely on the use of an energy transmitting device to measure whether the object is in focus or not. It is active in the sense that the device transmits some kind of energy to the environment, and relies on sensing how the environment responds to this energy. Common examples includes transmitting ultrasound, laser light or infrared light[11], measure the response and from this calculate the distance to the object in order to determine correspondence with the lens' focal length (see section 2.2).
Active devices complicate the system by the required extra hardware and power. Another drawback is that they in some cases cannot focus through glass, as the surface may reflect the signal. Furthermore, in a hyperspectral system, infrared and laser light can interfere with the image. A major advantage with many active autofocus systems is that they can focus without much external light, some without light at all. Passive systems will fail when the signal-to-noise ratio (SNR) in the image is too low[20].

Passive autofocus systems rely solely on measuring information in-camera, from light having passed through the attached lens. There are, as previously mentioned, two common ways to do this:

Phase detection autofocus works by measuring the phase of the incoming light using a beamsplitter, located between the lens and the camera's detector (e.g. a CCD), to split the light from it's original path and onto a dedicated focusing CCD sensor. Details in the image will be present on two different places on the

CCD (see figure 1). In many implementations, two separate autofocus CCDs are used, one for each point in the figure. By calculating the distance between the details, or rather, delay if scanning the CCD, one can calculate the lens position. For this reason, phase detection focusing is fast and reliable. The measured distance or delay give both knowledge where the lens is located, and in what direction it should move. There are two drawbacks considering implementation in the hyperspectral system. First, the need to place delicate components on the camera, in our case on the outside of the housing, makes for a fragile and expensive solution. The system needs careful calibration to work, and parameters will change between lenses. Furthermore, the beamsplitter will prevent any use of cross-polarization filtering (see section 3.1), due to the it's way of operation.

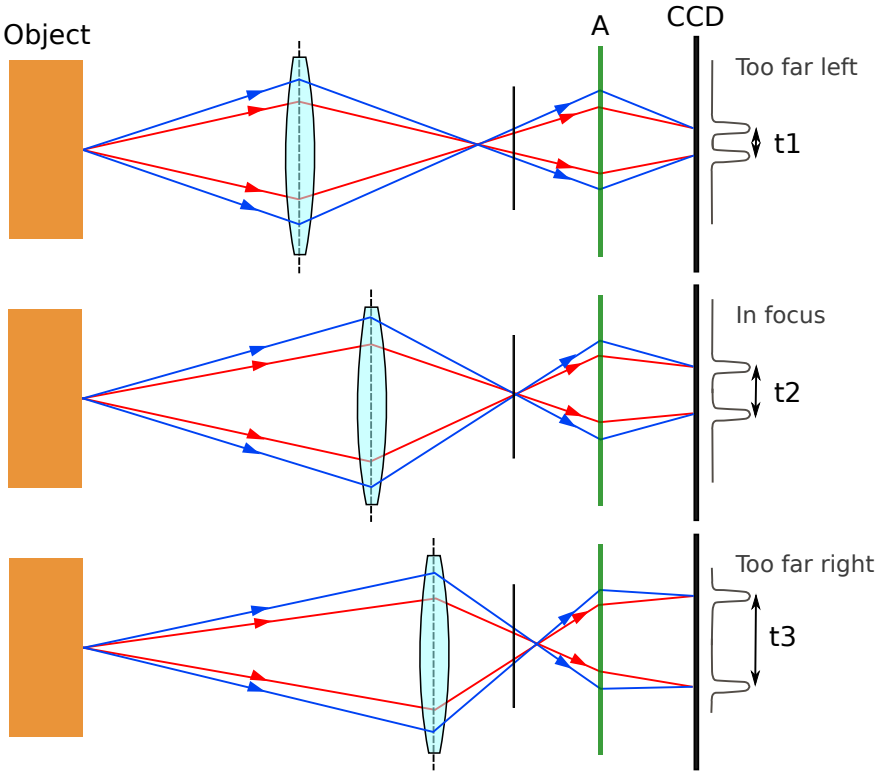


Figure 1: A phase detection autofocus scheme. t indicates measure time between peaks, when measuring over CCD. 'A' is a lenslet array. Based on [11, fig. 9.25]

Contrast measurement autofocus, as the name implies, involves a measurement of the contrast in the image. Using algorithms operating on the acquired image data, one can measure the degree of contrast in the image, either in the spatial domain or by a frequency analysis. Contrast measurement has several advantages. It requires no extra hardware components, and hence, no added cost to the system.

Moreover, it is easy to implement compared to the other methods. As mentioned, it operates directly on data from the camera. This gives two desirable properties: finding focus directly on the data is guaranteed to correspond to actual image focus, and it makes it flexible regarding change of lens. For a given offset from the focal length position, the signal will be similar regardless of what side of the position the offset is located. This presents a drawback compared to phase detection measurement, there is no way know which direction one should move the lens from one measurement. See figure 2.

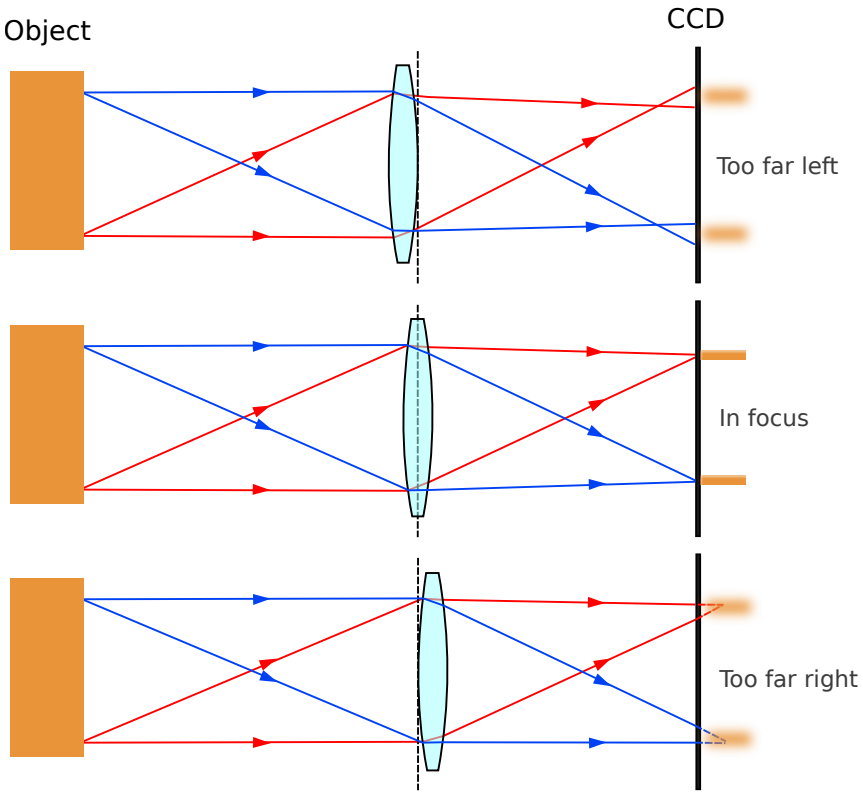


Figure 2: Contrast measurement autofocus. Note that contrast measurement focus provides no way to know which way too move the lens, since detected signal is similar for both cases.

1.5 Approach

While autofocus in cameras is a well studied subject, providing means of doing it in a continuous fashion is not well detailed. Research showed that today most modern DSLR-cameras provide a continuous autofocus function, as do most video cameras, but the underlying mechanisms are only detailed in application specific patents, related to full-scene capturing which is hard to apply to a line-scanning camera situation. One article, [6], mentioning a similar system as the one presented - but for a different camera - was found, but it is very scarce on information.

Having little to no research to base the system on, an approach for a continuous autofocus system had to be invented from scratch. Based on the successful work using contrast measurement focusing for the initial autofocus function, a passive solution was first developed and tested. This method was discarded due to unsatisfying results: the contrast changes in object makes this method unreliable. The analysis is presented in section 4.2.1.

Moving on from contrast measurement, both phase detection and active autofocus require extra equipment for the camera in question. An active solution was selected, as it is difficult to integrate a phase detection system in the camera setup and an active system can work regardless of light conditions. Due to time constraints, a ready-to-use distance measuring solution was required. Several sensors were investigated for use in the active system, both ultrasound and laser based. Ultrasound sensors were found to provide inferior accuracy and measuring rate compared to laser based sensors. Hence, a laser based triangulation sensor was selected.

Due to time constraints, only focus during continuous scanning is evaluated. For objects with high displacement changes, it might be possible to stop the scanning process while adjusting the focus. Potential drawbacks to this method is disturbances in the image and light differences. Since the applications up to now has utilized continuous scanning, and the target for the system has low topography changes, that is the chosen procedure for the proposed system.

1.6 Report outline

The report starts with a general background in the necessary theory, given in section 2. Next, a description of the camera setup is given in section 3, including relevant parameters for the system. Prototype work using passive autofocus is detailed in section 4. The implemented system, using an active autofocus solution, along with results are described and discussed in section 5. Finally, conclusion and suggestions for further work on the system are given in section 6.

2 Theory

2.1 Hyperspectral imaging

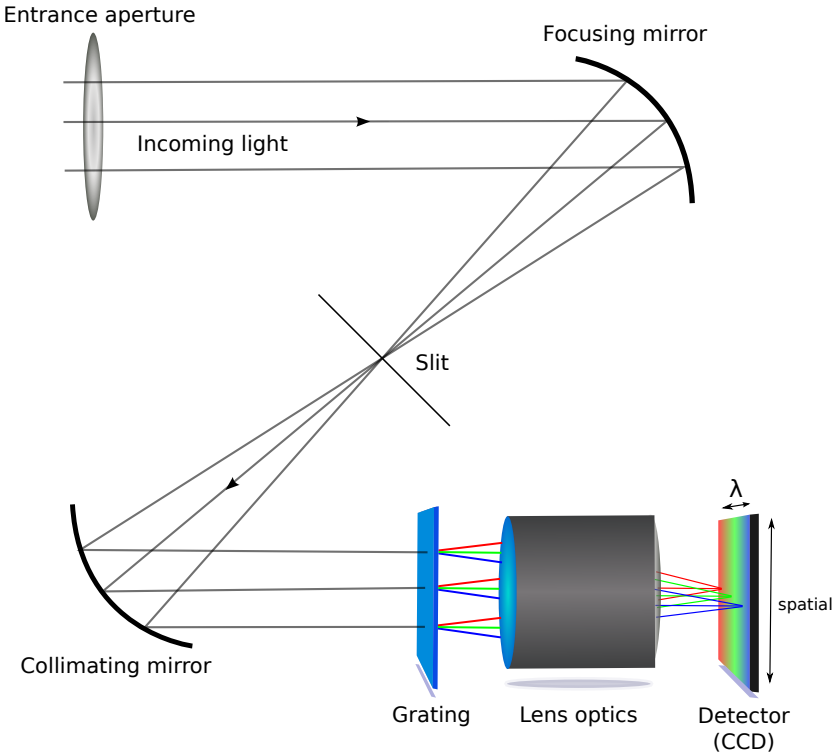


Figure 3: Principle for line imaging spectrometer. Based on [17].

As already touched upon in section 1.2, hyperspectral imaging consists of capturing images using a large portion of the electromagnetic spectrum. The spectrum usually varies from lower spectrum of visible light to infrared, giving information about an object not visible by the naked eye. This spectrum is commonly divided into segments as visible near-infrared (VNIR) in the range 0.4-0.97μm and short-wavelength infrared (SWIR) in the range 0.9-2.5μm[27].

One refer to the capturing of additional bands over the conventional three as multispectral and hyperspectral imaging. When a narrow portion of the spectrum is investigated using several bands, it is often referred to as multispectral imaging. Hyperspectral imaging, on the other hand, capture a significant part of the spectrum, giving more information about the spectrum profile of the imaged scene. There are different methods for realizing multispectral and hyperspectral imaging: filter wheel camera, tunable filter camera, fourier transform imager and linear variable filter imager, among others [7][22]. The technique used in the

HySpex VNIR-1600 camera is called line imaging spectrometer.

The concept of line imaging spectrometer is achieved by letting light shine on a grating. This light, having already passed through a slit, represents one line in the image. The grating diffract the different wavelengths onto different portions of a CCD. See figure 3. Consequently, only one line with height equal to one pixel is captured at a time. Hence, the camera has a very limited field of view (from here abbreviated as FOV), meaning the area visible in the camera. To get the full image of an object, one has to scan over the object. The previously mentioned "push-broom scanning" refers to this scanning process. For one captured image one obtains a matrix of data, with spatial coordinates on two axes and wavelengths on the third. This collected data is referred to as a hypercube, since the data is hyperspectral and varies in three dimensions.

2.2 Focus and lenses

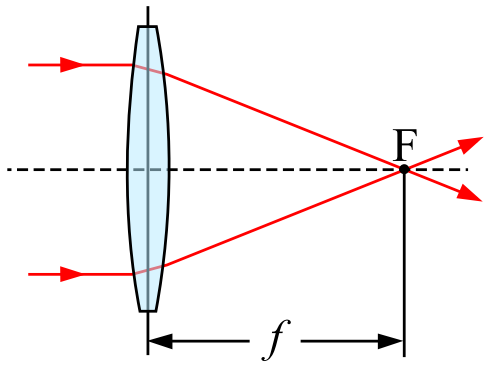


Figure 4: Focal length, f, for a lens with focus point in F. Derivative of [4]

Any camera lens will have a defined *focal length*, defining the distance where incoming collimated light is gathered by the lens into a point (see figure 4). This implicates that an object located a distance equal to the focal length apart from the lens, will be in focus for the camera. Furthermore, *depth of field* defines a distance interval within objects are "imaged with acceptable sharpness"[25]. "Acceptable sharpness" here imply that depth of field, while dependant on many parameters, is a subjective concept. However, for the scope of this thesis, the important part is that it gives focusing systems some margin to what is acceptable result. The image, however, will be at it's sharpest at exactly one focal length. The depth of field varies between lenses and is also dependant on the camera's aperture. For a treatment of the mathematics behind focus see [13].

2.3 Passive autofocus

For a push-broom camera scanning over a scene there exists at least two different methods for performing passive autofocusing.

By using the information obtained from one line only one can utilize algorithms operating on one dimensional data to measure the focus[20]. The resulting focus value will hence correspond to the latest position only. To combat noise in the capture process, additional stability can be added by using simple moving average of the previous focusing values. This, however, introduces a dependance on the previous values into the resulting value. In practice this can result in a delay in the system's response to changes in the object.

Similarly, by storing the image data in memory one can perform calculations of the focus value in two dimensions, giving an added stability to the focus measure. This will operate on data prior to the current position, also introducing a dependance on previous values, in similar fashion to the averaging process. As with the one dimensional focus, this can introduce a delay in the system response.

The procedures are detailed below.

2.3.1 1D algorithm

The energy of gradient, while not limited to 1D analysis, is the best suited algorithm for performing 1D focus analysis on data from the camera, as detailed in the project report[20].

The energy of an image is defined by

$$E_i = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |G_i(\omega, \nu)|^2 d\omega d\nu \tag{1}$$

where $G(\omega, \nu)$ is the Fourier transform of the image [3]. By Parseval's theorem this is equal to

$$E_i = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |g_i(x, y)|^2 dx dx \tag{2}$$

where $g_i(x, y)$ is the image data. Similarily the image energy for a 2D image gradient can be written as

$$E_{i,g} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |\nabla g_i(x,)|^2 dx dx \tag{3}$$

Limiting ourselves to one dimension, we can write an approximate discrete expression as

$$M_{grad} = \sum_n (x_{n+1} - x_n)^2 \tag{4}$$

for n values.

Simple moving average can be used to average the resulting values. It is defined as

$$x_{avg} = \frac{\sum_1^k x}{k} \tag{5}$$

where k is the number of samples to average over.

Energy of gradient operates on contrast in the image. As the process is comparing one pixel value with the next, a high difference (contrast) will give a high value.

2.3.2 2D algorithms

According to Subbarao et al.[26], energy of Laplacian is the recommended algorithm for 2D focus recognition. Newer approaches exists, however, such as the discrete wavelet transform[12][2]. This use of DWT show promising results and is evaluated as well.

Energy of Laplacian

-1	-4	-1
-4	20	-4
-1	-4	-1

Table 1: Laplacian matrix. Kernel for convolution with image data for energy of laplacian.

Utilizing the Laplacian in image processing can be achieved by convolving the data with the kernel given in table 1 [13] and squaring the result. In practice, given image data d, this can be written as[3]

$$M_{lap} = \sum_x \sum_y (d_{xx} + d_{yy})^2 \tag{6}$$

where

$$\begin{aligned} d_{xx} + d_{yy} = \quad & -d_i(x-1, y-1) - 4d_i(x, y-1) - d_i(x+1, y-1) \\ & -4d_i(x-1, y) + 20d_i(x, y) - 4d_i(x+1, y) \\ & -d_i(x-1, y+1) - 4d_i(x, y+1) - d_i(x+1, y+1) \end{aligned}$$

Energy of laplacian also operates on contrast in the image. A high difference in contrast between the center point and the surrounding values will yield a high output value.

Discrete wavelet transform

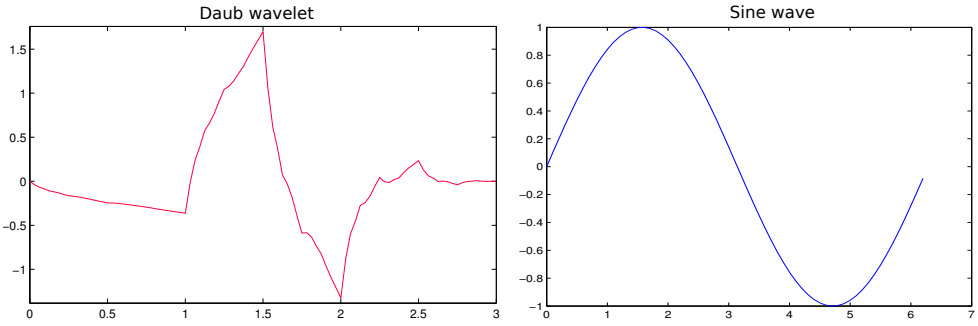


Figure 5: Comparison of a Daub wavelet function and a sine wave

The discrete wavelet transform (DWT) has many similarities with the fast fourier transform (FFT). They are both linear operations, generating a data set of coefficients which, given a set of basis functions, can regenerate the original signal. These basis functions consists of different sine and cosine functions in the FFT case, and different wavelets in the DWT case. Figure 5 illustrates the difference and complexity of the basis functions in the two cases. One of the most interesting aspects of the DWT is that is preserves both frequency and spatial information, in contrast to FFT, where only frequency can be restored. The reason for this is that while wavelets are localized in space, sine functions are not. For the FFT, one uses a given window width to truncate the different frequencies of the basis functions. In the DWT case, the window width will vary. One can use short, high-frequency wavelets to obtain the signal discontinuities (high-frequencies) and long, low-frequency wavelets to obtain the lower frequency spectrum.

For a one dimensional discrete signal x, it's DWT is defined by[28]

$$a_1(n) = \sum_{k=-N}^{N} \alpha_k x_e(n+k) \tag{7}$$

$$d_1(n) = \sum_{k=-M}^{M} \beta_k x_o(n+k) \tag{8}$$

where x_o and x_e are the odd and even points respectively, and their length is $\frac{length(x)}{2}$. α_k and β_k are the wavelet coefficients, with N and M samples respectively, mirrored around k=0. The coefficients depend on the wavelet used, and the number of coefficients are often referred to as taps[14]. $a_1(n)$ is called the *approximation* coefficients. It describes the low-pass information in the signal and it's length is half of x. Similarly, $d_1(n)$ is called the *detail* coefficients and

describes the higher frequency components of the signal. These coefficients are often denoted by L (low-pass) and H (high-pass).

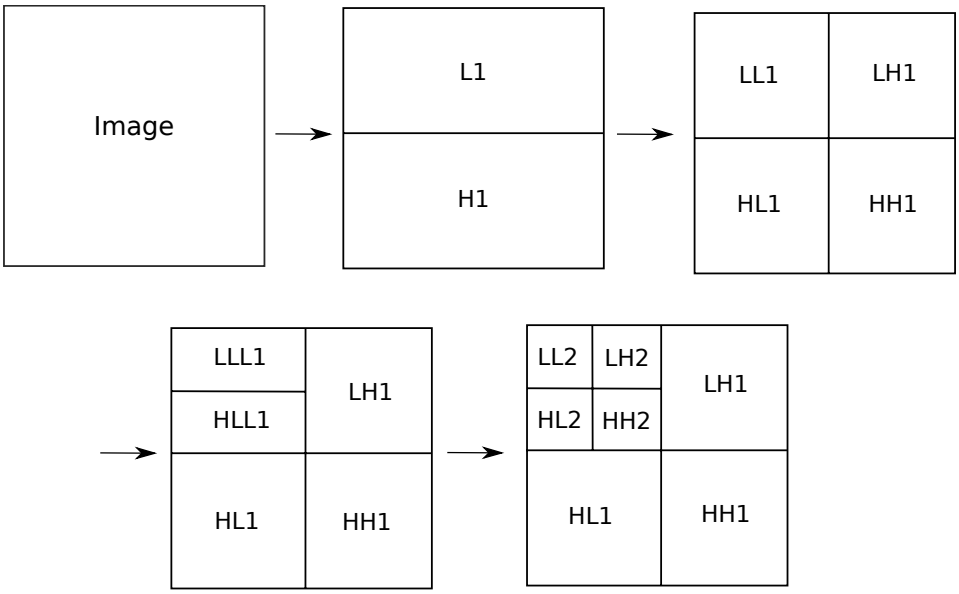


Figure 6: 2D DWT process for 2 levels. L indicates low frequency coefficients, H indicates high frequency coefficients and the number indicate the level.

To perform 2D DWT, one first applies the 1D DWT to every row in the image. From every row there is now two outputs, the L and the H coefficients. For each row, one performs a column-wise 1D DWT on the L and H coefficients separately. This results in 4 different data sets, LL1, LH1, HL1 and HH1, each having a dimension of N/2 x N/2 (given an initial length NxN). Together they constitute a level 1 2D DWT.

One may repeat the process again in order to obtain a higher level DWT. Usually this is done using the LL1 coefficients as input, giving new detail coefficients HH2, and approximation coefficients LL2. See figure 6.
Kautsky et. al [12] suggests a focus measure function utilizing 2D DWT to measure focus in an image. By using the high frequency data one can write a focusing algorithm as

$$M_{DWT} = \frac{\|h_w(f)\|^2}{\|f\|^2 - \|h_w(f)\|^2} \tag{9}$$

where $\|\cdot\|$ denotes the euclidean norm, f is the image data and $h_w(f)$ is the high frequency data resulting from the DWT transform. It operates on frequency information and grows as focus gets better. Higher level DWTs can be used by

using all the resulting high frequency data, as indicated above by HH1, HH2 etc. A 4-tap, level 2 DWT is seen as a good compromise between speed and quality for a focus system.

2.3.3 Noise

Spectral image sensors are almost exclusively based on photon detectors, which include multiple sources for noise. The incoming photons excite electrons, picked up by a detector, deciding the intensity of the incoming light. For the VNIR-1600 camera, the electron count can be modeled as[21]

$$N_{el}[i,j] = \eta[i,j] \cdot N_{ph}[i,j] + i_d[i,j] \cdot t_i \tag{10}$$

where parameters i and j indicate band number and pixel number respectively. η is the quantum efficiency, here defined to incorporate all losses through the system, including optical losses and losses due to grating efficiency, detector fill factor and quantum efficiency of the detector itself. N_{ph} is the number of photons arriving at the entrance aperture, during a chosen integration time t_i. Finally, i_d defines a dark current contribution, given as the electron count per time unit in one element of the detector. The dark current is the constant response of the detector, while not exposed to light. It is multiplied by the same chosen integration time, t_i, to give number of electrons.

From this equation it is easily seen that there is a balance between N_{ph} and $i_d \cdot t_i$, both parts proportional to the integration time. For low light situations, one has to increase the integration time in order to receive sufficient signal to make out the details in the image. This increases the noise in the signal stemming from the dark current in the detector elements. In addition one has to take into account additional noise in the light entering the aperture, coming for instance from the light sources and reflections from surroundings. See [16] for more on noise in image sensors.

The focus detection algorithms outlined above are heavily influenced by noise. Noise mask the details and adds false contrast to the image, potentially giving false positives for acquired focus. The algorithms are, however, not dependant on the source of the noise since they operate on the acquired data, merging all sources of noise together.

2.3.4 Band selection

As mentioned, it is required to make a selection of bands to operate on. A process using the luminance in a YCbCr color conversion is detailed in [2]. By using one band in each of the red, green and blue regions, one can calculate the luminance as follows:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{11}$$

where R, G and B are the values in the red, green and blue bands respectively.

2.3.5 Focus window

In order to focus on a region of the full image scene, one has to cut the input into what is referred to as a focus window. This region has to be small enough in order to give focus on the relevant area only, while large enough to provide sufficient data for the focus algorithm to operate on.

2.4 Active autofocus

As briefly touched upon in the introduction, there are several ways to achieve active autofocusing. However, the principle remains the same: if the focal length is known, measure the distance to the object and adjust accordingly. A distance measuring sensor can be used for this task. The sensor can either be placed directly in the FOV, as in some traditional full-scene cameras, or a certain distance outside of the FOV. By placing the measurement point in the FOV one can assure that the measurement is directly related to the focal length of the lens. For a scanning camera, however, it can be more suitable to place it a small distance ahead in scanning direction. This way the system will have measurements some time ahead, and can respond to changes in a more efficient manner. It also makes attaching an external sensor to a system less complicated. There are, however, some challenges connected to this way of placing the sensor, as will be discussed shortly.

2.4.1 Displacement measurement

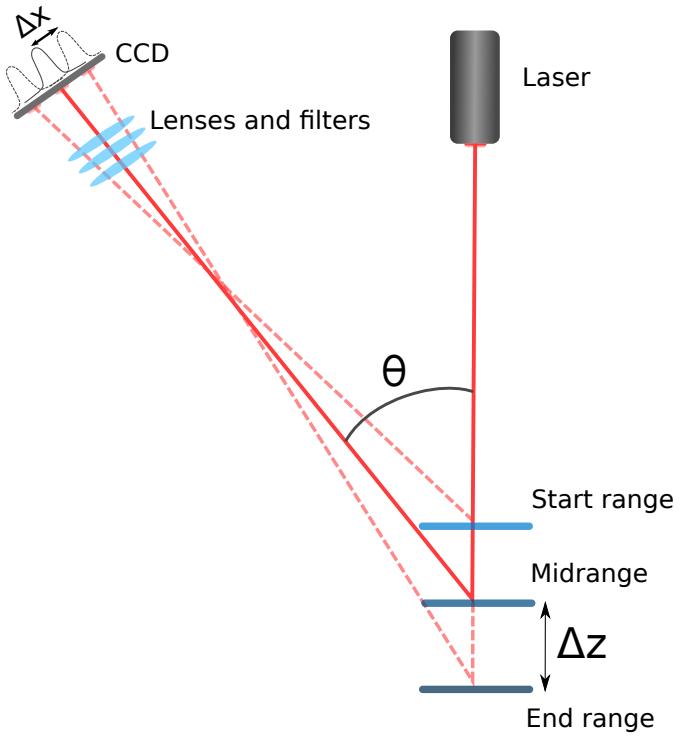


Figure 7: Principle for laser displacement sensors. Proportions are exaggerated.

A laser displacement sensor is used for measuring changes in the distance to the object relative to the focal length of the lens. A simple working principle is shown

in figure 7 [5][24]. A laser is shone on the object and reflected back to the sensor. The distance to the object gives the location of the spot on the CCD where the intensity will be highest, and from this position the sensor can calculate the distance by

$$\Delta z = \frac{\Delta x}{\tan \theta} \tag{12}$$

given that the difference $\tan \theta_1 - \tan \theta_2$ for the two positions are negligible.

