Tampere University

Mikko Niskanen

FOCUS STACKING IN UAV-BASED INSPECTION

Faculty of Information Technology and Communication Sciences
Master of Science Thesis
September 2019

ABSTRACT

MIKKO NISKANEN: Focus stacking in UAV-based inspection
Tampere University
Master of Science Thesis, 55 pages
September 2019
Master's Degree Programme in Electrical Engineering
Major: Embedded Systems
Examiner: Prof. Timo D. Hämäläinen
Keywords: focus stacking, UAV, drone, inspection, depth of field, focal depth, OpenCV, image registration, image fusion, liquid lens

In UAV-based inspection the most common problems are motion blur and focusing issues. These problems are often due to low-light environment, which can be compensated to some extent with shorter exposure times by using larger apertures and luminous lenses. Large apertures lead to limited depth of field and a solution called focus stacking can be used to extent the focal depth. The main goal of this thesis was to find out the feasibility of focus stacking in UAV inspection and a prototype system was designed and implemented.

The acquisition system was implemented with an industrial type camera and an electrical liquid polymer lens. The post-processing software was implemented with OpenCV computer vision library because libraries offer the best possibilities to affect the low-level functionality. Three algorithms were chosen for the image registration and three for the image fusion. In addition, improvements to the speed and accuracy of the registration were examined. The implemented system was compared to equivalent open-source applications in each phase and it outperformed those applications in general performance.

The most important goal was achieved and the system managed to improve the image data. A sequential acquisition system is not the best option on moving platform due to the perspective changes causing artifacts in image fusion. Also the optical resolution of the liquid lens was not enough for high resolution inspection imaging. However the idea of focus stacking works and the best solution for a mobile platform would be a multi-sensor system capturing the images simultaneously.

TIIVISTELMÄ

MIKKO NISKANEN: Focus stacking in UAV-based inspection
Tampereen yliopisto
Diplomityö, 55 sivua
Syyskuu 2019
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Sulautetut järjestelmät
Tarkastajat: Prof. Timo D. Hämäläinen
Avainsanat: focus stacking, UAV, drone, tarkastuslento, syvyysterävyys, OpenCV, nestelinssi

Lentävillä laitteilla kuvaamisessa yleisimpiä ongelmia ovat liikkeestä aiheutuva epäterävyys sekä tarkennusongelmat. Liike-epäterävyyden minimoimiseen tärkeimmät keinot ovat kuvanvakautus sekä mahdollisimman nopea valotus, jolloin vakautusta ei välttämättä tarvita. Nopea valotus vaatii valovoimaisen linssin ja kuvaamisen suurella aukolla, mikä aiheuttaa syväterävyysalueen kaventumista. Yksi mahdollinen ratkaisu tähän ongelmaan on focus stacking, missä eri etäisyyksille tarkennettu kuvasarja sulautetaan yhdeksi laajan syväterävyysalueen kuvaksi. Tämän työn tarkoituksena oli tutkia focus stacking -järjestelmän toteutuskelpoisuutta UAV-käytössä.

Työssä toteutettiin prototyyppijärjestelmä, johon valittiin nopea teollisuuskamera ja sähköinen neste-polymeerilinssi. Jälkiprosessointi päädyttiin toteuttamaan OpenCV-kirjaston avulla, sillä tämä ratkaisu antoi parhaat mahdollisuudet vaikuttaa järjestelmän matalan tason toimintaan. Toteutukseen valittiin kolme algoritmia kuvien kohdistamiseen ja kolme yhdistämiseen. Myös keinoja järjestelmän suorituskyvyn ja tarkkuuden parantamiseen tutkittiin. Järjestelmää verrattiin kahteen avoimen lähdekoodin ohjelmistoon jokaisessa vaiheessa ja jälkiprosessoinnin suorituskyky jopa ylitti vertailussa käytetyt ohjelmistot.

Projektin tärkein tavoite saavutettiin ja kuvadatan parantaminen menetelmällä onnistui. Sarjakuvausjärjestelmä ei kuitenkaan ole paras mahdollinen ratkaisu liikkuvalle alustalle, sillä muuttuva perspektiivi aiheuttaa häriöitä lopulliseen kuvaan ja pysähtyminen reittipisteissä hidastaisi lentoa huomattavasti. Työssä havaittiin myös nestelinssin optisen resoluution olevan riittämätön korkean resoluution tarkastuskuvauksiin. Focus stacking on kuitenkin ideana toimiva, ja paras ratkaisu liikkuvalle alustalle olisi monisensorijärjestelmä, jolla kuvasarjan kuvat otetaan yhtä aikaa.

PREFACE

Finally this long journey comes to an end. It has been a great experience to work at Intel Finland Oy and I am grateful for the opportunity to write my thesis among all the talented people.

I would like to thank my examiner Timo Hämäläinen and especially my thesis supervisor Ville Ilvonen for important advice and giving me a free hand with my thesis. I would like to thank also my co-worker Arno Virtanen and the rest of our team for sharing thoughts and opinions on my thesis.

The most important personal support for me has been my parents and friends. Thank you!

Tampere, 9.9.2019

Mikko Niskanen

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

ACM	Abstract Control Model
API	Application Programming Interface
(A)KAZE	A novel multiscale 2D feature detection and description algorithm in nonlinear scale spaces
BRIEF	Binary Robust Independent Elementary Features, a feature descriptor
BRISQUE	Blind/Referenceless Image Spatial QUality Evaluator
CMOS	Complementary Metal Oxide Semiconductor, a camera sensor type
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DoF	Depth of Field, the distance range where focus is acceptable
DSIFT	Dense SIFT, pixel-by-pixel SIFT algorithm
DSLR	Digital Single-Lens Reflex, a camera type
EEPROM	Electronically Erasable Programmable Read-Only Memory
ECC	Enhanced Correlation Coefficient, an image alignment algorithm
FAST	Features from Accelerated Segment Test, a feature detector
FLANN	Fast Library for Approximate Nearest Neighbors
FoV	Field of View
GPU	Graphical Processing Unit
GSD	Ground Sample Distance, the size of a pixel on a target plane
GUI	Graphical User Interface
HDR	High Dynamic Range
IQA	Image Quality Assessment
I^2C	Inter-Integrated Circuit, a serial bus type
LTS	Long Term Support
MTF	Modular Transfer Function, the ability of a lens to preserve contrast
ORB	Oriented FAST and Rotated BRIEF
PCNN	Pulse-Coupled Neural Network
PSNR	Peak Signal to Noise Ratio
PTP	Picture Transfer Protocol
RAM	Random Access Memory
RANSAC	RANdom SAmple Consensus, an estimation technique
RGB	Red-Green-Blue
ROS	Robot Operating System
SDK	Software Development Kit
SSD	Solid State Drive

SIFT	Scale-Invariant Feature Transform, a feature detector
SNR	Signal to Noise Ratio
SURF	Speeded Up Robust Features, a feature detector
UAV	Unmanned Aerial Vehicle
UI	User Interface
USAF	United States Air Force
USB	Universal Serial Bus

c	Diameter of circle of confusion
D_n	Depth of field near distance
D_f	Depth of field far distance
f	Focal length
H	Homography (transformation) matrix
h	Hyperfocal distance
N	F-number (aperture)
ω	Angle of view

1. INTRODUCTION

Today's infrastructure maintenance is not just repairing, but also optimizing the use of resources by monitoring. One of the oldest methods is visual inspection which has become relatively easy with recent technologies. When areas and targets difficult to access need to be inspected, a UAV (*Unmanned Aerial Vehicle*) may be the most convenient platform for carrying the camera equipment. This kind of targets can be e.g. power lines [1], wind turbines [62] or bridges [61]. Before the drone era the only way to inspect these targets was using cranes, helicopters or climbers. Utilizing drones in inspection is cheaper, faster and safer [62][60][2] but a flying camera platform causes certain issues.

One of the most common problems is motion blur caused by long exposure times and possible lack of image stabilization. The exposure duration depends directly on the level of light, and dark conditions can be compensated to some extent by using luminous lenses and large apertures. Depending on the camera and lens, large apertures may lead to narrow depth of field leaving some areas out of focus. This is a well known effect especially in macro imagery, where a technique called *focus stacking* is often utilized to extend the depth of field [59][33]. In focus stacking a set of images focused at different distances are aligned and fused into a single image covering the required focal depth. Other issues are usually related to auto-focus when targets are small or low-contrast, e.g. transmission lines or the blades of a wind turbine.

Focus stacking is often used in imagery but there are no publications on utilizing it in drone inspection. Therefore the main objective of this thesis is to find out if the focus stacking could be used in UAV-based inspection to enhance low-light imaging capabilities and also generally improve the image data SNR (*Signal to Noise Ratio*). To achieve this goal a prototype system for acquisition and post-processing will be implemented and evaluated. Due to the experimental nature of this project, some of the results will be represented already during the design phase.

The thesis is divided into five chapters, including this section. The second chapter opens up the basics of drone inspection and the third chapter explains the theory of focus stacking. In the fourth chapter the prototype implementation process and tests are documented, and the last chapter summarizes the results with conclusions and future considerations.

2. DRONE INSPECTION

A *drone inspection* is a flight mission where the objective is to map an object or capture some kind of data of the target. The type of the data depends on the purpose and the features of a drone being usually images or video, but it can also be something else like point clouds or magnetic field intensities. This section introduces the typical equipment and processing tools for acquiring and utilizing the data, focusing on imagery only. The most common issues arising are also explained in the last section.

2.1 Acquisition equipment

A typical inspection payload configuration is a gimbal holding the imaging equipment, usually at least one high resolution RGB (*Red-Green-Blue*) camera. Infrared thermal camera is also a common accessory and more advanced systems can utilize this together with the RGB for hyperspectral imaging. The gimbal can be mounted anywhere on the body of the drone and have some stabilization system to absorb any swings caused by wind.

Modular solutions are versatile and nowadays small industrial-type cameras and integrated solutions are becoming more common. In many cases the industrial cameras on the market do not have the resolutions or features like image stabilization or auto-focus found in DSLR (*Digital Single-Lens Reflex*) cameras, but on the other hand small sensors make smaller and cheaper lenses possible. The main camera should have a sensor large enough to prevent excessive electronic noise when using high sensitivities, and sometimes the more sensitive monochromatic cameras are used to enhance the low-light abilities. While some features are more important than others, there is not a single aspect alone to define how a camera system performs.

If the camera is a DSLR or an industrial model, it likely will not have a fixed lens and it has to be chosen separately. Obviously the lens has to be compatible with the sensor size (*format*), and the most important optical features are:

- Luminosity
- Optical resolution
- MTF (*Modular Transfer Function*)

Luminosity is defined with a dimensionless f-number determining how much light can pass through the lens. The f-number is calculated as the ratio of the focal length and the diameter of the aperture in the lens. In practice, the luminosity defines how short exposure times can be used and is therefore important when using mobile platforms.

The *optical resolution* simply determines the ability to preserve details and the unit is the same as with sensors, *megapixel.* The *modular transfer function* is a similar concept and determines the ability to preserve the regional variations in brightness – contrast. In practice the MTF describes how well small details are separated in the final captured image. It is usually represented in an MTF chart represented in figure 2.1 describing the transfer ratio for different number of line pairs per millimeter (*lp/mm*). [39]

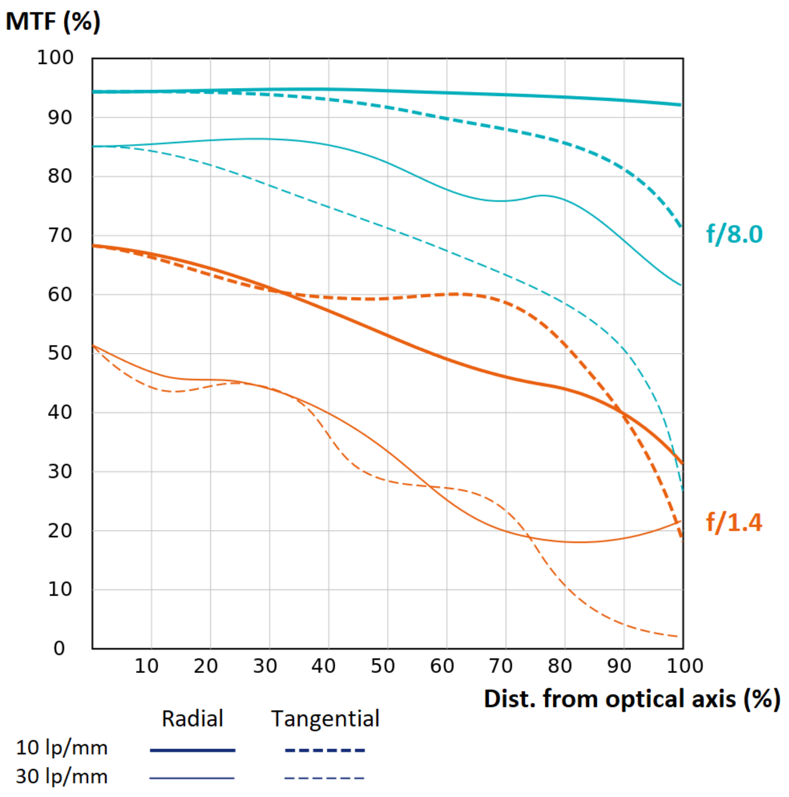


Figure 2.1 *An example of a modular transfer function chart, adapted from [71].*

Some lenses are variable in focal length (zoomable) but this may affect negatively to the overall quality due to increased motion blur. However, zooming can fit the wanted subject area better on the available amount of pixels and therefore enhance the GSD (*Ground Sample Distance*), which is a term used in digital photography to define the real-world size of a pixel on the target surface. Zooming is also slow, so it is not typically used on automated inspection flights.

2.2 Processing

While it is possible to just browse the collected data, it is often sufficient to use other techniques as well. Modern computing platforms, local and cloud based systems, have made it possible to process thousands of images within a reasonable time. In the inspection data analysis, *photogrammetry* is nowadays very popular because it makes it possible to use only 2D images to create precise surface maps and 3D models.

Photogrammetry is based on triangulation where two or more overlapping images captured from different camera positions are used. The common keypoints from those images are detected and the point coordinates calculated using the converging virtual lines of sight. When this is repeated for a set of overlapping images it is possible to form a point cloud of the subject. This is the basic principle behind most of the 3D reconstruction applications. A visualization of the idea is represented in figure 2.2. [24][26]

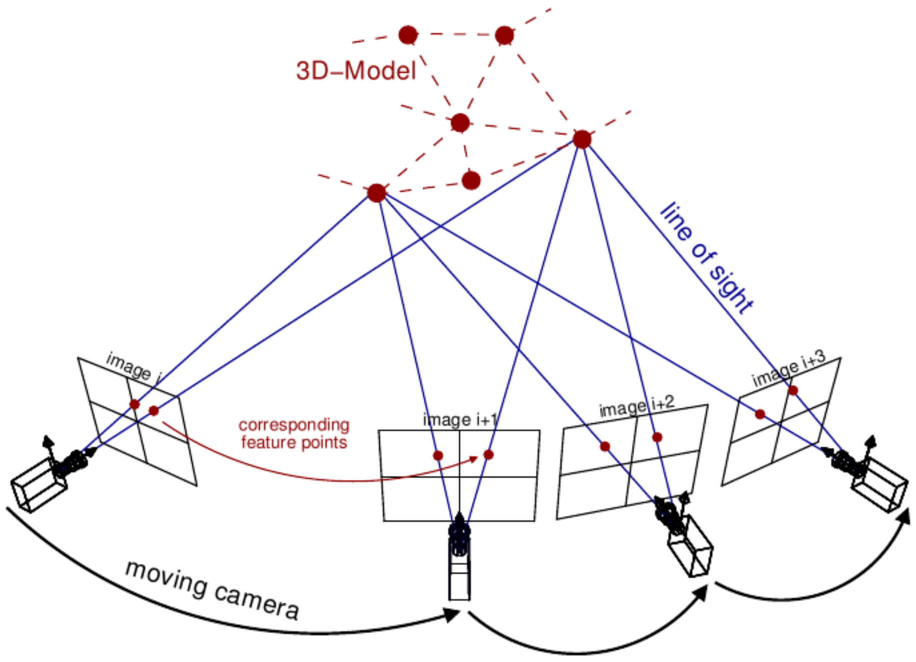


Figure 2.2 *The main principle of photogrammetry reconstruction from the documentation of Theia computer vision library [66].*

This kind of data processing is computationally expensive and it may take up even a couple of days for software tools to finish constructing big models, e.g. buildings [46]. Fortunately the point cloud calculation is highly parallelizable and can benefit a lot from high-performance graphical processing units. This requires also massive amounts of memory and high-speed internet connections when using cloud-based products [46]. There are many tools available for photogrammetry based 3D reconstruction e.g. Bentley ContextCapture [5] and Agisoft Metashape [3].

The biggest problem in inspection photogrammetry is usually the background that also gets rendered as the point cloud, increasing the amount of useless data. To solve this problem there are background detection and removal algorithms available, but also depth sensor data could be utilized. Other methods include a variety of image processing and machine learning techniques to detect abnormalities from the captured data. For example, there could be detection of corrosion on the blades of a wind turbine, rust on bolts of a transmission tower or defective solar panels.

