Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

BAP: Video on SEM
SEM control - Subgroup report

Bachelor Graduation Project (EE3L11)

Supervisor: Dr. ir. S. Vollebregt

Assisting Supervisor: Dr. Y. Zhang

Authors:
Tijmen ten Berge
Matthijs Spijker

January 28, 2026

Abstract

This report presents the design and implementation of an automated image-acquisition module for the FEI XL30 SFEG Scanning Electron Microscope (SEM). Older SEM models such as the XL30 lack built-in functionality for long-duration, unattended imaging, limiting researchers to manually capturing frames or taking only sparse images across an experiment. To address this limitation, a Python-based control subsystem was developed that communicates with the SEM via its RS232 serial interface and coordinates with an image-processing subsystem. Since samples may drift at high magnification levels during long-duration imaging, a drift correction feedback loop was implemented. The loop ensures that the region of interest is always kept in the centre of the frame, by both software stabilization and beam shift correction. A custom message protocol, robust error-handling framework, and graphical user interface were implemented to ensure reliable operation within the constraints of a legacy Windows 2000/Windows 7 environment. The resulting system enables stable, long-duration SEM video acquisition without the need for user supervision.

Preface

This report presents the results of the Bachelor Graduation Project (BAP) for the Electrical Engineering program at Delft University of Technology. The project was carried out within the “Video on SEM” BAP group and focuses on the development of an automated image-acquisition subsystem for the FEI XL30 SFEG Scanning Electron Microscope. Over the past weeks, we designed, implemented, and tested software capable of controlling the microscope, coordinating with the image processing subgroup, and enabling long-duration SEM video recording.

We would like to thank our supervisor, Dr. ir. S. Vollebregt, for his guidance, valuable insights, and continuous support throughout the project. We also thank our assisting supervisor, Dr. Y. Zhang, for making her test samples available for us to use.

We are grateful to the staff of the Else Kooi Laboratory for facilitating access to the XL30 SEM and allowing us testing time on the hardware. Finally, we would like to thank our fellow students and the members of the image processing subgroup for their collaboration and constructive discussions.

We hope that the results of this project contribute to more efficient long-term SEM studies within the faculty and form a solid foundation for further development.

Contents

1 Introduction

1.1. Scanning Electron Microscopy

SEMs, or Scanning Electron Microscopes, are extremely versatile tools that are used by researchers in all kinds of fields. From cell research in biology or mineral research in geology to material science and semiconductor research. Here, at the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), one such instrument is the FEI XL30 SFEG SEM. A picture of the setup in EEMCS is shown in figure 1.1. This SEM is located next to the ground-floor hallway of the low-rise, near the entrance to the Else Kooi Laboratory.

Figure 1.1: The scanning electron microscope at EEMCS

Unlike optical cameras, a SEM does not capture an image in a single instant. Instead, the microscope forms an image by scanning a focused electron beam across a sample horizontally, line by line [1][2]. At every pixel position, the detector measures a signal intensity produced by electron–sample interactions, and the full image is shown only after the entire scan has completed. High-quality SEM images therefore require relatively long scanning times per line to ensure sufficient signal-to-noise ratio. If the scanning time is reduced in order to produce frames at real-time video rates, the detector collects far fewer electrons, resulting in images with substantially more noise, reduced contrast, and visible artefacts. For this reason, true high-resolution, low-noise SEM video cannot be produced at live-video frame rates, but instead, individual high-quality frames must be acquired at slower intervals and combined into a video afterwards.

Despite their widespread use, electron microscopes are very large, expensive, and complex machines. As a consequence, older models are still widely used. The XL30, for example, is almost 25 years old. While these older microscopes may lack some of the advanced detectors or more modern automation options, these older models generally still offer adequate image quality for a lot of the research that they are used for.

One of the automation features that is absent on the XL30 is the functionality to automatically capture images at fixed time intervals. At the present moment, researchers using an XL-series SEM have limited options for detailed analysis of a sample over an extended period. They can either manually capture numerous images over the course of several hours, or take a single image at the beginning and another at the end of the experiment. Both approaches have drawbacks: the first is time-consuming and inaccurate, while the second

sacrifices detail of analysis or any occurrences between the two captured frames. As a result, neither method makes effective use of the researcher's valuable time. Our goal is to create a module that fully automates this workflow, from automatic image acquisition, to the production of a high-quality video. This will be done by providing the microscope with a set of instructions to generate multiple images at a predetermined time interval, after which the images are compiled into a single video file.

Because the sample will be recorded for several hours, and the experiment operates on the nanoscale, sample drift will likely occur. Over long periods, the sample can shift by hundreds of nanometres due to thermal fluctuations, charge build-up, or even vibrations from people walking near the microscope. Since something as simple as nearby foot traffic can already cause noticeable drift, it is reasonable to expect that the introduction of trams on the Mekelweg could contribute additional disturbance. To manage this, a two-tier drift-correction strategy is used. Small displacements are handled in software through frame-to-frame image stabilization, while larger shifts are corrected by sending beam-shift commands to the SEM control module. Without this correction strategy, prolonged sample drift could cause the region of interest to completely shift out of frame which would make the entire process of long time imaging completely ineffective.

Another challenge is the legacy computing environment on which the XL30 operates. The microscope is controlled by a Windows 2000 PC, while a separate Windows 7 machine is available for other equipment such as the EDAX detector. These two systems communicate via a serial connection, making modern integration and automation far from straightforward.

These limitations drive the need for the development of a system capable of automating image acquisition, correcting sample drift, and compiling the resulting images into a coherent video. Designing such a system that enables long-duration SEM experiments without constant supervision is the focus of the Bachelor Graduation Project of which this thesis is a part. The details of how this project is divided into subsystems will be clarified in a subsequent section.

1.2. Subdivision into subsystems

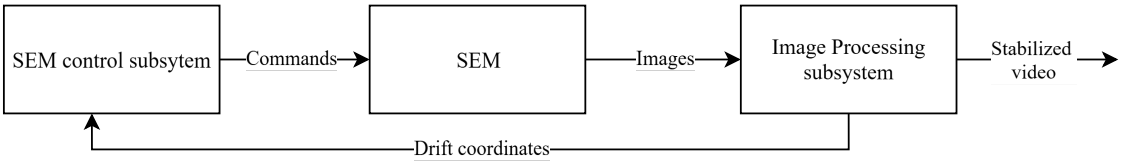


Figure 1.2: Block diagram overview of the system

In this project, the work is divided into two cooperating subsystems: the SEM control subsystem and the image processing subsystem. Together, they create an automated workflow for long-duration SEM imaging, as shown in Figure 1.2.

The SEM control subsystem communicates with the XL30 microscope through its serial interface. It sends all commands required for automated image acquisition, such as triggering frames at fixed intervals, and applies beam-shift corrections when drift is detected.

The image processing subsystem receives each image produced by the SEM and performs drift estimation, confidence assessment, and frame stabilization. It then compiles the stabilized frames into a high-quality video. When a significant shift is detected, the subsystem returns drift coordinates to the SEM control subsystem, forming a feedback loop that keeps the region of interest centered during long experiments.

This division allows both subgroups to work independently on their respective tasks while maintaining a unified system capable of automated, long-duration SEM imaging.

This report will dive into the workings of the SEM control subsystem.

1.3. State of the art

With the current state of the technology, all modern SEMs are able to make videos instead of only making pictures. However this project makes use of the XL30 SEM which is a model from the early 2000s. These models are not able to capture video, they can only make single images. There is a possibility for software expansion packs for these old SEM models. However, these expansion packs come with a lot more software than just video making capabilities which makes them expensive, so rarely worth it for the purpose of making SEM videos.

1.4. Outline of the report

This report outlines the design process of creating an automated SEM image acquisition loop with drift correction capabilities.

Chapter two defines the program of requirements, outlining the assumptions, mandatory constraints, and design trade-offs that shaped the development of the SEM control subsystem.

In chapter 3 the design of the subsystem will be discussed. First described is the communication with the SEM. Specifically the used message format, communication protocol and Python implementation will be addressed. Then, the communication structure with the image processing subsystem will be described. The development of a custom communication class is the focus of the section Next, the report delves into the design of a custom User Interface. After that, these elements are all combined into an automated image acquisition loop, both with and without drift correction functionality.

Chapter 4 describes the testing approach and presents the results, discussing which requirements were successfully met and how encountered issues were resolved.

Chapter 5 summarizes the project's conclusions and chapter 6 discusses limitations, possible improvements, and future extensions.

2

Program of requirements

To evaluate this project the following requirements were set. These are divided into requirements for the entire system and requirements for the SEM control subsystem.

2.1. Requirements of the entire system

2.1.1. Assumptions

- The microscope is controlled by a Windows 2000 PC, while a separate Windows 7 machine is available with serial connection.
- There is a beam shift limit of ± 20 μm in both the x and y directions [3].
- The resolution of the beam is 1.5 nm at 10 kV or higher and 2.5 nm at 1 kV.

2.1.2. Mandatory requirements

- The final product must not result in the need to reconfigure the cables at the back of the setup.
- The product must be designed using one or more pieces of software.
- The software must be developed to work on Windows 7.
- The implementation must not require additional software to be installed on the Windows 2000 PC.
- The system must yield a stable video output:
  - In the output video, the region of interest must not shift more than 2.5% of the field of view in the x- and y-direction.
- The system shall generate an output video file in either Standard Definition (712 × 484 px) or High Definition (1424 × 968 px), encoded in either MP4 or MKV format.
- The order of the frames of the output video must be equal to the order in which the frames are captured on the SEM.
- Both subsystems (SEM control & Image Processing) must be able to run separately as well as combined.
- The user must be able to specify a desired output FPS, the time interval between two image captures and the total number of frames of the output video.
- The processing time of the image processing software must not influence the timing between the capture of two images
- The software should be able to perform drift correction both physically and digitally. Digitally up to the given threshold and physically up to the beam shift limits described in 3.4.3.

- The user specified beam shift threshold must be no larger than one third of the field of view in x- and y-direction.
- The software won't support real time video recording from the SEM.
- When the sample has moved completely out of frame in one step, the software will not do any searching to catch such extreme movements.

2.1.3. Trade off requirements

- The system should interfere minimally with the existing hardware and software.
- The system should be implemented without adding extra hardware to the setup.
- The image processing software should prioritize sending a beam shift request to the control software to limit the time control has to wait for a potential shift

2.2. Requirements of the SEM control subsystem

2.2.1. Mandatory requirements

- The subsystem must be able to send commands to the SEM.
- The subsystem must be able to read parameters from the SEM.
- The subsystem must be able to detect when the communication with the SEM has failed, after which the entire system must shut down automatically.
- The subsystem must include an automated image acquisition loop.
- The subsystem must allow the user to configure the output video parameters:
  - Playback frame rate in frames per second (fps).
  - Time interval between two captured images in seconds.
  - Total number of frames in the final video.
  - The outpainting method of the video edges (`black, white, same, mean, mean_outpaint`)
  - The video format and name of the video.
- The subsystem must allow the user to configure the SEM parameters:
  - High tension voltage
  - Spot size
  - Lines per frame.
  - Line scan time [ms].
- The subsystem must be able to calculate the image acquisition time from the user set image parameters.
- The automatic operation of the SEM must not alter the image focus.
- The subsystem must be able to store all acquired images as .TIF files in a designated directory, accessible for the Image Processing subsystem.
- The subsystem must ensure that there are no more than two files at a time in the shared drive.
- The subsystem must be able to communicate with the Image Processing subsystem within 1 second.
- The subsystem must include shift correction functionality, based on the request of the Image Processing subsystem. The sending and processing of this request must not influence the time delay between two image captures.

3

Design process and choices made

This chapter outlines the design and decisions made during the development of the automated image acquisition system of the XL30 SEM. First the serial communication with the SEM is discussed, broken down into the data and message structure, and justifying the choice of a Python-based user interface. Then, the communication protocol between the control software and the Image Processing subsystem is described, followed by the rationale behind the selected user-interface approach. Finally, the image acquisition loop, including initialization, capture routines, file handling, and drift-correction logic, is presented. These design choices collectively form the foundation for a robust and maintainable automation framework. In figure 3.1 a block diagram of the SEM control subsystem and its connection to the Image Processing subsystem can be seen.

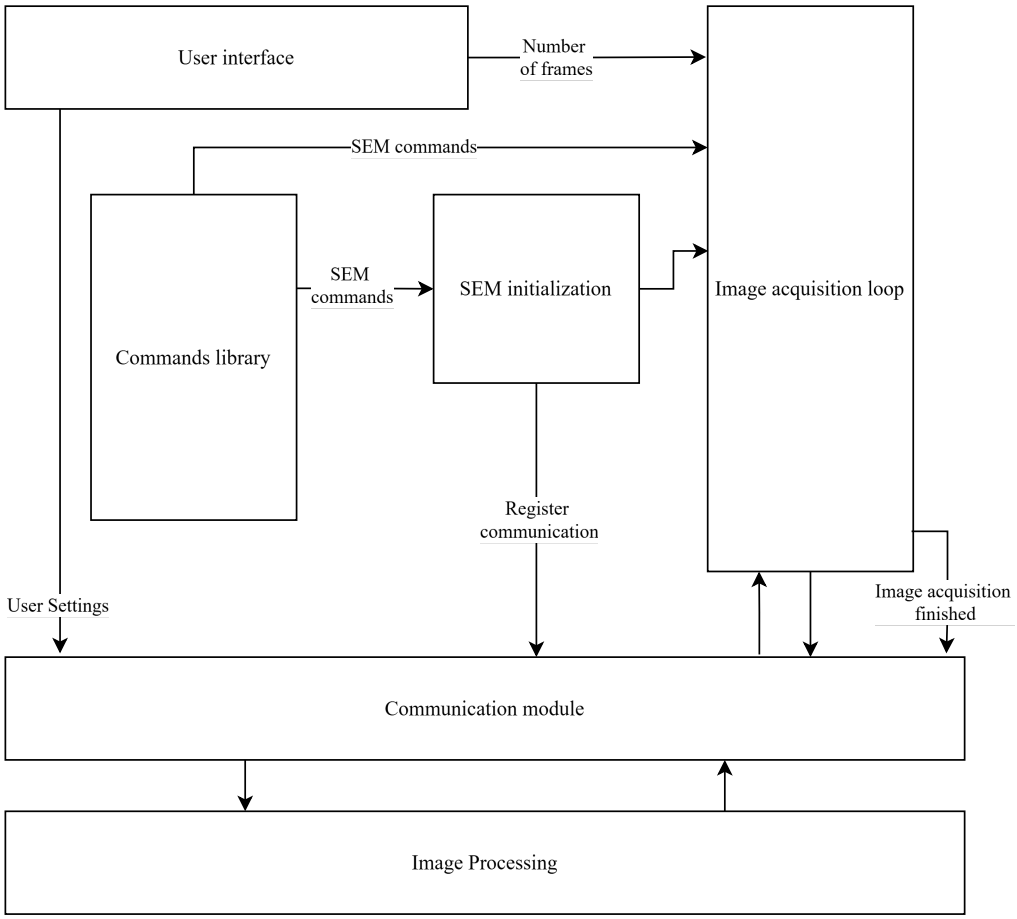


Figure 3.1: Block diagram of the SEM Control subsytem.

