
# CRISP GUI Guide

Version 1.0.0

Lewis Dean[1] and Robin de Freitas[2]

[1]lewisdean22@outlook.com

[2]robindefreitas123@gmail.com

29-07-2025

CRISP (Calibration and Research Imaging System for Plastic scintillators) integrates Raspberry Pi cameras in a prescribed setup with a ready to use software package to provide beam-side analytics of a proton beam. This document details the required setup of apparatus and offers a guide for the software. For further information, see the project reports within CRISP's GitHub repository[1] .

**Comments key:**

- **NOTE:** details something that directly impacts the user's ability to use the system in it's current state. Attention should be paid to these and the advice followed when using the system.
- **IMPROVEMENTS:** are comments for potential improvements to CRISP; however, their content does not affect the user's ability to use the system and should be treated as notes for further work only. For a summary of features one might consider adding to CRISP, please see Appendix A.

## How to use this document

- Initially, follow **section 1** for *installation and running of the application*.
- Use **section 2** to *check and inform the physical experimental design you intend to use*.
- Follow the walkthrough laid out in **section 3** to *prepare for and perform your experiment*.
- The further details given in **section 4** are intended *for future developers of CRISP*.

## Contents

# 1 Software Setup

## 1.1 Installation and usage

The CRISP source code can be found within its associated CRISP GitHub repository[2] . To run the application, the containerisation software Docker must first be installed (see the DockerDocs[3] ); the repository includes already built Dockerfiles, thus users must simply spin up these files to start CRISP. Since Docker is a Linux-based tool, we recommend cloning the CRISP repository into a Linux environment.

**Docker on Windows:** Running Docker on Windows requires a Linux virtual machine like Windows Subsystem for Linux (WSL) or Oracle VM VirtualBox – we recommend installing WSL[4] . Docker itself can be installed either as a command line interface tool or as the graphical interface Docker Desktop. We recommend the installation of Docker Desktop[5] . Before attempting to start CRISP, ensure Docker Desktop has already been started.

**NOTE:** Making extensive modifications to the CRISP codebase on Windows can quickly fill Docker's virtual disk space – for a guide on how to mitigate this, see Appendix B.

**CRISP Commands:** For those unfamiliar with Docker, the best option is to source the crisp shell script from within the repository as follows:

```
$ source ./crisp.sh
```

The resulting output is shown by Figure 1. Doing this allows dedicated CRISP functions to be called. For example, the commands `crisp` and `close_crisp` are used to start/close the CRISP software respectively. Running the `rebuild_crisp` command is only necessary after modifications of the CRISP codebase.

Alternatively, precise control over which CRISP sub-services to run is possible by directly using the Docker compose commands[6] . The custom crisp commands above are wrappers of these underlying Docker compose commands.

**NOTE:** The CRISP Dockerfiles were built and tested on devices with x86_64 architectures. If your device has a different architecture, it may be necessary to rebuild the Dockerfiles (use the `rebuild_crisp` command or learn more about builds in the DockerDocs[7] ).

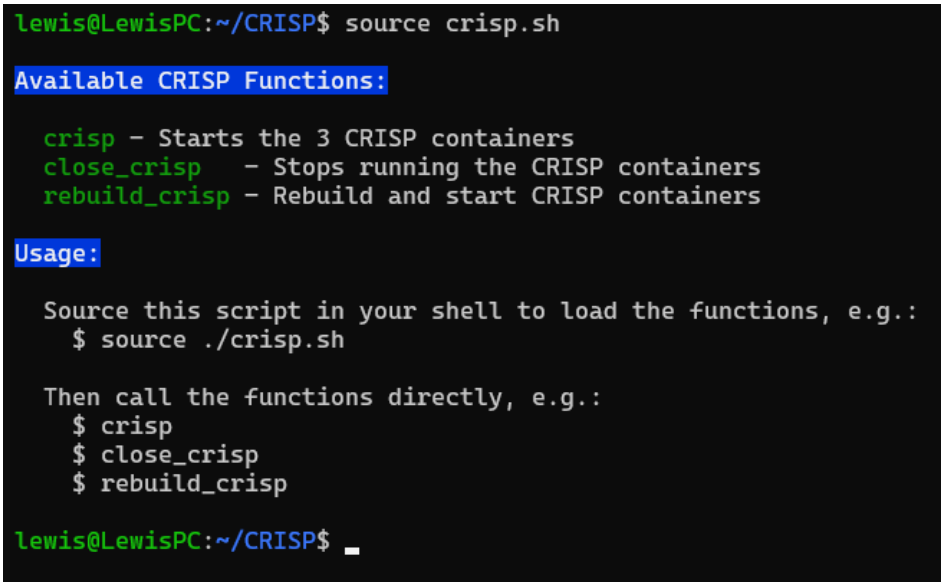



Figure 1: The command line guide printed upon sourcing `crisp.sh`.

## 1.2 Database

When the application is begun, a connection to a PostgreSQL[8] server will be initiated. Whilst there are no further steps to be taken if interacting with the application strictly through the user interface - if you want to probe the data in the database directly, a third party software can be used. For example, the command line interface PSQL[9] or the graphical interface PGAdmin[10] . The structure of the database is shown in appendix **??**. **WARNING:** If you edit the database through any external software there is no guarantee CRISP will work as intended.

**Backing up and restoring data:** The current version of CRISP requires these steps to be carried out in the command line. While the Postgres Backup documentation[11] can be used, the specific instructions used when testing the system are given below.

- BACKUP:

```
$ pg_dump -U postgres -F c -d crisp_database -h 127.0.0.1
        -b -f 'your_filename'_$(date -u +%Y.%m.%d.%H-%M).sql
```

- RESTORE:

```
$ pg_restore -U postgres --clean -h 127.0.0.1
        -d crisp_database 'your_filename'.sql
```

# 2 Apparatus

## 2.1 Scintillator

The scintillator used must be a transparent cuboid producing light in the visible spectrum. Before data collection, a polish is useful as well as careful handling. Any marks or blemishes on the block can have a detrimental impact on the data collected (shown in figure 2).