A more advanced and precise model for laser triangulation can be found in [19]. To obtain optimal resolution the data can be averaged by the sensor using simple moving average or by using the median [15].

2.4.2 Focus procedure

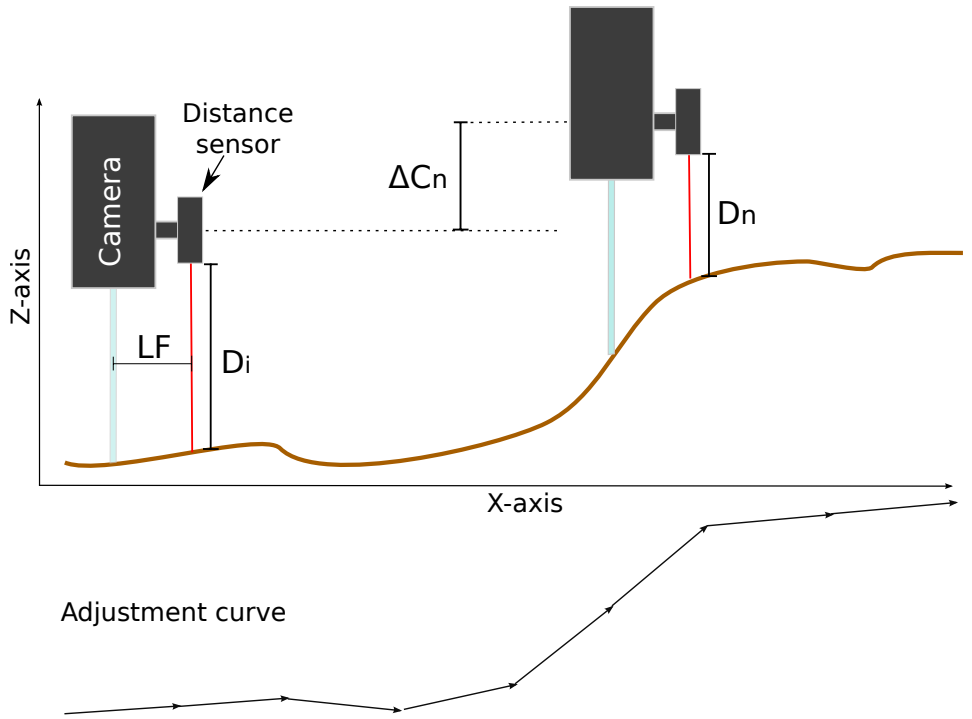


Figure 8: Displacement measurement autofocus concept, showing a camera at initial position and after position adjustment. Using the sensor, the next position for the camera can be calculated. LF is the distance between the camera FOV and the laser sensor measurement position. D_i is the initial distance measurement. ΔC_n is the displacement in camera position from initial position, relative to z-axis. D_n is the new distance measurement.

The basis for a distance sensor based system is depicted in figure 8. Using a distance sensor, in this case a laser based sensor, one measures the distance from the camera to the object. The sensor is mounted a certain distance ahead of the camera FOV, giving information about the height profile of the object ahead of time of arrival. Given that the camera is in focus at the beginning, adjusting for the displacement in distance to the object will ensure that the camera continues to be in focus. There are two caveats, however, due to the sensor positioning. Since the sensor is positioned a distance ahead of the FOV, the measured distance at initial position is not equal to the focal length (i.e focus distance), unless the object is flat. To compensate, the sensor must be moved a distance LF backwards before the initial focus distance measurement is taken. By moving the camera backwards, one can also map the initial "blind area" given by LF. Furthermore, due to the measurements being done ahead of arrival, all measurements have to be tagged with their position, relative to x-axis, at measurement time. By doing this, one can find the measurement to be used by subtracting their positions by LF.

With this in mind, to calculate the correct z-position for the camera, the following formula can be used

$$C_{next} = (D_i - D_n) + \Delta C_n \tag{13}$$

Here D_i is the initial distance measurement at FOV, ΔC_n is the displacement in camera position from the initial position, relative to z-axis, and D_n is the new distance measurement. By adjusting the camera at certain intervals, one achieves an adjustment curve similar to the one in the figure.

A more in-depth explanation on how to achieve this is given in section 5.3.

3 Camera system setup

3.1 Description of camera setup

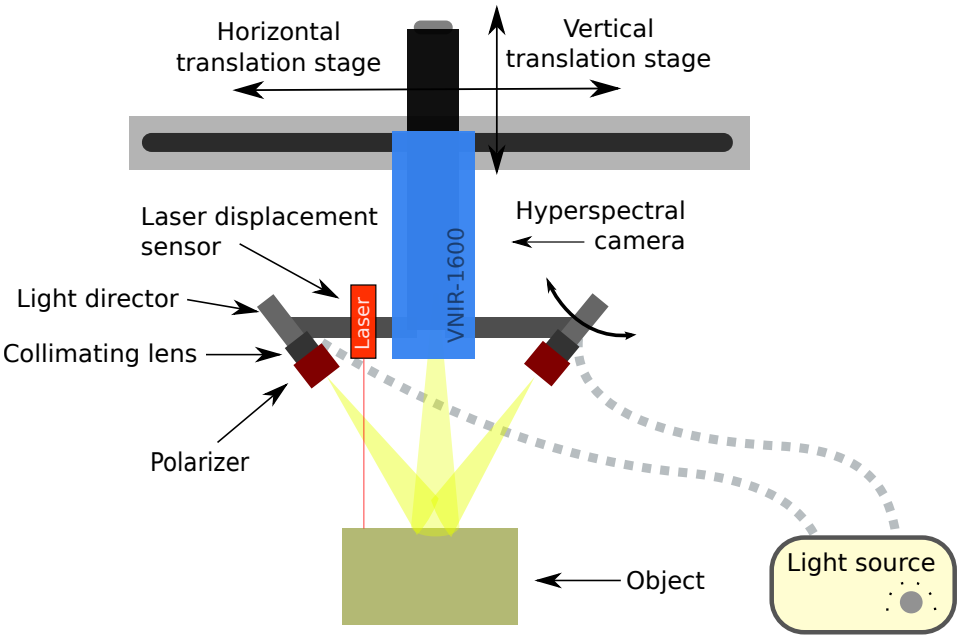


Figure 9: Sketch of camera setup

The camera is mounted on a custom made rig, made for the purpose of imaging patients lying on a bed. See figure 11. The setup consists of numerous components, each playing an important role in the system as a whole. See table 2 for an overview. The VNIR-1600 camera (figure 10) is, for the purposes of this work, equipped with a lens having a focal length of 30cm. The depth of field for the lens were found to be ~2.5mm on either side of the focal plane[20]. Polarizers mounted on the light emitters along with a polarizer mounted 90° on the camera greatly reduce the level of specular light in the images. The camera is mounted on a 10cm translation stage in vertical position. This stage is again mounted on a 49cm horizontal translation stage, together giving the camera two degrees of freedom. See figure 9. The horizontal stage serves the purpose of scanning the field of interest, while the vertical provides means of focusing. Furthermore, the rig contains a larger 100cm vertical translation stage for initial adjustment of height - this stage is not controlled in real time. The weight placed on the 10cm vertical translation stage is 8.85kg, while the weight on the 100cm vertical stage is 25kg.

The camera is controlled by a I/O and power box, connected to a framegrabber in a computer. The translation stages are controlled by stepper motor controllers,

connected to the computer by USB. The laser displacement sensor is connected to the computer by a RS422 to USB-converter.

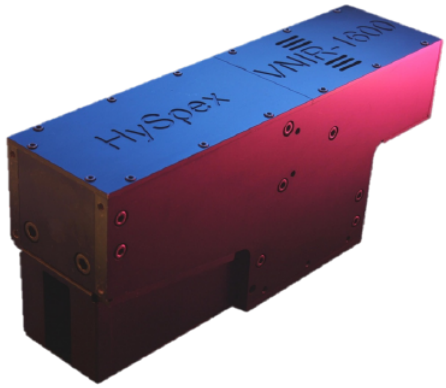


Figure 10: The HySpex VNIR-1600 camera [18]

Two different programs are used to interact with the camera, HySpex GROUND and HySpex AIR. Both are developed at Norsk Elektro Optikk. The most feature complete one, HySpex GROUND, is developed for laboratory use and other ground based applications. HySpex AIR, on the other hand, is specialized for airborne applications, for instance military reconnaissance. The latter includes a built server, serving the image stream live from the camera - a crucial component for the autofocus application, to be presented later. It provides no means to control translation stages, however, something GROUND does. Both programs are used in the research and in the presented work, as follows:

- To quickly investigate a scene, for instance to check initial focus quality, *HySpex GROUND* is used. The full scene image is saved to a .hyspex file, together with a header file. The former contains the raw data, and the latter describes the corresponding format of said .hyspex file, along with camera settings and band information (wavelengths).

- For realtime analysis of data, *HySpex AIR*'s server is used to provide a convenient way to interact with the camera. Data is served by TCP/IP protocol. The application provides means for controlling integration time and capture rate. Moreover one can enable software binning of the data, an averaging process compiling several bands into one band. This reduces the bandwidth required per line of data.

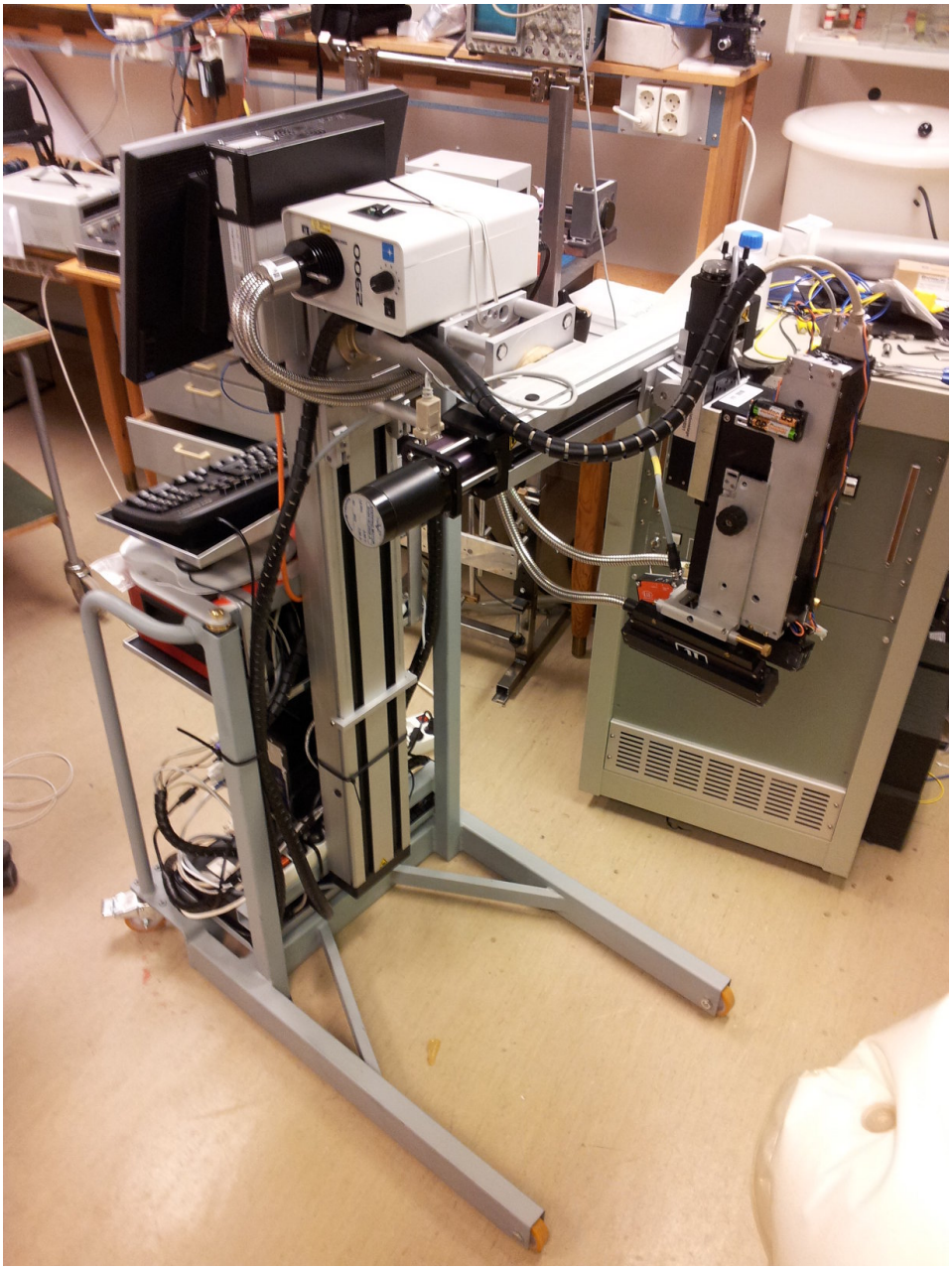


Figure 11: Camera setup. Custom made rig for scanning patients lying on a bed.

Figure 12: Position of the laser displacement sensor on the rig.

Component	Part name	Serial no.
Hyperspectral camera	HySpex VNIR-1600	S0002
Framegrabber	TeleDyne DALSA X64-CL iPro 1	637024
Vertical translation stage	Standa 8MT175-100	-
Horizontal translation stage	Iselautomation LF4 L490mm	261912
1 x laser displacement sensor	Micro-Epsilon OptoNCDT1302	-
1 x light source	Illumination Technologies Model 2900	-
2 x light emitters	-	-
2 x collimating lenses	-	-
2 x light polarizers	VersaLight VLR-100-NIR	-
2 x stepper motor controllers	Standa 8SMC1-USBhf	-

Table 2: Component overview for the camera setup

3.2 Setup parameters and limitations

The various components provide many necessary features, but in order to utilize them properly a understanding of their parameters and limitations is required.

The OptoNCDT 1302 **displacement laser sensor** provides measurements in a 200mm range, starting at 60mm distance from the laser. The measuring rate and accuracy is given in table 3. The cost of the sensor was 10000 NOK, all necessary equipment included. The sensor is used with an averaging of 75 samples, giving 10 measurements per second. The spot size is 2.1-2.3mm within the measuring range, depending on the distance from sensor [15].

If tilted, the laser sensor's accuracy is reduced. The claimed accuracy deviation is listed in table 4. Due to the way the light emitters are positioned on the camera system, the laser has to be positioned so that a tilt at a slight angle relative to surface is required in order to hit same area as camera FOV. See figure 12. This has to be taken into consideration when evaluating the system performance.

The translation stages are controlled by sending parameters of speed (an integer number) and desired position (in what is referred to as "steps", also an integer). There is no given conversion between steps and metric distance, or from speed to actual movement in mm/sec. Furthermore, querying the position of the stage returns number of steps from initial position - a position set to a program-defined number at initial execution of program. There is hence no way in software of determining in what absolute physical position the stage is. While the stages support varying speeds, a too high speed will cause vibrations in the rig, causing undesired movements in the image. Also it is expected that a too high speed can wear out the stages faster.

Component	**Speed**	**Accuracy**
Vertical translation stage	Max: 10mm/sec	Full step: $2.5\mu m$. 1/8 step: $0.31\mu m$.
Horizontal translation stage	Max: 167mm/sec*	Unknown.
Laser displacement sensor	750Hz	No averaging: $100\mu m$ Avg. 64 samples: $40\mu m$

Table 3: Specifications from manufacturer for the crucial components in the autofocus system [23][10][15]. *Spindle pitch of device unknown, least possible maximum speed listed.

Angle	**X-axis**	**Y-axis**
$\pm 5°$	0.24 mm	0.24 mm
$\pm 15°$	0.4 mm	0.4 mm
$\pm 30°$	1 mm	1 mm

Table 4: Typical measurement errors given the angle of the laser sensor relative to target. Data from [15], calculated to mm for a 200mm sensor.

4 Passive autofocus

4.1 Method and implementation

For evaluating the use of contrast measurement algorithms, the algorithms were implemented in Matlab and several test images captured with the camera. The Matlab codes for the algorithms are included in sections 7.6.1 and 7.6.2 in appendix. The DWT library used is found in [8]. A simple test procedure were developed to test the images with all the algorithms. The code can be found in section 7.6.3 in appendix, and the results are included below.

4.2 Prototyping

4.2.1 Test images results and discussion

In order to evaluate passive autofocusing, a flat, wooden box is imaged. The wooden texture (figure 13) is homogeneous and has a low contrast, ideal for prototyping measurements due to the stable SNR. The flat surface aids control and consistency in the measurements. Moreover, the wood texture share some resemblances to the texture of skin, the prioritized target for the system. The images in figures are normalized in order to better display the actual data the algorithms are working on. Furthermore, the resulting focus values are normalized for comparison purposes.

Figure 13: A section displaying the texture of the wooden box.

In figure 14 the wooden box is tilted at a slight angle, while the vertical camera

position is constant, yielding a gradually defocused imaged. Height change is linear, with the end being 6mm above the beginning. All algorithms indicate the worsening of focus quality, however, while the 2D algorithms descend in a near-linear fashion, the 1D algorithm falls quickly at the beginning. This is a desirable quality, as it is easier to pinpoint exact focus location when the algorithm show greater discriminability. The 2D laplacian is clearly least affected by line to line noise, stemming from the camera. While the curves for the 2D algorithms are generally smoother throughout the sample than the 1D counterpart, the better noise tolerance is especially visible in the right part of the figure in the figure. While not seen in the image due to the normalization, the directed light from the light emitters falls outside the camera's field of view, giving a dark image and hence a lot of noise in the image. This noise contributes to a large contrast change in the image, giving higher values for the focusing algorithms as explained in 2.3.3. The 1D algorithm, operating only on one line at a time is at a great disadvantage compared to the 2D algorithms. Averaging over the same number of lines as the 2D sample size is not enough to compensate. In conclusion, this result show that it is possible to detect focus using passive autofocus on a homogeneous sample.

Figure 15 show similar measurements, but this time the box is not tilted, i.e. the whole scan is in focus. In principle, a constant value is desired, as the whole sample has the same focus quality. The results demonstrates that a homogeneous sample with only minor contrast differences can still make the algorithms vary substantially. The lowest values for the 1D, 2D laplacian and 2D DWT algorithms are, respectively, around 30%, 45% and 70% of the highest values. Once again, the 2D laplacian show the highest noise tolerance, yielding a smoother curve than the other two. The 2D DWT also show good noise tolerance, and it has an overall low responsivity to contrast changes. The 1D algorithm demonstrates a lot of fluctuations that cannot be from contrast itself, but which are stemming from the noise in the image. These results clearly demonstrate a problem with using passive algorithms for focus detection. Although the focus quality is constant, the algorithms fluctuate to a great extent. This will either lead to false refocusing or a very high threshold for refocus. Comparing the values to those in figure 14, one has to go all the way to around line 420 to get below the lowest values. The displacement of the object in this area is around 4.6mm[1]. This is outside the depth of field of ~2.5mm.

The final result is shown in figure 16. Here a piece of colored paper is fastened to the sample, generating a large contrast between the wood texture and the patch of paper. As seen in the figure, the algorithms react enormously to this contrast change, and it demonstrates why a passive system will not work on samples which are not sufficiently homogeneous. There is no way for the system to know whether the image is suddenly out of focus or whether there is a contrast change in the image. Normal full-scene cameras do not experience this problem as it uses the same position for comparison. Not having that possibility, using

[1]Basic calculation yield 6mm * 420 lines / 550 lines = 4.6mm

contrast measurement methods is found incredibly difficult.

One can add to this the consequences of using averaging. Averaging over 20 lines, assuming a framerate of 100 frames per second, one has to wait 200ms to be sure that the new measurement is correct. This is due to high contrast changes being left in the buffer will influence the result to a large extent. This further complicates a contrast measurement system.

4.2.2 Benchmarking

Algorithm	**ms/call**	**% @ 100FPS**	**% @ 33.3FPS**
Energy of gradient (1D)	0.058	0.58%	0.19%
Energy of laplacian (2D)	0.9	9%	3%
DWT (2D)	15.2	100%	50.67%

Table 5: Benchmarking results for the algorithms. Table shows ms/call for a 20 line sample, and how much of the alloted time is spent at 10ms per frame and 30ms per frame.

The C++ implementation of the algorithms were benchmarked and the results are given in table 5. Code can be found in section 7.7 and data from the profiling is given in section 7.3, both in the appendix. Percentages of the time spent compared to the frame rate is calculated as well. Naturally, the algorithms are not required to do a calculation on every new frame, but it is desirable and gives a good reference for speed comparison.

Energy of gradient is the fastest, only spending at most 0.58% of the alloted time at 100 frames per second, leaving 99.42% to other operations. Energy of laplacian is also fast, leaving 91% of the time window to other operations. The DWT algorithm, on the other hand, spends more time than what is given between the frames. At lower speed, 30 frames per second, it spends 50.67% of the time doing calculations. From experience with the system, this leaves too little time to other operations like data cutting, displaying of image, focusing calculations and so on. It is possible that a major speedup of the DWT calculations is possible, by optimizing code for this task or by implementing CUDA/OpenCL algorithms. That is, however, out of the scope for this project. In conclusion, both energy of gradient and energy of laplacian is suitable from a time consumption perspective, while the DWT algorithm looks too slow, but further investigation is needed in order to conclude.

4.2.3 Concluding remarks

Based on the results from the conducted tests, a contrast measurement autofocus system for continuous focusing cannot be recommended. It is not possible to

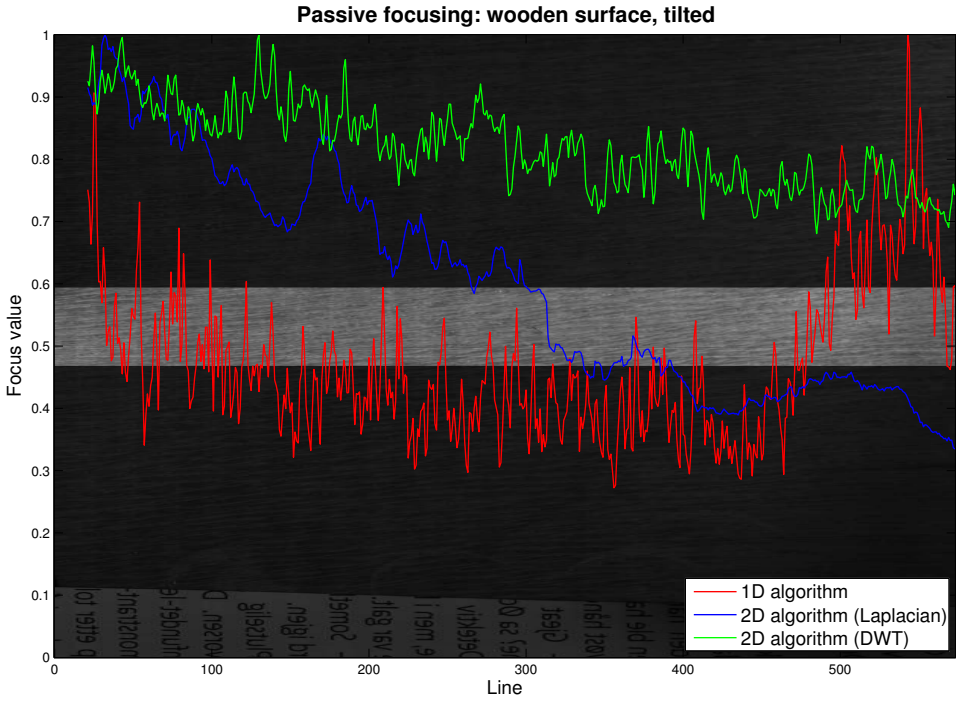


Figure 14: Focus measurements on a flat, wooden surface. The surface is tilted, while camera position is constant, resulting in the image gradually becoming defocused. Only a window in the middle of the image is used in the focus calculations, the rest is toned down. Height change is linear, with the end being 6mm above the beginning.

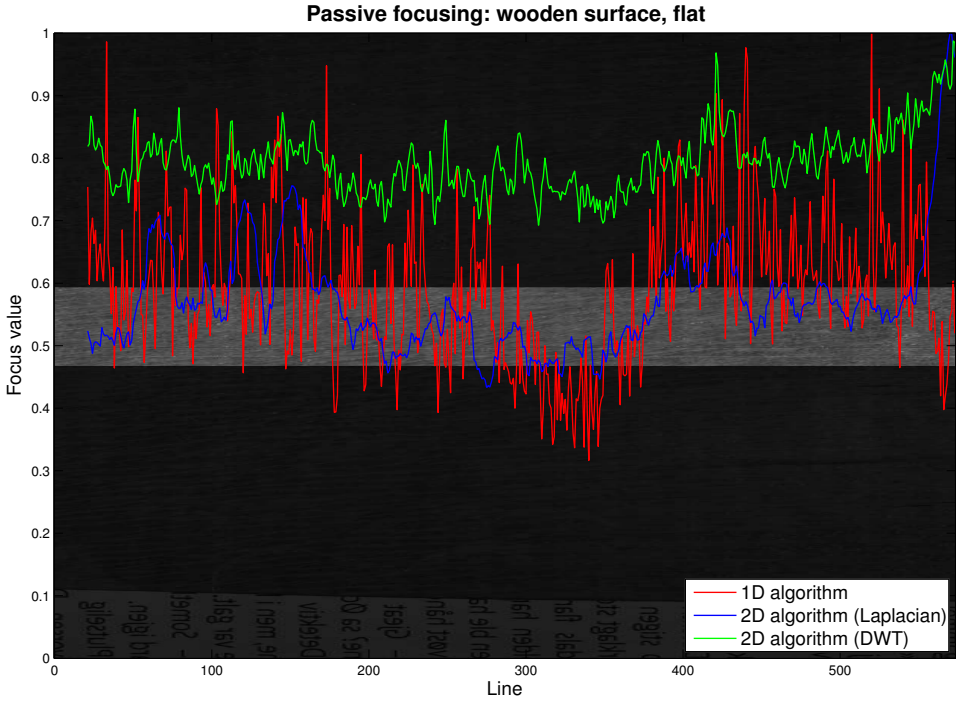


Figure 15: Focus measurements on a flat, wooden surface. The lens position relative to sample is held constant, at the focal length. Only a window in the middle of the image is used in the focus calculations, the rest is toned down.

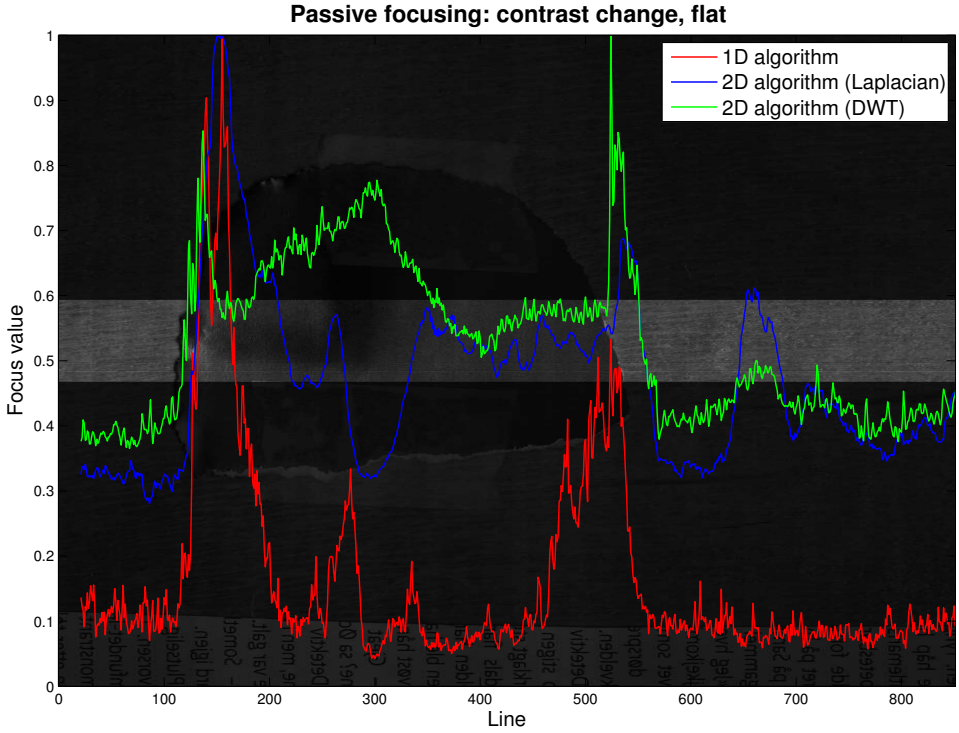


Figure 16: Focus measurements on a flat, wooden surface with a piece of colored paper generating contrast in the image. The lens position relative to sample is held constant, at the focal length. Only a window in the middle of the image is used in the focus calculations, the rest is toned down.

distinguish between contrast changes in the images and loss of focus quality, due to contrast changes influencing the algorithms to a large extent. This is further encumbered by the noise in the data, inflicting delays if any certain measurement of actual focus quality is desired.