3.1. Communication with the SEM

3.1.1. Message structure

The SEM can be controlled by an external computer through a RS232C[4] serial port [5]. The commands sent by this external computer are handled by the Serial Control Server (SCS) module of the SEM software. This software supports both single block and multiblock messages. For our application, the choice was made to implement the single block communication format, since they are much easier to implement and communication is more reliable. Moreover, the high communication speeds available when opting for the multiblock messages, are not required for our application.

The message format for single block messages of length `n+1` can be seen in figure 3.2.

ID	LENGTH	OPCODE	ERROR	DATA FIELD	CHECK-SUM
byte 0	byte 1	byte 2	byte 3	byte 4...n	n+1

Figure 3.2: Diagram of message format.

The value of the `ID` byte is always set to the hexadecimal value `0x05`. This byte is used by the SCS module to determine which protocol handler to invoke. Byte 1, the `LENGTH` byte, specifies the length of the outgoing message, and byte 2, `OPCODE`, determines which operation the SCS should execute. The opcodes are structured such that all odd values correspond to `write` commands, while all even values correspond to `data-request` commands. In byte 3, the `ERROR` byte, only the most significant bit (bit 7) is used. This bit acts as an error indicator and must always be set to `0` by the sender. If an error occurs, the SCS sets this bit to `1`; all remaining bits in this byte are unused. The `DATA FIELD` contains either the data to be delivered to the machine (when sending a request) or the data requested by the sender (when receiving a response). This field may contain integers, floating-point values, or error codes, depending on the operation. Finally, the `CHECKSUM` byte is used by both sides of the communication to validate the received message. If the computed checksum does not match the received value, the message is discarded and the error bit is set. The checksum is computed according to equation 3.1.

$$\texttt{checksum} = \left(\sum_{i=0}^{n} \text{byte}_i \right) \bmod 256. \tag{3.1}$$

3.1.2. Data representation

All data is stored in multiples of four bytes. These may contain integers, floating-point values, or error types, depending on the requested operation. See Appendix A for the encoding of these data types.

Integers: Integers are represented using two 2-byte values, following the Intel (little-endian) byte-ordering convention. This means that the least significant byte is transmitted first.

Floats: Floating-point values are encoded according to the IEEE 754 single-precision format [6]. In this representation, bit 31 stores the sign, bits 30–23 store the biased exponent, and bits 22–0 store the mantissa (fraction). In Python this is done using the `struct` module [7], which allows conversion from Python values to C structs, represented as `bytes` objects.

Error types: Error types are generated by the SCS whenever an error condition is detected. These values are of type `unsigned long` and are assigned according to the error definitions provided in the file `gloerr.h` (see Appendix C).

3.1.3. SCS Responses

Once a message is received by the SCS, several types of responses are possible.

Command successful: If a complete command message is received successfully and the checksum is valid, the SCS returns a copy of the original message. The reply therefore contains the same length, opcode, and data field as the request.

Return data: If a complete data–request message is received successfully and the checksum is valid, the SCS returns the original message with the data field replaced by the requested data.

Error: If a message triggers an error (for example, when supplied parameters are out of range or the checksum is invalid) the SCS sets the error bit to 1 and returns a reply in which the data field contains the corresponding error code.

No response: If a message is so malformed that the SCS cannot determine the appropriate error condition (for example, when the LENGTH byte is incorrect), the message is discarded and no reply is sent.

3.1.4. Python commands library

To implement communication over the serial interface, the Python package `PySerial` was used [8]. Because the software must run on a Windows 7 system, a legacy Python version compatible with this operating system was selected. The complete source code is provided in Appendix B.1. The `PySerial` module provides easy access to the computer's serial port and performs the required handshaking procedures during initialization. For compatibility with the SCS software, the serial interface is configured with a byte size of eight bits, one stop bit, and a baud rate of 9600 baud.

The Python code constructs messages according to the protocol described in Section 3.1.1. After each transmission, the software verifies whether the message was received correctly. Two cases are distinguished. For `write` commands, the SCS is expected to return a reply identical to the transmitted message, which serves as an acknowledgement. For `data request` commands, the software checks whether the response has the correct length, whether the error bit is cleared, and whether the checksum is valid. Only if these conditions are satisfied is the message considered successfully received, and can the received data be processed.

Using these principles a custom Python commands library container was constructed containing the commands necessary for automatic image acquisition.

3.1.5. Capturing an image

While most commands have an intuitive application, the opcode library does not include a Capture image function. Therefore, this functionality was implemented by altering the filter mode of the SEM. [9]

To acquire an image from the SEM software, the Python interface first configures the system's filter mode. The filter is set to `Average(1)`, which instructs the SEM to switch to slow-scan mode and to apply an averaging filter over a single frame. An averaging depth of one effectively disables noise reduction while still enabling the slow-scan acquisition mode required for image capture.

After the filter mode is set, the SEM begins the scan. Once the scan completes, the SEM automatically transitions the filter mode to `Freeze`. The Python code uses this behaviour as an indicator that the image has been fully acquired. During the capture process, the software repeatedly reads the current filter-mode status from the SEM. As long as the mode remains `Average`, the scan is still in progress. When the filter mode changes to `Freeze`, the software concludes that the acquisition is complete and that the image is stored in the XL image buffer, after which it can be stored in the output directory.

3.1.6. SEM Communication errors

It is possible that communication with the SEM fails for various reasons. For example, if the serial cable is not connected properly, no communication can take place. To ensure user-friendly and robust operation, the software must be able to detect such critical failures. For this purpose, a communication error safeguard was implemented. Each command in the Python command library returns `True` if the acknowledgement message received from the SCS is valid, otherwise it returns `False`. Initialization of the SEM (see section 3.4.1) proceeds only if the first command is acknowledged successfully. If this is not the case, the software retries the same command up to four additional times, this is based on the Stop and wait ARQ protocol [10]. If the acknowledgement remains invalid after the fifth attempt, the software shuts down by sending a `Done` message to the Image Processing subsystem (see section 3.1.7). The reason for retrying multiple times is to prevent the entire system from shutting down in response to a single incidental communication error, such as a transient bit flip [11] on the serial line. The Go-Back-N protocol was also considered to make sure all the messages would be received, but it was found to be too difficult to implement [12].

3.1.7. Alternative: C++ library

Besides the serial-communication approach described in the previous section, the XL software also provides an alternative based on a C++ library that can be executed directly on the Windows 2000 PC. The XL Interfacing Handbook [5] documents the full set of commands in the form of C++ functions, and this approach was initially considered. However, some practical limitations led to the decision not to pursue it.

Mainly, the SEM control software is designed for a 16-bit architecture. Although the SEM software provides thunking capabilities to allow 32-bit programs to interface with the 16-bit components [13], relying on this mechanism would restrict development and testing to legacy 32-bit systems. In practice, this would prevent the use of modern 64-bit computers for development. To address this limitation, the possibility of using a virtual Windows 2000 machine on a 64-bit host was explored.

However, this approach also proved infeasible. The required thunking modules depend on legacy Windows 2000 drivers that are no longer commercially available and are not supported by modern virtualization platforms. Moreover, no documentation or other resources could be found regarding these drivers. In addition, selecting Python as the implementation language ensures that the final software remains easily maintainable and adaptable to new use cases. As a result, the C++ library was deemed impractical, and communication over the serial connection was selected as the more robust and maintainable solution.

3.1.8. Alternative: External microcontroller

An alterative that was explored in the early stages of the project was to implement the software on an external microcontroller, such as a Raspberry Pi. In this approach, the microcontroller would communicate with the Windows 2000 PC via the serial line, and transmit the final video output (and, if required, the captured images) to the user's laptop via a WiFi connection. However, this solution introduced several impracticalities. First of all, adding extra hardware to the setup would further increase the system complexity of an already complex setup. Moreover, this implementation would have no meaningful technical advantage over controlling the SEM via the Windows 7 PC, as it relies on the same serial line technology. Consequently, the microcontroller-based solution was abandoned early in the design process.

3.2. Communication with the Image Processing subsystem

To support the feedback loop between the SEM control subsystem and the Image Processing subsystem, a custom communication structure was implemented. This communication is handled by an intermediate communication module, which routes messages between both subsystems. A block diagram of the architecture is shown in Fig. 3.3. The full Python implementation can be seen in Appendix B.4.

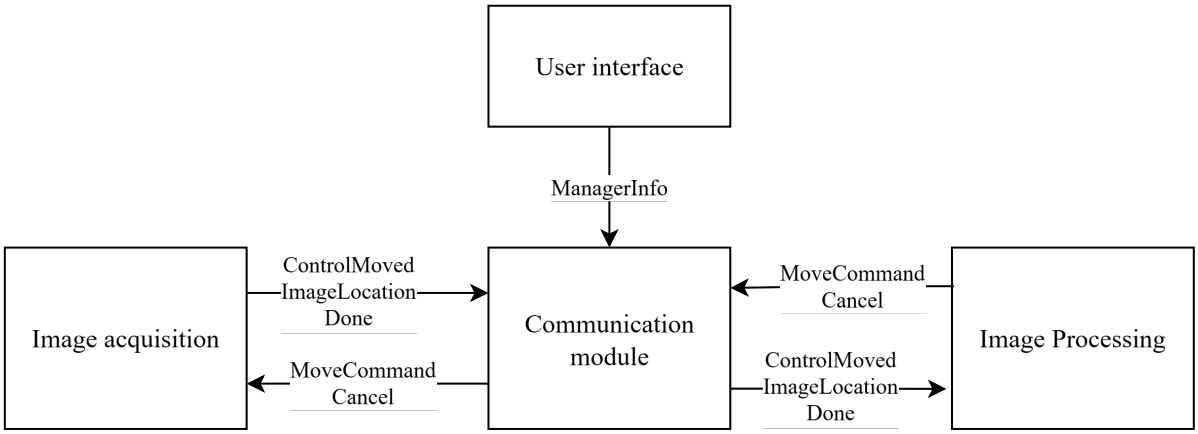


Figure 3.3: Block diagram of the communication module

3.2.1. Initialization of the communication

Before any data exchange can occur, both subsystems must register themselves with the communication module. The initialization proceeds as follows:

1. Each subsystem connects to the communication module using a predefined identifier. The SEM control subsystem connects under the name `CONTROL`, and the Image Processing subsystem connects under `ANALYSIS`. Attempts to connect using any other identifier result in an error. The protocol only continues once both subsystems are connected.

2. After successful registration, the `CONTROL` subsystem sends a `ManagerInfo` message. This message contains the user-defined video parameters: the output directory for the final video, the playback frame

rate, and the time interval between two image captures. Additionally, the message includes the parameter `MaxDriftPercentage`, which specifies the maximum allowed sample drift (expressed as a percentage of the image resolution). Drift above this threshold triggers a hardware beam shift on the SEM instead of a software correction.

3.2.2. Messages

A message consists of three fields: the message type, the message data, and the intended receiver. The available message types are the following:

- `ManagerInfo`: Initialization message sent by `CONTROL` to `ANALYSIS`. The data field contains a list of strings encoding the video and acquisition parameters (see Section 3.2.1).
- `ImageLocation`: Sent by `CONTROL` to `ANALYSIS`. The message contains the full file path of the most recently captured image.
- `MoveCommand`: Sent by `ANALYSIS` to `CONTROL`. It requests a beam shift and contains two floating-point values representing the desired beam shift in millimetres.
- `ControlMoved`: Sent by `CONTROL` to `ANALYSIS` to acknowledge that a beam shift has been executed successfully.
- `Cancel`: Sent by `ANALYSIS` when a critical error occurs that prevents further processing. Upon receipt of this message, the `CONTROL` subsystem terminates image acquisition and sends a `Done` message to `ANALYSIS`.
- `Done`: Sent by `CONTROL` after image acquisition has completed or when a critical error occurs. It signals to `ANALYSIS` that no further images will be produced and that the output video can be assembled.

3.2.3. Communication protocol

To ensure that the receiver reads the correct message, the following protocol was implemented:

1. The sender sends a message, which is stored in the so-called `queue`.
2. The receiver checks if a message is available, by checking if `len(queue) > 0` and if the identifier of the message corresponds with its own identifier (i.e. if the message is meant for the receiver). If this is the case, the protocol may proceed to the next stop. Otherwise, the message is ignored.
3. The receiver reads the message by checking the message type and interpreting the accompanying data field.
4. After successful processing, the receiver removes the message from the queue, to prevent it from being handled again.