The diffuse reflection caused by such marks on the scintillator may impact the data even if not directly on the face being imaged and should be considered in the setup if a perfectly smooth scintillator is not possible. An example of this is shown in figure 7

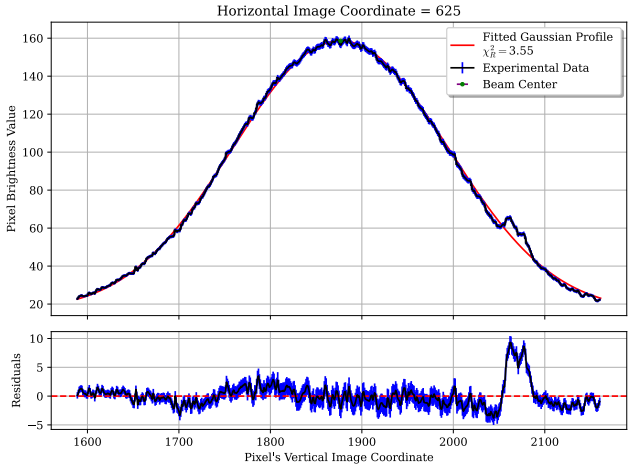
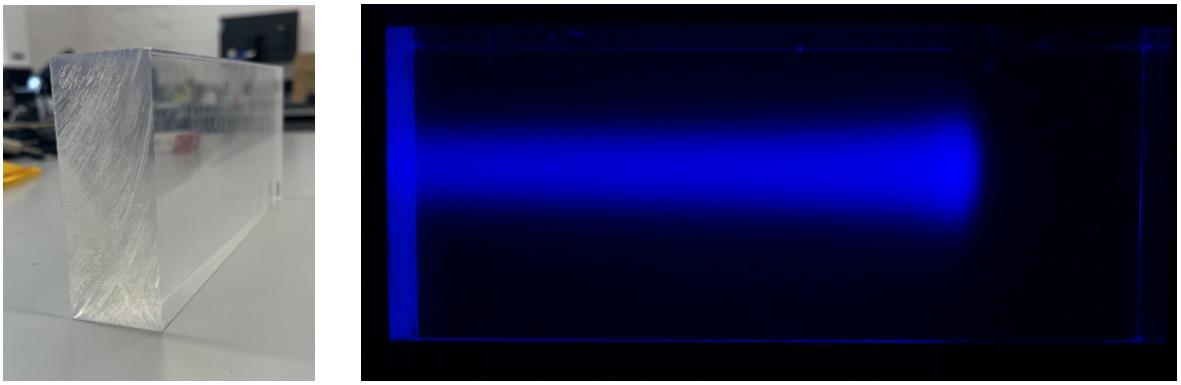



Figure 2: A plot showing the effect of a scratch on the scintillator is shown. The effect can be seen as a deviation from the model around pixel 2075.



(a) (b)

Figure 3: (a) shows a scratched face of the scintillator (resulting from sawing of the scintillator). On the left hand side of (b) the effects can be clearly seen as a bright face from diffuse reflection at the surface.

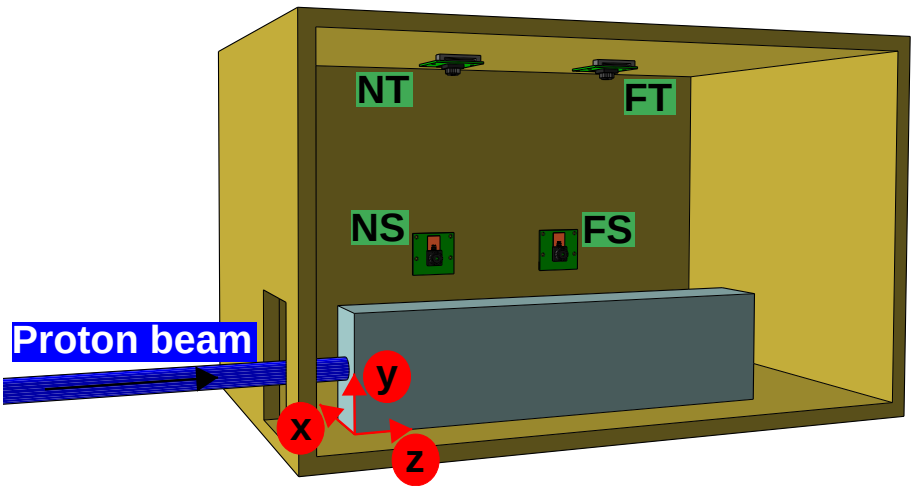



Figure 4: Example schematic for a CRISP box setup.

## 2.2 Box design

The first and foremost consideration in imaging your scintillator is housing it in a light tight, non-reflective environment. The presence of large amounts of external light can significantly affect the data collected, especially if not a uniform background but rather a bright artefact.

The next is the mounting positions of cameras within the box. These must be placed such that they image orthogonal to the beam axis – crucially, CRISP does not support cameras with their optical axis along the beam axis. The system was tested with cameras placed fairly centrally on the expected beam axis and results cannot be guaranteed for large deviations from this. The main constraint is from the areas covered by the calibration patterns from a cameras perspective as shown in figure 5. While regions outside of these may still provide valid results the accuracy of the homography falls off sharply outside the region and the uncertainties are difficult to quantify. These additional uncertainties are not built into CRISP.

## 2.3 Cameras

The development of CRISP involved a box setup with four Arducam IMX519 cameras. We recommend this camera model as they are relatively cheap and allow their focus to be controlled remotely through computer commands (as opposed to through manual adjustment).

