GenIe

Group 9 Design Report

Jelmer van der Graaf (s3172279),
Ernests Malnacs (s3126099),
Petru Manea (s3204766),
Levi Ockels (s2952017),
Estere Salina (s3126080),
Gheorghe Turca (s3197999)

Supervisor: dr. Lorenzo Gatti
Client: ing. Jan van Nieuwkasteele
2026

UNIVERSITY OF TWENTE.
Twente membranes
Sfi

Table of Contents

1. Introduction

1.1. Context and Use Case

The Soft Matter Fluidics and Interfaces group (SFI) and our client, the process engineer of the group - Jan van Nieuwkasteele[1], use high-speed (150-230 fps) microscopic cameras in order to visualize microscopic processes in (micro-)fluids in their experiments. Specifically, the experiment our software will be directly used for is to look at hydrogen bubble formation during electrolysis. There is an existing paper that investigates an acidic solution with a platinum anode to induce electrolysis, but this is quite costly as platinum is a rare element. A member of SFI researches the effects of different solutions and anode materials to find a cost effective alternative, currently opting for an alkaline solution with a nickel anode. The current setup used for this experiment, shown in Figure 1, uses a FLIR GS3-u3-23S6c-c camera for recording the bubble formation and movement processes and later analysis.

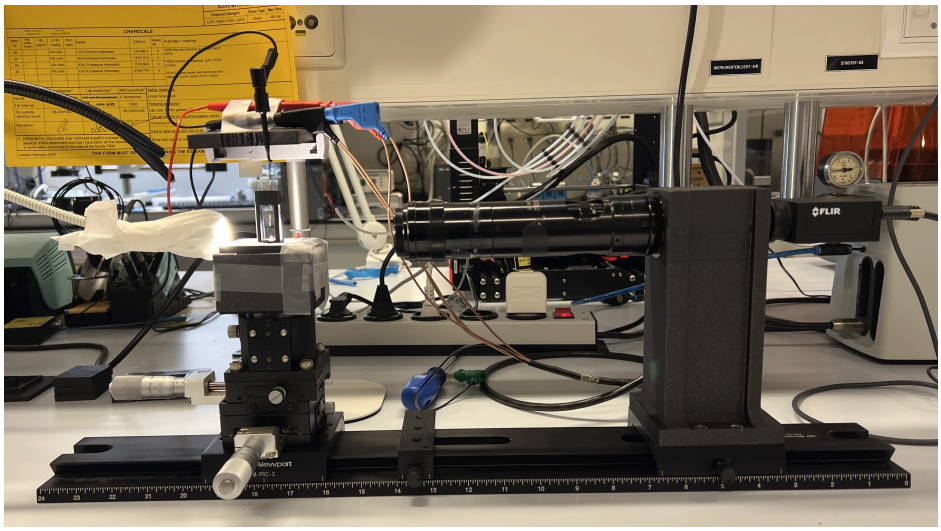

Figure 1 — Current experiment setup for electrolysis analysis

1.2. The Problem

The camera control software currently used to control the camera during the experiment, Teledyne SpinView, has multiple issues limiting its usability:

The first and main issue is inefficient memory allocation and video data saving at higher frame rates, causing the system's RAM to fill up with buffered video frames in mere minutes, even in cases where the system has 64GB of RAM. This causes the application to drop frames it cannot process, which causes bad footage with missing frames. This issue considerably limits the ability to record longer running experiments.

Secondly, the UI of SpinView seems to be more of an afterthought than a primary consideration. The application looks and feels more like a proof of concept of what is possible with Teledyne's SDK - Teledyne Spinnaker SDK.

Lastly, SpinView is built on the Teledyne Spinnaker SDK, a closed-source, and proprietary SDK. This means that, if the client ever wished to publish the application as open-source, they would possibly have to deal with licensing issues or restrictions. An additional reason to use open-source software is that we had no way of telling whether the SDK or the SpinView software was responsible for the issues regarding buffer mismanagement. As the camera supports the GenICam standard, a generic interface protocol for controlling scientific and machine vision cameras, we therefore decided to base our application's camera control part on a well-maintained, open-source project for control of GenICam cameras: Aravis[2].

[1] https://people.utwente.nl/j.w.vannieuwkasteele
[2] https://github.com/AravisProject/aravis

1.3. Project details

Our goal was to create a custom application that would allow our client to easily control the camera and record their experiments for extended periods of time. The UI had to be nonintrusive and intuitive, letting our client spend more time on the experiment they are recording, rather than spending time on figuring out how to change a camera setting. To put it simply; fix the aforementioned problems.

The project was initially called “Control Software to Allow for High-Speed Video Capturing and Streaming of a Microscope Camera” by the client. As the original name of the project is quite a mouthful, we have decided on a shorter name for our application: GenIe. This name comes from the name of the camera protocol that a lot of machine vision and scientific cameras use - GenICam.

The project was supervised by Lorenzo Gatti[3].

[3] https://people.utwente.nl/l.gatti

2. Requirements Elicitation

2.1. Capturing Requirements

In order to capture requirements, we asked the client to demonstrate how the current software was being used to see what the issues with it were and what features it offered. We recorded which functionalities are necessary by observing which features were used during the demonstration and asking the client which ones they would like to have present in the final product.

Later, in another meeting, the client showcased other similar software for different cameras, and demonstrated their use to make the desired outcome even more clear. This was an opportunity to capture some less critical requirements, as well as to have a brief discussion regarding recording formats. This prompted us to research the cameras' supported formats, and set them as requirements, since different formats result in different frame-rates for the camera, which is important for the use case of the client.

While examining the application that was supplied with the camera, we noticed major flaws with the user interface, such as:

- Nested popup windows (a pop up window opening from a popup window).
- Important settings "hidden" in secondary views, not on the main page.
- Unclear drop down boxes that performed actions without notifying the user.
- Unclear labeling of what buttons do (checkbox that said "Power", which turned off the camera and closed the application when clicked).
- No preview of the camera while adjusting important camera parameters.

2.2. Functional Requirements

High Priority

1. The program shall give a live preview of the camera's feed to the user.
2. The program shall allow the user to record video from the camera in RGB8, Mono8, Mono16 and the respective Bayer formats to uncompressed video files.
3. The program shall allow the user to select a region of the camera feed with a variable resolution to capture.
4. The program shall allow the user to record the Raw video from the camera to the hard drive without a time limit imposed by RAM (storage should be the limiting factor instead).
5. The program shall allow the user to choose the filename and location for the recorded video.
6. The program shall allow the user to adjust the following settings for the camera capture:
    - The resolution
    - The exposure time
    - The gain
    - The sensor's region of interest (as a rectangular portion of the camera feed)
    - The frame rate
    - The white balance
7. The program shall have accurate timestamps for each image frame.
8. The program shall be useable on a Windows 10/11 machine.

Medium Priority

1. The program can be compiled on a Linux machine (Ubuntu 22.04 LTS).
2. The program shall allow the user to set a LUT (Look-Up Table) for the camera sensor.