2.3 Challenges

Auto-focus issues depend on the implementation of a system and the focusing techniques are generally based on either passive or active solutions. Active methods use external systems to measure the distance to the target e.g. infrared or laser devices. Passive methods use either phase detection or contrast measuring, and the faster phase detection is more used nowadays even if it increases the complexity of a system.

The basic working principle of the phase-difference auto-focus is represented in figure 2.3. The incoming light is reflected onto the focusing sensors through microlenses (in the AF lens array) acting as the focusing points in the image area. From the location difference of the same object on the AF sensors it is possible to find out to which direction and how much the lens should be adjusted, reducing straddling typical for the contrast based method and speeding up the focusing system. [31][8]

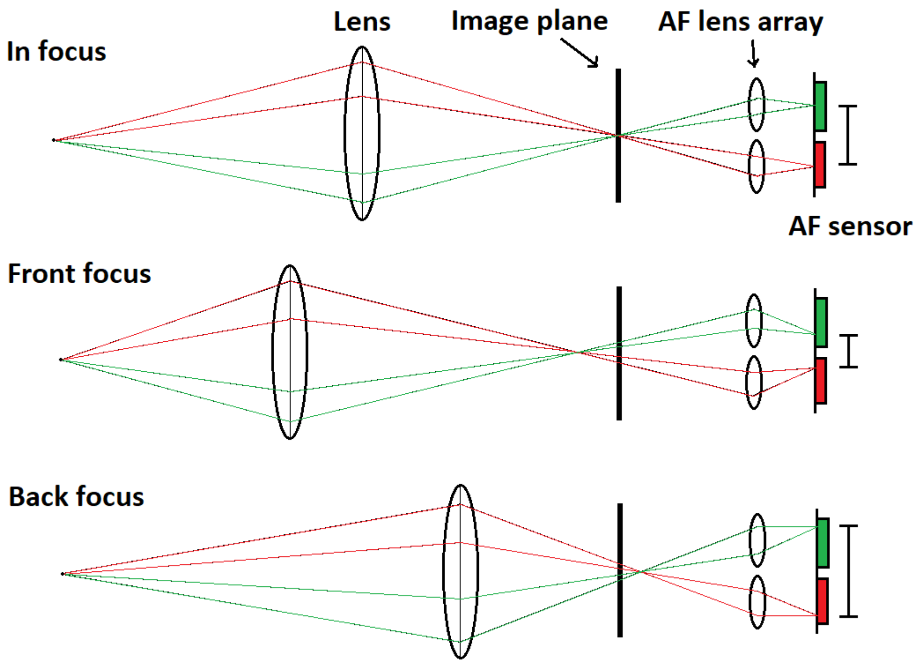


Figure 2.3 *The basic principle behind phase-difference auto-focus, adapted from [8].*

Both of these techniques require the subject having high enough contrast with many detectable features like sharp edges. If ambient light is weak or the subject is very small or smooth, there may not be enough contrast edges for the focusing system to compare. This is why occasional focusing at wrong distances happen even with the best focusing systems. The active methods may work even in complete darkness depending on the technique used, but are rare in normal cameras because of their complexity or price.

As already mentioned, the exposure times on mobile platforms should be decreased as much as possible to reduce motion blur. This can be achieved using high sensor sensitivity or large apertures, but using high sensitivity causes electrical noise and large apertures limited depth of field. The effect of aperture size on depth of field is represented in figure 2.4.

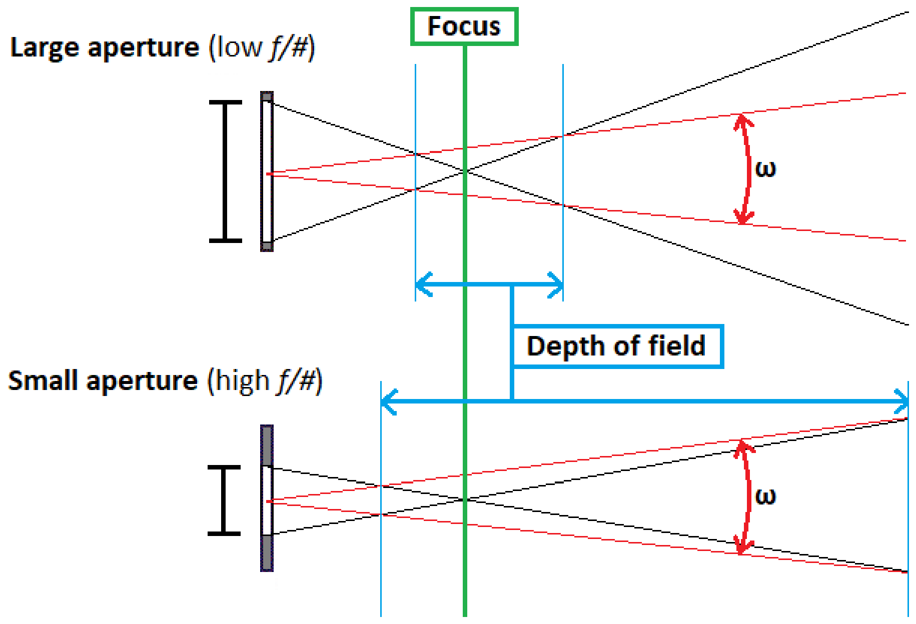


Figure 2.4 *Depth of field explained, adapted from [18].*

The depth of field implies the range of distance that can be considered sharp. In figure 2.4 the *angle of view* is represented as the red cone with ω. When the aperture is large, the light rays from wider area can enter the lens but are naturally from shallower depth range due to the restricting angle of view. [18][27, p. 24–26] In general the depth of field is subjective concept because the amount of blur needed or allowed depends on the desired application. In photography shallow depth of field is normally used to fade out the background (the *bokeh-effect*), and this can also be utilized to help machine vision e.g. when only the targets on the foreground are needed in a point cloud.

Depth of field can be calculated quite easily with *hyperfocal distance*. If a lens is focused at infinity, half of this distance is the point after which everything seems to be in acceptable focus. This focus however is not the sharpness possible because a lens can be focused absolutely at one distance only, and it also requires the lens to be able to focus at infinity. For applications where only fixed focus can be used, e.g. in camera surveillance, the lack of sharpness may not be critical. Hyperfocal distance h for a system can be calculated as

$$h = \frac{f^2}{Nc} + f \tag{2.1}$$

where f is the focal length of the lens, N is the f-number and c is the diameter of the circle of confusion.

Circle of confusion is the blurred circle forming on a sensor or film surface if an ideal spot of light was captured, and it is usually defined with the total resolution required. There are also theoretical limits for this value, for example an ideal lens with relative aperture of f/4.5 can reproduce a circle of confusion of only 0,003 millimeter maximum diameter directly on the axis due to aperture diffraction. This diffraction is also a noticeable reason for images being sharper when captured with large apertures, as seen below in figure 2.5. [27, p. 26–27]

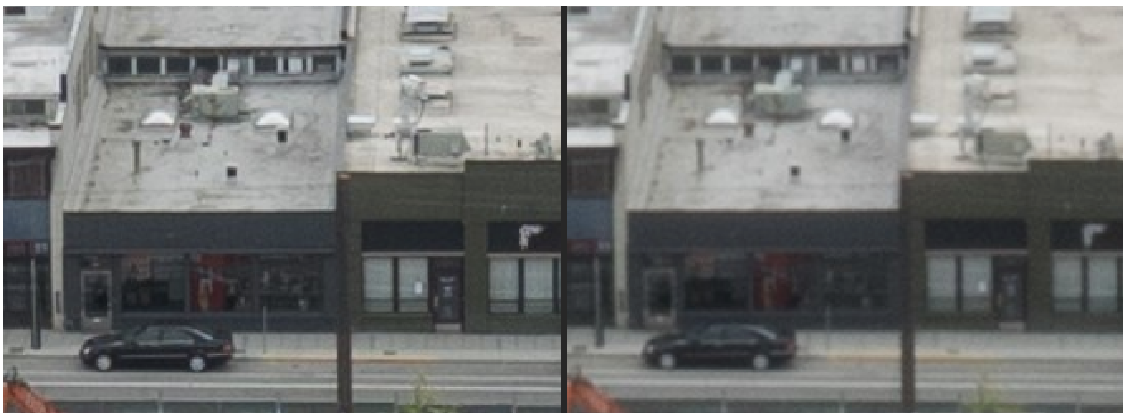


Figure 2.5 *The effect of aperture diffraction on image resolution. Cropped from images captured by Samuel Spencer with Sony Cyber-shot DSC-RX10 III using f/2.8 on the left and f/11 on the right image. [64]*

With the hyperfocal distance it is possible to calculate the near and far distances of depth of field. The near distance D_n is solved with

$$D_n = \frac{s(h-f)}{h+s-2f} \tag{2.2}$$

and the far distance D_f with

$$D_f = \frac{s(h-f)}{h-s} \tag{2.3}$$

where s is the focus distance, h the hyperfocal distance and f the focal length of the lens [27, p. 26–27]. The table 2.1 contains theoretical example values calculated at different f-stops for a 16 mm lens focused at 3 m distance with circle of confusion of 0,006 mm. In reality the diameter of circle of confusion would change, becoming larger with smaller apertures.

Table 2.1 *Theoretical hyperfocal distances (infinity focus) and depths of field (focus at 3 m) for a 16mm lens.*

f-number	h (m)	DoF (m)
1.4	30,5	0,6
2.0	21,4	0,9
2.8	15,3	1,2
4.0	10,7	1,8
5.6	7,6	2,8
8.0	5,4	4,9
11.0	3,9	11,3

From the calculated values it is easy to notice how the apertures affect the depth of field and minimum infinity focus distances in a drone inspection. Often drones fly less than five meters away from the target and in that case using the determined parameters the focus at infinity would be usable only with apertures smaller than f/4.0. Also in practice the luminosity of this f-stop would be applicable in bright daylight but may cause problems on an overcast day or in other low-light conditions. The shallowness of the depth of field when using large apertures may become a problem especially when using a fixed focus lens because then the only way to adjust the sharpness is to drive the drone to the right distance.

3. FOCUS STACKING

3.1 Introduction

One of the problems is the limited depth of field when using large apertures. The obvious solution would be using smaller apertures and higher sensitivities with good image stabilization, but if this is not possible something else has to be figured out.

The solution proposed in this thesis is using *focus stacking*, which is a technique used mainly in close-range imaging where DoFs are extremely limited. The basic principle represented in figure 3.1 is to take multiple images focused at different distances and fuse the acquired image stack to one image having extended DoF.

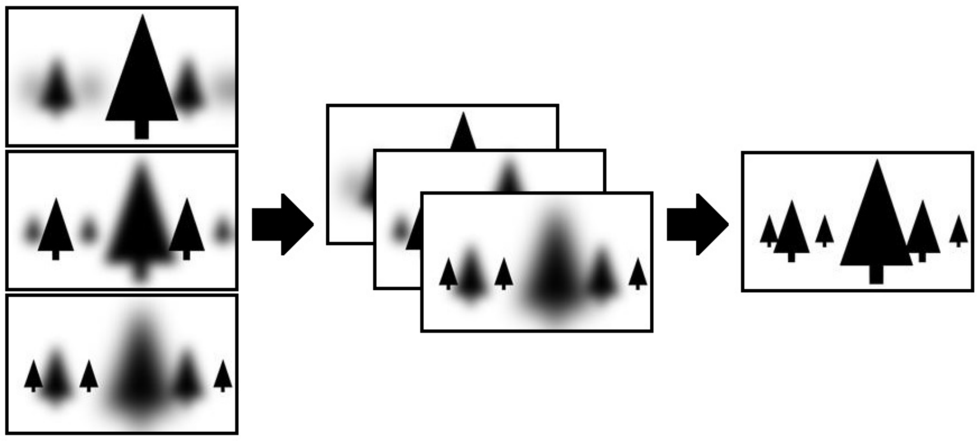


Figure 3.1 *The idea of focus stacking, adapted from [13].*

At first capturing the image stack might sound trivial, but there are a couple of important basic things to take into account:

- **Camera movement:** Obviously if the perspective or camera position changes, also the relative position of subjects at different distances changes in the captured images. It is possible to reposition subjects on images in laboratory conditions but this is not very feasible in normal imagery where background is usually too complex [57]. Because of that the camera system and focus control should be as fast as possible.

- **Focus distances:** The steps in focus distances must be sufficient to have enough overlapping sharp areas for the image alignment (registration). If there are too few points for the registration algorithms, misalignment occurs between the images if the alignment doesn't fail completely.

For the fast focusing there are few different approaches. One is to use just fast enough traditional motorized glass lenses, which in principle should offer the best optical performance. There are in-camera solutions using this technique, for example the Olympus OM-D [50] or Lumix G-series [56]. Another option is to use some advanced solution, for example a liquid lens [7] that may be extremely fast but the optical performance may not be on par with the traditional lenses. The third solution is constructing a multi-sensor system, utilized e.g. in the Light.co L16 pocket camera [36] and in the Nokia 9 smartphone [48]. In multi-sensor solutions the main objective usually is not to increase the depth of field but make it easier to control the bokeh-effect afterwards as an effect, or even construct super-resolution images. When considering speed, the most convenient solution is obviously the multi-sensor where there are no separate focusing steps slowing down the acquisition process.

All the mentioned speed aspects are important if the objective is to form only one composite image, but of course not compulsory in inspection purposes where the images can be handled separately. However, the stacking can help handling the data by providing all the information in only one image, but if the final result contains too much artifacts (misalignment, halo, blur), it may have the opposite effect.

3.2 Image registration

The first step in the post-processing is *image registration*, where each image in the stack is warped to match one anchor image. Warping is based on different motion models illustrated in figure 3.2.

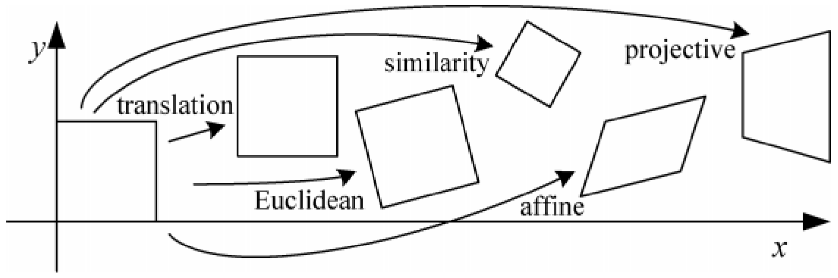


Figure 3.2 *Planar motion models [67, p. 5].*

The planar (2D) motion models define how an image can be transformed. There are five basic transformations [67, p. 5–7]:

- **Translation**, where the the image is shifted.
- **Euclidean**, where image is shifted and rotated.
- **Similarity**, where in addition to the Euclidean also the scale is changed.
- **Affine**, where the image is also sheared.
- **Projective** transformation or *homography* is the combination of all above, but in addition changes the perspective of the image.

The time complexity increases with the transformation models, so it is convenient to prevent searching for any unnecessary types of transformations to speed up the application. In theory the similarity transformation should be enough for focus stacking but if the camera turns even a small amount, projective transformation have to be used. This situation is quite unusual in focus stacking where the camera is normally mounted in fixed position unlike in e.g. handheld scene photography or drone inspection.

The projective transformation denotes linear transformation between two planes and can be represented as

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = H \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.1}$$

where H is the homography matrix, (x'_1, x'_2, x'_3) the warped coordinate and (x_1, x_2, x_3) the original coordinate [29, p. 33–34].

There are two main approaches to find the parameters for transformation, *pixel-based* (also known as *direct*) and *feature-based* methods. Pixel-based methods compare images pixel by pixel, when feature-based try to extract distinctive features from the images and then find the corresponding ones.

Pixel-based. In practice the direct methods just search for alignment where most of the pixels agree. There are different algorithms to find the corresponding pixels, for example the well-known block matching where a cost function is calculated for a block of pixels in every possible location in the image and then compared directly to the candidate blocks in the anchor image. Computationally this is very intensive and therefore different improvements to the basic principle have been invented, for example using Fourier transforms, early jump-out and coarse-to-fine pyramid techniques. [67, p. 20–41][11]

Feature-based. In this context a feature means basically a segment of pixels having some recognizable form and properties. The basic idea in feature based detection is at first to determine the interest points that are usually located in high contrast areas containing the largest amount of sharp shapes and variance in intensity. There are many ways to determine these candidate areas containing edges, the most common being based on weighted convolution like derivative, gradient or Laplacian filters. This type of a *maxima/minima* mask is formed by reducing image noise by blurring and then calculating the peak values using the mentioned calculations. The resulting contrast mask can be used also in the image fusion stage, which will be explained in detail in the next subsection.