**NOTE:** Raspberry Pi computers do not have the Arducam drivers installed by default; a guide on driver installation can be found in the associated Arducam documenta-

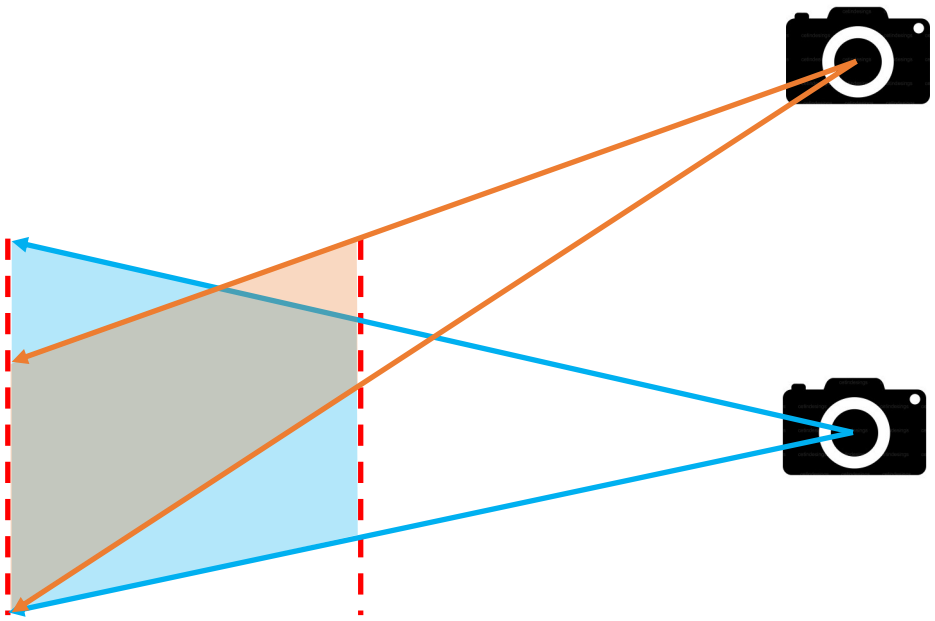


Figure 5: The calibrated regions are shown as shaded regions between two red-dashed calibration planes.

tion[12] . You can test whether installation was successful with a quick `rpicam-hello` command – a camera feed window should open.

## 2.4 Imaging with Raspberry Pi Computers

CRISP is designed to coordinate image acquisition across a set of cameras which are each controlled by their own Raspberry Pi computer. Remote camera control from a user's local device is achieved through SSH connections to each Pi. For simple image-taking, e.g. camera calibration, the backend of CRISP will execute an appropriate rpicam (aka libcamera) command on the Pi. For the more sophisticated image-taking undertaken during test/real proton beam runs, a dedicated Python imaging script is ran on each Pi. For more information on the latter, see Section 4.2.

Without peripherals and a monitor in order to navigate a Raspberry Pi's desktop environment, the two main options for controlling it outside of CRISP are:

1. SSH into the Pi and use its command-line interface
2. Download a VNC tool, e.g. RealVNC[13] , so that you can remotely navigate the Raspberry Pi OS graphical interface

Such tools could come in handy if facing any dependency issues on a Raspberry Pi.

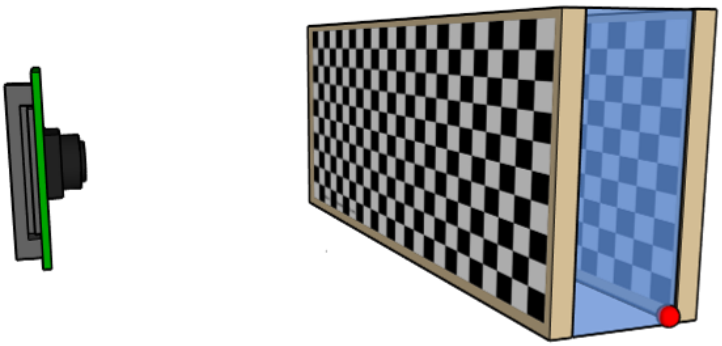


Figure 6: (work-in-progress render) A schematic of a side-mounted camera to help visualise how the near and far calibration boards should be placed to ensure the full scintillator volume is calibrated for. The Red dot indicates the origin of the box's coordinate system.

**NOTE:** At the time of writing, CRISP in its entirety has been successfully deployed with four Raspberry Pi 4Bs. The other Raspberry Pi models should be compatible. In theory, any device controllable over SSH could be employed, although CRISP's imaging script may need replacing with some suitable equivalent for that device (e.g. if without access to Python). Furthermore, the imaging script was written with the aim of controlling Arducam IMX519 camera settings, hence specific controls (e.g. auto-focus) may need commenting-out if alternative camera modules are used. Such updates may be essential if errors appear when CRISP attempts to use the imaging script.

## 2.5 Coordinate system

See Figure 4. The coordinate system shown is a convention which must be followed so that the backend logic for pinpointing via homographies gets implemented correctly. That is, the origin should be placed at the bottom right corner of the scintillator's anterior face, as would be seen by a proton beam upon entry.

