Development of a tool to analyze back-focus characteristics of fix-focal cameras

Björn Lind

DIVISION OF PRODUCT DEVELOPMENT | DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2022

MASTER THESIS

Development of a tool to analyze back-focus characteristics of fix-focal cameras

Björn Lind

LUND
UNIVERSITY

Development of a tool to analyze back-focus characteristics of fix-focal cameras

Copyright © 2022 Björn Lind

Published by

Department of Design Sciences
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Product Development (MMKM05)
Division: Product Development
Supervisor: Per-Erik Andersson
Co-supervisors: Emil Nordström & Martin Nyman
Examiner: Giorgos Nikoleris

Abstract

Fix-focal cameras require careful alignment of the sensor relative to the lens during assembly in order to produce a sharp image. Axis developed wide- and fish-eye network cameras are, for a number of reasons, focused at infinity when aligned.

This thesis develops the software for a new inhouse tool to be used at Axis to analyze the back-focus effect on the camera's focus characteristics. The tool is a machine called Modus, which uses a variable focus collimator to measure the sharpness of objects at distances not limited to infinity.

The main purpose of this new tool is to answer the following questions:

- How can knowledge of the back-focus effects for a given camera be harnessed to decide on back-focus offsets to achieve desired focus characteristics during alignment?
- What are the sensor placement tolerances to achieve a certain image sharpness? Can this information be used to decide between active or passive alignment?

A fish-eye camera, Koi, is analyzed with the new software written for Modus. The results show that the focus moves from infinity to 0.3 m within a back-focus window of 5 µm, with focus changes of ± 0-5%. The depth of focus for a number of image sharpness levels is also measured.

The limitation of the applicability of the results for the Koi camera lie in the sample size. In this thesis only one unit of the camera is analyzed. In order to draw conclusions about appropriate back-focus offsets a larger sample size in needed.

Keywords: Focus, Image sharpness, Back-focus, Alignment, Modus, Collimator, Depth of focus

Sammanfattning

Vid sammansättningen av kameror ställs höga krav på finjustering av sensorn och objektivets relativa position för att den resulterande bilden ska bli skarp. Axis-utvecklade nätverkskameror fokuseras, av olika anledningar, i oändligheten.

Examensarbetet utvecklar mjukvara för ett nytt verktyg, Modus, som ska användas in-house för att undersöka effekten av kamerans backfokus på bildens skärpa. Verktyget är en maskin som är utrustad med en justerbar kollimator som gör det möjligt att mäta skärpan av objekt som även är närmare än oändligheten.

Huvudsyftet med det nya verktyget är att kunna samla in data som ska besvara de följande frågorna:

- Hur kan information om effekterna av backfokus användas för att bestämma lämpliga backfokus-offsets ute i produktion för att på så vis uppnå önskvärd bildskärpa för en kamera vid olika objektavstånd?
- Vad är toleransen för placering av sensorn för att uppnå en specifik bildskärpa? Kan denna information användas för att bestämma om en produkt ska sammanfogas med aktiv eller passiv *alignment*?

En fish-eye kamera, *Koi*, analyseras med det nyutvecklade Modusverktyget. Resultaten visar att fokus skiftar från oändligheten till 0.3 m då backfokus ändras med 5 µm. Skärpan förändras ± 0-5 % för alla objektavstånd. Toleransen mäts också upp vid ett antal olika nivåer av bildskärpa.

Begränsningen av resultatets användning ligger främst i provstorleken. Endast en enhet av Koi-kameran har analyserats. För att kunna uttala sig om lämpliga backfokus-offsets behövs analyser utav fler enheter.

Nyckelord: Fokus, Bildskärpa, Backfokus, Alignment, Modus, Kollimator

Acknowledgements

This thesis was carried out on behalf of the PTS department at Axis. It could not have been possible without the support of the generous people who work here. The author would like to offer his gratitude to the following people.

To Emil Nordström and Martin Nyman, my supervisors at Axis, for their support and guidance from beginning to end.

To Johan Salomonsson and Kajsa Binder for their help and patience with regards to the electrical work carried out in this thesis.

To Björn Hansson, supervisor during my 2019 summer internship, for his continued helpfulness and for encouraging me to do my thesis at PTS.

To Viktor Billberg, who designed the mechanical parts for Modus and whose co-operation has been greatly appreciated.

To Per-Erik Andersson, my supervisor at Lund University, for his insightful comments on the contents of the thesis report.

Björn Lind
Lund, February 2020

Table of contents

1 Introduction

1.1 Background

1.1.1 Axis Communications

Axis Communications is a technology company mainly focused on the development of network camera systems and peripherals intended for surveillance. The product catalog is substantial, ranging from basic fixed-focal cameras, pan-tilt-zoom cameras , thermal imaging cameras to explosion-protected cameras. Development takes place in Lund, Sweden with more than 2000 employees.

The production of the cameras is outsourced to electronics manufacturing services-companies (EMS) around the world. While the physical production takes place here, much of the production equipment is specialized and therefore developed internally at Axis. This task falls upon the Operations-department, which oversees the process of readying a product for mass production. Production Test System (PTS) is a part of Operations, developing software and hardware for camera assembly- and testing stations that are placed at the EMS.

1.1.2 Alignment

During the assembly of the camera, the camera sensor and lens must be carefully aligned to produce a sharp image. The alignment can be divided into three different actions on the sensor's position: tilting it about the x- and y-axis, positioning it along the z-axis and centering it on the x- and y-axis interception. Figure 1.1 illustrates the coordinate system used. This figure shows the sensor tilted about the x-axis, but similarly the sensor can be tilted about the y-axis. The distance between the lens and the sensor along the z-axis is called the ***back-focus***.

Axis cameras are aligned using one of two methods: passive- or active alignment. Passive alignment is also known as mechanical alignment because the sensor and lens are attached mechanically at a predetermined location relative to one another,

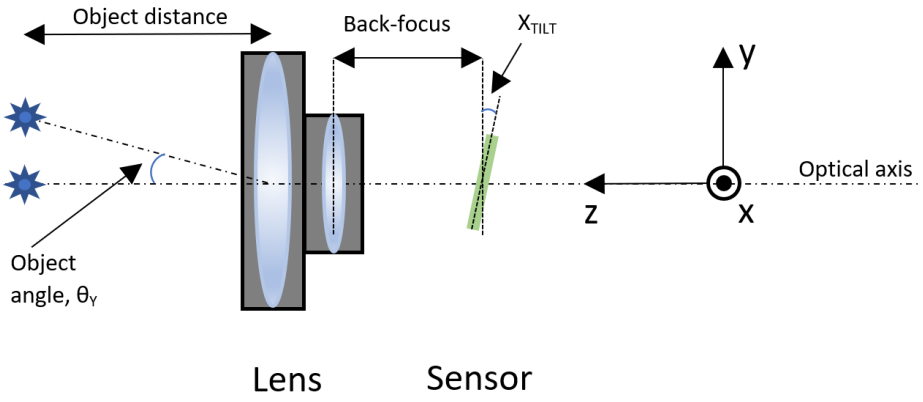


Figure 1.1: Coordinate system used when discussing alignment

without taking the resulting image into account. Active alignment on the other hand, takes the resulting image into account by analyzing the picture and making small adjustments to the sensor-lens alignment to find the optimal location for that specific unit. As a result, small differences in each sensor or lens unit can be accommodated. In short: passive alignment results in each unit produced having the same alignment, while active alignment results in each unit produced having the same focus characteristics (ideally).

For the purpose of active alignment, PTS has developed a production machine: IBAS and its successor IBAS 2, where IBAS is an acronym for Image Based Alignment System. The sensor and lens are inserted into separate fixtures which can move relative to each another. Five collimators focused at infinity each project a target cross-hair onto the lens. The collimators are placed such that the cross-hairs are imaged onto the sensor's center and towards the four corners. The back-focus and tilt of the sensor relative the lens is then adjusted such that an acceptable level of focus is achieved at all positions. The sensor and lens are then joined with a UV-curing adhesive. Figure 1.2 shows an IBAS 2 located at the office in Lund.

Each camera that is assembled in IBAS 2 is aligned such that it's focused at infinity. For a specific lens the depth of field is unique, i.e. the distance between the nearest and farthest objects with acceptable focus, see section 2.2 for more details. The farthest object is infinitely far away and the nearest object somewhere closer. Since the alignment is with collimators focused at infinity, this is where the focus will be the best, and dropping off for closer objects.

Figure 1.2: IBAS 2 machine

1.1.3 Modus

Without in-depth knowledge of the optics it can be difficult to determine the focus characteristics at object distances closer than infinity. For this purpose, a machine called Modus has been developed by PTS to analyze the focus characteristics of a camera at object distances anywhere from a few decimeters all the way to infinity. This is achieved with a variable focus collimator, described more in detail in section 2.1.1.

In addition to variable object distances, Modus can also offset the object from the optical axis to simulate picturing objects which lie towards the edge of the field of view. For example, a wide-angle camera might have a field of view of 80°, and with Modus it's possible to analyze the focus across this entire range.

Figure 1.3 shows the inside of the Modus machine. The variable focus collimator sits at the top, projecting a cross-hair down on the camera. Figure 1.4 shows how the collimator can rotate about the y-axis to simulate objects offset from the optical axis of the lens. The optical axis of the lens coincides with the z-axis.

1.1.3.1 Recent additions to Modus

Modus was previously limited to analyzing optical assemblies, i.e. sensors and lenses which have been aligned and adhesively joined in IBAS 2. The collimator could scan through a desired range of object distances and the camera took a picture at each object distance. The focus score (described more in detail in section 2.4) was calculated and it's possible to find where the camera is actually focused. There is some shrinkage of the adhesive as it cures, which can offset the alignment between the sensor and the lens and cause the unit to fail subsequent focus tests which are performed at the EMS during production.

Recently installed mechanical additions to Modus have made it possible to mount a sensor and lens in the machine prior to them being glued together. This new hardware consists of an assembly of components known as the sensor stack, pictured in figure 1.5. The sensor stack consists of two goniometers stacked on one another. A goniometer is an instrument that can rotate the attached object precisely about one axis. The rotation center of the goniometers is designed such that it coincides with the camera sensor, when it's attached to its fixture, as seen in figure 1.6. This allows the sensor to rotate about the x- and y-axis in the plane of the sensor. This picture also shows the attachment of the blue compressed air tubing, which, through the routing inside the sensor fixture, creates a low pressure that holds the sensor in place.

The entire sensor stack moves along the z-axis by a linear actuator that is situated

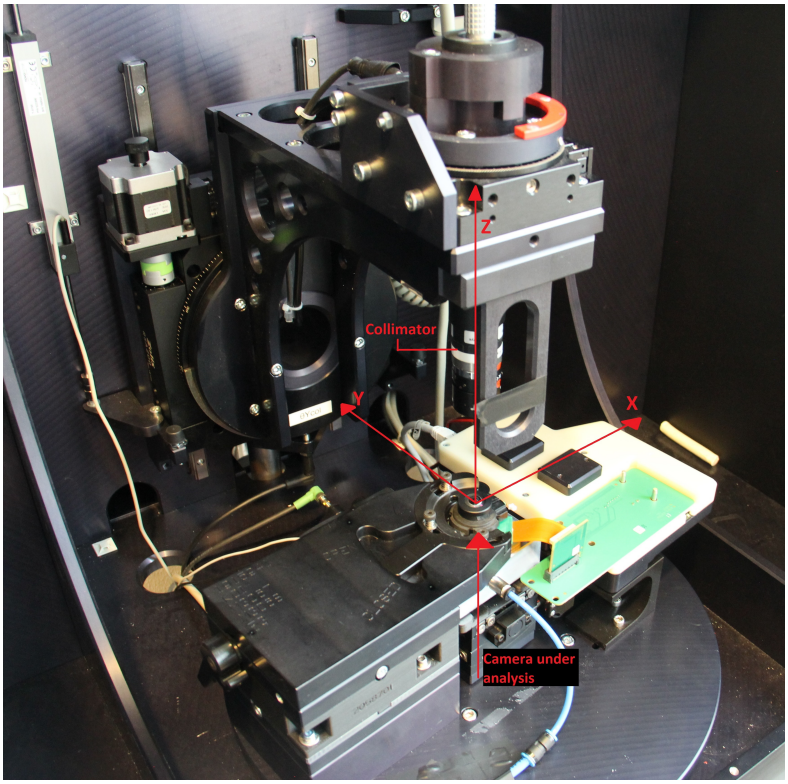


Figure 1.3: Interior of the Modus machine, showing the collimator, an installed camera and the definition of the machines coordinate system

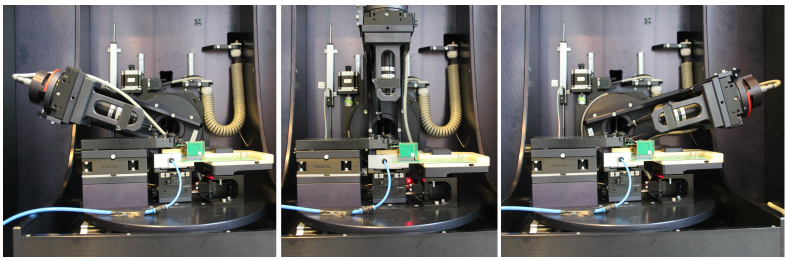


Figure 1.4: Rotation of the collimator about the y-axis, -70°, 0° and +70° respectively

below. The sensor stack is also fixed to a X-Y translation table which allows translation of the sensor along the x- and y-axis and thus the possibility of centering it on the optical axis.

Because Modus was not originally designed with the intention of including a sensor stack, geometrical constraints have imposed some limitations on the movement of

the motors to avoid physical collision. Figure 1.4 illustrated the angular limitation of the collimator rotation about the y-axis. The rotation is limited to $\pm$ 70°. Since the alignment procedure measures four corners, the collimator must rotate in both directions and thus the preceding comment means that the alignment procedure is carried out at $\pm$70°. The following list summarizes the currently existing degrees of freedom in Modus.

- Translate sensor along z-axis
- Translate sensor along x- and y-axis
- Rotate sensor + lens about z-axis
- Tilt sensor about x- and y-axis
- Rotate the collimator about y-axis
- Rotate the collimator about z-axis
- Move the cross-hair in collimator to simulate object from 15 cm to infinity

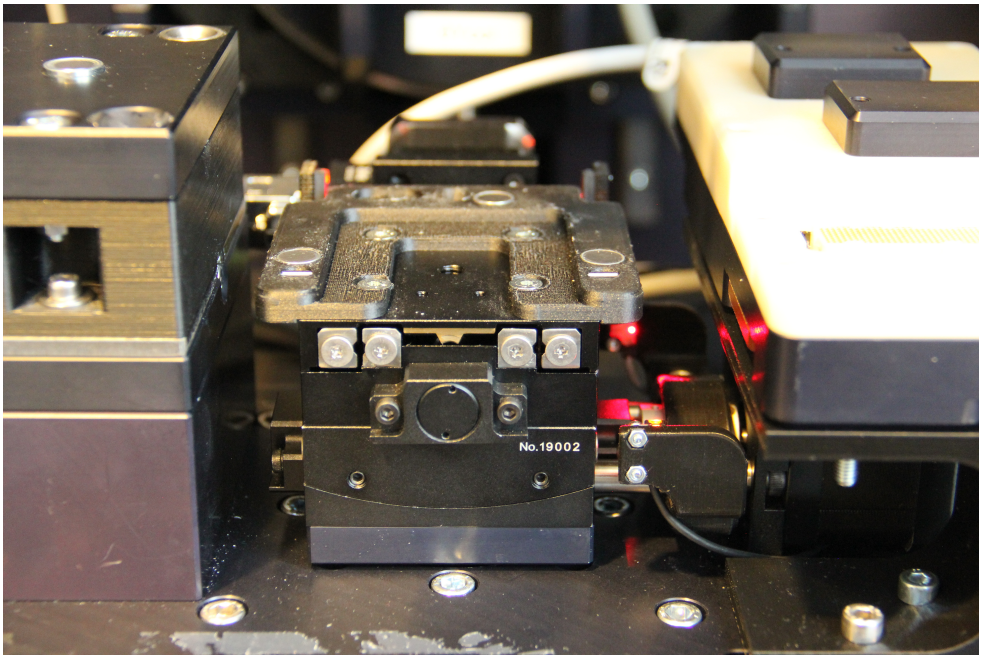


Figure 1.5: The sensor stack

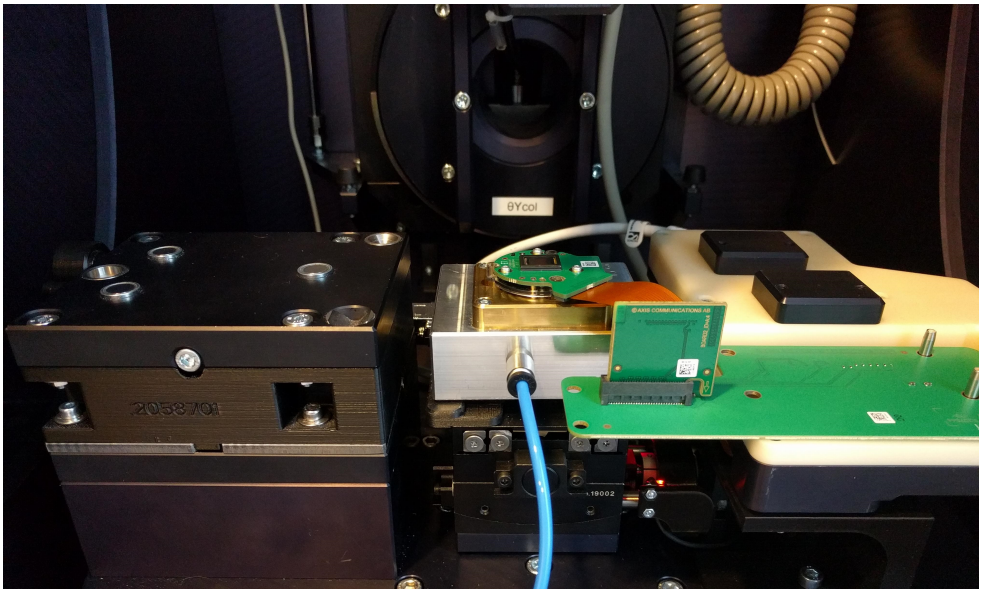


Figure 1.6: The sensor stack with the sensor fixture mounted on top

1.2 Assignment

1.2.1 Problem formulation & purpose

The objective of this thesis project is to develop a method which can experimentally find the focus characteristics of a camera prior to gluing the sensor to the lens. The focus characteristics of the camera encompasses the relationship between the focus score (image sharpness) and the parameters listed below. The object distance, angular offset and rotational symmetry parameters can already be analyzed in Modus for a glued unit.