The points of interest, also known as *keypoints*, are usually primitive shapes such as corners and lines presented in figure 3.3. [67, p. 42–46][34]

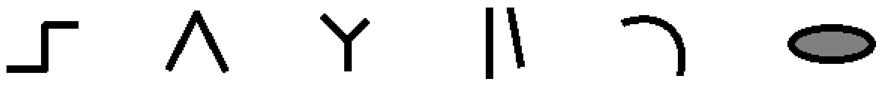


Figure 3.3 *The most common shapes of keypoints in feature-based detection, adapted from [34, p. 217].*

Finding the features. The algorithms used to find the keypoints are called *detectors*, and the algorithms to describe the features of a keypoint are called *descriptors*. Some algorithms work better in certain conditions than others depending on e.g. the resolution, contents and dynamic range of an image. One of the most well-known detector algorithms is the *Harris corner detector* which detects a corner point from the intensity changes in a local window around an interest point [34], and another more recent example is the FAST (*Features from Accelerated Segment Test*, 2006). After detection the keypoins are described, usually a vector of values is computed from a keypoint so that it can be identified invariant to transformation and this information is used later in the matching phase. For example, the BRIEF (*Binary Robust Independent Elementary Features*, 2010) is a common binary descriptor algorithm. Described keypoints are also often called *feature points.* [67, p. 42–46][34] There are many algorithms to choose from, of which the most famous are [68]:

- SIFT (*Scale-Invariant Feature Transform*), 1999
- SURF (*Speeded Up Robust Features*), 2006
- ORB (*Oriented FAST and Rotated BRIEF*), 2011
- KAZE/Accelerated KAZE (*A novel multiscale 2D feature detection and description algorithm in nonlinear scale spaces*), 2012

After detection and description, features can also be filtered regarding the information content to find the most usable ones before matching. For example, the SIFT algorithm does that because there is no reason to use keypoints having poor contrast or uncertain location in a large size image with severe transformation [45].

Matching the features. Matching can be done using *brute force* comparing every single feature point to each other but in this case the time complexity is n^2. The next logical solution would be using *indexing schemes*, an idea based on the fact that usually corresponding keypoints are geometrically relatively close to each other in the images. These techniques can exploit e.g. slicing or nearest neighbour algorithms (e.g. found in FLANN (*Fast Library for Approximate Nearest Neighbors*)[44][43][42]) to gather up initial lists of the most likely corresponding feature points which are then compared using more accurate methods. Some filtering can also be applied to the initial matches, like the well-known *Lowe's ratio test*, where the ratio of the distance to the closest and second closest neighbour is compared to a threshold value (often 0.7, based on Lowe's observations) [38, p. 19–20].

For the accurate comparison, one popular approach is to use all the data with least squares estimation and another is to use samples of the data for estimation. A widely used example of the sample based estimation is the RANSAC (*RANdom SAmple Consensus*) algorithm, where the idea is to find a set of *inliers* meaning the subset of points having coherence in some specific dimension, or more generally, points fitting the "true" model within specified error threshold. After this the model parameters for geometric transformation can be estimated using only the chosen sample leading to the best result, unlike in the least squares method where parameters are estimated using all the feature points. [67, p. 46–50]

Solving the parameters. Many of the geometric transformations are linear and can be solved with linear regression, for example translation, similarity and affine, but the nonlinear models like homography require iterative solutions [67, p. 51]. To clarify the idea, the perspective transformation can be written also as

$$\begin{aligned} \hat{x}' &= \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \\ \hat{y}' &= \frac{(1+h_{11})y + h_{10}x + h_{12}}{h_{20}x + h_{21}y + 1} \end{aligned} \tag{3.2}$$

where $(\hat{x}', \hat{y}')$ is the estimated point, (x, y) the original point and $(h_{00}...h_{21})$ are the transformation coefficients [67, p. 52]. According to Szelinski, when iteratively reweighted least squares is commonly used, some other methods like Gauss-Newton approximation can be better, leading to a simplified formula [67, p. 53]

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix} \tag{3.3}$$

where there are the same parameters present as in the previous formula. In the end, the warping is done by calculating the new pixel coordinates with the resolved transformation parameters.

Other techniques. As the *deep neural networks* have gained popularity, there are promising solutions proposed also for homography estimation, for example the HomographyNet [14], another solution [49] based on that and an unsupervised method [47]. This method released in 2016, examines two different convolutional neural network architectures using the principles of classification and regression. According to the results [14][49] the neural networks perform even better than the reference setups using ORB and RANSAC in terms of speed and error. The neural network solution has also the advantage of adapting to certain environment or properties i.e. motion blur. In advance to the use for image registration, this method would be especially suitable for visual odometry, and solutions have already been proposed [35].

3.3 Image fusion

The last stage in the focus stacking is to fuse the images, where the basic idea is to find the areas in focus from the images in the stack and combine those into one sharp image. Most of the problems in image fusion are related to artifacts due to misalignment or motion and the *halo-effect*, glowing light edges forming around objects. There are many different fusion algorithms today, divided into four different domains [37]:

- Multi-scale transform methods
- Feature space transform methods
- Spatial domain methods
- Pulse coupled neural networks

The multi-scale transform is based on "decomposition-fusion-reconstruction" workflow where images are at first decomposed into multi-scale image stacks to which a transform is applied. The resulting coefficients are then fused and those fused coefficients are used to construct the final image. Some of the well-known methods are Laplacian pyramid, Gaussian pyramid and discrete wavelet transform. [37]

In the feature space methods the idea is in measuring the activity level (or clarity) of images and utilize this information with sliding window technique to make shift-invariant fusion possible. This is based on the idea that in detected features there is also information about the focus. For example, the DSIFT (*Dense SIFT*, pixel-by-pixel SIFT algorithm) can be used to detect local features to approximate the focus level of a window. [37]

The spatial domain methods usually work pixels or blocks, evaluating the focus of an area or pixel with some measurement [37], for example spatial frequency and variance of gray levels [30]. Techniques using segmentation suffer more from blocking artifacts, but apparently some novel pixel-by-pixel techniques have reached good results maintaining the spatial consistency in the final image [37].

The PCNN (*Pulse-Coupled Neural Network*) is based on biological neural system, so it models animal/human perception. Under the hood, also neural networks use the principles explained above. The weakness of PCNN is the large amount of free parameters causing variance in performance. [37]

There are solutions for the issues related to moving objects causing ghosting. In the comparative review from 2012, Srikantha and Sidibé evaluated different ghosting detection and removal methods in HDR (*High Dynamic Range*) imaging based on e.g. variance, entropy, prediction, pixel order, multi-thresholding and bitmaps. The results were promising but there were still not a single best algorithm. [65]

4. PROTOTYPE IMPLEMENTATION

This section describes the design and implementation flow of the prototype system. Tests were conducted along the implementation because many decisions are based on the results and experiments. The focus was on the system functionality and existing components were utilized as much as possible.

4.1 High-level description

This prototype was designed primarily for testing purposes but keeping the possible drone deployment in mind. The drone platforms often use Linux-based operating systems on companion computers with ROS (*Robot Operating System*) as the middleware [51][4][10]. The ROS is a framework providing tools and libraries for modular robotic system development and operation with node-based process control and inter-process communication system [23]. However, this prototype wasn't implemented directly on ROS because it wasn't necessary for testing the concept and would have made the development more complex.

The system is divided into two sections, acquisition and post-processing. The high-level architecture is represented in figure 4.1.

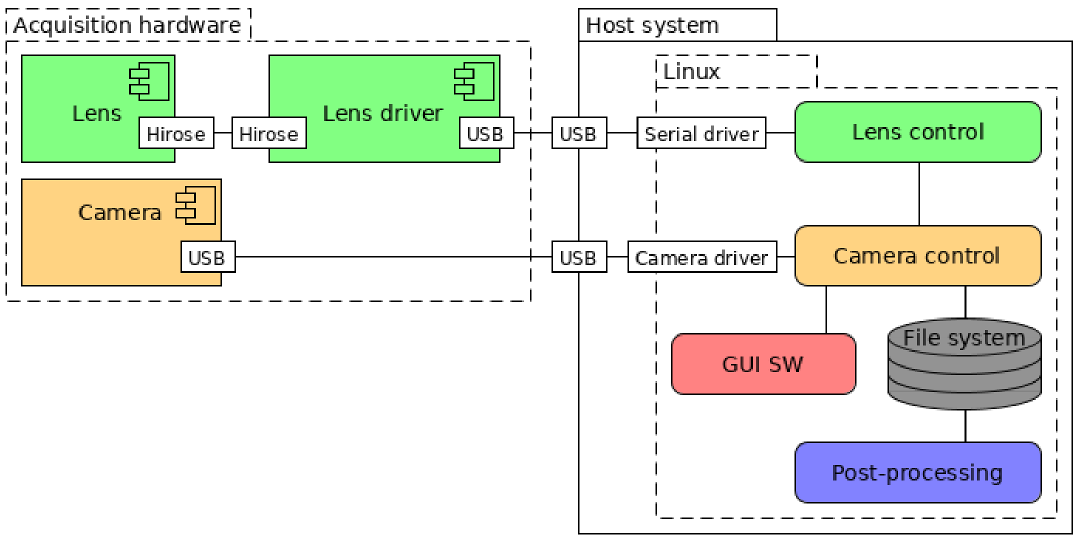


Figure 4.1 *High-level system architecture.*

In the block diagram the camera and lens are presented as separate, and the choices and more accurate system descriptions are explained in the following sections. On the host side there are the control software for camera and lens running on Linux, and to make the testing easier, a simple streaming user interface. The actual focus stacking post-processing was implemented as a separate background process where the image alignment and fusion happens. The captured images are saved to the file system before post-processing to emulate the separate on-board and backend systems because the post-processing may be too heavy and slow to be executed on-board.

The development system was a desktop PC with a 4-core Intel i7-7700K CPU (*Central Processing Unit*), 32 GB RAM (*Random Access Memory*) and a 512 GB M.2 SSD (*Solid State Drive*) running Ubuntu 18.04.2 LTS (*Long Term Support*) operating system. In this case a separate GPU (*Graphical Processing Unit*) was primarily not used. Python was selected as the main programming language because it is handy for agile prototyping and there were also other project related reasons to use it. Keeping the possible ROS platform in mind, Python version 2.7 had to be used because the ROS version 1 does not support the newer Python versions.

4.2 Hardware

4.2.1 Camera

Requirements for the camera were straightforward; it should be lightweight, fast and have a sufficient API (*Application Programming Interface*) to allow integration with the focusing system. Some initial test images were shot (handheld) using a Sony RX-1 R II compact camera, which has a 42 megapixel full frame (35 mm) sensor and a 35 mm fixed lens. For the prototype configuration two different options were considered:

- A mid-sized dedicated UAV camera-lens system based on DSLR (*Digital Single-Lens Reflex*) cameras.
- A lightweight industrial camera with suitable lens setup.

The UAV DSLR camera has very similar properties to a regular DSLR camera such as regular lens mount, image stabilization and auto-focus. The camera uses PTP (*Picture Transfer Protocol*) over USB (*Universal Serial Bus*) for control messages and data transfer. Unfortunately, the control system of that camera was complex and uses undefined button emulations to control the focus, so other options were lifted in priority.

For this project, a FLIR industrial camera was chosen based on availability, hardware properties and intuitive API. It is small and relatively cheap, it has a 25,4 mm (1" format) 20 megapixel CMOS (*Complementary Metal Oxide Semiconductor*) sensor capable of acquiring images at a maximum frame rate of almost 20 fps and it uses the common industrial C lens mount. The 1" sensor format is in the large end of the industrial cameras, most being smaller, e.g. 2/3" or 1/3". Large sensor helps to reduce noise on high sensitivities but may make finding a suitable lens configuration more difficult. Also another similar but over three times faster 5 megapixel industrial camera model was used as comparison. [20]

The FLIR is fully USB-powered and has many options for multi-frame capture and synchronization. It has automatic white balance and exposure controls and possibility to operate in one-shot or continuous mode. In order to achieve the highest acquisition speeds possible, a hardware buffer can be used. There are also options to use either hardware or software triggers providing more possibilities to synchronize the capturing with the focusing system.

4.2.2 Lens

With the selected FLIR camera, there were three different options as the lens setup:

- Ready-to-go motorized lens.
- Self-made electro-mechanical solution with manual lens.
- A liquid lens system, integrated or modular.

The motorized lens option seemed to be the way to go at first, but after a brief research it became clear that there are no C-mount lenses suitable for normal imagery, especially for the 1" sensor format. Because C-mount is in practice only used in movie, industrial and surveillance cameras, the lenses don't usually have motorized focus but motorized aperture or zoom. The few options found were also so heavy-weight and had so slow focus control (i.e. 5 seconds from end to end) that those couldn't have been considered. Those lenses also would have required some partially or fully self-made implementation to control the lens.

The self-made solution could use a fixed focal length industrial lens and e.g. a stepper motor to rotate the focus ring. The transmission would be implemented with 3D-printed parts using either a belt drive or gears. A very similar commercial solution was found shortly after considering the idea, DJI Focus [15], which utilizes the exactly same principle with a remote control and it can be used with almost any manually adjustable lens. Implementing this solution would have required a lot of extra work but for prototyping it would be a feasible idea.

The liquid lens system was finally chosen for this project due to the speed and the light weight of the existing products. Also, based on search, liquid lenses have not been tested in UAV applications before. However, apparently this technology has been researched in mobile phone development [25][28] in advance to microscopical and industrial imagery, but released products have not been seen yet.

There are two alternative options suitable for this project, Varioptics variable focus liquid lenses based on the principle of electrowetting and Optotune focus-tunable shape-changing polymer lenses. The electrowetting works by using electric fields to adjust the shape of a liquid lens formed with oil and water in a container [17]. The principle of the polymer lens is quite similar having two chambers filled with two different materials except those are separated with the elastic polymer membrane and there is a coil-actuator changing the pressure [6]. The working principles are illustrated in figure 4.2.

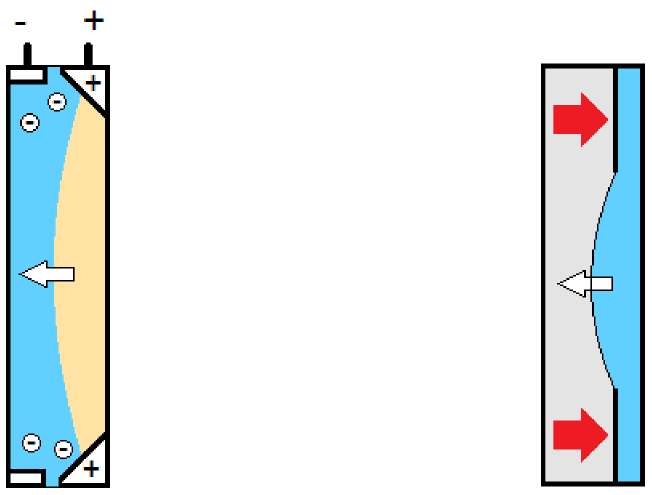


Figure 4.2 *The principles of electrowetting (left) and shape-changing polymer membrane (right).*

The problem with Varioptics is the small aperture sizes of just a few millimeters (2,5–3,9 mm) that become a problem when wide FoV (*Field of View*) and apertures are required. There are also lenses with integrated Varioptic lenses but those have apertures of only f/5 and a FoV of 17,54° at best [16]. This problem can not be fully solved with Optotune either, but they offer lenses with larger apertures (3–16 mm) [17] which can be used to achieve about 28° FoV without vignetting (cropping the edges) with certain off-the-shelf lenses [55]. While about 40–60° would be the optimal FoV, 28° is enough for prototyping purposes. Also because the current driven control is fast allowing maximum settling times of 25 ms and there are enclosed models available operating in temperatures from -20 to 65 °C [54], the Optotune was chosen for this project.

The demand for working distance range was from 2 m to infinity so a *front lens configuration* had to be used meaning that there is a normal fixed focal length lens between the camera and focus-tunable lens. With Optotune's configuration table and configuration tool the model EL-16-40-TC-VIS-5D was selected providing a focal power range from -2 to +3 diopters. According to the table, a 35 mm lens would be the shortest focal length option for 1" sensor, but from Optotune's application notes an example configuration was found using a 30mm f/2.0 Schneider Xenon-Topaz series lens [55]. Because this lens was already tested and confirmed to work without vignetting, it was the safest choice. The assembled camera system for prototyping is shown in figure 4.3.