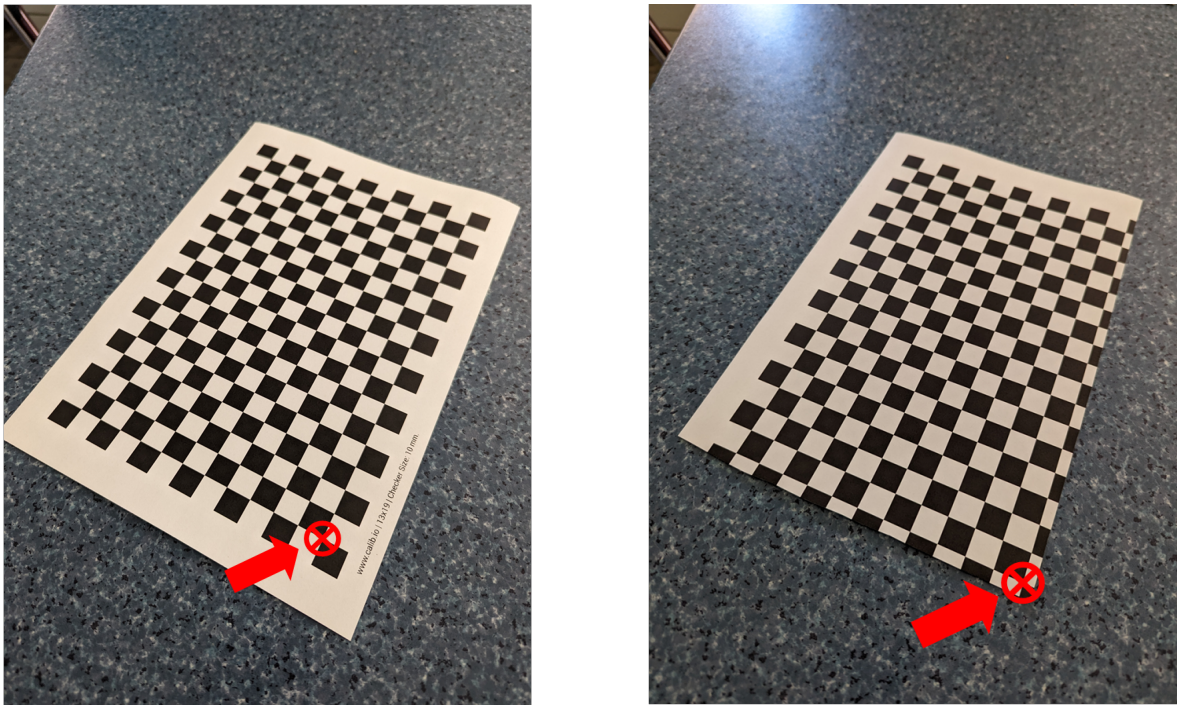
## 2.6 Calibration boards

CRISP uses plain, chessboard calibration patterns to calibrate the cameras. Websites like this pattern generator[14] are easy to use to generate a print of the required pattern. This version was tested using 10 mm square size and while a smaller size could possibly improve the uncertainty on the final measurements it may also increase the computation time. Further investigation would be needed to determine whether this decrease in size would significantly decrease the uncertainty or increase the

computation.

While there is this constraint on the type of pattern, there is no prescription of how that pattern must be made other than it should, crucially, be a flat plane in the calibration images. We achieved this by fixing printed calibration patterns to stiff card or pieces of board. Whatever size patterns is generated and whatever method is used to create the board, careful measurements must be taken of the grid size in each direction and the thickness of the final board.

Thought should be given as to the actual region defined by the calibration pattern. The first identified point is marked in figure 7a - corresponding to the first intersection fully surrounded by checkers. Consider for example a face of width 4 cm. If using a printed calibration board, there will likely be a boarder of around 0.5 cm which when added to the width of a 1 cm square means that over a third of the width is not encompassed in the calibrated region. A smaller square size will help this, but if a larger size is still preferable, the pattern can be cropped through the squares (as shown in figure 7b).



(a) (b)

Figure 7: (a) shows a typical, printed, 10 mm, chessboard calibration pattern. (b) shows the same pattern with cropped edges. Both have a red marker indicating one of the most extreme corners that will identified.

# 3 CRISP Usage Walkthrough

This overview outlines the expected work flow for using CRISP, with the subsequent sections describing the steps in more detail.

**NOTE:** Complimentary flowcharts, for the setup and experimental stages of this walkthrough, can be found in Appendix D and Appendix E, respectively.

1. Open CRISP
2. Add Raspberry Pi
3. Test camera
4. Optimise camera focus
5. Create setup
6. Calibrate cameras
    - Near/Far face calibration
    - Scintillator edge selection
7. Create an experiment
8. Complete a test run
9. Analyse a test run
10. Complete a real run
11. Perform analysis

## 3.1 Open CRISP

First, use the `crisp` command (ensure `crisp.sh` has been sourced first) to start running CRISP. After this, navigate to localhost:3000 within your browser, where you will be greeted with CRISP's homepage.

## 3.2 Add Raspberry Pi

The first step is to add a Raspberry Pi computer by inputting its username, IP address and password into the form found on the homepage. The added Pi will then appear in the top panel wherein one can find a switch that attempts to establish an SSH connection to it. If needed, you can edit a Pi's details by clicking on its username and completing the edit form.

**NOTE:** Users may wish to set static IP addresses on their Raspberry Pis. This is because, if the router is using DHCP, the configured Raspberry Pi IP addresses may be subject to change (users would then need to manually update IP addresses in the GUI). CRISP has been successfully deployed without static IP addresses, therefore this step is optional.

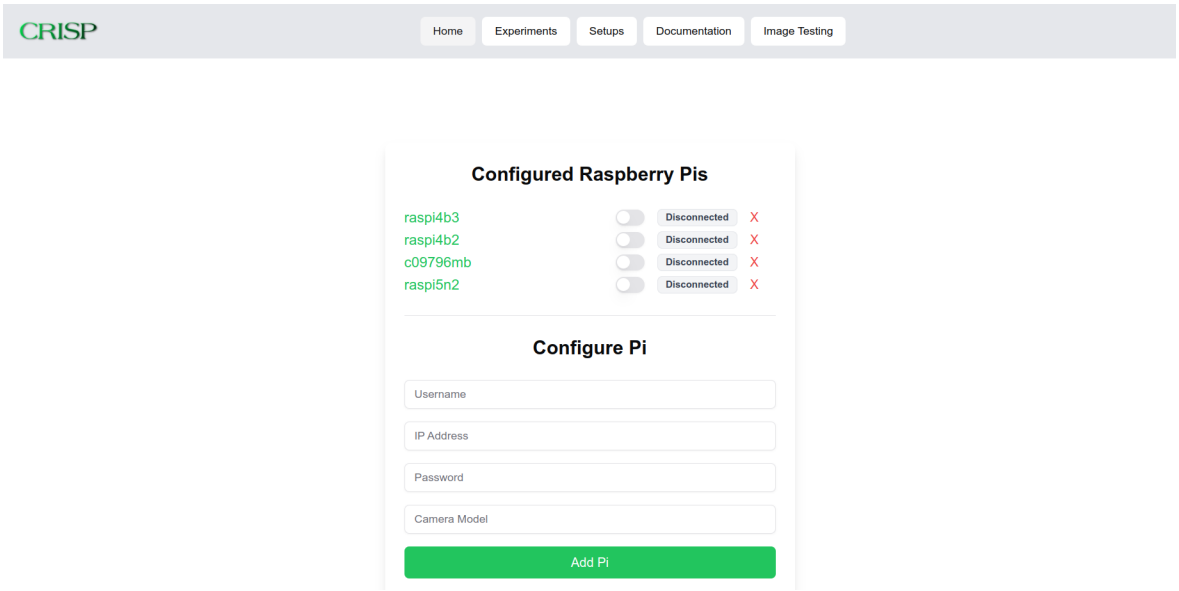



Figure 8: CRISP homepage is shown

**NOTE:** Anything can be input for the 'Camera model' as currently this parameter is only semantic.

## 3.3 Test camera

With a Pi configured and an SSH connection to it established, the camera's functionality can be tested in the 'Image testing' tab. Select a camera and and input the requisite camera settings.