Low Priority (Bonus Goals)

1. The program shall allow two or more cameras to record video simultaneously, synchronized on time.
2. The program shall allow each connected camera to operate with independent settings.
3. The program shall allow the user to view logs of any interaction with the camera:
    - (Dis)connecting a camera
    - Recording format changes
    - Frame rate changes
    - Starting / Stopping recording

   - Failures caused by poor connection
   - Error messages and troubleshooting information
4. The program shall allow exporting log data to a file.
5. The program shall display a scale bar for measurements within the camera preview.

2.3. Non-functional Requirements

1. The program should not allow nested pop-up windows.
2. The program should limit the amount of unnecessary controls visible on the main recording interface.

2.4. Additional Requirements Gathered From User Testing

During the process of the project we gathered additional requirements from user testing. The process of these user tests will be discussed in Section 5. These requirements include:

1. A “set to center” button for the Region Of Interest
2. Show “time elapsed” during recording
3. Notifications for starting/stopping a recording
4. Showcase the amount of frames captured
5. Showcase the amount of frames discarded due to error
6. With the recording, save the frame rate at which it was recorded
7. An option to automatically increment file names
8. An option to choose a default save location
9. When choosing a file location, open the last used folder
10. A setting to preset the duration of a recording
11. A warning for low storage remaining
12. A function to reset to the default camera or go to the camera selection screen
13. OS level notifications as feedback for actions

3. Design

3.1. Handling High Frame Rate, High Resolution Video Saving

The main issue with SpinView is the RAM filling up. In our first meeting with the client, we quickly discovered that for each frame the camera was capturing, it seemed that a new thread was created to save it to the disk. The amount of switching between threads cost a lot of time which obviously limits the maximum throughput and thus the RAM could not be emptied up fast enough.

Our solution to this problem was to limit the amount of times the frame data was copied in our pipeline. In total, our code only copies data twice, once when we are displaying the preview for the camera, and once when we save the file to disk. Another issue that we ran into while trying to optimize writing data to disk was that by default files are written to disk in small chunks for smaller latency. However, in our case, since the frame rates of the camera are high, the cost of dividing the data into smaller chunks and saving them caused the disk to slow down such that it could not save the video data fast enough. The solution was to test different chunk sizes to find a large enough size which would allow for saving to disk efficiently to keep up with the video stream.

3.2. Design Of User Interface

The goal for the user interface was to create an intuitive and easy to use application. We first identified some issues with SpinView, namely issues with unclear navigation such as relying on multiple pop-up windows for accessing settings. Sometimes there will be a pop-up for a single, specific setting, or nested pop-ups, which, in our opinion and that of our client, adds unnecessary actions and disrupts the user's workflow.

One of the first things we thought about was the application being modular. We envisioned that some settings might be more important than others, which depends on the user's needs. Which is why we believe that showing everything at once is not necessary. The user should have the option to tab the panels and display only panels they deem necessary. Moreover, having a dockable interface would allow users with multi-display setups to detach panels and move them onto another screen, enabling more efficient use of screen space. We thus decided to divide the interface into panels, with each panel serving a specific functionality. Initially, we considered using the same UI library as SpinView, QT ADS. However, using this library would have required us to restructure the panels into separate ones, adding unnecessary complexity. We instead used dockable widgets from Qt, as they offered all the features we needed with simpler integration.

3.2.1. Layout of Components

For the layout of the UI, we tried to create something that would feel intuitive and familiar. This is why we chose to group together parts of the program that are somewhat similar in functionality. For example, all video related setting are placed on the left side. Settings that are adjusted via a slider are placed inside a scrollable widget, while settings that require more complex adjustments – the Region of Interest and Look Up Table – are placed together in a separate widget. Additionally, we thought that it made the most sense to place all recording related actions, such as starting and stopping the recording, underneath the live preview, as this is how it is also done on mobile phones for example. Lastly, we decided to place the settings related to storage (free disk space left and choosing a file path) underneath the start/stop recording button. It seemed intuitive that a person would choose a file path and then start a recording, so we placed the buttons close together. The main layout of the application is shown in Figure 2.

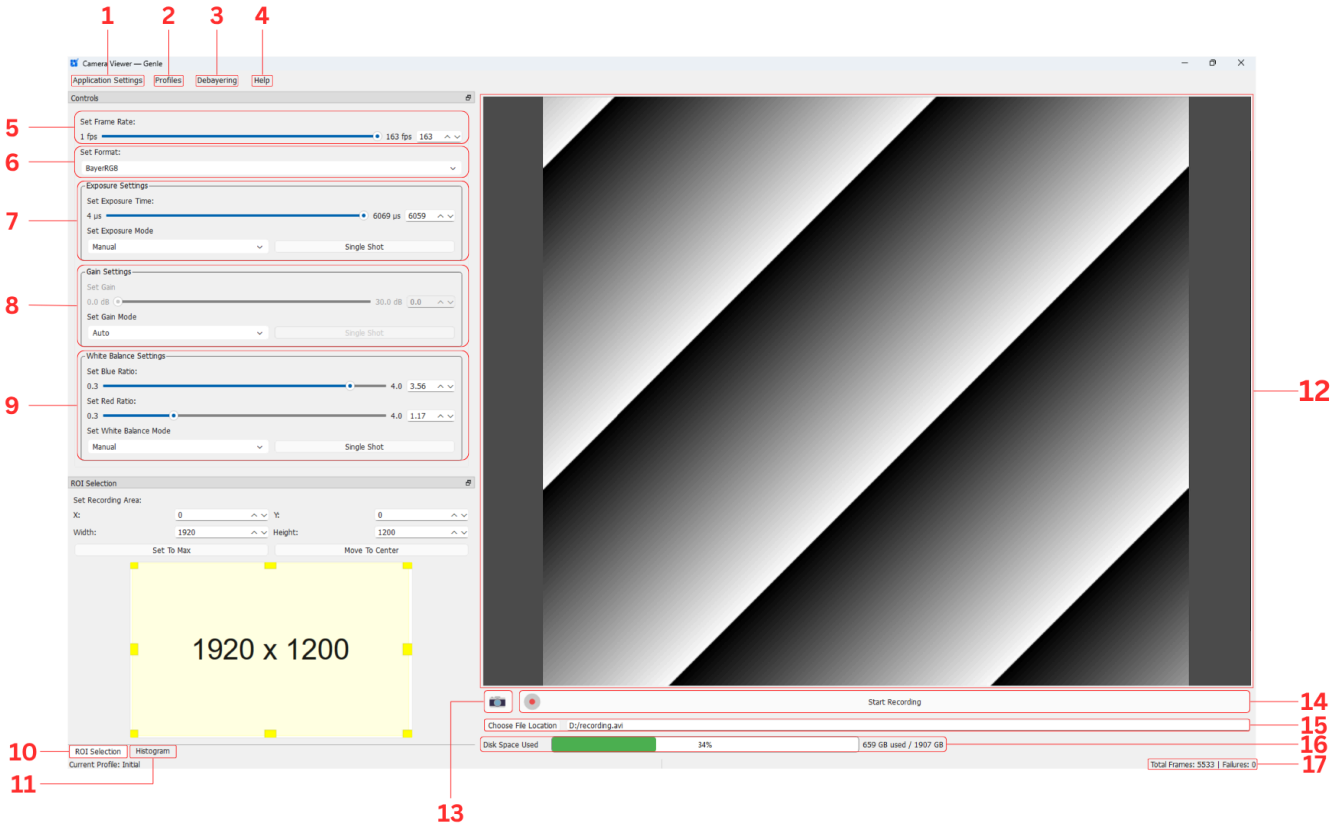

Figure 2 — Program layout with annotated features

Miscellaneous Settings

1. Application Settings
2. Profile Settings
3. Footage Debayering
4. Help Menu

Camera Settings

5. Frame Rate
6. Color Format
7. Exposure Time
8. Gain Levels
9. White Balance
10. Region of Interest (ROI)
11. Color Correction (LUT)

Capture controls

12. Camera Preview
13. Screenshot
14. Start/Stop recording
15. Save Location
16. Disk Space Tracker
17. Frame Tracker

3.2.2. Region of Interest Selector

In high speed cameras, Region of Interest (ROI) is used to select which part of the camera sensor gets recorded. This is useful since it makes it possible for the user to record only the area they are interested in without wasting space on their disk saving parts of the image that don't contain the observable experiment. More importantly, most high speed cameras allow the user to record at higher frame rates if they select a smaller ROI, making it possible to record high speed experiments.