Limiting ourselves to homogeneous samples only, a continuous contrast measurement focusing system might be possible, but one can question the quality of the achieved focus one would obtain with such a system. Further tests would need to be conducted. For an algorithm in such a system, the 2D laplacian is recommended due to high noise tolerance, ease of implementation and excellent speed.

5 Active autofocus

5.1 Description of underlying problem

Based on the conclusion from the passive autofocus tests, an active solution was sought out. Several solutions were considered to perform the distance measurement. Laser based systems were found as the optimal solution, due to supreme accuracy, response time and localization.

The proposed active autofocus system rely upon the process explained in section 2.4.2. The laser displacement sensor detailed in section 3.2 is used to measure the distance to target. By placing the laser sensor in a small distance ahead of the camera's FOV (given the scanning direction), one can make a distance profile in realtime while scanning. By following this profile, one can adjust the camera to ensure that the object is in focus throughout the image.

While the solution is simple in principle, there are several challenges to overcome in order to obtain optimal results:

- The laser spot being a certain distance in front of the FOV causes two problems. The first is the offset between the measuring spot and the FOV. The measurements have to be delayed by this laser-to-FOV-distance, and any errors in the measurement of this distance will map the values at the wrong location in scanning direction, potentially resulting in worse focus quality. Measuring this distance can be cumbersome due to the camera being a line scanning camera, making the use of rulers more difficult as they cannot be seen in camera. This can be solved by clever and accurate calibration. The second problem is the "blind area" at start of capture. Prior to capture the system has no knowledge about the height profile in the area between the measuring spot and FOV. This can easily be solved by moving the camera backwards by the laser-to-FOV-distance and then profiling the area while moving the camera back to initial position.

- The translation stages provide only relative position given in a custom unit, steps, as detailed in section 3.2. Due to the laser sensor measuring in millimeters, an accurate mapping between steps and millimeters is required. Furthermore, the speed is given in an arbitrary linear scale, requiring a mapping of the speed to physical distance. These issues can all be solved by various automatic calibration methods, requiring no knowledge from the user about the system.

- The current setup requires the laser to be tilted at a slight angle as described in 3.2. This gives rise to two problems, one being reduced accuracy of the sensor. The other problem is that the measured distance does not correspond to the actual distance from camera to object. One possible solution would be to measure the angle and from this measure the actual height, but this can

better be solved by mapping the change in measured distance to the change in translation stage position in a calibration process.

- The limited speed of the translation stage, also detailed in section 3.2, will give a limit to what changes in object height can be compensated for within the given time constraints of a continuous scanning system. These limitations should be investigated in order to provide guidelines for avoiding out-of-focus images.

5.2 The proposed continuous autofocus system

The presented autofocus system is based on the system developed during the project work detailed in the introduction. This gives a complete focusing system, with both initial focusing and continuous focus adjustment during image capturing.

The most important features are:

- Automatic initial autofocus, with opportunity for manual adjustment.
- Continuous autofocus during image capturing.
- Two speed settings, normal (2.1mm/sec) and high (6.2mm/sec) giving an option to prioritize speed over quality for object with low displacement changes.
- Image preview during capturing for quality inspection[2].
- Plotting of sensor distance measurements and translation stage position for quality assessment.
- Detailed calibration dialog to aid proper calibration of the system.
- Controls for manual control of translation stages.
- Displays information about image stream.

Operating instructions for the system, along with troubleshooting tips, can be found in section 7.1 in appendix.

5.3 Implementation

5.3.1 Overview

The active autofocus system is developed in C++ as a standalone application, using the Qt framework. The implementation runs on Windows only, due to dependence on external, Windows-only libraries. The continuous focus function is, in principle, independent on data from the camera, and can run on it's own.

[2]Image preview works in normal speed mode only. This is due to the TCP/IP communication between the programs not being able to keep up with the large data size in high speed capture.

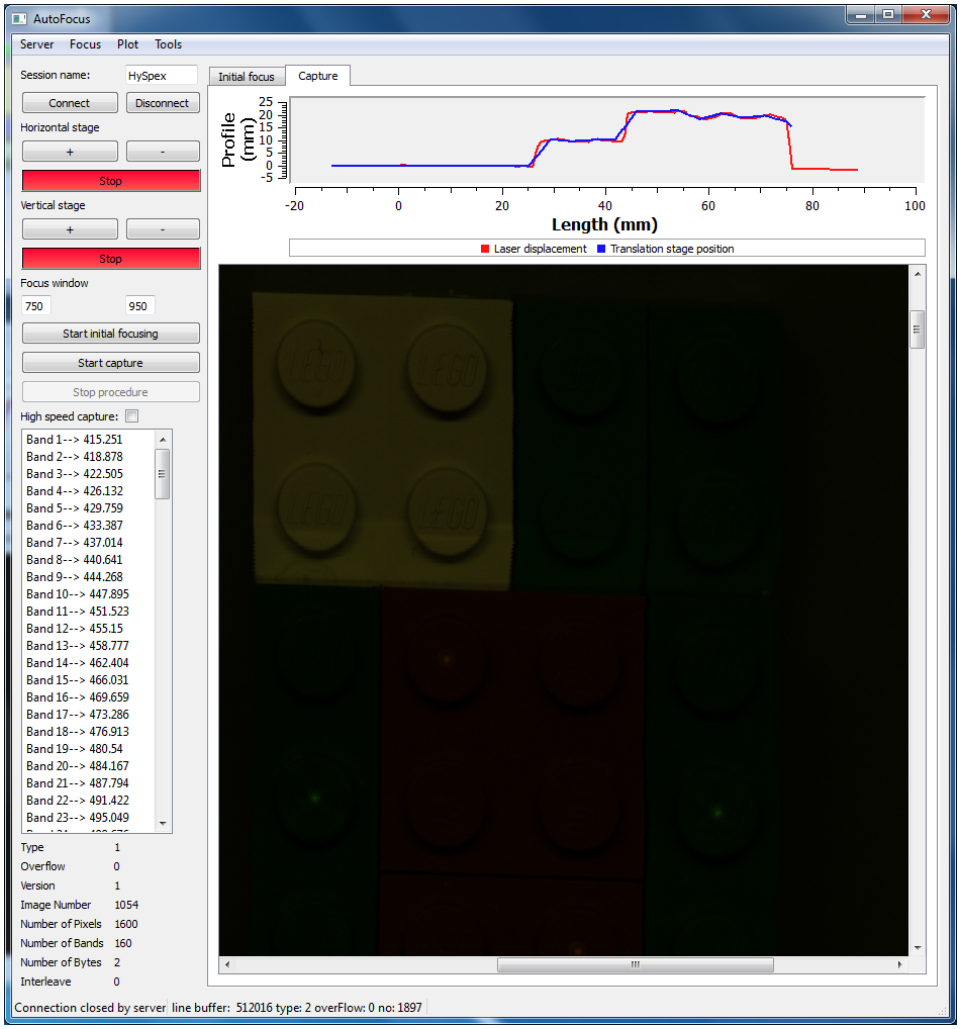


Figure 17: Image of the user interface for the autofocus system.

However, if initial focusing is to be used or a preview of the captured image is desired in the program, it can connect itself to HySpex AIR to retrieve image data. The system can also control the image capture in HySpex AIR by remote control commands over virtual serial ports. This means that the entire capture process can be controlled from the program, considerably easing the process for the user. Ideally the focus system would be integrated into the capture software, however, this was not possible for this work due to the unavailability of the source code.

An overview of the process is shown in figure 18. The process starts by the user adjusting the camera in correct position and angle. Next, the user performs

initial focusing by using the initial focusing function. It is crucial that the laser spot is positioned in the same area as the selected focus window for the initial focus. After acquiring correct focus position, the user may start a capture by clicking the "Start capture"-button. From here the system performs what is referred to as a pre-run, a process running prior to the actual capture procedure. The pre-run procedure is detailed in section 5.3.2. At completion of the pre-run, the continuous focus process takes over, continuously taking measurements and adjusting the camera if required. The continuous focus process is detailed in section 5.3.3.

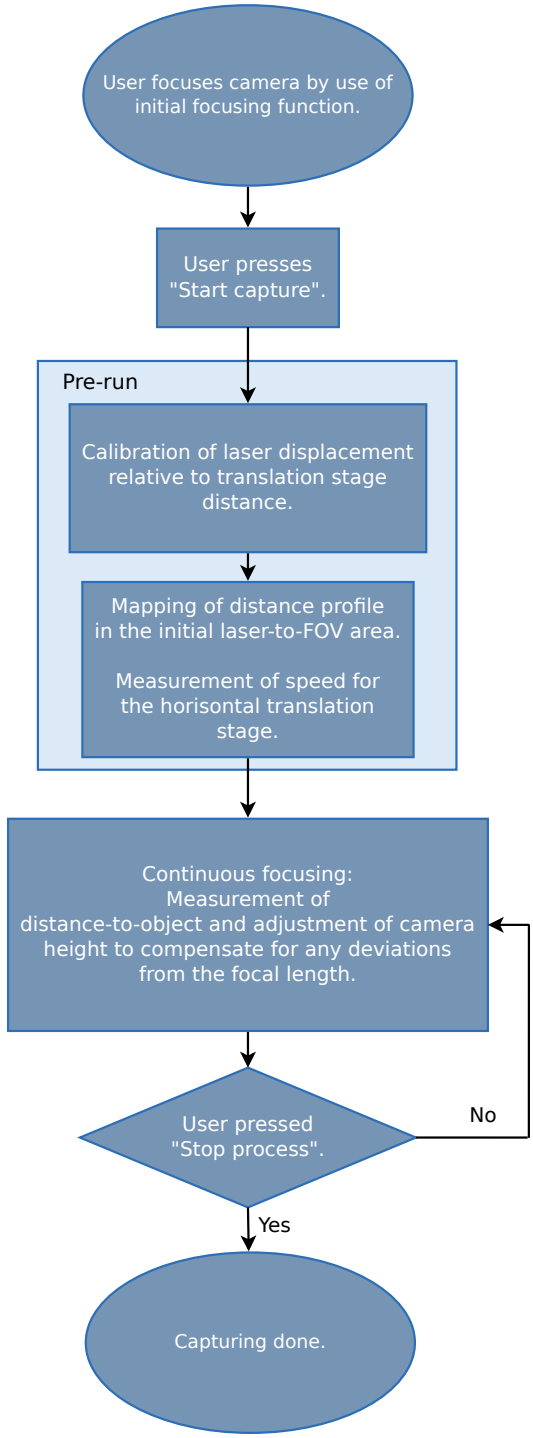


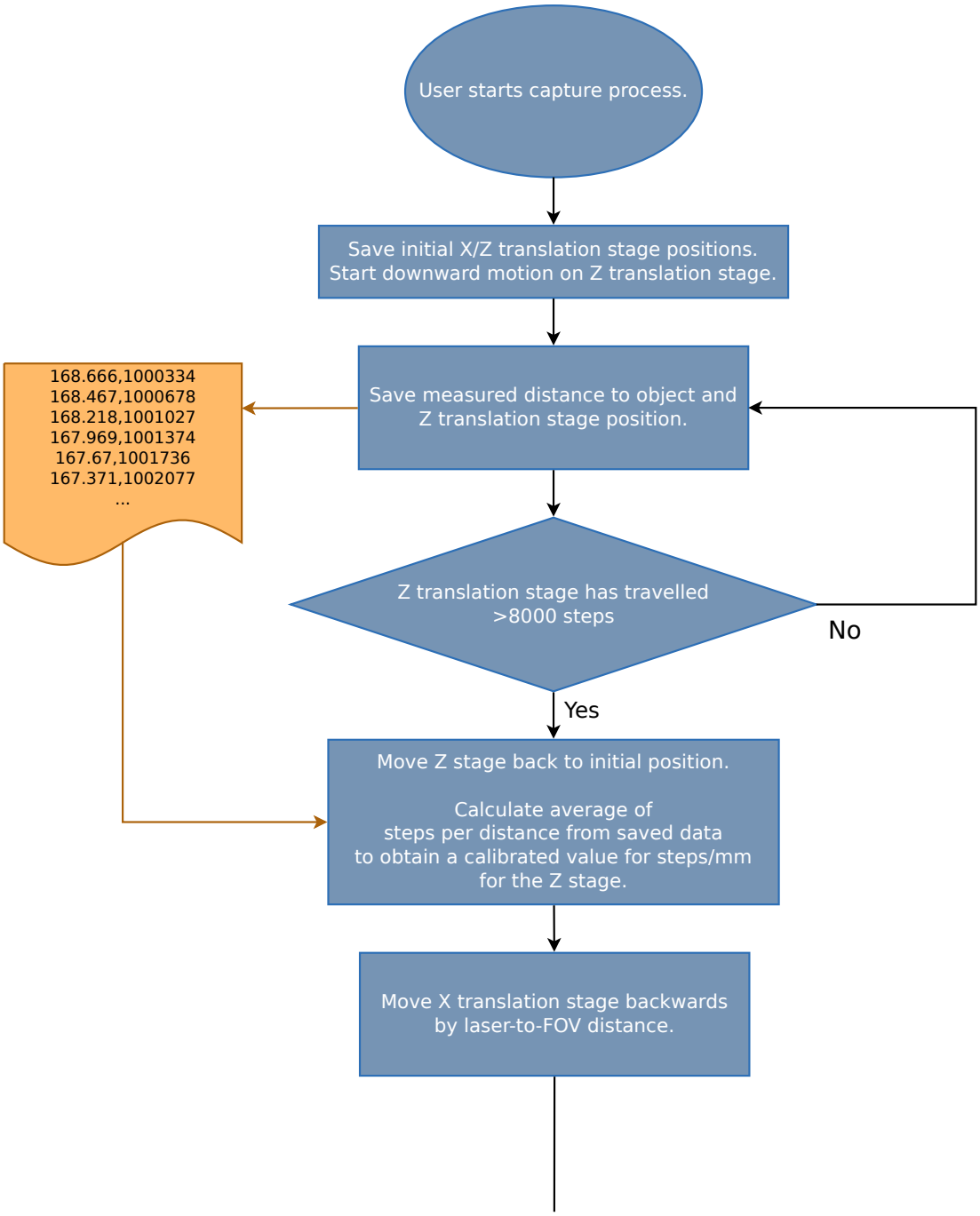
Figure 18: Capture procedure overview.

5.3.2 Pre-run

The purpose of the pre-run is to collect information about parameters in the system needed before performing an image capture. In short, the process consists of the following steps:

1. A calibration of the steps per mm for the vertical translation stage is conducted, using the laser to measure the distance to target. This process is detailed in section 5.4.2.
2. Measure the reference focus distance by moving the laser to the camera's initial FOV.
3. The topography in the area between the laser spot and the camera FOV is mapped and saved.
4. The speed for the horizontal translation stage is measured.

The full process is detailed in figure 19.


User starts capture process.
Save initial X/Z translation stage positions.
Start downward motion on Z translation stage.
Save measured distance to object and
Z translation stage position.
168.666,1000334
168.467,1000678
168.218,1001027
167.969,1001374
167.67,1001736
167.371,1002077
...
Z translation stage has travelled
>8000 steps
No
Yes
Move Z stage back to initial position.
Calculate average of
steps per distance from saved data
to obtain a calibrated value for steps/mm
for the Z stage.
Move X translation stage backwards
by laser-to-FOV distance.

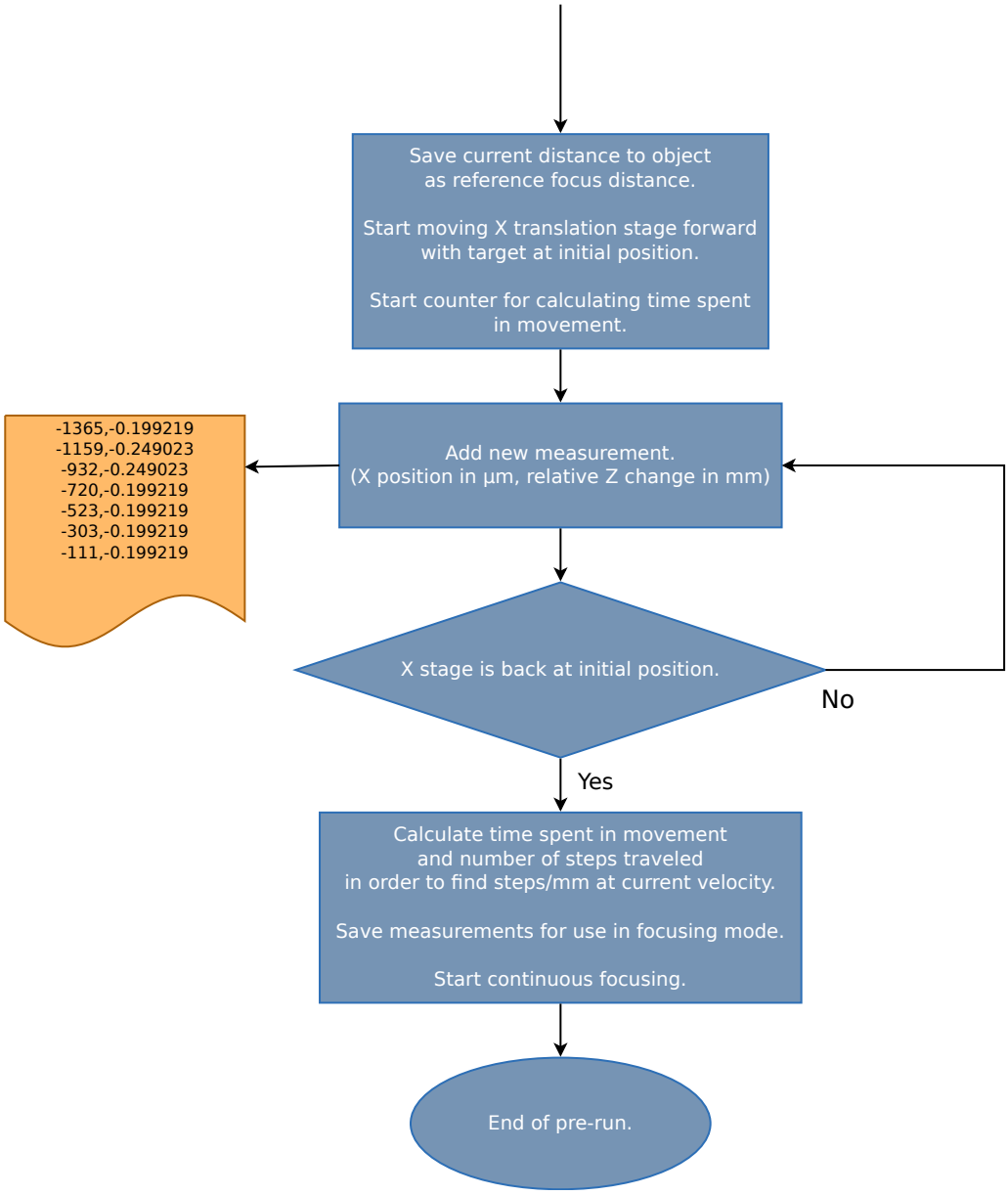


Figure 19: Pre-run procedure.

5.3.3 Focus mechanism

The focus process is shown in figure 20. A refocusing period is defined, and within these refocusing periods measurements are constantly read out from the sensor and added to a stack. Every time the translation stage has traveled this distance, an adjustment of the camera is performed using the latest measurement on the stack. The refocusing period is set to 3mm, a number considered balanced between accuracy in scanning direction and giving enough time for performing adjustments. Furthermore, an optional condition is to add a threshold for when to perform an adjustment. Due to the noise from the translation stage, making constant adjustments at slightly varying speeds can be a nuisance. A threshold of 0.5mm is therefore added to the code, being too low to affect the focus quality and considerably lessening the adjustment rate on flat surfaces. This condition is enabled in the implementation. Furthermore, another threshold is added when making adjustments when the translation stage is currently running. The stage will pause slightly when issued a new command, therefore we do not want to interrupt it if is running and the desired position is within 0.5mm of the previously issued one. The speed for the measurements is calculated using the formula found by the calibration procedure (cf. section 5.4.2).

Pseudo-code for the addMeasurement() function is given below:

```
function addMeasurement()

  // Get change in distance to object, relative to initial recorded
      distance from sensor.
  current_distance = laserSensor->getDistance()
  IF current_distance is valid
    distance_change = current_distance - initial_distance
  ELSE
    exit function
  ENDIF

  // Get current position for the translation stages
  xstage_position = XTranslationStage->getPosition()
  zstage_position = ZTranslationStage->getPosition()

  // Get position at laser point (in steps) relative to initial
      position, and convert to micron
  xposition_in_um = ((xstage_position - xStageInitialPosition) /
      XSTEPS_PER_MM + LASER_TO_FOV_DISTANCE_MM) * 1000


  // Calculate position to which the Z-stage should move in steps,
      given measured distance change
  distance_change = current_distance - distance_at_focus
  zfinal_position = zstage_position + distance_change *
      CALIBRATED_Z_STEPS_PER_MM

  // Add positions to stack
  distanceValues.add(xposition_in_um, zfinal_position)

end function
```

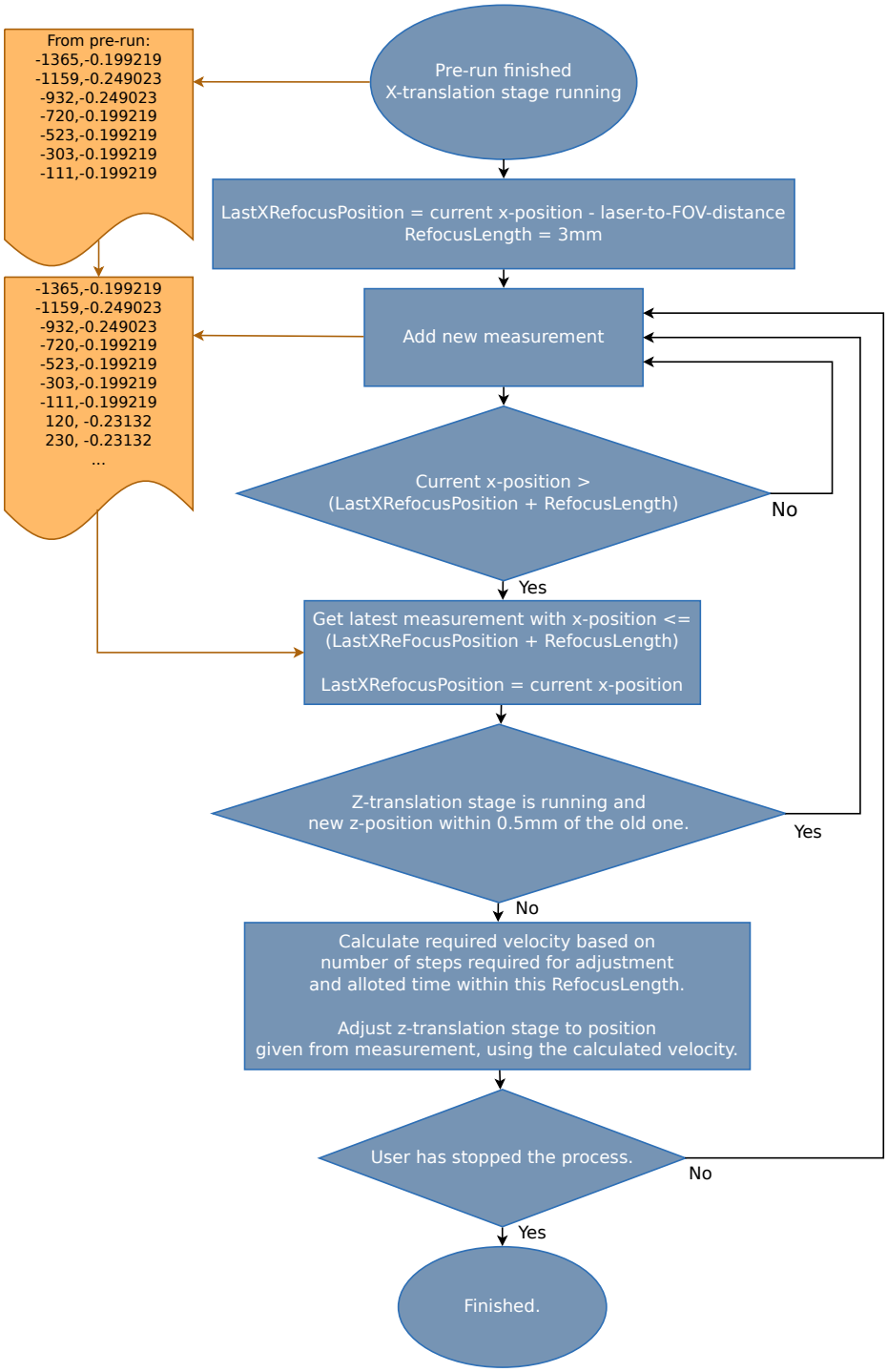


Figure 20: Continuous focus procedure.

5.4 Calibration

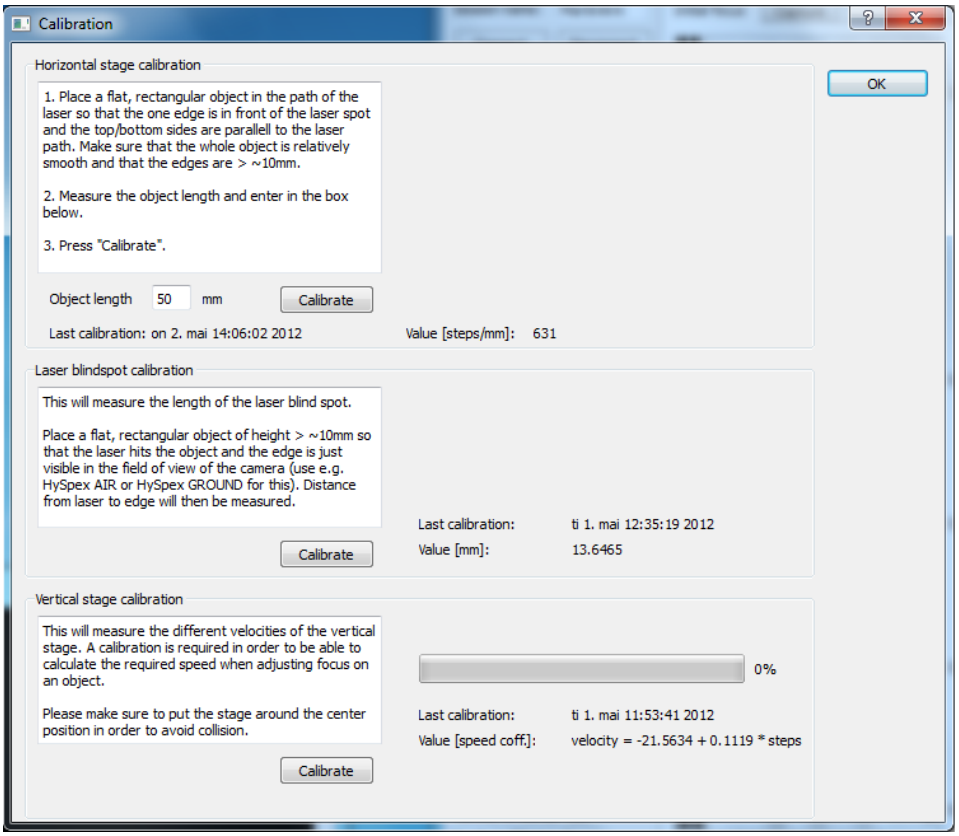


Figure 21: Image of the calibration dialog. The dialog guides the user through the calibration routines and shows last calibration values.