3.3. User interface (UI)

3.3.1. Command line UI

When designing the user interface, a command line only user interface was initially considered. So for each setting a command line prompt would be asked and the user would need to fill it in. There are two reasons why this wasn't done. The main reason is that a command line only can be quite unintuitive for the user. When testing the messages a program named DOSbox was used to open a DOS testing program included with the XL software (see Section 4.1). This program had a command line only user interface and was quite unintuitive and sometimes infuriating to use. The second reason is that a command line only user interface can go wrong more easily. All the settings that are used with the SEM are predetermined from a set list. So, when a user types something else, the SEM will send out an error. This is why a graphical user interface was chosen. The drop-down menus make sure the user is not able to fill in wrong settings [14], and the buttons make it very clear what can be done with the software.

3.3.2. Graphical User Interface (GUI) in Python

As was mentioned in section 3.1.7, the XL software also provides C++ code. This code includes a shell of a graphical user interface. As was also stated in section 3.1.7 this code turned out to be infeasible due to the requirement of legacy Windows 2000 drivers. However, this was not the only reason for choosing Python over C++. Python is a more well known programming language. This means that if people want to change the code for the User Interface, it is easier when Python is used over C++. This is also the reason Tkinter was used over other UI Python libraries. Tkinter is already included with every Python install [15]. Therefore, there is a lot of documentation on Tkinter making it easier for the users to change the code. The last reason why Python was chosen for the user interface, is that all the other modules are programmed in Python. A user interface made in Python thus makes for smooth integration between modules. The full python implementation of the UI can be seen in Appendix B.2.

3.3.3. Error handling

Throughout the system, errors can occur. These can be errors from the SEM [5], errors from the SEM Control module, or errors from the Image Processing module. When these errors occur, these error need to be shown to the user. This is done by returning an error when a function where errors can occur finishes. When no errors occur, the error will be an empty string. When the function return an empty string, the software knows no error has occurred. However, when the system fails an error message is set. When this error message is read by the interface, a pop-up window will be created that will show the error and the time it occurred. The time is included in the error message so users know when an error occurred if they may be away from the SEM during the error.

3.4. Image acquisition loop

3.4.1. Microscope Initialization

The first function seen in the image acquisition loop is the microscope initialization function. The microscope initialization function ensures that the SEM has the correct settings before beginning the image acquisition. The settings that are initialized are: Scan mode, line time, and lines per frame. The scan mode is always set to full frame mode, this ensures that when capturing a photo, the photo is of the entire frame instead of a smaller part of it. The line time and lines per frame are used for the signal to noise ratio of the captured image [16] [17]. The line time and lines per frame are chosen from predetermined values seen in the user manual [9]. It does not change the image resolution. The user has to manually adjust it in the XL software. This is because the opcode library does not include a function for changing the image resolution [5]. Between all the messages that are send to the microscope, a `sleep` function is placed. This is to make sure that the microscope has enough time to execute the command and return an acknowledgement.

Also included in the Python command library are the commands for setting the spot size, setting the high tension voltage and turning on the beam. These commands were eventually omitted in the microscope initialization. See section 4.3 for the reasoning behind this decision.

The microscope initialization function also creates a settings file for the Image Processing submodule. This is a `.txt` file with the sesttings, the analysis submodule needs to function properly. These settings are: the path location of the captured images, the time interval between images in seconds, the amount of frames per second, the max percentage of drift before requesting a beam shift, the out-painting method, and the image name and extension.

A `.txt` file was chosen to transport these settings between modules for two reasons. The first reason is that the subsystems need to be able to run separately. The control subsystem has the UI where the user chooses all the settings, so a permanent file that is included in the folder with all the acquired images is the best way to ensure the settings are delivered properly to the Image Processing subsystem. Another reason for usage of a `.txt` file, is because it is easy to read out and easy to make in Python. This means that the code is mode user friendly then when other file types are used.

3.4.2. Image acquisition loop without drift correction

At the beginning of the image acquisition loop, before the code enters an actual loop, the communication with the Image Processing is opened. If both subsystems are connected then the code enters the loop. The loop exits when the user specified amount of frames are made. In the loop four functions are called. These functions are blanking off, blanking on, make photo, and the move file function. How a photo is made with the SEM is described in section 3.1.5.

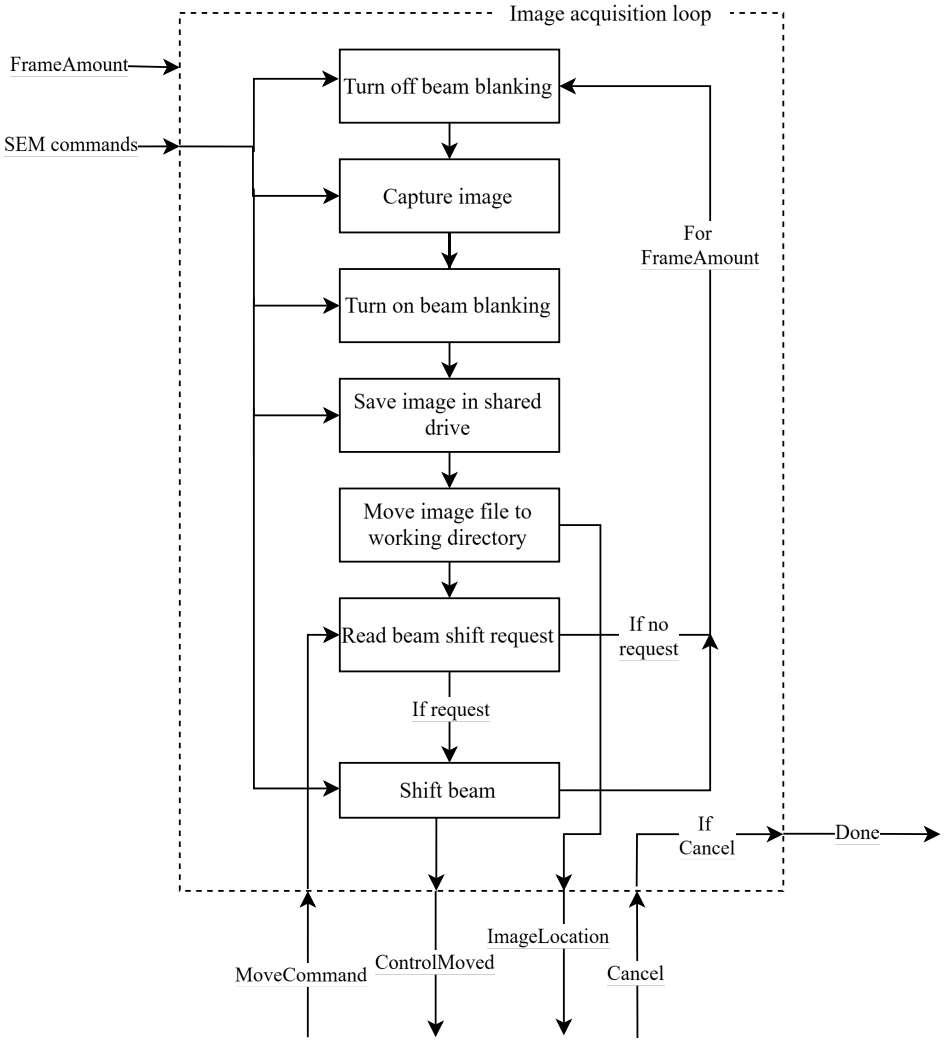


Figure 3.4: Block diagram of the image acquisition loop

If a sample is exposed to an electron beam, some of the electrons will be absorbed by the material [18]. This results in a darker image on the SEM [2]. To prevent this phenomenon the electron beam is redirected after every image capture. This is called beam blanking [9]. The beam blanking will stay on until the loop begins anew. Therefore the first function in the loop is beam blanking off. There was chosen to do this at the beginning of the loop instead of at the end to ensure that the beam blanking is on for the entire loop and the beam blanking is turned off before capturing an image.

Because the software runs on the Windows 7 PC and the SEM runs on the Windows 2000 PC, the `MoveFile` function was created. All images are saved in a shared drive between both PCs, to make sure none of the images get overwritten or a video with too many frames is made the shared drive needs to be empty and all the images need to be in their own folder. The `MoveFile` function works by scanning the shared drive for files and then putting the file names in a list. Then it checks if the file extension is `.tif`, if so the function uses `shutil` [19] to move the file to the correct folder. If the file extension is not `.tif` it deletes the file. This happens for the first two files in the folder which will be the saved images. The `shutil` package is used because the folders that the files are moved between are on different drives. The `shutil` move function, moves the files between folders, with the `os` package it only renames the file [20]. After the files are moved the software waits till the specified interval time and then the loop start anew.

3.4.3. Image acquisition loop with drift correction

The main difference between the image acquisition loop with drift correction and the loop without drift correction, is that there is now a check after the files are moved if the Image Processing subsystem has send a message to the SEM control subsystem. There are two possible messages that can be sent. The first message is a `Cancel` message, this message is sent when there occurs an error in the analysis submodule. When this message is sent, the loop exits and the software sends a `Done` message to the Image Processing subsystem, after which the video is compiled with the frames available up to that point. The second message that can be sent is the `MoveCommand`. This is a request for a beam shift. The beam shift parameters correspond to the absolute position of the beam on the SEM (maximum $\pm 20 \mu m$ [3]). This means the exact new beam position needs to be sent, not the amount of shift in the X and Y direction. These coordinates are passed as the data field in the message sent by the Image Processing subsystem. The full implementation of both the loops and initialization can be seen in Appendix B.3.

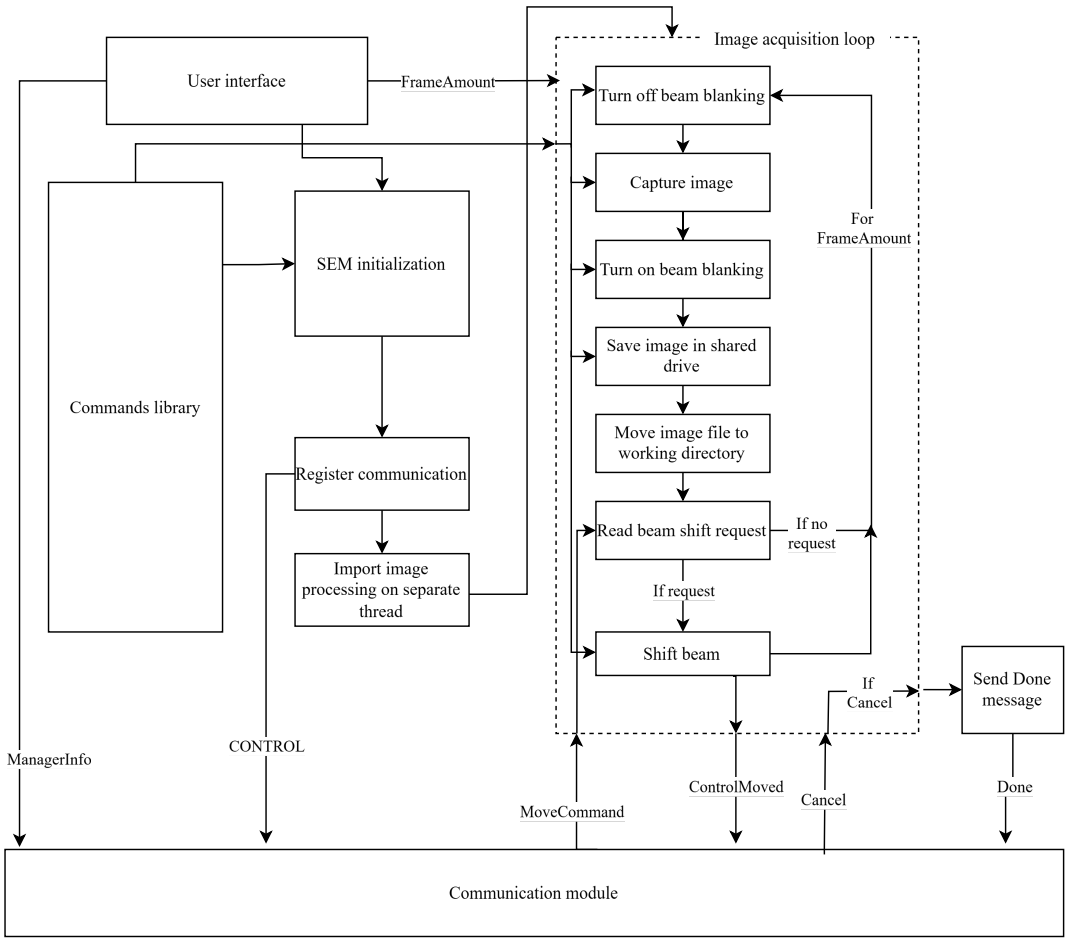


Figure 3.5: Block diagram of the total SEM control subsystem

3.5. Summary

In this chapter the message structure and the design dissensions for the automated image acquisition were discussed. A full block diagram of the entire subsystem can be seen in figure 3.5. The communication with the SEM is done over the serial line using single block RS232 messages. The message structure, data representation, the possible SCS responses and how the system handles errors from the SEM was discussed. The Python library that was developed for the system is also described and can be seen in Appendix B. Due to a simple make photo command not existing for the SEM, the process of capturing an image is also described. Two alternative for communication were also described and refuted, the C++ library and an external microcontroller. Next, the communication with the image processing subsystem is described. This is a custom communication protocol made specially for the system. The initialization of the communication is described

along with the message structure, and the protocol itself.

Then, the user interface is discussed. There was described why a command-line-only UI was rejected and why Python was chosen for the GUI. Also is described how the error handling works in the GUI. Lastly, the full image acquisition loop is discussed. How the loop initializes the microscope, how the image acquisition is handled without drift correction, and what the difference is when drift correction is implemented are all described. All these design choices make the basis for a reliable automated image acquisition.