**IMPROVEMENTS:** A valuable addition to this would be a display of all photos taken with this method as well as full GUI capacity for database interaction (i.e. ability to delete test images).

## 3.4 Optimise camera focus

For each camera you intend to use in an experiment it is important to try and optimise the focus for where the beam will be. An automation for this is not currently built into CRISP and so it is recommended to use the 'Image testing' tab to test the focus on an object placed roughly in the position of where the beam will be (e.g. you could place

a calibration pattern within the plane bisecting the scintillator along the camera's optical axis). The focus is controlled by adjusting the 'lens position' parameter. Once a preferable value is found, this should be noted for use in the calibration steps.

**IMPROVEMENTS:** An extra automation would be to incorporate a focus optimising algorithm directly into the calibration pipeline. One possible image feature to analyse, as done in some already established autofocus mechanisms, is Laplacian variance. Further investigation could also explore how focus impacts the quality of Bragg peak determination.

## 3.5 Create setup

Each setup used must first be calibrated prior to data collection. Setups are created in the 'Setup' tab where your preexisting setups can also be seen. Click on the setup entry in the table to interact with that setup.

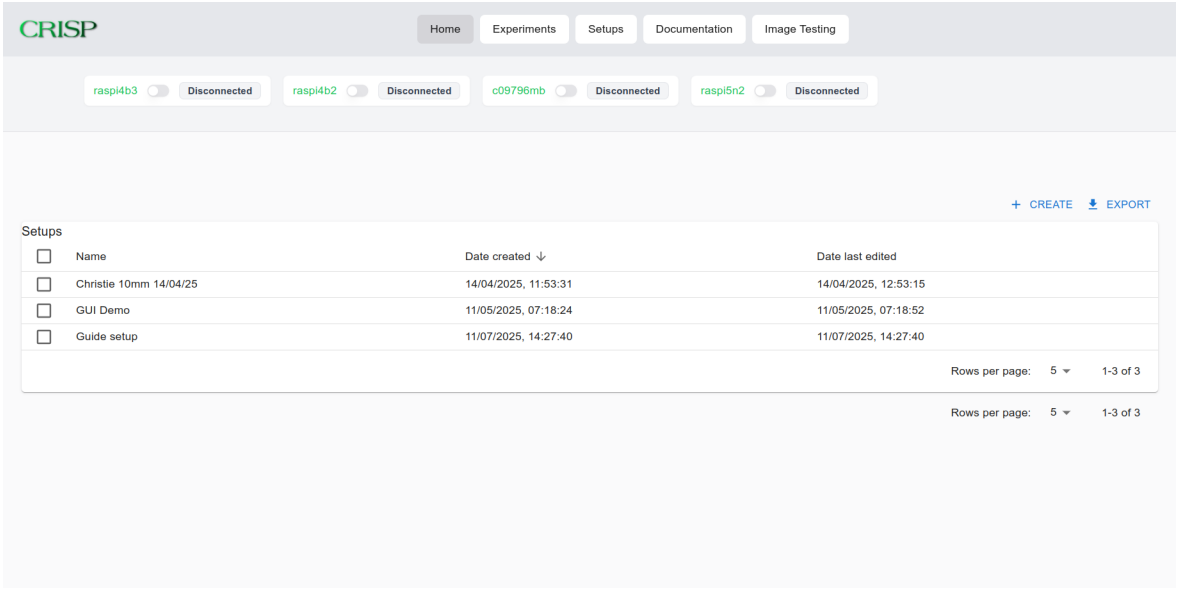



Figure 9: The landing page of the setup tab can be seen to show a list of the current, existing setups as well as a create new setup button.

**IMPROVEMENTS:** It would be better if when a new setup is created the user is routed to that specific setup page rather than the list of setups.

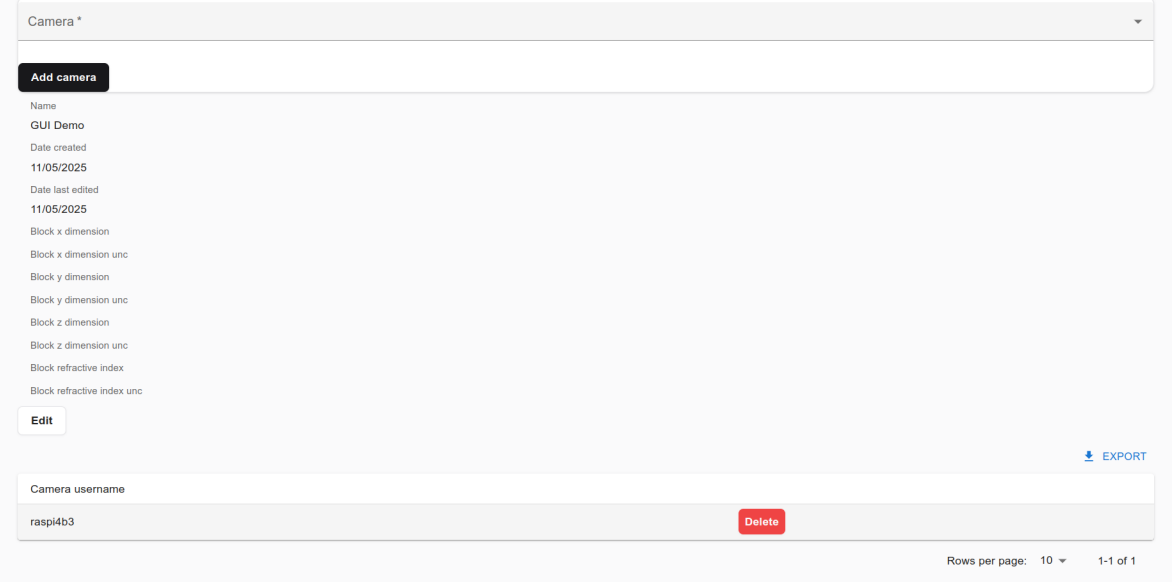



Figure 10: Summary page of a specific setup is shown.