For the ROI selector we wished to create an intuitive way to select which region of the camera gets captured. The existing solution only allows selecting the region of interest if the camera preview is turned off, which makes it difficult for the user to align the region of interest with the object of the recording. Our goal for implementing the ROI was to make it possible for the user to be able to easily align the ROI with the object of the recording, therefore a main requirement for the ROI selector was that it can be adjusted while the preview is running.

Another core requirement was to make the manipulation of the size and place of the ROI intuitive. The existing application utilized click and drag selectors, which only allows the user to roughly select the ROI and not fine adjustments. We decided to model the ROI selector to mimic the behavior of a long used method of selecting part of an image: a crop tool.

Our solution, shown in Figure 3, similar to standard crop tools, uses handles on the edges and corners of the region to either make the region larger or smaller. The yellow box is the region of interest that will be

recorded, and the dark gray box is the maximum size the region can be set to. The yellow selection box is thus enclosed in the larger gray box to mimic how the selected area would look like on the sensor.

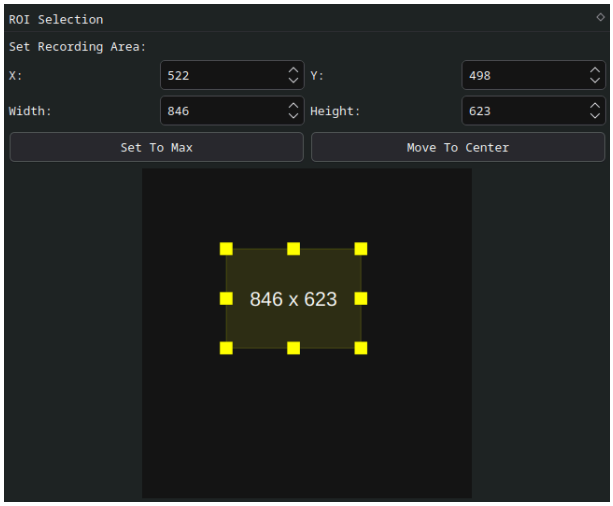

Figure 3 — Region Of Interest Selection

Like crop tools, our solution allows dragging the box when clicked in the yellow area, allowing you to align it with the object of the recording. In addition to the crop-like selector we also added input boxes at the top to allow the user to input precise pixel values for the region if required. For convenience we also added ROI centering and maximize buttons, which can be useful to reset the camera from a previous experiment, as well as easy alignment of the ROI if the object of the recording is in the center of the camera.

In addition, we added input boxes for allowing very precise inputs. In scientific experiments, often reproducible results are required and so it must be possible to recreate a specific setup. This will make that possible.

During the first user test we got the feedback that our implementation of the ROI was a noticeable improvement over the existing solution SpinView provided. While observing the user, we also noticed that it was immediately clear how the ROI should be used. One issue with the ROI selector was the scaling of the handles, the user had difficulty with the selectors when the ROI was small, therefore we decided to dynamically scale the handles based on the size of the ROI, making them smaller when the ROI is small and larger when the ROI is large.

3.2.3. Look up Table Selector

Look-Up Tables (LUT) are an essential tool for color correction in media processing. The idea being that the colors can be mapped to other colours. For videos, this means that LUTs can add contrast or reveal crucial details for a more in-depth analysis of the footage, which is especially useful for videos shot for research purposes. The way a LUT maps colors is by breaking up the bytes the color is saved as. For example, take the “RGB8” format, where each color is divided into 3 bytes, one for red, one for green, and one for blue. This byte value represents the range of 0 to 255 for each of the aforementioned colors, combining them into one color for us to see. The LUT can then map the color 0,0,0 (black) to 128,128,128 (grey), or any other combination. Furthermore, the camera we were issued has its own LUT operate on twice the amount of values as given, to have added precision in gradients. Following the “RGB8” example, this means that we now have values 0 to 511 for each of red, green and blue to map to another value.

Another functionality of LUT tables is to shift the spectrum. If any part of the color spectrum isn’t important, you can shift the colors to not include those particular colors, or smooth over them with the colors you are actually interested in. Unfortunately, we did not implement this part of the LUT.

The SpinView program uses a graph to display the LUT and the color histogram (the frequency count of each color value to see the effects of the LUT) and allows the user to draw freely using a Bézier curve or a spliced curve to then adjust the standard linear mapping. This can be done for each spectrum color of red, green or blue for RGB8 images separately or all together. It also has options for grey scaling (Mono8).

We decided to use a Bézier curve projected onto a histogram and to adjust all the colors together. This means that our program cannot achieve the same precision and mapping combinations as other software. We cannot map 0,255,0 to 0,0,255 in the RGB8 format for example. We chose in favor of a single graph as opposed to three or four since it was deemed part of the MVP and we couldn't foresee if we had enough time to implement the more complex version. Moreover, there was a worry that it would make the UI excessively elaborate and overwhelm potential users.

We did however make the color histogram, when supported by the current color capture of the video, differentiate between red, green and blue and reflected that with the respective colors. Now users can intuitively view which colors need to be corrected for.

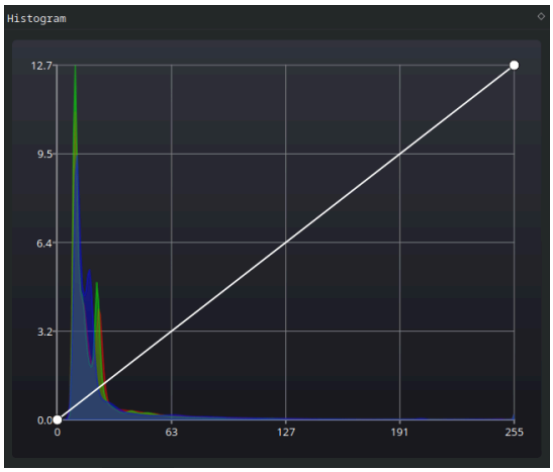

Figure 4 — Histogram and Look-Up Table Controller

3.2.4. Recording Controls

The recording section combines the file selection, the start/stop recording button, and disk monitoring, as shown in Figure 5, since we wished to cluster related settings so that the users do not have to search far for related settings and actions.