- The back-focus
- The object distance
- The object angular offset from the optical axis, see figure 1.1 above
- The rotational symmetry of the camera, i.e. how the focus of an object offset from the optical axis varies as the lens rotates about the z-axis

The aim is that gathering data about the relationship between the focus and the above mentioned parameters should inform the decision making about whether to:

- Offset the back-focus from the aligned position found by IBAS 2 in order to

improve the focus for near-field objects, i.e. find a middle ground where the focus is above an acceptable level for the entire depth of field

- Use passive or active alignment for a camera, based on the back-focus tolerance required to achieve a certain image sharpness for objects at different distances

This will be achieved by making use of the new hardware additions to Modus that were described in section 1.1.3.1. The thesis works includes the software development for the new degrees of freedom, the gathering of focus data from one type of Axis camera, and the analysis of this data to answer the questions posed above.

The intention is that this new version of Modus will be used in-house during the development of new cameras to decide on the appropriate alignment method and settings in a production environment.

1.2.2 Limitations

This thesis is limited to the software development of Modus and the collection and analysis of optical data for a given camera. The development of new hardware has been done by a mechanical engineer at Axis and is not included in this report. The current state of the hardware in Modus was described in section 1.1.3.1.

In addition to this, the type of camera that is to be analyzed within the scope of this thesis is limited to fix-focal lenses, with neither variable focal length nor focus. Such varifocal lenses introduce two more parameters defining the focus characteristics and thus raises the complexity and the amount of data that can be collected.

1.3 Report disposition

This report begins with an introduction of some theoretical concepts of optics that are relevant for understanding the work done and results presented in this report.

The following part of the report describes the methodology followed throughout the project. Descriptions of software development principles followed and the tools used to develop the software are included. This section also introduces the camera that was chosen for analysis, and what types of focus scans were performed on it using the newly developed software.

The result section describes how the new software for Modus works, how it is or-

ganized and the user interface that was developed for interacting with the machine. Plots of the focus score relationship to back-focus, object distance and angular offset are presented for the specific camera analyzed. Parts of the complete data collection are extracted to separate plots for easier analysis.

The report concludes with a discussion on the result, how it can be used and what should be improved. A number of ideas for future development that could not be realized within the constraints of this thesis are also presented. Finally, the thesis is summarized in the conclusion, reviewing the results in terms of the goals presented in this introductory chapter.

2 Theory

2.1 Fundamental optics

Geometrical optics is an area of physics which models light as straight rays, where the direction of the rays can be altered by refraction or reflection. It is a simplification where the wavelength of light is mathematically zero and the wave-related phenomena such as diffraction and interference are non-existent.

Despite these simplifications, geometrical optics is useful in modeling how thin lenses converge or diverge light rays. Central to geometrical optics is Snell's law of refraction and the thin lens approximation. The former describes how a light ray is refracted when passing from one medium to another, and is given by equation 2.1. $n1$ and $n2$ are the refractive indices of the respective mediums, i.e. material constants, while I_1 and I_2 is the angle of the light ray relative to the normal in the respective medium.

$$n_1 sin(I_1) = n_2 sin(I_2) \tag{2.1}$$

By analyzing the refraction of a ray through a spherical surface, restricting the analysis to *paraxial* rays and neglecting the thickness of the lens, the well-known thin lens formula is derived:

$$\frac{1}{a} + \frac{1}{b} = \frac{1}{f} \tag{2.2}$$

Here, *a* and *b* is the distance from the lens to the object and the image, see figure 2.1. f is the focal length of the lens. Convention dictates that the object position *a*, is positive when left of the lens and the image position *b*, is positive when right of the lens. The material dependency, i.e. refractive index *n* enters in the definition of the focal length:

$$\frac{1}{f} = (n-1)\left[\frac{1}{R_1} - \frac{1}{R_2}\right] \tag{2.3}$$

where R_1, R_2 are the radii of the front and back surface of the lens, respectively, see figure 2.2.

The above theory and equations were collected from [1].

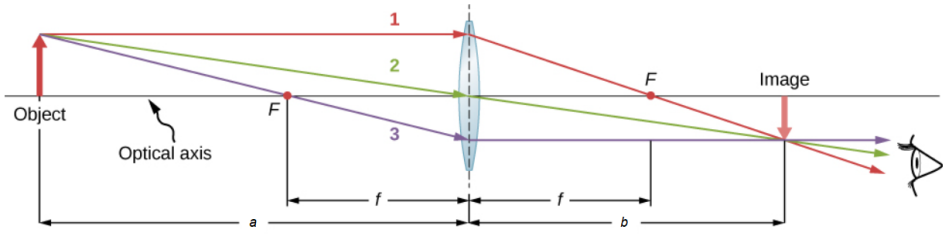


Figure 2.1: Parameters in the Thin lens formula[2]

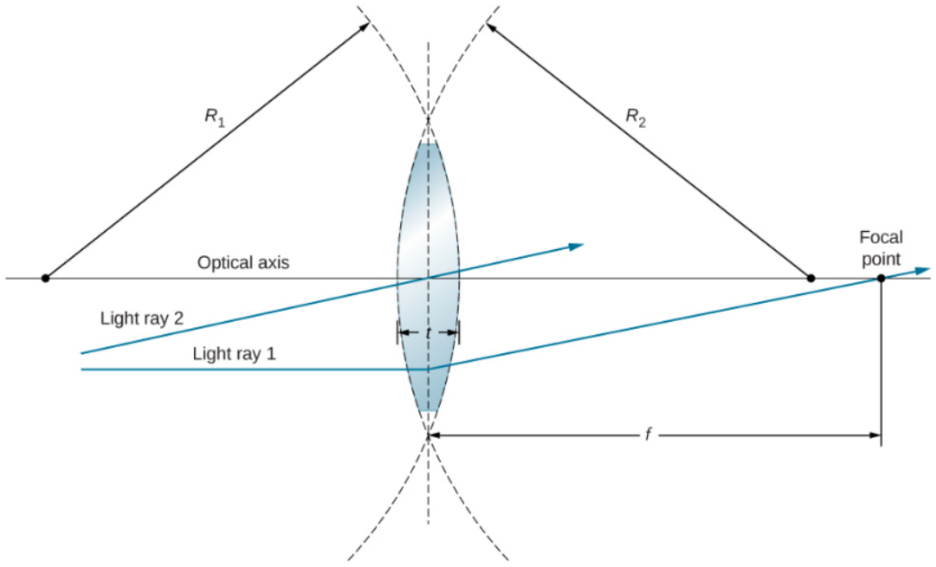


Figure 2.2: Parameters in the definition of focal length[2]

2.1.1 Collimation

Parallel rays incident on a convex (positive) lens converge at a single point at the lens' focus. The distance from the focus to the lens is the focal length, *f*, of that lens. Rays incident on a convex lens, originating from the front focus of the lens will be parallel after the lens. This collecting of light rays and emitting them in parallel rays is known as *collimation*.

Collimation can be achieved with a device called a collimator. A light source illuminates a reticle inside the collimator which in turn is projected out of the collimator by a convex lens. The collimators used in the machines developed by PTS all have reticles with a cross-hair pattern. Placing the cross hair at the lens' front focus point results in parallel rays leaving the collimator and to an observer the cross hair will appear to be infinitely far away.

A variable focus collimator is a collimator where the position of the cross hair relative to the internal lens can be changed and consequently the observed distance to the cross hair can be varied from a few decimeters to infinity. This type of variable collimator is installed in Modus. Collimators are available with different focal lengths, the one used in Modus has a focal length of 25 mm. Figure 2.5 shows a cross section of the components inside a variable focus collimator, the figure was retrieved from the installation manual of the collimator in question.

The ***defocus*** of the collimator is the offset of the cross-hair from the collimator lens' focus, such that a negative defocus implies that the cross-hair is located closer to the collimator lens than its focus point, see equation 2.4 and figures 2.3 - 2.4.[3]

$$defocus = a_c - f_c \tag{2.4}$$

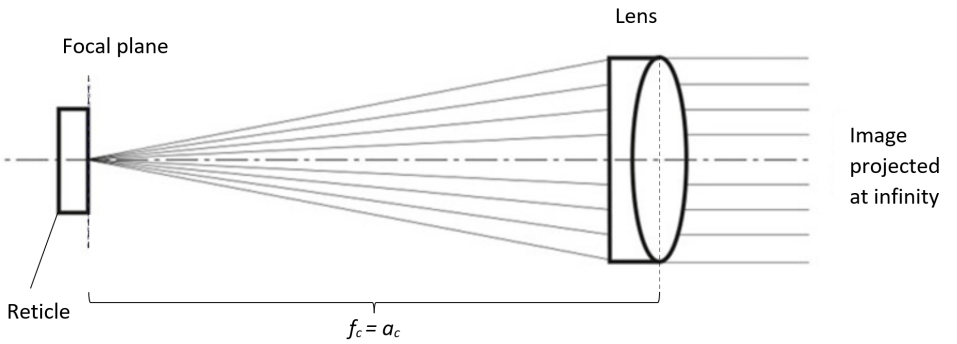


Figure 2.3: Cross section of a variable collimator

Defocus is directly connected to the object distance the cross-hair would appear to be at for a person or camera looking down the barrel of the collimator. The equation that relates the defocus to the object distance can be derived by considering equation 2.2 for a two lens system where the image b_c of the collimator lens becomes the object a_L for the camera lens, plus the distance, d between the two lenses:

$$b_c = \frac{a_c f_c}{a_c - f_c} \implies a_L = d - b_c \tag{2.5}$$

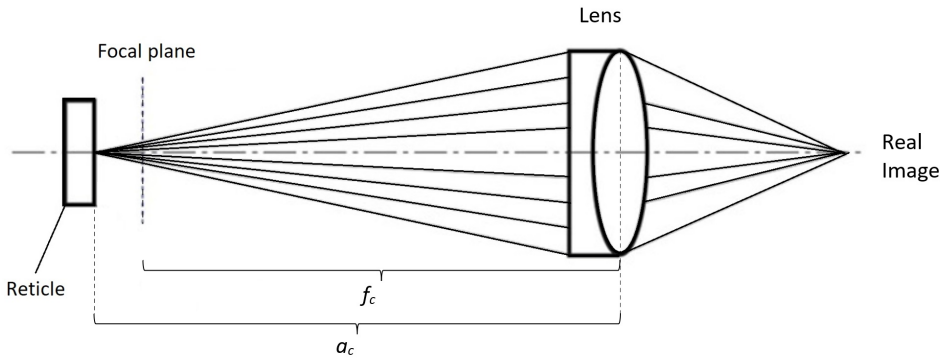


Figure 2.4: Cross section of a variable collimator

By combining equation 2.4 and 2.5, the relationship between the defocus and camera object distance, a_L is given by equation 2.6.

$$a_L = \frac{f_c(f_c - Defocus)}{Defocus} + d \tag{2.6}$$

The sign convention follows the convention used in optics.
The positive direction corresponds to the direction of light.
The step counter (positional value) should be increased in
the controller to move in the positive direction.

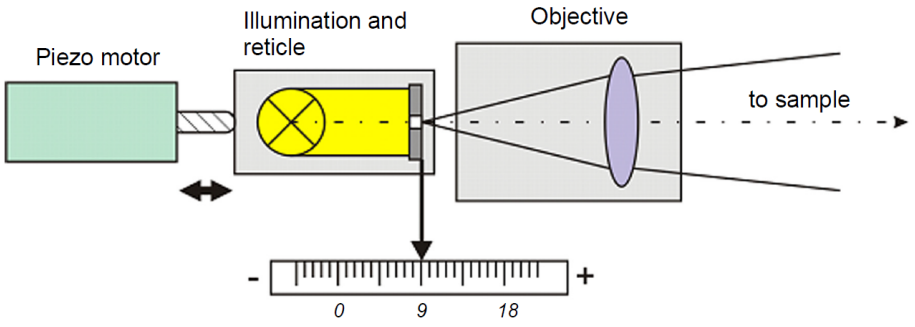


Figure 2.5: Cross section of a variable collimator

2.2 Camera systems

Cameras consists of one or more lenses that focus light onto a light sensitive surface. In digital cameras, this light sensitive surface is an electronic image sensor, with the two most common types being charged couple device (CCD) and CMOS sensor. The theory and equations presented in this subsection are retrieved from [4].

Three parameters determine the exposure of a digital image: the camera aperture (iris), shutter speed and sensor gain (ISO). The aperture is a physical opening inside the lens which restricts the amount of light that can pass through. The relative size of the aperture is given by the f-number f/N, which is defined by equation 2.7, where N is the f-number, f the focal length and D the diameter of the aperture opening. This equation shows that for a given physical aperture size (iris) the f-number decreases with increasing focal length.

$$N \equiv \frac{f}{D} \tag{2.7}$$

2.2.1 Depth of field

The depth of field (DOF) of a camera is the distance between the nearest and farthest objects that can be imaged with acceptable focus. Acceptable focus is a subjective value, often defined by the size of the *circle of confusion* (CoC). The circle of confusion is the size of an image of a point source. Lenses are imperfect in reality and therefore even with an infinitesimal point source, its image will be a circle with finite diameter. The size of the circle of confusion is different for different lenses and sensors and is determined as the largest circle which will still be perceived by the human eye as a point, when viewed from a standardized viewing distance. The size of the circle of confusion for many camera systems is on the scale of 0.01 - 0.03 mm.

DOF is mathematically defined in terms of the CoC as:

$$DOF = \frac{2u^2 Nc}{f^2} \tag{2.8}$$

where u is the distance to the object from the camera, N the f-number, c the circle of confusion and f is the focal length of the camera lens.

2.2.2 Depth of focus

The depth of focus is the image side conjugate of the depth of field. In other words, it's the distance between the farthest and nearest position of the image sensor relative to the camera lens that will image an object with acceptable focus. The depth of focus determines the tolerances of the placement of the image sensor or film.

Mathematically, depth of focus is calculated by:

$$t = 2Nc\frac{v}{f} \tag{2.9}$$

where v is the image distance, i.e. the distance between the lens and the image plane where the object in question is imaged. v can be difficult to determine, and since its scale is on the order of f, a simplified version of the equation is often used as an approximation of the depth of focus:

$$t \approx 2Nc \tag{2.10}$$

2.2.3 Fixed-focal lenses

Fixed-focal lens is a lens where the focus is fixed. The focal length and aperture cannot be changed (no zoom). These types of cameras often have short focal lengths and the focus is usually set to the *hyperfocal distance*. The hyperfocal distance is the distance beyond which all objects can be brought into acceptable focus, given a specified circle of confusion. Equation 2.11 shows the relationship between the hyperfocal distance, the focal lenth, f-number and circle of confusion.

$$H = \frac{f^2}{Nc} + f \tag{2.11}$$

2.3 Optical aberrations

Optical aberrations are imperfections in optical systems which causes light originating from a point to be focused across a larger area. Optical aberrations represent the deviation of a real lens from the assumptions of paraxial rays made in the field of geometrical optics.

There are a number of different optical aberrations. They're divided into monochromatic and chromatic aberrations, i.e. those which are present even for a single wavelength of light and those which are wave-length development.

One type of optical aberration is astigmatism. This causes objects which are offset from the optical axis ($\theta_Y > 0°$) to be focused at different points in the tangential and sagittal planes. The tangential plane is the plane that is subtended by the chief ray and the optical axis. The sagittal plane is the plane normal to the tangential plane. Figure 2.6 show the effect of astigmatism and the definition of the tangential and sagittal planes. The result of astigmatism is that for a given back-focus, as the object moves further away from the optical axis the object will become more blurry. In terms of the MTF focus score, it will decrease with increasing offset.

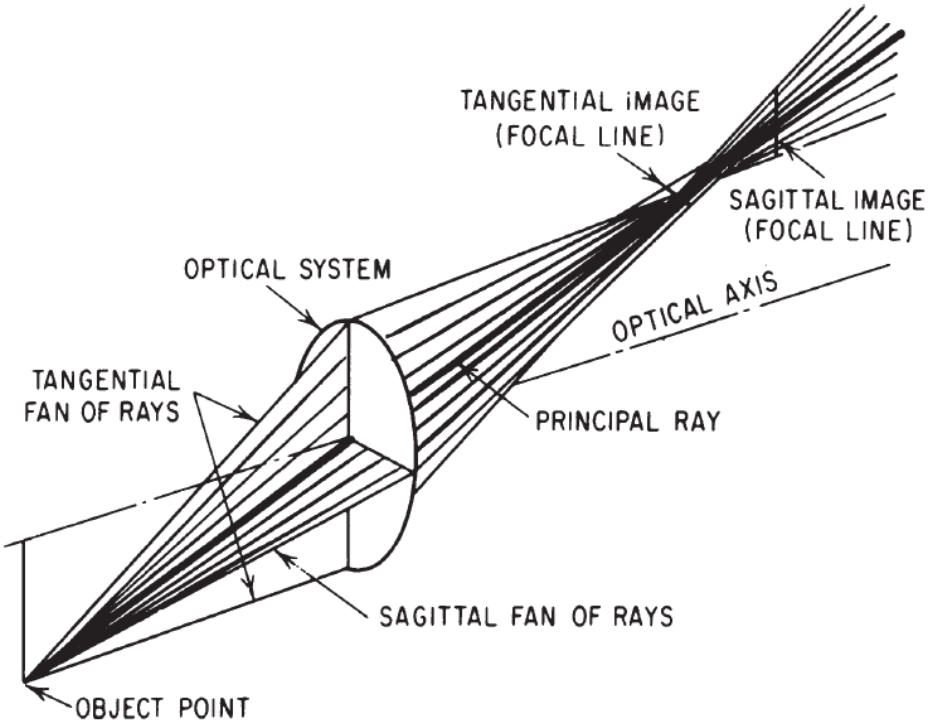


Figure 2.6: Astigmatism and the definition of the sagittal and tangential planes [4]

2.3.1 Image sensors

In modern digital cameras, Active-pixel sensors (APS) are the most common type of sensor, of which the CMOS (complementary metal-oxide-semiconductor) sensor is the most well-known. Axis cameras exclusively use CMOS sensors. The pixel pitch of a sensor is the number of mm/pixel, i.e. a measure of the size of the pixels on the sensor.

The spatial resolution of a camera is its ability to distinguish between fine details and is often reported as the number of line pairs per millimeter the camera is able to resolve. Figure 2.7 shows an example of how finer details (more line pairs per millimeter) is not resolvable for the given camera with the specific pixel pitch [5].