4

Testing and results

4.1. Testing methods

Besides the obvious testing of software on the microscope, multiple testing methods were used. A DOS program was included in the XL software. With this DOS program it was possible to send example commands to the microscope. To use this DOS program on a PC DOSbox was used. With the DOSbox serial protocol [21] the PC could send commands via the serial port to the SEM without the need for python code. Therefore the message that was send could be read by a serial monitor. The Windows 2000 PC already had a serial monitor pre-installed, putty. There was one problem with putty, that was that putty would translate the binary message to ASCII, making it unreadable for debugging purposes.

To properly read the messages sent by DOSbox and eventually the software, a testing setup was configured by connecting the laptop running the DOS program to the Windows 7 PC over the serial line. The received messages could then be read using a serial port monitor, which had the capability of displaying the messages in hexadecimal instead of ASCII. With that it could be checked if the correct messages were being sent by the software. Virtual COM ports were also considered during testing. This way, there would be no need for a physical COM port to be connected to the PC. This would speed up the testing process because code could be tested on multiple machines simultaneously. However, these virtual COM ports ended up not being compatible with the DOSbox software, so they could not be used to test the end product.

4.2. Test results

A few commands were chosen to be excluded from the microscope initialization. These commands are: setting the spot size, setting the high tension voltage, and turning on the beam. These settings were tested, and it was found that setting these values would alter the images focus. The image would turn out blurry, even if the user had put it in focus before. The autofocus function was also used to fix this but it was less accurate then a manual focus. To ensure that the final video was in focus these settings were excluded from the initialization, and the user is required to set these settings before running the software.

During testing some errors occurred when testing the automated image acquisition loop. The messages that were received from the SEM didn't make sense. It turned out that the messages of multiple commands were put together in to a message that was unreadable for the program. To combat these errors, the response buffer is now emptied after a message is read, so the full buffer won't interfere with the next message. After implementing this fix, the system would still give some errors. This time the error would not come from the message send by the SEM, but the program would give the error that it had disconnected from the SEM. It turned out that the SEM would only send an acknowledge message when it was done executing a command. To fix this the `time.sleep()` command was used to wait a certain amount of seconds depending on the command that was executed.

Lastly, some issues were encountered with the `SaveTiffImage` command, that saves the image file from the SEM image buffer to a `.tif` file in a specified directory. The opcode library dictates to use a 16 bit mask where the user can specify certain image characteristics such as whether to imprint a databar on the image or whether to overwrite if a file with the same name already exists. However, through testing it was found that this 16 bit mask should be followed by two 0 bytes. That is because the SEM expects the data to be sent in multiples of four bytes. After implementing this change, the command worked as expected.

4.3. Requirements

After testing the control submodule, it was found that almost all requirements were met. All the requirements of the whole system were met, and only one of the requirements of the control subsystem was not met. First off, there is no interference with the existing system. No cable rearrangement is done, no extra hardware is added, and the system works with the existing software on both Windows PC's. The control submodule is able to be run separately from the image processing module. The subsystem is able to send commands and read parameters to/from the SEM, the details of which are discussed in section 3.1. The system is able to detect when the SEM has failed. How this is implemented is discussed in section 3.1.6 and 3.3.3. The subsystem has an automated image acquisition loop, the loops made are discussed in sections 3.4.2 and 3.4.3. The system is able to calculate image acquisition time as can be seen in Appendix B.3. The system can save a captured image to to .TIF file as is discussed in section 3.1.5. The system ensures that there are no more than two files in the shared drive, the details of which are discussed in section 3.4.2. The system is able to communicate with the image processing subsystem within the given time and is able to handle a beam shift request without influencing the time delay between image captures, this is discussed in sections 3.2 and 3.4.3. Also, the final User Interface can be seen in Figure 4.1.

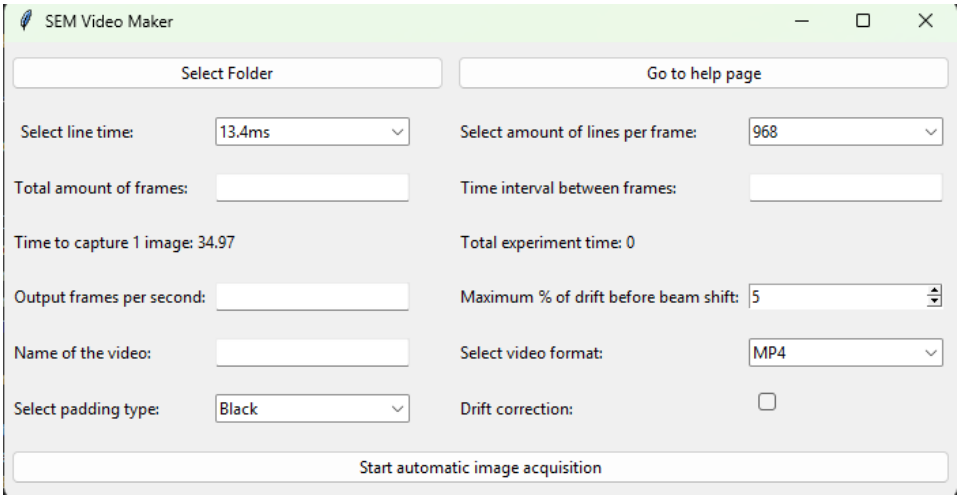


Figure 4.1: The final GUI

The only requirement not met by our subsystem is the requirement that the system must allow the user to configure the following SEM parameters:on voltage, spot size, lines per frame, and line scan time. The only two that were managed to be completed are lines per frame and line scan time. The reason why this requirement was not met is discussed in section 4.2. Why the requirement "The automatic operation of the SEM must not alter the image focus" is met, is also discussed in section 4.2.

5

Conclusion

The goal of the BAP-project was to implement an automated image acquisition for the XL30 SEM. This is implemented in two different ways. The first way is an image acquisition loop without drift correction. This loop captures images with the SEM over a time period. While this is a nice additional feature to the SEM setup, the video will be less reliable on larger magnification. This is why the second way was implemented, an image acquisition loop with drift correction. When the sample has drifted too far to one side the software is able to correct this drift via a beam shift. The implementation with beam shift is more complex, but makes for more stable video at larger magnification. To implement the correct command messages, DOSbox and a serial port monitor were used to test the outputs of the software. With this the correct message could be sent. The other goal of this project was to make an User Interface that was user-friendly. This was done via the use of Tkinter, a Python library for making a clean and user-friendly UI. When looking back at the program of requirements, all but one of the requirements were met. It turned out that setting the high tension voltage via the serial line caused the focus of the image to change, which is why this feature was omitted in the final product. Therefore nearly all requirements are met a robust and reliable piece of software could be made.

6

Discussion

While our software works as required for a proof-of-concept, in order to have a complete, user-friendly product, several changes and additions could be made. There are multiple features that could be added to the software in the future. In the current state of the software the user is expected to clean up the shared drive before starting the software. A feature to implement is a pop-up window that would show when the user wants to start the automated image acquisition, that the shared drive will be emptied and if the user wants to keep files on the shared drive. If so, the user must first move those files to another directory. Another feature that could be added is a non full frame mode. This is possible with the SEM and could be a nice feature for some use cases.

When the SEM image resolution is set to high definition, it allows only two values for the lines per frame setting. Currently, this hurdle is mentioned in the help page, but it would be more elegant to lock the other options or create a pop-up. Furthermore, the COM port is not selectable, this is mainly a problem when switching between PCs. It could be nice to add a COM port selector.

As mentioned in section 3.1.7, the main image acquisition loop exits when a `Cancel` message is detected. However, in the current implementation the `Cancel` message is never sent, and therefore never received, hence the loop will only exit when the desired number of images is made. The idea of the `Cancel` message is that either subsystem has the ability to declare that a critical error has occurred and operation must quit. Yet, this feature remains unused, so the extra safeguard is fairly redundant.

Additionally, the beam blanking feature is something that already exists in the SEM. In the configuration menu the user can choose to automatically blank the beam if the filter mode is set to Freeze. This makes our forced beam blanking in the image acquisition loop unnecessary. Yet, it is still kept as a failsafe, when the user accidentally might forget to turn on the automatic beam blanking in the SEM. Also, the `BeamBlankingOff` function is used to verify the communication with the SEM at the beginning of the loop.

The last feature that could be improved is the time interval. This time interval is now hard coded with sleep functions with a set value, and therefore quite inaccurate. Currently, all time delays are summed by hand and then used in the computation for how long to wait before starting the next capture. A great improvement on the software would be to compute this value dynamically. This could for example be done by reading out when the images are moved from the Shared drive to the output directory, as this timing is always the same. This would yield much greater timing accuracy and therefore create a better tool for research.

Bibliography

[1] F. E. Optics, All you wanted to know about electron microscopy...

[2] M. Australia, Scanning electron microscopy. [Online]. Available: https://myscope.training/pdf/MyScope_SEM.pdf.

[3] Xl30 s feg scanning electron microscope, XL30, Philips Electron Optics and FEI.

[4] Fundamentals of RS-232 Serial Communications, https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html, Analog Devices, 2001.

[5] Philips Electron Optics and FEI, "Control and communication with external computers and programs," in XL Interfacing Handbook. 2002.

[6] David A. Patterson, John L. Hennesy, "Computer Organization and Design, The Hardware/Software Interface," in Elsevier Inc., 2014, ch. 2: Instructions: Language of the Computer.

[7] Struct — interpret bytes as packed binary data, https://docs.python.org/3/library/struct.html, 2025.

[8] pySerial Documentation, https://docs.python.org/3/library/tkinter.html, 2020.

[9] Philips Electron Optics and FEI, The XL FEG/SFEG/SIRION Scanning Electron Microscope Operating Manual, 2002.

[10] P. van Miegehm, Data Communications Networking. Techne Press, 2011.

[11] K.-J. Li, Y.-Z. Xie, F. Zhang, and Y.-H. Chen, "Statistical inference of serial communication errors caused by repetitive electromagnetic disturbances," IEEE Transactions on Electromagnetic Compatibility, vol. 62, no. 4, pp. 1160–1168, 2020. doi: 10.1109/TEMC.2019.2932855.

[12] S. Ahmadi, Mobile WiMAX. Elsevier, 2011.

[13] 32 bit XCLLIB Thunking, FEI Company, 1997.

[14] tkinter.ttk — Tk themed widgets, https://docs.python.org/3/library/tkinter.ttk.html, 2025.

[15] tkinter — Python interface to Tcl/Tk, https://docs.python.org/3/library/tkinter.html, 2025.

[16] S. C. T. Kai Liang Lew Kok Swee Sim, "Single image estimation techniques for sem imaging system," JOIV, 2025.

[17] S. D. Naresh Marturi and N. Piat, "Scanning electron microscope image signal-to-noise ratio monitoring for micro-nanomanipulation," Agence Nationale de la Recherche, 2014.

[18] S. Fakhfakh, K. Raouadi, and O. Jbara, "Behaviour of dielectric materials under electron irradiation in a sem," in Advanced Topological Insulators. Wiley-Scrivener, 2019, pp. 281–330.

[19] shutil — High-level file operations, https://docs.python.org/3/library/shutil.html, 2025.

[20] os — Miscellaneous operating system interfaces, https://docs.python.org/3/library/os.html, 2025.

[21] Configuration:serialport, https://www.dosbox.com/wiki/Configuration:SerialPort, 2009.

A

Encoding of SEM command parameters

As mentioned in section 3.1, when sending a `write` command to the SEM, data parameters need to be passed. This data can be either of type integer, float or string. This appendix describes how each data type is encoded in the command message.

A.1. Integers

The data always needs to be encoded in multiples of four bytes. That means that two 16 bit integer values can be passed. These are encoded according to the Intel (little-endian) ordering convention. That is, the bytes are sent from least significant to most significant. An example can be seen below:

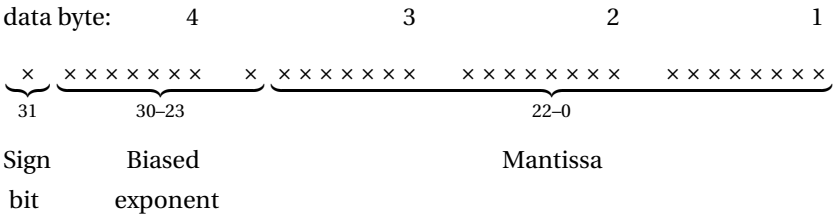
integer:	1		2	
data byte:	1	2	3	4
	LSB_1	MSB_1	LSB_2	MSB_2

The numbers 1 and 256 are represented as (in hex)

`01 00 00 01`

A.2. Floats

Floating point numbers are represented according to the single precision IEEE 754 format [6]. The 32 bit number contains three fields: the sign (1 bit), the biased exponent (8 bits) and the mantissa (23 bits).



A.3. Strings

Strings are encoded according to the ASCII encoding. The length of a string is determined by the number of characters in the string + one empty byte to indicate the end of the string. The total must be rounded up to a multiple of four bytes. As an example:

`example_string = '1234':` length = 4 + 1 = 5, so the total length will be rounded to 8.