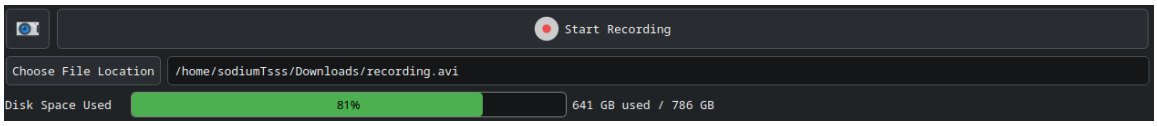

Figure 5 — Recording Control Section

For selecting the output location for your recordings, the section contains a "Choose File Location" button that opens the operating system's file management application to choose where to save the file. The file manager tries to open in a standard directory –like "User/Documents/" on Windows, or "home/User/" on Linux for convenience, as most people save projects in these directories. These features align with other applications that interact with files.

In addition, there is a file path field to the right of the button, where the user can manually input the file path they wish to save at, allowing the user to quickly change their recording names from e.g. "recording.avi" to "experiment.avi" without having to go through the full file selection process. We did not want to make the user select the folder location again to simply change the name of the recording file.

Once a valid file path is selected, the screenshot and "Start recording" buttons will become enabled. If the path is not valid, i.e. they don't have permission to write to that location, the recording and screenshot buttons will be disabled.

When selecting a file, we also added an option for auto-incrementing the file names. When enabled, the program will automatically increment the file name, by adding a number in brackets as a suffix if a file with the same name already exists in that folder. For example, if the file is called "recording.avi", when pressing record, the next file will be created as "recording(1).avi". We added the auto-increment function after the

first user test, where we received feedback requesting an option for quickly recording multiple videos without having to manually change the name or overwriting the recording each time.

Another function implemented based on feedback from the user test is the ability to set a default file directory, which is loaded upon startup of the program and is saved between sessions. This makes it easier to work on a project, as this allows for setting the working directory once where afterwards you only need to change the file name, or to set it to a custom base location and save time switching between the locations multiple projects.

We also added a disk space usage bar to make it clear how much space is left for recording. While this was not a listed requirement, we felt that it made it more intuitive for the user to see how much free space is available on the mounted disk. The disk space usage bar also displays the percentage of used disk space. Next to it on the right, used space is listed in gigabytes for a more specific representation of how much storage space is left.

The used space on the bar is shown in green until the disk reaches 90% usage to indicate that the disk has an acceptable amount of free space left. After that, the color turns to orange to indicate the storage is close to becoming full. We thought of this as an early warning so the user has enough time to take an action. When the disk reaches 97% usage, the color will change to red warning the user that their storage device is about to be full and that they should soon stop recording. We selected this specific threshold because the recording speeds can become quite fast, sometimes reaching 300-500 MB/s, meaning the remaining storage will be filled in seconds if the volume is small. We acknowledge that this design is not fool proof as a small disk volume can mean that 3% is less then 15GB (the disk is less then 500GB). Assuming that the speed of data is 500MB/s, that gives the user 30 seconds to shut off the program.

However, the program will automatically stop the recording if the available space on the mounted disk is less than 1GB, so the disk space will never run out. We picked 1GB because of how high the data rate for recording is at higher resolutions and frame-rates, this gives enough time and space for our application to signal our recording code to stop recording and to save the current frames that are in transit to disk, as well as correctly saving the video container such that it has valid header data so it can be opened and viewed. This is because we found out that the computer usually crashes when their operating system is also located on that disk and a user completely fills up their disk, which could result in data loss.

3.2.5. General Camera Controls

For controlling the more general settings of the camera like exposure, frame rate, video format, gain, etc. we decided to go for a modular approach as shown in Figure 6, as different GenICam cameras might support control of different sets of features. For example, some cameras allow the user to control the frame rate, while others don't so, based on what the connected camera supports, we hide or display the available features in the control section. This limits any confusion a user could have about which settings are actually useable, as only the settings they can control will be displayed.

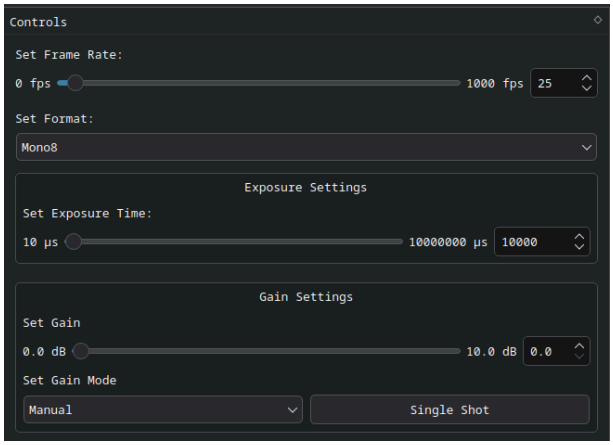

Figure 6 — Camera control section

An example of how hiding features works is the difference between the exposure settings box and the gain settings box. The test camera used for Figure 6 does not support Automatic or Manual modes for the exposure setting, therefore the mode selection box that can be seen for the gain settings is hidden for the exposure settings section. Also note the Single Shot button next to the mode selector, which tells the camera to try to adjust the setting to what it deems as optimal, helping the user select an estimate value automatically.

Each section of the controls allows the user to change that particular camera setting. For settings that have ranges, like gain and exposure, we have added both sliders as well as input boxes for inputting specific values. This allows the user to first easily select the value they wish to use using the slider, and, if they wish to adjust it more precisely, they can use the input box.

Additionally, sometimes changing values or formats can influence the maximum of other values for certain settings. For example, if the user changes the ROI (see Section 3.2.2) to a smaller area, the maximum frame rate will increase. Thus, the slider of the frame rate settings will automatically update to reflect these changes. Conversely, making the ROI larger will make the maximum frame rate smaller, decreasing the maximum frame rate. In this case, if the frame rate that was previously set is not inside the new range, the new value will be set as close as possible to the old value. The reason why we implemented this edge case this way was to ensure legal values and we thought it to be easiest to code.

These settings can also be saved or loaded by using so called “profiles”. When creating a profile all current values of the generic settings will be saved, with default/safe null values for settings that are unsupported by the active camera. The same is true for loading profiles; It will try to load all settings, but unsupported settings are ignored. It is of course also possible to update or remove a profile, and the current / last loaded profile is shown in the status bar on the bottom left of the application. Furthermore, it is also possible to set a default profile which will be loaded upon startup of the program, so you don’t have to manually load a profile every time you start the program when working on a project. There is also a special “Initial” profile that is hardcoded into the program and cannot be updated or removed by the user, as this is intended as a safe fallback profile to reset to safe default settings when wanting to return to a default view or if the current settings causes issues.

3.3. Designing Against User Error

Since experiments that are being recorded take time to set up, we want to prevent any accidental loss of data due to user error. One example could be the user closing the application while making a recording, causing the recording to be lost.

The most important part to secure against accidental user error would be anything that handles recording or recorded data. For example, the previously mentioned issue of accidentally closing the application while recording. This could lead to the recording becoming corrupted or missing some essential data, therefore we have implemented pop-ups with warnings, that let the user know that something they are about to do could be destructive see Figure 7.

