Format: some projects are well suited to keeping a lab book, either physical or electronic, which records all your daily tasks, recorded values, results, plots, thoughts, useful numbers, etc. For other projects, this documentation may come in a different form, e.g. organised notes or minutes from project meetings, GitHub commit records, or progress reports.

Whatever form you and your supervisor think is appropriate, your project documentation should:

- Include a project plan formed during the first few weeks of the project. This should include key milestones and dates.
- Be legible and clearly laid out, with dates against every entry/contribution.
- Include regular entries or updates. Most projects will make weekly progress, and so should have weekly updates to the documentation collection.

Lachlan Catto and Glen Pearce's Lab Record

2/8/24

We went through the Risk Assessment
This included emergency exits, lab hazards and risk control strategies

Today we familiarised ourselves with the microscope apparatus and SDK software. We setup a GitHub folder and organised the folders to suit the project. We commenced a project plan for how we are going to approach the project. We created a python file and created our own code from the ground by modifying the code that Matt had setup for us. A python file was created for each task, namely camera viewing and stage movement. Within the stage movement file, we were able to move the stage in all 3 directions the desired step amount and make it go to the home positions.

We started considering how the monolayer locating algorithm was going to work. Some ideas included sweeping across and moving down once it detects an edge through colour or contrast recognition; taking photos as it does so including overlap and stitching the photos together at the end.

Next week

- [x] Check FOV
- [x] Start scanning algorithm
- [x] Take photos
- [x] Save Photos
- [x] Get accurate location + coord system
- [ ] Formalise project plan and steps

9/8/24

We worked on the project plan together, brainstorming our ideas on the whiteboard. We established the key steps that need to be done in the project including creating a GUI, calibrating the origin, creating a conversion between nm and pixels, and a scanning algorithm.

Glen

I developed the fundamental backend basics of the program. I defined many useful functions such as converting the stage encoder to nm, an easy to use movement function which moves the stage to specified coordinates and waits until the movement completes. I added a basic nm -> pixel conversion for the coordinate system. This was achieved by moving the stage one camera screen length and measuring the distance with the nm coordinate system and using the known screen size to obtain a ratio. I added a basic terminal input which allows the user to specify the location of the opposite corners of the sample which enabled me to develop a simple scanning movement algorithm to sweep across the sample. Using the basic conversion obtained, I was able to ensure a slight overlap of the images when scanning the sample and I adjusted the algorithm to take images along the path. I ran into an issue where images of sample were skipped when changing direction, this was fixed by adding additional image acquisition step edge cases to the algorithm. I then added an additional thread which stitches the images together into one large image as the frames are passed in. This is quite slow and will need to be addressed next week. Maybe a different multiprocessing or more efficient overlay method is required. I also ran into an issue with the images not stitching correctly, however this was just do to a 90$\degree$ rotation of the camera which was fixed by adding a counter rotation step. It would be good if this was added as on option to the calibration window. This all resulted in the first basic working implementation of the backend code, allowing us to scan a whole sample easily. Once this implementation is refined we can move on to basic image processing to identify monolayers.

Lachlan

I started designing and coding the GUI for viewing the image and controlling the stage. I spent today familiarising myself more with git including how commits work as well as Tkinter which is used for creating a GUI. I learned how to produce frames and tabs in a GUI and display images, text and buttons. I then implemented what I learnt to display a test image of a monolayer in the GUI as well as a few buttons which don't do anything yet.

Next week I will continue working on the GUI, creating buttons used for navigation on the image and text displaying the position that the image will be in based off its locating from ThorLabs. I will also create a calibration tab that is used for manually calibrating the start and end points for the sweeping algorithm.