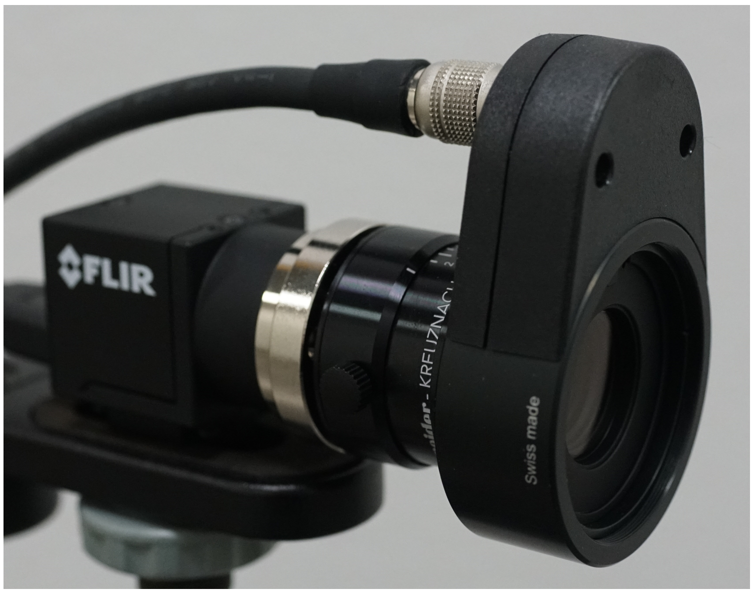
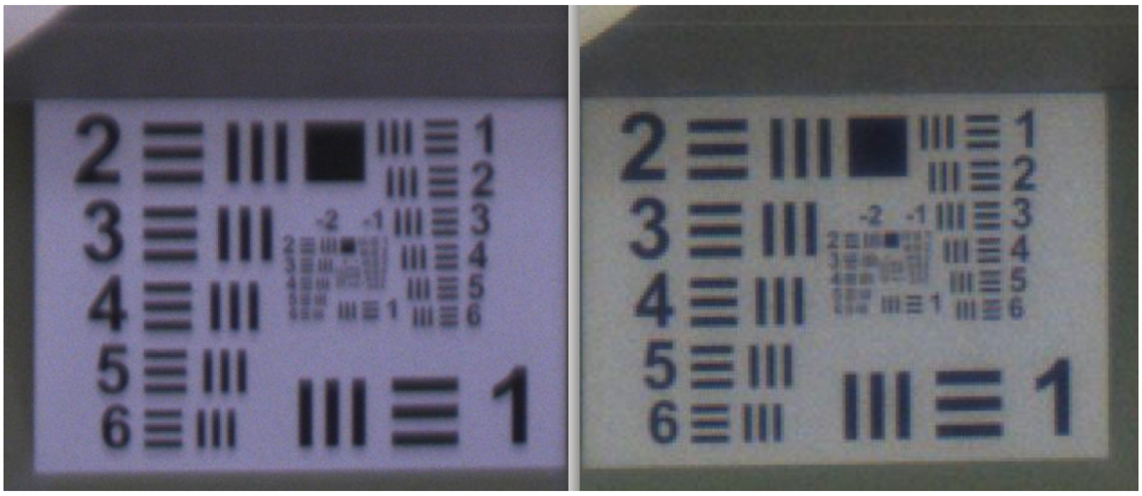


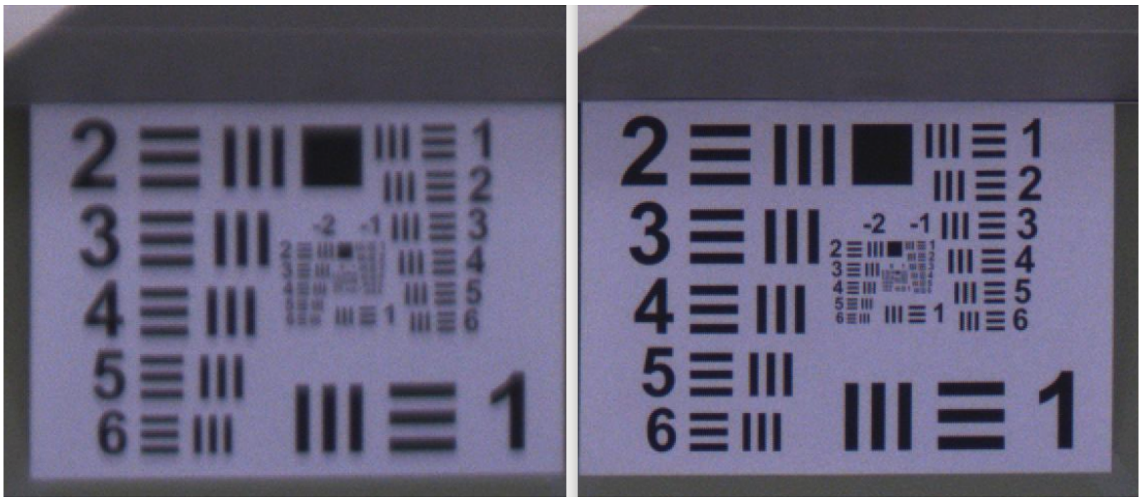
Figure 4.3 *The camera-lens configuration; a FLIR industrial camera, a Schneider Xenon-Topaz 30mm lens and an Optotune EL-16-40-TC focus-tunable lens.*

The separate lens controller, an industrial model of Optotune Lens Driver 4i was available in a USB-stick style closed metal enclosure. It uses a 6-way Hirose HR 10 G cable to transfer the control current to the lens and to read the temperature sensor and lens EEPROM (*Electronically Erasable Programmable Read-Only Memory*) via I^2C (*Inter-Integrated Circuit*) protocol. The driver module is fully USB-powered like the camera and follows a simple serial protocol which makes it easy to integrate to any host system. [53]

Unfortunately later when the lens was available, it was noticed that the optical resolution and luminosity may not be good enough to improve the quality compared to another lens with shorter focal length and adequate infinity focus. For example, a 16 mm lens with the same camera produced almost identical resolution but naturally with better FoV and also slightly sharper edges when compared to the 30 mm + liquid unit, which can be seen in figure 4.4a.



(a)



(b)

Figure 4.4 *The resolution comparison between the liquid lens system and a 16 mm lens focused to infinity (a) and the lens system with and without the liquid lens unit (b).*

From the figure 4.4b the smoothing caused by the liquid polymer lens can be clearly observed. In the end, the resolution decrease may not be a problem, but the smoothed edges make the image appear unsharp, therefore blurring the important small details. Within the scope and objective of this thesis the resolution may be enough, but for a real application this type of a liquid lens may not be the best option.

4.3 Control software

4.3.1 Lens control

At first the Optotune focus tunable lens was tested using the *Optotune Lens Driver Controller* application available only for Windows operating system. This software provides a simple graphical user interface to control the lens in different modes [53]:

- **Current**, where user can set the coil current.
- **Focal Power**, where user can set the wanted focal power in diopters. This mode uses the built-in temperature compensation.
- **Analog**, where lens is controlled with analog voltage applied directly on one pin in the driver module.
- **Sinusoidal**/**Rectangular**/**Triangular**, where user can define the frequency and upper and lower current levels of the control signal.

The physical driver hardware is based on Atmel ATmega32U4 microcontroller and uses USB-serial to communicate [53]. An interface was implemented as a Python class using *pySerial* software serial module to send char array control messages via the Linux ACM (*Abstract Control Model*) device interface.

The driver responds to most of the commands, usually confirming the sent values or giving a status byte informing any occurred errors. Some commands produce response only in case of an error to reduce latency, for example the current and focal power setting commands. This is taken into account with the interface, where the messages related to general settings have longer timeout than those related to the focal power settings. The default timeout value for focal power setting commands is 25 ms (the maximum settling time of the lens), and for other commands expected to send responses the timeout is 100 ms. The lens might settle even faster than the maximum value, especially when only very small changes in the focus are needed [54].

In the initialization of the class the available serial ports are mapped and each one is pinged with the handshake command until the right one is found. After this the start command can be used to reset the driver any time and the current can be set straight away. If the focal power mode is used, the temperature limits have to be set before changing the mode and setting the focal power.

Some problems occurred in the tests . The first issue was a loose Hirose connector on the lens end, but this was solved by slightly bending the pins in the connector. The second issue was more critical because after a weekend when the lens was not connected, the focal power mode stopped working at all with error code indicating corrupted or faulty EEPROM. The error was tracked to the lens but it could not be solved locally and the changing procedure would have taken too much time so the direct current control was used from this point on. This was still problematic due to the lack of the temperature compensation that could be estimated but not accurately. A simple linear compensation for testing was implemented using the temperature sensor of the lens where the "cold" start and "warm" running values were measured and based on subjective minimum and maximum focus current values a temperature coefficient was calculated. Because the camera and lens both affect the temperature and it it’s not measured at the moment of power-up in this case, the calibration is based on measured temperatures.

4.3.2 Camera control

The camera acquisition software was implemented as a Python script using the *PySpin* module of the Spinnaker SDK (*Software Development Kit*). This script uses the lens control interface and works as standalone also like the rest of the implemented modules, except the camera configuration module containing functions to set e.g. acquisition and trigger modes. The number of images in stack and the start and stop lens control current values in milliamperes can be defined in this module.

The FLIR camera can acquire images in continuous, multi-frame or single-frame modes. In this application a synchronization method is needed between the capture and focusing, which was decided to be implemented with software triggering because otherwise a hardware control unit should have been implemented. According to the FLIR technical application notes the latencies with asynchronous software trigger are in the range of tens of microseconds from the camera perspective [21] and the trigger can be used in the continuous mode so this should not slow down the system too much. Also, when other compulsory delays are considered such as the focusing (25 ms) and exposure (a few milliseconds), the short control latencies generally under a couple of milliseconds are not really significant, at least in prototyping phase. This is also the reason for implementing the delays needed just by using the Python time module.

Because saving to disk is relatively slow, the camera control software first acquires images to the buffer and only after the acquisition loop retrieves and saves the images. The buffer is located in the RAM of the host system, so the overall performance and speed of the USB affect the resulting frame rate. Also the exact time of exposure depends on conditions because the camera automation is used to define the exposure time and white balance. The resulting exposure time and frame rate can be asked from the camera and used to determine the delay for the acquisition loop. The rough idea of timing the operations in the loop is represented in figure 4.5.

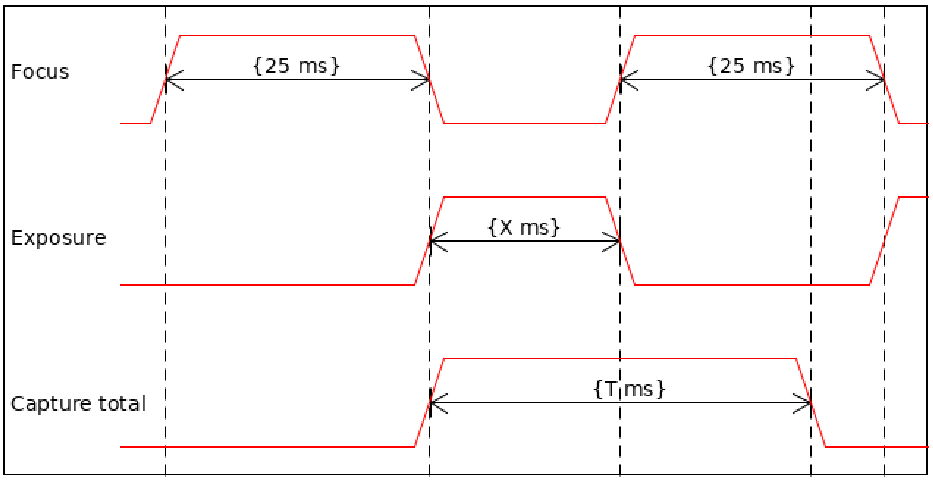


Figure 4.5 *The camera-lens synchronization.*

In the loop there are two delays; the lens settling time (25 ms) and a capture delay (T ms, exposure and saving to the buffer) depending on aspects mentioned previously. Focusing must be finished before triggering the camera, and because saving the image to the buffer does not interfere with the focusing those can be done simultaneously. Therefore the delay added to the lens settling time in normal situation is calculated from the resulting frame rate of the camera, from which the fixed focusing time is then subtracted. If the delay is shorter than the exposure time, then the delay is set to be the same as the exposure time.

The small latencies from sending commands and executing other lines are enough to prevent i.e. overlapping triggering. Also the mentioned latencies are not too long, because in test runs the actual frame rate wasn't too bad compared to the best possible free-running frame rate. The average frame rates of five test runs per configuration are represented in table 4.1. One acquisition loop run captures five images and all runs were done with the same optics in the same conditions (office environment). "Focus stacking" indicates if the lens driver was connected and the focusing delay used within the loop. Both cameras were configured to use the maximum gain (sensitivity) to get as short exposure times as possible.

Table 4.1 *Average frame rates and exposure times of five runs with different configurations.*

Camera	Focus stacking	Calculated speed (fps)	Measured speed (fps)	Stream speed (fps)	Exposure time (ms)
20 MP	No	15,6	12,7	12,9 *	3,77
	Yes	15,6	12,6	12,8 *	3,78
5 MP	No	72,6	67,6	66,2	0,14
	Yes	39,8	37,0	65,5	0,15

* Before updating to OpenCV version 3.4.5. Other frame rates did not change after the update.

The calculated speed is calculated from the resulting loop delay explained earlier and the focusing delay, so any other latencies are not taken into account. The measured frame rate is calculated from the number of images captured and the time spent to execute the loop. The stream frame rate is calculated as an average of the rates saved in free-running mode (no focus stacking involved). All measurements were implemented using the *time()* function of the Python time module, because it was accurate enough for this project.

If the only delay was the focusing time, the absolute maximum frame rate would be 1 s / 0,025 s = 40 frames per second, and with the faster 5 MP camera results come close to that. The calculated maximum speed was 39,8 fps and the actual speed 37,0 fps. At the time these measurements were executed, an OpenCV version 3.2 was used but later, after updating to version 3.4.5, the free-running frame rates with the 20 MP camera increased to an average of 16,6 fps. This didn't affect any other measured frame rate, so it can be concluded that using the software trigger slows down the acquisition to some extent.

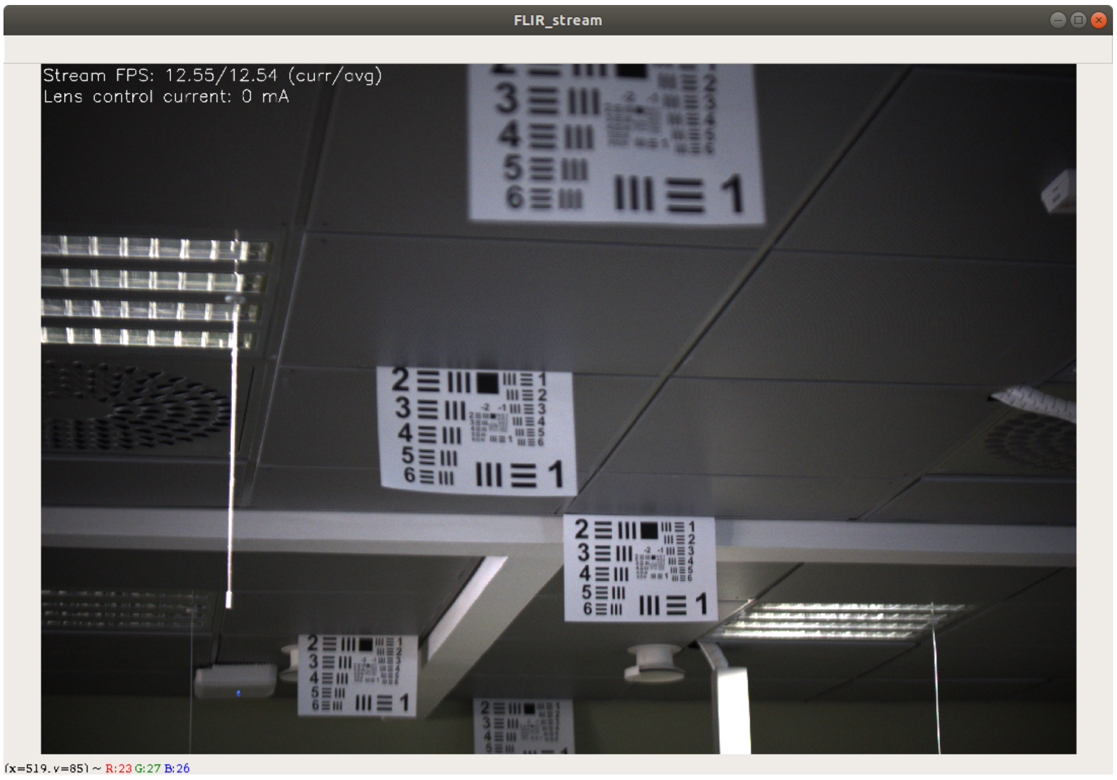
With the 5 MP camera the limiting factor is the focus control, so other latencies than the focusing and exposure times affect the resulting speed. The approximate total remaining latency consisting of sending the commands and executing those on the hardware can be calculated by subtracting the focus and exposure times from one loop-cycle duration calculated with the measured frame rate:
1 / 37,0 - 25 ms - 0,15 ms = 1,877... ms $\approx$ 1,9 ms.

Still, the biggest issue with the high resolution camera seems to be the slow buffering system. For example, when the system was run on a laptop with USB 2.0, the frame rate with the 20 MP camera was only 3 fps at best. There is also a local hardware buffer in the camera, but in the models used it is only 240 MB in size. One image reserves space of four times the maximum resolution so with the 20 MP model only two images will fit into the hardware buffer making it unavailing [22].

4.3.3 Test user interface

Even if the acquisition can be done as a stand-alone one-shot execution, for testing it is convenient to have a simple interface to stream the images continuously and execute the commands (acquire stack, adjust focus). The on-camera automation also requires continuous acquisition to adjust exposure and white balance. Spinnaker SDK has its own interface called *Spinview* which has everything possible, but the singleton type camera instance management does not allow using the camera from more than one programs at a time.