B

Python code

B.1. Commands library

```
#######################################################
#   SEM commands library                              #
#   Authors: Matthijs Spijker & Tijmen ten Berge      #
#   Last updated: 12-12-2025                          #
#######################################################

import serial
import struct
import time

# Initialize serial communication
ser = serial.Serial(port='COM2', bytesize=8, stopbits=1, baudrate=9600, timeout=2)

# Converts floats to the IEEE 754 representation
def float2ieee(val:float) -> bytes:
    val_ieee = struct.pack('<f', val)
    return val_ieee

# Converts integers to a bytes object
def int2hex(val:int) -> bytes:
    val_intel = struct.pack('<H', val)
    return val_intel

# Converts text strings to the SEM format
def text2hex(text:str) -> bytes:
    encoded = text.encode('ascii') + b'\x00' #add empty character for end of string
    needed_padding = (4 - (len(encoded) % 4)) % 4 # padding to round to multiple of 4
    final = encoded + (b'\x00' * needed_padding)
    return final

# Adds checksum to given message
def addChecksum(msg_wo_checksum:bytes) -> bytes:
    checksum = sum(msg_wo_checksum) % 256
    message = msg_wo_checksum + bytes([checksum])
    return message

# Checks if the error flag of a response is set
def no_error_flag(response:bytes):
    mask = 1 << 7
    return(response[3] & mask) == 0

# Checks if the checksum of the response is valid
def is_correct_checksum(response, length):
```

```
    msg_wo_checksum = response[0:length-1]
    received_checksum = int(response[length-1])
    expected_checksum = sum(msg_wo_checksum) % 256
    return(received_checksum == expected_checksum)

# Checks if the response is equal to the original message
def acknowledge(message:bytes):
    time.sleep(0.5)
    response = ser.read(len(message))
    return (message == response)

# Returns the data field of the response message
def read_data(length:int) -> bytes:
    response = ser.read(length) # read response of same length as message
    if no_error_flag(response):
        if is_correct_checksum(response, length):
            incoming_data = response[4:length-1] # data is stored from byte 4 onwards.
            ↪  Checksum is omitted
            return incoming_data
        else:
            print("Incorrect checksum")
    else:
        print("Error flag detected")

def SetHighTension(OnOff:bool): #Turns the beam on or off
    id = 5
    length = 9
    opcode = 5
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(OnOff) + int2hex(0)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def SetHighTensionVoltage(BeamVoltage:float): #Sets the value of the beam voltage. Typical
↪  values are 5kV, 10kV, 20kV
    id = 5
    length = 9
    opcode = 3
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + float2ieee(BeamVoltage)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    print("Beam set to voltage:", BeamVoltage)
    return True

def SetBeamShift(shiftX, shiftY): #Shifts the electron beam. shiftX and shiftY are the exact
↪  beam coordinates in mm.
    id = 5
    length = 13
    opcode = 81
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + float2ieee(shiftX) +
    ↪  float2ieee(shiftY)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
```

```
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def GetMagnification() -> float: #Reads the current magnification level of the SEM
    id = 5
    length = 9
    opcode = 12
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + float2ieee(0)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    response = read_data(length)
    magnification = struct.unpack('<f', response)[0]
    print("Current magnification:", magnification)
    return magnification

def SetBeamBlank(val:bool): # beam blank off -> 0, beam blank on -> 1
    id = 5
    length = 9
    opcode = 63
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(val) + int2hex(0)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def SetProbeCurrent(spotsize:float): # Sets the spot size, recommended value: spotsize = 3.0
    id = 5
    length = 9
    opcode = 7
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + float2ieee(spotsize)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def SetFullFrameScanMode(): #Sets the SEM scan mode to full frame
    id = 5
    length = 9
    opcode = 17
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(7) + int2hex(0) # use
    ↪  scanmode = 7 for full frame scanning
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def SetLineTime(line_time:int): # Sets the line time of a single scan. Values are according to
↪  table 3 in documentation
    id = 5
    length = 9
```

```
    opcode = 21
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(line_time) + int2hex(0)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def SetLinesPerFrame(lines_per_frame:int): # Sets the number of lines in a single scan. Values
↪   are according to table in the documentation
    id = 5
    length = 9
    opcode = 19
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(lines_per_frame) +
    ↪   int2hex(0)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def SetFilterMode(): # Sets the filter mode to Average(1). Used for taking a single capture.
    id = 5
    length = 9
    opcode = 75
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(2) + int2hex(0) # set filter
    ↪   mode to average(1)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    if not(acknowledge(message)):
        print("ERROR: Message was not received")
        return False
    return True

def ReadFilterMode(): #Reads the current filter mode, until it is set to Freeze.
    id = 5
    length = 9
    opcode = 74
    flag = 0
    msg_wo_checksum = bytes([id, length, opcode, flag]) + int2hex(0) + int2hex(0)
    message = addChecksum(msg_wo_checksum)
    while True:
        ser.write(message)
        response = read_data(length)
        current_filter_mode = response[0]
        if current_filter_mode == 3: # if the filter mode is set to freeze
            print("Image complete!")
            break
        print("Busy taking image")
        time.sleep(3)

def SaveTiffImage(file_name:str): #Saves image in the SEM image buffer to the given name and
↪   directory
    id = 5
    length = 9 + len(text2hex(file_name))
    opcode = 84
    flag = 0
```

```
    mask = b'\x10\xc0\x00\x00' #"1100 0000 0001 0000", magnification for printing, graphics
    ↪  bitplane, if file exists -> overwrite
    msg_wo_checksum = bytes([id, length, opcode, flag]) + mask + text2hex(file_name)
    message = addChecksum(msg_wo_checksum)
    ser.write(message)
    time.sleep(10)
    response = ser.read(9 + len(text2hex(file_name)))
    if response != message:
        print("ERROR: Message was not received")
        return False
    return True

if __name__ == "__main__":
    pass
```

B.2. User Interface

```
#########################################################
#   User Interface                                      #
#   Authors: Matthijs Spijker & Tijmen ten Berge        #
#   Last updated: 12-12-2025                            #
#########################################################

import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
from tkinter import *
import MainLoop
import re


class App(tk.Tk):
    def __init__(self):
        super().__init__()
        container = ttk.Frame(self) #initialize the frame were the other frame are projected
        ↪  onto
        container.pack(fill="both", expand=True)
        self.folder = "" #makes sure folder is visable in all frames
        self.loop = MainLoop.MainLoopSolo(self) #initialize an instance of MainLoop so all
        ↪  functions can be properly used in the code
        #initialize the Frames so we can switch between main and help frame
        self.frames = {}
        for F in (MainFrame, HelpFrame):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(MainFrame)

    #Function for switching frames
    def show_frame(self, frame_class):
        frame = self.frames[frame_class]
        frame.tkraise()
        self.title(frame.title_text)

    #Function for selecting folder
    def fileselect(self):
        self.folder = filedialog.askdirectory()


class MainFrame(ttk.Frame):
    #initialize the title
```

```
    title_text = "SEM Video Maker"
    def __init__(self, master, controller):
        super().__init__(master)
        #The 3 lines below make sure the functions only_float, only_numbers and
        ↪  illegalcharacters can be used by the entry boxes
        validcmd = (self.register(self.only_float), '%P')
        vcmd = (self.register(self.only_numbers), '%P')
        validill = (self.register(self.illegalcharacters), '%P')
        #initializing the controller
        self.controller = controller
        self.img_acquisition_time = 34.97 #the standard imgage acquisition time
        #The combobox for the line time gets initialized
        self.n = tk.StringVar(value="13.4ms")
        self.n.trace_add("write", lambda *args: self.updateImageLabel())
        self.linetime = ttk.Combobox(self, textvariable=self.n)
        self.linetime['values'] = ('1.68ms', '3.36ms', '6.72ms', '13.4ms', '20.0ms', '40.0ms',
        ↪  '60.0ms', '120.0ms')
        self.linetime.state(['readonly'])
        #The combobox for the lines per frame gets initialized
        self.m = tk.StringVar(value="968")
        self.m.trace_add("write", lambda *args: self.updateImageLabel())
        self.linesperframe = ttk.Combobox(self, textvariable=self.m)
        self.linesperframe['values'] = ('484', '968', '1452', '1936', '2420', '2904', '3388',
        ↪  '3872')
        self.linesperframe.state(['readonly'])

        self.frameam = tk.StringVar() #The string variable that tracks what is in a entry gets
        ↪  initialized
        self.frameam.trace_add("write", lambda *args: self.updateTotalLabel()) #When the
        ↪  variable get updated the function updateTotalLabel gets activated
        self.frameamount = ttk.Entry(self, textvariable=self.frameam, validate='all',
        ↪  validatecommand=vcmd) #Entry gets initialized with the right validate command

        self.framessec = tk.StringVar()
        self.framenum = ttk.Entry(self, textvariable=self.framessec, validate='all',
        ↪  validatecommand=validcmd)

        self.solobutton = ttk.Button(self, text="Start automatic image acquisition",
        ↪  command=self.startMainLoop) #button gets initialized with the correct function

        self.filebutton = ttk.Button(self, text="Select Folder",
        ↪  command=self.controller.fileselect)
        self.helpbutton = ttk.Button(self, text="Go to help page", command=lambda:
        ↪  self.controller.show_frame(HelpFrame))

        self.edgespin = tk.Spinbox(self, from_=5, to=25)
        self.edgespin.bind("<MouseWheel>", self.onScrollEvent)

        self.intervalstring = tk.StringVar()
        self.intervalstring.trace_add("write", lambda *args: self.updateTotalLabel())
        self.intervalentry = ttk.Entry(self, textvariable=self.intervalstring, validate='all',
        ↪  validatecommand=vcmd)

        self.filenamestring = tk.StringVar()
        self.filenameentry = ttk.Entry(self, textvariable=self.filenamestring, validate='all',
        ↪  validatecommand=validill)

        self.paddingtypestring = tk.StringVar(value="Black")
        self.paddingtypecombo = ttk.Combobox(self, textvariable=self.paddingtypestring)
        self.paddingtypecombo['values'] =  ('Black', 'White', 'Blur', 'Same', 'Mean')
        self.paddingtypecombo.state(['readonly'])
```

```
        self.videoformatstring = tk.StringVar(value="MP4")
        self.videoformatdrop = ttk.Combobox(self, textvariable=self.videoformatstring)
        self.videoformatdrop['values'] = ('MKV', 'MP4')
        self.videoformatdrop.state(['readonly'])

        self.img_acquisition_timelabel = ttk.Label(self, text="Time to capture 1 image:
        ↪  34.97")

        self.total_experiment_label = ttk.Label(self, text="Total experiment time: 0")

        self.driftcheckvar = tk.IntVar()
        self.driftcheck = ttk.Checkbutton(self, variable=self.driftcheckvar, onvalue=1,
        ↪  offvalue=0)

        self.linetimelabel = ttk.Label(self, text="Select line time:")
        self.linesperframelabel = ttk.Label(self, text="Select amount of lines per frame:")
        self.frameamountlabel = ttk.Label(self, text="Total amount of frames:")
        self.fpslabel = ttk.Label(self, text="Output frames per second:")
        self.edgelabel = ttk.Label(self, text="Maximum % of drift before beam shift:")
        self.intervallabel = ttk.Label(self, text="Time interval between frames:")
        self.filenamelabel = ttk.Label(self, text="Name of the video:")
        self.fileextentionlabel = ttk.Label(self, text="Select video format:")
        self.paddinglabel = ttk.Label(self, text="Select padding type:")
        self.driftchecklabel = ttk.Label(self, text="Drift correction: ")

        self.filebutton.grid(row=0, column=0, padx=5, pady=10, columnspan=2, sticky="we")
        self.helpbutton.grid(row=0, column=2, padx=5, pady=10, columnspan=2, sticky="we")

        self.linetimelabel.grid(row=1, column=0, padx=(10, 5), pady=10, sticky="w")
        self.linetime.grid(row=1, column=1, padx=(0, 30), pady=10, sticky="we")
        self.linesperframelabel.grid(row=1, column=2, padx=5, pady=10, sticky="w")
        self.linesperframe.grid(row=1, column=3, padx=(0, 10), pady=10, sticky="we")

        self.frameamountlabel.grid(row=2, column=0, padx=5, pady=10, sticky="w")
        self.frameamount.grid(row=2, column=1, padx=(0, 30), pady=10, sticky="we")
        self.intervallabel.grid(row=2, column=2, padx=5, pady=10, sticky="w")
        self.intervalentry.grid(row=2, column=3, padx=(0, 10), pady=10, sticky="we")

        self.img_acquisition_timelabel.grid(row=3, column=0, padx=5, pady=10, columnspan=2,
        ↪  sticky="we")
        self.total_experiment_label.grid(row=3, column=2, padx=5, pady=10, columnspan=2,
        ↪  sticky="we")

        self.fpslabel.grid(row=4, column=0, padx=5, pady=10, sticky="w")
        self.framenum.grid(row=4, column=1, padx=(0, 30), pady=10, sticky="we")
        self.edgelabel.grid(row=4, column=2, padx=5, pady=10, sticky="w")
        self.edgespin.grid(row=4, column=3, padx=(0, 10), pady=10, sticky="we")


        self.filenamelabel.grid(row=5, column=0, padx=5, pady=10, sticky="w")
        self.filenameentry.grid(row=5, column=1, padx=(0, 30), sticky="we")
        self.fileextentionlabel.grid(row=5, column=2, padx=5, pady=10, sticky="w")
        self.videoformatdrop.grid(row=5, column=3, padx=(0, 10), pady=10, sticky="we")

        self.paddinglabel.grid(row=6, column=0, padx=5, pady=10, sticky="w")
        self.paddingtypecombo.grid(row=6, column=1,  padx=(0, 30), pady=10, sticky="we")
        self.driftchecklabel.grid(row=6, column=2, padx=5, pady=10, sticky="w")
        self.driftcheck.grid(row=6, column=3, padx=5, pady=(0, 10), sticky="we")

        self.solobutton.grid(row=7, column=0, padx=5, pady=10, columnspan=4, sticky="we")

    def updateImageLabel(self):
```

```
        self.img_acquisition_time = float(self.linetime.get()[:-2])/1000 *
        ↪  float(self.linesperframe.get()) + 22.0
        img_acquisition_timelabeltext = "Time to capture on image: " +
        ↪  str(round(self.img_acquisition_time, 2)) + "s"
        self.img_acquisition_timelabel.config(text=img_acquisition_timelabeltext)

    def updateTotalLabel(self):
        if self.frameamount.get() != "" and self.intervalentry.get() != "":
            total_experiment_time = int(self.frameamount.get()) *
            ↪  int(self.intervalentry.get())
            total_experiment_text = "Total experiment time: " + str(total_experiment_time) +
            ↪  "s"
            self.total_experiment_label.config(text=total_experiment_text)

    def only_float(self, new_value):
        if new_value == "":
            return True
        try:
            float(new_value)
            return True
        except ValueError:
            return False

    def only_numbers(self, new_value):
        if new_value == "":
            return True  # allow deleting everything
        return new_value.isdigit()  # allow only numbers

    def onScrollEvent(self, event):
        if event.delta > 0:
            self.edgespin.invoke("buttonup")
        else:
            self.edgespin.invoke("buttondown")

    def illegalcharacters(self, new_value):
        # Allow only A-Z, a-z, 0-9, - and _
        return bool(re.fullmatch(r"[A-Za-z0-9\-_]*", new_value))

    def startMainLoop(self):
        if (float(self.intervalentry.get()) - self.img_acquisition_time) < 1:
            self.raiseErrorWindow("ERROR: Frame interval too small for image acquisition
            ↪  time.", "")
        elif self.driftcheckvar.get() == 1:
            folder = self.controller.folder
            ErrorMessage, timeTaken =
            ↪  self.controller.loop.InitializeMicroscope(self.framenum.get(),
            ↪  self.linetime.get(), self.linesperframe.get(), self.edgespin.get(), folder,
            ↪  self.paddingtypecombo.get(), self.filenameentry.get(),
            ↪  self.videoformatdrop.get())
            if ErrorMessage != "":
                self.raiseErrorWindow(ErrorMessage, timeTaken)
            else:
                ErrorMessage, timeTaken =
                ↪  self.controller.loop.MainLoopDriftCorrection(self.frameamount.get(),
                ↪  self.intervalentry.get())
                if ErrorMessage != "":
                    self.raiseErrorWindow(ErrorMessage, timeTaken)
        else:
            folder = self.controller.folder
```

```
            ErrorMessage, timeTaken =
             ↪ self.controller.loop.InitializeMicroscope(self.framenum.get(),
             ↪ self.linetime.get(), self.linesperframe.get(), self.edgespin.get(), folder,
             ↪ self.paddingtypecombo.get(), self.filenameentry.get(),
             ↪ self.videoformatdrop.get())
            if ErrorMessage != "":
                self.raiseErrorWindow(ErrorMessage, timeTaken)
            else:
                ErrorMessage, timeTaken =
                 ↪ self.controller.loop.MainLoop(self.frameamount.get(),
                 ↪ self.intervalentry.get())
                if ErrorMessage != "":
                    self.raiseErrorWindow(ErrorMessage, timeTaken)

    def raiseErrorWindow(self, ErrorMessage, timeTaken):
        win = tk.Toplevel()
        win.wm_title("Error Occured")
        Errorlabel = ttk.Label(win, text=ErrorMessage)
        Errorlabel.config(font=("TkDefaultFont", 10))
        Errorlabel.grid(row=0, column=0, padx=30, pady=20)
        if timeTaken != "":
            timelabeltext = "Error Occured at: " + timeTaken
            timeLabel = ttk.Label(win, text=timelabeltext)
            timeLabel.config(font=("TkDefaultFont", 10))
            timeLabel.grid(row=1, column=0, padx=30, pady=20)
        button = ttk.Button(win, text="Okay", command=win.destroy)
        button.grid(row=2, column=0, padx=30, pady=0)

class HelpFrame(ttk.Frame):
    title_text = "Help Page"
    def __init__(self, master, controller):
        super().__init__(master)
        self.controller = controller
        self.master
        self.helpbutton = ttk.Button(self, text="Go to main page", command=lambda:
         ↪ self.controller.show_frame(MainFrame))
        self.title = ttk.Label(self, text="Help Frame")

        self.helpbutton.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="we")
        self.title.grid(row=1, column=0, columnspan=4, padx=10, pady=10, sticky="we")


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

B.3. Image acquisition main loop