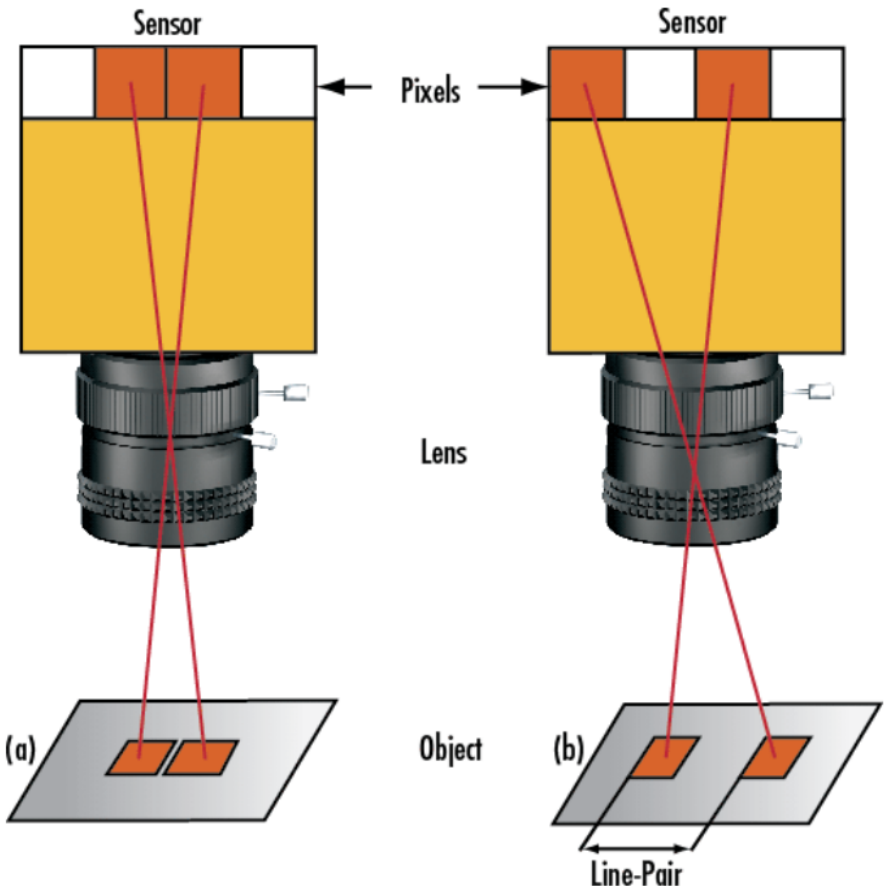


Figure 2.7: Increasing spatial frequency of the line pairs leads to unresolved details [5]

2.4 Modulation transfer function

The Modulation transfer function (MTF) is one of the most common ways of specifying the resolution of a camera. It's a measurement of how well a camera reproduces patterns of black and white lines with increasing spatial frequency, i.e. increasing lines per millimeter. In general, when a pattern of black and white lines with relatively high spatial frequency is measured, the contrast between the black and white line will be lower than for a pattern with lower spatial frequency.[6]

Figure 2.8 shows an example of an MTF-curve for a lens. The y-axis depicts the image contrast of the line pairs. As the previous paragraph explained, the contrast decreases with decreasing spatial frequency. The figure also shows how the MTF is dependent on the aperture size of the lens. With decreasing aperture the MTF is better, as seen by the difference between the MTF-curve for the lens at f-number 5.6 compared to 2.0. Figure 2.8 also shows the theoretical maximum MTF at different apertures, known as the diffraction-limit since the effects of diffraction limit the resolution from being higher than this.

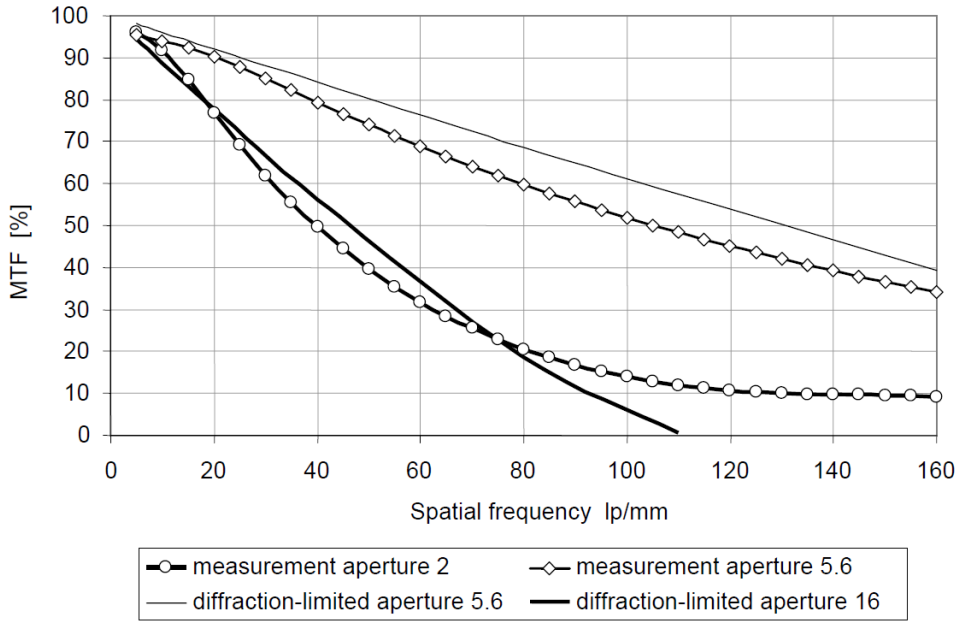


Figure 2.8: Example of an MTF-curve for a lens [6]

The above figure presented the MTF for a lens at different apertures. However, there is an MTF associated with the image sensor/photographic film as well. MTF has the advantageous property of being multiplicative, i.e. the MTF of the combined

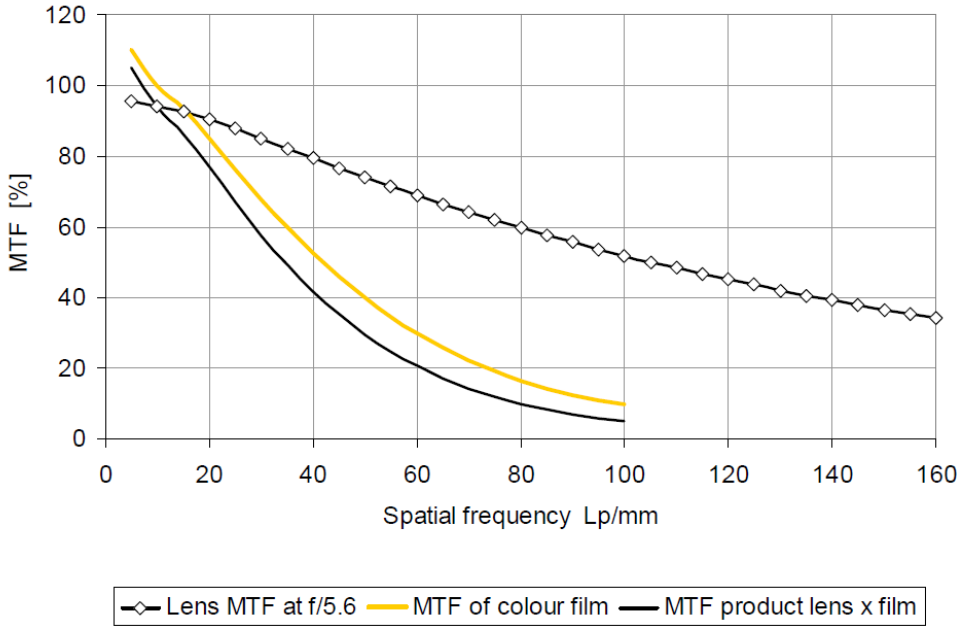


Figure 2.9: Product of the MTF of a lens and color negative film [6]

lens/sensor is the multiplication of the two individual MTFs. Figure 2.9 below shows how the combined MTF of a lens at f/5.6 and color film relate to the individual MTFs.

When comparing the resolution of two cameras, or the MTF of a camera where certain parameters are being varied such as back-focus, object angle or object distance, it's desirable to have a single numeric value to compare with. While the MTF-curves could be compared, this is not efficient in practise. Common methods is to compare the MTF-50% values, i.e. the spatial frequency where the MTF is 50%. Another method is to integrate the MTF-curve from 0 lp/mm to the Nyquist frequency. This second method is how the *MTF focus score* is computed at Axis and therefore all references throughout to the *focus score* refer to the integral of the MTF-curve from 0 lp/mm to the Nyquist frequency.

The reticle in the collimator in Modus is a crosshair, which means that a camera looking into the collimator will see a crosshair that appears to be at some defined distance. Existing algorithms implemented by image engineers at Axis are used to calculate the MTF of the camera based on this image of the crosshair.

2.4.1 Nyquist frequency

The Nyquist frequency of a camera is a measurement of the absolute limiting resolution of its image sensor. In general, the Nyquist frequency is defined as half of the sampling rate of a discrete signal processing system. Since imaging by a digital image sensor is essentially a discrete signal processing system, the Nyquist frequency is half of the number of pixels per unit distance, i.e. the inverse of the pixel pitch. As an example a digital image sensor of width 9 mm, 1400 pixels wide has a sampling frequency of 1400 / 9 = 155 pixels/mm. The Nyquist frequency is then 155/2 = 77.5. This means that the image sensor can resolve at most 77.5 line pairs per millimeter.

3 Methodology

3.1 Overview of methodology

As described in the background chapter, the mechanical parts for the new degrees of freedom in Modus where designed by a mechanical engineer at Axis. The work involved in this thesis is listed below, and further explained in following sections.

- Restructuring the old software source code of Modus, see section 3.3.2
- Developing software to measure the image distortion, section 3.3.3
- Developing software to control the motors for the new hardware, section 3.3.4
- Developing software to run focus scan programs, section 3.3.5
- Developing software to align a lens and sensor in Modus, section 3.3.6
- Developing a graphical user interface, section 3.3.7
- Designing and building a new electrical panel, section 3.4
- Gathering focus data for a particular Axis camera, section 3.5

Hardware should be interpreted as motorized equipment such as rotation tables, goniometers, collimators. It also includes electronic sensors of varying types (inductive, photoelectric, etc.). The stepper-motors used in Modus are all controlled through motor drivers of the brand *Moons*, cf. [7].

Similarly, the phrase *software development* is used frequently throughout this report and should be interpreted as the iterative process of:

1. Planning how to write source code to implement a new feature
2. Writing the source code in the C#- or Python programming language
3. Testing the functionality of the new source code
    - A new feature without hardware interaction is tested directly

- A new feature requiring interaction with hardware is tested along with the whole Modus software when connected to the machine

4. Submitting the new source code for review by Axis supervisors

3.2 Software development methodology

Before describing how the new software for Modus was developed, the development environment and tools used throughout the thesis are described in this section. A few design principles that were followed are also briefly explained.

3.2.1 Development environment

The software development environment adopted for this thesis was predetermined by standards set up by PTS at Axis. It's Windows-based and centered around the .Net-framework, with C# as the primary programming language. Microsoft Visual Studio was used as the integrated development environment (IDE), where the code was written, tested, debugged and built for release. Existing code libraries and APIs were imported using NuGet-packages, which is also the standard way of how software developed at PTS is distributed internally.

3.2.1.1 Version control

Git is an open-source distributed version control system that is used to track and manage changes in software source code over time [8]. It was used in conjunction with Gerrit, which is a web-based collaboration tool for software development teams [9]. As changes were made to the existing Modus software and new features were implemented, supervisors and other team members at Axis could review and suggest changes. This iterative process went on until everyone was satisfied with the result, at which point the feature/update was merged into the existing code-base.

3.2.1.2 Additional tools

In addition to C#, Python was used to write scripts for data visualization. Two python libraries were helpful in completing this: *Matplotlib* and *Numpy*. The former is a library with plotting functions resembling Matlab's plotting. The latter is a library for numerical computations, with support for multi-dimensional arrays and matrices and a large number of mathematical functions.

3.2.2 Design principles

This section describes a few of the software design principles that were followed when writing software for Modus.

3.2.2.1 Object-oriented programming

The most general principle adopted, perhaps more of a design philosophy, was object-oriented programming. It rests on four design pillars: encapsulation, data abstraction, polymorphism and inheritance. The following paragraphs describe the design pillars, as defined by [10].

Encapsulation describes how data should be implemented in classes which are used externally. Modification and access to the data within the class is only possible through methods which are made public, i.e. the methods define rules of how the data may be interacted with.

Abstraction is the concept of declaring an interface to some functionality without actually defining how the functionality is implemented. In other words, using the functionality of another class should only not require knowledge of the inner workings of the class, merely the workings of the interface.

Inheritance defines relationships between objects and mechanics for grouping objects into logical families. The ”is a” relationship is central, e.g. a Car-class can inherit from a Vehicle-superclass since a car *is a* vehicle.

Polymorphism consists of static and dynamic polymorphism. The former refers to method overloading, meaning that a method in a class could have several different implementations depending on the parameters supplied to it. Dynamic polymorphism refers to method overriding: having methods in inherited classes overriding methods in the parent class with the same signature. A very basic example would be a method in a Vehicle-super class that returns the number of wheels of the vehicle. The implementation in a subclass *car* and another subclass *motorcycle* would differ.

3.2.2.2 DRY - Don’t repeat yourself

Don’t repeat yourself is a principle of software development which aims to limit code repetition. Repeated patterns of code should be identified and properly abstracted. In practice this means that a block of code that is repeated should be placed in an appropriately named method and referenced whenever needed. The main advantage is code maintainability. Subsequent changes to the block of code will only need to change in one place, rather than several. This minimizes the risk of bugs

and unforeseen errors. DRY also dictates that methods should be short and effective, serving a single purpose.[11]

3.2.2.3 Design for testing

Design for testing means that the functionality of the software can be tested during development. This is achieved by adhering to the last point made in the previous section - methods should serve a single purpose. This enables the developer to write tests that only test a specific, limited functionality of the program without influences from other parts. Therefore methods should ideally also be as independent as possible, relying only on its input parameters and private methods and variables defined in the same class.

C# offers the possibility of writing Unit Tests in which the developer writes a small program to test a specific public function. Conditions necessary for running the test are initially set up, but should be kept to a minimum. The method is then called with the input parameters and its returned result is validated against what's expected from it. The output will depend on the input parameters and it is therefore considered good practise to write unit tests which test a number of different combinations of parameters. Common, extreme and invalid values of the input parameters should be tested to see that the method behaves as expected.[12]

When writing software that controls hardware, incorporating testing can be somewhat more challenging. Ideally, a simulated version of the hardware is desirable, i.e. a simulated machine. Calls to hardware, such as a request for a motor to move to a specific position could then be replaced by an appropriate time delay. With a simulated machine it would be possible to test the software as a whole without having access to the hardware. This is desirable since the hardware might not yet be completed when the software is being written.

3.3 Software development details

3.3.1 Existing software

The software written for Modus prior to this thesis was written for the purpose of analyzing optical assemblies, i.e. sensors and lenses that have been aligned and glued together in IBAS 2. The main capability is to scan the collimator through a range of object distances and measure the focus at each object distance. This can be done at different object angles, θ_Y, and rotation angles, θ_Z

Figure 3.1 is an overview of the starting point of this thesis with regards to the Modus software. Each element in the figure is a software module, in practice a C#-project. Each module consists of one or more classes. The *ModusHardware* and *ModusMachine* modules are responsible for setting up and controlling hardware. The *ModusMain* module contains classes for calibration, calculating the MTF and estimating the target's location on the sensor. The *CameraInterface* establishes a connection between the Modus software and the camera sensor, allowing images to be captured at request.

The *SimpleFocusScan*-module is, as the name suggests, a program to perform a simple scan of the collimator and measure the focus score. This module builds an executable file that the user launches to initiate a scan. With proper abstraction this module should not be concerned with how the machine should be set up or operate. It should simply provide the settings for the current camera connected, the desired object distances and object angles to measure the focus at and perhaps process the result in some manner.

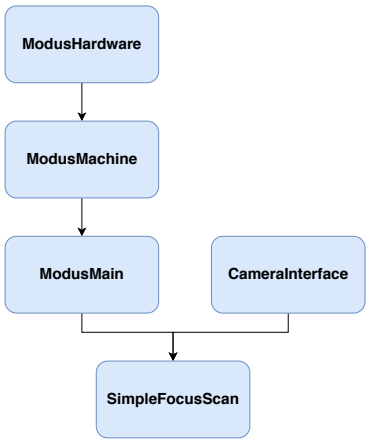


Figure 3.1: The existing Modus software

3.3.2 Restructuring existing software

A number of organizational changes were made to the existing Modus source code before beginning to implement new features. Much of the source code in the SimpleFocusScan-module was moved to the ModusMain-module, because it dealt with importing and processing machine settings and setting up connections. With these functions now abstracted to the ModusMain-module, when new features are developed for Modus and a *ComplexFocusScan*-module for example, is created, it won't have to repeat the set-up that was previously in the SimpleFocusScan-module. Both modules will simply access these functions through the interface provided by the ModusMain-module.

Additionally, the ModusMain-module consisted of a class called *Modus* which had many responsibilites. There is a software design principle known as the *Single Responsibility Principle*, which states that, in object-oriented programming, classes should have only *one reason to change* [13]. As is now, the Modus-class would have to change if there were changes to how the settings for the motors, camera and the scan program. Therefore, to better divide responsibility, everything related to importing and running focus scans was encapsulated in a new class called ScanProgramRunner. The responsibility of importing camera settings and connecting to the camera was abstracted into a ModusProduct-class.

3.3.3 Distortion polynomial

To reduce the amount of computation needed in the MTF analysis, the full-size input image is cropped to an area around the cross. The image of the target is projected to different parts of the sensor depending on its position relative to the optical axis. In order to crop the correct area on the image a relationship between the angular offset of the target and position on the sensor is required. This relationship is provided as a polynomial function, referred to here as the *distortion polynomial*. This distortion polynomial is not always readily available for a given lens, and therefore a method to measure it in Modus was developed.

Initially, a calibration of the sensor's angular offset about the z-axis is performed such that its long side is parallel with the machine's x-axis. Rotation of the collimator about the y-axis should then only result in an offset of the target along the x-axis in the image. By sampling the target location in the image for different collimator angles (object-optical axis angular offset), the relationship can be determined. A polynomial is then fitted to the sampled data, and the coefficients of the polynomial is saved for future reference.

3.3.4 Motor control

All motors in Modus are bipolar stepper motors controlled by one Moons step motor driver each. The driver works like an interface between the Modus software and the hardware of the motor. Commands to move, stop or home the motor are sent from the software to the motor driver. The driver has control circuitry to decode these commands and is then responsible for timing and supplying current to the phases of the bipolar stepper motor to fulfill the command. The exact motor driver used in Modus is the Moons MSST5-Q-RE, which communicates using the Modbus protocol over RS-485, see [7] for details about the motor driver.

With five new degrees of freedom in Modus, five additional motor drivers were installed. Each motor driver was configured through Moons' proprietary configuration software *ST Configurator*. Figure 3.2 shows the interface of the ST Configurator tool through which the motor driver is configured with settings related to the motor it's driving and the communication network. This includes the motor phase current, encoder settings, input/output signals, communication ID and more.