There are several calibrations needed in order for the system to function properly. While it is possible to do a one time measurement and hard code the values, it is desirable to give the flexibility to change equipment, yet keep things simple for the user. Consequently, various methods were invented to ease these measurements. Some of the calibrations are entirely automatic, while others require minor preparations. An image of the calibration dialog from the program is shown in figure 21.

5.4.1 Horizontal stage calibration

There are two parameters needed for the horizontal translation stage, steps per mm and steps per second. A new calibration is needed when changing the translation stage to another model.

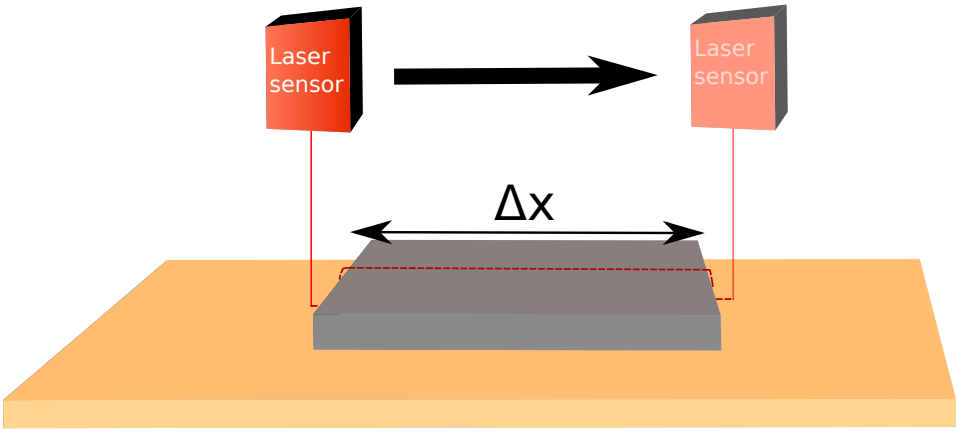


Figure 22: Calibration procedure for measuring steps per mm for a horizontal translation stage. The number of steps traveled is measured by system, Δx is measured manually by user. The laser path and profile is outlined in red.

Steps per mm is user calibrated. First one takes a flat, rectangular object and measures it's length, e.g. by use of a caliper. The length in mm is entered into the calibration dialog and the object is placed in front of the laser spot, see figure 22. The user then clicks the "Calibrate" button and the program performs the calibration. By using the laser to measure the start and end points of an object by height displacement, one can find the desired value by dividing the traveled steps by the object length measured by the user. See section 7.8.2 in appendix for the code performing this procedure. It is important that the object is placed parallel to the laser path. In other words, the laser has to travel the same length as is manually measured.

Steps per second is measured during the pre-run period. As the stage moves backwards as described in 5.3.2, a counter keeps track of the time spent in movement. At reaching the destination, the number of steps traveled is divided by the time spent. This procedure takes place at every image capture and is fully automated.

5.4.2 Vertical stage calibration

For the vertical stage, there are two relevant parameters. In order to be able to adjust the camera to correct position based on the measurements from the sensor, the steps per mm is crucial. A calibration of this value is needed before every image capture. Furthermore, to adjust with proper speed, a measurement of the different velocities is required. This process should be run when changing the

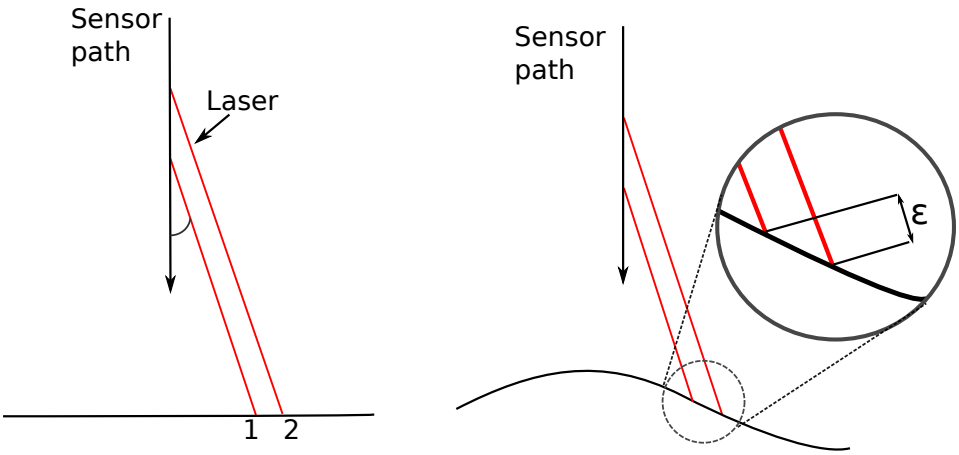


Figure 23: Challenges connected to calibration of steps per mm for the vertical translation stage by use of a tilted laser displacement sensor.

translation stage to another model.

Steps per mm is calibrated by using the laser displacement sensor. The calibration process, a part of the pre-run as previously detailed, consists of moving the sensor downwards while continuously measuring the distance to the object. By measuring the difference in steps for each difference in distance, one obtain the number of steps per mm. This can be averaged over several measurements. Ideally, if the sensor is positioned perpendicular to the surface, the measured values are directly translatable to camera distance relative to object. However, given a tilt of the sensor, knowing the camera height would depend on knowledge of the angle of the sensor. See left example in figure 23. This angle is not trivial to measure, especially not with a high accuracy. Hence, it is desirable to be independent of the sensor tilt. This can be achieved by using the measured values directly in the system, with no conversion. If the system is focused at the beginning, knowing the exact direct camera-to-object distance is not important, but rather use the initial measurement as the "focal length" to follow.

Keeping the laser tilted give rise to another potential issue. The right drawing in figure 23 show a situation where the laser will measure at an undesirable location. Due to the vertical movement, the measure spot in the horizontal plane will move. If the object is not flat, a distance difference, indicated by ϵ in the figure, will impact the measurement. As long as the sensor is tilted, this situation is not avoidable. The larger the tilt, the stronger impact. Consequently, it is highly recommended to keep the laser perpendicular to the object for optimal performance.

The **velocity calibration** is performed by moving the stage at different

velocities and measure the time spent at traveling a given distance. The system is calibrated at speeds ranging from 100-1000, at steps of 100. The resulting data is analyzed by linear regression, and a expression for the required velocity, y, needed to travel x number of steps given 1 second is produced.

5.4.3 Laser-to-FOV distance calibration

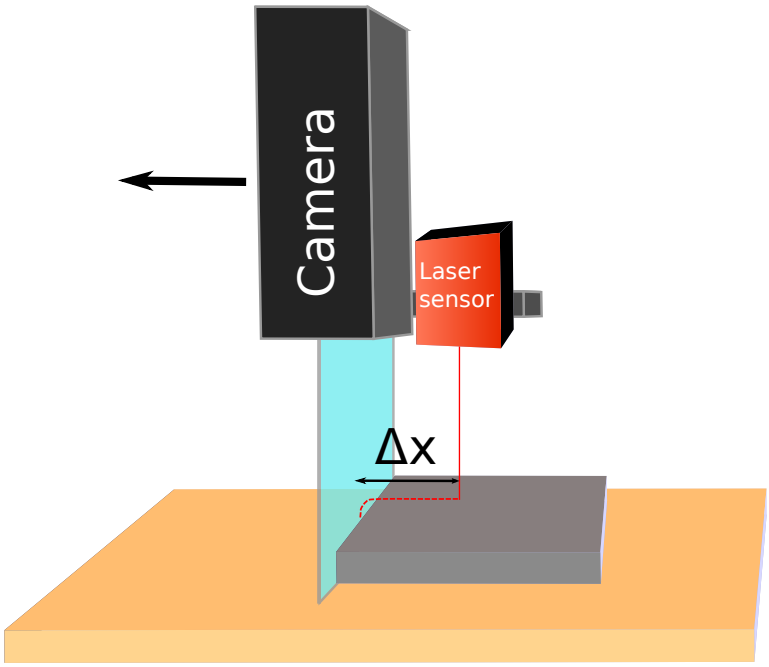


Figure 24: Calibration procedure for measuring laser-to-FOV distance. The object is placed alongside the FOV, indicated in light blue, and the laser signals when the end of the object is reached. The length, indicated by Δx, is then calculated by the system.

A similar procedure to the one for the horizontal stage is used in order to measure the distance from the laser spot to the camera's FOV. See figure 24. A flat, rectangular object is placed parallel to the FOV by use of HySpex AIR or HySpex GROUND, so that the edge is just visible in the image. By pressing the calibrate button, the system starts moving the laser sensor towards the initial location of the FOV. When it detects a displacement above a threshold of 1cm, the system stops the motion. The number of steps traveled are recorded, and by multiplying this value by the steps per mm, found in the previous calibration, one obtains the distance in mm between the laser spot and the FOV. A new calibration is needed if the laser sensor's position is altered or another lens is used.

5.5 Limitations

There are several limitations to this system, some intrinsic to the system while others are implementation specific.

The XY-resolution (i.e. object plane) for the laser sensor is not known. No measurements have been conducted to establish this limit. The laser spot size is 2.1-2.3mm, and hence we can only state that the resolution has to be lower than this number. In reality it is likely much smaller.

As mentioned before, high speed adjustment of the vertical translation stage causes vibrations in the camera, giving artifacts in the image. This is a physical limitation, giving a choice between no artifacts in the image or high speed adjustments.

The system only adjust focus along the scanning direction. Naturally any sufficiently curved objects (>2.5mm) in the direction perpendicular to this will gradually fall outside the depth of field of the lens, causing unfocused images. This is lens specific and is important to consider when imaging objects curved considerably within the FOV.

The sensor has a range from 60-260mm, meaning that it will only work with lenses having a certain focal length. It is possible to move the sensor up or down as required, but if using lenses with focal lengths smaller than 60mm + lens-to-camera-edge-length (keeping in mind that the lens is not placed at the edge of the camera), one has to start considering placement, accuracy and range of the sensor. Luckily, similar sensors with smaller ranges and much higher precision are available.

5.6 Results and discussion

For all the results, the laser is tilted 15^o.

5.6.1 Sensor accuracy

The optoNCDT 1302 sensor performed well on all tested surfaces, with the exception of a high contrast sample, shown in figure 25, and some areas of a stone sample. Among the tested surfaces were cardboard, wood, printed paper, skin, stone, colored plastic (LEGO) and ketchup.

The measurements for the high contrast sample are shown in figure 26. The plot shows distance to object while moving in the direction of the green arrow in figure 25. The peak at around 2.2 - 3 seconds is the black bar indicated by the red arrow. The ripples seen from 3 seconds and outwards is caused by the alternating line color.

The reason for this result is due to the difference in reflected light from the black and white areas. Since the contrast change areas are smaller than the laser spot, only a portion of the laser spot is reflected strongly. As the sensor use the diffuse part of the reflected light, this is of great importance for the measurement calculation. Running the autofocus system on this sample caused inaccurate results, as the system would compensate for the incorrect measured changes in the distance. This is an important result which shows the limitations for scanning printed text samples with high contrast. Other printed samples with images and lower contrast text samples did not exhibit the same behavior.

For the stone sample, inaccurate measurements were seen when hitting a deep valley at size comparable to the laser spot size. Here the effect is similar to the contrast change, although the underlying mechanism differ. Only a portion of the laser spot is reflected back at the sensor, due to slope of the valley walls and this gives inaccurate measurements or in worst case no measurement at all.

Further tests showed that the sensor would fail to find the distance if subjected to high intensity light. Failure to measure the distance is indicated on the sensor by a red LED, and in software an error code is given instead of the measured distance. This can happen if the light intensity from the light source is high and the laser spot falls within the lit area. The reason for this is that the sensor fails to distinguish the laser light from the specular light from the light source. Due to the polarizers, a high light intensity is required for a sufficient SNR in the images. The problem is especially pronounced if imaging objects with a high reflectance, such as ketchup or other liquids. Care has to be taken when adjusting the light sources and placing the laser sensor to avoid this from occurring.

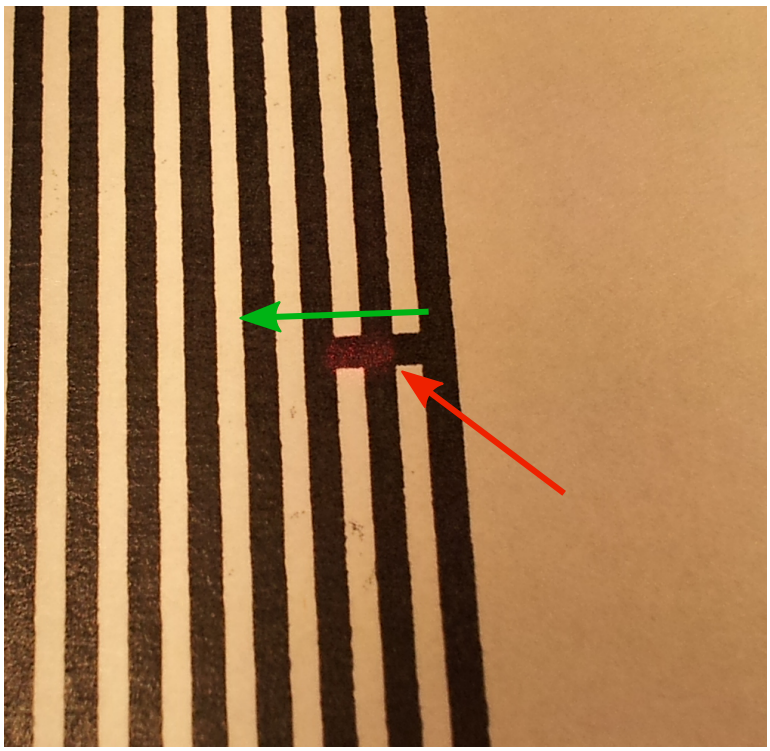


Figure 25: High contrast sample, demonstrating sensor inaccuracies. The green arrow indicates direction of motion, while the red arrow indicate the position for the peak seen in figure 26.

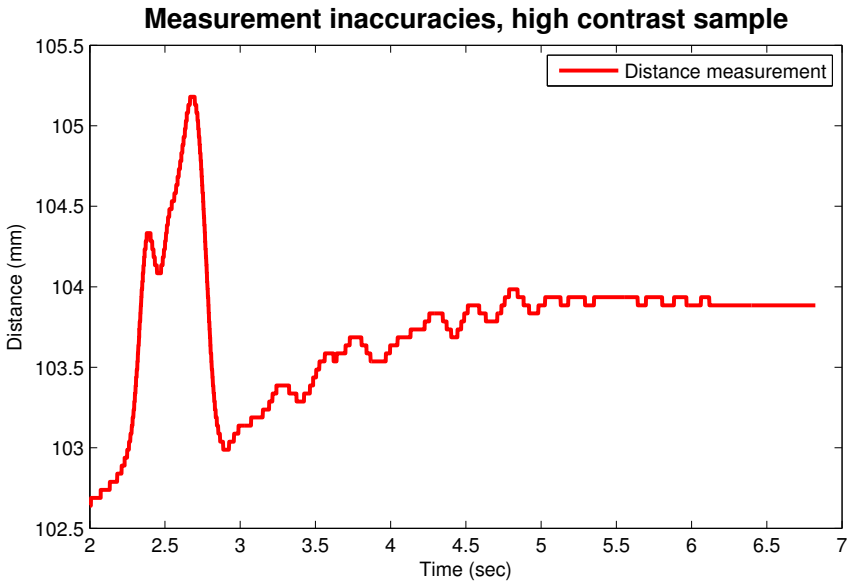


Figure 26: Measurement output for the distance measurement of the sample in figure 25. Sensor-to-object distance is constant.

5.6.2 Calibration accuracy

Type	Automatic	Manual	Deviation
X stage distance [steps/mm]	631	645	2.22%
X stage normal speed [steps/sec]	1323	-	-
X stage high speed [steps/sec]	3906	-	-
Laser-to-FOV [mm]	13.65	13.5	-1.1%

Table 6: Manually and automatically measured values for the horizontal translation stage and laser-to-FOV distance.

Object	Steps per mm	Deviation
Flat table	1555	Reference.
Arm	1729	11.2%
Stone	1667	7.2%

Table 7: Calibrated values for the vertical translation stage, based on measurements from the laser displacement sensor. Laser is tilted 15°.

Due to the system's high reliance on calibrated values, a proper evaluation of the calibration accuracy is important. The results are listed in table 6 and 7. Manual measurements were conducted where possible.

First, the horizontal translation stage distance has a deviation between the automatically calibrated and manually measured distances of 2.22%. The manually measured value were measured using a ruler and the laser spot. Aligning the center of the laser spot to 0mm, the stage was moved 30000 steps by software. The center of the laser spot ended at 46.5mm, yielding 645 steps/mm. This deviation is small and one also has to consider errors in the manual measurement. The likely cause for the difference is the laser spot size, being 2.1-2.3mm, giving room for errors in the XY-plane when measuring the number of steps traveled. While the calibrated value is used in various internal calculations, the system intrinsically operates in steps, and any calculated number by this value is normally calculated back by using the same value. The exceptions are calculation of the laser-to-FOV distance for displaying in mm in the calibration dialog, displaying distance in mm in plots and calculating the refocusing distance from mm to steps.

Of these values, only the laser-to-FOV distance is a crucial parameter for the focus quality given by the system. The calibrated laser-to-FOV value has a deviation of -1.1% compared to the manually measured value. The automatically calibrated value use the steps per mm for the horizontal stage for distance calculation, therefore the small deviation discussed previously for that value will also impact on this number. In operation, the calibrated value is calculated back to steps using the same calibrated steps per mm, in practice removing any effect from this value. The manual measurement were performed

by using a caliper, aligning one point into the FOV and the other at the center of the laser spot. It is important to note that both these values are dependent on the manual adjustment of the object into the camera's FOV. A deviation this small will not alone impact the focus quality, and the ease of use of the calibration procedure outweighs the negative effects given such a small error.

For the steps per mm values for the vertical stage listed in table 7, the calibration was performed by the program following the process described in section 5.4.2. The table served as a reference reading, as the measurement location problems discussed previously is avoided on this sample due to the flat surface. For the arm and stone samples, the calibrated values vary from the reference by 11.2% and 7.2% respectively. These are big numbers, given that they translate directly to errors in adjustment of the camera. The reason for the deviations is naturally explained by the fact that the surfaces are curved, giving different distance measurements to object for the same vertical displacement (cf. section 5.4.2). One can argue that this calibration should be a one time calibration, using a flat target as reference. There is, however, one disadvantage to this approach: If the sensor tilt angle is changed only slightly, a recalibration is needed. Given that the sensor might be knocked out of position, finding a flat, suitable object and doing a recalibration in the midst of the process of imaging patients is not desirable. Furthermore, it doesn't change the fact that the sensor should be positioned perpendicular to target for accuracy reasons as well, in which putting the calibration in the pre-run is the superior approach as a calibration is then performed anew, given the current surface properties, before each scan.

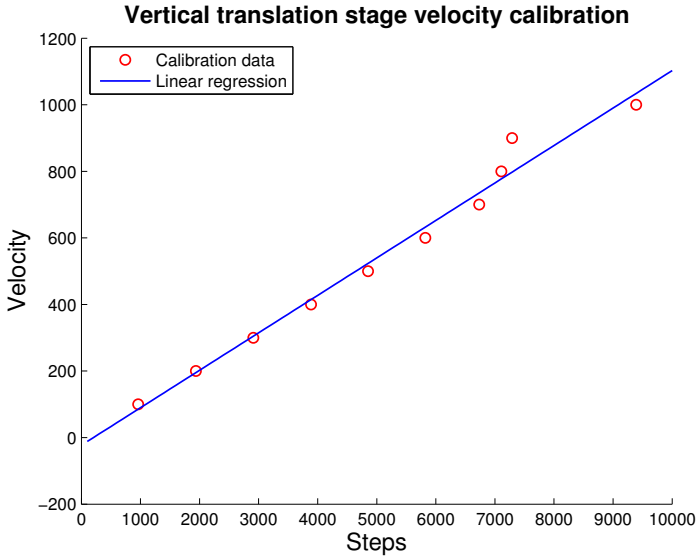


Figure 27: Calibration data for velocity calibration for vertical translation stage, with fitted, linear function. The data indicate number of steps traveled in one second.

Measurements for the vertical translation stage velocity calibration is shown in figure 27. The measurements are for differing speed settings from 100-1000 at 100 intervals. A function for finding desired speed value given number of steps to adjust for was calculated by the program using linear regression. The resulting calibrated function was

$$speed = -22.8728 + 0.112553 * steps$$

Although most values fit the linear curve, the values at velocities of 800 and 900 differ from the expected distance. The reason for this is unclear, especially since the 1000 value looks fine. Further investigations are needed in order to fully evaluate the impact of these deviations. However, the calibrations in general were found to cause too high velocities, in the sense that the camera reached it's target some time before starting next adjustment. The differences at velocities of 800 and 900 might have contributed to this, but as it happened on lower speed settings as well, it is considered unlikely. This is not desired, as we seek a motion as slow and fluent as possible. The effect might be related to the refocus period being longer than expected, as discussed later in section 5.6.5. Since the calibrations in itself worked fine, a manual adjustment using a multiplier of 0.75 was applied (chosen by trial and error) to the calibrated value before use. See code in section 7.8 in appendix. This made the camera hit target well in the alloted time. Still, based on these results, inaccurate speeds settings by the system in the 700-1000 range should be expected. The system keeps track of desired focus position regardless of the velocity, but it can cause the system to reach focus position later than desired, reducing the focus quality in an area.

5.6.3 Sensor influence on image

In order to ensure that the images are not influenced by the laser light, a simple test was conducted. The camera was adjusted to focus position, directed at a homogeneous, white object. The laser was positioned 13.5mm apart from the FOV. Four sample images were captured, with the combination of switching the laser and light on and off. The camera position was constant, hence the images have one spatial and one time dimension. Visual inspection of the images showed no influence. To reduce the noise, a MNF-transform were applied to the images, and an analysis performed. The MNF-transform is a common tool in analyzing hyperspectral images [9]. The laser was only visible in the second band in the transform, and only with the light source switched off. See figure 28.

This result is as expected. Laser light has a high intensity, hence, with no other light sources it is no surprise that the camera picks up some of the reflected light, albeit weakly, a distance apart from the FOV. However, with the light source turned on, the intensity of the reflected laser light is far too low to have an effect on the image. Therefore we can state that the sensor, if positioned a distance sufficiently far away from the FOV, can safely be used in the system from an image influence perspective. Further tests would be required to establish a limit for this distance.

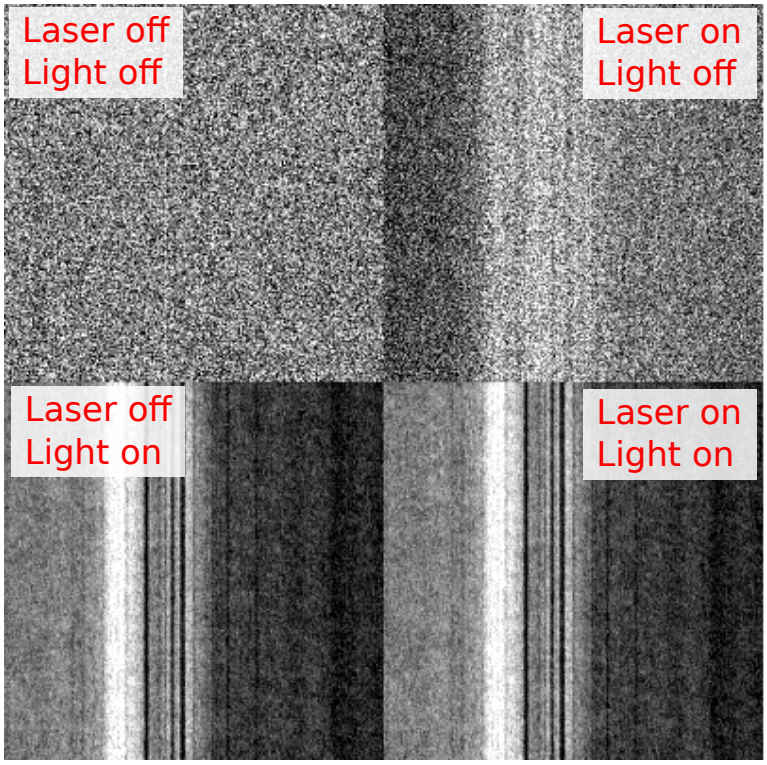


Figure 28: MNF analysis, showing MNF band number 2, of the laser light influence in an image. Only the area around the laser spot is shown.

5.6.4 Overview for the focus tests

Several focus tests were conducted for the system. Before starting an analysis of the tests, it is important to discuss the common parameters for the tests and establish what can be read out from the results, and what can not.

First of all, any errors in the steps per mm for the vertical translation stage will not be evident in the plotted data from the system (shown in figure 29, 31 and 33). The reason for this is that any position for the vertical translation stage is converted to mm in order to be plotted alongside, and this conversion uses the steps per mm parameter. In other words, even if the graphs follow each other in an exact manner, the focus might still be bad quality if the calibration is wrong. This will naturally include inaccurate measurements from the sensor itself. What the graphs do tell us, however, is how the system is able to respond to the changes in topography. Using any deviations in the graphs in comparison to the depth of field for the lens should therefore serve more as a guidance for further manual inspection of focus quality, rather than a numerical analysis.

Secondly, all the focus tests showed refocusing lengths of 4.1-4.3mm. This is 36%-41% higher than the set refocusing length of 3mm. It is uncertain why this happens, but mistakes in the programming is a possibility. Calibration should not influence this reading, as mm in the plot is an internal value using the same steps per mm conversion as the translation stage. If the calibration is off by an offset, so should the hard coded 3mm value be by the same offset when translated to steps, but still display as 3mm in the plot. This discrepancy is worth investigating in further work on the system.

5.6.5 Focus test: Arm with fake wound

The primary target for the system is imaging of skin with wounds, as mentioned in the introduction. While the system could not be tested on an actual wound due to limited supply and arrangement difficulties, a fake wound on an arm served as a testing sample. The wound consisted of ketchup and mayonnaise, with crafted valleys and craters to resemble the topography of an actual wound. The ketchup and mayonnaise combination is liquid and share some of the color characteristics for a wound. Furthermore, the arm was lifted about 2cm at the elbow in order to make the test more difficult for the system. The arm was therefore considerably sloped compared to the horizontal camera axis. In practice the camera path would be adjusted parallel to the arm. Three runs were conducted, one with normal scanning speed, one with high speed scanning enabled and one without the focusing activated, the latter serving as a comparison. Excerpt from the full scans are shown in figure 30. Data from the system are plotted in figure 29. Calculation of the angle of the slope from this data give 17°, relative to the horizontal axis.

The substantial slope of the arm is seen in the plot for the non-focused sample in figure 29. Looking at the normal speed run, the camera follows the arm exactly, except for a minor case in one spot. At around the 90mm position there is a small bump in the wound. The largest distance between the translation stage and distance measurement around this position is ~1.5mm, being within the DOF, but it requires further inspection as explained in 5.6.4. The log file show a 4.53mm adjustment at a velocity of 449 at 92mm, meeting it's target position. The reason for the failure to follow the height change exactly is the refocusing length being too long in this case. A smarter analysis of the collected samples could help in this regard. A dynamic refocusing period based on analysis of the collected samples might work better and should be investigated. The deviation at the same position increases for the high speed sample, being ~2.2mm, still within the DOF. The log show a desired adjustment of 5.35mm at 92mm, with a velocity of 1572. This is over the allowed speed for the system, meaning that the adjustment was made with an actual velocity of 1000. Consequently, the system could not keep up with the displacement, but this is an imposed restriction.