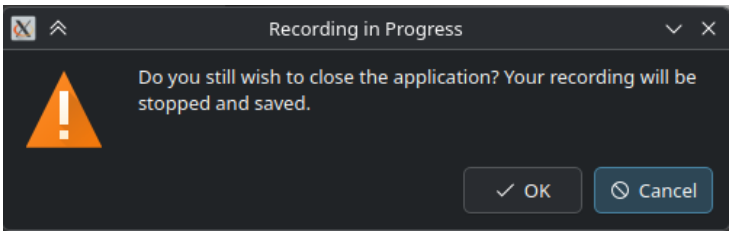

Figure 7 — Exit warning pop-up

For all pop-ups of similar kind that prevent the user from doing something destructive, we have carefully selected the default button - the button that is usually highlighted by the operating system as seen in blue in Figure 7 as well as the button that gets pressed if the user presses enter, to correspond to the least destructive option, of usually doing nothing, and just canceling the operation. For these pop-ups we have also made the “close warning window” button cancel the destructive operation as well.

Another important issue arises when the user has limited space on their disk. We have created a monitoring system while recording that constantly is checking how much space the disk has left. If the user starts a recording and their space is below 2 GB a warning will ask if the user wants to record anyway. If the user is recording, and their disk space reaches 1 GB remaining, the recording will automatically stop and correctly save, preventing the user from losing the recording or fully filling up the disk by accident. We added the 2 GB remaining warning for specific cases where the user wants to capture a short video, use low frame rate, or records at a small resolution and the video allow to fit reasonably within the limit.

In general across the application we have implemented the enabling and disabling of various components of the UI such that the user can not cause an invalid state of either the recording process or the camera. For example in Figure 8 you can see that the start recording, and screen capture buttons are grayed out, meaning that they are disabled. This is due to the user not having selected a file location to save the recording to, therefore the user cannot start a recording with no file selected. In other areas of the UI we have applied this same design pattern, like camera settings selectors, disabling input for settings that cannot be changed during recording, as well as some settings that are adjusted automatically which first need to be changed to manual mode.

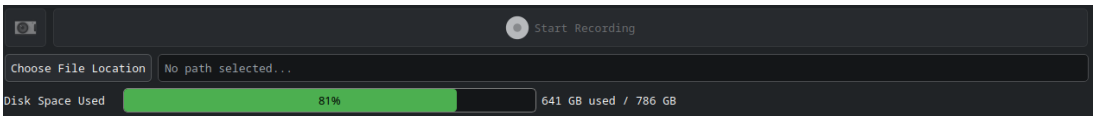

Figure 8 — Disabled Elements of Recording Section

3.4. Image Processing and Debayering of Raw Recordings

After exploring the different formats of the camera, we saw that the camera outputs the fastest frame rate video in its Raw format. This Raw format is an unprocessed output from the camera, meaning that it has not yet had its Bayer filter mosaic removed as illustrated in Figure 9.

For this video to be useful in analysis it has to be Debayered to remove the mosaic on the image. For this we tried two algorithms that we parallelized on the GPU. The first algorithm we tried was bilinear interpolation (Figure 10). The other algorithm we tried was Malvar-He-Cutler Linear Image Demosaicking (MHC) [1], which is an extension on bilinear interpolation that attempts to reduce the blur created by bilinear interpolation, as well as sharpen the edges of objects by applying various filters to the image (Figure 11).

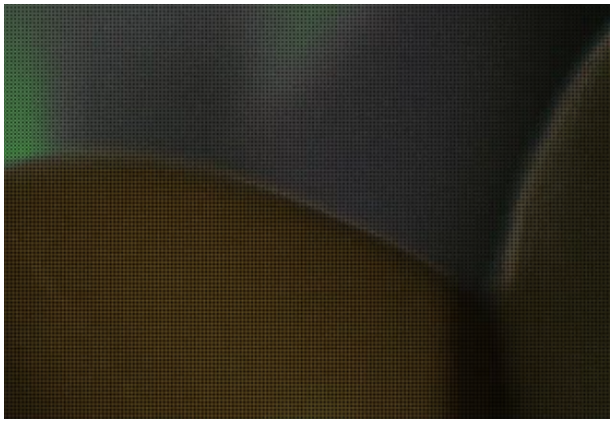

Figure 9 — Raw camera output

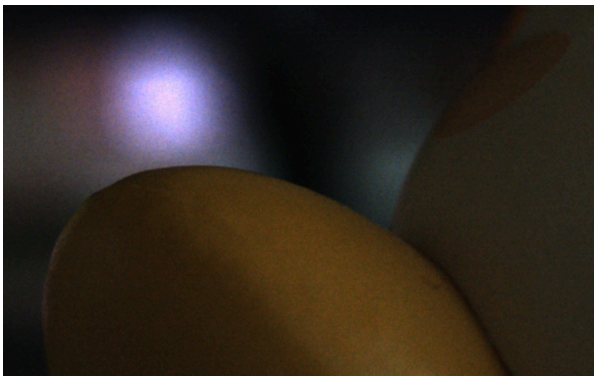

Figure 10 — Bilinear interpolation

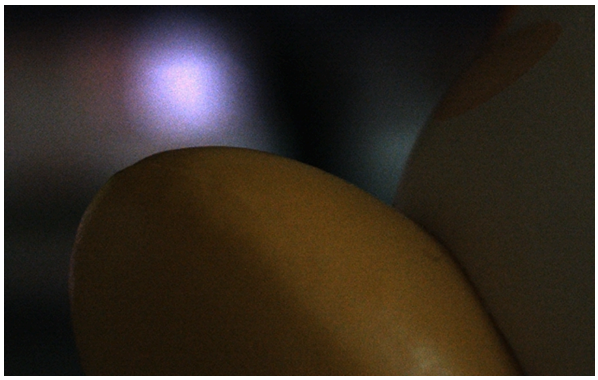

Figure 11 — MHC Debayering

We compared the two algorithms for Debayering using a GPU profiling tool called RenderDoc, which allows us to measure how long it took the GPU to process our Raw images. When measuring Bilinear Debayering, we observed that one frame took 26.84µs, while MHC took 32.44µs, giving a ~6µs difference. So it is marginally slower to use MHC, but since the limiting factor of recording is disk write speed, which is in the order of milliseconds, the difference between the two algorithm speeds is negligible.

Comparing the resulting images of the algorithms, one can see that the MHC result has better defined edges, as well as a sharper image with less artifacting around the edges compared to Bilinear interpolation, as seen in Figure 11. Since the camera will mostly be recording high contrast scenarios that are well lit, we decided to choose MHC Debayering as our algorithm, since it would yield the best separation of recorded objects, as well as a sharper image.

The live camera preview uses the MHC algorithm to yield a live colored and Debayered image in the preview, so that the user can more easily align the camera and monitor the process without having to manually apply Debayering first. This does not cause any issues with recording speeds, since it does not interact with the disk.

We tested if it was possible to apply this Debayering process while the user is actively recording the file, but implemented tests yielded unreliable recording speeds due to having to write three times as much information to the disk compared to the Raw output. This would cause the buffer overflow issue that this project was aiming to solve, namely - high priority requirement 4.

Our solution for this problem was creating a dedicated application that is used to Debayer Raw recordings after they have been recorded and saved to disk. This application is purely a command line application, and does not have a UI. Our decision to create it this way was prevent having to integrate the Debayering process in our main application, since that would require managing a new thread alongside the other parts of the application. Instead we make our main application launch the Debayering application for each Debayering job, and let the operating system handle threading and resource allocation for the particular job.