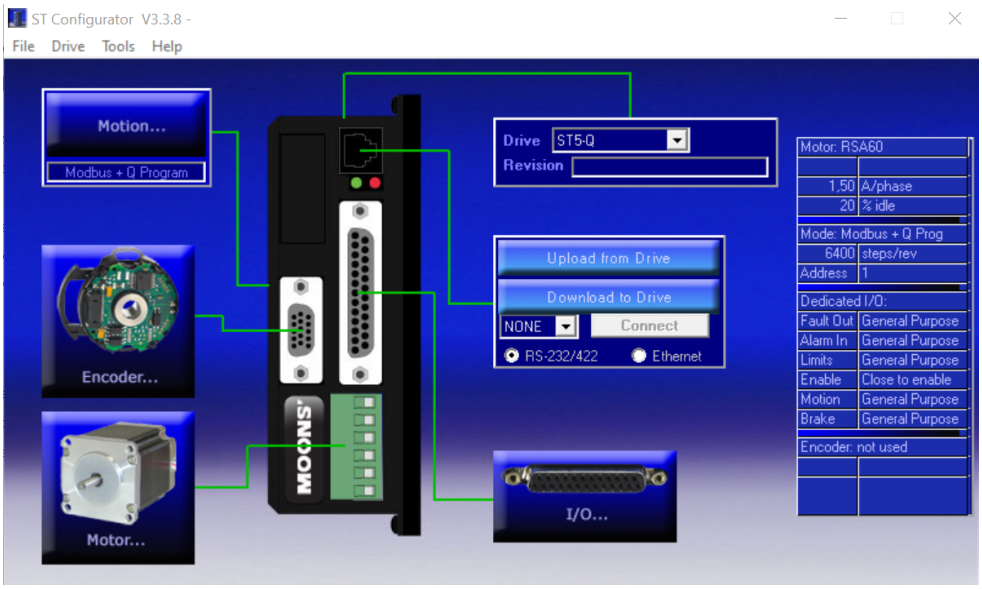


Figure 3.2: ST Configurator software for configuring the Moons motor drivers

3.3.5 Focus scan programs

A large part of the software development phase was spent on working out how to do focus scans which incorporate the back-focus. The implementation was done in the new class *ScanProgramRunner*, part of the ModusMain-module. Initially, each type of scan was implemented in its own function. The different types of scans are listed below. These scans could be invoked at whatever object angle (θ_Y) and rotation angle (θ_Z) desired.

1. Fixed defocus, fixed back-focus
2. Scan defocus, fixed back-focus
3. Fixed defocus, scan back-focus
4. Scan defocus, scan back-focus

Repeating code became an issue with the above structure, since (2) and (4) both scan the defocus, and similarly (3) and (4) both scan the back-focus. A different approach was developed, where all types of scans are handled through a common function, the *RunScanJob*-function. Instead of differentiating between fixed- and scanned defocus (or back-focus), it was decided to treat everything as scans. If only one position is requested the scan start and end positions are set equal.

With this new structure a list of all the measurement points is created by evaluating the scan start, stop and step size. A measurement point consist of a discrete position for back-focus, defocus, θ_Y and θ_Z. Running a scan is then simply a matter of looping through the measurement points, capturing an image and calculating the MTF focus score at each point.

3.3.6 Alignment in Modus

In order to be able to analyze the back-focus characteristics of a camera at object angles other than $\theta_Y = 0°$, the sensor must be aligned with the lens. Otherwise, points on the center that are equidistant from the sensor's center point will be focused at different object distances. A subsequent back-focus scan will then yield a misrepresentation of the relationship between the back-focus, defocus, object angle and focus score.

The procedure of alignment was described in section 1.1.2. Since IBAS 2 is a production machine, speed of operation is important. Therefore it has five collimators focused at infinity in fixed orientations. A single back-focus scan (z-scan) then gathers data at all five locations on the sensor (center and four corners).

Since Modus only has a single collimator, the z-scan must be performed five times, once for each location on the sensor. The five locations are reached by combining the collimator's rotation about the y-axis and the camera's rotation about the z-axis.

When writing the code for the alignment procedure, use was made of the existing Alignment module, which is a software module developed at PTS that contains all the functions for calculating the z-offset and tilt-offset of the sensor relative the lens, based on the focus data from a z-scan.

3.3.7 Graphical user interface

It was important that the resulting software be practical to use for any engineers at Axis intending to use Modus, regardless of whether they were familiar with the underlying software or not. For this purpose, a graphical user interface was developed through which a user can interact with the machine and receive useful feedback about the machine status, motor control and scanning progress.

There are two main subsystems available for developing graphical user interfaces in the .NET framework: WinForms and Windows Presentation Foundation (WPF). WinForms is essentially a class library with wrappers to access native windows graphical elements (windows, buttons, textboxes etc.) while WPF is a GUI framework for developing .NET applications. WPF is newer and offers several advantages over WinForms, such as better performance, improved separation between UI and data through bindings, improved security and flexibility. In contrast, WinForms is considered easier to learn and uses a graphical toolbox for UI design while WPF design is XML-based.[14, 15]

For the purposes of this project, either system is capable of producing the desired outcome. WPF was chosen because of the above mentioned advantages and because many of the recently developed GUIs at PTS uses WPF. The author also has more experience in working with WPF than WinForms. The GUI was developed in parallel with the business logic of the software as new features were implemented. For example, when alignment was implemented for Modus, a new panel in the GUI was introduced along with controls to start the alignment procedure, display the alignment settings and results. Adopting this workflow allowed for both the new feature and its accompanying UI to be tested simultaneously.

3.4 Electrical control panel

To accommodate the newly installed degrees of freedom, a new electrical control panel was required. The electrical control panel is where all electronic control systems for the motors, collimator, safety systems and more are located. The previous version of Modus featured four motorized degrees of freedom, and thus the electrical panel had four individual motor drivers. Figure 3.3 shows a view of the the old electrical panel. Cable management was essentially non-existent, the placement of the motor drivers (black boxes at the bottom) was inefficient and inaccessible, and there was no documentation for the control circuitry. Therefore, it was decided to completely strip the control panel and redo the layout and wiring from scratch. The main goals of the new electrical control panel was to:

- Improve the use of space by rearranging the motor drivers
- Improve the cable management
- Improve readability by labeling wires and electrical components
- Add support for additional motor drivers and sensors

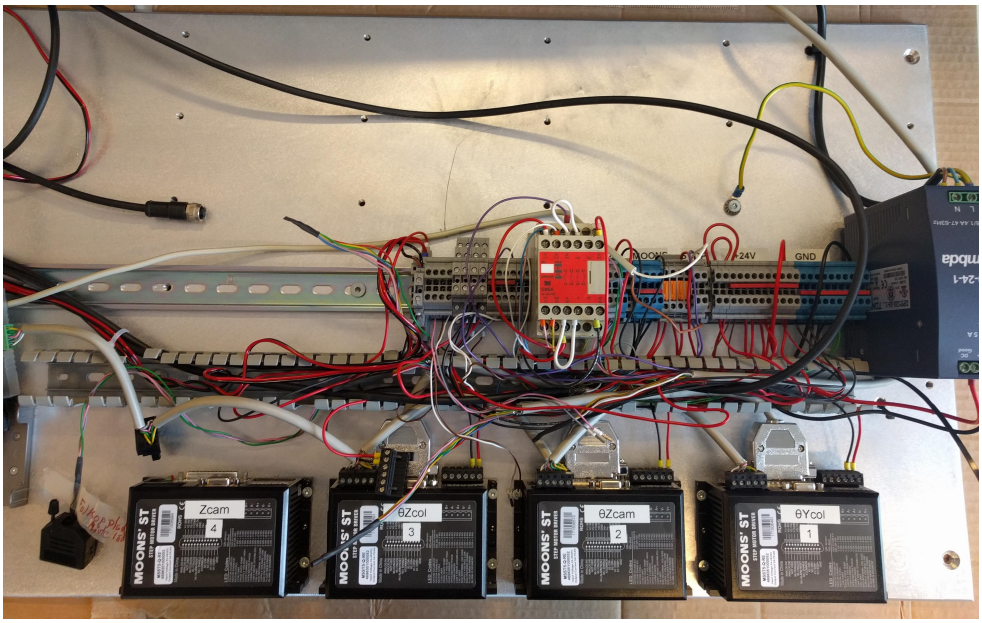


Figure 3.3: The old electrical panel for Modus

The layout of the new electrical panel followed a simple principle of grouping together components that serve the same purpose. Physical space was the main limita-

tion of the layout since Modus was originally designed for only four motor drivers, as seen in the previous figure. Existing electrical schematics for other Axis machines were adapted to the new Modus electrical panel as a temporary solution before an electrical engineer at Axis drew a proper electrical schematic. Electrical schematic construction was beyond the scope of this project mainly because of time constraints and lack of access to electrical schematic software.

The aluminium sheet in figure 3.3 functioned as the mounting panel for the electrical control equipment for the machine. Terminal blocks, power supplies, fuses and relays were mounted on a DIN-rail that was bolted to the sheet metal. The input was 240 VAC, transformed to 24 VDC in a power supply transformer. The aluminium sheet, as well as the other metallic surfaces of the machine are earthed through the 240 VAC earth cable. All electrical components reference the same common on the power supply's DC output side.

Everything related to power distribution was placed towards the left end of the rail, such as the power supply and terminal blocks to which each electrical component can connect to +24 VDC and ground. The entire DIN-rail was surrounded by wiring ducts, containing the bulk of the cable and allowing for access to the DIN-rail from all directions. The motor drivers were bolted directly to the aluminium sheet, oriented to take up the least amount of panel area. Section 4.1 includes photos of the completed electrical panel with labels for each of the different groups of terminal blocks.

A number of standard electrical tools and materials were used in the layout and construction of the electrical panel. These include, but are not limited to wires of varying size and color, wire cutters, wire strippers, soldering station. D-sub connectors of varying pin count, single-, double- and triple-leveled terminal blocks, terminal blocks with built-in manual switches, wiring ducts and electrical measurement equipment such as multimeter and oscilloscope.

3.5 Data collection

The Axis product catalog is extensive and split into a number of different categories. Because Modus uses the same fixtures as IBAS 2 it was necessary to select a camera which is assembled using the IBAS2 machine. Adhering to the limitations of the project, a fix-focal camera was chosen. It was decided to analyze a fish-eye lens camera internally referred to as Koi (as in koi fish because it belongs to a family of cameras with fisheye lenses).

The Koi camera features a 1.65mm F2.8 lens with FoV $> 180°$and a designed depth of field from 300 mm to infinity. Koi belongs to a category of cameras known as fixed-dome cameras. These are compact cameras with a dome casing. Their application area is virtually any environment where unobtrusive surveillance is required [16]. Figure 3.4 shows a generic type of fixed-dome camera in its assembled state. Figure 3.5 shows the sensor and lens of the Koi camera.

Figure 3.4: A generic camera in the Axis fixed-dome family

During sensor-lens alignment and assembly in the IBAS2 machine, 5 collimators project a target at infinity onto the Koi camera. One center collimator and four corner collimators, mounted at 150°field-of-view and in a 3:2 aspect-ratio. Figure 3.6 show an illustration of approximately the location of the five targets on the sensor when aligned. Due to physical limitations in Modus discussed in the Results section, alignment in Modus is possible at a maximum of 140° FoV, and as such the corner targets in Modus are slightly closer to the sensor center than in IBAS2.

The purpose of the data collection for the Koi camera was to establish the relation-

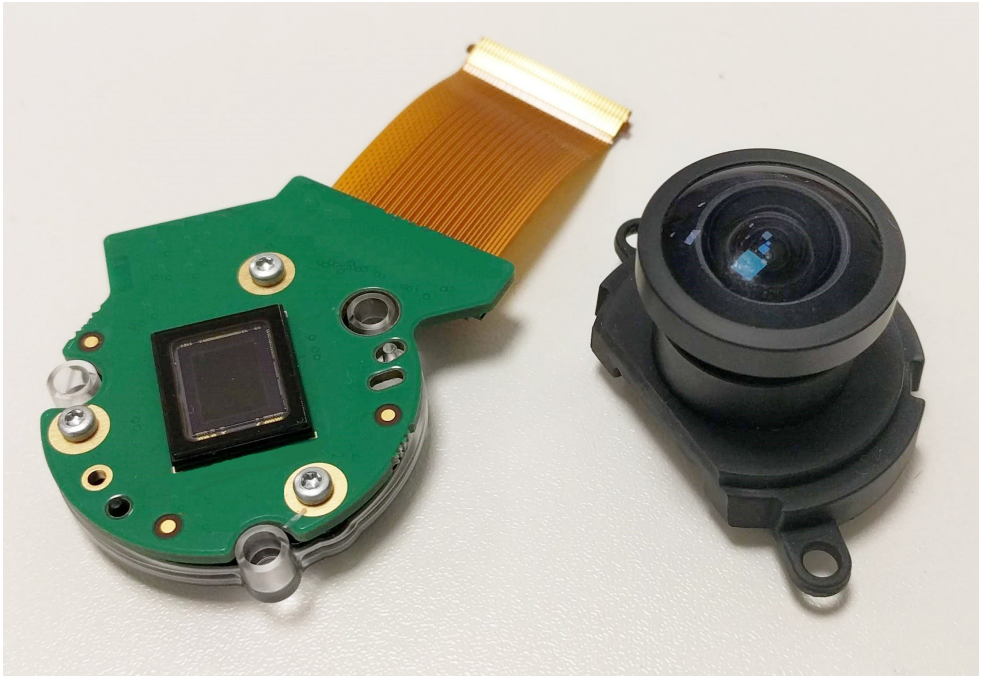


Figure 3.5: The sensor and lens of the Koi camera

ship between focus score and varying object distances, object angles (θ_Y) and back-focus (z). The sensor's orientation about the z-axis was not included in this scan, whose contribution to focus variance would be the result of non-rotational symmetry in the lens. Before the scan program, the alignment procedure was run, adjusting the sensor's z-position and x-y-tilt to be within the alignment criteria. Table 3.1 lists the sweep parameters and their respective start, end and step size. The total number of measurement points adds up to $9 \cdot 8 \cdot 7 = 504$. For each measurement point a cropped image was captured and the MTF-focus score calculated. The complete data collection were then plotted in a number of illustrative graphs, presented in section 4.4.

Table 3.1: Koi data collection parameters

Parameter	***Start***	***Stop***	***Step size***
Defocus (mm)	-2.0	0.0	0.25
θ_Y (°)	0	70	10
Z-offset (mm)	-0.15	0.15	0.05

Referring to the section on optical aberrations, see 2.3, the MTF focus score is

expected to drop off with increasing object angle (angular offset from the optical axis), which corresponds to the collimator rotating further from its homed position of 0°.

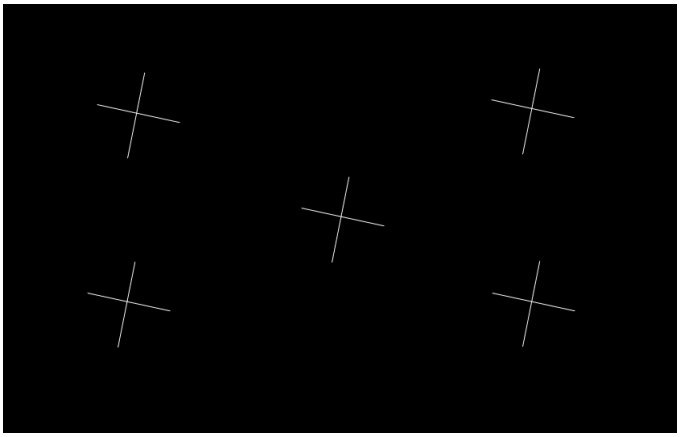


Figure 3.6: The location of the five collimator targets on the sensor during alignment

4 Results

4.1 Electrical control panel

The development of the new electrical panel was described in section 3.4. Figure 4.1 shows the completed panel before installation in Modus. The components attached on the DIN-rail are labeled X1-X14 and their function is listed in the right hand of the photo. The nine motor drivers are mounted below the DIN-rail, outside the cabling channels. The RS-485 serial communication between the drivers and the computer is seen in the bottom part of the picture as the grey cable wired in parallel between all the motor drivers.

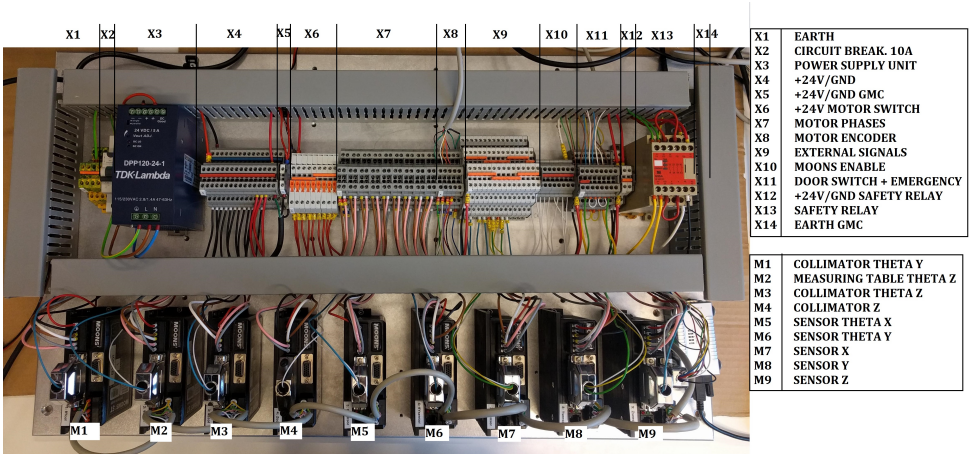


Figure 4.1: The electrical panel before installation in Modus

4.2 Software development

4.2.1 Software structure overview

The software developed for Modus is split into six different modules. Their hierarchical structure is illustrated in figure 4.2, where an arrow from A to B should be interpreted as B is dependent on A. The dashed line between the ModusGUI module and the other modules implies partial dependency, i.e. certain functions of the GUI are dependent on functionality provided by the other modules. In practice, the modules correspond to C#-projects, each containing one or more C#-classes. Each module is responsible for providing a specific functionality or a group of related functionalities.

In figure 4.2 and the block diagrams in subsequent sections, the color coding indicates whether the module existed prior to this thesis or not. Purple indicates an existing module that was used as is. Blue indicates an existing module that was further developed or rewritten to support new functions. Finally, green indicates completely new module or functionality. The following sections describes in detail a selection of the modules.