A very simple interface program was implemented with the limited UI (*User Interface*) components of OpenCV computer vision library. The program initializes the camera, checks if the lens driver is connected and starts a loop streaming the images captured in continuous mode to a window shown in figure 4.6.



```
Stream FPS: 12.55/12.54 (curr/avg)
Lens control current: 0 mA
```

```
(x=519, y=85) ~ R:23 G:27 B:26
```

Figure 4.6 *The test interface.*

In the window there are the stream frame rates and lens control current visible, but no controls due to the lack of elements in the GUI (*Graphical User Interface*) API of the OpenCV. Controls are pinned to keyboard; the stack acquisition can be started immediately by pressing key "Enter", one image from the stream can be saved with "0" and the focus can be adjusted manually with keys "1" and "2". After the program is closed with "Esc", it handles the deinitialization of the camera. There are no other functionalities as this UI was needed only to ease testing.

4.4 Post-processing software

The development of the post-processing software was started with initial research before access to the acquisition hardware was available to find out if the basic idea would be feasible at all. There are not too many existing image fusion applications intended for focus stacking and an initial overview of the possibilities and alternatives had to be obtained. Also it was important to find some baseline to compare the performance of the implemented system.

Three different ways were selected to begin with; either to use existing applications directly, modify the existing open-source applications or use software libraries only. In the first option the applications would be used as they can, which may limit the possibilities to affect the performance. Also saving the images between registration and fusion can not be avoided unless there is already an integration available. Modifying existing software would be possible if there are some APIs or source codes available and the software could be modified to work that way. Making everything with libraries allows more control over the higher abstraction level depending on the libraries used, but may also require more manual work. Because of these aspects the two latter options, modifying existing software or using libraries only, were preferred.

After the initial research there was only one considerable combo of two existing open-source applications: *Hugin*, a program for image stitching and registration [12] and *Enfuse*, a program for image fusion [41]. Hugin has an API, but the source code examination of Enfuse revealed combining these for this project being too complex to implement in reasonable time. There is also a GUI version of Hugin running Enfuse but that would not have been convenient to use within this project. Hence, the implementation was decided to be self-made with computer vision libraries, for example *Theia* on which the Enfuse is based, or OpenCV. Because there are plenty of information and examples available related to OpenCV and it supports Python 2, it was chosen as the main image processing library for this prototype. In OpenCV there are some internal GPU optimizations, but unfortunately the support for Python is weak.

In the development many different images and image sets were used to find the best solutions, but for the sake of comparability and clarity only certain images are represented in this document. The sets are gathered with the imaging equipment described earlier in a relatively dark environment to emulate bad conditions. The camera used is the 20 MP model and in the test images there are five A4-size USAF (*United States Air Force*) 1951 resolution test charts between 1,2 meters starting from 2 meters distance from the camera.

4.5 Stack registration

4.5.1 OpenCV

In OpenCV 3.4.5 there are implementations for the feature detection-description algorithms mentioned in section 3.2.2 [52]. In this project the focus was only in the non-patented (free to use) algorithms ORB and KAZE/AKAZE. For comparison, also the OpenCV's dense algorithm ECC (*Enhanced Correlation Coefficient*) [19] and hybrid solutions using the ORB and AKAZE for rough alignment and ECC for refinement were also tested.

The basic workflow for feature-based image registration in OpenCV is straightforward (example code for ORB):

1. Convert both images to grayscale; this improves the computational performance.

```
im1Gray = cv2.cvtColor(imgToWarp, cv2.COLOR_BGR2GRAY)
im2Gray = cv2.cvtColor(imgBase, cv2.COLOR_BGR2GRAY)
```

2. Detect and compute the descriptors for both images; in this case either ORB or AKAZE.

```
orb = cv2.ORB_create(ORB_MAX_FEATURES)
keypoints1, descriptors1 = orb.detectAndCompute(im1Gray, None)
keypoints2, descriptors2 = orb.detectAndCompute(im2Gray, None)
```

3. Match the descriptors of the images; in OpenCV there are several brute force solutions and also a FLANN-based option. In addition to the *match(...)*, there are also *knnMatch(...)* and *radiusMatch(...)* functions.

```
matcher = cv2.DescriptorMatcher_create(
                    cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING)
matches = matcher.match(descriptors1, descriptors2, None)
```

4. Sort and filter the matches, using for example the *Lowe's ratio test*. This stage was found to be unnecessary for the AKAZE.

```
matches.sort(key=lambda x: x.distance, reverse=False)
goodCount = int(len(matches) * 0.7)
matches = matches[:goodCount]
```

5. Extract the location of selected matches.

```
points1 = np.zeros((len(matches), 2), dtype=np.float32)
points2 = np.zeros((len(matches), 2), dtype=np.float32)
for i, match in enumerate(matches):
    points1[i, :] = keypoints1[match.queryIdx].pt
    points2[i, :] = keypoints2[match.trainIdx].pt
```

6. Use the extracted points to find the (homography) with some robust estimation technique, for example RANSAC.

```
h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)
```

The pixel-based method is even easier to use because there is a straightforward function *findTransformECC(...)* in OpenCV taking the images as input and returning the estimated homography. One of the arguments is a hint matrix which can be used to give the algorithm a rough starting point, and the execution speed depends a lot on how bad the displacement is. Without giving any starting point the ECC algorithm is very slow and inaccurate with large offsets, and may not find the transformation at all in some given count of iterations. In the tests the ECC was used with default stopping criteria and the iteration count to get results in a reasonable time.

The selected algorithms were tested in terms of accuracy, speed and robustness implying the general success with difficult (blurred, dark, small) test cases. Naturally the accuracy is important because otherwise information will be lost, but with large datasets also the processing speed must be adequate. For example, an inspection dataset of 500 image stacks would take more than four hours to finish with a 30 second cycle time and if an on-board solution was needed the fusion would have to be done in a couple of seconds.

4.5.2 Algorithm comparison

The accuracy of the alignment algorithms was measured using **mean average corner error** [14]. The used method follows the same basic principle except in this method a test image is warped with a ground truth transformation matrix and after that the estimated transformation of the original and warped image is computed. The error is calculated with the estimated homography using four corners of a rectangular window in the image, where the shifted coordinates in each corner are computed for the true and estimated homographies. The actual measured property is the average of the pixel shift errors as Euclidian distances. The idea is illustrated in figure 4.7.

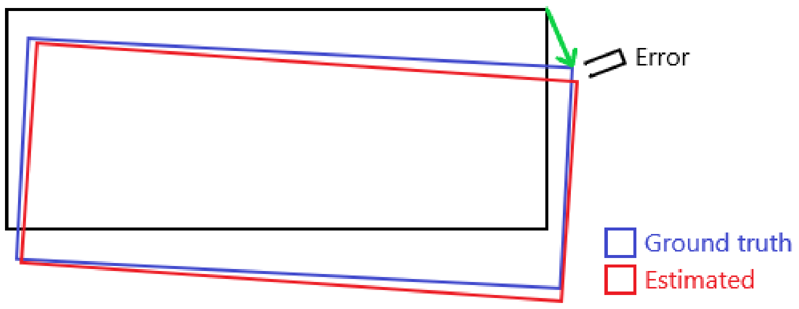


Figure 4.7 *The relative average corner error: The average error of true and estimated corners is calculated and then normalized with the true average shift amount.*

The first test was implemented as an automated loop where all the four algorithms

ORB, AKAZE, ORB-ECC and AKAZE-ECC were run on two different images. One of the two images was a sharp, high quality image with high amount of details and the other a partially blurry low quality image with much less possible keypoints. In this test the loop was run 10 times to get the average error, but as expected, there was no variation in the error between cycles. There are a lot of control options in the OpenCV implementations, for example with ORB it is possible to determine the maximum amount of features, and in AKAZE there is a response threshold value whether to accept a point [52]. Some options like the different matcher function alternatives were briefly tested, but if there were no major improvement the default functions and settings were used.

Three different transformation cases were tested, because the algorithms tend to be more invariant in some dimensions than others; small offset, medium offset with more sideways shift and large offset where the alignment is way off. The transformations are all perspective and captured from real life situations. The blended original and warped high quality test images in the three different cases are shown in figure 4.8.

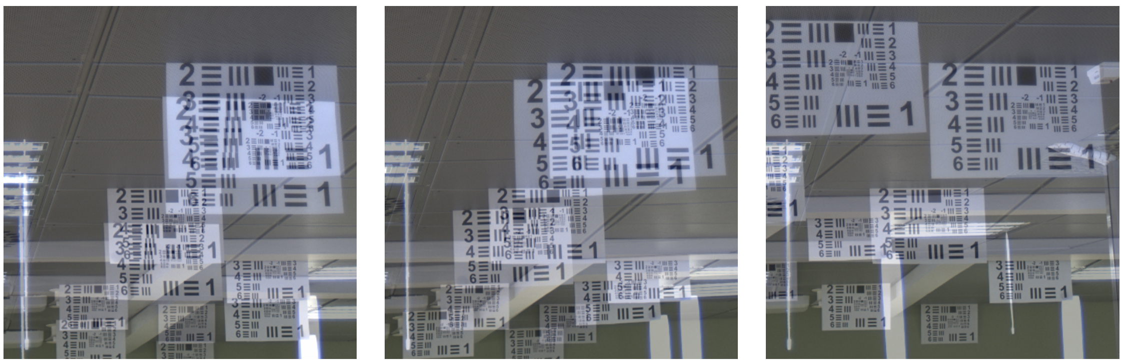


Figure 4.8 *Test cases visualized. From left to right; small offset, medium offset and large offset.*

The mean average corner errors on the high and low quality images are shown in figure 4.9. The results of using only the ECC algorithm alone were not included in the same charts because it was so slow and prone to fail that the values weren't consistent with the scale. The ECC was tested once and in all cases the error was more than 250 pixels, execution times being from 20 to 47 seconds. For the large offset case on the high quality image the ECC failed to find the transformation before its convergence.

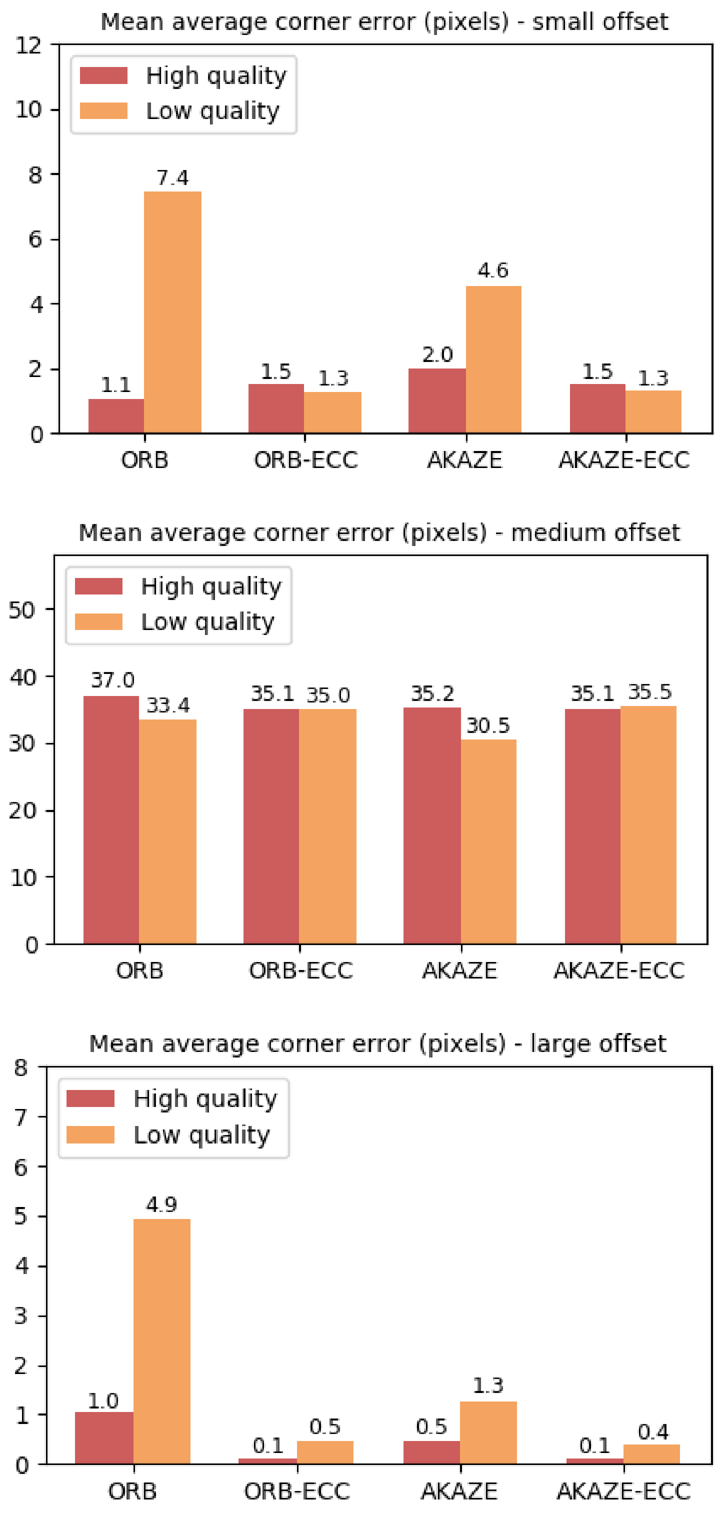


Figure 4.9 *Results of the three transformations on the high and low quality test images.*

With the low quality image the role of the ECC refinement rises. For example, in the first case where the offset is small, the difference in error is remarkable and in a fused result an error of just a few pixels may cause too much ghosting depending on the demands and the original resolution. None of the algorithms could solve the medium offset properly even if it is just slightly different to the small offset case. The best precision was achieved with the largest offset, and the most likely reason to that may be the easier separation of inliers when true and estimated points are located further from each other.

In the development phase also other cases and options were briefly tested. When comparing the ORB and AKAZE with extremely bad quality images, the ORB usually reached at least some result with large error when AKAZE either got quite close or failed totally. Filtering the outliers with ORB seemed to be important; in some cases with extremely low quality images the ORB failed completely when using a ratio of 0,5 or 1,0, but on average the ratio of 0,9 performed best.

To find out the best ORB settings for each image, a simple iterative calibration function for ORB was implemented, where the best maximum amount of points from 1000 to 9000 and the ratio of good matches from 0,5 to 0,9 were searched for every case. This calibration tended to pick up smaller values for the maximum amount of points with low quality images, but the ratio selection seemed to be quite random. In the end, this calibration did not work as expected; sometimes it improved the accuracy but more often it made the final result even worse compared to the fixed values found by trial and error.

The same error test with execution time measurement was run on a set of randomly selected images to find out the average differences. The mean average errors and average execution times are represented side by side in figure 4.10 to make the comparison easier.

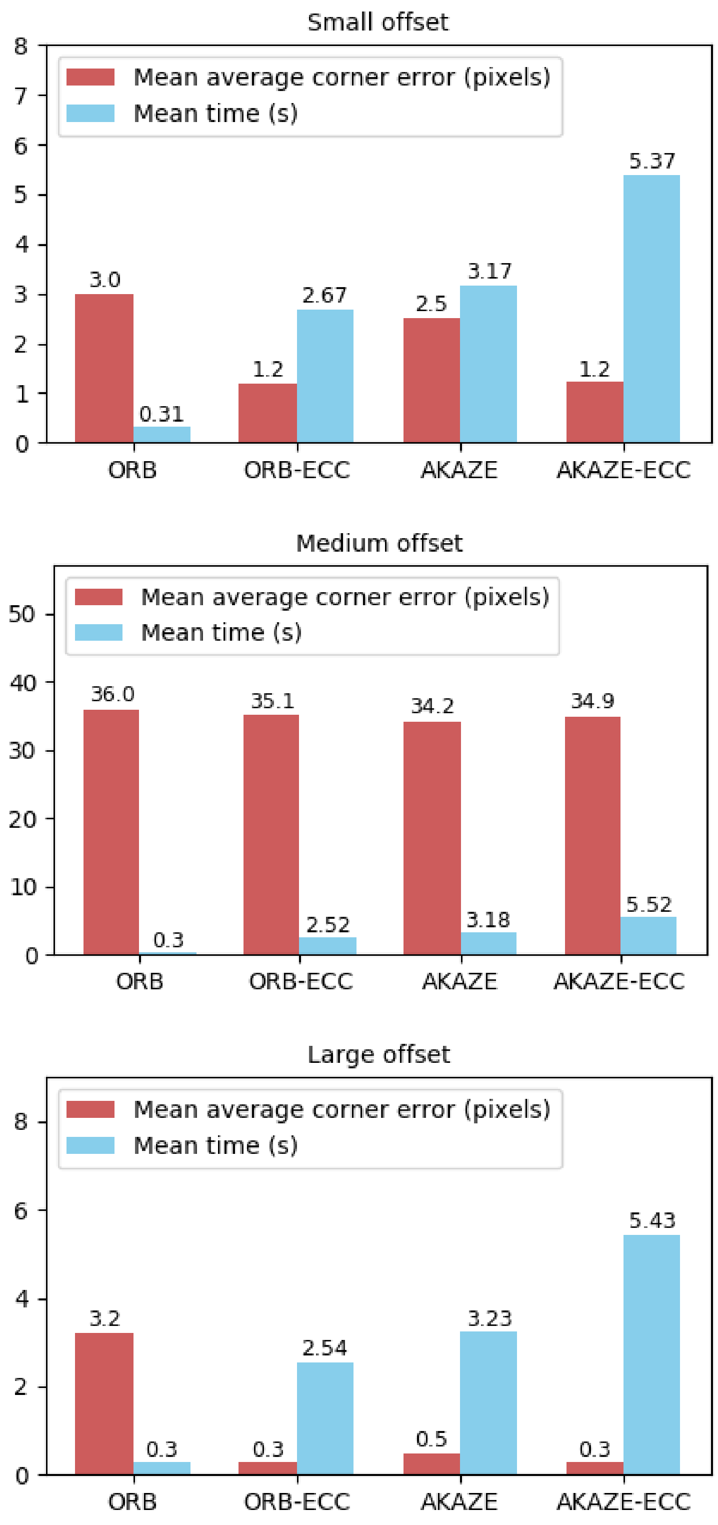


Figure 4.10 *The average errors and execution times for a set of random images.*

The AKAZE may be more accurate than the ORB, but it is also slower compared to the ORB-ECC which has the best overall accuracy vs. time performance. On the other hand the ORB alone has superior speed and adequate accuracy, so the choice between these two depends on the use case.