We created a queuing system for our application as seen in Figure 12, that launches this Debayering application for each job in the background.

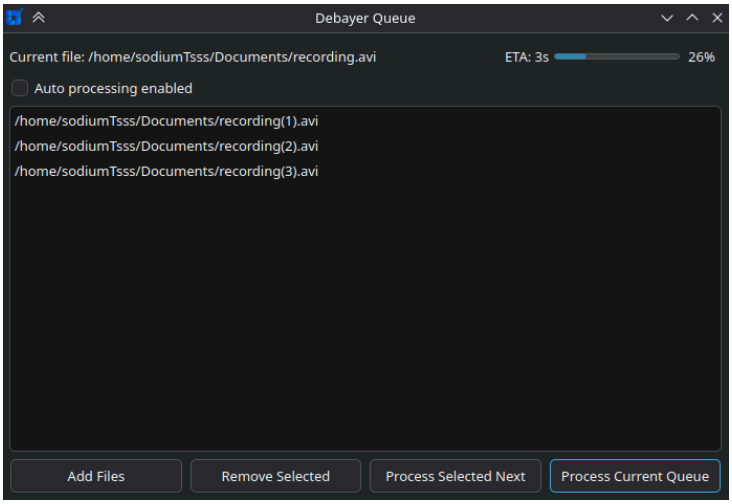

Figure 12 — Debayering Queue

When a user makes a recording in the Raw video mode, the recording is added to the queue of files that need Debayering. The user can then click the “Process Current Queue” button, which will automatically start Debayering the files in the background in order of the queue. While processing the files, the queue window shows the current progress of the file being processed as well as an ETA for when the process should be finished. In the worst case scenario for a 10 minute recording at full resolution of 1920x1080 at 164fps, the Debayering process can take around five minutes, so roughly half the recording time. This is however highly dependent on the hardware that the application is running on, since disk write speed is our main limiting factor. For files that are smaller in resolution than the previous example, the Debayering times decrease substantially, since the required disk writing bandwidth is reduced due to the file size being smaller.

Along with “Process Current Queue” the user can also enable “Auto processing” which will make the application automatically start Debayering jobs after a Raw recording is made, which is ideal if the user is making multiple short recordings in succession, since for small recordings the Debayering time is short and does not utilize much resources, thus it will not interfere with the recording process.

If the user has other videos not currently in queue they wish to Debayer, they can use the “Add Files” option to add recordings from disk to the queue. If the user wants to reorder the files in the queue, they can select the file and click “Process Selected Next” which will move the file to the top of the queue, to be processed next.

The user can still make recordings without any issues when there are unprocessed files in the queue, since the queue saves the file paths of the unprocessed files and the processing is only started when either the user manually starts it, or if the “Auto Processing” option is enabled, automatically after a recording is stopped.

While files are being processed, it is highly advised for the user not to make high resolution and high frame rate recordings. Since the Debayering process is resource intensive, this could cause the recording application to start discarding frames, as it doesn't have enough system resources to save them to disk fast enough.

Due to time constraints, we have not managed to add a warning to notify the user that files are being processed if the user tries to start a recording. Ideally this warning would only appear when files are being processed and the user is trying to record a high-resolution, high frame rate recording, since it is still viable to record with a lower resolution and frame rate while the Debayering processes are running.

Additionally we made sure that in cases where the system or application crashes, the power is lost, or simply the user closes the application, that the original video file is preserved. We only remove the original unprocessed file when we have confirmed that the Debayering process completed successfully. Therefore, if during the process of Debayering something unexpected happens, the original recording file is still preserved.

3.5. App Icon

The application logo as shown in Figure 13 is built around a stylized letter "G", derived from the name of the application: GenIe. The "G" is designed to resemble a closing camera shutter, reflecting the app purpose. Additionally, small bubbles rise from the "G", referencing the experiment of visualizing hydrogen bubble formation during electrolysis, conducted by the Twente Membranes group.

Figure 13 — Application icon of GenIe

3.6. User Manual

The "Help" menu (number 4 in Figure 2) will allow the user to open up the User Manual. It was written to answer some of the questions we received during user testing regarding how to install the application and particular use cases of it. The manual aims to help anyone of any skill level use the application. Although, for some parts of the manual, deeper technical knowledge may be needed, for example, about what drivers are.

Additionally the menu contains an "About" section which lists the purpose of the application as well as the authors.

4. System Testing

As unit testing would have been difficult to implement for our project, due to the project being mostly user interface focused with few back end components that require either a real or virtual camera to function, we decided to perform system testing to verify that the application functioned correctly according to user requirements, as well as user testing for the UI which will be described in the next section.

For system testing, we had tested the system while developing it as well as after completing major features. We will describe the main system tests we performed while developing our application in this section.

4.1. RAM Usage Test

Testing that RAM usage usage didn't increase during a long recording was important for this project, as the possibility of making long recordings was a primary requirement of the application. Without RAM issues, the length of the recording depends solely on the available free disk space.

We tested RAM usage by setting up a recording with the highest camera settings, in this case a resolution of 1920x1080px and a frame rate of 164 FPS. After this was set up, we ran a 15 minutes long recording, while a system monitoring application was also open on the computer to measure system resource utilization.

The result of the test was that RAM usage only increased at the start of the recording, and afterwards it stabilized. RAM usage stayed constant with no change during the entire recording, confirming that our application can record for extended periods of time without filling up RAM.

4.2. Frame Loss/Error Test

Testing frame retention - testing if any frames sent from the camera were lost by our system - is also important, as dropped or discarded frames will result in a video with inconsistent timing between frames, making it hard for the user to do any time critical measurements using the recorded videos. Therefore, we performed a frame loss test to see if any frames from the camera are lost or discarded during a long recording. We set up a failure counter in the program for this, that keeps track of discarded or lost frames (Figure 14). This test was performed at the maximum settings of the camera, with the same set up as the previous test.

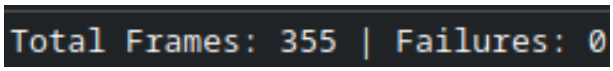

Figure 14 — Counter of total frames and frames that were lost or discarded

The result of the test was that over the 15 minute recording period there were no failures, confirming that the recordings we were making contained all of the frames sent by the camera were correctly being saved to disk.

4.3. Frame Timing Consistency Test

During one of our user tests, the user mentioned that they performed a "stopwatch test" with the previous recording application and noticed that after a few minutes the video time was out of sync with the stopwatch being recorded. This indicated that the video with their application was not being saved correctly, leading to drifting of timing over the recording period. This makes it difficult for the user to make time critical scientific measurements using the recordings, therefore we decided that this test would also be useful for testing our application.

The basic set up of this test was different than the stopwatch test. We created a small program for a micro-controller, in our case a RP2350, that precisely blinked a LED on and off every 250ms. We pointed the camera at the LED when we made our recordings. For our application side of the test, we used various frame rates. We tested both lower frame rates (30 and 164 FPS), as well as high frame rates (1500 and 2000 FPS). This was done due to possible differences in the way the video was being saved at different frame rates. After the recording parameters were set up, we started a 15 minutes long recording.