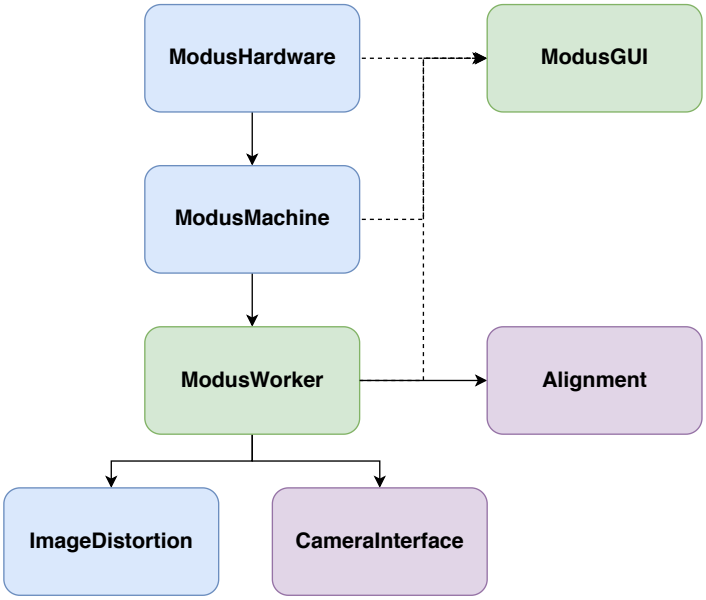


Figure 4.2: Modus software modules hierarchical structure

4.2.2 Modus modules details

The following sections describe in more detail the different modules of the Modus software.

4.2.2.1 ModusHardware

This module is responsible for establishing a connection with the hardware. This includes all motor drivers, the collimator position controller and LED controller. Figure 4.3 illustrates the classes of the ModusHardware module, their public properties and methods as well as how the classes depend on each other. The input parameter for the ModusHardware constructor is an instance of the *MachineSettings* class. This instance is supplied by the executable application, for instance by parsing a machine settings configuration file.

The *HardwareAvailability* class is responsible for connecting to the collimator and motor drivers via a Modbus protocol over RS-485. The *MachineSettings*-object described previously contains the settings for establishing this connection.

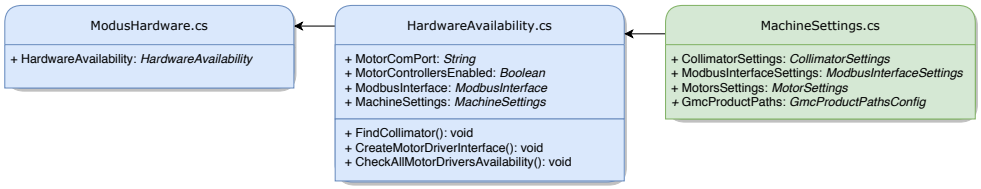


Figure 4.3: Overview of the ModusHardware module

4.2.2.2 ModusMachine

ModusMachine abstracts the hardware into groups which correspond to how the hardware is assembled in the machine, see figure 4.4. The interface to the controls for the collimator, both the object distance motor, and rotation of the collimator about the y- and z-axis are abstracted into a *CollimatorStack* class. Input parameter to the ModusMachine constructor is a *HardwareAvailability* object from the ModusHardware module.

Similarly, the *SensorStack* class is an abstraction which contains all the motors for moving and rotating the camera sensor. The *MeasuringTable* class represents the physical part in the machine which the SensorStack is attached to and provides methods for rotating the camera sensor and lens about the z-axis.

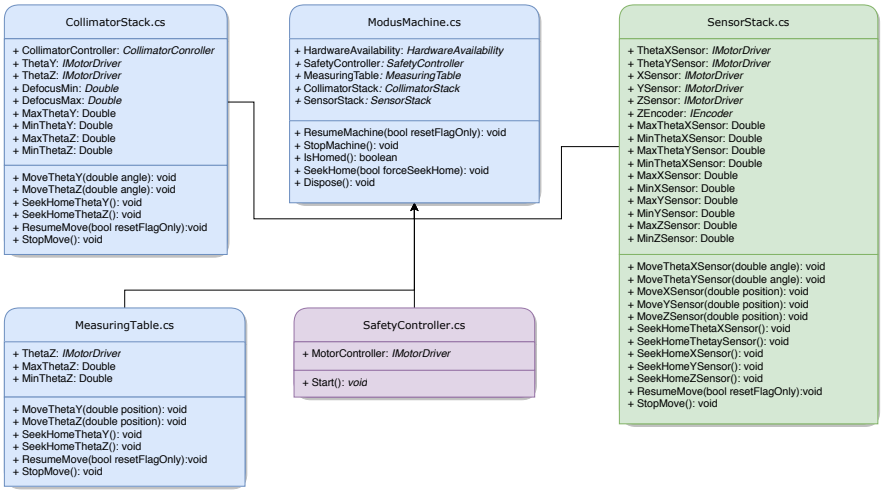


Figure 4.4: UML diagram for the ModusMachine module

4.2.2.3 ModusWorker

The ModusWorker module is the main workhorse of the Modus software. The *ModusWorker* class contains instances of several other classes within the module. The *AngleController* and *TranslationController* classes contains methods for calibrating the camera such that the sensor's x- and y-axis line up with those of the machine. It also finds a sensor z-coordinate at which the focus is above a specified threshold.

The *ScanProgramRunner* class performs focus scans, requiring a *ScanJob* object as input parameter in the *RunScanJob* method. The *ScanJob* class defines all parameters required for setting up and running a scan: start- and stop position, step size for z, defocus, θ_Y and θ_Z. This class is described in more detail in section 4.2.3.2.

The *ModusAlignment* class provides methods for running an alignment procedure in Modus. This is described more in detail in section 4.2.3.1

4.2.2.4 ModusGUI

The ModusGUI module contains the layout and design of the different parts of the GUI, along with the code-behind, i.e. the action performed of the different controls in the GUI when interacted with. Section 4.3 describes the function of the GUI in detail.

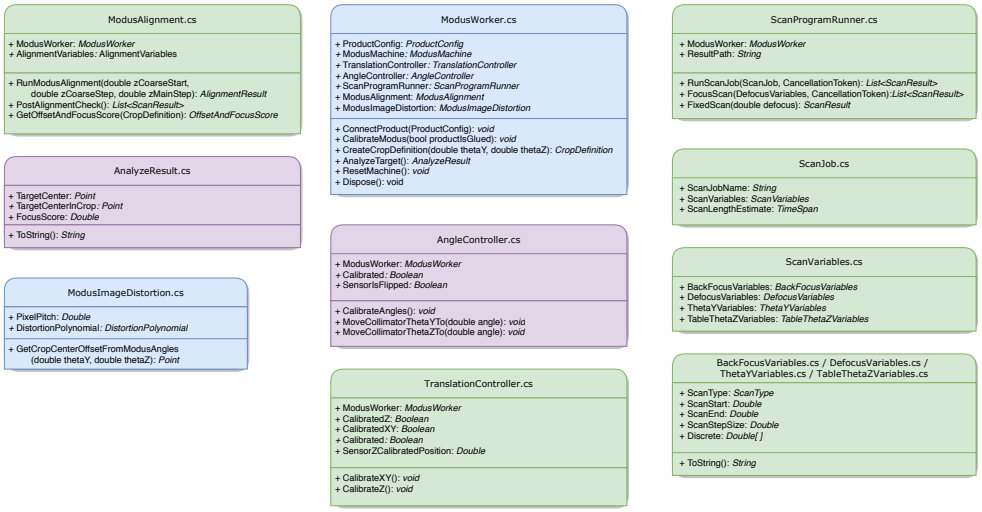


Figure 4.5: Class diagram for the ModusWorker module

4.2.2.5 CameraInterface and ImageDistortion

The CameraInterface module contains classes and methods for establishing a connection with the camera. It provides the possibility of connecting to the camera directly via SSH or via the Generic Maincard Controller (GMC). The GMC is a generic PCB which all different types of cameras can be connected through. This module is referenced by the ModusWorker in its camera setup-methods.

The purpose of the ImageDistortion module is to provide methods for fitting a polynomial to measured data. The number of decimals in the resulting polynomial coefficients is minimized by evaluating the goodness of the fit to the measured data.

4.2.3 Alignment and focus scanning

Two classes briefly mentioned in section 4.2.2.3 require further explanation: the ModusAlignment- and ScanProgramRunner-class.

4.2.3.1 Modus Alignment

Section 1.1.2 explained how the sensor and lens are aligned in the IBAS 2 machine. The alignment procedure in Modus is essentially the same, apart from a few distinct differences. IBAS 2 is equipped with five collimators: one center and four corner collimators. In contrast, Modus has only a single collimator which can rotate

about the y- and z-axis, along with rotation of the sensor about the z-axis, to project the cross-hair at any location on the sensor (within the limits described in section 1.1.3.1).

Consequently, the Modus alignment procedure begins by performing a coarse and main z-scan at the sensor's center position, followed by rotation of the collimator and sensor to perform the scans at a corner point. This is repeated for all the four corners. Once this is done, the data gathered is identical to what IBAS 2 gathers in a single coarse and main z-scan. The computation of the alignment is identical in Modus and IBAS 2. This was possible by importing the Alignment software module used in IBAS 2 into the Modus software environment.

4.2.3.2 ScanProgramRunner

As previously stated the *ScanProgramRunner* class is responsible for running scan jobs. The parameters for the scan jobs are imported through a scan job configuration file (.toml). Toml is an easy, minimal configuration file format. Figure 4.6 shows an example of what a scan job configuration file with toml-syntax might look like. This file is imported and parsed to populate the classes shown previously at the bottom right of figure 4.5.

The *ScanProgramRunner* class contains the method *RunScanJob*, taking an instance of the *ScanJob* class as input parameter. This method is essentially a sequence of nested for-loops where each loops through a specific scan job parameter. The pseudo-code below shows the order of the nested for-loops. Rotation of the collimator and the sensor is generally slower than movement of Z or the defocus, so by placing the former two in the two most outer loops the total time for the scan job to complete is lowered.

```toml
[[ScanJobs]]
    ScanName = "Koi_UnitComparison"

    [ScanJobs.ScanVariables]

        [ScanJobs.ScanVariables.BackFocusVariables]
        ScanType = "Sweep"                  # "Sweep" or "Discrete"
        SweepType = "RelativeToAlignment"   # "RelativeToAlignment" or "Absolute"
        ScanStart = -0.000015
        ScanEnd = 0.000015
        ScanStride = 0.000005
        Discrete = [0.001950, 0.001960, 0.001970]

        [ScanJobs.ScanVariables.DefocusVariables]
        ScanType = "Sweep"                  # Possible: Sweep, Search, Discrete
        CoarseStart = -0.002
        CoarseEnd = 0.000
        CoarseStride = 0.0005
        FineIterations = 0
        FineStride = 0.000

        SearchFocus = 0.15
        SearchFocusTolerance = 0.005
        SearchStart = -0.005
        SearchStride = 0.001
        SearchMinimumDefocus = -0.004
        SearchMaximumDefocus = 0.000
        Discrete = [0.0]

        [ScanJobs.ScanVariables.ThetaYVariables]
        ScanType = "Sweep"              # Possible: Sweep, Discrete
        ScanStart = 0.0
        ScanEnd = 70.0
        ScanStride = 10.0
        Discrete = [0.0]

        [ScanJobs.ScanVariables.TableThetaZVariables]
        ScanType = "Discrete"           # Possible: Sweep, Discrete
        ScanStart = 0.0
        ScanEnd = 0.0
        ScanStride = 0.0
        Discrete = [0.0 ]
```

Figure 4.6: Example of a scan job configuration file

```
RunScanJob(ScanJob)
{
    foreach(SensorThetaZ)
    {
        RotateToNextThetaZ();

        foreach(CollimatorThetaY)
        {
            RotateToNextThetaY();

            foreach(SensorZPosition)
            {
                MoveToNextZ();

                foreach(CollimatorDefocus)
                {
                    MoveToNextDefocus();
                    CaptureImage();
                    CalculateMTF();
                    WriteResultToFile();
                }
            }
        }
    }
}
```

Listing 4.1: Pseudo code for RunScanJob-method

4.2.3.3 External references

In addition to the modules outlined in the preceding sections, the Modus software also makes use of a number of external modules, i.e. existing software APIs and libraries. The APIs are imported for use as NuGet packages. APIs developed both internally at Axis and externally by a third-party were used. Below follows a list of modules used and a brief description of their function.

Internal Axis APIs/libraries:

- **Imaging**: Capturing fullsize/cropped images from the camera sensor. Image processing and analysis methods for finding a cross-hair in the image and determining the focus by means of the MTF- or gradient method.
- **GeneralClient/GMCC**: Configuring and communicating with an Axis generic

main card.

- **StepperMotorController**: API for controlling stepper motor drivers from the Moons manufacturer and using the Modbus protocol.

External APIs/libraries:

- **NModbus4**: A C# implementation of the Modbus protocol, providing all the functions as specified by the Modbus protocol.
- **MathNet.Numerics**: Methods and algorithms for numerical computations.
- **NLog**: Open-source logging for .NET environment. Used to log the actions performed by Modus and the state of the machine.
- **TriopticsCollimatorController**: API for controlling a variable object distance collimator.
- **Extended.Wpf.Toolkit**: Provides extension tools for WPF.
- **Nett**: Library for reading and writing TOML-files.

4.2.4 Data processing and visualization scipts

As the pseudo-code in listing 4.1 showed, the RunScanJob-method writes the results from the scan to file. Specifically, it's a comma-separated text file (.csv). For each measurement point (combination of the scan job parameters) the MTF is calculated and the result written to a separate row in the text file. Figure 4.7 shows an example of a file output. The parameter start, stop and step sizes preamble the result. The first 9 measurement points of the particular scan in question is shown. As the figure illustrates, for each measurement point the back-focus (z), defocus, collimator θ_Y, sensor θ_Z, focus score and target x- and y-coordinate is reported. Lines preambled by the hash-symbol, #, are comments and subsequent parsing of the file can ignore this.

This raw data can then be visualized in a number of ways, section 4.4 demonstrates this. Python scripts were written that import the result file, perform some basic data processing, creates plots and then saves the figures to a desired location on disk. These scripts are run from the Python command line or through a Python IDE.

```
# Scan start-time: 180659
# Collimator theta-Y scan type = Sweep
# Scan start = 0,000000
# Scan end = 70,000000
# Scan coarse stride = 10,000000

# Measuring table theta-Z scan type = Discrete
# Discrete values = [0]

# Back-focus scan type = Sweep
# Sweep-type = Relative to alignment
# Scan start = -0,000015
# Scan end = 0,000015
# Scan stride = 0,000005

# Collimator defocus scan type = Sweep
# Scan coarse start = -0,002000
# Scan coarse end = 0,000000
# Scan coarse stride = 0,0005
# Scan fine iterations = 0,000
# Scan fine stride = 0,000000


# [Back-focus, Defocus, Coll@Y, Table@Z, Focus, TargetX, TargetY]


0.001751, -2, 0, 0, 0.179732, 1749, 1265
0.001751, -1.5, 0, 0, 0.170895, 1750, 1265
0.001751, -1, 0, 0, 0.16129, 1750, 1265
0.001751, -0.5, 0, 0, 0.151416, 1750, 1265
0.001751, 0, 0, 0, 0.140076, 1750, 1265
0.001756, -2, 0, 0, 0.196905, 1749, 1265
0.001756, -1.5, 0, 0, 0.192522, 1750, 1265
0.001756, -1, 0, 0, 0.185203, 1750, 1265
0.001756, -0.5, 0, 0, 0.175033, 1750, 1265
```

Figure 4.7: Example of a scan job result output file

4.3 User interface

4.3.1 Graphical user interface

The graphical user interface is split into six different tabs which each serve a particular function, as shown below in figure 4.8a through 4.8f.

The control panel (fig. 4.8a) allows the user to control all the motors of the machine. Each motor can be homed and moved in either direction. All motors can also be controlled at once by stopping or resuming motion, and by homing all simultaneously. In addition to motor control, the control panel also allows for control of the collimator by setting the LED brightness and positioning the object by controlling the defocus. The figure shows the control panel in a disabled state, prior to loading the machine settings. The machine settings are specified in a TOML-configuration file, which contains all the settings pertaining to the motors, collimator, additional settings paths and the Modbus communication protocol. The machine settings are loaded through the menu: File - Load Machine settings.

The Load Product tab is where the user loads settings for a specific camera. The contents of the configuration file are displayed for the user and can be changed in-

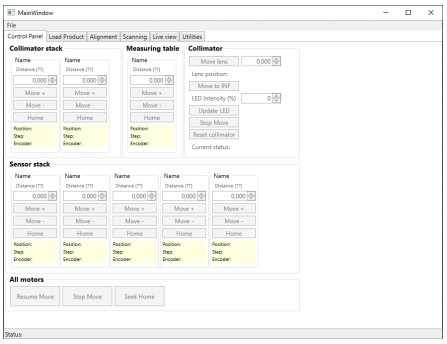
teractively, before the camera is connected. The Connect-button establishes a connection between the software and the GMC, allowing for images to be captured, as well as camera settings such as shutter speed and signal gain to be configured. Upon successful connection, the Calibrate-button will be enabled. Its function is to run a number of calibration procedures. It first moves the sensor in the z-direction, looking for a position where the focus is above the specified calibration minimum focus. This position is required for the ensuing calibrations.

In it's homed position the motor controlling the sensor's rotation about the z-axis will offset the sensor's y-axis from the machine's y-axis. This is adjusted for by rotating the collimator $\pm 15^{\circ}$, registering the target's x- and y-location on the sensor and then calculating the offset angle from these locations. The third and final calibration is to position the sensor in x & y such that the target is centered on the sensor when the collimator is at 0°.

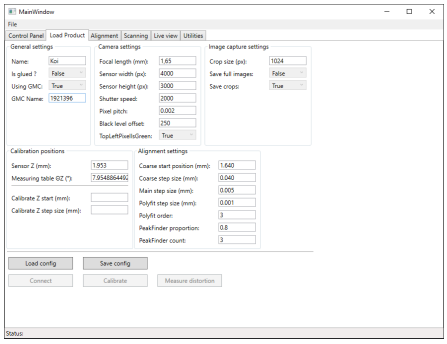
The alignment tab allows the user to align the currently connected and calibrated camera. The alignment procedure can be cancelled at any time, which requires the procedure to restart from the beginning if started again, i.e. a pause-resume function is currently not implemented. The user has the option to enable a post-alignment check to be done, which essentially means that after the alignment the focus is measured once more at the center point and four corner points. The result is displayed in the post-alignment result box. The aligned z-coordinate is also displayed after the completion.

Through the scan job tab, the user can load a scanjob-file, edit the settings and then start the focus scan. The live view tab continuously updates, showing the lastest image that was captured, the focus score calculated, and the location of the crosshair on the sensor.