Cameras/Pis which have been added via the homepage can be added to your setup via the drop down menu shown at the top of figure 10. Added cameras will appear in the table at the bottom of the page.

The parameters of the block can be edited using the edit button seen in the middle portion of the page.

## 3.6 Calibrate cameras

On initially clicking on a camera which you have added to a setup, you will be presented with a form (shown in figure 11) asking you to define key camera position, as defined in section 2.5. The lens position must also be input here.

**NOTE:** Functionality for distorition calibration is not fully complete and will prevent the user progressing to the rest of the calibration so should NOT be enabled. If using the same IMX519 cameras as tested here the effects of distortion are likely minimal to none.

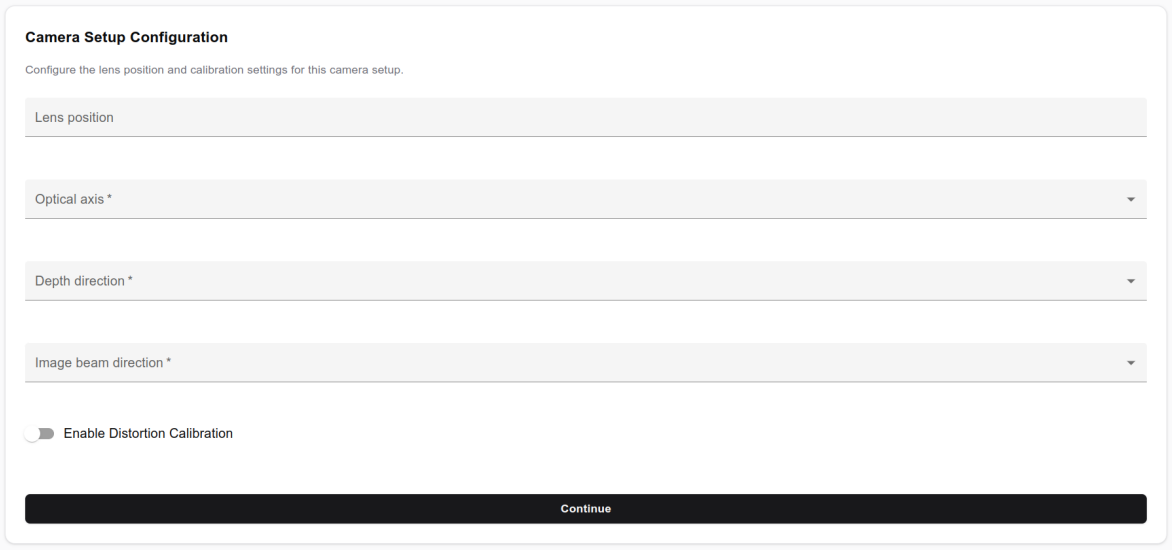



Figure 11: The form which asks the user to define the camera's position (optical axis, depth direction and image beam direction - as defined in section 2.5) is shown. Lens position and enabling distortion calibration must also be input here.

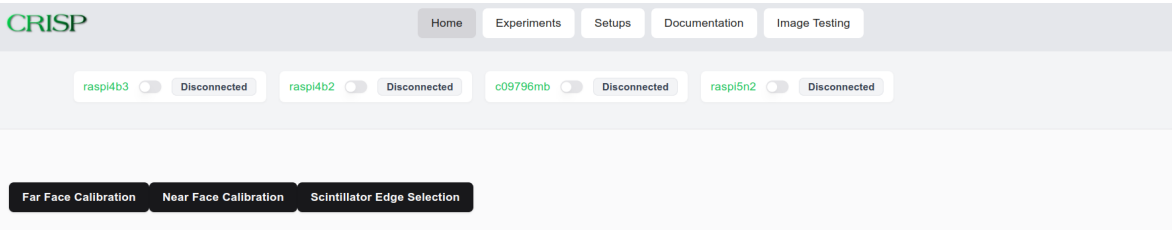



Figure 12: Page which provides links to the 3 necessary calibrations - near face calibration, far face calibration and scintillator edge selection.

Having submitted the form, you will land on the calibration hub page shown in figure 12. All of these three sections must be completed for a cameras setup to be complete.

**IMPROVEMENTS:** There is opportunity here for a much more informative interface, offering more information to the user on state of completion of the various calibration steps, as well as indicating on the previous page which cameras have complete calibrations.

### 3.6.1 Near/Far face calibration

See Figure 6 as a reference.

### 3.6.2 Scintillator edge selection

## 3.7 Create an experiment

With your fully calibrated setup in place, you are ready to begin an experiment. Proceed to the 'Experiments' tab where you can create a setup. Select the appropriate setup from the drop down and give your experiment a name. Now select the experiment you have just created from the table to proceed to the next step.

## 3.8 Complete a test run

In almost all instances you will first want to perform a test run in order to optimise the exposure settings of your cameras. To do this click the 'Add test run' button and fill in the beam run number and beam parameters in the pop-up.

You will then be taken to the test run page for this beam run where there will be a list of all of the cameras in your setup. For each, click on 'Create settings' and define the range of camera settings which you want to test. The settings can be viewed in a drop down of each camera.