```
#########################################################
#   Image acquisition main loop                         #
#   Authors: Matthijs Spijker & Tijmen ten Berge        #
#   Last updated: 12-12-2025                            #
#########################################################

import time
import os
import shutil
import threading
from commands import *
from Analysis import main, messages, communication
import datetime
#from Analysis.messages import message_types as mt
```

```
#from Analysis.messages import Message
#from Analysis.communication import CombinedCommunication as cc

class MainLoopSolo():
    def __init__(self, main):
        self.main = main
        self.sleep_time = 0
        self.pathLocation = ""
        self.img_acquisition_time = 0
        self.settingsList = []
        self.linetime_dict = {
            '1.68ms' : 3,
            '3.36ms' : 4,
            '6.72ms' : 5,
            '13.4ms' : 6,
            '20.0ms' : 7,
            '40.0ms' : 8,
            '60.0ms' : 9,
            '120.0ms': 10
        }
        self.lines_per_frame_dict = {
            '121' : 0,
            '242' : 1,
            '484' : 2,
            '968' : 3,
            '1452' : 4,
            '1936' : 5,
            '2420' : 6,
            '2904' : 7,
            '3388' : 8,
            '3872' : 9
        }


    def MainLoopDriftCorrection(self, frameAmount:str, frameInterval:str):
        frameAmountNum = int(frameAmount)
        frameIntervalNum = int(frameInterval)
        source = r"Z:"
        destination = self.pathLocation

        msg = messages.Message(messages.message_types.ManagerInfo, self.settingsList,
        ↪  "ANALYSIS")
        comm = communication.CombinedCommunication(msg)
        comm.connect("CONTROL")

        thread = threading.Thread(target=main.analysis_main, args=(comm,))
        thread.start()

        if comm.check_connected("ANALYSIS"):
            print("Communication module ready")

            #START UP IMAGE ACQUISITION LOOP:
            print("Starting image acquisition with drift correction")
            for i in range(frameAmountNum):

                # CHECK IF COMMUNICATION WITH THE SEM IS OPERATIONAL
                if not(FunctionsUsed.BlankingOff()):
                    #show error
                    date = str(datetime.datetime.now()).split(".")[0]
                    timetaken = date.split(" ")[1]
                    ErrorMessage = "ERROR: Could not connect to microscope"
                    # send done message to ANALYSIS
```

```
                msg_done = messages.Message(messages.message_types.Done, True, "ANALYSIS")
                comm.send(msg_done)
                return ErrorMessage, timetaken

            # CAPTURE THE CURRENT IMAGE AND SAVE TO FILE
            FunctionsUsed.MakePhoto(i)
            FunctionsUsed.BlankingOn()

            # MOVE IMAGE FILE TO OUTPUT DIRECTORY
            print(source, type(source))
            print(destination, type(destination))
            destination_file = self.MoveFile(source, destination)

            # SEND MESSAGE TO ANALYSIS THAT IMAGE IS READY
            img_ready = messages.Message(messages.message_types.ImageLocation,
            ↪  destination_file, "ANALYSIS")
            comm.send(img_ready)
            print("Photo ready and stored!")

            print("Waiting until next capture.")
            time.sleep(frameIntervalNum - self.img_acquisition_time - 22)

            # CHECK IF THERE IS A MESSAGE IN THE QUEUE
            print("Checking if there is a message in the queue.")
            if comm.message_available("CONTROL"):
                incoming_message = comm.receive("CONTROL")
                print("Detected message in queue")
                if incoming_message.message_type == messages.message_types.Cancel:
                    print("Detected CANCEL message from ANALYSIS")
                    msg_done = messages.Message(messages.message_types.Done, True,
                    ↪  "ANALYSIS")
                    comm.send(msg_done)
                    return ErrorMessage, timetaken

                # CHECK IF DRIFT CORRECTION IS NEEDED #
                elif incoming_message.message_type == messages.message_types.MoveCommand:
                    (shiftX, shiftY) = incoming_message.message_content
                    FunctionsUsed.BeamShift(shiftX, shiftY)
                    print("Shift request detected")
                    time.sleep(2)
                    move_completed = messages.Message(messages.message_types.ControlMoved,
                    ↪  True, "ANALYSIS")
                    comm.send(move_completed)
                else:
                    continue

        # SEND MESSAGE THAT IMAGE ACQUISITION IS FINISHED
        msg_done = messages.Message(messages.message_types.Done, True, "ANALYSIS")
        comm.send(msg_done)

    # WAIT FOR ANALYSIS THREAD TO FINISH
    print("Done message sent, waiting for ANALYSIS to finish...")
    thread.join()

    # RETURN EMPTY ERROR
    print("Output video compiled and stored in directory")
    return "", ""

def MainLoop(self, frameAmount:str, frameInterval:str):
    frameAmountNum = int(frameAmount)
    frameIntervalNum = int(frameInterval)
    source = r"Z:"
```

```
        destination = self.pathLocation

        msg = messages.Message(messages.message_types.ManagerInfo, self.settingsList,
        ↪  "ANALYSIS")
        comm = communication.CombinedCommunication(msg)
        comm.connect("CONTROL")

        thread = threading.Thread(target=main.analysis_main, args=(comm,))
        thread.start()

        if comm.check_connected("ANALYSIS"):
            print("Communication module ready")

            #START UP IMAGE ACQUISITION LOOP:
            for i in range(frameAmountNum):
                # CHECK IF COMMUNICATION WITH SEM WORKS
                if not(FunctionsUsed.BlankingOff()):
                    # display error
                    date = str(datetime.datetime.now()).split(".")[0]
                    timetaken = date.split(" ")[1]
                    ErrorMessage = "ERROR: Could not connect to microscope"
                    # send done message to ANALYSIS
                    msg_done = messages.Message(messages.message_types.Done, True, "ANALYSIS")
                    comm.send(msg_done)
                    return ErrorMessage, timetaken
                FunctionsUsed.MakePhoto(i)
                FunctionsUsed.BlankingOn()

                time.sleep(frameIntervalNum - self.img_acquisition_time - 22)

                # MOVE IMAGE FILE TO OUTPUT DIRECTORY
                destination_file = self.MoveFile(source, destination)

                # SEND MESSAGE TO ANALYSIS THAT IMAGE IS READY
                img_ready = messages.Message(messages.message_types.ImageLocation,
                ↪  destination_file, "ANALYSIS")
                comm.send(img_ready)
                print("Photo ready and stored!")

            # SEND MESSAGE TO ANALYSIS THAT IMAGE ACQUISITION IS FINISHED
            msg_done = messages.Message(messages.message_types.Done, True, "ANALYSIS")
            comm.send(msg_done)

        # WAIT FOR ANALYSIS THREAD TO FINISH
        print("Done message sent, waiting for ANALYSIS to finish...")
        thread.join()

        # RETURN EMPTY ERROR
        print("Output video compiled and stored in directory")
        return "", ""


    def InitializeMicroscope(self, framesPerSecond:str, linetime:str, lines_per_frame:str,
    ↪  maxDriftPercentage:str, folder:str, outpainting:str, filename:str, extention:str):
        # CHECK IF COMMUNICATION WITH SEM IS AVAILABLE BY SETTING THE HIGH TENSION VOLTAGE
        if not(FunctionsUsed.SetFullFrame()):
            date = str(datetime.datetime.now()).split(".")[0]
            timetaken = date.split(" ")[1]
            ErrorMessage = "ERROR: Could not connect to microscope"
            return ErrorMessage, timetaken
        time.sleep(1)
```

```
        self.img_acquisition_time = linetime * lines_per_frame

        FunctionsUsed.SetPictureQuality(self.linetime_dict[linetime],
        ↪  self.lines_per_frame_dict[lines_per_frame])
        time.sleep(1)

        # MAKE FILE SETTINGS.TXT WITH NECESSARY USER SETTINGS FOR PICTURE ANALYSIS
        self.pathLocation = folder
        path = os.path.join(folder, "settings.txt")
        self.settingsList.append("output_location: " + folder)
        self.settingsList.append("interval: " + str(self.sleep_time))
        self.settingsList.append("fps: " + framesPerSecond)
        self.settingsList.append("max_shift_percentage: " + maxDriftPercentage)
        self.settingsList.append("Out-painting: " + outpainting)
        self.settingsList.append("File name: " + filename)
        self.settingsList.append("Extention: " + extention)
        with open(path, "w") as f:
            for item in self.settingsList:
                f.write(f"{item}\n")
        return "", ""

    # MOVE .TIF FILE FROM SOURCE DIRECTORY TO DESTINATION DIRECTORY
    def MoveFile(self, source, destination) -> str:
        file = os.listdir(source)
        if file[0].split('.')[1] == "TIF":
            source_file = os.path.join(source, file[0])
            destination_file = os.path.join(destination, file[0])
            final_destination_file = destination_file
            shutil.move(source_file, destination_file)
        else:
            source_file = os.path.join(source, file[0])
            os.remove(source_file)
        if len(file) != 2:
            if file[1].split('.')[1] == "TIF":
                source_file = os.path.join(source, file[1])
                destination_file = os.path.join(destination, file[1])
                final_destination_file = destination_file
                shutil.move(source_file, destination_file)
            else:
                source_file = os.path.join(source, file[1])
                os.remove(source_file)
        return final_destination_file


class FunctionsUsed():
    def MakePhoto(i):
        SetFilterMode()
        ReadFilterMode()
        SaveTiffImage(r"d:/users/shared/" + str(i) + r".tif")
        time.sleep(20)

    def SetPictureQuality(linetime, lines_per_frame):
        SetLineTime(linetime)
        SetLinesPerFrame(lines_per_frame)
        print("Picture quality set to: linetime:", linetime, ", lines per frame:",
        ↪  lines_per_frame)

    def BlankingOn():
        SetBeamBlank(1)
        print("Beam blanking turned on")

    def BlankingOff():
```

```
        for i in range(5):
            if SetBeamBlank(0):
                print("Beam blanking turned off")
                time.sleep(0.5)
                return True
            elif i == 4:
                print("ERROR: Beam blanking could not be set.")
                return False
            else:
                time.sleep(0.5)
                continue
        SetBeamBlank(0)
        print("Beam blanking turned off")

    def SetSpotSize(spotSize):
        SetProbeCurrent(spotSize)
        print("Spotsize set to", spotSize)

    def SetBeamVoltage(voltagenum):
        SetHighTensionVoltage(voltagenum)
        SetHighTension(1)
        print("High tension set to", voltagenum)

    def SetFullFrame():
        for i in range(5):
            if SetFullFrameScanMode():
                print("Scan mode: full frame")
                return True
            elif i == 4:
                print("ERROR: Beam could not be turned on")
                return False
            else:
                time.sleep(0.5)
                continue

    def BeamShift(shiftX, shiftY):
        SetBeamShift(shiftX, shiftY)
        print("Beam shift:", shiftX, ",", shiftY, "performed.")