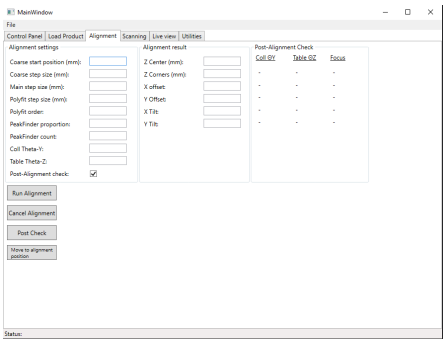
Finally, the utilities tab is a tool-type tab, where currently the object distance can be calculated given the collimator's focal length, the camera lens' focal length, the defocus and the distance between the collimator lens and the camera lens.



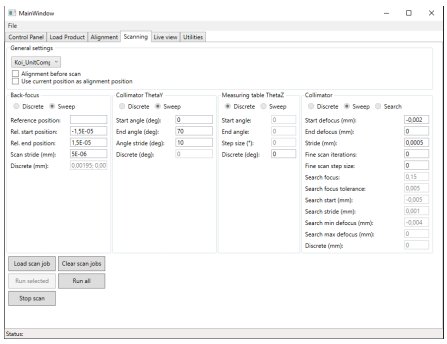
(a) The control panel tab



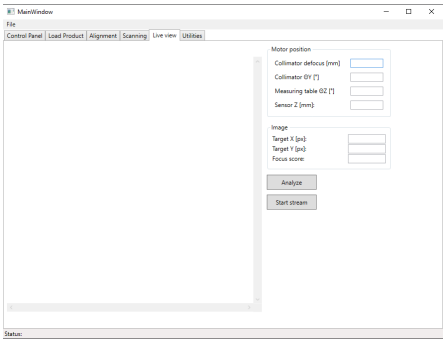
(b) The product loading tab



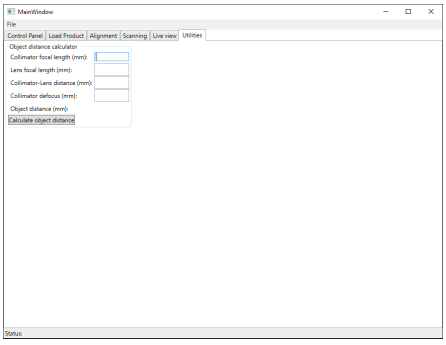
(c) The alignment tab



(d) The scan job tab



(e) The live view tab



(f) The utilities tab

Figure 4.8: The graphical user interface for Modus

4.3.2 Setup and start of a scan

This section describes the steps involved in setting up the Modus machine, loading a product and running a scan, from start to finish. The intention of this section is that anyone looking to use the machine in order to collect some focus data about a specific camera should be able to do so independently by following the steps laid out below.

Step 1: Installation of camera:

1. Install the lens and sensor into the machine, onto the respective fixture of each, see figure 4.9 & 4.10
2. Switch on the compressed air which develops suction between the camera sensor and its fixture, improving the fixation.
3. Close all doors to the machine
4. Turn on the power to the generic main card (GMC)
5. Connect the USB- and Ethernet-cable to your computer

Step 2: Software start-up:

1. Start the ModusGUI software
2. Load the machine settings through File $->$ Load machine settings. Navigate to the machine settings configuration file (filetype TOML).
3. Wait for the software to establish a connection with the motors and collimator
4. Home the motors if they are not already homed. In the control panel tab, click the Seek Home-button. This should not be required between subsequent runs of the software unless the power has been cut to the machine or something went wrong, meaning that the motor has lost its position.
5. In the Load product tab, click Load config and navigate to the desired product configuration file.
6. Click the connect-button to connect the software to the GMC. If the connection times out, cycle the power to the main card and try again.
7. Click the calibrate-button to calibrate the loaded camera.
8. If no distortion polynomial is available, click the Measure distortion-button to begin measuring it.

Step 3: Alignment and scan program start

1. In the Alignment tab, review the alignment settings. Make changes as needed.
2. Check the post-alignment checkbox
3. Start the alignment procedure by clicking the Run Alignment-button.
4. If, for any reason, the alignment procedure should be cancelled, click the Cancel Alignment-button.
5. Upon alignment completion, review the results in the display.

Step 4: Scan program start

1. In the Scanning tab, load a scan job file by clicking the Load scan job-button and navigate to the desired file.
2. Review the scan job parameters and change as desired.
3. Select the scan job to run in the drop down box at the top and the click on Run selected. Alternatively, click on Run all to run all the loaded scan jobs in sequence.
4. Wait until the scan completes, cancel at any time by clicking on the Stop scan-button.

Step 5: Result output and completion

1. Navigate to the results output folder to find the result from the alignment and the scan jobs.
2. Analyze and visualize the raw data using one or more of the provided Python scripts or differently.
3. Close the ModusGUI application. This closes the connection with the generic main card, the motor drivers and the collimator.
4. Disconnect the USB- and Ethernet-cables.
5. Turn off the power to the genereic main card and switch off the compressed air.
6. Remove the sensor, lens and fixtures from the machine.

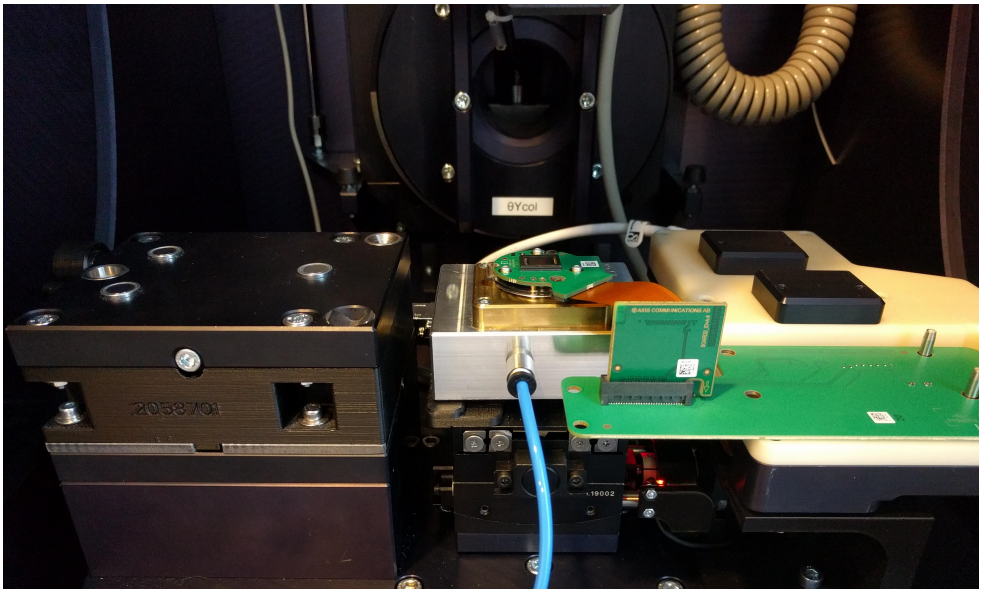


Figure 4.9: Installation of a sensor fixture and sensor

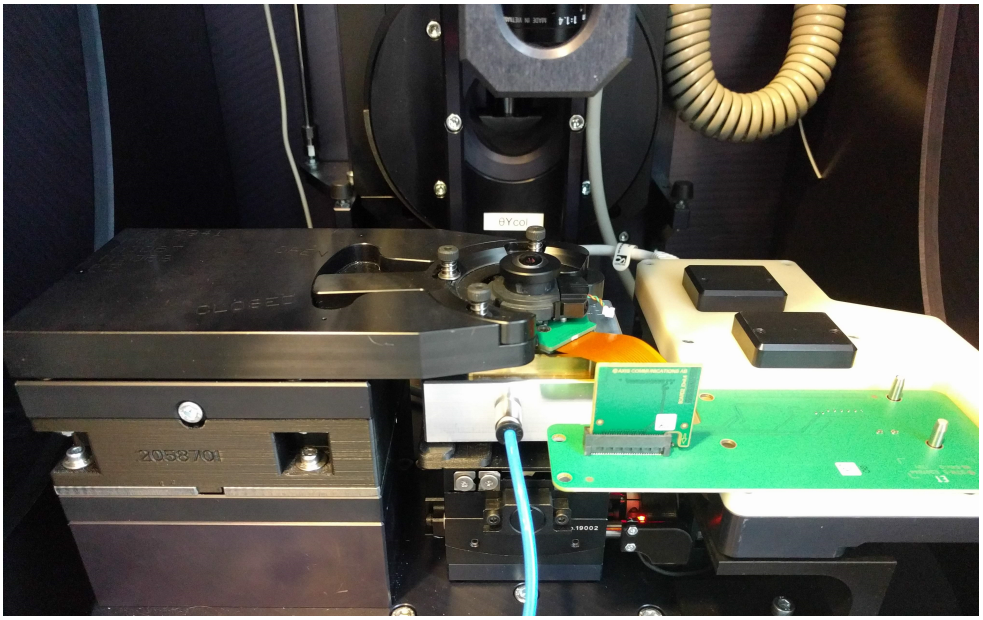


Figure 4.10: Installation of lens fixture and lens

4.3.3 Configuration files

The user must supply two configuration files in order to initialize the machine and connect to a camera: the machine settings configuration file and the product configuration file. Below follows a brief description of what's included in each file and how they're structured.

Machine settings configuration:

This configuration file contains settings for the 9 different motors, the Modbus communication protocol, the collimator as well as local paths to settings for the GMC. Figure 4.11 shows an extract from the Modus machine settings configuration file currently in use. It shows the settings for local GMC paths, the collimator and the motor controlling the rotation of the collimator about the y-axis.

Camera product configuration:

The product configuration file for the Koi camera is shown in figure 4.12. It contains information about the specific camera such as sensor size, focal length as well as calibration and alignment parameters (start- and stop position, step size).

ModusMachineSettings.tml

```
[GmcProductPaths]
    ProductPath = "C:\\GIT\\super_ibas2\\IBAS2Configuration\\Products"
    FpgaImagePath = "C:\\Stubbar"
    MotorFirmwarePath = "C:\\Stubbar"
    PowerSequencePath = "C:\\GIT\\super_ibas2\\IBAS2Configuration\\PowerSequences"
    MotorConfigPath = "C:\\GIT\\super_ibas2\\IBAS2Configuration\\Optics"
    SensorConfigPath = "C:\\GIT\\super_ibas2\\IBAS2Configuration\\Sensors"

[CollimatorSettings]
    CurrentCollimator = "Collimator25MM"

    [CollimatorSettings.Collimator25MM]
    InfinityPositionSteps = 62400
    InfinityPositionMM = 9.75
    LensFocalDistanceMM = 25.0
    LowLimitMM = 0.5
    HighLimitMM = 17.5

    [CollimatorSettings.Collimator75MM]
    InfinityPositionSteps = 57600
    InfinityPositionMM = 10.7
    LensFocalDistanceMM = 75.0
    LowLimitMM = 0.5
    HighLimitMM = 17.5


[Motors.ModbusInterface]
    AutoDetect = true
    FindControllerId = 1

[Motors.CollimatorThetaY]
    Simulate = false
    MotorId = 1

    [Motors.CollimatorThetaY.MotorParameters]
        HomingOffset = 0.00
        StepsPerRev = 6400
        RevPerSiUnit = 28.6478897565
        InvertDirection = false
        Velocity = 500
        FastVelocity = 1500
        Acceleration = 50
        Deceleration = 50
        HomingVelocity = 250
        HomingAcceleration = 50
        HomingDeceleration = 75
        Backlash = 0.002617993878
```

Figure 4.11: Part of the machine settings configuration file for Modus

```
# Product configuration file for Modus

# General info
ProductName = "Koi"                         # Internal axis camera name, e.g. Koi, Superfisken etc.
ProductIsGlued = false                      # Indicates whether the sensor and optics are glued to each other.
ResultFileName = "result.csv"               # Result file name incl. extension
CollimatorToLensVerticalDistance = 0.080    # [SI-UNIT]. The vertical distance between the collimator and lens.

# Calibration parameters
ZCalibrationPosition = 0.001953             # [SI-UNIT]. Position where focus is good enough (@ defocus = 0) to perform the other calibrations
ZCalibrationStart = 0.001600                # [SI-UNIT]
ZCalibrationStride =    0.000040            # [SI-UNIT]
ZCalibrationFineStride = 0.000010           # [SI-UNIT]
ZCalibrationMinFocus = 0.15
MeasuringTableThetaZCalibrationPosition = 7.95488644925065      # Degrees

# Alignment parameters:
AlignmentCollimatorThetaYAngles = [ 50.0, -50.0 ]           # [Degrees] Do not include 0 degrees here
AlignmentMeasuringTableThetaZAngles = [ -45.0, 45.0 ]       # [Degrees] Do not include 0 degrees here
AlignmentCoarseZStartPos = 0.001640                         # [SI-UNIT]
AlignmentCoarseZStepSize = 0.000040                         # [SI-UNIT]
AlignmentZStartPos = 0.001640                               # [SI-UNIT]
AlignmentZStepSize = 0.000005                               # [SI-UNIT]
AlignmentPolyFitStepSize = 0.0000005                        # [SI-UNIT]
AlignmentPolyFitOrder = 3
AlignmentPeakFinderProportion = 0.8
AlignmentPeakFinderCount = 3

# Polynomial coefficients for estimating target position based on collimator angle.
# Remove or leave blank to measure in machine:
DistortionPolynomialCoeffs = [ "0.03", "0" ]
InverseDistortionPolynomialCoeffs = [ "35.03", "0.03" ]

# Camera parameters:
ShutterSpeed = 2000
CropDimension = 1024            # [Pixels]
TopLeftPixelIsGreen = true
SaveFullImages = false
SaveCrop = true
UsingGMC = true
GmcProductName = "1921396"
```

Figure 4.12: The product configuration file for the Koi camera

4.4 Data collection

4.4.1 Koi focus scan data

Focus data was collected for a Koi unit at the measurement points described in section 3.5. With the scan limited to three sweep parameters (back-focus, defocus, object angle) the focus score can be treated as a function of three independent variables:

$$focus = f(z, d, \theta_Y) \tag{4.1}$$

where z, d and θ_Y is the back-focus, defocus and object angle respectively. When $d = 0$ the cross-hair is placed at the collimator lens' focal point, resulting in the cross-hair appearing to be infinitely far away. When $d < 0$ the cross-hair appears closer than infinity and when $d > 0$ it appears beyond infinity. Beyond infinity simple means that the location of the image plane of a camera looking at an object beyond infinity will be in front of the camera's rear focal plane.

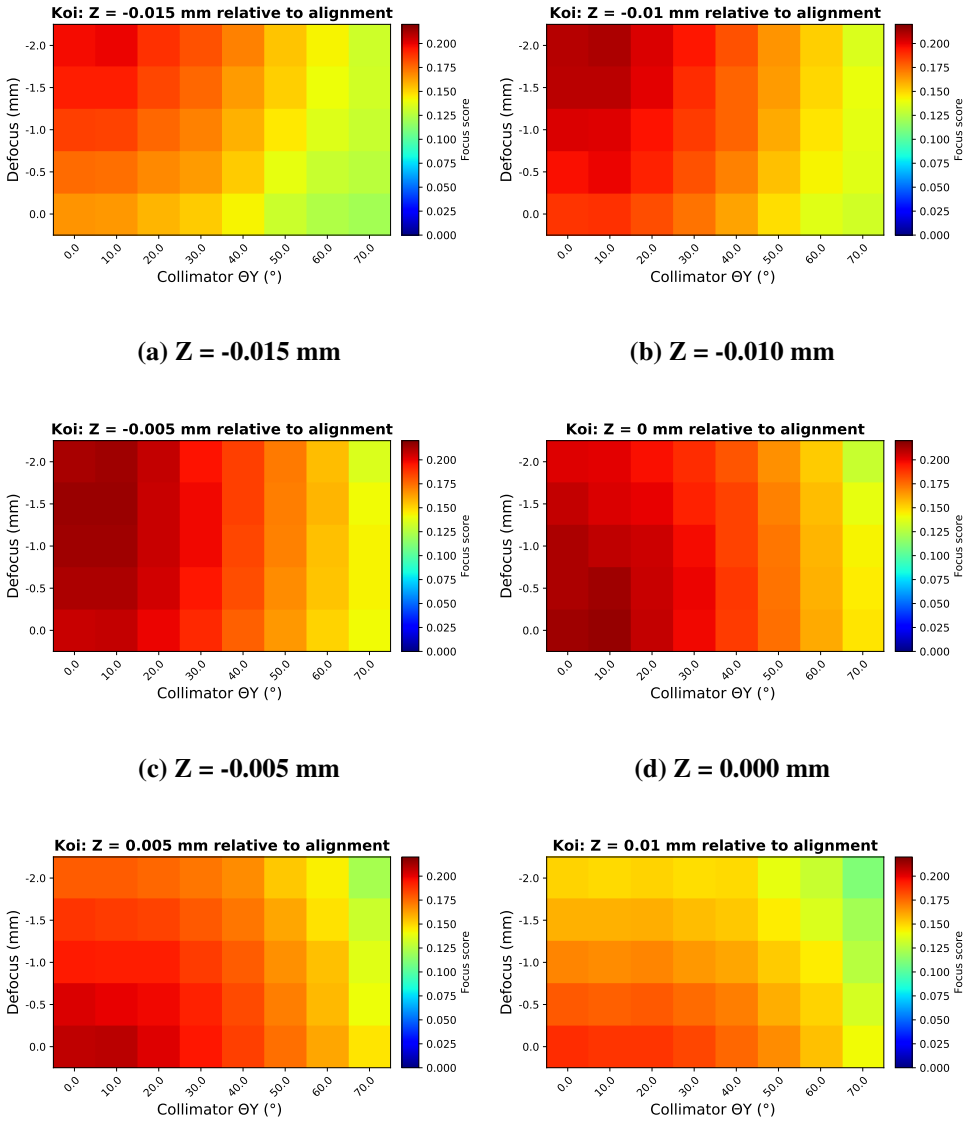
The focus score was calculated at 504 measurement points. The duration of the scan was approximately 25 minutes, with an average of 3 seconds per measurement point. This duration does not include the set-up of the machine, neither the alignment procedure. The duration of the alignment procedure can vary from unit to unit. Some units require more than one iteration of the alignment procedure. Each iteration takes approximately 5 minutes to complete (60 seconds per alignment position $\times$ 5 alignment positions).

The complete data collection is presented in a number of contour plots, see figure 4.13a through 4.13f. The focus score is color represented and each contour plot represents a specific offset from the aligned back-focus (z). The x-axis represents the viwing angle, θ_Y, and the y-axis the defocus. A defocus of -2.0 mm corresponds to an object distance of 25 cm. In figure 4.14a to 4.14f a bilinear interpolation has been applied to the data set.

As seen in figure 4.13 and 4.14 focus score is maximized when the z-offset, defocus and collimator θ_Y = 0. Focus at the near-point, when defocus = -1.75, is maximized at a z-offset of approximately -0.005 mm. The contours also illustrate that the maximum focus at 70° occurs at the same z-offset as for 0°, regardless of the defocus.