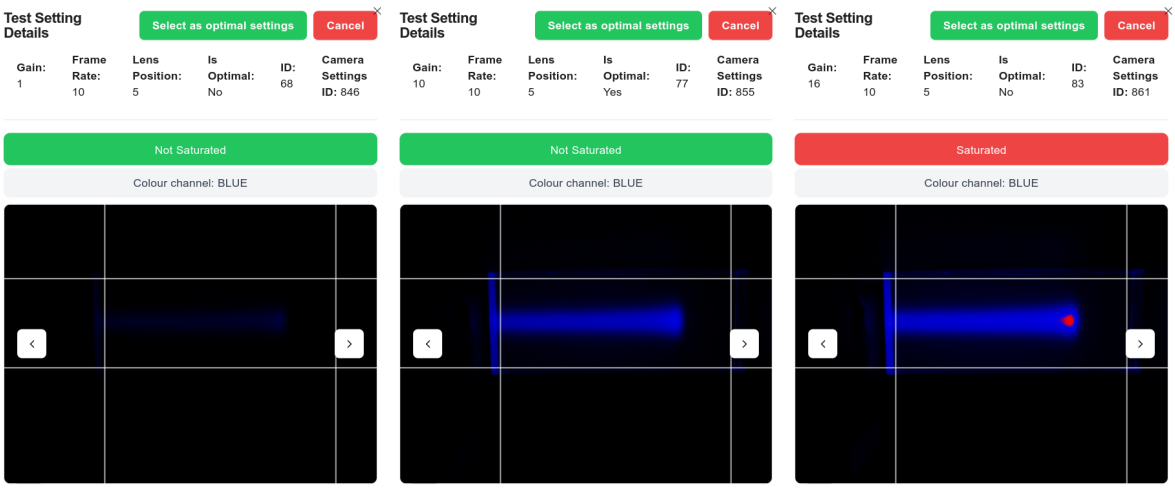
Once all cameras have been assigned settings you will be able to click the 'Take data' button at the bottom of the page. *THIS DOES NOT BEGIN DATA COLLECTION.* Rather it will progress you to a page detailing the connection status of each camera, the idea being you will have a chance to spot any connectivity issue before initiating the beam. When ready, 'Start data collection' will trigger the data collection.

## 3.9 Analyse a test run

Once data collection is complete you will be returned to the beam run's page. Here you will find the images for each setting of each camera. An optimisation of the settings will have been performed in the blue channel, selecting the brightest image with no saturation. This will be indicated for each camera in their dropdown of settings. If all settings resulted in saturated images, none will be shown as optimal.

Clicking on a particular setting's entry will pop up the data taken at that setting - some examples of which are shown in figure 13. Greyscale, red, green and blue images can be scrolled through - with saturated pixels in each highlighted in the image. In this pop-up you are able to overwrite the automated optimisation and reselect the optimal settings. Whilst it is recommended to use the identified settings to maximise the quality of data - if you do have a need to choose settings - it is recommended that

you select them here rather than inputting them manually down the line. This not only limits the risk of a typo but ensures you are seeing the quality of the image that you will retrieve on the real run.




(a) Undersaturated (b) Optimally saturated (c) Oversaturated

Figure 13: An example (aspect ratio adjusted) of the results of a test run are shown for undersaturated (a), optimally saturated (b) and oversaturated (c) camera settings. White lines on all images indicate the region of interest. A red overlay can be seen in (c) with each overlaid point corresponding to a saturated pixel. Buttons to switch to a different colour channel can be seen on the left and right of each image while a button to select the image as having optimal settings can be seen at the top.

**IMPROVEMENTS:** Gain optimisation is currently completed automatically in the blue colour channel. It is possible to manually inspect for saturation however this may become time consuming and tedious. Another desirable feature, somewhat linked, is an option to 'reset' the optimal settings. In the first instance this would provide a fix for the first issue too as a user could select the colour channel in which to reset.

### 3.10 Complete a real run

### 3.11 Perform analysis

# 4 CRISP Codebase

## 4.1 CRISP Tech Stack

If you are a developer wanting to expand CRISP, you should expect to become familiar with the following tech stack:

- PostgreSQL[15] - relational database management system - we used SQLModel to manipulate the Postgres database with Python code.
- FastAPI[16] - Python web framework used for CRISP's backend.
- Next.js[17] - React-based framework from which CRISP's frontend is built.
- React Admin[18]
- Docker[19] - containerisation software used to synchronise the operation of CRISP's frontend, backend and database.
- PiCamera2[20], OpenCV[21], NumPy[22], SciPy[23] for imaging/analysis logic.

The broader system architecture can be seen in figure 14.

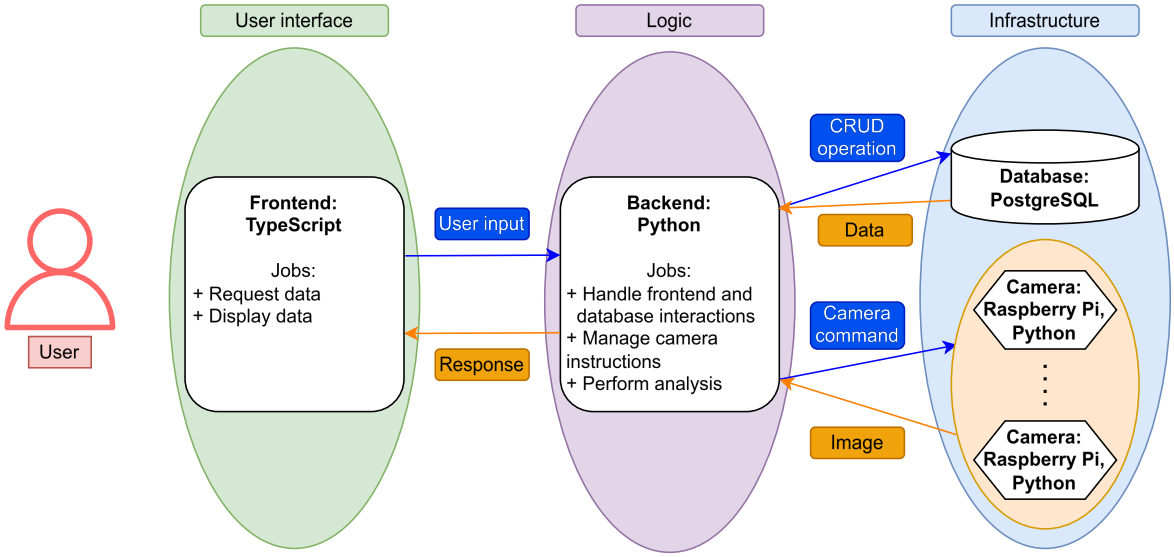



Figure 14: A schematic diagram of CRISP's system structure is shown. A user interface layer can be seen to send user inputs to to a logic layer which can subsequently call Create, Read, Update or Delete (CRUD) operations on a database in the infrastructure layer which returns data. The logic layer can also send instructions to a set of Raspberry Pi cameras in the infrastructure layer, which then return images. Finally, having also dealt with any analysis, the logic layer sends a response to the user interface.