After the recording time had elapsed, we wanted to verify if the time had not drifted. For this we measured the amount of frames between the blinks of the LED at the start of the video and also at the end of the video. If the amount of frames was different, this would indicate that time drift had occurred. At each end

of the video we measured multiple blinks to take an average for that particular part of the video to get rid of potential variance in the blink timing, although this was not necessary in the end since the frame count between the blinks was the same.

After measuring the beginning and end of the video we compared the frame timing. For all frame rates tested we observed a time drift of one frame, which over a 15 minutes video is negligeable, and could be caused by what was counted as the start and end of a LED blink. Therefore, this test verified that our application records videos with negligeable time drift, allowing the user to make time critical measurements with our applications recordings.

5. User Testing

In total we conducted two sessions of user testing. The first was aimed at identifying any missing features that we may have overlooked and took place on week five of the project. We presented our first implementation of the project with the required features and asked the user to try the application out with their normal workflow. Then we asked the client their thoughts about the application and if there are any missing or ambiguous features.

We performed a second user test around week eight, after implementing the feedback from the first test. We once again instructed the client and another potential user to try using the software with their normal workflow. After the test, we asked for the participants opinions on the application.

5.1. First User Test

Once our program had reached the state of having all high and medium priority features, we conducted our first user test. From this test we received a lot of very useful feedback about very simple features that we had overlooked. From this user test we elicited these features that still needed to be implemented:

- Rename BayerRG8 to RawRG8
- Add a “set to center” button for the region of interest
- While recording show the time elapsed
- Add a way to notify the user that the camera started/is/stopped recording
- Add a way to show how many frames were captured and how many were discarded due to errors
- Add icons to the start and stop buttons
- With the recording save the frame rate at which it was filmed
- When clicking choose file location open the last folder that was used/selected
- Instead of overriding an existing recording automatically make a new file by incrementing a number in the file name
- Add a setting to set a default save file location
- Allow the user to change the chosen file location by clicking on the displayed file path in addition to the button
- Separate choosing the save location from choosing the recording name
- Add a setting to preset the duration of a recording (low priority)
- Add a warning when there is low storage for making a new recording
- Check frame timing and time drift (Timer test)

5.2. Second User Test

During this test, we noted down any comments that the users had about the application, as well as our observations about their behavior during the process, which helped us identify some problem areas within the application.

The explicitly stated high or medium priority features or details that were missing or were unclear:

- Add a function to reset to the default camera / go back to camera selection screen
- Add OS level notification as feedback for actions
- Improve readability of items in light mode (Specifically ROI)
- Improve image / screenshot button feedback
- LUT histogram graph usage / meaning not completely clear
- LUT histogram reset and / or improved controls information
- Possible Debayer queue processing interference with recording

Possible issues and improvements we observed:

- Some items seemed quite small on the user’s screen
  - Change sizes of elements in the recording section
- Recording feedback could be improved
  - Not always clear if a recording is active
- (De)active state of the settings wasn’t always clear
- Current icons for items seem to be OS dependent

- Debayering progress could have better feedback

Lastly, we noted down some low priority improvements as well:

- Add a custom application icon
- Option to disable CMD / terminal prompt when opening application
- Add tooltips to settings to improve clarity

6. Final List of Features

We managed to implement all but one of the high and medium priority functional requirements, however, we did not have enough time to work on bonus/low priority features, except for one. Below is a list of the functional requirements that we did and did not manage to implement.

6.1. Implemented Features

High Priority

1. The program shall give a live preview of the camera to the user.
2. The program shall allow the user to record video from the camera in RGB8, Mono8, Mono16 and the respective Bayer formats to uncompressed video files.
3. The program shall allow the user to select a region to capture from the camera.
4. The program shall allow the user to record Raw video from the camera to the hard drive without a time limit imposed by RAM (Storage should be the limiting factor).
5. The program shall allow the user to choose the filename and location for the recorded video.
6. The program shall allow the user to adjust the following settings for the camera capture:
   - The resolution
   - The exposure time
   - The gain
   - The sensor's region of interest (as a rectangular portion of the camera feed)
   - The frame rate
   - The white balance
7. The program shall be useable on a Windows 10/11 machine.

Medium Priority

1. The program shall be functional on a Linux machine (Ubuntu 22.04 LTS).
2. The program shall allow the user to set a LUT (Look-Up Table) for the camera sensor.

Low Priority

1. The program shall allow exporting log data to a file.

Additional Features

1. A "set to center" button for the ROI
2. Show "time elapsed" during recording
3. Notifications for starting/stopping a recording
4. Showcase the amount of frames captured
5. Showcase the amount of frames discarded due to error
6. With the recording, save the frame rate at which it was recorded
7. An option to automatically increment file names
8. An option to choose a default save location
9. When choosing a file location, open the last used folder
10. A setting to preset the duration of a recording
11. A warning for low storage remaining
12. A function to reset to the default camera or go to the camera selection screen
13. OS level notifications as feedback for actions
14. Add a custom application icon
15. Option to disable CMD / terminal prompt when opening application

6.2. Features Not Implemented

High Priority

1. The program shall have accurate timestamps for each image frame.

Low Priority (Bonus Goals)

1. The program shall allow two or more cameras to record video simultaneously, synchronized on time.
2. The program shall allow each connected camera to operate with independent settings.
3. The program shall allow the user to view logs of any interaction with the camera:

   - (Dis)connecting a camera
   - Recording format changes
   - Frame rate changes
   - Starting / Stopping recording
   - Failures caused by poor connection
   - Error messages and troubleshooting information
4. The program shall display a scale bar for measurements within the camera preview.
5. Add tooltips to settings to improve clarity

7. Contributions and Process

7.1. Individual Contributions

For the most part, we didn't formally decide on specific roles for everyone. Together we decided on a list of things that had to be done and team members naturally chose the tasks that fit their expertise. Throughout the project the main roles of team members were:

- Jelmer: Histogram, Profiles, Picture Function, Code Quality & Quality Assurance.
- Ernests: Team Coordinator, Presenter, Programmer for main camera interactions in code.
- Petru: User testing, Quality Assurance, worked on the application manual.
- Levi: Initial design of class structure, designer and implementor of LUT functionality, problem solving on various topics, overseeing and writing the manual.
- Estere: Worked on the front end of the UI, writing reports, making presentations.
- Gheorghe: Worked on the front end of the UI, specifically the camera recording controls, app icon, worked on the report and manual.

7.2. Process

For dividing up our work we used two week long sprints, with the end of a sprint featuring a peer review session where we presented our work from the previous two weeks. Each sprint we had decided on specific milestones that we wished to accomplish. In the table we also included a date by which we had intended to complete the specific milestone.

Milestone	**Sprint**	**Date**
Proposal & Proof Of Concept	Sprint 1	18-02
Basic Viewing And Recording System	Sprint 2	11-03
Camera Control Options	Sprint 3	25-03
Documentation & Bonus Features	Sprint 4	08-04
Final Program	Sprint 5	16-04

While we managed to accomplish all of our intended milestones, in the ended we switched Sprint 4 and Sprint 5 around, as we had completed the final program before we started writing the report.