Looking at the images, they look focused in the desired area (the wound) throughout the entire scan. The focus quality falls quickly to the right of the wound. This is natural, due to the high displacement change where the arm curves downwards, putting this area outside the depth of field. The focus system only adjust focus along the scanning direction, using the laser path for measurements. There is no discernible difference between the two speed modes, even in the problem area at around 90mm, being the bottom of the wound. This means that the system can handle some deviation from the distance measurement, due to the depth of field of the lens. Unfortunately, it is hard to give an exact actual deviation for the lens from it's focal length at this position, we can only state the relative number of ~2.2mm. Inspection of the problem area at the end of

the wound is difficult as the the contrast is low, but the normal speed scan looks marginally better, although both can be considered to be in focus. The overall comparison of the non-focused image to the focused ones clearly demonstrate the potential of the system, showing that the system works as intended on this sample at both normal and high speed setting.

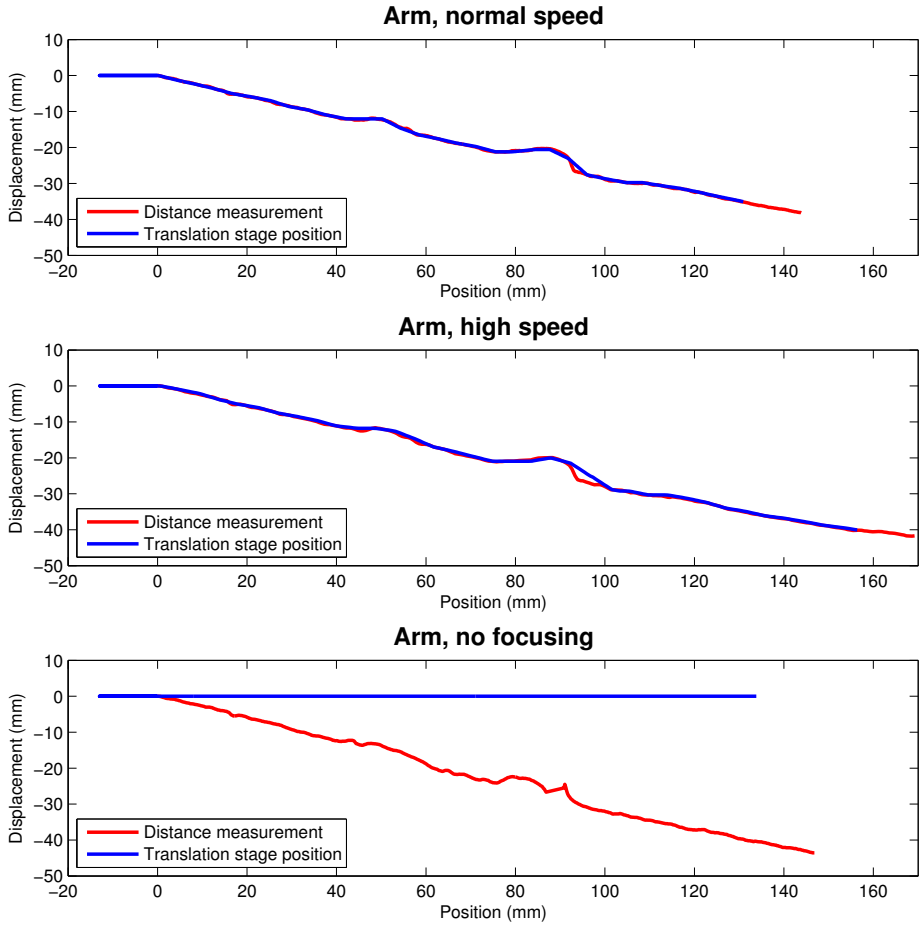


Figure 29: Distance measurement and translation stage position data plotted for the arm sample. Data are for full scans.

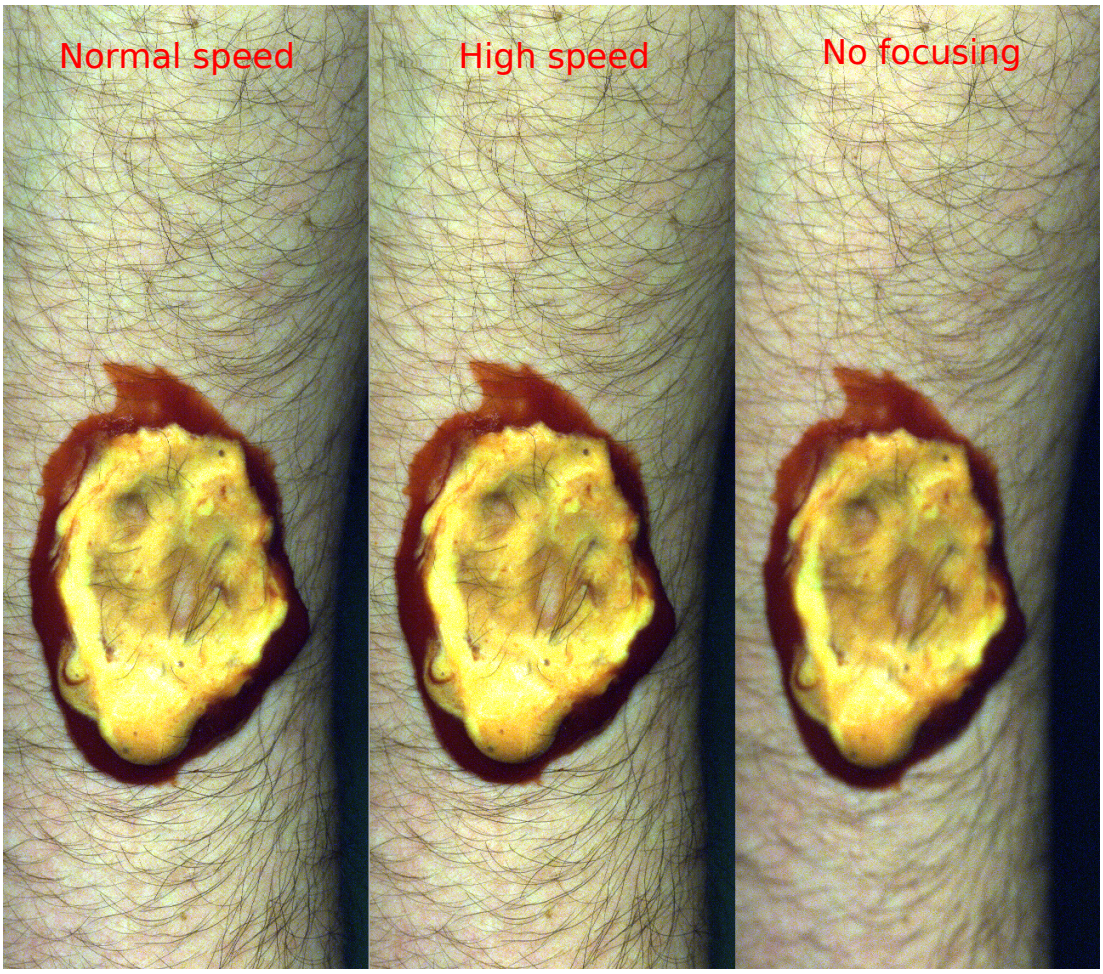


Figure 30: Excerpt from focus test on an arm with a fake wound, consisting of ketchup and mayonnaise. Scan direction is downwards in image.

5.6.6 Focus test: Lego blocks

To further test the potential of the system, lego blocks were used to build a two step rectangular structure. Lego blocks have a base height of 9mm, with the knobs adding an additional 1.8mm. The first step is hence 9mm and the second is 18mm above the ground. The camera was focused at the table a distance before the structure, from where the capture was initiated. Excerpts from the full scans are shown in figure 32. Data from the system are plotted in figure 31.

The measured total length of the lego blocks is 50mm in the plot, while actual measurement is 48mm. This is an deviation of 4%, showing that the calibration is a little off. Yet, keeping in mind the 5cm traveled distance, this is within what one can expect from the calibration accuracies discussed previously. As the system is not dependent on previous values, it will not get less accurate over time, hence this 4% deviation is not important quality-wise in itself.

Looking at the result, it is clear that the abrupt changes in height pose a challenge for the system. The system reaches all the targets within the alloted time (cf. plot and the log file in section 7.5.2). This puts weight behind the manual adjustment of the velocity calibration. Furthermore, the translation stage follows the distance measurements well on the flat areas on the plateaus. For the large displacements, the translation stage starts movement a distance before, having time to adjust to the proper height before reaching target, but also leaving the image out of focus in these areas. It is important to notice that changing where the adjustment starts can make the system reach focus position in time in one area, but this will make other parts less focused. Looking at the plots, the system seem to do a good compromise. The areas out of focus in figure 32, right before the steps, are hard to see as they are not well lit due to the structure blocking some of the light from one of the light emitters. An important visible effect, however, is that the straight lines of the figure looks bent, as the camera moves up and down. The structure geometry in the image is hence affected by the adjustments, which is not desirable. This is unavoidable as long as the system does not stop the scanning motion for the adjustments. The effect is less pronounced in the high speed capture due to the smaller adjustment time, but considerable distortions from vibrations in the camera can be seen in the image, stemming from the higher speed movements. One can argue that the maximum speed is not optimized for this rig, but the influence will depend on how far the camera is adjusting. There might also be cases where quick adjustments is preferable compared to having an image without distortions in the adjustment areas. If the object to be imaged is still, however, it would in this case be better to reduce the refocus length, rather than using high speed capturing. Large adjustments over short time will change the physical pixel relation considerably, an important aspect to consider if doing measurements involving distance or area calculations in object plane.

The focus quality in the focused areas looks good. Comparing these areas

to the non-focused image shows the large difference in quality.

As mentioned in the introduction, only continuous scanning is evaluated. This lego sample is a good example of an object which could benefit from stopping the capturing before adjusting. If primary target is samples with big gaps, it can be worthwhile to inspect how "pause and adjust"-scanning impacts the images.

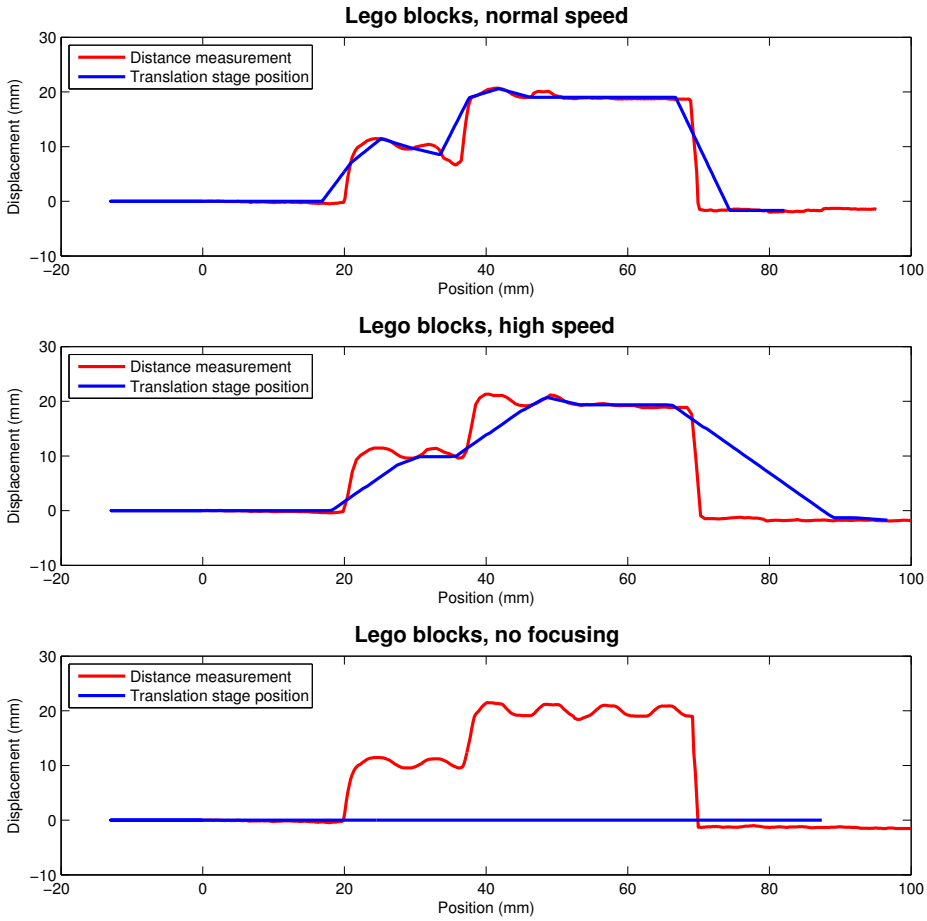


Figure 31: Distance measurement and translation stage position data plotted for the lego sample. Data are for full scans.

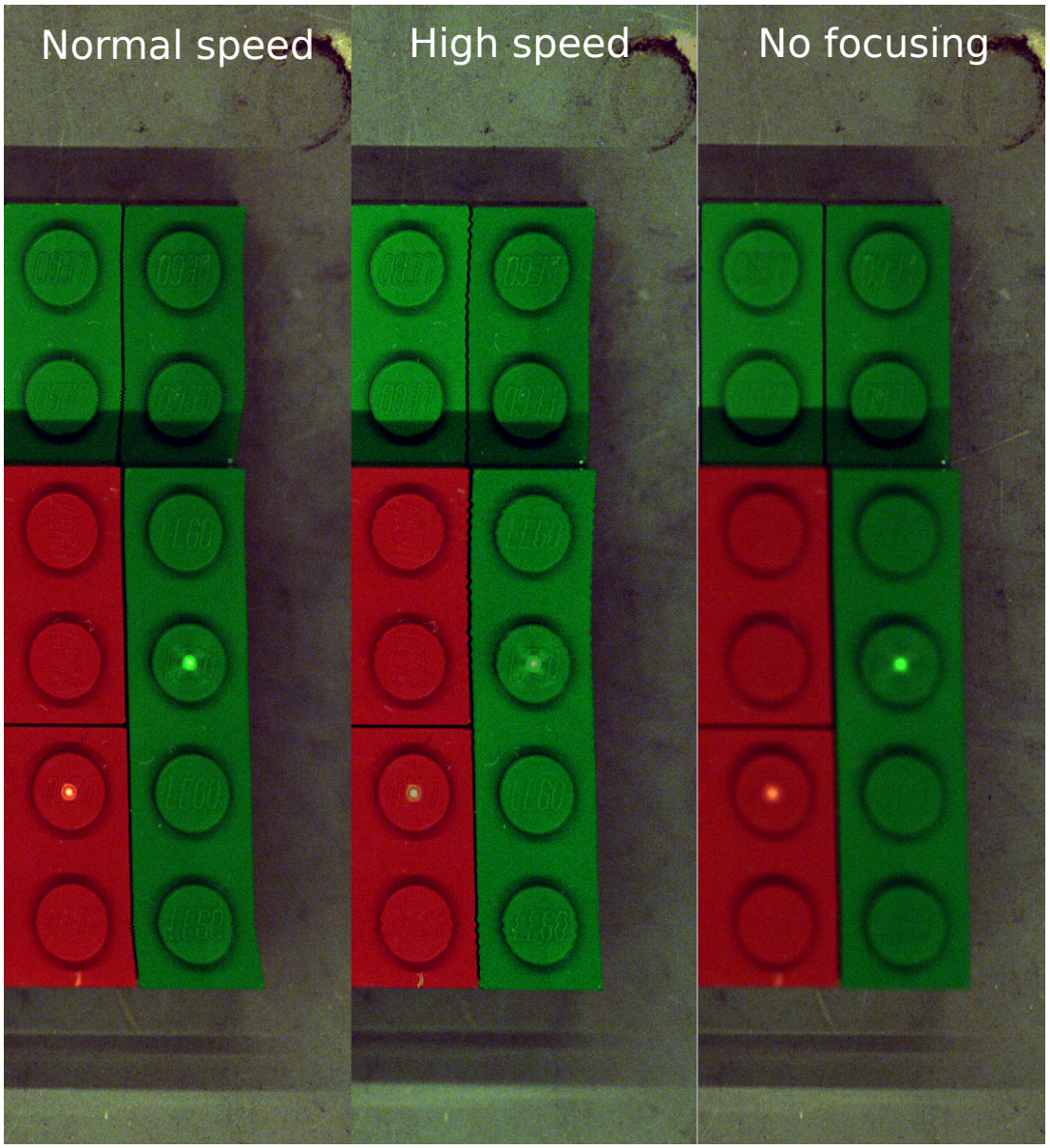


Figure 32: Excerpts from focus test on lego blocks with two height levels. Scan direction is downwards in image.

5.6.7 Focus test: Stone

Geology and stone analysis is an interesting application for hyperspectral imaging. Stones are rarely flat, hence continuous focusing is an important component in the performing an analysis. The stone used is shown in figure 36 in the appendix. Initial focus was done on one end of the stone and the rest of the stone was captured including the adjustment down to table-level. Excerpts from the full scans are shown in figure 34. Data from the system are plotted in figure 33.

The resulting graphs in figure 33 are as expected. The system follows the curves of the stone exactly, given that the changes are sufficiently small. At the stone-to-table gap the system struggles, as also seen in the lego samples. An interesting observation is the difference in the steepness of the gaps from the stone to the table. For the normal and high speed tests the steepness is similar, but in the no focusing case the fall is close to vertical. One possible explanation is the difference in measurement position in object plane, due to the lack of adjustment of the vertical translation stage. Another possible explanation is that the laser measures the profile better as it is moving down, getting line of sight to new areas.

Inspection of the focus quality in the images is challenging due to the texture of the stone, but the "No focusing"-graph indicate that a difference should be visible in both the first and second half of the stone, with the lens meeting focus position right after the middle at the 30mm position. The largest difference in the first half is 3.54mm at a position of 12.26mm, possibly outside the DOF having considered the sources of errors and measurement considerations in 5.6.4. After the 30mm position, the difference is gradually growing from 0 at 30mm to 11.55 at 47mm, the position where the system starts having problems following the changes. Looking at the images, a clear difference is shown in the first half. The area overall is sharper, and is it especially visible in the yellow parts. 3.54mm in the system can therefore be considered outside the 2.5mm DOF range, not surprising as the number is 41.6% higher, above any expected measurement inaccuracy. A difference between the normal and high speed mode images is not visible. After the middle area, the focus quality is slowly becoming alike in the focused images and the non-focused one, before departing again as predicted by the graph. Near the end the difference is more pronounced, with the non-focused image clearly being out of focus. Both the focus adjusted images look good. However, at the end of the stone the high speed image has better focus, in contrast to the previous test. Looking at the graphs, it is hard to tell why this happens. If anything, the normal speed image should have better focus in this area, as both start the downward motion at around the same position and the normal speed has better time for adjustment. One can argue that the calibration errors calculated in 5.6.2 make the graphs show wrong position for the actual adjustment position relative to the physical position. However, the two images are not consistent with the graphs in this regard as the graphs are similar, but the images are not, hence

it is considered unlikely. Further investigations is needed in order to conclude on this phenomena. Nevertheless, it is clear that gaps in the topography higher than the DOF is a problem for the system, as is to be expected.

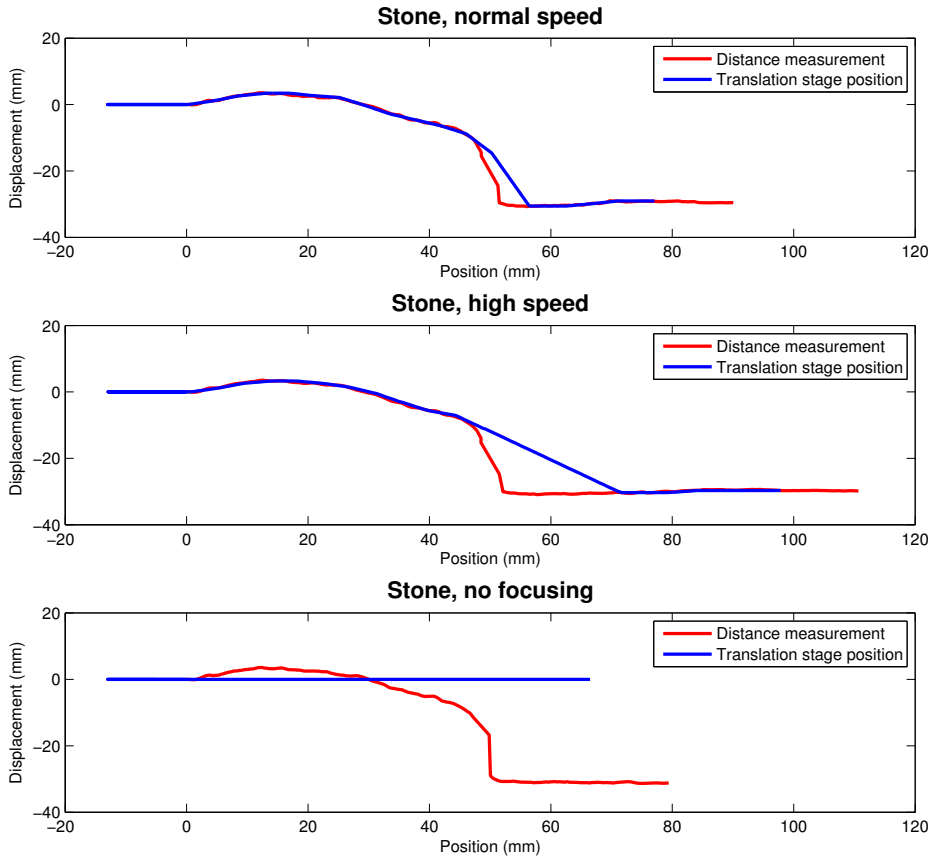


Figure 33: Distance measurement and translation stage position data plotted for the stone sample. Data are for full scans.

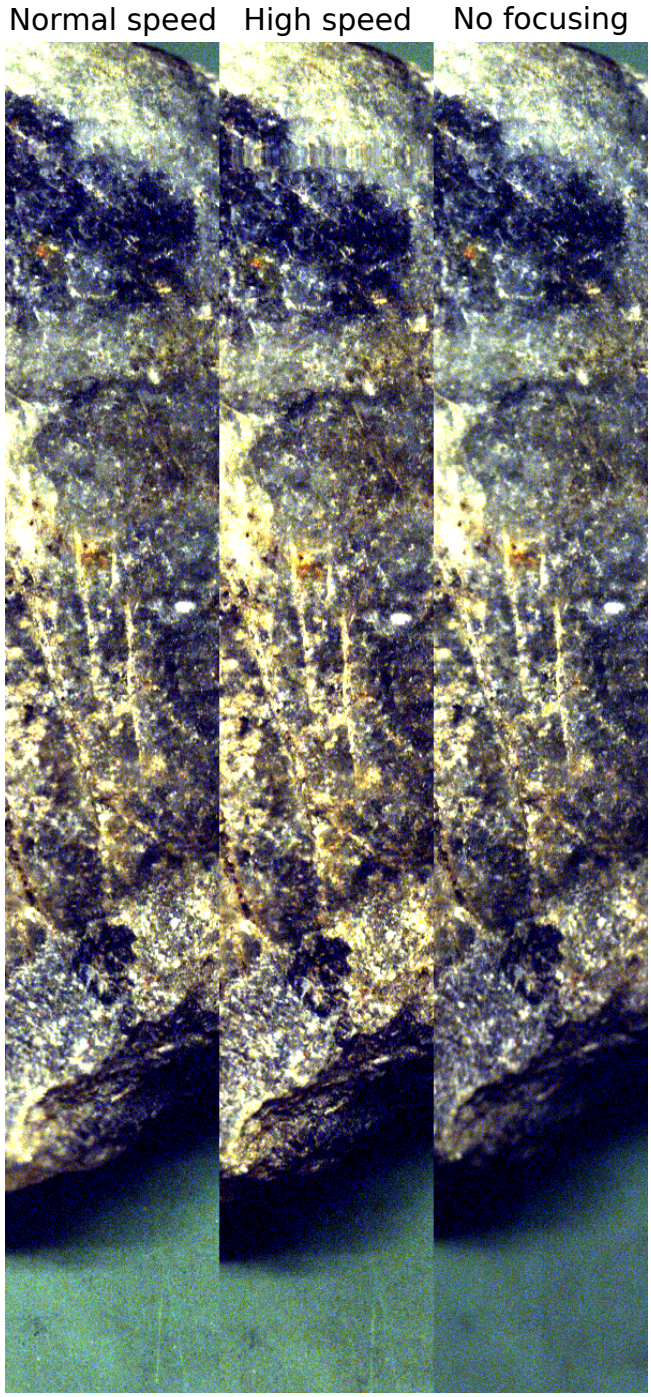


Figure 34: Excerpts from focus test on a stone. Scan direction is downwards in image.

5.7 System evaluation

Based on the results, the system functions as intended, giving great focus quality for slowly variating samples. In the light of the target area being skin and wound imaging, the system will likely perform well in this regard. It is also proven that the system can handle other situations, however, considerations must be made when imaging objects with a high topography variation. If scanning moving objects, like patients, high speed mode is recommended to lessen disturbances in the images from the movements. Otherwise, normal speed is the best option.

Most limitations are physical and setup related, especially adjustment speed and laser placement issues. The most pressing issues are the laser tilt and in the way the system determines when to adjust. The laser should be placed perpendicular to the object, to remove calibration errors. Furthermore, today's system is based on fixed length refocusing periods, while dynamic refocusing positions has a potential to work better. This will, however, require more complicated code as a part of evaluating the data for when to do optimal adjustments. This is hereby left as a possible improvement vector for the system.

Establishing a limit for when vibrations are disturbing the image quality is difficult, as seemingly both adjustment length and speed both affect the amount of vibrations. Looking at the log files, we can try say something quantitatively about the phenomena. For the lego sample, the normal speed image has visible artifacts in three places, corresponding to the 7.1mm, 33.4mm and the 66.7mm positions, adjusting with velocities of 639, 941 and 1883 (actual 1000) respectively. The 639 adjustment show much smaller disturbances. They all adjust for displacements higher than 7mm. The lower adjustment velocities show no problems. Looking at the high speed arm test, no disturbances can be seen, although it is adjusting at maximum speed over 5.4mm. In conclusion, adjustments over 5.4mm-7.1mm with a velocity over 600 can give disturbances in the image. Erring to the side of caution, compensating for gaps over 5mm in one adjustment, is not recommended if no disturbances is desired. Making the camera setup more robust or reducing the weight of the equipment will likely help in this regard.

To sum up we revisit the desired focus system qualities from the introduction:

- *Low to no requirement for operator interaction and training.* This requirement is fulfilled. No knowledge about focusing or cameras is required in order to use the continuous focusing function. Calibration of the system is also straightforward, with the bonus being that all calibration procedures requiring user interaction is one time only given no changes to the system.

- *Rapid focus recognition.* Due to the laser measuring ahead of the camera FOV, the focus recognition is instant and adjustment speed is only dependent on the system components. The implemented solution utilize the adjustment

capabilities of the system fully.

- *High reliability, regardless of object to be focused and lighting conditions.* This is an ideal, for any system there will be objects giving complications. The presented system works well under most conditions, and steps can be taken to improve the reliability on difficult samples. The system works without any light at all, but can experience problems at extremely high light intensities.
- *High accuracy.* The focus quality after adjustment is shown to be excellent, given proper calibration.
- *Low cost.* The system is dependent on a laser displacement sensor which costs money. Seen in context of the system as a whole, the cost is low.
- *If external components are required, these should have low weight and volume.* The laser sensor is small and weights little, using no considerable space and adding no considerable weight to the system. Furthermore, it is likely possible to incorporate the sensor into the camera itself, giving an optimal placement.

6 Conclusion and further work

Passive focus mechanism show little potential for continuous focusing systems, due the difficulties for the algorithms to separate contrast changes in the object from actual focus quality. Noise is also a limiting factor. For homogeneous objects, a passive focusing system might be possible, using energy of laplacian algorithm on two dimensional data, but the performance one can obtain is questionable.