Following the contours from left to right, i.e. with increasing collimator angle, the focus score consistently decreases at all defocus levels. In other words, the focus

decreases as the target moves from the sensor's center point towards the edge, no matter if the object is at infinity or at the lens' near-point.

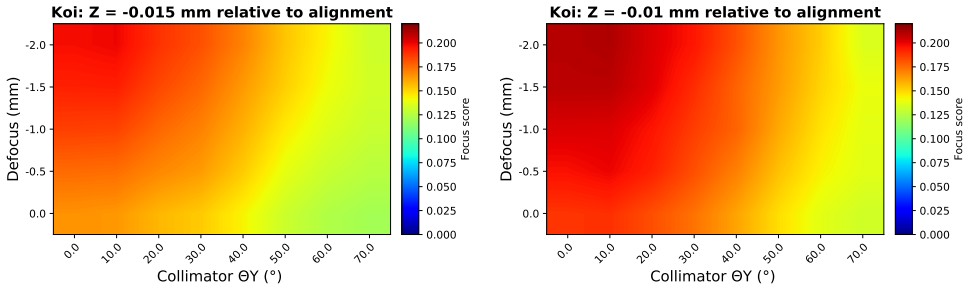


(a) Z = -0.015 mm **(b) Z = -0.010 mm**

(c) Z = -0.005 mm **(d) Z = 0.000 mm**

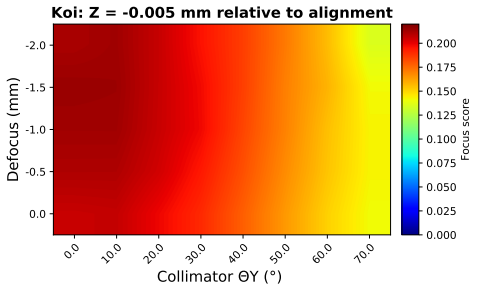
(e) Z = 0.005 mm **(f) Z = 0.010 mm**

Figure 4.13: Contours of focus vs. collimator θ_Y and defocus at different z-offsets

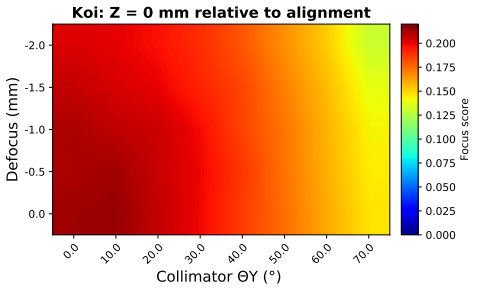


(a) Z = -0.015 mm

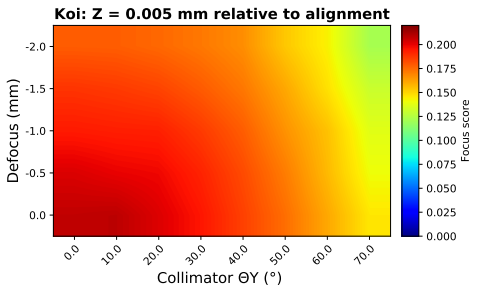
(b) Z = -0.010 mm



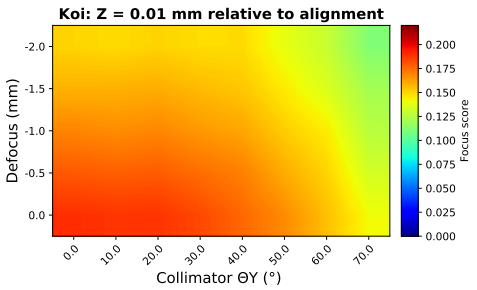
(c) Z = -0.005 mm



(d) Z = 0.000 mm



(e) Z = 0.005 mm



(f) Z = 0.010 mm

Figure 4.14: Contours of focus vs. collimator θ_Y and defocus at different z-offsets with bilinear interpolation

From the complete scan data illustrated in the contour plots, the focus score can be extracted as a function of z, at $\theta_Y = 0, 70°$, as well as at infinity and the near-point. This is shown in figure 4.15, where the data has also been fitted by a third order polynomial. The blue curve in the image shows the relationship between the back-focus offset (z-offset) and the focus score for a target at infinity (defocus = 0) and 0° object angle. The red curve is the same relationship, but now with the target at the near-point of 300 mm (defocus = -2 mm). The green and black curves illustrate the same relationships at an object angle of 70°. The coefficients of the polynomials fitting the four data sets is presented in table 4.1.

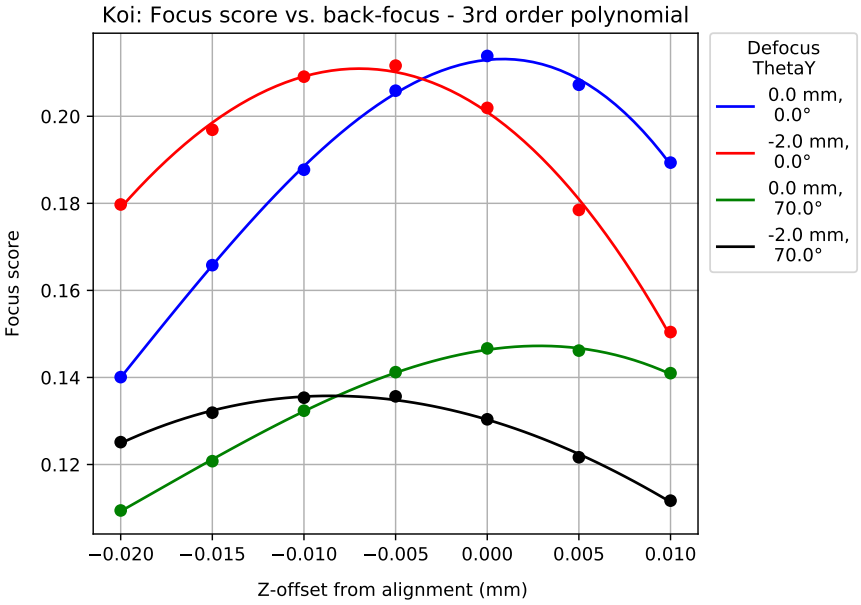


Figure 4.15: Focus vs. backfocus at infinity/nearpoint & center/edge

Table 4.1: Coefficients for focus-backfocus polynomial regression

Defocus (mm)	$\theta_Y\,(°)$	p_3	p_2	p_1	c
0	0	−4.066e4	−2.425e2	4.262	2.130e−1
-2.0	0	−8.251e4	−2.166e2	−2.901	2.010e−1
0	70	−1.814e3	−9.857e1	6.088e−1	1.464e1
-2.0	70	3.949e2	−6.819e1	−1.251	1.303e−1

The top focus scores, based on the polynomial fit, has been extracted from figure 4.15 and are presented in table 4.2. The colored percentages show the difference relative to the focus score at the alignment position (z-offset = 0, θ_Y = 0), i.e. where the focus score is 0.214.

Table 4.2: Top focus score at infinity/near-point and center/edge

Z-offset (mm)	$\boldsymbol{\theta_Y}$ **(°)**	$\boldsymbol{Focus(\infty)}$	**Focus (300 mm)**	**Diff.**
0	0	0.214	0.201	0.013 (6%)
-0.005	0	0.206 (-3.7%)	0.210 (+4.3%)	0.004 (2%)
0	70	0.147	0.131	0.016 (11 %)
-2.0	70	0.142 (-3.4%)	0.135 (+3.1 %)	0.007 (5 %)

With the polynomial fit it is possible to determine the depth of focus at different focus scores. Section 2.2.2 introduced the depth of focus concept. The depth of focus can be seen as the tolerance of placement of the sensor, i.e. the size of the window where the sensor must be placed in order to achieve a certain focus score.

Figure 4.16 illustrates how the depth of focus can be extracted from the relationship between the focus score and the back-focus. With a focus requirement of 0.20 at $\theta_Y = 0°$, a horizontal line at 0.20 intersects the two curves at four back-focus values. The two central intersections are of interest, since between these two points the focus score will be above 0.20 both at infinity and at the near-point. This depth of focus is approximately 7-8 μm.

Similarly, at $\theta_Y = 70°$ a focus requirement of 0.13 results in a depth of focus of approximately 14 μm. Table 4.3 shows the depth of focus at three different focus score requirements at $\theta_Y = 0°$ and $\theta_Y = 70°$, respectively.

Table 4.3: Sensor z-position tolerance at varying focus requirements and collimator angles

Focus requirement	$\boldsymbol{\theta_Y}$ **(°)**	***Min. z (mm)***	***Max. z (mm)***	***Depth of focus (mm)***
0.20	0	−0.0070	0.0005	0.0075
0.18	0	−0.0120	0.0050	0.0170
0.16	0	−0.0160	0.0085	0.0245
0.13	70	−0.0112	0.0003	0.0115
0.12	70	−0.0156	0.0062	0.0218
0.11	70	−0.0196	0.0107	0.0303

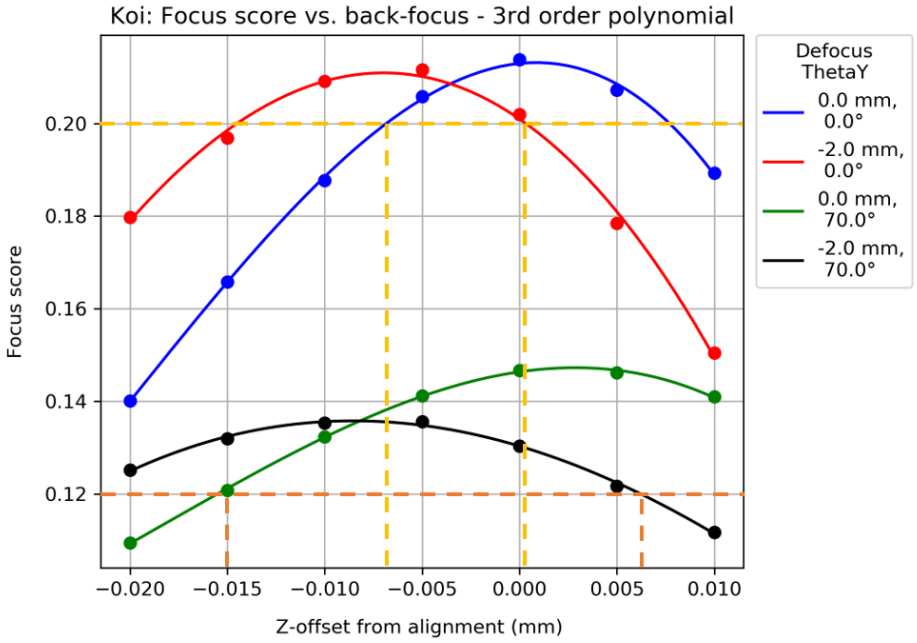


Figure 4.16: Back-focus tolerance to satisfy focus score 0.20 and 0.13

To better visualize how the depth of focus changes with focus score, figure 4.17 was extracted from the polynomial fits. It shows the relationship between the focus score requirement and the depth of focus. As can be seen, the gradient of the depth of focus (back-focus tolerance) curve is steeper at $\theta_Y = 70°$ than $\theta_Y = 0°$. Thus, by lowering the focus score requirement by a certain amount, the depth of focus will increase more towards the edge of the sensor than the center.

Figure 4.18 shows the same result as figure 4.17, however here the focus score requirement is represented as a percentage relative to the highest focus score. That is, 100 % for the $\theta_Y = 0°$-curve corresponds to the highest focus measured at this angle. Likewise, 100 % for the $\theta_Y = 70°$-curve corresponds to the highest focus score measured at this angle. Thus, this figure makes it possible to read out the depth of focus at both object angles, given a focus requirement relative to the alignment position. For example, at 95 % relative focus score, the depth of focus is approximately 9 μm and 13 μm, at $\theta_Y = 0, 70°$ respectively.

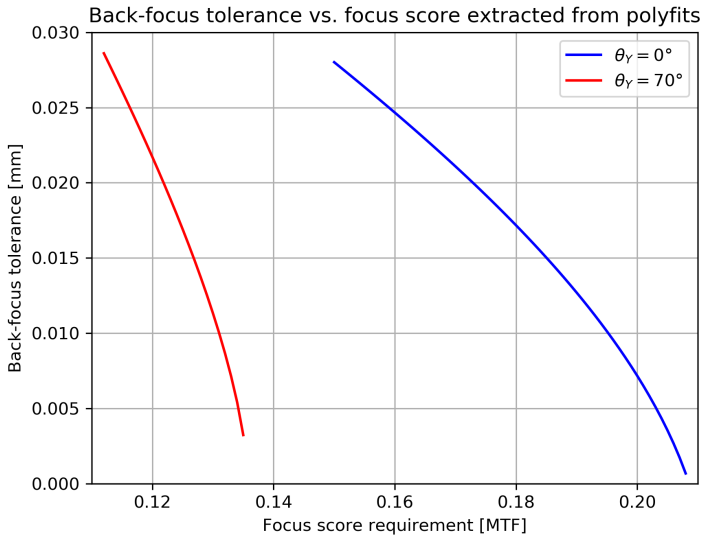


Figure 4.17: Depth of focus vs. focus score requirement

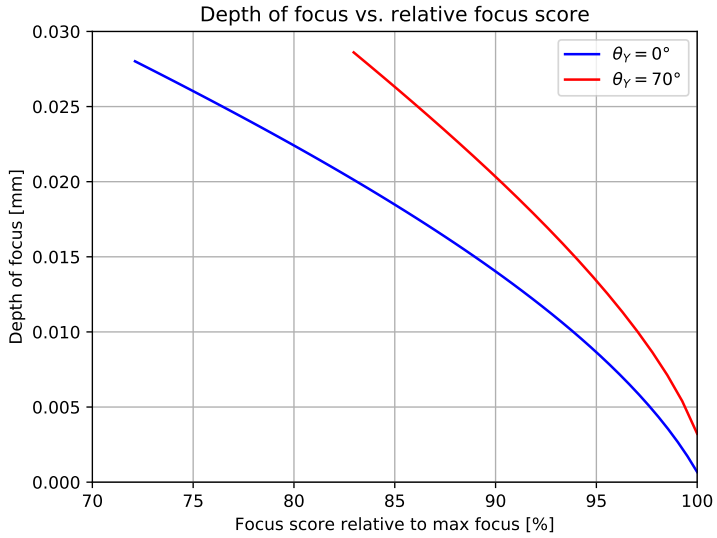


Figure 4.18: Depth of focus vs. relative focus score requirement

5 Discussion

The following sections discuss the results that were presented in the previous section. With regards to the hardware and software, the discussion is mainly focused around a discussion about the user friendliness and suggested improvements.

The discussion on the analysis of the Koi camera and the results from the focus data collection goes over the validity and confidence in the result and what can be done to improve on this. Additionally, arguments for further data collection is made in order to better answer the questions posed at the beginning of this thesis.

5.1 Hardware

The new hardware installed in Modus has mostly worked well. There is however some concern as to the attachment between the sensor fixture and the sensor stack. Sometimes after a scan program has completed, the sensor has moved out of alignment, indicating that it has moved somehow. The part on the top of the sensor stack is 3D-printed and the it's possible that the tolerances are not good enough to keep the sensor fixture in place as it's rotated and moved up and down during a scan program. A new version of the 3D-printed will be manufactured, this time CNC-milled aluminium, which should improve the tolerance.

The limitations of the rotation of the collimator about the y-axis was discussed in section 1.1.3. Limited to $\theta_Y = \pm 70°$, the maximum FoV that can be measured is 140°. However, Axis cameras with fish-eye lenses have FoVs of more than 180°. If Modus were to be redesigned, it would be a top priority to be able to scan FoVs up to and perhaps beyond 180°.

The electrical panel that was created for the new hardware is a significant improvement on the old one in terms of cable management and efficiency of the use of space. However, in hindsight it would have been better to split the electrical panel into two. As of now, there is hardly any space available on the panel for more components and the space between the cabling channel and terminal blocks is too small. This makes it difficult to feed wires through the cabling channel and into the desired terminal

block. There should simply be more room for your hands when it's necessary to do work on the wiring. It's especially difficult with the electrical panel installed in the machine. It's placed on the underside of the machine and all the way in the back, requiring one to sit on one's knees and reach in the back. It's an uncomfortable experience to service the electrical panel when it's installed in the machine, and it would be highly prioritized to move it into a more easily accessible position.

5.2 Software development

5.2.1 Testability

Section 3.2.2.3 described the principle of writing testable software and the idea of having a simulated version of the machine for the development cycle. Unfortunately, a simulated machine was never developed for the Modus software, mainly due to time constraints and the fact that throughout the project the real Modus machine was available to test on. Due to the lack of a simulated Modus machine, it's not possible to test new features or bug fixes remotely. Therefore, even though the hardware is installed and working, it's still relevant to develop a simulated machine to have for future development and debugging.

5.2.2 Software structure

Section 4.2.2 described how the software for Modus was divided into a number of modules. In hindsight, the ModusWorker module could benefit from being split into two separate modules. One module should be responsible for loading the camera's configuration settings and connecting it to the GMC through the CameraInterface-module. This module should also contain the calls to methods for capturing the image and calculating the MTF focus score. This way, a camera could essentially be loaded into Modus and images be captured without having to connect and set up the entire machine. While perhaps not something that would be necessary to do on a daily basis, it would be a more logical abstraction of the software, instead of having the ModusWorker module be responsible for everything. The second Module would then contain the functionality which requires both the Modus machine and the camera to be set up, for example calibration, alignment and running focus scan jobs.

5.2.3 Future development of the Modus software

To improve the user experience a number of new software features could be developed. One of the more significant would be a way to interact with Modus remotely while it's running a focus scan of a camera. A focus scan can take a long time if a large number of measurement points are provided. If a server is set up on the computer that's running the Modus software, the user could connect to the server remotely and request information about the status of the focus scan, view the preliminary results and be alerted if something went wrong and the scan fails. Even better would be if the server would allow the user to remotely send control commands to create new scans, start the alignment procedure and cancel at any time. Alternatively, the computer that's controlling Modus could be connected through with a remote desktop environment.

At the moment the software supports focus scans of the four parameters θ_y, θ_Z, defocus and back-focus. There is no functionality for controlling motors on a varifocal lens, i.e. the aperture, zoom or focus motors. While this was not within the scope of this thesis, it would be the next step in development of the machine to support also varifocal lenses, since these type of cameras constitute a large portion of Axis' product catalog.