The Hugin software was tested as a comparison but it could not solve even the smallest offset. There are a lot of options to adjust the Hugin performance and actually the amount of control points had to be increased to get anything out, but other than that it didn't have an effect on anything else than the execution time. The times were still measured to get and idea of the speed, which was 5,6 seconds on average with the lowest possible amount of control points.

4.5.3 Accuracy improvements

Precise registration of partially matching blurry images is a challenging task and one error can ruin the final result. If only one image is used as the anchor point to which all the other images in a set are compared (the traditional way), finding the transformation will fail more likely when moving forward within the set because the overlapping sharp areas change over the images. To solve that, one obvious method was to compare adjacent images against each other, not against the one anchor image. Also a pyramid order, where two adjacent images are aligned and fused after which the fused images of the two sets are aligned and fused again and so on, was tested. The idea is illustrated in figure 4.11.

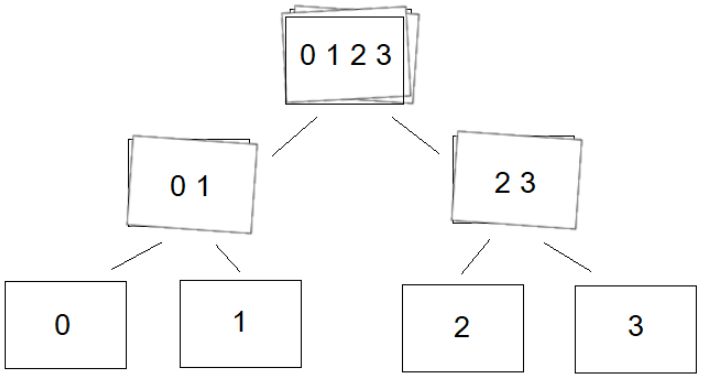


Figure 4.11 *The idea of the pyramid fusion to increase registration success rate.*

The idea was to increase the amount of overlapping sharp areas to increase the precision of the alignment. This decreased the amount of failures, but caused even the smallest errors to accumulate from the bottom upwards. This same principle was tested also by applying the fusion on every adjacent image (i.e. indexes 0 and

1, 1 and 2...) but as expected, that did not work very well because of the error accumulation. Fusing the images naturally takes time, and to minimize the number of needed fusions a different solution had to be developed.

Because the transformation is based on the homography coefficients, the homography matrices can be multiplied between each other to get the total chained transformation over several partial transformations. For example, if the transformations between the first and the second, and the second and the third images are known, the relation between the first and the third image can be calculated by multiplication of the two homography matrices. Apparently this idea (represented in figure 4.12) is used at least in medical applications, e.g. to align CT-scan (*Computed Tomography*) images [69].

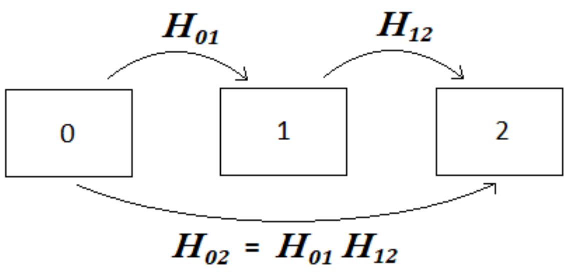


Figure 4.12 *The idea of the chained homographies to increase registration success rate.*

With this technique it is possible to obtain the homographies from the adjacent images and warp the images without excessive fusions. The chained homography method was compared to the traditional method where in this case every image from index 1 forwards is compared to the anchor at index 0. Because an image stack acquired with the system can't be used for testing as there will be the distortion caused by the lens even in perfect conditions, an artificial image stack was created by blurring one image gradually in vertical direction. The chaining proved to be very effective also in focus stacking; in addition to preventing failures with ORB, it also improved the precision with AKAZE as can be seen from the results represented in figure 4.13.

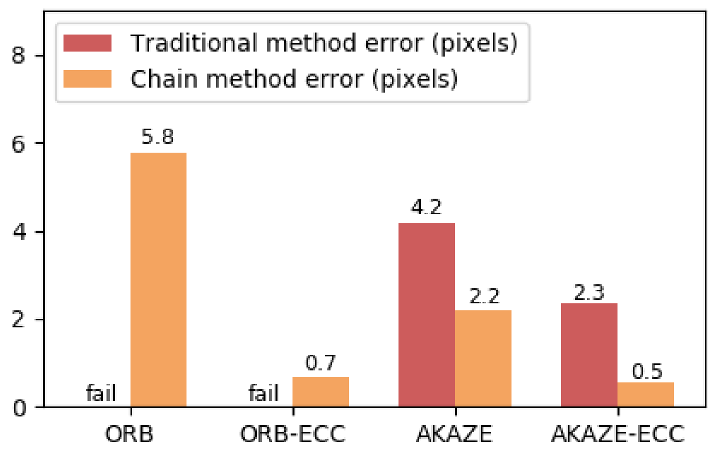


Figure 4.13 *The mean average pixel errors of the traditional and the chain method on an artificial image stack.*

Compared to Hugin this solution performed much better as it can be seen from the results. In many cases Hugin can not solve the transformation at all, and when it does it is not as accurate as the chaining method. On the other hand, with the chaining method the problem is that when one link fails, the incorrect transformation matrix propagates the error to all the following transformations. This is why it would be convenient to somehow notice when the registration between two images fails. In a set this could be used to find the best sequence e.g. the failed alignments at the beginning or at the end end of the image sequence could be skipped. Skipping a single image somewhere in the middle does not work because if the transformation could not be found for two adjacent images, it is very likely that it can not be found for the next image either.

A function to find the good sequence was implemented, where each estimated homography will be checked against a fixed threshold value. The value used to determine if the registration is not totally failed is simply the **norm of the matrix**, which implies the amount or magnitude of the transformation. The threshold value was determined by trial and error, because the offsets change on every set. One possible solution to improve the filtering would be to use statistical methods to give the threshold some weight on each image set, but the whole idea is to filter out only the clearly failed ones. The function implemented takes a list of homography matrices as an input and returns the start and stop indexes of the found sequence.

The next method to improve the success rate on low quality images was to sharpen the images before registration. The assumption is that when image contains more high frequencies, algorithms could find more keypoints. This idea was tested with a mix of different images and a very basic method to sharpen the images was used;

create a blurred version of the original image with Gaussian blur and subtract it from the original image. The normal and pre-sharpened registration were executed for every image and an average of the errors was calculated. The results for different cases are represented in figure 4.14.

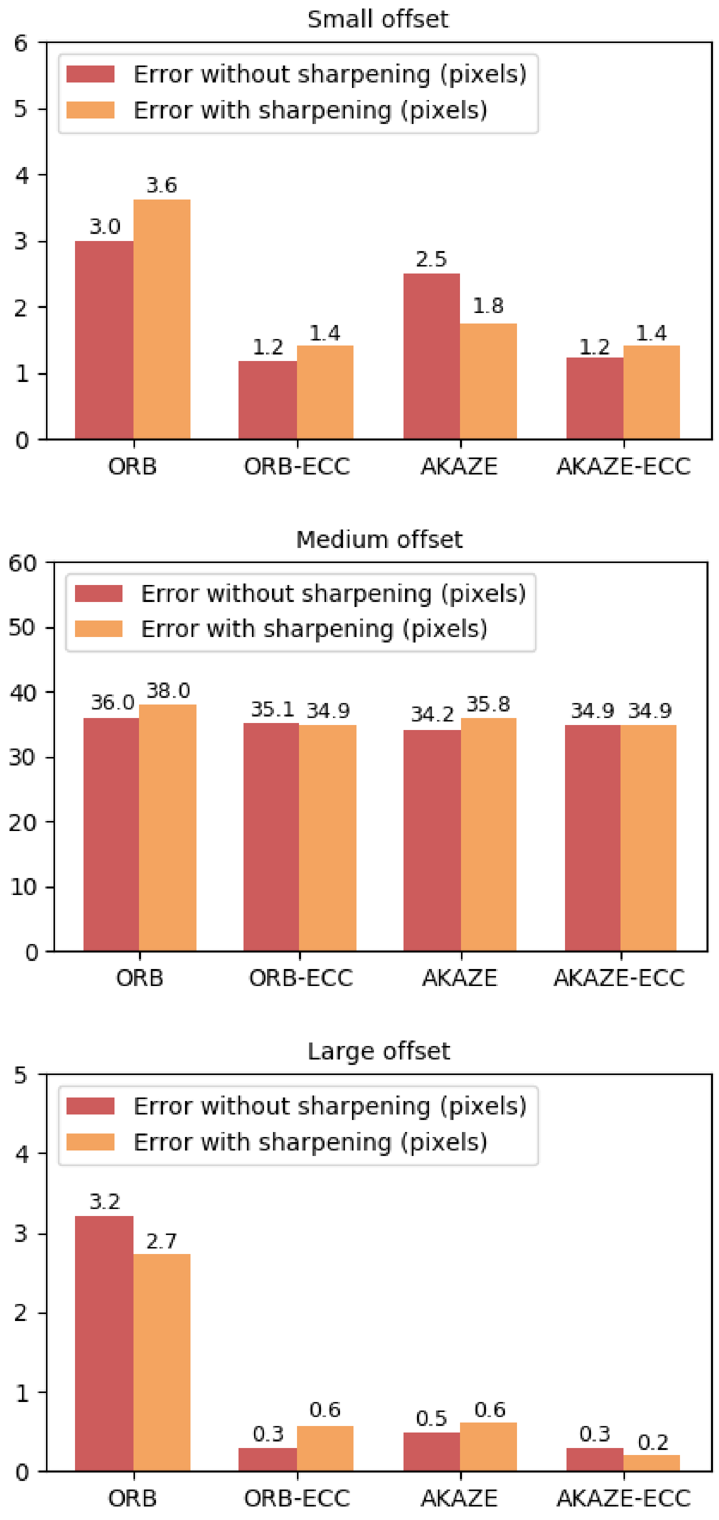


Figure 4.14 *The effect of pre-sharpening to registration accuracy on average.*

There is no distinct improvement nor decrease in the accuracy on average, but in some cases the pre-sharpening seems to even reduce the accuracy. This leads to a conclusion that sharpening should not be used before registration.

4.5.4 Speed improvements

To get an idea if the system could somehow do the post-processing on-board, some speed improvement methods were briefly tested. Some tasks in image registration could be accelerated with graphical processing units, but in this project the goal was to find out the performance on CPU. For example, there is a GPU accelerated implementation of the AKAZE algorithm achieving even 10–20 times better execution times than the original CPU implementation [58]. After basic things such as unnecessary copying and improvements achieved with small tweaks were handled, the next step was to utilize all the resources available. Because Python and OpenCV use only one core as default, it was obvious to find tasks that can be multi-threaded.

In this thesis the weight wasn't in the optimization, so only high level threading was tested to find out if multiple cores could improve the speed. This was implemented simply using the Python's *threading* module to run each entity of feature detection, extraction and matching on a separate thread. The test was conducted with image remapping (warping) and saving images to disk to be comparable with Hugin. By default Hugin uses multi-threading only for some tasks, but mainly not for the image registration [9]. However, there is a possibility to use GPU at least for remapping in Hugin and also this option was included (with the integrated Intel Graphics HD 630) for comparison. Also the OpenCV warping can be run on the integrated GPU using OpenCL [52] and dedicated Intel runtime, but that wasn't included in this test. The CPU multi-threading didn't have any noticeable effect on OpenCV remapping when briefly tested. The results are given in figure 4.15.

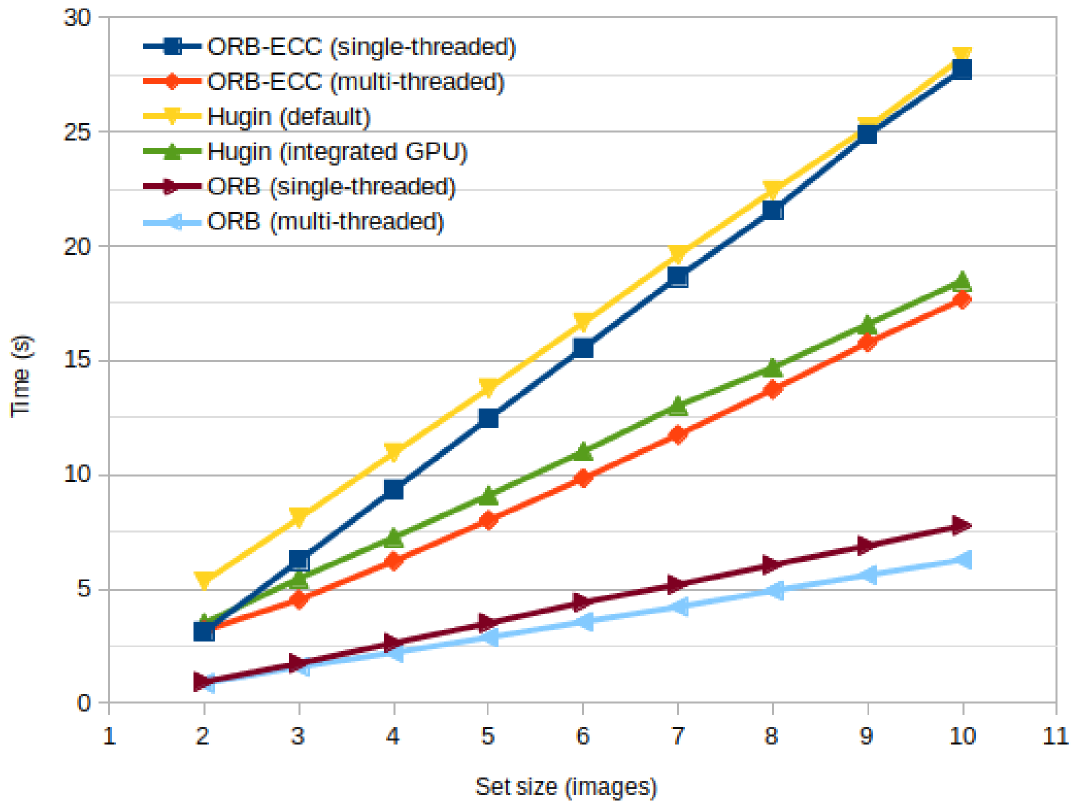


Figure 4.15 *The effect of multithreading on ORB variations and Hugin for comparison.*

The self-implemented application uses as many threads as there are images, so the speed-up should be quite linear until all physical cores are in use. With 4 cores available this should be the limit, but the application still keeps going faster with even 10 images. This is because there are many other aspects affecting the actual speed like memory performance and other internal software and hardware implementations, so the performance increase will never be directly relative to the number of cores. Naturally, in the case of just two images the threading doesn't need to be used, but in this test a separate thread for it was started to get an idea of the possible slow-down caused by thread management.

The median speed-up of the CPU multi-threading with ORB-ECC was 57 %. With two images the slow-down was about 1 %, but with three the improvement was already 38 %. Using the integrated GPU speeds up the Hugin significantly, 51 % (median) from just the case of two images, and the speed could be expected to increase with a more powerful GPU. The ORB alone is much faster than any other option here as it was seen in the previous tests, but the multi-thread median speed-up was only 23 %. AKAZE-ECC achieved a median speed-up of 60% and AKAZE 47 % (not shown in the figure). From this it can be concluded that the performance increase does not depend on just the hardware but also the solutions in the algorithm implementation.

There are also other methods to maybe increase the processing speed, for example the image pyramids where images are scaled into smaller images and the registration is started from the smallest image upwards until the needed accuracy is achieved. This could be beneficial especially with the otherwise slow ECC algorithm.

4.6 Stack fusion

In this thesis the goal was not to develop a fusion algorithm from scratch, so existing solutions were searched and compared. In addition to a couple of other functional, though unusable methods due to major artifacts, three different implementations were chosen to be tested:

- A Laplacian pyramid approach, based on the Wang's and Chang's paper [70], implemented with OpenCV by Sami Jawhar [32].
- OpenCV's ready-to-go implementation of Mertens-Kautz-Van Reeth exposure fusion algorithm [40]. The open source image fusion software Enfuse utilizes this same algorithm.
- A self-made primitive mosaic reconstruction based on Laplacian variance and coefficient of variation as the sharpness criteria.