Using an active autofocus method works well. The optoNCDT 1300 sensor has the required accuracy and measurement rate for providing measurements in such a system. Placing the sensor ahead of the camera field of view, and making a profile of the topography changes worked as intended, providing excellent focus recognition. The limits for adjusting the focus is largely limited by physical properties in the constructed rig. There is a limit to how fast the translation stage can move, and a high adjustment speed will cause vibrations in the camera, causing artifacts in the images. If no disturbances in the image is desired, compensating for gaps over 5mm within one refocusing period is not recommended on the current camera setup.

The invented calibration methods performed as intended, easing the calibration process while providing sufficient accuracy. It is strongly recommended to place the sensor perpendicular to the object (i.e. not tilted), in order to improve the accuracy and reliability. The sensor can fail to measure on some high contrast samples and if high light intensity hits the measurement area.

High speed mode is recommended when imaging moving targets with relatively flat surfaces, like arms, hands or legs. The higher speed reduces chance of patient movement affecting the image, while focus adjustment is adequate. Normal speed is recommended for still objects, due to the longer adjustment window.

The desired properties set out for the system is considered fulfilled. The system has low requirements for operator training and interaction, good reliability under most conditions, high accuracy and low cost. Focus recognition is instant, adjustments only being limited by translation stage properties. External components add no considerable weight or space to the system, and has potential to be included in the camera itself.

There are many improvement vectors for the system. Using dynamic refocusing periods can better predict when to start adjustment of the stage for optimal focusing. Furthermore, "pause and adjust"-scanning might be possible without image interference, and can improve the focus quality on demanding samples. The refocus period is shown to be off from the desired value, meaning there is a flaw somewhere in the system. This should be investigated in order to ensure the code and all it's underlying mechanisms are correct. Finally, the camera rig should be made more robust in order to handle adjustments with a higher speed.

References

[1] Scientific American. How do bats echolocate and how are they adapted to this activity? http://www.scientificamerican.com/article.cfm?id=how-do-bats-echolocate-an. Accessed 25 May 2012.

[2] C.Y. Chen, R.C. Hwang, and Y.J. Chen. A passive auto-focus camera control system. *Applied Soft Computing*, 10(1):296–303, 2010.

[3] M.S.T. Choi and A. Nikzad. Focusing techniques. *Machine vision applications, architectures, and systems integration: 17-18 November 1992, Boston, Massachusetts*, 1823:163, 1992.

[4] Wikimedia Commons. Focal length. http://upload.wikimedia.org/wikipedia/commons/8/8b/Focal-length.svg. Accessed 25 May 2012.

[5] R.G. Dorsch et al. Laser triangulation: Fundamental uncertainty of measurement. *Applied Optics*, 33(7):1306–1314, 1994.

[6] R. Fontana et al. Autofocus laser system for multi-NIR scanning imaging of painting surfaces. *Proc. SPIE*, 8084:808405–1–808405–9, 2011.

[7] N. Gat. Imaging spectroscopy using tunable filters: a review. In *Proc. SPIE*, volume 4056, pages 50–64, 2000.

[8] R.C. Gonzalez, R.E. Woods, and S.L. Eddins. *Digital Image Processing Using Matlab*. Dorling Kindersley, 2004.

[9] A.A. Green et al. A transformation for ordering multispectral data in terms of image quality with implications for noise removal. *Geoscience and Remote Sensing, IEEE Transactions on*, 26(1):65 –74, January 1988.

[10] isel Germany AG. Technical data, LES 4 with spindle drive.

[11] R.E. Jacobson. *The manual of photography: photographic and digital imaging*. Media manuals. Focal Press, 2000.

[12] J. Kautsky, J. Flusser, B. Zitová, and S. Simberová. A new wavelet-based measure of image focus. *Pattern Recognition Letters*, 23(14):1785–1794, 2002.

[13] E. Krotkov. Focusing. *International Journal of Computer Vision*, 1(3):223–237, 1988.

[14] R. Lang and A. Spray. VLSI word size and precision analysis of the discrete wavelet transform.

[15] Micro-Epsilon. Instruction manual optoNCDT 1302.

[16] D.B. Murphy. *Fundamentals of light microscopy and electronic imaging*. Wiley-Liss, 2001.

[17] Norsk Elektro Optikk. Hyspex prinsipp. http://www.hyspex.no/images/about_hypspec_img/HySpex%20prinsipp.jpg. Accessed 25 May 2012.

[18] Norsk Elektro Optikk. HySpex VNIR-1600 camera. http://www.hyspex.no/images/index/vnir1600.png. Accessed 25 May 2012.

[19] R. Poprawe. *Tailored Light 2: Laser Application Technology*. Springer, 2011.

[20] S.T. Seljebotn. Autofocus in hyperspectral imaging, project report, 2011.

[21] T. Skauli. Sensor-informed representation of hyperspectral images. In *Proc. SPIE*, pages 733418–733418–8, 2009.

[22] T. Skauli, FFI. Hyperspectral sensor technology. http://hyperinet.multimediacampus.it/images/Skauli.pdf. Accessed 25 May 2012.

[23] Standa. Standa 8MT175 specifications sheet.

[24] D. Stöbener et al. Distance measurements with laser triangulation in hot environments. In *Proc. XVII IMEKO World Congress*, pages 1898–1901, Dubrovnik, Croatia, 2003.

[25] L.D. Stroebel, R.D. Zakia, and I. Current. *Basic photographic materials and processes*. Focal Press, 2000.

[26] M. Subbarao and J.K. Tyan. Selecting the optimal focus measure for autofocusing and depth-from-focus. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):864–870, 1998.

[27] K. Tatsumi et al. Onboard spectral calibration for the Hyperspectral/Multispectral sensors. In *Proc. 'Hyperspectral 2010 Workshop', Frascati (Italy) March*, page 17–19, 2010.

[28] J.S. Walker, T.Q. Nguyen, and Y.J. Chen. A low-power, low-memory system for wavelet-based image compression. *Optical Engineering, Research Signposts*, 5:111–125, 2003.

[29] wavelet1d. C++ implementation of discrete wavelet transform. http://code.google.com/p/wavelet1d/. Accessed 14 March 2012.

7 Appendix

7.1 Instructions

1. Adjust camera scan path as parallel to the object plane as possible.

2. If not present, add a virtual serial port loop from the port configured in HySpex AIR (default COM15) to COM16 used by the system.

3. Start HySpex AIR and AutoFocus.exe.

4. In AutoFocus, enter a session name. This equals the folder on the harddrive in which the captured images will be placed.

5. Click "Connect".

6. Wait a while for a background to be captured. This is indicated by the top colored bar in HySpex AIR turning green. HySpex AIR can now be minimized if desired, or leave it open for image preview purposes.

7. If the camera is not focused, run the initial focus procedure. Place the vertical stage sufficiently high (a bit over where focus should be) and press *"Start initial focusing"*. When focus peak is found, click *"Stop procedure"*. If the peak is not the highest point in the graph, click on what you consider the focus peak.

8. If high speed capture is desired, mark the relevant checkbox. The high speed capture is three times faster than normal speed, but the focus quality might suffer depending on the extent of topography changes in the object. Image preview is not shown during high speed capture due to system limitations.

9. Click *"Start capture"*. When capturing is done, press *"Stop procedure"*. Mark down the number of the captured file in HySpex AIR for categorization.

10. The system is ready for another capture, no reconnect is necessary. However, if a session name change is desired, click "End session" in HySpex AIR and go back to step 3. Otherwise, simply press *"Start capture"* again.

11. When done, close AutoFocus (it's preferable to wait until translation stages have returned to initial position). Click "End session" in HySpex AIR, followed by "End program".

7.2 Troubleshooting

Problem	Possible solutions
Captured image has lines in it, at places with a high topography change.	This is caused by vibrations in the camera. The high speed of the adjustment will cause the camera to vibrate, if a large height change is adjusted for over a short period of time. Try running in normal speed mode, or lower the maximum speed.
Laser LED is red and/or focus is not adjusted at times during capture.	This happens mostly in two different cases: 1. The laser spot is inside the area lit by the light source, and the light intensiy is high. The laser sensor cannot see the the laser spot, drowning in the external light. 2. If the laser is tilted, a displacement can block the line of sight for the sensor regarding the laser spot. Try to reposition the object, or do not tilt the laser.
Camera is adjusted wrong.	This can happen if the laser sensor is having problems measuring the distance. Cracks and small valleys are especially challenging. Furthermore, high contrast samples like printed text can in some instances give wrong measurements. Try running the diagnostic and measurement tool for the laser sensor to investigate, or try repositioning the sample. Make sure the sensor is clean (read manual before cleaning!).
Camera suddenly stops adjusting, although it is clear it should adjust further.	Make sure that the vertical translation stage is not at it's maximum or minimum position. There is unfortunately no warning in the program when this happens. Adjust the stage to a height with more leeway, and try capturing again.

7.3 Profiling results

All the profiling is done with the "GNU gprof" profiler. Relevant excerpts from the output data is given below.

7.3.1 Energy of gradient

Profiling results for the energy of gradient algorithm. "gradient_focus" is the parent function for the focus calculation. "self" corresponds to seconds spent in function, while "children" corresponds to seconds spent in all child function. Function and type parameters are omitted. Each function call consume 140 ms / 2416 calls = 0.058 ms/call.

```
1 index % time    self  children    called         name
2
3 [1]     63.4    0.02    0.24                     focus_prep [1]
4                 0.14    0.00    2416/2416        gradient_focus [4]
5                 0.10    0.00 2967846/2967846     std::vector::operator= [6]
6                 0.00    0.00       1/1           std::vector::_M_insert_aux [17]
7 -----------------------------------------------
8                 0.14    0.00    2416/2416        focus_prep [1]
9 [4]     34.1    0.14    0.00    2416             gradient_focus [4]
```

7.3.2 Energy of laplacian

Profiling results for the energy of laplacian algorithm. "laplacian_focus" is the parent function for the focus calculation. "self" corresponds to seconds spent in function, while "children" corresponds to seconds spent in all child function. Function and type parameters are omitted. Each function call consume 2180 ms / 2416 calls = 0.9 ms/call.

```
1 index % time    self  children    called         name
2                                                  <spontaneous>
3 [1]     93.5    0.03    2.29                     focus_prep [1]
4                 2.18    0.00    2416/2416        laplacian_focus [2]
5                 0.11    0.00  2967846/2967846    std::vector::operator= [6]
6                 0.00    0.00       1/1           std::vector::_M_insert_aux [17]
7 -----------------------------------------------
8                 2.18    0.00    2416/2416        focus_prep [1]
9 [2]     87.9    2.18    0.00       2416          laplacian_focus [2]
```

7.3.3 DWT

Profiling results for the DWT algorithm. "dwt_focus" is the parent function for the focus calculation. "self" corresponds to seconds spent in function, while

"children" corresponds to seconds spent in all child function. Function and type parameters are omitted. Each function call consume 36700 ms / 2416 calls = 15.2 ms/call.

```
index % time    self  children    called         name
                                                 <spontaneous>
[1]     99.6    0.03   37.36                     focus_prep [1]
                0.58   36.70    2416/2416        dwt_focus [2]
                0.08    0.00   2967846/3052406   std::vector::operator= [18]
                0.00    0.00       1/1           std::vector::_M_insert_aux [35]
-----------------------------------------------
                0.58   36.70    2416/2416        focus_prep [1]
[2]     99.3    0.58   36.70    2416             dwt_focus [2]
                0.31   36.39    2416/2416        dwt_2d [3]
-----------------------------------------------
                0.31   36.39    2416/2416        dwt_focus [2]
[3]     97.8    0.31   36.39    2416             dwt_2d [3]
                3.15   33.21    4832/4832        dwt2 [4]
                0.01    0.00   21744/31408       std::vector::vector [21]
                0.01    0.00    4832/4832        void std::vector [22]
                0.00    0.00   77312/191931872   std::vector::_M_insert_aux [7]
                0.00    0.00    7248/7248        std::vector::operator= [23]
                0.00    0.00    4832/11751424    filtcoef [10]
                0.00    0.00   14496/14496       std::vector::insert [29]
                0.00    0.00    2416/2416        per_ext2d [30]
-----------------------------------------------
                3.15   33.21    4832/4832        dwt_2d [3]
[4]     96.9    3.15   33.21    4832             dwt2 [4]
                1.22   30.89 5870880/5870880     dwt1_m [5]
                1.09    0.00 33703200/191931872  std::vector::_M_insert_aux [7]
                0.01    0.00    9664/31408       std::vector::vector [21]
                0.00    0.00    4832/11751424    filtcoef [10]
-----------------------------------------------
                1.22   30.89 5870880/5870880     dwt2 [4]
[5]     85.5    1.22   30.89 5870880             dwt1_m [5]
               20.34    2.56 11741760/11741760   convfftm [6]
                1.73    1.80 11741760/11741760   downsamp [8]
                0.36    1.61 5870880/5870880     per_ext [9]
                0.71    1.09 11741760/11751424   filtcoef [10]
                0.49    0.00 35225280/35225280   std::vector::erase [13]
                0.20    0.00 11741760/11741760   std::vector::erase [14]
-----------------------------------------------
               20.34    2.56 11741760/11741760   dwt1_m [5]
[6]     61.0   20.34    2.56 11741760            convfftm [6]
                2.56    0.00 79003200/191931872  std::vector::_M_insert_aux [7]
-----------------------------------------------
                0.00    0.00   77312/191931872   dwt_2d [3]
                0.76    0.00 23483520/191931872  std::vector::insert [11]
                1.09    0.00 33703200/191931872  dwt2 [4]
                1.80    0.00 55664640/191931872  downsamp [8]
                2.56    0.00 79003200/191931872  convfftm [6]
[7]     16.6    6.22    0.00 191931872           std::vector::_M_insert_aux
```

7.4 Sample images

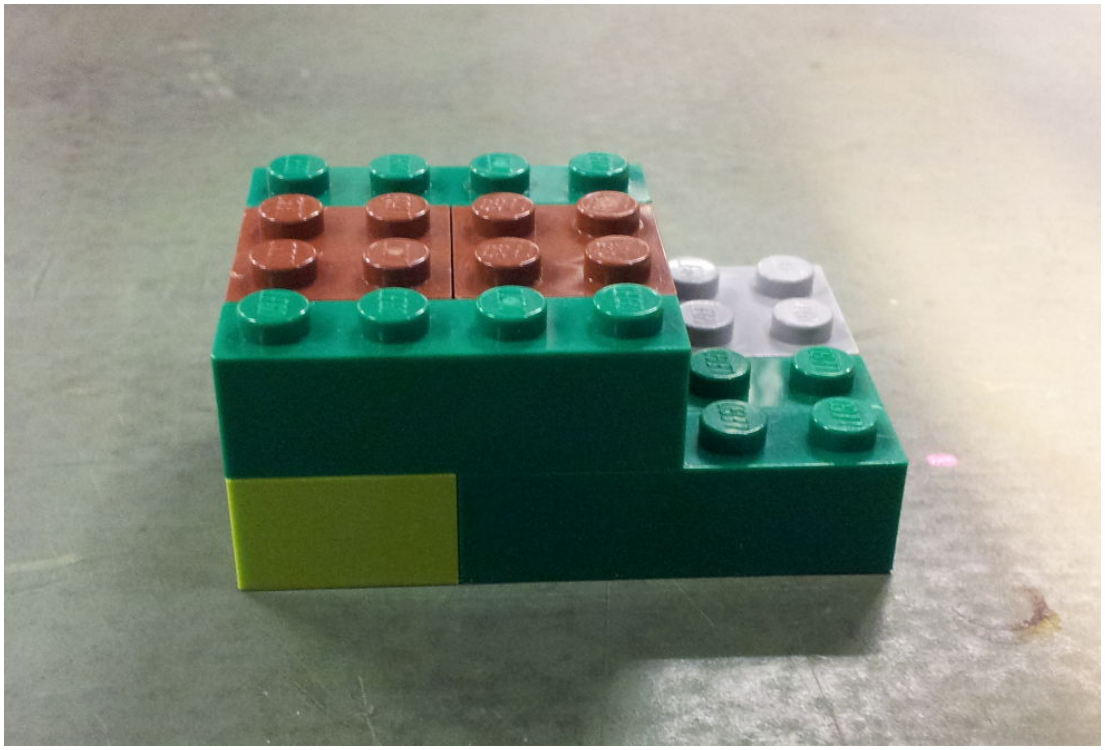


Figure 35: Image of the lego sample used in testing.

Figure 36: Image of the stone sample used in testing.

7.5 Focus test logs

The inital measurements are from the steps per mm calibration for the vertical stage, in the pre-run. Next is the calculated steps per mm value. Last are the desired adjustments found by the system.

7.5.1 Arm with fake wound

Normal speed:

```
Displacement: 174.095 , pos: 1015291
Displacement: 173.846 , pos: 1015644
Displacement: 173.597 , pos: 1016006
Displacement: 173.348 , pos: 1016340
Displacement: 173.148 , pos: 1016696
Displacement: 172.8 , pos: 1017046
Displacement: 172.65 , pos: 1017395
Displacement: 172.451 , pos: 1017745
Displacement: 172.252 , pos: 1018265
Displacement: 172.003 , pos: 1018601
Displacement: 171.854 , pos: 1018951
Displacement: 171.654 , pos: 1019301
Displacement: 171.455 , pos: 1019686
Displacement: 171.206 , pos: 1020138
Displacement: 170.957 , pos: 1020472
Displacement: 170.808 , pos: 1020822
Displacement: 170.559 , pos: 1021191
Displacement: 170.31 , pos: 1021543
Displacement: 170.11 , pos: 1021912
Displacement: 169.911 , pos: 1022230
Displacement: 169.712 , pos: 1022586
Displacement: 169.463 , pos: 1022984
Z Steps per mm: 1729
X_pos: 0.187005, Adjusting -1.39445 mm, velocity: 129
X_pos: 4.33439, Adjusting -1.19838 mm, velocity: 110
X_pos: 8.49287, Adjusting -1.32909 mm, velocity: 123
X_pos: 12.6561, Adjusting -1.49855 mm, velocity: 140
X_pos: 16.813, Adjusting -0.956622 mm, velocity: 85
X_pos: 20.9683, Adjusting -1.11163 mm, velocity: 101
X_pos: 25.1474, Adjusting -1.7085 mm, velocity: 162
X_pos: 29.3391, Adjusting -1.02024 mm, velocity: 91
X_pos: 33.5071, Adjusting -1.51128 mm, velocity: 141
X_pos: 37.6719, Adjusting -1.08039 mm, velocity: 98
X_pos: 50.1601, Adjusting -2.65298 mm, velocity: 258
X_pos: 54.309, Adjusting -1.93002 mm, velocity: 184
X_pos: 58.4976, Adjusting -1.1498 mm, velocity: 105
X_pos: 62.6529, Adjusting -1.28803 mm, velocity: 119
X_pos: 66.8098, Adjusting -1.00752 mm, velocity: 90
X_pos: 71.0285, Adjusting -1.48178 mm, velocity: 138
X_pos: 79.336, Adjusting 0.62753 mm, velocity: 51
X_pos: 87.6498, Adjusting -2.56969 mm, velocity: 249
X_pos: 91.8193, Adjusting -4.53383 mm, velocity: 449
X_pos: 95.9572, Adjusting -1.19318 mm, velocity: 109
X_pos: 100.097, Adjusting -1.04858 mm, velocity: 94
X_pos: 108.282, Adjusting -0.943898 mm, velocity: 84
X_pos: 112.36, Adjusting -0.839213 mm, velocity: 73
X_pos: 116.507, Adjusting -1.12493 mm, velocity: 102
X_pos: 120.669, Adjusting -1.2406 mm, velocity: 114
X_pos: 124.823, Adjusting -1.14575 mm, velocity: 104
X_pos: 128.978, Adjusting -1.18566 mm, velocity: 108
```

High speed:

```
Displacement: 174.294 , pos: 1015284
Displacement: 173.995 , pos: 1015647
Displacement: 173.746 , pos: 1015990
Displacement: 173.497 , pos: 1016349
Displacement: 173.298 , pos: 1016702
Displacement: 173.049 , pos: 1017052
Displacement: 172.85 , pos: 1017389
Displacement: 172.601 , pos: 1017755
Displacement: 172.252 , pos: 1018313
Displacement: 172.003 , pos: 1018624
Displacement: 171.754 , pos: 1018974
Displacement: 171.604 , pos: 1019317
Displacement: 171.256 , pos: 1019834
Displacement: 171.156 , pos: 1020174
Displacement: 170.957 , pos: 1020533
Displacement: 170.758 , pos: 1020880
Displacement: 170.559 , pos: 1021229
Displacement: 170.359 , pos: 1021579
Displacement: 170.16 , pos: 1021938
Displacement: 170.011 , pos: 1022291
Displacement: 169.812 , pos: 1022628
Displacement: 169.612 , pos: 1022978
Z Steps per mm: 1729
X_pos: 0.507132, Adjusting -1.09543 mm, velocity: 294
X_pos: 4.8336, Adjusting -1.14806 mm, velocity: 309
X_pos: 9.1775, Adjusting -1.69983 mm, velocity: 475
X_pos: 13.5325, Adjusting -1.36032 mm, velocity: 373
X_pos: 17.8938, Adjusting -0.947947 mm, velocity: 249
X_pos: 22.2884, Adjusting -1.57895 mm, velocity: 439
X_pos: 26.6212, Adjusting -1.01966 mm, velocity: 271
X_pos: 30.9699, Adjusting -1.18161 mm, velocity: 320
X_pos: 35.3423, Adjusting -1.50087 mm, velocity: 416
X_pos: 39.6799, Adjusting -0.721226 mm, velocity: 182
X_pos: 48.3962, Adjusting -0.817235 mm, velocity: 210
X_pos: 52.7971, Adjusting -2.07808 mm, velocity: 589
X_pos: 57.1189, Adjusting -2.40023 mm, velocity: 686
X_pos: 61.4786, Adjusting -1.27299 mm, velocity: 347
X_pos: 65.8399, Adjusting -1.44072 mm, velocity: 398
X_pos: 70.1949, Adjusting -1.39734 mm, velocity: 384
X_pos: 83.2995, Adjusting 0.99653 mm, velocity: 264
X_pos: 87.626, Adjusting -1.4598 mm, velocity: 403
X_pos: 91.9635, Adjusting -5.35107 mm, velocity: 1572
X_pos: 96.336, Adjusting -3.76692 mm, velocity: 1096
X_pos: 100.686, Adjusting -0.995373 mm, velocity: 264
X_pos: 105.041, Adjusting -1.11336 mm, velocity: 300
X_pos: 113.762, Adjusting -0.978022 mm, velocity: 258
X_pos: 118.117, Adjusting -1.20648 mm, velocity: 327
X_pos: 122.525, Adjusting -1.63678 mm, velocity: 456
X_pos: 126.811, Adjusting -0.954309 mm, velocity: 252
X_pos: 131.184, Adjusting -1.27935 mm, velocity: 349
X_pos: 135.561, Adjusting -0.754772 mm, velocity: 192
X_pos: 139.911, Adjusting -0.974552 mm, velocity: 258
X_pos: 144.266, Adjusting -1.08502 mm, velocity: 291
X_pos: 148.637, Adjusting -0.761134 mm, velocity: 194
X_pos: 153.016, Adjusting -0.917293 mm, velocity: 240
```

7.5.2 Lego blocks

Normal speed:

```
Displacement: 168.666 , pos: 1000334
Displacement: 168.467 , pos: 1000678
Displacement: 168.218 , pos: 1001027
```

```
Displacement: 167.969 , pos: 1001374
Displacement: 167.67 , pos: 1001736
Displacement: 167.371 , pos: 1002077
Displacement: 167.222 , pos: 1002423
Displacement: 166.973 , pos: 1002773
Displacement: 166.674 , pos: 1003309
Displacement: 166.425 , pos: 1003642
Displacement: 166.226 , pos: 1003995
Displacement: 165.977 , pos: 1004335
Displacement: 165.678 , pos: 1004756
Displacement: 165.429 , pos: 1005182
Displacement: 165.229 , pos: 1005548
Displacement: 164.98 , pos: 1005885
Displacement: 164.781 , pos: 1006251
Displacement: 164.582 , pos: 1006604
Displacement: 164.333 , pos: 1006960
Displacement: 164.134 , pos: 1007306
Displacement: 163.885 , pos: 1007640
Z Steps per mm: 1555
X_pos: 16.8003, Adjusting 7.12219 mm, velocity: 639
X_pos: 20.9651, Adjusting 4.3299 mm, velocity: 384
X_pos: 25.1268, Adjusting -1.74662 mm, velocity: 147
X_pos: 29.2821, Adjusting -1.3717 mm, velocity: 113
X_pos: 33.4406, Adjusting 10.4193 mm, velocity: 941
X_pos: 37.5864, Adjusting 1.72026 mm, velocity: 145
X_pos: 41.7448, Adjusting -1.55691 mm, velocity: 130
X_pos: 66.6735, Adjusting -20.7183 mm, velocity: 1883
```

High speed:

```
Displacement: 168.616 , pos: 1000338
Displacement: 168.417 , pos: 1000690
Displacement: 168.168 , pos: 1001040
Displacement: 167.969 , pos: 1001377
Displacement: 167.67 , pos: 1001727
Displacement: 167.421 , pos: 1002086
Displacement: 167.172 , pos: 1002432
Displacement: 166.923 , pos: 1002795
Displacement: 166.674 , pos: 1003212
Displacement: 166.425 , pos: 1003603
Displacement: 166.226 , pos: 1003940
Displacement: 165.977 , pos: 1004296
Displacement: 165.777 , pos: 1004643
Displacement: 165.429 , pos: 1005140
Displacement: 165.229 , pos: 1005474
Displacement: 165.03 , pos: 1005833
Displacement: 164.831 , pos: 1006183
Displacement: 164.582 , pos: 1006565
Displacement: 164.333 , pos: 1006898
Displacement: 164.084 , pos: 1007270
Displacement: 163.885 , pos: 1007591
Displacement: 163.686 , pos: 1007925
Z Steps per mm: 1539
X_pos: 17.9271, Adjusting 9.31384 mm, velocity: 2454
X_pos: 22.3106, Adjusting 7.13645 mm, velocity: 1872
X_pos: 26.6545, Adjusting 2.18389 mm, velocity: 549
X_pos: 35.3645, Adjusting 11.3853 mm, velocity: 3008
X_pos: 39.7132, Adjusting 6.51072 mm, velocity: 1705
X_pos: 44.0967, Adjusting 3.15335 mm, velocity: 808
X_pos: 48.4406, Adjusting -1.33268 mm, velocity: 321
X_pos: 65.8891, Adjusting -10.4587 mm, velocity: 2760
X_pos: 70.2567, Adjusting -16.9071 mm, velocity: 4484
X_pos: 92.0254, Adjusting -0.512021 mm, velocity: 102
```