if __name__ == "__main__":
    pass
```

B.4. Communication module

B.4.1. Messages class

```
from __future__ import annotations

from enum import Enum, EnumMeta
from typing import Union

message_types = Enum('Types', [
        ('ManagerInfo', 0),
        ('ControlMoved', 1),
        ('ImageLocation', 2),
        ('MoveCommand', 3),
        ('Cancel', 4),
        ('Done', 5)
    ])

class Message:
    """Class that encodes the content of a message"""
```

```
def __init__(self, message_type: EnumMeta, data, receiver: str):
    # subfunction for this function's errors
    def error(the_message_type: EnumMeta, requested_type: Union[type, str], received_type:
    ↪  type):
        raise TypeError(f"{the_message_type} should receive data as {requested_type}. Got
        ↪  {received_type} instead")

    if message_type in message_types:
        self.message_type = message_type
    else:
        raise ValueError(f"Message type {message_type} is not a valid message type")
    # Check if data is of the correct (defined) type. The definitions are:
    if message_type == message_types.ManagerInfo:
        if not (isinstance(data, list)
                and all(isinstance(line, str) for line in data)):
            error(message_type, list, type(data))

    elif message_type == message_types.ControlMoved:
        if not isinstance(data, bool):
            error(message_type, bool, type(data))

    elif message_type == message_types.ImageLocation:
        if not isinstance(data, str):
            error(message_type, str, type(data))

    elif message_type == message_types.MoveCommand:
        if not (
                isinstance(data, tuple)
                and len(data) == 2
                and all(isinstance(i, float) for i in data)
        ):
            error(message_type, "a tuple of two floats", type(data))

    elif message_type == message_types.Cancel:
        if not isinstance(data, bool):
            error(message_type, bool, type(data))

    elif message_type == message_types.Done:
        if not isinstance(data, bool):
            error(message_type, bool, type(data))

    else:  # Catch all, should not happen
        # as the ValueError should have been raised if the message type was wrong.
        # Can only happen when you have defined the message type, but not the
        ↪  corresponding data type.
        raise RuntimeError(
            "The message type check was valid, but the message type is not implemented in
            ↪  the data type check"
        )
    self.message_content = data # if all checks were good, then we get here and we can
    ↪  assign the data

    if receiver in ['CONTROL', 'ANALYSIS']:
        self.receiver = receiver
    else:
        raise RuntimeError(f'Receiver {receiver} is not an allowed receiver')

def __str__(self):
    return f'{self.message_type}: {self.message_content}'

def __repr__(self):
    return str(self)
```

B.4.2. Communications class

```
from abc import ABC, abstractmethod
import os

from typing import Dict

try:
    from Analysis.messages import Message, message_types
except ModuleNotFoundError as e:
    if e.name in ('Analysis', 'Analysis.messages'):
        from messages import Message, message_types
    else:
        raise

class CommunicationProtocol(ABC):
    """This base class enforces what functions a comms method should have"""
    @abstractmethod
    def connect(self, initializer: str) -> bool:
        pass

    @abstractmethod
    def send(self, message: Message):
        pass

    @abstractmethod
    def receive(self, receiver: str) -> Message:
        pass

    @abstractmethod
    def message_available(self, receiver: str) -> bool:
        pass

    @abstractmethod
    def check_connected(self, entity: str) -> bool:
        pass

    @abstractmethod
    def close(self, entity: str):
        pass

class FolderCommunication(CommunicationProtocol):
    """The communication implementation for using just a folder

    Protocol description
    This protocol acts like a communication class, except that it uses pre-existing data in a
    ↪ given folder.
    The messages are created based on the files that exist in that given folder.
    First the protocol searches for a .txt file with the manager data. This txt file should be
    ↪ given like this:
        PARAMETER1_NAME: PARAMETER1_VALUE
        PARAMETER2_NAME: PARAMETER2_VALUE
        PARAMETER3_NAME: PARAMETER3_VALUE
        etc

    Then it goes through the rest of the filenames. These are all assumed to be images, and
    ↪ will generate an
    ImageLocation messagetype. If the filename ends in _M or _F it is assumed that the between
    ↪ that image and the
    previous image a move or focus operation has been executed respectively. If a filename is
    ↪ found to have such an
    ending, a move or focus message is generated before also generating the ImageLocation
    ↪ message for that image
```

```
    The sending functions are implemented but do not contain any logic
    """

    def __init__(self, folder_location):
        self.folder_location = folder_location
        self.queue = []
        self.connect_status = list()
        self.connect_status.append('CONTROL') # Fake that control exists

    def connect(self, entity: str) -> bool:
        self.connect_status.append(entity)
        self.queue = []
        file_list = [f for f in os.listdir(self.folder_location) if
        ↪  os.path.isfile(os.path.join(self.folder_location, f))]
        file_list = [f'{self.folder_location}\\{f}' for f in file_list]
        not_img_msg = [] # list to store the messages that are not img locations
        # the order matters here. Because we search for a txt first, the first message will
        ↪  always be managerdata
        for f in file_list:
            if f.endswith('.txt'):
                with open(f, 'r') as file:
                    data = [line for line in file]
                    # data now is a list of lines
                msg = Message(message_types.ManagerInfo, data, 'ANALYSIS')
                self.queue.append(msg)
                not_img_msg.append(f)
        for f in not_img_msg:  # remove all non img locations from file_list
            file_list.remove(f)
        for f in file_list:
            filename = f.split('.')[0]
            if filename.endswith('_M'):
                self.queue.append(Message(message_types.ControlMoved, True, 'ANALYSIS'))
            msg = Message(message_types.ImageLocation, f, 'ANALYSIS')
            self.queue.append(msg)

        if len(self.queue) > 0:
            self.queue.append(Message(message_types.Done, True, 'ANALYSIS')) # add a done
            ↪  message to signal start of vid generation
        return len(self.queue) > 0

    def send(self, message: Message):
        pass # the file communication thing does not do sending

    def receive(self, receiver: str) -> Message:
        # assumes  that a message is available
        queue = [msg for msg in self.queue if msg.receiver == receiver]
        msg = queue.pop(0)
        self.queue.remove(msg)
        return msg

    def message_available(self, receiver: str) -> bool:
        queue = [msg for msg in self.queue if msg.receiver == receiver]
        return len(queue) > 0

    def check_connected(self, entity: str) -> bool:
        return entity in self.connect_status

    def close(self, entity: str):
        pass # nothing to close

class CombinedCommunication(CommunicationProtocol):
```

```
    def __init__(self, initial_message: Message):
        self.queue = list()
        self.queue.append(initial_message)
        self.connect_status = list()

    def connect(self, entity: str) -> bool:
        self.connect_status.append(entity)
        return entity in self.connect_status # if this returns False then wow

    def send(self, message: Message):
        self.queue.append(message)

    def receive(self, receiver: str) -> Message:
        # assumes  that a message is available
        queue = [msg for msg in self.queue if msg.receiver == receiver]
        msg = queue.pop(0)
        self.queue.remove(msg)
        return msg

    def message_available(self, receiver: str) -> bool:
        queue = [msg for msg in self.queue if msg.receiver == receiver]
        return len(queue) > 0

    def check_connected(self, entity: str) -> bool:
        return entity in self.connect_status

    def close(self, entity: str):
        self.connect_status.remove(entity)

def read_manager_data(message: Message) -> Dict[str, str]:
    manager_data = dict()
    if not message.message_type == message_types.ManagerInfo:
        raise ValueError("Message does not contain manager info")
    delimiter = ':'
    for line in message.message_content:
        if line.startswith('#'):
            continue
        key, val = line.split(delimiter, 1) # split at the first delimiter, all before that is
        ↪  the key, all after is val
        manager_data[key.strip()] = val.strip(' \n') # use strip to remove whitespace and
        ↪  newlines after the delimiter
    required = ['max_shift_percentage', 'output_location', 'fps', 'interval']
    missing = set(required) - manager_data.keys()
    if missing:
        raise AttributeError(f"Not all required parameters have been received. Be sure to send
        ↪  them in one message.\n"
                             f"Missing parameter(s): {', '.join(missing)}")
    return manager_data

# needed data:
# max x and y shifts # only if live
# file storage location
```

C

Error codes

```
#ifndef _GLOERR_H_
#define _GLOERR_H_

/*-----------------------------------------------------------------------*/
/*
    $Archive:   I:/p90nt/archive/inc/gloerr.h,v  $
   $Revision:   1.233  $
       $Date:   Apr 23 2003 13:54:04  $
    $Modtime:   Apr 23 2003 08:41:32  $
     $Author:   acr  $
*/
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*  Copyright (c) 1995, Philips Electronics N.V.                         */
/*                                                                       */
/*  Volume    : PCP                                                      */
/*                                                                       */
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*  module gloerr.h defines the globally available error codes           */
/*                                                                       */
/*  Messages prefixed with '@' are presented as warnings.                */
/*                                                                       */
/*  Messages prefixed with '#' cause the Error( ) call to wait until     */
/*  the user presses the OK button.                                      */
/*                                                                       */
/*  NB: Codes must be in the range: 0xC1000000 - 0xC17FFFFF              */
/*                                                                       */
/*-----------------------------------------------------------------------*/


/*------------------- Column Control Handler Error Codes ----------------*/

#define COL_GROUP                                   0xC10B0000uL
/* Column group */

#define COL_IMAGESIZE_RANGE                         0xC10B0001uL
/* @Column - ImageSize out of range */
#define COL_TILTANG_RANGE                           0xC10B0002uL
```

```
/* @Column – TiltAngle out of range */
#define COL_SPOT_RANGE                          0xC10B0003uL
/* Column – ProbeCurrent out of range */
#define COL_MAGNIF_RANGE                        0xC10B0004uL
/* Column – Magnification out of range */
#define COL_SCANROT_RANGE                       0xC10B0005uL
/* Column – ScanRotation out of range */
#define COL_BEAMSFT_RANGE                       0xC10B0006uL
/* Column – Beamshift out of range */
#define COL_WORKDIST_RANGE                      0xC10B0007uL
/* Column – WorkingDistance out of range */
#define COL_STIG_RANGE                          0xC10B0008uL
/* Column – Stigmator out of range */
#define COL_TILTCOR_RANGE                       0xC10B0009uL
/* Column – TiltCorrection out of range */
#define COL_DYN_ONOFF_RANGE                     0xC10B000AuL
/* Column – DynamicFocus On/Off out of range */
#define COL_CROSS_ONOFF_RANGE                   0xC10B000BuL
/* Column – Crossover On/Off out of range */
#define COL_CROSS_VAL_RANGE                     0xC10B000CuL
/* Column – CrossoverValue out of range */
#define COL_PROBE_RANGE                         0xC10B000DuL
/* Column – Probe Current out of range */
#define COL_DSTIG_RANGE                         0xC10B000EuL
/* Column – Delta Stigmator out of range */
#define COL_SMC_RANGE                           0xC10B000FuL
/* Column – Spot Magn coupling On/Off out of range */
#define COL_CROSS_NOT_IN_SLOWSCAN               0xC10B0010uL
/* Column – Crossover selection, while not in Slow Scan mode */
#define COL_NO_FEG                              0xC10B0011uL
/* Column – Only applicable or a FEG system. */
#define COL_NOT_FOR_FEG                         0xC10B0012uL
/* Column – Not applicable for a FEG system. */
#define COL_DYN_FOC_ON                          0xC10B0013uL
/* @Column – Warning : Dynamic Focus is ON */
#define COL_TILTCORR_WITH_ROTATION              0xC10B0014uL
/* @Column – Scan rotation is not zero. Tilt correction is not useful. */
#define COL_SNOR_RANGE                          0xC10B0015uL
/* @Column – Magn. or WD not in range for UHR Mode */
#define COL_NORMAL_RANGE                        0xC10B0016uL
/* @Column – WD not in range for HR Mode */
#define COL_ROTATION_WITH_TILTCORR              0xC10B0017uL
/* @Column – Tilt correction is active, Scan rotation is not useful. */
#define COL_AAM_RANGE                                                   0xC10B0018uL
/* Column – Align Angle Manual out of range */
#define COL_SNOR_RANGE_MAGN_HIGH                0xC10B0019uL
/* @Column – Magnification too high for HR Mode */
#define COL_SNOR_RANGE_MAGN_LOW                 0xC10B0020uL
/* @Column – Magnification too low for UHR Mode */
#define COL_SNOR_RANGE_WD_HIGH                  0xC10B0021uL
/* @Column – Working Distance too high for UHR Mode */
#define COL_SNOR_RANGE_HT_HIGH                  0xC10B0022uL
/* @Column – Hightension too high for UHR Mode */
#define COL_SNOR_RANGE_WD_LOW                   0xC10B0023uL
/* @Column – Working Distance too low for UHR Mode */
#define COL_EDAX_NOT_ALLOWED                    0xC10B0024uL
```

```
/* @Column - EDAX Lens Mode Not Allowed */


/*---------------------- Position Error codes -------------------------*/

#define POS_GROUP                                   0xC11E0000uL
/* Position group */

    /*-------------------------------------------------------------
        the POS will not generate an error ack
        to the various stage handlers
        like reject timeout, should not also try to send an Ack
        this will only make the situation worse
      -------------------------------------------------------------*/