All of the solutions above use the Laplacian filter and related calculations as the contrast criteria. The Laplacian operator calculates the second order derivative mask and is therefore usually more effective to give information about the high frequencies in an image than other methods (Prewitt, Sobel, Robinson Kirsch) calculating the first order derivatives. For example, all the methods above are based on the Laplacian operation, which can be represented as [52]

$$\Delta src = \frac{\delta^2 src}{\delta x^2} + \frac{\delta^2 src}{\delta y^2} \tag{4.1}$$

where *src* is a two-dimensional signal (image). An equivalent convolutional solution can be calculated with a kernel

$$k = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{4.2}$$

The Laplacian pyramid works by decomposing the images into different spatial resolution pyramids and fusing the layers [70]. After that the pyramid is fused and finally an inverse Laplacian will be performed to obtain the final result.

The Mertens-Kautz-Van Reeth algorithm forms a weight map based on contrast, saturation and exposedness, and uses also the Laplacian as the contrast criteria. It also uses the Laplacian pyramid to form the final image, so the final stage is quite similar to the Wang's and Chang's method [40]. Of course, image fusion requires many other steps and everything affects the result, including filter choices and sharpening, or performing the operations separately for every channel.

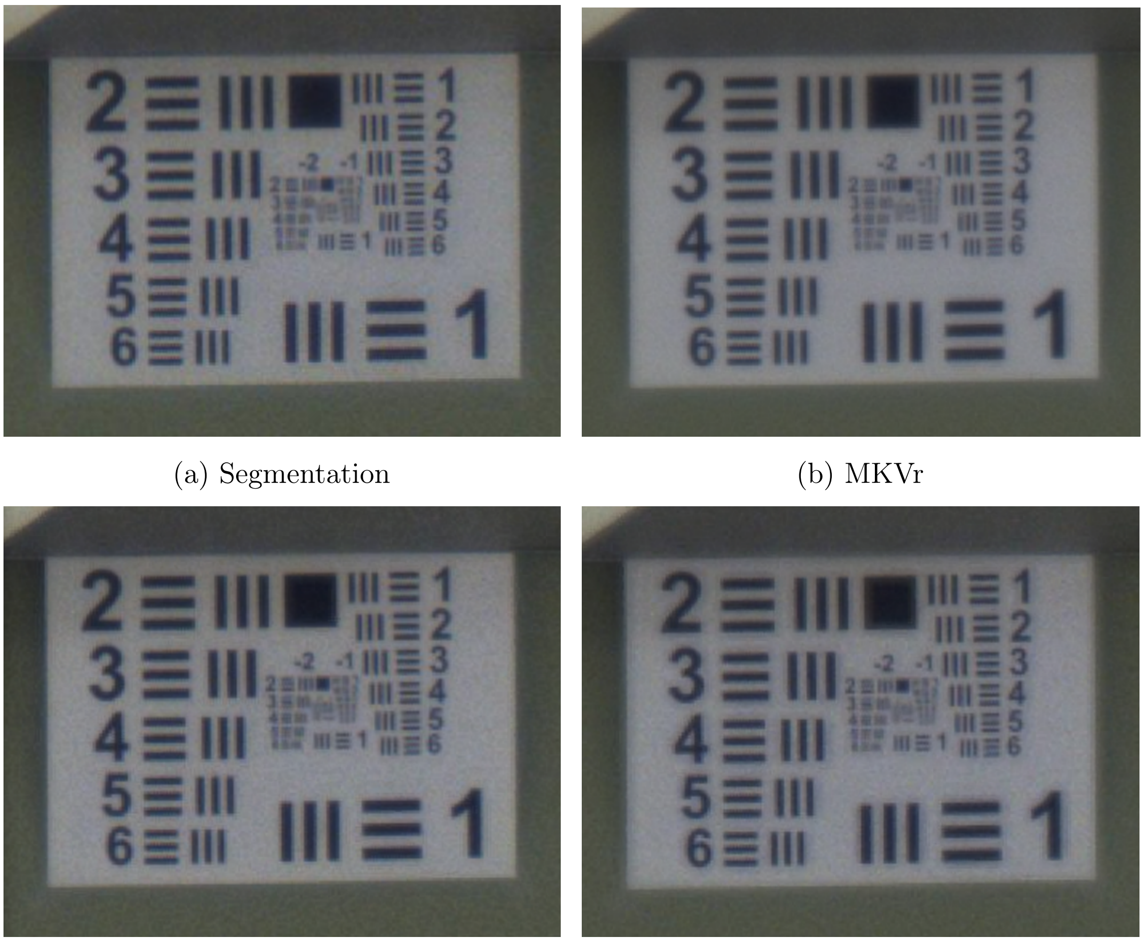
The self-made solution divides the images into segments, which are then compared and finally the sharpest ones are selected to form the final image. There are two methods to measure the contrast of a segment; the variance of Laplacian calculated from a Gaussian blurred gray-scale segment, and the coefficient of variation calculated by dividing the standard deviation by the mean of the pixel values of a segment. If the indexes suggested by these measurements are too far from each other, the segment is divided into subsegments and the maximum Laplacian variance of these subsegments will be the new criteria. This solution naturally leads to visible artifacts with difficult image sets because in this primitive version there is not any technique used to prevent seams. The windowing makes it faster, and also here utilizing the pyramid technique would solve this problem. To filter out most of the incorrectly selected segments, there is a loop checking the north, east, south and west neighbours of every segment, in that order and from left to right downwards. If there are three neighbour segments with the same index, the index under inspection will be changed to that index. It is also possible to use steps smaller than the window. From this point ahead the algorithm names will be abbreviated as "LP", "MKVr" and "segmentation".

The selected algorithms and Enfuse were compared using mainly visual evaluation, but also measurements based on relative sharpness and amount of detectable features. Four different quality measurements were used:

- Variance of the values calculated by the Laplace operator.
- Count of the keypoints detected by the FAST detector.
- An implementation [63] of referenceless IQA (*Image Quality Assessment*) model BRISQUE (*Blind/Referenceless Image Spatial QUality Evaluator*).
- PSNR (*Peak Signal to Noise Ratio*), a measurement usually used to measure compression losses.

Like with the registration algorithms, image sets of five with varying properties were used in the development and tests; dark, blurry, high quality, with and without motion to introduce ghosting. The selected set length turned out to be the best for this range (2–6,8 m), because less might not cover the whole range and more may make the result blurrier. When registration was needed, the ORB-ECC was used to create an aligned set.

The first test was conducted with a high quality image set captured in dark conditions by manually adjusting a 16 mm lens. In the first comparison no sharpening was used to find out how the fusion methods affect the resulting quality in the terms of sharpness, details, artifacts and how the natural appearance remains. The fused images cropped to a target board 6,8 meters away from the camera are shown in figure 4.16. The small numbers -2, -1 and 1 around the center of the target board are 10 mm tall.


(a) Segmentation (b) MKVr

(c) LP (d) Enfuse

Figure 4.16 *Fusion results of a high quality set.*

The resulting quality of the segmentation method is in practice the same as the sharp areas of the original images if the segments were selected ideally. The MKVr produces clearly soft, even foggy low contrast result but the details are still there. The LP is almost identical to the original image, but with slightly more noise and higher contrast. Surprisingly Enfuse suffers a grainy halo effect on the edges, but there are many options to tweak and maybe solve this problem. With Enfuse there is also similar softness present as in the MKVr result. The measured properties of this test are represented in table 4.2.

Table 4.2 *Measured properties of the first test (image set captured by a normal 16 mm lens). On BRISQUE less is better.*

	Set AVG	Segmentation	MKVr	LP	Enfuse
Lap. var.	68	70	56	112	116
FAST kp	60 397	94 375	54 190	108 109	120 949
BRISQUE	52	51	41	47	46
PSNR	-	34,1	35,2	33,4	34,2
Time (s)	-	8,44	5,72	32,33	19,88

The *Set AVG* is the average of the same measurements conducted on the input images, and because the PSNR is relative to the original images it does not have that value. The segmentation method performs worse than expected, but the image set was already high quality so it possible that some segments on the homogeneous areas get selected incorrectly. The results are not fully consistent but the Mertens-Kautz-Van Reeth based methods (MKVr and Enfuse) seem to win, even though the outputs seem smooth and blurry.

The possibility to enhance the output of the OpenCV MKVr by sharpening was experimented. The same sharpening method was used as in the registration pre-sharpening tests, because it was not too aggressive and did not produce excessive noise. Two methods were tested; sharpening the result image only after the fusion, and sharpening the image both before and after. The results are shown in figure 4.17.

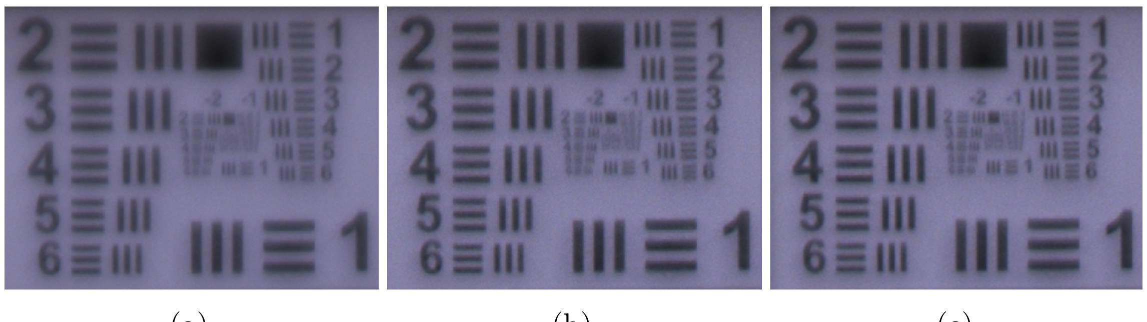
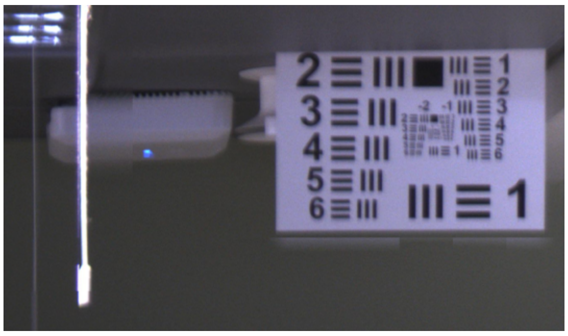
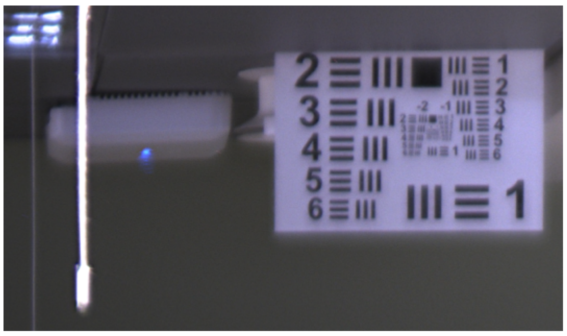

(a) (b) (c)

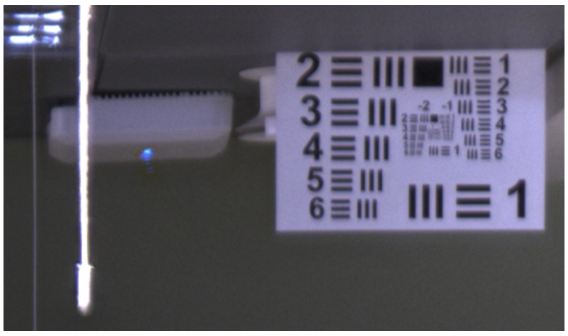
Figure 4.17 *Using sharpening with the OpenCV MKVr algorithm. (a) No sharpening, (b) sharpening after fusion and (c) sharpening before and after.*

The sharpening has a significant effect on the smoothness of the image. Obviously the fidelity isn't better, but details are easier to observe by human vision from distance. The smoothness is still there and compared to the LP method, the LP still wins in sharpness and naturalness. Even if the sharpening before and after may look the best, it was slightly worse within the measured properties than sharpening after the fusion. Therefore the after-sharpening was used with the OpenCV MKVr in the following tests.

The next test was conducted with an image set captured with the liquid lens configuration in the same environment. The results (figure 4.18) are again cropped to see the small differences better, and in the image there is a target 5,6 m away with other objects; the white lines are lamp switch cords hanging from the ceiling, and there is a router with bright power led in the background.


(a) Segmentation


(b) MKVr (sharpened)


(c) LP

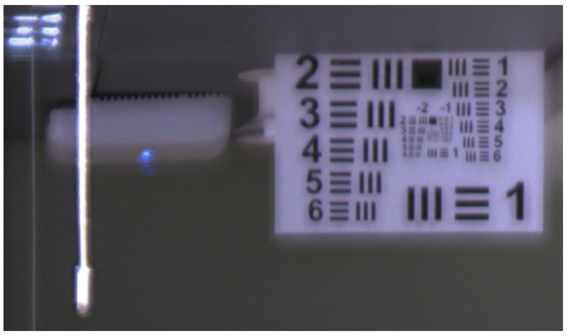

(d) Enfuse

Figure 4.18 *Fusion results of a set captured with the liquid lens system.*

The results are similar to the previous test, but now the artifacts of the segmentation method are clearly visible. The filter used by this method works surprisingly well, but sometimes "bad" segments still appear and of course the fidelity depends on the window size and step used. The softness becomes even more dominant with the methods based on the MKVr algorithm (b, d) and there is a bold halo around the target in the image fused by Enfuse (d). Again the visual winner is the Jawhar's implementation of the Wang's and Chang's Laplacian pyramid algorithm. The measured properties are represented in the table 4.3 in a similar manner than in the previous test.

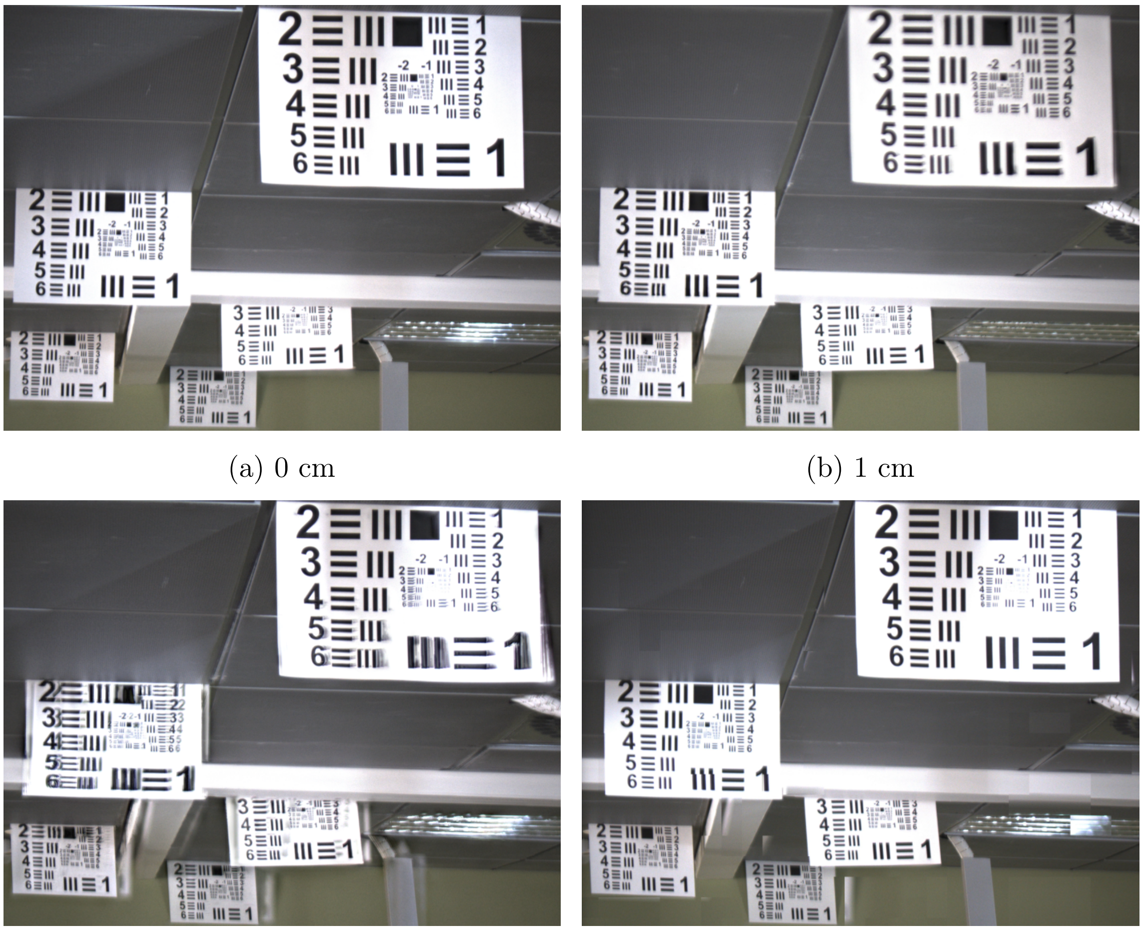
Table 4.3 *Measured properties of the second test (image set captured with the liquid lens system). On BRISQUE less is better.*

	Set AVG	Segmentation	MKVr	LP	Enfuse
Lap. var.	44	42	85	83	82
FAST kp	30 728	29 301	46 296	42 237	46 808
BRISQUE	49	45	30	42	33
PSNR	-	34,0	33,8	33,5	33,9
Time (s)	-	8,78	5,72	32,45	20,19

The measurements are again conflicting with the visual evaluation indicating the MKVr being the best despite the smoothness and halo. The other methods are not far behind except the segmentation, which can be explained with poor selection of image areas. These tests imply that sorting images by quality similarly to human perception is not an easy task. Deep learning methods could perform better with this kind of a problem due to the similarities to biological neural systems. Speed-wise the OpenCV MKVr performs the best, but also the other methods could probably be optimized close.