7.5.3 Stone

Normal speed:

```
Displacement: 162.689 , pos: 1000324
Displacement: 162.44 , pos: 1000664
Displacement: 162.191 , pos: 1001017
Displacement: 161.942 , pos: 1001367
Displacement: 161.644 , pos: 1001733
Displacement: 161.395 , pos: 1002067
Displacement: 161.096 , pos: 1002423
Displacement: 160.797 , pos: 1002776
Displacement: 160.448 , pos: 1003350
Displacement: 160.249 , pos: 1003655
Displacement: 159.95 , pos: 1004008
Displacement: 159.701 , pos: 1004361
Displacement: 159.402 , pos: 1004755
Displacement: 159.104 , pos: 1005201
Displacement: 158.954 , pos: 1005548
Displacement: 158.805 , pos: 1005901
Displacement: 158.705 , pos: 1006234
Displacement: 158.506 , pos: 1006613
Displacement: 158.307 , pos: 1006940
Displacement: 158.107 , pos: 1007293
Displacement: 157.908 , pos: 1007672
Displacement: 157.659 , pos: 1007999
Z Steps per mm: 1632
X_pos: 0.179081, Adjusting 1.19547 mm, velocity: 102
X_pos: 4.36292, Adjusting 1.59498 mm, velocity: 141
X_pos: 8.43582, Adjusting 0.863971 mm, velocity: 71
X_pos: 16.7718, Adjusting -0.879289 mm, velocity: 72
X_pos: 20.9319, Adjusting -0.634804 mm, velocity: 48
X_pos: 25.13, Adjusting -2.42892 mm, velocity: 221
X_pos: 29.3027, Adjusting -2.62623 mm, velocity: 240
X_pos: 33.4691, Adjusting -1.85662 mm, velocity: 166
X_pos: 37.6307, Adjusting -1.84988 mm, velocity: 165
X_pos: 41.8494, Adjusting -2.46569 mm, velocity: 225
X_pos: 45.9937, Adjusting -5.67279 mm, velocity: 532
X_pos: 50.1331, Adjusting -16.1305 mm, velocity: 1536
X_pos: 62.3217, Adjusting 0.787377 mm, velocity: 63
X_pos: 66.3344, Adjusting 0.910539 mm, velocity: 75
```

High speed:

```
Displacement: 162.689 , pos: 1000331
Displacement: 162.44 , pos: 1000677
Displacement: 162.191 , pos: 1001030
Displacement: 161.992 , pos: 1001370
Displacement: 161.693 , pos: 1001723
Displacement: 161.395 , pos: 1002070
Displacement: 161.096 , pos: 1002416
Displacement: 160.697 , pos: 1002933
Displacement: 160.498 , pos: 1003260
Displacement: 160.299 , pos: 1003606
Displacement: 160 , pos: 1003969
Displacement: 159.452 , pos: 1004617
Displacement: 159.402 , pos: 1004861
Displacement: 159.153 , pos: 1005211
Displacement: 158.954 , pos: 1005564
Displacement: 158.805 , pos: 1005923
Displacement: 158.655 , pos: 1006279
Displacement: 158.506 , pos: 1006638
Displacement: 158.307 , pos: 1006985
Displacement: 158.107 , pos: 1007344
Displacement: 157.858 , pos: 1007688
Z Steps per mm: 1632
```

```
X_pos: 0.438986, Adjusting 1.14583 mm, velocity: 290
X_pos: 4.77655, Adjusting 1.53983 mm, velocity: 402
X_pos: 9.11569, Adjusting 0.749387 mm, velocity: 177
X_pos: 17.8193, Adjusting -0.773284 mm, velocity: 184
X_pos: 22.2377, Adjusting -1.18934 mm, velocity: 302
X_pos: 26.5468, Adjusting -1.90196 mm, velocity: 504
X_pos: 30.8906, Adjusting -2.73284 mm, velocity: 740
X_pos: 35.2678, Adjusting -2.53248 mm, velocity: 683
X_pos: 39.6181, Adjusting -1.46446 mm, velocity: 380
X_pos: 43.9493, Adjusting -4.60784 mm, velocity: 1271
X_pos: 48.3328, Adjusting -19.7059 mm, velocity: 5550
X_pos: 78.8368, Adjusting 0.624387 mm, velocity: 142
```

7.6 Passive autofocus prototyping code

7.6.1 f_gradient.m

```
function [ g_x ] = gradient_energy( input, scale )

input = double(input);
g_x = zeros(size(input));

for n=1:numel(input)-1,
    if n <= numel(input)
        g_x(n) = input(n) - input(n+1);

    end
end

g_x = sum(g_x)^2;
```

7.6.2 f_laplacian.m

```
function [ focus_value ] = f_laplacian( input )

[rows cols] = size(input);
focus_sum = 0;
kernel = [-1 -4 -1; -4 20 -4; -1 -4 -1;];
for x=2:rows-2,
    for y=2:cols-2,
        focus_sum_temp = kernel(1,1)*input(x-1, y-1)...
        + kernel(1,2)*input(x-1, y)...
        + kernel(1,3)*input(x-1,y+1)...
        + kernel(2,1)*input(x, y-1)...
        + kernel(2,2)*input(x,y)...
        + kernel(2,3)*input(x, y+1)...
        + kernel(3,1)*input(x+1, y-1)...
        + kernel(3,2)*input(x+1, y)...
        + kernel(3,3)*input(x+1, y+1);
        focus_sum = focus_sum + focus_sum_temp^2;
    end
end
focus_value = focus_sum;
end
```

7.6.3 focus_sim_general.m

The same code is used for all the images by changing filename and other relevant data.

```
clear all

base_path = '/media/LinDisk/Master/3101/';
project_path = 'HySpex/';

filename = 'HySpex_505_VNIR_1600_SN0002_corr';
%filename = 'HySpex_508_VNIR_1600_SN0002_corr.hyspex';

% Set lines manually
lines = 1029;

% Get header and read out no. of lines in image
fid=fopen([base_path, project_path, filename, '.hdr'], 'r+');
while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
    m = regexpi(tline, 'lines = (.*)', 'tokens', 'once');
    if numel(m) > 0,
        lines = str2num(char(m));
    end
end
fclose(fid);
%

samples = 1600;
bands_num = 160;

%image_rows = [135 550]; %For tilted sample
image_rows = [30 900];
image_columns = [1 samples];
image_bands = [55 41 12];
header_offset = 4104192; %Important! Can vary from file to file, is dependent on
     no. of bands.
datatype = 'uint16';
format = 'bil';



X=multibandread([base_path, project_path, filename, '.hyspex'],[lines,samples,
    bands_num],datatype,header_offset,format,'ieee-le', {'Row','Range',
    image_rows}, {'Column','Range',image_columns},{'Band','Direct', image_bands
    });

% YCbCr conversion
Y   =    0.299 * X(:,:,1) +  0.587 * X(:,:,2) +   0.114 * X(:,:,3);
clear X

[rows cols dim] = size(Y);

% Normalize sample to prevent light intensity from interfering
for row=1:rows,
    Y(row, :) = Y(row, :) / mean(mean(Y(row, :)));
end
% Number for averaging (or image sample size for 2D)
num_rows = 20;
focus_values = zeros(rows-num_rows, 3);

% 1D algorithm
for row=1:rows,
    if row > num_rows,
        outline = 750:950;
        sample = Y(row-num_rows:row, outline);
```

```
59          [x1 y1] = size(sample);
60          focus_sum = 0;
61          for y=1:y1,
62              focus_sum = focus_sum + gradient_energy(sample(:,y));
63          end
64          focus_values(row-num_rows, 1) = focus_sum;
65      end
66  end
67
68  % 2D algorithm Laplacian
69  for row=1:rows,
70      if row > num_rows,
71          outline = 750:950;
72          sample = Y(row-num_rows:row, outline);
73          focus_sum = f_laplacian(sample);
74          focus_values(row-num_rows, 2) = focus_sum;
75
76      end
77  end
78
79  % 2D algorithm DWT
80  for row=1:rows,
81      if row > num_rows,
82          outline = 750:950;
83          sample = Y(row-num_rows:row, outline);
84          [ C S ] = wavefast(sample, 2, 'db4');
85          % C is coefficients (approx, vertical detail, horizontal detail,
86          % etc..). Indices is given by S.
87          % We don't want approx coeffecients, so we skip first S(1,1)*S(1,2)
88          % numbers
89          % Apply Euclidean norm (sqrt(xn^2 + xn-1^2 + ... + x1^2)
90          detail_norm = sqrt(sum(arrayfun(@(x) x^2, C(S(1,1)*S(1,2):end))));
91          image_norm = sqrt(sum(arrayfun(@(x) x^2, sample(:))));
92          focus_sum = detail_norm / (image_norm - detail_norm);
93          focus_values(row-num_rows, 3) = focus_sum;
94          % Display progress (takes a while...)
95          disp((row-num_rows)/(rows-num_rows)*100)
96
97      end
98  end
99
100 % Normalize
101 focus_values(:,1) = focus_values(:,1) / max(max(focus_values(:,1)));
102 focus_values(:,2) = focus_values(:,2) / max(max(focus_values(:,2)));
103 focus_values(:,3) = focus_values(:,3) / max(max(focus_values(:,3)));
104
105
106 % Set the range of the axes
107 % The background image will be stretched to this.
108
109 min_x = 0;
110 max_x = numel(focus_values(:,1));
111 min_y = 0;
112 max_y = max(focus_values);
113
114 % Get size of image
115 [image_x image_y] = size(Y);
116 % Create RGB image
117 targetImage = zeros(image_x, image_y, 3);
118 targetImage_base = Y/max(max(Y));
119 % Tone down unused area
120 for row=1:image_y,
121     if ismember(row, outline),
122     else
123         targetImage_base(:, row) = 0.25 .* targetImage_base(:, row);
124     end
125 end
126 % Fill RGB-image (grayscale)
```

```
127 targetImage(:,:,1) = targetImage_base;
128 targetImage(:,:,2) = targetImage_base;
129 targetImage(:,:,3) = targetImage_base;
130
131 % Rotate and flip
132 targetImage = flipdim(targetImage, 1);
133 targetImage = imrotate(targetImage, -90);
134 % Scale to figure
135 h_image = imagesc([min_x max_x], [min_y max_y], targetImage);
136 hold on;
137 focus_values
138 plot(num_rows+1:numel(focus_values(:,1))+num_rows,focus_values(:,1),'r-','
        linewidth', 1);
139 plot(num_rows+1:numel(focus_values(:,2))+num_rows,focus_values(:,2),'b-','
        linewidth', 1);
140 plot(num_rows+1:numel(focus_values(:,3))+num_rows,focus_values(:,3),'g-','
        linewidth', 1);
141 hold off;
142
143 legends = cell(3,1);
144 legends(1) = {['1D algorithm']};
145 legends(2) = {['2D algorithm (Laplacian)']};
146 legends(3) = {['2D algorithm (DWT)']};
147
148 title('Passive focusing: contrast change, flat', 'FontWeight','Bold','FontSize'
        ,16)
149 xlabel('Line', 'FontSize',14);
150 ylabel('Focus value','FontSize',14);
151 h_legends = legend(legends);
152 set(h_legends,'FontSize',14);
153 % grid on;
154 % colormap(gray);
155 %imageTransparency = 0.9;
156 %alpha(h_image, imageTransparency);
157 % set the y-axis back to normal.
158 set(gca,'ydir','normal');
159 set(gca, 'LooseInset', [0,0,0,0]);
160
161
162 % Set aspect ratio to match image
163 %pbaspect([1 (image_rows(2)-image_rows(1))/(image_columns(2)-image_columns(1))
        1])
```

7.7 Benchmarking code

The DWT code is utilizing the wavelet1d library, see [29].

7.7.1 FocusBench.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include "Image.h"
5  #include "focus_alg.h"
6  #include "wavelet2s.h"
7
8
9  using namespace std;
10 using namespace ENVI;
11 using namespace FOCUS;
12
```

```
13 // Function to be benchmarked
14 double dwt_focus(vector<vector<double> > &sample_data){
15
16     vector<double> dwt_output;
17     vector<double> flag;
18     vector<int> length;
19     int levels = 2; // Decomposition levels
20     string wavelet_type = "db4";
21     dwt_2d( sample_data, levels, wavelet_type, dwt_output, flag, length);
22
23     int approx_end = length.at(0) * length.at(1); // Start of details
           coefficients (in other words, end of approx coeffs)
24
25     // Euclidian norm of sample.
26     double norm_sum = 0;
27     for (int x = 0; x < sample_data.size(); x++){
28         vector<double> yline = sample_data.at(x);
29         for (int y = 0; y < yline.size(); y++){
30             norm_sum += yline.at(y)*yline.at(y);
31         }
32     }
33     double sample_norm = sqrt(norm_sum);
34
35     // Euclidian norm of detail coeffs.
36     norm_sum = 0;
37     for (int i = approx_end; i < dwt_output.size(); i++){
38         norm_sum += dwt_output.at(i)*dwt_output.at(i);
39     }
40     double dwt_norm = sqrt(norm_sum);
41
42     double focus_sum = dwt_norm / (sample_norm - dwt_norm);
43     return focus_sum;
44
45 };
46
47 // Function to be benchmarked
48 double laplacian_focus(vector<vector<double> > &sample_data){
49
50     double focus_sum = 0;
51     // Index starts at 1 and ends at size()-1 to exclude sides/corners, as they'
           re included by the algorithm
52     for (int x = 1; x < sample_data.size()-1; x++){
53         vector<double> yline = sample_data.at(x);
54         for (int y = 1; y < yline.size()-1; y++){
55             focus_sum +=
56             // left row
57             - 1 * sample_data.at(x-1).at(y-1) - 4 * sample_data.at(x-1).at(y) -
                   1 * sample_data.at(x-1).at(y+1)
58             // middle row
59             - 4 * yline.at(y-1) + 20 * yline.at(y) - 4 * yline.at(y+1)
60             // right row
61             - 1 * sample_data.at(x+1).at(y-1) - 4 * sample_data.at(x+1).at(y) -
                   1 * sample_data.at(x+1).at(y+1);
62
63         }
64     }
65
66
67     return focus_sum*focus_sum;
68
69 };
70
71 // Function to be benchmarked
72 double gradient_focus(vector<vector<double> > &sample_data){
73
74     double focus_sum = 0;
75     // Index starts at 1 and ends at size()-1 to exclude sides/corners, as they'
           re included by the algorithm
```

```
76      for (int x = 0; x < sample_data.size()-1; x++){
77          vector<double> yline = sample_data.at(x);
78          for (int y = 1; y < yline.size()-1; y++){
79              focus_sum += sample_data.at(x).at(y-1) - sample_data.at(x).at(y);
80          }
81      }
82
83
84      return focus_sum*focus_sum;
85
86  };
87
88
89
90
91  double focus_prep(ImageHeader imHeader, Image<it_BIL, uint16_t> imImage){
92      vector<vector<double> > sample_data(imHeader.lines, vector<double>(imHeader.
            samples));
93      // Create image sample (20x1600) (shoddy implementation, but this is only
            for algorithm testing..)
94      int counter = 0;
95      int limit = 20;
96      for (int i = 0; i*imImage.line_size+imImage.band_size < imImage.size(); i++)
            {
97          vector<double> line_data(imImage.data.begin()+i*imImage.line_size+1*
                imImage.band_size, imImage.data.begin()+i*imImage.line_size+2*
                imImage.band_size);
98          sample_data.push_back(line_data);
99          if (sample_data.size() >= limit){
100             while (sample_data.size() > limit){
101                 sample_data.erase(sample_data.begin()); //Remove front sample to
                        keep only 20 lines in sample
102             }
103         }
104         else {
105             continue; // Sample too small, continue
106         }
107         //cout << "Counter: " << counter << " sample size: " << sample_data.size
                () << endl;
108         counter++;
109         //cout << laplacian_focus(sample_data) << endl;
110         //laplacian_focus(sample_data); // Don't print to avoid stdout delay
111         //dwt_focus(sample_data); // Don't print to avoid stdout delay
112         gradient_focus(sample_data);
113     }
114
115
116 }
117
118
119
120 int main ()
121 {
122     char headerFile[] = "/home/sveint/Dropbox/Prosjekt/Samples/
            cody_kidney_surface_VNIR_1600_SN0004_10000_us_2x_2011-08-15
            T134755_corr_rgb_sample.hdr";
123     char imageFile[] = "/home/sveint/Dropbox/Prosjekt/Samples/
            cody_kidney_surface_VNIR_1600_SN0004_10000_us_2x_2011-08-15
            T134755_corr_rgb_sample.hyspex";
124     ImageHeader imHeader;
125     Image<it_BIL, uint16_t> imImage = Image<it_BIL, uint16_t>();
126     cout << "Focus␣test\n";
127     cout << "Reading␣header\n";
128     bool readSuccess = imHeader.read(headerFile);
129     if (readSuccess){
130         cout << "Header␣file␣successfully␣loaded!\n";
131         cout << imHeader << endl;
132
```

```
133     }
134     else {
135         cout << "ERROR: Couldn't read header file.\nPlease check filename, path
                and permissions\n";
136     }
137     cout << "Loading image data\n";
138     bool readSuccess2 = false;
139     readSuccess2 = read_image(imImage, imHeader, imageFile);
140     if (readSuccess2){
141         cout << "Image data successfully loaded!\n";
142         cout << "Size: " << imImage.size() << "\n";
143     }
144     else {
145         cout << "ERROR: Couldn't read image data.\nPlease check filename, path
                and integrity\n";
146     }
147
148     focus_prep(imHeader, imImage);
149
150     return 0;
151 }
```

7.8 System code

7.8.1 XAutoFocus.cpp

```
1  /*
2  Continuous autofocus solution by Svein Tore Seljebotn (sveint@gmail.com)
3  This work is licensed under a Creative Commons Attribution-NonCommercial 3.0
       Unported License.
4  See http://creativecommons.org/licenses/by-nc/3.0/ for more information.
5  Last revision May 2012
6  */
7
8  #include "XAutoFocus.h"
9
10 using namespace FOCUS;
11
12
13 // If you wonder about positive/negative signs in the code,
14 // keep in mind that positive isn't necessarily in the direction in which the
       stage is moving.
15
16 int XAutoFocus::FOCUS_REFOCUS_DIST = 3; // travelled mm before refocus
17 int XAutoFocus::FOCUS_XSTEPS_PER_MM= 630; // steps per mm for the translation
       stage (is overwritten by calibrated value)
18 int XAutoFocus::FOCUS_ZSTEPS_PER_MM= 100; // steps per mm for the translation
       stage (is overwritten by calibrated value)
19 int XAutoFocus::FOCUS_MEASUREMENT_DISPLACEMENT= 20; // mm between laser spot and
        camera FOV (is overwritten by calibrated value)
20 int XAutoFocus::FOCUS_FRAMERATE = 100; // ms between per frame (is overwritten
       by settings value)
21
22
23
24 XAutoFocus::XAutoFocus(){
25   mScanSpeed = 150; // for 30 ms per frame
26   mPreRunPhase = 0;
27   mXPosition = -1;
28   mZPosition = -1;
29   mPreRunCounter = 0;
30   mDeactivateFocus = false;
31
```

```
32 };
33
34 // Load calibrated settings into object
35 void XAutoFocus::init(){
36   Settings::load();
37   FOCUS_XSTEPS_PER_MM = Settings::XCalStepsPerMm;
38   FOCUS_MEASUREMENT_DISPLACEMENT = Settings::LaserCalMm;
39 };
40
41 void XAutoFocus::setTranslationStage(TranslationStage* ts){
42   mTranslationStage = ts;
43   mXInitialPos = mTranslationStage->getPosition(mTranslationStage->
         getHorizontalDeviceId());
44   mZInitialPos = mTranslationStage->getPosition(mTranslationStage->
         getVerticalDeviceId());
45   mXLastRefocusPos = mXInitialPos;
46
47 }
48
49 void XAutoFocus::setDisplacementSensor(DisplacementSensor* ds){
50   mDisplacementSensor = ds;
51 }
52
53 void XAutoFocus::setDeactivateFocus(bool state){
54   mDeactivateFocus = state;
55 }
56
57 void XAutoFocus::reset(){
58   mPreRunPhase = 0;
59   mDisplacementValues.clear();
60 }
61
62 void XAutoFocus::setScanSpeed(bool high){
63   if (high){
64     mScanSpeed = 450;
65   }
66   else {
67     mScanSpeed = 150;
68   }
69
70 }
71
72
73 // For plotting
74 pair<double, double> XAutoFocus::getLatestDisplacement(){
75   if (!mDisplacementValues.empty()){
76     return pair<double, double>(-double(mDisplacementValues.back().xpos_in_um)
           /1000, -(mDisplacementValues.back().measured_displacement -
           mInitialDisp)-double(mTranslationStage->getPosition(mTranslationStage->
           getVerticalDeviceId()) - mZInitialPos) / FOCUS_ZSTEPS_PER_MM);
77   }
78   else {
79     return pair<double, double>(0.0, 0.0);
80   }
81 }
82
83 pair<double, double> XAutoFocus::getLatestZTranslationPosition(){
84   return pair<double, double>(
85     -double(mTranslationStage->getPosition(mTranslationStage->
           getHorizontalDeviceId()) - mXInitialPos)/FOCUS_XSTEPS_PER_MM,
86     -double(mTranslationStage->getPosition(mTranslationStage->
           getVerticalDeviceId()) - mZInitialPos) / FOCUS_ZSTEPS_PER_MM
87     );
88
89 }
90
91 bool XAutoFocus::isInPreRun(){
92   return mPreRunPhase < 6;
```

```
93
94  }
95
96  bool XAutoFocus::isPlotDataReady(){
97    return mPreRunPhase >= 4;
98
99  }
100
101 // Main loop function. This function first perform a "prerun", basically
        calibrating the laser and mapping the first blind area with length
        FOCUS_MEASUREMENT_DISPLACEMENT
102 // then running the main update() function at completion.
103 void XAutoFocus::run(){
104   mTranslationStage->setSpeed(mScanSpeed);
105   switch(mPreRunPhase){
106
107     // Starting calibration of the laser. Start movement a certain distance
108     case 0:
109       mXInitialPos = mTranslationStage->getPosition(mTranslationStage->
              getHorizontalDeviceId());
110       mZInitialPos = mTranslationStage->getPosition(mTranslationStage->
              getVerticalDeviceId());
111       mTranslationStage->setSpeed(400);
112       mTranslationStage->goToPosition(mTranslationStage->getVerticalDeviceId(),
              mZInitialPos + 8000);
113       mPreRunPhase = 1;
114       break;
115     // Add measurement to DisplacementSensor
116     case 1:
117       if (mTranslationStage->isRunning(mTranslationStage->getVerticalDeviceId())
              ){
118         mDisplacementSensor->addCalibratedPosition(mTranslationStage->
              getPosition(mTranslationStage->getVerticalDeviceId()));
119       }
120       else {
121         mTranslationStage->setSpeed(400);
122         mTranslationStage->goToPosition(mTranslationStage->getVerticalDeviceId()
              , mZInitialPos);
123         mPreRunPhase = 2;
124       }
125       break;
126
127     // Wait until reaching initial position
128     case 2:
129       if (!mTranslationStage->isRunning(mTranslationStage->getVerticalDeviceId()
              )){
130         mPreRunPhase = 3;
131         FOCUS_ZSTEPS_PER_MM = mDisplacementSensor->getCalibratedStepsPerMm();
132         cout << "Z Steps per mm: " << mDisplacementSensor->
              getCalibratedStepsPerMm() << endl;
133       }
134       break;
135
136     // Moving X stage backwards in order to get displacement measurement at FOV
137     case 3:
138       mTranslationStage->goToPosition(mTranslationStage->getHorizontalDeviceId()
              , mXInitialPos + FOCUS_MEASUREMENT_DISPLACEMENT * FOCUS_XSTEPS_PER_MM
              );
139       mPreRunPhase = 4;
140       break;
141
142     // Moving X stage forwards again and begin measurement
143     case 4:
144       if (!mTranslationStage->isRunning(mTranslationStage->getHorizontalDeviceId
              ())){
145         mInitialDisp = mDisplacementSensor->getDisplacement(); // Set distance
              to object at camera FOV
```

```
146         mPreRunPos1 = mXLastRefocusPos = mTranslationStage->getPosition(
                mTranslationStage->getHorizontalDeviceId());
147         mTranslationStage->goToPosition(mTranslationStage->getHorizontalDeviceId
                (), mXInitialPos);
148         mPreRunPhase = 5;
149       }
150       break;
151
152     // Do measurements (in the first FOCUS_MEASUREMENT_DISPLACEMENT area)
153     case 5:
154       if (!mTranslationStage->isRunning(mTranslationStage->getHorizontalDeviceId
              ())){
155         mPreRunPhase = 6;
156         // Calculate translation stage speed
157         mPreRunPos2 = mTranslationStage->getPosition(mTranslationStage->
                getHorizontalDeviceId());
158         int steps_travelled = abs(mPreRunPos2 - mPreRunPos1);
159         int time_spent_ms = mPreRunCounter * FOCUS_FRAMERATE;
160         mXStepsPerSec = double(steps_travelled * 1000) / time_spent_ms;
161         cout << "Steps␣per␣sec:␣" << mXStepsPerSec << endl;
162         cout << "X␣Steps␣per␣mm:␣" << FOCUS_XSTEPS_PER_MM << endl;
163         // Start horizontal movement
164         mTranslationStage->setSpeed(mScanSpeed);
165         mTranslationStage->startBackward(mTranslationStage->
                getHorizontalDeviceId());
166         mPreRunCounter = 0; // reset counter
167
168       }
169       else {
170         mPreRunCounter++;
171         addDisplacementValue();
172
173       }
174       break;
175
176     // Begin normal operation
177     case 6:
178       update();
179       break;
180
181   }
182 }
183
184 void XAutoFocus::update(){
185   // Get x position
186   int xcurrent_position = mTranslationStage->getPosition(mTranslationStage->
          getHorizontalDeviceId());
187   int zcurrent_position = mTranslationStage->getPosition(mTranslationStage->
          getVerticalDeviceId());;
188
189   // Check x-position for whether we should refocus
190   try {
191     if (abs(((xcurrent_position - mXLastRefocusPos) / FOCUS_XSTEPS_PER_MM)) >
            abs(FOCUS_REFOCUS_DIST)){
192       mXLastRefocusPos = xcurrent_position;
193       int zfinal_position = zcurrent_position;
194       if (!mDisplacementValues.empty()){
195         int target_xpos = double(xcurrent_position-mXInitialPos)/
                FOCUS_XSTEPS_PER_MM - FOCUS_REFOCUS_DIST;
196         bool loop = true;
197         while (loop){
198           if (!mDisplacementValues.empty() && mDisplacementValues.front().
                  xpos_in_um >= target_xpos*1000){
199             zfinal_position = mDisplacementValues.front().zfinal_position;
200             mDisplacementValues.pop_front(); // Delete from stack
201           }
202           else {
203             loop = false;
```

```
204           }
205         }
206
207
208         // Number of steps to adjust (z-stage)
209         int number_of_steps = abs(zcurrent_position - zfinal_position);
210
211         // We demand a certain threshold in order to avoid constant, yet
                unecessary movement
212         // The sound can be annoying if one constantly have to adjust with
                different pitch.
213         // Our depth of field is much higher than ~0.5 mm, hence no adjustment
                is necessary.
214         if (abs(double(number_of_steps)/mDisplacementSensor->
                getCalibratedStepsPerMm()) > 0.5){
215
216           //cout << "Adjusting: steps: " << number_of_steps << ", mm: " <<
                  displacement_change << endl;
217           //cout << "number of steps: " << zcurrent_position << " - " <<
                  target_zpos << endl;
218           // Time to adjust: Refocus period / steps per sec for x stage
219           double time_for_adjustment = abs(double(FOCUS_REFOCUS_DIST *
                  FOCUS_XSTEPS_PER_MM) / mXStepsPerSec); // seconds
220
221           // Target velocity: Using the linear regression from calibration. It
                  gives the required velocity if we had one second
222           // to adjust. We don't, so divide by the available time to get the
                  actual velocity.
223           // setSpeed() has number-checking, so don't worry about limits
224           int target_velocity = double((Settings::ZCalA_Coeff + Settings::
                  ZCalB_Coeff * abs(number_of_steps))) / time_for_adjustment;
225
226           // The calibration is a bit off, so 3/4 of the value is better.
227           mTranslationStage->setSpeed(int(double(target_velocity*0.75)));
228
229           // We don't want to interrupt if stage is performing a huge adjustment
                .
230           // Optimal code taking into consideration directions etc would be
                  quite complex,
231           // testing easy solution which should work in most cases...
232           bool adjust = true;
233
234           // If new position is within 0.5mm of the last AND the stage is
                  running, don't resend command (as it will pause the stage).
235           if (mTranslationStage->isRunning(mTranslationStage->
                  getVerticalDeviceId()) && abs(double(mZLastSentPosition -
                  zfinal_position))/mDisplacementSensor->getCalibratedStepsPerMm()
                  < 0.5){
236             adjust = false;
237           }
238
239
240           mZLastSentPosition = zfinal_position;
241
242           if (!mDeactivateFocus && adjust){
243             cout << "X_pos:␣" << -double(xcurrent_position-mXInitialPos)/
                    FOCUS_XSTEPS_PER_MM << ",␣Adjusting␣" << double(
                    zcurrent_position - zfinal_position) / mDisplacementSensor->
                    getCalibratedStepsPerMm() << "␣mm,␣velocity:␣" << int(double(
                    target_velocity*0.75)) << endl;
244             mTranslationStage->goToPosition(mTranslationStage->
                    getVerticalDeviceId(), zfinal_position);
245           }
246         }
247       }
248     }
249   }
250   catch (const char * msg){
```