Focus scanning in Modus is currently quite limited to doing sweeps of parameters between a start and and end point, with a fixed step size. It would also be useful to have some smarter algorithms which search for a given focus. For instance, if the user would like to find between what back-focus the focus score is above 0.18, Modus could start at some back-focus which is known to be larger than the result, and then decrease the back-focus with relatively large step size. By analyzing when the desired focus is approaching, the algorithm would adapt to a smaller step size and finish when the focus score is within some tolerance, for example 0.01. The back-focus distance would then be reported. This procedure would then be repeated for the second back-focus that satisfies the focus score 0.18 $\pm$0.01, c.f. figure 4.16.

5.3 Graphical user interface

5.3.1 User experience improvement

The GUI is a significant improvement on the user experience of Modus. Previously, the user would run Modus from an executable without any visual feedback apart from the logging output. The machine settings, product configuration and all focus scan

positions were imported upon program launch and then the scan proceeded. If the scan failed for some reason, the entire program would exit and the user would have to start-over from scratch, connecting again to the machine, initializing all motors and the collimator and loading the camera.

With the new structure of the source code, errors upon initialization, calibration, alignment, focus scans or any other procedure can be caught and the user can alter settings interactively in the GUI and retry immediately. There is also constant feedback about the motors' and collimator's status and position, as well as a live view which always displays the latest image of the cross-hair target, the focus score and the target's position on the sensor.

If the source code for the GUI was to be rewritten, it could have been appropriate to adopt a design pattern which better separates the source code controlling the GUI, i.e. callback-functions for button clicks and such, from the business logic of the program. This could be achieved by implementing the Model-View-View Model (MVVM) design pattern. The Model component in this case contains the business logic of the program, the View-component contains the XAML-design code and the View Model facilitates the interaction between the two primarily through the use of WPF-bindings. The improvement would be that the code for the GUI and the code for the Modus functionality are completely separated and not dependent on each other.

5.4 Data collection

Section 4.4.1 presented the results for the focus scans performed on the Koi camera. The data showed the focus score decreasing with increasing θ_Y, which can be attributed to the optical aberration astigmatism that was presented in section 2.3. It's also noted that the focus score is highest at the same back-focus for every θ_Y. This shows that the field-curvature aberration of the camera is small, not being measurable for the back-focus step size of 5 μm.

The back-focus tolerance for different focus scores was extracted from the data and presented in table 4.3. A discussion on the validity and confidence in these results is in order. The data was fitted with a third order polynomial. A third order polynomial has two critical points, a local maximum and a local minimum, and everywhere else it's monotonic, i.e. tending towards $\pm\infty$. This is not how the relationship between the focus score and the back-focus behaves in reality. The relationship is unimodal, with a single global maximum. Thus, it is important to realize that the polynomial fitted to the data may only be used for interpolation and not extrapolation.

The results show the sensitivity of the Koi camera. Within just 5 μm back-focus the top focus score shifts from infinity to the near-point of 300 mm. However the depth of field is very large. By offsetting the back-focus by μm the top focus score may have moved to the near-point, but table 4.2 showed that the focus score at infinity and $\theta_Y = 0°$ only decreased by 3.7 %. Likewise, at $\theta_Y = 70°$, the focus score decrease of a target at infinity when offsetting the back-focus by 5 μm was 3.4 %. The focus score for an object at the near-point increased by 4.3 % and 3.1 % at $\theta_Y = 0, 70°$, respectively, for the same 5 μm back-focus offset.

Since the near-point and infinity represent the two extreme distances of the object space, and the focus score within this range changes on the order of $\pm 4\%$, it can be concluded that the focus score for an object at any position between the near-point and infinity would be within this same range. Thus, for the measured Koi unit, the relevant back-focus offset is between 0 and 5 μm and the focus score changes to be expected is below 5 % throughout the object space, between 0 and 70 $^\circ$ θ_Y.

The *ideal* back-focus offset is somewhere between 0 and 5 μm. However, the ideal offset depends on criteria that must be specified by the project developing the camera. The results here have not taken into consideration the application areas of the camera. For surveillance in a convenience store for example, the range of object distances to be imaged sharply might lie between 0.5 and 25 m. In that case, focus data should be gathered at these object distances instead of at 0.3 m and infinity. Because of the large depth of field of fish-eye lens cameras, the focus score results at these distances would be very similar to the results presented here for 0.3 m and infinity.

While the results provide information about the back-focus tolerance at different focus score requirements, it only holds true for the particular unit that was analyzed. In order to be able to draw conclusions about the Koi camera in general, several more units would have to be analyzed. With a larger sample size, the variance between units of the same camera could be shown statistically by calculating the mean, standard deviation and confidence intervals. With this information, the foundation would be stronger for making decisions about whether to offset the sensor from the alignment position in order to increase the focus score at object positions closer than infinity.

5.5 Future role of Modus

5.5.1 Development phase

The preceding results and discussion show that Modus can be used during the development phase of new cameras at Axis. At the moment it is limited to fixed-focal cameras. Using Modus, the focus score can be measured as a function of the object distance, back-focus, object angle and rotational angle.

By analyzing several units of a camera in Modus, the relationship between the back-focus and focus can be statistically proven. This information can form the foundation for decisions on whether to offset the back-focus when gluing the sensor and lens in IBAS 2 in order to improve the focus for objects closer than infinity.

5.5.2 Production phase

Modus could also be used out in production. After the sensor and lens have been glued in IBAS 2, the rest of the camera is assembled and several tests are performed. Among these are focus tests. Because Modus is fitted with a variable focus collimator, a unit which fails these subsequent focus tests could be disassembled and analyzed in Modus at, for instance infinity and the near-point. Using a graph such as figure 4.15 previously shown, the back-focus offset from the alignment position could be determined. By measuring this at the five positions on the sensor, the tilt offset could also be calculated. This could provide details about how the UV-curing adhesive has misaligned the sensor, something which today is not known.

Two approaches to measuring the misalignment of failed units in production can be envisioned. The first is to, as the previous paragraph explained, compare the measurements from Modus of a failed unit with that of measurements of non-glued units of the same camera in Modus. However, since it is expected that there is some variance between units, it is currently not known how accurate this misalignment prediction could be. Instead, the focus score saved during the alignment in IBAS 2 for each unit could be used. If the unit then fails, the measured focus score in Modus could be directly compared to the focus scores measured in IBAS 2 during the alignment. The calculated misalignment would then be based only on the specific unit of the camera, and not an average of a number of units.

The above ideas are hypothetical, and would require significant testing to validate. A number of different cameras should be measured in the IBAS 2 located at the office in Lund and then in Modus to see if the results are comparable.

6 Conclusion

This thesis set out with the goal of developing a tool for analyzing the focus characteristics of Axis cameras prior to sensor-lens assembly. The aim was to expand on the capabilities of the Modus machine to also be able to find the relationship between the MTF focus score and the back-focus, defocus and object angle. With a more complete picture of the focus characteristics it should be possible to make informed choices about what back-focus to aim for during production in order to achieve the desired depth of field.

Modus has been updated with new mechanical parts and electronics to support the analysis of a camera where the lens and sensor are not glued together. Several new software features have been implemented that can scan the back-focus, defocus and object angle and calculate the focus score at each combination of the parameters. A graphical user interface was developed to interact with the machine, allowing anyone to use it with ease.

Koi, a fish-eye lens camera was analyzed as part of the development of Modus. The results from the focus scan of Koi provided a mapping of the focus score within a certain range of the sweep parameters. Some key plots and conclusions could be drawn from this set of focus data:

1. The relationship between the back-focus tolerance and the focus score
2. The back-focus offset required to alter the focus score at object distances other than infinity
3. The misalignment of a glued unit by analyzing it in Modus at several different object distances

While the results answers the questions posed at the beginning of the thesis, they are limited to the specific unit of Koi that was analyzed. As discussed in the previous chapter, it is necessary to analyze a number of units in order to find out how (1) and (2) above varies between units.

Future development should focus on developing Modus capabilities to analyze varifocal lenses. The GUI should be made more responsive and user friendly. Smarter focus scanning algorithms should be implemented to cut down on scan durations.

Finally, Modus role in the production line should be studied further, since it can potentially provide further information about why units that were successfully aligned in IBAS 2 fail subsequent focus tests.

References

[1] Göran Jönsson. *Våglära och optik*. Lund, Sweden: Teach Support, 2013.

[2] *Thin Lenses*. URL: https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_University_Physics_(OpenStax)/Book%3A_University_Physics_III_-_Optics_and_Modern_Physics_(OpenStax)/02%3A_Geometric_Optics_and_Image_Formation/2.05%3A_Thin_Lenses. Retrieved 2022-02-12.

[3] *CamTest ColMot - Optical target projectors*. URL: https://trioptics.com/wp-content/uploads/2021/01/CamTest_ColMot2.0_Optical_Target_Projectors.pdf. Retrieved 2022-02-12.

[4] Warren J. Smith. *Modern Optical Engineering*. New York, USA: McGraw-Hill, 2000.

[5] *Introduction to Modulation Transfer Function*. URL: https://www.edmundoptics.eu/knowledge-center/application-notes/optics/introduction-to-modulation-transfer-function/. Retrieved 2020-02-18.

[6] H.H. Nasse. *How to read MTF curves*. 2008. URL: https://lenspire.zeiss.com/photo/app/uploads/2018/04/Article-MTF-2008-EN.pdf. Retrieved: 2020-02-18.

[7] *ST Series Two Phase DC Stepper Motor Drives*. URL: https://www.moonsindustries.com/series/st-series-two-phase-dc-stepper-motor-drives-b01020201. Retrieved 2020-02-13.

[8] *Git - about Git*. URL: https://git-scm.com/about. Retrieved 2020-02-06.

[9] *Gerrit - About Gerrit*. URL: https://www.gerritcodereview.com/about.html. Retrieved 2020-02-06.

[10] Munish Chandel. *What are four basic pinciples of Object Oriented Programming*. URL: https://medium.com/@cancerian0684/what-are-four-basic-principles-of-object-oriented-programming-645af8b43727. Retrieved 2020-02-06.

[11] Arvind Singh Baghel. *Software design principles DRY and KISS*. URL: https://dzone.com/articles/software-design-principles-dry-and-kiss. Retrieved 2020-02-07.

[12] *Unit testing best practices with .NET Core and .NET Standard*. URL: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices. Retrieved 2020-02-13.

[13] *Think you understand Single Responsibility Principle?* URL: https://hackernoon.com/you-dont-understand-the-single-responsibility-principle-abfdd005b137. Retrieved 2020-02-14.

[14] *Difference between Winforms vs WPF*. URL: https://www.educba.com/winforms-vs-wpf/. Retrieved 2020-01-21.

[15] *WPF vs. WinForms*. URL: https://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/. Retrieved 2020-01-21.

[16] *Axis Fixed Dome cameras*. URL: https://www.axis.com/sv-se/products/fixed-dome-cameras. Retrieved 2020-02-18.

A Time plan

Figure A.1 and A.2 illustrates the original time plan and the actual time plan. The actual time plan differs from the planned in some areas. Firstly, the tasks were broken down into smaller tasks, and some tasks omitted all together. Some tasks took less time than expected. For example the data processing- and analysis task was planned for four weeks, but in reality the effective time spent was roughly two weeks spread out over a number of weeks.

In contrast, some tasks had not been considered by the time the original time plan was written. For instance, implementation of alignment for Modus was not something that was anticipated during the planning phase, taking approximately two weeks to complete. Likewise, the need for the graphical user interface emerged as the project ensued and required one to two weeks of development.

There was some delay in the manufacturing of the new hardware for Modus, which is the reason for the extension of the thesis into February. This delay meant that installation & testing of the new hardware had to be postponed a few weeks, and as a result also the product analysis part of the project.

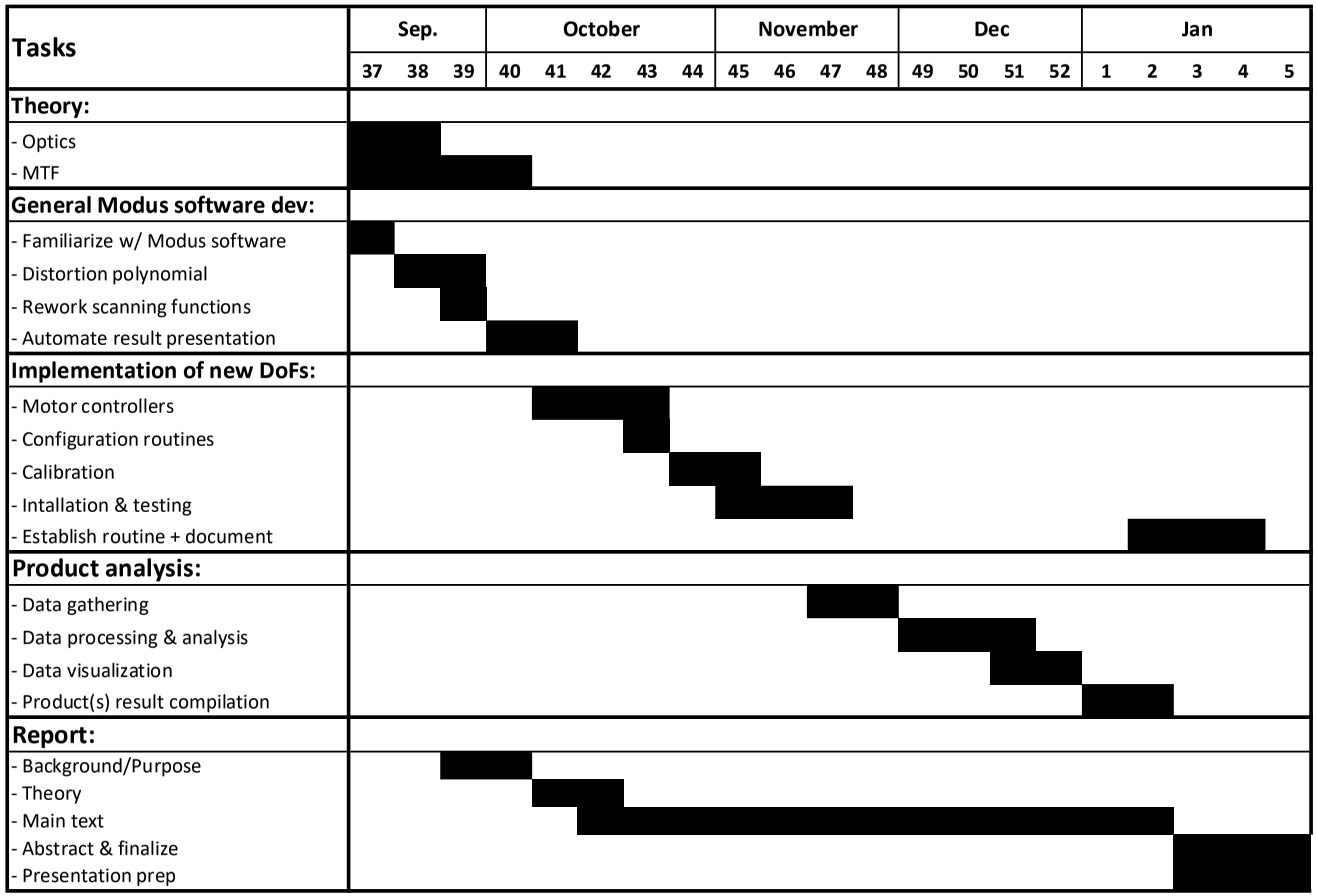


Figure A.1: Project time plan

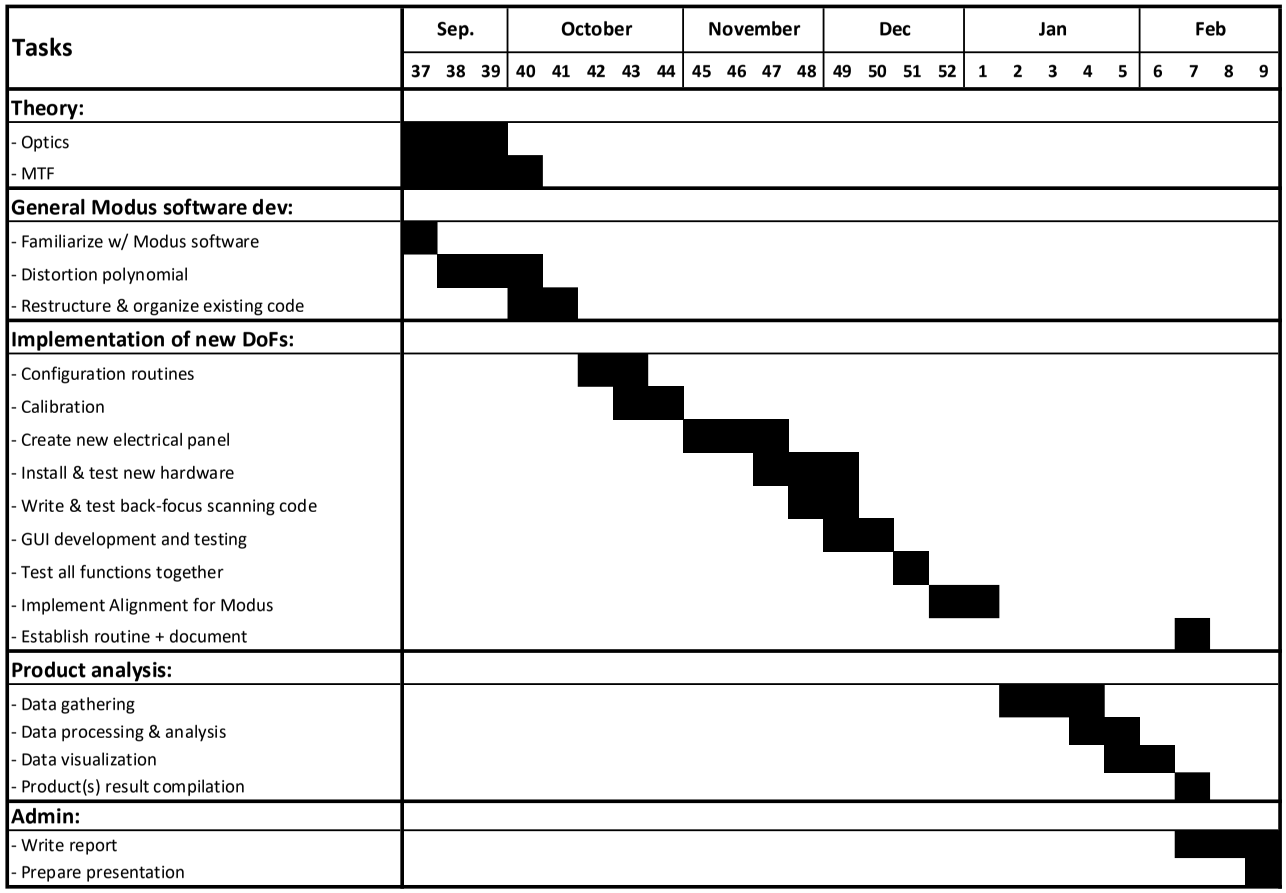


Figure A.2: Actual time plan