Next week

- [x] Formalise Project Plan
- [x] Speed up image stitching
- [x] Finalise basic implementation
- [x] Fix disgusting algorithm code
- [x] Offset coordinate system such that pixel coords start at 0
- [x] Predefine canvas size for stitched image
- [ ] Create GUI
- [x] Add movements buttons to GUI
- [x] View live position on GUI
- [ ] Add camera rotation calibration
- [ ] Update License
- [x] Fix color conversion

16/8/24

We created a more detailed project plan

Glen

This week we wrote up a detailed project plan and discussed the timeline. I then continued to work on the backend code where I added detailed comments to the main code. I then modified the image stitching and generation code to place images more accurately and blend them better. I fixed the color conversion and made the algorithm function much neater and professional.

Lachlan

I continued working on the GUI, adding buttons for movement and labels for viewing live positions in nm. I attempted to add a zoomed-in photo that you could move around using crop and resize functions in PhotoImage. I successfully created an image that was zoomed in to the desired amount, but the buttons I made did not move the viewing position of the photo. Part of the problem was the image was only called once but never updated. I had to add an infinite loop that constantly updated the photo. This still did not work but I used this as a solution to the live position text. I had a similar problem with the live view not displaying so I created an infinite loop through a function that called itself after 100ms. This function was called once to initiate it. I coded on my personal computer so the code did not immediately work when we transferred it to the desktop connected to the ThorLabs equipment and tested it on that. We were quickly able to identify the errors and missing parts to complete the code.

Next Week

- [ ] Add camera rotation calibration
- [x] Add live camera view to GUI in main and calibration tabs
- [ ] Start on a basic flow for the GUI
- [ ] Update License
- [x] Post processing with OpenCV
- [x] Create contours with OpenCV
- [x] Monolayer Class
- [ ] Speed up processing

23/08/24

Glen

This week I managed implement the post processing of the stitched image to enable us to identify monolayers. I added a monolayer class as to have a neat way of storing the data, this also contains functions to assess the quality of the monolayer as they are identified. This class also stores images of each individual monolayer so that they can be assessed visually later. One issue I had with the post processing for the monolayer detection was the conversion to greyscale. To be efficient with detection I wanted to bias the greyscale such that monolayers (mostly red) would appear brighter than everything else, particularly the green background. The simple approach would have been to just consider the red chanel of the image when converting to greyscale however objects that were white in the original image (such as other debris) would be just as bright. To account for this, I introduced a negative offset for any green and to a lesser extent blue pixels, clamping the result to the allowed range of 0 to 255. This meant that areas with significant green or

blue components would appear darker on the greyscale image, allowing for more contrast against the monolayers for easier detection.
Additionally, this week I also spent time refactoring and speeding up the image stitching and processing code as this was a major sink of computation time and resulted in the program taking a long time to run. To address this I implemented several steps, firstly removing excess image saves, next enabling concurrent processing of images, using some masking techniques and finally switching some of the larger image calculations to PyTorch which allowed for GPU based CUDA acceleration. All in all this reduced the image stitching computation time by approximately 100 times. I also changed the saving and post processing steps to use scaled down versions of the canvas, speeding up the code by a further 60 times at least and saving a large amount of storage space.

Lachlan

On the GUI I created a canvas which displayed a live view of the image from the camera. I ran into image queueing and threading issues so I fixed this by changing where the threading was being called. I tested everything in a test GUI file before trying to implement it into the main GUI file. One thing I was able to successfully do in the test GUI was move around a stock photo with buttons. This was not working last week because of threading and garbaging issues. I implemented a slider which will be used for changing the zoom and a 360 degree interactive wheel which will be used for rotating the camera. I had to change everything to be inside a class so it is easier to call functions that will update the windows such as the wheel.

Next Week

- [ ] CAMERA SETTINGS
- [ ] Make move and wait (& relative) generalised for z axis too.
- [ ] Add pixel -> location function
- [ ] Fix Monolayer class image color conversion in quality functions
- [ ] Add move specific distance function
- [ ] Add Autofocus
- [ ] Speed up
- [ ] Table of statistics
- [ ] Add a thread which is given the canvas and scales it down during image processing and displays it
- [ ] Threaded final image save
- [ ] Add a save and downscale image function
- [ ] Stage accuracy