```
251     cout << "Error! " << msg << endl;
252   }
253
254   addDisplacementValue();
255
256
257 }
258
259
260
261 void XAutoFocus::addDisplacementValue(){
262   try {
263     // Add measured value to stack along with projected position
264     // We want position to be relative to starting position, not current
          position
265     double displacement = mDisplacementSensor->getDisplacement();
266     if (displacement < 1){ // 0 is not valid value-> out of range or error
267       return;
268     }
269
270     int xposition = mTranslationStage->getPosition(mTranslationStage->
          getHorizontalDeviceId());
271     int zposition = mTranslationStage->getPosition(mTranslationStage->
          getVerticalDeviceId());
272     // Position for this measurement in X-direction in micron (relative to
          inital position!)
273     int xpos_in_um = (double(xposition - mXInitialPos)/ FOCUS_XSTEPS_PER_MM -
          FOCUS_MEASUREMENT_DISPLACEMENT) * 1000;
274     // Calculate change in vertical stage
275     // How much should we adjust relative to initial position?
276     double displacement_in_mm = displacement - mInitialDisp;
277
278     //cout << "CALIBRATED STEPS: " << mDisplacementSensor->
          getCalibratedStepsPerMm() << endl;
279     //cout << "DISPLACEMENT: " << displacement << endl;
280     //cout << "DISPLACEMENT CHANGE:" << displacement_in_mm << endl;
281     XFocusPoint point;
282     point.xpos_in_um = xpos_in_um;
283     point.measured_displacement = displacement;
284     point.zcamera_position = mCurrentCameraPosition;
285     // Calculate what final position this corresponds to
286     point.zfinal_position = zposition + displacement_in_mm*mDisplacementSensor->
          getCalibratedStepsPerMm();
287     point.zdisplacement_in_mm = displacement_in_mm;
288     //cout << "point: " << point.measured_displacement << endl;
289     mDisplacementValues.push_back(point);
290   }
291   catch (const char* msg){
292     cout << "Error in adding displacement: " << msg << endl;
293   }
294
295 }
```

7.8.2 CalibrationDialog.cpp

```
1 /*
2 Continuous autofocus solution by Svein Tore Seljebotn (sveint@gmail.com)
3 This work is licensed under a Creative Commons Attribution-NonCommercial 3.0
4 Unported License.
5 See http ://creativecommons.org/licenses/by-nc/3.0/ for more information.
6 Last revision May 2012
7 */
8
9
```

```
10
11  #include "CalibrationDialog.h"
12
13  CalibrationDialog::CalibrationDialog(){
14
15    mXCalPhase = 0;
16    mZCalPhase = 0;
17    mZCalSpeed = 100;
18    mZCalCounter = 0;
19    setupUi(this);
20    readSettings();
21
22
23  };
24
25
26  void CalibrationDialog::setDisplacementSensor(DisplacementSensor* ds){
27    mDisplacementSensor = ds;
28  }
29
30  void CalibrationDialog::setTranslationStage(TranslationStage* ts){
31    mTranslationStage = ts;
32    mXCalInitialPos = mTranslationStage->getPosition(mTranslationStage->
          getHorizontalDeviceId());
33    mZCalInitialPos = mTranslationStage->getPosition(mTranslationStage->
          getVerticalDeviceId());
34
35  }
36
37  void CalibrationDialog::startXCalibration(){
38    // If user presses button again, stop procedure
39    if (mXCalPhase > 0){
40      mXCalPhase = 5;
41    }
42    else {
43      mXCalPhase = 0;
44    }
45    mXCalTimer = new QTimer();
46    connect(mXCalTimer,SIGNAL(timeout()),SLOT(performXCalibration()));
47    mXCalTimer->start(30); // 30 ms should be accurate enough
48  };
49
50  void CalibrationDialog::performXCalibration(){
51    // Procedure:
52    // 1. Measure displacement. We assume we're on one of the edges. Start X stage
53    // 2. Keep measuring until we notice a drop over the threshold.
54    // 3. Wait for displacement to stabilize..When stabilized, we're over the edge
          , so save the position (steps) as pos1
55    // 4. Keep going until we notice a rise/fall over the threshold. If so, save
          the position as pos2.
56    // 5. Calculate steps per mm and save the data.
57    double disp_threshold = 10; //in mm
58    double new_displacement = 0;
59    double mm_input = 0;
60    int number_of_steps = 0;
61    int steps_per_mm = 0;
62    int temp_position = 0;
63    QSettings* settings;
64    QDateTime dateTime = QDateTime::currentDateTime();
65    QString dateTimeString = dateTime.toString();
66    switch(mXCalPhase){
67      case 0:
68        cout << "Starting␣calibration" << endl;
69        mXCalInitialPos = mTranslationStage->getPosition(mTranslationStage->
              getHorizontalDeviceId());
70        mXCalLastDisplacement = mDisplacementSensor->getDisplacement();
71        mTranslationStage->setSpeed(700);
```

```
72        mTranslationStage->startBackward(mTranslationStage->getHorizontalDeviceId
              ());
73        mXCalPhase++;
74        XCalibrateButton->setText("Stop");
75        break;
76      case 1:
77        new_displacement = mDisplacementSensor->getDisplacement();
78        if (abs(mXCalLastDisplacement - new_displacement) > disp_threshold){
79          // Set position here, as that will be the edge of the object
80          mXCalPos1 = mTranslationStage->getPosition(mTranslationStage->
                getHorizontalDeviceId());
81          mXCalPhase++;
82        }
83        break;
84      case 2:
85        new_displacement = mDisplacementSensor->getDisplacement();
86        temp_position = mTranslationStage->getPosition(mTranslationStage->
              getHorizontalDeviceId());
87        // Due to focus spot of laser not being too well defined, wait for some
              steps
88        if (abs(temp_position - mXCalPos1) < 2000){
89          mXCalLastDisplacement = new_displacement;
90          cout << "mXCalLastDisplacement set to: " << mXCalLastDisplacement <<
                endl;
91        }
92        else {
93          mXCalPhase++;
94        }
95        break;
96      case 3:
97        // Wait until we fall/rise again
98        new_displacement = mDisplacementSensor->getDisplacement();
99        cout << "Disp diff2: " << abs(mXCalLastDisplacement - new_displacement) <<
              endl;
100       if (abs(mXCalLastDisplacement - new_displacement) > disp_threshold){
101         mXCalPos2 = mTranslationStage->getPosition(mTranslationStage->
                getHorizontalDeviceId());
102         mXCalPhase++;
103       }
104       break;
105     case 4:
106
107       mXCalPhase++;
108       settings = new QSettings();
109       mm_input = atof(XCalibrateGapLength->text().toLocal8Bit().constData());
110       number_of_steps = abs(mXCalPos2-mXCalPos1);
111       steps_per_mm = number_of_steps / mm_input;
112       cout << "Calibration: X stage, steps per mm: " << steps_per_mm << endl;
113       Settings::refresh();
114       Settings::XCalDate = dateTimeString;
115       Settings::XCalStepsPerMm = steps_per_mm;
116       Settings::save();
117       break;
118
119     case 5:
120       mTranslationStage->goToPosition(mTranslationStage->getHorizontalDeviceId()
              , mXCalInitialPos);
121       mXCalTimer->stop();
122       delete mXCalTimer;
123       mXCalPhase = -1;
124       readSettings();
125       XCalibrateButton->setText("Calibrate");
126       break;
127   };
128
129 }
130
131 void CalibrationDialog::startZCalibration(){
```

```
132   // If user presses button again, stop procedure
133   if (mZCalPhase > 0){
134     mZCalPhase = 3;
135   }
136   else {
137     mZCalPhase = 0;
138   }
139   mZCalCounter = 0;
140   mZCalSpeed = 100;
141   ZCalibrateProgBar->setValue(0);
142   mZCalTimer = new QTimer();
143   connect(mZCalTimer,SIGNAL(timeout()),SLOT(performZCalibration()));
144   mZCalTimer->start(50);
145 };
146
147 void CalibrationDialog::performZCalibration(){
148
149   QSettings* settings;
150   QDateTime dateTime = QDateTime::currentDateTime();
151   QString dateTimeString = dateTime.toString();
152   pair<double, double> result;
153   switch(mZCalPhase){
154     case 0:
155       cout << "Starting␣calibration" << endl;
156       mZCalInitialPos = mTranslationStage->getPosition(mTranslationStage->
              getVerticalDeviceId());
157       mZCalPhase++;
158       ZCalibrateButton->setText("Stop");
159       break;
160     case 1:
161       if (mZCalCounter < 1){
162         ZCalibrateProgBar->setValue(mZCalSpeed * 100 / 1000);
163         mZCalPos1 = mTranslationStage->getPosition(mTranslationStage->
              getVerticalDeviceId());
164         mTranslationStage->setSpeed(mZCalSpeed);
165         if ((mZCalSpeed / 100) % 2 == 1){
166           mTranslationStage->startForward(mTranslationStage->getVerticalDeviceId
                ());
167         }
168         else {
169           mTranslationStage->startBackward(mTranslationStage->
                getVerticalDeviceId());
170         }
171         mZCalCounter = 1;
172
173       }
174       else if (mZCalCounter < 39){ // 2 sec - 50ms
175         //pass
176         mZCalCounter++;
177       }
178       else if (mZCalCounter >= 39){ // last time, save pos
179         mTranslationStage->stop(mTranslationStage->getVerticalDeviceId());
180         mZCalPos2 = mTranslationStage->getPosition(mTranslationStage->
              getVerticalDeviceId());
181         mZCalCounter = 0; //reset
182         mZCalVelocities.push_back(mZCalSpeed);
183         mZCalResults.push_back(double((abs(mZCalPos2 - mZCalPos1))) / 2); //We
              want velocity required for # steps in 1 sec, not 2. Hence half the
              number of steps
184         cout << "Speed:␣" << mZCalSpeed << "␣result:␣" << mZCalResults.back() <<
               endl;
185         if (mZCalSpeed > 999){
186           // Speed test done
187           mZCalPhase++;
188           ZCalibrateProgBar->setValue(100);
189         }
190         else {
191           mZCalSpeed += 100;
```

```
192
193         }
194       }
195       break;
196
197     case 2:
198
199       mZCalPhase++;
200       result = getLinearRegressionValue(mZCalResults, mZCalVelocities); // We
              want v = a + b*steps
201       cout << "Final result.. a: " << result.first << " b: " << result.second <<
              endl;
202
203       Settings::refresh();
204       Settings::ZCalDate = dateTimeString;
205       Settings::ZCalA_Coeff = result.first;
206       Settings::ZCalB_Coeff = result.second;
207       Settings::save();
208
209       break;
210
211     case 3:
212
213       mTranslationStage->setSpeed(700);
214       mTranslationStage->goToPosition(mTranslationStage->getVerticalDeviceId(),
              mZCalInitialPos);
215       mZCalTimer->stop();
216       delete mZCalTimer;
217       mZCalPhase = -1;
218       readSettings();
219       ZCalibrateButton->setText("Calibrate");
220       break;
221   };
222
223 }
224
225 void CalibrationDialog::startLaserCalibration(){
226   // If user presses button again, stop procedure
227   if (mLaserCalPhase > 0){
228     mLaserCalPhase = 3;
229   }
230   else {
231     mLaserCalPhase = 0;
232   }
233   mLaserCalTimer = new QTimer();
234   connect(mLaserCalTimer,SIGNAL(timeout()),SLOT(performLaserCalibration()));
235   mLaserCalTimer->start(50);
236 };
237
238 void CalibrationDialog::performLaserCalibration(){
239
240   // Procedure:
241   // 1. Measure displacement. Save position. Start X stage
242   // 2. Keep measuring until we notice a drop over the threshold. If drop, save
          position and stop.
243   double disp_threshold = 10; //in mm
244   double new_displacement = 0;
245   double mm_input = 0;
246   int number_of_steps = 0;
247   QDateTime dateTime = QDateTime::currentDateTime();
248   QString dateTimeString = dateTime.toString();
249   switch(mLaserCalPhase){
250     case 0:
251       cout << "Starting calibration" << endl;
252       mLaserCalInitialPos = mLaserCalPos1 = mTranslationStage->getPosition(
              mTranslationStage->getHorizontalDeviceId());
253       mLaserCalLastDisplacement = mDisplacementSensor->getDisplacement();
254       mTranslationStage->setSpeed(200);
```

```
255       mTranslationStage->startForward(mTranslationStage->getHorizontalDeviceId()
              );
256       mLaserCalPhase++;
257       LaserCalibrateButton->setText("Stop");
258       break;
259     case 1:
260       new_displacement = mDisplacementSensor->getDisplacement();
261       if (abs(mLaserCalLastDisplacement - new_displacement) > disp_threshold){
262         mLaserCalPos2 = mTranslationStage->getPosition(mTranslationStage->
                getHorizontalDeviceId());
263         mLaserCalPhase++;
264       }
265       break;
266     case 2:
267
268       mLaserCalPhase++;
269       number_of_steps = abs(mLaserCalPos2-mLaserCalPos1);
270       Settings::refresh();
271       Settings::LaserCalDate = dateTimeString;
272       cout << "Steps: " << number_of_steps << " XCalStepsPerMm: " << Settings::
              XCalStepsPerMm << endl;
273       Settings::LaserCalMm = double(number_of_steps) / Settings::XCalStepsPerMm;
274       Settings::save();
275       break;
276
277     case 3:
278       mTranslationStage->goToPosition(mTranslationStage->getHorizontalDeviceId()
              , mLaserCalInitialPos);
279       mLaserCalTimer->stop();
280       delete mLaserCalTimer;
281       mLaserCalPhase = -1;
282       readSettings();
283       LaserCalibrateButton->setText("Calibrate");
284       break;
285   };
286
287 };
288
289
290
291
292 void CalibrationDialog::readSettings(){
293   Settings::refresh();
294   lastXCalibrationValue->setText(QString::number(Settings::XCalStepsPerMm));
295   lastXCalibrationDate->setText(Settings::XCalDate);
296   lastZCalibrationValue->setText(QString("velocity = %1 + %2 * steps").arg(
          Settings::ZCalA_Coeff, 0, 'f', 4).arg(Settings::ZCalB_Coeff,  0, 'f', 4))
          ;
297   lastZCalibrationDate->setText(Settings::ZCalDate);
298   lastLaserCalibrationDate->setText(Settings::LaserCalDate);
299   lastLaserCalibrationValue->setText(QString::number(Settings::LaserCalMm));
300
301 }
302
303 // http://en.wikipedia.org/wiki/Simple_linear_regression
304 // Returns the values (pair(a,b)) in the expression y = a+bx .
305 pair<double, double> CalibrationDialog::getLinearRegressionValue(vector<double>
        x, vector<double> y){
306   if (x.empty() || y.empty()){
307     return pair<double,double>(0,0);
308   }
309
310   double sum_x = 0.0;
311   double sum_xy = 0.0;
312   double sum_y = 0.0;
313   double sum_xx = 0.0;
314   double sum_yy = 0.0;
315   for (int i = 0; i < y.size(); i++) {
```

```
316     sum_x += x.at(i);
317     sum_y += y.at(i);
318     sum_xx += x.at(i) * x.at(i);
319     sum_xy += x.at(i) * y.at(i);
320     sum_yy += y.at(i) * y.at(i);
321   }
322
323   double a = 0.0;
324   double b = 0.0;
325
326   b = (x.size()*sum_xy - sum_x * sum_y) / (x.size()* sum_xx - sum_x * sum_x);
327   a = (sum_y/x.size()- b * (sum_x / x.size()));
328
329   return pair<double, double>(a,b);
330
331 }
```

7.8.3 DisplacementSensor.cpp

```
1  #include "DisplacementSensor.h"
2  #include <tchar.h>
3
4  static const int SENSOR_RANGE = 200;
5  static const int SENSOR_SMR = 60;
6
7  void DisplacementSensor::Error(string err, int sensor){
8
9  }
10
11 DisplacementSensor::DisplacementSensor(){
12
13   initSensor();
14   mCalibratedStepsPerMmValue = -1;
15
16
17 }
18
19 DisplacementSensor::~DisplacementSensor(){
20   ERR_CODE err;
21   err= CloseSensor (sensorInstance);
22   if (err!=ERR_NOERROR){
23     cout << "CloseSensor error!" << endl;
24   }
25   err= ReleaseSensorInstance (sensorInstance);
26   if (err!=ERR_NOERROR){
27     cout << "ReleaseSensor error!" << endl;
28   }
29 }
30
31 void DisplacementSensor::initSensor(){
32
33   // Creating the instance
34   sensorInstance= CreateSensorInstance(SENSOR_ILD1302);
35
36   ERR_CODE err;
37   err= SetParameterString (sensorInstance, _T("IP_Interface"), _T("RS232"));  //
         also valid for RS422 to USB Converter (with RS232 driver emulation)
38   if (err!=ERR_NOERROR)
39     Error (_T("SetParameterString (IP_Interface, RS232)"), sensorInstance);
40
41   err= SetParameterString (sensorInstance, _T("IP_Port"), _T("COM4"));
42   if (err!=ERR_NOERROR)
43     Error (_T("SetParameterString (IP_Port, COM4)"), sensorInstance);
44
```

```
45    err= OpenSensor (sensorInstance);
46    if (err!=ERR_NOERROR)
47      Error (_T("OpenSensor␣(RS232,␣COM11)"), sensorInstance);
48
49    err= SetParameterString (sensorInstance, _T("S_Command"), _T("Get_Settings"));
50    if (err!=ERR_NOERROR)
51      Error (_T("SetParameterString␣(S_Command,␣Get_Settings)"), sensorInstance);
52
53    err= SensorCommand (sensorInstance);
54    if (err!=ERR_NOERROR)
55      Error (_T("SensorCommand␣(Get_Settings)"), sensorInstance);
56
57    int iErr= 0;
58    err= GetParameterInt (sensorInstance, _T("SA_ErrorNumber"), &iErr);
59    if (err!=ERR_NOERROR && err!=ERR_NOT_FOUND)
60       Error (_T("GetParameterInt␣(SA_ErrorNumber)"), sensorInstance);
61    if (iErr!=0)
62      {
63    char cErr[1024], buf[1024];
64    DWORD len= sizeof (cErr);
65    err= GetParameterString (sensorInstance, _T("SA_ErrorText"), cErr, &len);
66    #if _MSC_VER >= 1500
67        _stprintf_s (buf, _countof (buf), _T("Sensor␣returned␣error␣code␣after␣
              command␣Get_Settings\n%d:␣%s"), iErr, cErr);
68    #else
69        _stprintf (buf, _T("Sensor␣returned␣error␣code␣after␣command␣Get_Settings\
              n%d:␣%s"), iErr, cErr);
70    #endif // _MSC_VER >= 1500
71      }
72
73    double range;
74    err= GetParameterDouble (sensorInstance, _T("SA_Range"), &range);
75    if (err!=ERR_NOERROR)
76      Error (_T("GetParameterDouble␣(SA_Range)"), sensorInstance);
77
78    err= SetParameterInt (sensorInstance, _T("SA_AvType"), 0); // Simple moving
          average
79    if (err!=ERR_NOERROR)
80      Error (_T("SetParameterInt␣(SA_AvType)"), sensorInstance);
81
82    err= SetParameterInt (sensorInstance, _T("SA_MovingCount"), 75); // No of
          samples...gives 10 real measurements per sec
83    if (err!=ERR_NOERROR)
84      Error (_T("SetParameterInt␣(SA_MovingCount)"), sensorInstance);
85
86
87    _tprintf (_T("Sensor␣range:␣%.0f␣mm\n"), range);
88
89  }
90
91  double DisplacementSensor::getDisplacement(){
92    ERR_CODE err;
93
94    int avail = 0;
95    err= DataAvail (sensorInstance, &avail);
96    // avail contains now the number of available values.
97
98
99
100   int rawData = 0;
101   double scaledData = 0;
102   err= Poll (sensorInstance, &rawData, &scaledData, 1/*only one value*/);
103   double data_in_mm = 0;
104   if (rawData >=40 && rawData <= 4055){
105     data_in_mm = ((double) rawData * (1.02/4096)-0.01) * (double)SENSOR_RANGE +
            (double)SENSOR_SMR;
106   }
107   // rawData contains the latest raw value received from sensor.
```

```
108   // scaledData contains the latest scaled value.
109   return data_in_mm;
110 }
111
112 void DisplacementSensor::addCalibratedPosition(int stage_position){
113   double disp = getDisplacement();
114   // if out of range (->0mm), don't save value...
115   if (disp < 1){
116     return;
117   }
118   if (!mCalibratedValues.empty()){
119     if (disp < mCalibratedValues.at(mCalibratedValues.size()-1).first){
120       mCalibratedValues.push_back(pair<double, int>(disp, stage_position));
121     }
122   }
123   else {
124     mCalibratedValues.push_back(pair<double, int>(disp, stage_position));
125   }
126   cout << "Displacement:␣" << disp << "␣,␣pos:␣" << stage_position << endl;
127   mCalibratedStepsPerMmValue = -1; // Invalidate cache
128 }
129
130
131 int DisplacementSensor::getCalibratedStepsPerMm(){
132   // Return if cached
133   if (mCalibratedStepsPerMmValue > 0){
134     return mCalibratedStepsPerMmValue;
135   }
136   int size = mCalibratedValues.size();
137   if (size == 0){
138     return -1;
139   }
140   else {
141     double avg_diff_sum = 0;
142     for (int i = 0; i < mCalibratedValues.size()-1; i++){
143       double mm_diff = abs(mCalibratedValues.at(i).first - mCalibratedValues.at(
              i+1).first);
144       int steps_diff = abs(mCalibratedValues.at(i).second - mCalibratedValues.at
              (i+1).second);
145       avg_diff_sum += steps_diff / mm_diff;
146     }
147     mCalibratedStepsPerMmValue = avg_diff_sum/(size-1);
148     return abs(mCalibratedStepsPerMmValue);
149   }
150
151
152 }
```

7.8.4 TranslationStage.cpp

```
1  #include "TranslationStage.h"
2
3  TranslationStage::TranslationStage(){
4    t_isInit = false;
5    t_currentPos = 0;
6    t_currentSpeed = 1200;
7    t_speedDivisor = 8;
8    t_changeCounter = 300;
9
10
11 }
12
13 /* Initializes/refreshes translation stage. Always call before first use */
14
```

```
15 void TranslationStage::init(){
16
17   if( USMC_Init( t_DVS ) )
18     {
19       std::cout << "Translation error!";
20       return;
21
22     }
23
24   mDeviceIdX = 0;
25   mDeviceIdZ = 1;
26   for(int i = 0;i<t_DVS.NOD;i++){
27     int serial = atoi(t_DVS.Serial[i]);
28     if (serial == 4027){ // Serial no for vertical stage
29       mDeviceIdZ = i;
30       std::cout << "Vertical stage ID: " << mDeviceIdZ << endl;
31     }
32     if (serial == 1750){ // Serial no for horizontal stage
33       mDeviceIdX = i;
34       std::cout << "Horizontal stage ID: " << mDeviceIdX << endl;
35     }
36
37     if (USMC_GetMode(i,t_mode)){
38       cout << ("Translation error: mode!") << endl;
39       return;
40     }
41
42     //////////////
43     // Important to set these (limit switches) correct, otherwise the stages don
           't run!
44     if (serial == 1750){ // Don't know why, but this is different for the stages
            (due to one being from Isel I guess)
45       t_mode.Tr1T   = TRUE;
46       t_mode.Tr2T   = TRUE;
47     }
48     else {
49       t_mode.Tr1T   = FALSE;
50       t_mode.Tr2T   = FALSE;
51     }
52
53     t_mode.Tr1En = TRUE;
54     t_mode.Tr2En = TRUE;
55     t_mode.SyncINOp = FALSE;
56     t_mode.SyncInvert = 0;
57
58     t_mode.TrSwap = FALSE;
59
60
61     t_mode.ResetD = FALSE;
62
63
64
65     if( USMC_SetMode(i, t_mode))
66     {
67       cout << "Translation error: mode! Device: " << i << endl;
68     }
69
70     if( USMC_SetCurrentPosition(i, 1000000) ) // We don't want to deal with
           negative values..
71     {
72       std::cout << "Translation error: setposition!" << endl;
73       return;
74     }
75   }
76
77
78
79   t_isInit = true;
```

```
80  }
81
82  void TranslationStage::goToPosition(int deviceId, int pos){
83    if (!t_isInit){
84      this->init();
85    }
86    if (!t_isInit){
87      return;
88    }
89    if( USMC_GetStartParameters(deviceId, t_StPrms) ){
90      std::cout << "Start␣params␣failed!";
91      return;
92    }
93    t_StPrms.SlStart = true;
94    t_StPrms.WSyncIN = FALSE;
95    t_StPrms.LoftEn = FALSE;
96    t_StPrms.SDivisor = t_speedDivisor;
97    t_currentSpeed = t_speed;
98    if( USMC_Start(deviceId, pos, t_currentSpeed, t_StPrms) )
99      std::cout << "Start␣failed!";
100
101 }
102
103 /* Starting forward motion (downward) */
104
105 void TranslationStage::startForward(int deviceId){
106   if (!t_isInit){
107     this->init();
108   }
109   if (!t_isInit){
110     return;
111   }
112   USMC_GetState(deviceId,t_State);
113   t_currentPos = t_State.CurPos;
114
115   goToPosition(deviceId, t_currentPos+400000);
116
117 }
118
119 /* Starting backwards motion (upwards) */
120
121 void TranslationStage::startBackward(int deviceId){
122   if (!t_isInit){
123     this->init();
124   }
125   if (!t_isInit){
126     return;
127   }
128   USMC_GetState(deviceId,t_State);
129   t_currentPos = t_State.CurPos;
130   goToPosition(deviceId, t_currentPos-400000);
131
132 }
133
134 /* Stops motion */
135
136 void TranslationStage::stop(int deviceId){
137   if( USMC_Stop(deviceId) )
138     std::cout << "Couldn't␣stop.␣Time␣to␣pull␣the␣plug?";
139 }
140
141 /* Check whether stage is currently running */
142 bool TranslationStage::isRunning(int deviceId){
143   USMC_GetState(deviceId,t_State);
144   return t_State.RUN;
145 }
146
147 /* Set the speed */
```

```
148 void TranslationStage::setSpeed(float speed){
149   if (speed < 50){
150     speed = 50;
151   }
152   if (speed > 1000){ // Anything over 1000 makes z-stage occasionally slip and
          lose track of position.
153     speed = 1000;
154   }
155   t_speedDivisor = 1;
156   if (speed <= 600){
157     speed = 8*speed;
158     t_speedDivisor = 8;
159   }
160   t_speed = speed;
161
162 }
163
164 /* Returns current position of translation stage */
165
166 int TranslationStage::getPosition(int deviceId){
167   if (!t_isInit){
168     this->init();
169   }
170   if (!t_isInit){
171     return 0;
172   }
173   if (USMC_GetState(deviceId,t_State)){
174     return 0;
175   }
176   else {
177     return t_State.CurPos;
178   }
179
180
181 };
182
183 int TranslationStage::getVerticalDeviceId(){
184   return mDeviceIdZ;
185 }
186
187 int TranslationStage::getHorizontalDeviceId(){
188   return mDeviceIdX;
189 }
```