#define POS_UNKNOWN_STAGE                           0xC11E0001uL
/* Unknown stagetype in MachineData    */
#define POS_NO_MOTOR_STAGE                          0xC11E0002uL
/* Motorized stage action requested, but manual stage in system */
#define POS_TILT_RANGE                              0xC11E0003uL
/* Position, Stage tilt out of range */
#define POS_TILT_NOT_ZERO                           0xC11E0004uL
/* @Position, Stage tilt is not zero. Set to zero and Home again */
#define POS_UNSUPPORTED_FUNCTION                    0xC11E0005uL
/* The selected functionality is not supported by this stagetype */
#define POS_INVALID_COORD_SYSTEM                    0xC11E0006uL
/* Software error, invalid coordinate system used */
#define POS_TILTMAP_ERROR                           0xC11E0007uL
/* Move not allowed by tilt map */
#define POS_STAGE_NOT_HOMED                         0xC11E0008uL
/* Stage not homed; use Stage - Home */
#define POS_LOADLOCK_BUSY                           0xC11E0009uL
/* LoadLock busy; move command ignored */
#define POS_LOADLOCK_INACTIVE                       0xC11E0010uL
/* The LoadLock is inactive */
#define POS_SW1_ERROR                               0xC11E0011uL
/* Internal SW error Position */
#define POS_SW2_ERROR                               0xC11E0012uL
/* Internal SW error Position */
#define POS_AXIS_MOVING                             0xC11E0013uL
/* The stage is busy moving */
#define POS_BEAM_SHIFT_RANGE                        0xC11E0021uL
/* Manual stage, BeamShift out of range */
#define POS_BEAM_SHIFT_HFW                          0xC11E0022uL
/* @Manual stage, BeamShift not used, Magnification too low */
#define POS_REJECT_TIMEOUT                          0xC11E0023uL
/* Timeout during wait for end of command rejects by stage handler */
#define STACOM_FILE_NOT_OPENED                      0xC11E0024uL
/* Server: Stage mapping file can not be opened */
#define STACOM_FILE_SETUP_READ_ERROR                0xC11E0025uL
/* Server: Setup data not read from stage mapping file */
#define STACOM_FILE_GLOBALALLOC_ERROR               0xC11E0026uL
/* Server: Global memory for mapping data cannot be allocated */
#define STACOM_FILE_MAPPINGRESULT_ERROR             0xC11E0027uL
/* Server: Global mapping data not read correctly */
#define STACOM_FILE_ENC_READ_ERROR                  0xC11E0028uL
```

```
/* Server: Encoder mapping data not read correctly */
#define STACOM_FILE_ROTATION_READ_ERROR                 0xC11E0029uL
/* Server: Rotation mapping data not read correctly */
#define STACOM_FILE_NAME_INCONSISTENT                   0xC11E002AuL
/* Server: Name of stage mapping file is not consistent with global mapping data */
#define STACOM_FILE_DIFFERENT_INSTRUMENT                0xC11E002BuL
/* Server: Stage mapping file was created on an other instrument */
#define POS_NO_MOTOR_Z_AXES                             0xC11E002CuL
/* Pos : This stage does not have a motorised Z axes  */
#define POS_MOTOR_Z_AXES_NOT_CPLD                       0xC11E002DuL
/* Pos : Z is not coupled to working distance */
#define POS_ENDMOVE_TIMEOUT_ERROR                       0xC11E002EuL
/* Pos : Wait for end move timeout , system has waited too long */
#define POS_ENDMOVE_TIMEOUT_WARNING                     0xC11E002FuL
/*@ Pos : Wait for end move timeout, system will retry again */

/*-------------- CTB PROM sw errors ---------------------------------------- */
/*               (keep numbers adjacent; see CTBDRIV.H)                      */

#define CTB_NOERROR                                     0xC1240100uL
/* CTB: No error (Ignore) */
#define CTB_BADCOMMAND                                  0xC1240101uL
/* CTB: Illegal Command Received */
#define CTB_BADRESPONSE                                 0xC1240102uL
/* CTB: Illegal Response Received */
#define CTB_BADDEVICESELECTED                           0xC1240103uL
/* CTB: Illegal Device Selected */
#define CTB_DEVICESTILLOFF                              0xC1240104uL
/* CTB: Device Still Off */
#define CTB_BADMODE                                     0xC1240105uL
/* CTB: Illegal Mode */
#define CTB_DEVICENOTRESPONDING                         0xC1240106uL
/* CTB: Device Not Responding */
#define CTB_DEVICEMALFUNCTIONING                        0xC1240107uL
/* CTB: Device Malfunctioning */
#define CTB_DEVICEOVERLIMIT                             0xC1240108uL
/* CTB: Device Over Limit */
#define CTB_DEVICESTILLINERROR                          0xC1240109uL
/* CTB: Device Still In Error */
#define CTB_DEVICEDRIFTING                              0xC124010AuL
/* CTB: Device Drifting */
#define CTB_DATABASEINCONSISTENT                        0xC124010BuL
/* CTB: Database. Inconsistent */
#define CTB_FEGBADDATA                                  0xC124010CuL
/* CTB: Illegal Data received from gun power supplies */
#define CTB_FEGTIMEOUT                                  0xC124010DuL
/* CTB: Time-Out in communication with gun power supplies */
#define CTB_P80BADDATA                                  0xC124010EuL
/* CTB: Illegal Data Received in Communication with MCCB */
#define CTB_P80TIMEOUT                                  0xC124010FuL
/* CTB: Time-Out in Communication with MCCB */
#define CTB_UNMATCHINGSOFTWAREVERSIONS                  0xC1240110uL
/* CTB: Versions of MCCB and CTB software do not match */
#define CTB_BADSTATE                                    0xC1240111uL
/* CTB: Illegal State */
#define CTB_BADITEMSIZE                                 0xC1240112uL
```

```
/* CTB: Illegal Item Size */
#define CTB_DBADDROUTOFRANGE                    0xC1240113uL
/* CTB: DataBase Address Out Of Range */
#define CTB_MEMORYFAULT                         0xC1240114uL
/* CTB: Memory Fault */
#define CTB_BADDEVICETYPE                       0xC1240115uL
/* CTB: Illegal Device Type */
#define CTB_BADDEVICE                           0xC1240116uL
/* CTB: Illegal Device */
#define CTB_HARDWARERESET                       0xC1240117uL
/* CTB: Hardware Reset Occurred */
#define CTB_CENTRALSUPPLYOUT                    0xC1240118uL
/* CTB: Gun Power Supplies are Disabled */
#define CTB_BADPARAMETER                        0xC1240119uL
/* CTB: Illegal Parameter */



#define SCS_GROUP                               0xC1250000uL
/* SCS group */
#define SCS_UNKNOWN_MESSAGE                     0xC1250001uL
/* SCS application: Unknown operation code */
#define SCS_NOT_ALLOWED                         0xC1250002uL
/* SCS application: Operation not allowed */
#define SCS_NOMEMORY                            0xC1250003uL
/* SCS application: Not enough memory available */
#define SCS_EDAM_ERROR                          0xC1250004uL
/* SCS application: The EDAM function returned an error */
#define SCS_NO_INTERFACE                        0xC1250005uL
/* SCS application: The EDAM interface library (EDAINT.DLL) is not there */
#define SCS_NOT_IMPLEMENTED                     0xC1250006uL
/* SCS application: The function is not (yet) implemented */
#define SCS_ABSENT                              0xC1250007uL
/* SCS application: The SCS application is absent */
#define SCS_STOP                                0xC1250008uL
/* SCS application: The SCS transmission is stopped/interupted */
#define SCS_TIMEOUT                             0xC1250009uL
/* SCS application: Timeout while sending an SCS multi-block message */
#define SCS_TRANSMISSION_ERROR                  0xC125000AuL
/* SCS application: Transmission error while sending an SCS multi-block msg */
#define SCS_PARAMETER_ERROR                     0xC125000BuL
/* SCS application: Parameter error */


#define ACNT_GROUP                              0xC1260000uL
/* ACCOUNT group */

#define     ACNT_FILE_NOT_FOUND                 0xC1260001uL
/* Accounting: file not found */
#define     ACNT_ILLEGAL_VECTOR_PART            0xC1260002uL
/* Accounting: file corrupted (vector part) */
#define     ACNT_ILLEGAL_FILE_TYPE              0xC1260003uL
/* Accounting: file corrupted (file type) */
#define     ACNT_NO_BIN_NO_TXT_FILE             0xC1260004uL
/* Accounting: file corrupted (no bin, no text) */
#define     ACNT_ILLEGAL_ITEMS                  0xC1260005uL
```

```
/* Accounting: file corrupted (illegal items) */
#define     ACNT_NO_ITEMS_FOUND                 0xC1260006uL
/* Accounting: file corrupted (no items found) */
#define     ACNT_WRITE_ERROR                    0xC1260007uL
/* Accounting: write file error */
#define     ACNT_WRONG_PASSWORD                 0xC1260008uL
/* @Accounting: Access denied: username - password mismatch*/
#define     ACNT_NEW_PASSWORD                   0xC1260009uL
/* @Accounting: new password required */
#define     ACNT_FILE_CORRUPTED                 0xC126000AuL
/* Accounting: file corrupted (file length)  */
#define     ACNT_INCOMPLETE_SAVE                0xC126000BuL
/* Accounting: file corrupted (during save)  */
#define     ACNT_NO_MEMORY                      0xC126000CuL
/* Accounting: no memory available */
#define     ACNT_CHECKSUM                       0xC126000DuL
/* Accounting: for this user account file is corrupted (checksum)  */
#define     ACNT_UNKNOWN_USER                   0xC126000EuL
/* @Accounting: user name unknown */
#define     ACNT_LOGAPP_NOT_RUNNING             0xC126000FuL
/* Accounting: logging application not running */
#define     ACNT_CANT_DEL_USER                  0xC1260010uL
/* Accounting: cannot delete this user */
#define     ACNT_MAXIMUM_USERS                  0xC1260011uL
/* Accounting: maximum number of users reached */
#define     ACNT_CANT_CREATE_USER               0xC1260012uL
/* Accounting: cannot create this user */
#define     ACNT_MINIMUM_PSWD_LEN               0xC1260013uL
/* @Accounting: At least 4 characters needed, password not changed */
#define     ACNT_PSWD_RETYPE                    0xC1260014uL
/* @Accounting: New password not correctly retyped */
#define     ACNT_PSWD_OLD_NEW                   0xC1260015uL
/* @Accounting: not initialized yet, see manual */
#define     ACNT_VALUE_OUT_OF_RANGE             0xC1260016uL
/* Accounting: Value out of range */
#define     ACNT_NAME_TO_SHORT                  0xC1260017uL
/* @Accounting: User group name: at least 3 characters */
#define     ACNT_ILLEGAL_PATH                   0xC1260018uL
/* Accounting: Illegal directory path */
#define     ACNT_SETGUN_FILLIM                  0xC1260019uL
/* Accounting: Gun might have been changed, Access denied */
#define     ACNT_CREATE_LOG_FILE_ERROR          0xC1260020uL
/* Accounting: Could not create Logfile, disk full? */
#define     ACNT_SUPERVISOR_NEEDS_ALL           0xC1260021uL
/* @Accounting: you cannot limit the access rights for the user with supervisor rights */
#define     ACNT_DOUBLE_NAME_USR                0xC1260022uL
/* @Accounting: user name already in use */
#define     ACNT_NO_NAME_USR                    0xC1260023uL
/* @Accounting: no user found with this name */
#define     ACNT_NAME_USR_ID                    0xC1260024uL
/* Accounting: name does not match user id */
#define     ACNT_DEL_USR_DIRECTORY              0xC1260025uL
/* @Accounting: directories and or user files for this user should be removed by using the filema


#define SEL_GROUP                               0xC1270000uL
```

```
/* System Event Logging group */

#define      SEL_FILE_CREATION_ERROR                0xC1270001uL
/* SE Logging: cannot create file */
#define      SEL_MEMORYALLOC_ERROR                  0xC1270002uL
/* SE Logging: out of local heap space */
#define      SEL_MEMORYLOCK_ERROR                   0xC1270003uL
/* SE Logging: local memory cannot be locked */
#define      SEL_FILEWRITE_ERROR                    0xC1270004uL
/* SE Logging: cannot write to log file */
#define      SEL_FILEOPEN_ERROR                     0xC1270005uL
/* SE Logging: cannot open file */
#define      SEL_FILEREAD_ERROR                     0xC1270006uL
/* SE Logging: cannot read file */
#define      SEL_INVALID_FILENAME                   0xC1270007uL
/* SE Logging: invalid file name */
#define      SEL_PATH_NOT_FOUND                     0xC1270008uL
/* SE Logging: path not found */
#define      SEL_INVALID_INIFILE                    0xC1270009uL
/* SE Logging: cannot find initialization file */


/*---------------------------------------------------------------------------*/
/*  WARNING: Do not use group codes above 0xC17F0000uL
*/
/*---------------------------------------------------------------------------*/

/*- Voeg Info-codes hier toe, geheel moet in commentaar blijven. De range
        van een InfoCode is 1..127 (dec), 0 betekent geen info-code.    ---*/

/*- Nota Bene: Don't use infocodes for new developments!
 *              Not compatible with SCODES
 */

/*------ ***InfoCode*** addition:

*** PARSER_FILE_OPEN_ERROR                       0xC12E0001uL
*** PARSER_UNEXPECTED_END_OF_FILE                0xC12E0002uL
*** PARSER_BUFFER_OVERFLOW                       0xC12E0003uL
*** PARSER_DIGIT_EXPECTED                        0xC12E0004uL
*** PARSER_UNKNOWN_LEXEME                        0xC12E0005uL
*** PARSER_SYMBOL_MISMATCH                       0xC12E0006uL
*** PARSER_WRONG_FILE_FORMAT                     0xC12E0007uL
*** PARSER_WRONG_FILE_ID                         0xC12E0008uL
*** PARSER_UNKNOWN_VERSION_NR                    0xC12E0009uL
1   Stage Position Table
2   Tilt Map
3   Stage Test Data

------------------- End  addition */

#else
#error file gloerr.h already included!
#endif /*  _GLOERR_H_  */
```