4.7 Impact of camera motion

The ghosting related to motion depends on how the targets are located in the depth field. If there are targets close by and far away, it is obvious that more ghosting will appear with smaller movement using regular fusion methods. To find out the approximate limits for close-range inspection, a test was conducted where the camera was moved by small steps sideways. Again the ORB-ECC was used and when different fusion methods were evaluated. The fused results of image sets captured with 0, 1 and 2 cm steps corresponding the total offsets of 0, 5 and 10 cm are represented in figure 4.19. The images a–b were fused with the Wang's and Chang's method and there is also an image fused with the segmentation algorithm (d) for comparison.



(a) 0 cm (b) 1 cm

(c) 2 cm (d) 2 cm segmentation

Figure 4.19 *The fused results of image sets captured with 0, 1 and 2 cm camera offset steps (a–c) using the Laplacian puramid method and using the segmentation method (d).*

Without any motion the result is flawless as expected, and 1 cm step only blurs the smallest details a bit. With 2 cm step the images are distorted and ghosting is apparent. If the targets were further away from the camera and closer to each other in depth direction, this would give more motion range because the relative offset between targets would appear smaller.

The segmentation method keeps the target details unchanged because it does not use individual pixels but larger blocks instead. Of course, there are strong artifacts and distortion but that could not be avoided if the camera moves. If this method was developed more, it could achieve even better results in these cases by distorting the individual blocks and detecting natural edges to follow in the images.

From this it can be concluded that if the sequential capturing is used, the drone should not move during acquisition. Based on the system speed measurements in the subsection 4.3.2, the maximum air speed sideways for a set of 5 images captured with the 20 MP camera can be about 0,05 m / 0,397 s $\approx$ 0,13 m/s. For the faster 5 MP camera the same speed limit would be 0,05 m / 0,135 s $\approx$ 0,37 m/s.

The same limits for sets of 3 images would be 0,21 m/s and 0,62 m/s. The typical speeds in structure inspections are usually under 1 m/s, so this system might be usable in motion when inspecting flat surfaces like walls but not deep structures like pylons without stopping in every waypoint.

5. CONCLUSIONS AND FUTURE WORK

The overall implementation of the prototype succeeded well. The biggest difficulty was to find a camera fast enough with high resolution sensor and a compatible lens with fast focus control. There are practically no industrial lenses with electronic focusing on the market intended for normal photography, and two options were considered; whether to build a self-made electro-mechanical solution or to use a liquid lens setup. The liquid lens was chosen even if it provided only mediocre field of view, but it was fast and compact solution. Later it was found that the polymer lens reduces the luminosity and optical resolution to an amount where it may not be usable in a real application. However, it was enough for prototyping and tests, and it also presented a challenge of using low quality images in the post-processing system. Afterwards it seemed that the electro-mechanical solution could have been better in the terms of image quality and even speed.

The development of the post-processing software revealed many unpredictable issues. The image registration is a crucial stage in the stacking system and it was noticed that only partially overlapping images are difficult to align. The open-source tool selected for comparison did not succeed with most of the test cases, but the principles of the self-made implementation proved to be very effective. It was found out that neither of the feature- or pixel-based methods were better than the other alone, but when used together the combination of ORB and ECC produced the best results. Speed is an important aspect when considering on-board usage, and the conclusion was that this system could not be implemented as an on-board solution. However this is not necessarily and issue because post-processing can be done afterwards e.g. in cloud.

Another challenge was to find an existing image fusion solution which could be used directly together with the registration system, but finally a few implementations were found. The state-of-the-art methods could achieve better results, but those should have been implemented from scratch and it would have gone out of the scope of this thesis. The best of the tested methods was visually the Jawhar's implementation of Wang's and Chang's method based on the Laplacian pyramid, and it could possibly be optimized to achieve better speeds. The OpenCV's implementation of the Mertens-Kautz-Van Reeth algorithm was also a good option, which could be used in real-life but it suffered the same fogginess as the open-source application Enfuse if there were any alignment error. It would also be possible to implement the post-processing system using neural networks or other non-traditional methods maybe improving the adaptability and speed.

Utilizing the basic principle of focus stacking does enhance the depth of field of the images, but a system using sequential capturing is beneficial only if the platform does not move during the acquisition. If the images are captured in motion, it is very likely that ghosting and artifacts will be present in the final fused result images decreasing the amount of information. This is the main issue depending on how severe and what type of the artifacts are, because stopping in every waypoint slowing down the inspection and consuming more energy is not a viable option. Solutions for that could be a faster camera and focusing system, or a multi-sensor system where all the images could be acquired simultaneously. It is also possible to develop fusion methods fusing disparate images at the expense of preserving the true geometry, or to use only the acquisition system without the fusion section. 3D sensing methods, for example a depth camera could be utilized to control the focus range and steps. However, in the end the focus stacking is almost exclusively applicable in solutions where the camera and targets don't move unless a multi-sensor system can be used.

BIBLIOGRAPHY

[1] BSHARK starts up manufacturing base, grid inspection drone. *Fuel Cells Bulletin*, 2018(11):5, Nov. 2018.

[2] Drone inspection - an efficient monitoring solution for energy sector. *Chemical Industry Digest*, Mar 2018.

[3] Agisoft. Agisoft MetaShape - Photogrammetric processing of digital images and 3D spatial data generation. Available: `https://www.agisoft.com/`, 2019 (accessed 21.2.2019).

[4] A. Al-Kaff, F. M. Moreno, L. J. S. José, F. García, D. Martín, A. de la Escalera, A. Nieva, and J. L. M. Garcéa. VBII-UAV: Vision-based infrastructure inspection-UAV. In *Recent Advances in Information Systems and Technologies*, pages 221–231. Springer International Publishing, 2017.

[5] Bentley Systems Inc. Bentley ContextCapture - reality modeling software. Available: `https://www.bentley.com/en/products/brands/contextcapture`, 2019 (accessed 21.2.2019).

[6] M. Blum, M. Büeler, C. Grätzel, and M. Aschwanden. Compact optical design solutions using focus tunable lenses. volume 8167, 2011.

[7] M. Blum, M. Büeler, C. Grätzel, and M. Aschwanden. Compact optical design solutions using focus tunable lenses. *Proceedings of SPIE - The International Society for Optical Engineering*, 8167, 2012.

[8] Canon Inc. Technology used in digital SLR cameras. Available: `http://www.canon.com/technology/now/input/dslr.html`, year unknown (accessed 8.2.2019).

[9] Canonical Group Ltd. Hugin bug report: OpenMP for align_image_stack. Available: `https://bugs.launchpad.net/hugin/+bug/1266059`, 2014 (accessed 7.5.2019).

[10] J. Chen, J. Wu, G. Chen, W. Dong, and X. Sheng. Design and development of a multi-rotor unmanned aerial vehicle system for bridge inspection. In *Intelligent Robotics and Applications*, pages 498–510. Springer International Publishing, 2016.

[11] Y.-S. Chen, Y.-P. Hung, and C.-S. Fuh. Fast block matching algorithm based on the winner-update strategy. *IEEE Transactions on Image Processing*, 10(8):1212–1222, Aug. 2001.

[12] H. community. Hugin - panorama photo stitcher. `http://hugin.sourceforge.net/`, 2019 (accessed 17.5.2019).

[13] S. Cox. Focus stacking tutorial for landscape photography. Available: `https://photographylife.com/landscapes/focus-stacking-tutorial-for-landscape-photography`, year unknown (accessed 13.2.2019).

[14] D. DeTone, T. Malisiewicz, and A. Rabinovich. Deep image homography estimation. *CoRR*, abs/1606.03798, 2016.

[15] DJI. DJI Focus - advanced, precise follow focus system. Available: `https://www.dji.com/uk/focus`, 2019 (accessed 18.3.2019).

[16] Edmund Optics Inc. Liquid lens cx series fixed focal length lenses. Available: `https://www.edmundoptics.com/f/liquid-lens-cx-series-fixed-focal-length-lenses/39466/`, 2019 (accessed 18.3.2019).

[17] Edmund Optics Inc. Liquid lenses in imaging. Available: `https://www.edmundoptics.com/resources/application-notes/imaging/liquid-lenses-in-imaging/`, 2019 (accessed 18.3.2019).

[18] Edmund Optics Inc. Depth of field and depth of focus. Available: `https://www.edmundoptics.com/resources/application-notes/imaging/depth-of-field-and-depth-of-focus/`, year unknown (accessed 11.2.2019).

[19] G. D. Evangelidis and E. Z. Psarakis. Parametric image alignment using enhanced correlation coefficient maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1858–1865, Oct 2008.

[20] FLIR Integrated Imaging Solutions, Inc. Machine vision - area scan cameras. Available: `https://www.flir.eu/iis/machine-vision/`, 2019 (accessed 18.3.2019).

[21] FLIR Integrated Imaging Solutions, Inc. Time between software asynchronous trigger and start of integration. Available: `https://www.ptgrey.com/KB/10089`, 2019 (accessed 26.3.2019).

[22] FLIR Integrated Imaging Solutions, Inc. Understanding buffer handling. Available: `https://www.ptgrey.com/tan/11174`, 2019 (accessed 27.3.2019).

[23] O. S. R. Foundation. ROS (Robot Operating System) documentation. `http://wiki.ros.org/Documentation`.

[24] C. Fraser. What is photogrammetry? *Quality Digest*, Jun. 2015 (accessed 6.2.2019).

[25] Gannon Burgett. This smartphone camera uses liquid crystal and electrical currents to focus fast and efficiently. Available: `https://www.imaging-resource.com/news/2015/10/05/this-smartphone-camera-uses-liquid-crystal-and-electrical-currents-to-focus`, 2015 (accessed 14.6.2019).

[26] Geodetic Systems, Inc. The basics of photogrammetry. Available: `https://www.geodetic.com/v-stars/what-is-photogrammetry/`, 2019 (accessed 6.2.2019).

[27] A. Greenleaf. *Photographic optics*. Photographic Optics. Macmillan, 1950.

[28] Habeeb Onawole. Huawei may launch a triple camera phone with liquid lens technology this year. Available: `https://www.gizmochina.com/2019/01/22/huawei-may-launch-a-triple-camera-phone-with-liquid-lens-technology-this-year/`, 2019 (accessed 14.6.2019).

[29] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[30] W. Huang and Z. Jing. Evaluation of focus measures in multi-focus image fusion. *Pattern Recognition Letters*, 28(4):493 – 500, 2007.

[31] J. Jang, Y. Yoo, J. Kim, and J. Paik. Sensor-based auto-focusing system using multi-scale feature extraction and phase correlation matching. *Sensors*, (15):5747–5762, Mar. 2015.

[32] S. Jawhar. Focus stacking in python using Laplacian pyramids. Available: `https://github.com/sjawhar/focus-stacking`, 2018 (accessed 8.5.2019).

[33] G. Kibby. Focus stacking in macrophotography and microphotography. *Field Mycology*, 20(2):51 – 54, 2019.

[34] S. Krig. *Interest Point Detector and Feature Descriptor Survey*, pages 217–282. Apress, Berkeley, CA, 2014.

[35] R. Li, S. Wang, Z. Long, and D. Gu. UnDeepVO: Monocular visual odometry through unsupervised deep learning. *ArXiv e-prints*, 2017.

[36] Light.co. Light L16 Camera. Available: `https://light.co/`, 2019 (accessed 11.3.2019).

[37] Y. Liu, S. Liu, and Z. Wang. Multi-focus image fusion with dense SIFT. *Information Fusion*, 23:139 – 155, 2015.

[38] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[39] S. McHugh. Lens quality: MTF, resolution & contrast. Available: `https://www.cambridgeincolour.com/tutorials/lens-quality-mtf-resolution.htm`, 2019 (accessed 5.2.2019).

[40] T. Mertens, J. Kautz, and F. V. Reeth. Exposure fusion. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pages 382–390, Oct 2007.

[41] A. Mihal and C. Spiel. Enfuse. `http://enblend.sourceforge.net/index.htm`, 2016 (accessed 17.5.2019).

[42] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[43] M. Muja and D. G. Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV)*, pages 404–410, 2012.

[44] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

[45] L. My-Ha, W. Byung-Seok, and J. Kang-Hyun. A Comparison of SIFT and Harris conner features for correspondence points matching. In *17th Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV)*, Feb. 2011.

[46] H. Mäki. *Capture, Analysis and Photogrammetry of Drone Imagery*. Tampere University of Applied Sciences, Dec. 2018.

[47] T. Nguyen, S. W. Chen, S. S. Shivakumar, C. J. Taylor, and V. Kumar. Unsupervised deep homography: A fast and robust homography estimation model. *CoRR*, abs/1709.03966, 2017.

[48] Nokia. Nokia 9 Pureview. Available: `https://www.nokia.com/phones/en_int/nokia-9-pureview/#hero`, 2019 (accessed 11.3.2019).

[49] F. E. Nowruzi, R. Laganiere, and N. Japkowicz. Homography estimation from image pairs with hierarchical convolutional networks. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 904–911, Oct. 2017.

[50] Olympus Corporation. Utilising focus stacking and focus bracketing in insect photography. Available: `https://cameras.olympus.com/stack/en/`, 2015 (accessed 24.5.2019).

[51] S. Omari, P. Gohl, M. Burri, M. Achtelik, and R. Siegwart. Visual industrial inspection using aerial robots. *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*, Feb. 2015.

[52] OpenCV team. OpenCV computer vision library. `https://opencv.org/`.

[53] Optotune. Electrical Lens Driver 4i manual, Mar. 2018.

[54] Optotune. Electrically tunable large aperture lens EL-16-40-TC datasheet, Nov. 2018.

[55] Optotune. Focus tunable lenses - and how to use them in machine vision. Available: `https://www.optotune.com/applications/machine-vision`, Apr. 2018 (accessed 18.3.2019).

[56] Panasonic. In-camera focus stacking. Available: `https://www.lumixgexperience.panasonic.co.uk/learn/expert-advice/in-camera-focus-stacking/`, 2019 (accessed 24.5.2019).

[57] Y. Piao, H. Qu, M. Zhang, and M. Cho. Three-dimensional integral imaging display system via off-axially distributed image sensing. *Optics and Lasers in Engineering*, 85:18 – 23, 2016.

[58] A. Pieropan, M. Björkman, N. Bergström, and D. Kragic. Feature descriptors for tracking by detection: a benchmark. *CoRR*, abs/1607.06178, Aug 2016.

[59] Q. Qian and B. K. Gunturk. Extending depth of field and dynamic range from differently focused and exposed images. *Multidimensional Systems and Signal Processing*, 27(2):493–509, Apr 2016.

[60] A. Raval. Inspection drones take off as flying robots replace rigworkers. *FT.com*, Sep 2015.

[61] J. Seo, L. Duque, and J. Wacker. Drone-enabled bridge inspection methodology and application. *Automation in Construction*, 94:112–126, 2018.

[62] A. Shihavuddin, X. Chen, V. Fedorov, A. Nymark Christensen, N. Andre Brogaard Riis, K. Branner, A. Bjorholm Dahl, and R. Reinhold Paulsen. Wind turbine surface damage detection by deep learning aided drone inspection analysis. *Energies*, 12(4):676, 2019.

[63] K. R. Shrimali. No reference Image Quality Assessment using BRISQUE model. Available: `https://github.com/krshrimali/No-Reference-Image-Quality-Assessment-using-BRISQUE-Model`, 2019 (accessed 24.6.2019).

[64] S. Spencer. Lens shootout: Sony RX10 III destroys the competition. Available: `https://www.dpreview.com/news/5980899251/sony-rx10-iii-lens-comparison`, 2016 (accessed 4.4.2019).

[65] A. Srikantha and D. Sidibé. Ghost detection and removal for high dynamic range images: Recent advances. *Signal Processing: Image Communication*, 27(6):650 – 662, 2012.

[66] C. Sweeney. Theia Multiview Geometry Library: Tutorial & Reference. `http://theia-sfm.org`.

[67] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–104, Jan. 2006.

[68] S. A. K. Tareen and Z. Saleem. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10, Mar. 2018.

[69] P. Thévenaz. StackReg - An ImageJ plugin for the recursive alignment of a stack of images. Available: `http://bigwww.epfl.ch/thevenaz/stackreg/`, 2011 (accessed 24.6.2019).

[70] W. Wang and F. Chang. A multi-focus image fusion method based on Laplacian pyramid. *JCP*, 6:2559–2566, 2011.

[71] WikiMedia Commons. MTF lens.svg (GNU free documentation license). Available: `https://commons.wikimedia.org/wiki/File:MTF_lens.svg`, 2019 (accessed 21.2.2019).