During the project process we made sure to schedule weekly meetings with our supervisor, in order to ensure that we are on the right track and get feedback on our ideas. Additionally, we kept in contact with the client throughout the design process, especially during the beginning, for requirement elicitation, and near the end, for user testing.

For organizational purposes, we had elected Ernests Malnacs[4] as our primary contact person for communication with the supervisor and client.

[4] e.malnacs@student.utwente.nl

8. Future Work And Expansions

8.1. Improve compatibility with other GenICam cameras

It should be possible to use the application for other cameras that support the GenICam interface, as the main camera controls are built using Aravis. During the project however, we only had access to one camera for testing. We did do some very quick tests during the final few days of the project, with about three other cameras to gauge how well our application works on other cameras. Initially there where a couple of small breaking bugs, but after fixing these it seemed to work fine for the most part, albeit with some small issues: Some settings displayed weird values (e.g. gain showing huge values, probably due to the camera using a different unit for time) or manufacturers that don't fully comply with the GenICam standard sending unrealistic limits (e.g. a camera stating that it supported up to 1.000.000 FPS), causing confusion and reduced usability.

Future work could thus conduct further testing using different hardware and generalize some of the settings, or improve the handling of edge cases if manufacturers do not fully comply with the GenICam standard.

8.2. Built-in Measuring Tool

During a discussion with our client, we received feedback that a tool which allows for measuring objects in the application preview is a desired feature. The client showed us a different microscope with a different application, which allowed the user to calibrate a measuring tool against a known size, in the client's case, a glass ruler. The calibration would work by putting the glass ruler in the camera frame, then selecting the calibration tool, and selecting the distance between the ruler marks. Then the user would manually enter the real distance between the ruler marks. We think that if one would implement this solution, a way to relate distance on screen to real distance could be by applying a pixel to real distance ratio, basically giving each pixel a real distance value.

We did not tackle this problem due to a few factors:

- Camera lenses create a slight curve around the edges of the image called lens distortion. This results in an issue where measurements are not uniform across the whole image. A solution to this would be to allow the user to set a lens distortion correction map.
- Applying an overlay showing the scale bar causes the same issues as will be discussed in Section 8.4. Applying the overlay would mean doing so on the GPU, requiring then to also get the processed image back from the GPU to save it to disk.
- Creating a repeatable set-up with a non-static microscope set up. Since calibration requires for the user to align their glass ruler at the same distance as the observable experiment for the measurements to be accurate, a way to repeatably align the glass ruler should be devised.

8.3. Improving LUT Functionality

One limiting factor of the current software is that the LUT channels for sub-colors of multicolor formats or the illumination (gamma) are all changed at the same time instead of separately. In our implementation, the class that counts the pixels of the frame to show in the histogram is also the same class that renders the histogram graph. Obviously, if one wants individual graphs, step one would be creating classes for each of these channels. A class can then be created to count the frames with separate widgets for each graph that deals with a sub color or the illumination channel. During testing we did run into a problem where, at least with the specific camera we were issued, we either had to set all LUT channels together at the same time OR set one entry of one array for all entries of all channels in the table. If not done this way, undocumented behavior occurred and it would not set the LUT properly on the camera (e.g. when the illumination array was left empty the Red table would not update).

Secondly, there should be a way to set the values more precisely to allow for more accurate color correction. Rather than just simply dragging a curve, the user should have an option to, for example, manually input the specific desired values.

Thirdly, the ability to export the LUT values of the camera that were set could be implemented to allow specific scenario's to be accurately re-created, which is paramount for scientific experiments.

Lastly, cutting off part of the color spectrum is not intuitive, due to the use of a Bézier curve for the LUT controls. Adding a function to change the curve to a smaller part to the graph (e.g. changing the bounds of the curve) could be realized.

8.4. Live Debayering

Our current implementation of the software requires the user to add their recording to a queue in order to apply Debayering to it after the recording has finished. This takes time, thus a nice quality of life improvement would be to implement live Debayering, which would be happening alongside the recording process.

The main problem that we encountered in the process of trying to implement live Debayering was that it added too much time to the recording pipeline. This resulted in dropped frames at higher frame rates and resolutions. The main bottleneck is getting data from the GPU to the CPU side for writing to disk, which adds just enough time for our system to fail.

We had tried adjusting our pipeline to double buffer on the GPU such that a frame could be processed while the previous frame was being written to disk, but this barely helped, possibly due to the high frame rate. A solution that could possibly work is to switch the GPU pipeline to Vulkan, such that you can have more direct control over how buffers are shared between the CPU and GPU side. One might also try to implement even bigger buffer counts on the GPU side, on the order of 10 or even 100 frames, since most recent GPUs have high amounts of VRAM, though we are not sure if this would work due to high frame rate requirements.

The current queuing system could also have the following improvements:

- Ability to pause the Debayering process
- Processing multiple files in parallel, if allowed by system resources
- Option to keep original unprocessed video file
- Drag and drop reordering files in the queue

8.5. Timestamps

Although timestamps were initially listed as a high-priority requirement, we did not manage to implement them. Each frame was intended to be saved with an embedded timestamp from the camera, which was meant to help with determining the experiment's timeline. In order to retrieve the timestamps from the camera it's possible to use GenICam's Chunk Data, which isn't an issue. However, actually figuring out how to "burn in" the timestamps on each frame may require more effort. Because, in order to "burn in" live timestamps you probably would need to utilize the GPU, which brings us back to the same issues as with live Debayering.

8.6. Simultanious Recording With Multiple Cameras

The code background we have created has been designed to be able to handle multiple cameras, therefore adding support for recording with multiple cameras would most likely not be too difficult. A concern we have for this though, is that recording with two cameras at maximum resolution and frame rate might overwhelm the disk data is being written to, especially on older systems, but we think it is feasible to record with two cameras with careful execution.

A more interesting aspect of multiple camera recording would be synchronizing cameras. With our current implementation, since the cameras need to be in a "recording mode" to display the preview, there would not be a way to reliably tell the cameras to start recording at the same time. Therefore a computer side implementation most likely would be the better way to proceed. The cameras also support GPIO input which allows the user to connect signals to the camera that tell it to for example start a recording or change a setting, which could be used to synchronize the cameras through hardware instead.

Another challenge with recording and controlling multiple cameras at once is the issue of synchronizing (or not synchronizing) the settings for all cameras and how best to handle these settings and camera

previews in the UI. It is not obvious whether settings should be synchronized between all cameras or if each camera should have independent settings, after all, it is possible to argue for and against both; synchronized settings make it easier to set up a scene and reduce the number of unique UI elements, on the other hand, it is possible the the cameras do not support the same settings or each cameras could need different values. Perhaps a hybrid approach could be used: By allowing the user to synchronize certain settings between all cameras (such as FPS and format) it would be possible to have the best of both worlds, although the current state and effects of synchronization should be well presented to prevent confusion, mishaps, and frustration.

9. AI Statement

We hereby declare that no generative artificial intelligence (AI) tools were used in the conception, development, analysis, or writing of this project report. All work presented is the result of our group's independent effort and understanding.

The sole exception to this declaration is the formulation of this AI usage statement itself, which was generated with the assistance of ChatGPT for the purpose of accurately and clearly communicating this disclosure.

Bibliography

[1] P. Getreuer, "Malvar-He-Cutler Linear Image Demosaicking," pp. 83–89, 2011.