## 4.2 Beam Run Imaging Script

**NOTE:** The script to be ran on each Pi can be found within the CRISP repository at `\backend\src\video_script.py`. This script was created for Python 3.9, so any errors when using it could be due to differences in Python version.

When users select some beam run imaging settings via the GUI and they begin the data acquisition, their inputs are first passed to the backend, after which the Raspberry

Pis are told over SSH to run their local copy of `video_script.py` with the appropriate settings. The user inputs are passed as arguments to this Python script – the list of currently implemented arguments can be seen within the script's `parse_arguments()` function. You need not manually add the imaging script to any Pis as CRISP will automatically transfer it to connected Pis if necessary.

All current instances of running this script from CRISP's backend involve commands which include the `-log` argument. As a result, if any unexpected errors occur when using the script, you can SSH into a pi (or another means of accessing its file system) and read through a file called `metadata.log`. This includes the settings of each image taken, in addition to useful debug statements. Finding this log, alongside the images taken in that beam run, is simple thanks to the directory hierarchy system imposed by CRISP. For example, during an experiment with an id of 1, suppose you perform a test beam run that is assigned an id of 2. You want to find the log on a Raspberry Pi with the username example-pi. It can be accessed from:

```
\home\example-pi\experiment_id_1\test_beam_run_id_2
```

The library PiCamera2 (should be installed on Raspberry Pis by default) allows control of the cameras through Python code. Before image-taking, we have tried to tune the camera settings such that images are as raw as JPEGs can be. Settings are applied through a dictionary of controls passed to the Picamera object. In our case, this takes the following form:

```
controls_dict = {
        "AnalogueGain": args.gain,
        "ColourGains": (1.0, 1.0),
        "NoiseReductionMode": 0,
        "Contrast": 1.0,
        "Saturation": 1.0,
        "Sharpness": 1.0,
        "FrameDurationLimits": (frame_duration, frame_duration),
        "ExposureTime": int(frame_duration / 2)
    }
    if args.lens_position is not None:
        controls_dict["AfMode"] = controls.AfModeEnum.Manual
        controls_dict["LensPosition"] = args.lens_position

    picam2.set_controls(controls_dict)
```

**IMPROVEMENTS:** It is possible that `"AwbEnable": False` also needs asserting for automatic white balance to be disabled since it might override `"ColourGains": (1.0, 1.0)`

otherwise. This should be checked soon!

The settings intend to minimise post-processing/filtering of the images. The analogue gain set is that specified by the user through the GUI. The `frame_duration` is inferred from the user-inputted frame-rate, and the exposure time is set to half of the frame duration to avoid rolling shutter artefacts.

Once image-taking has finished, they are archived into a single tarball before being streamed over Ethernet and subsequently saved to CRISP's database.

**IMPROVEMENTS:** CRISP is currently hardcoded to use JPEGs during main/test runs. This was done despite JPEG being a lossy format because we wanted quick beam-side analysis over Ethernet. Future developers could explore alternative approaches; `video_script.py` has arguments allowing PNG images with a high bit-depth to be taken, albeit requiring more memory space. Additionally, the lossless TIFF format, commonly used in scientific imaging, could be a format added in the future.

## 4.3 Analysis Logic

The details of CRISP's image analysis will not be explained here. Instead, we refer readers to a series of project reports accessible from the CRISP repository. Below is a short summary of some potential topics-of-interest within each:

- **Robin's Semester 1 report**
  - Homographies and how to compute their errors.
- **Lewis' Semester 1 report**
  - List of vector equations used during the 3D pinpointing method.
- **Lewis' Semester 2 report**
  - Edge detection around scintillation light to restrict region of image analysis.
  - Image analysis routine applied to April 2025 proton beam data.

When we fit to the beam-axis scintillation light distribution, we require the Bortfeld model. This fitting logic can be seen within `\backend\src\modified_pybragg.py`. As the name suggests, besides some small adaptations, we have made use of an already existing library called PyBragg – its repository can be found by clicking here[24] .

# Appendices

## A Proposed CRISP Extensions

- WAL Archiving of database
- Algorithm for automating a camera's focus optimisation
- Ability to open camera streams on a Raspberry Pi (useful for checking FOV and for preparing calibration images).

## B Compacting a WSL Virtual Disk

After rebuilding CRISP many times, unused/dangling images can accumulate; these can be culled with `docker image prune -a`. Running `docker system prune -a` removes all unused containers too. These commands are sufficient on Linux-native devices for clearing up lots of disk space accumulated by Docker . However, on Windows, these commands delete data without immediately clearing disk space. This is because the virtual hard disk used by WSL does not automatically compact itself in order to use the gaps created after any deletions. Below is a guide on how to compact the Docker VHDX to recover disk space.

1. Run Command Prompt as administrator
2. Enter:

   ```
   diskpart
   ```

   This starts a Windows tool for managing disk partitions.
3. Enter:

   ```
   select vdisk file="\path\to\your\DockerVHDX"
   ```

   Replace the filepath with that to your Docker VHDX.
4. Enter:

   ```
   attach vdisk readonly
   ```
5. Enter:

   ```
   compact vdisk
   ```

Alternatively, experiment with hot reloading when developing CRISP so less rebuilding is needed, and hence the need to compact the Docker VHDX is less urgent.

# C Database structure

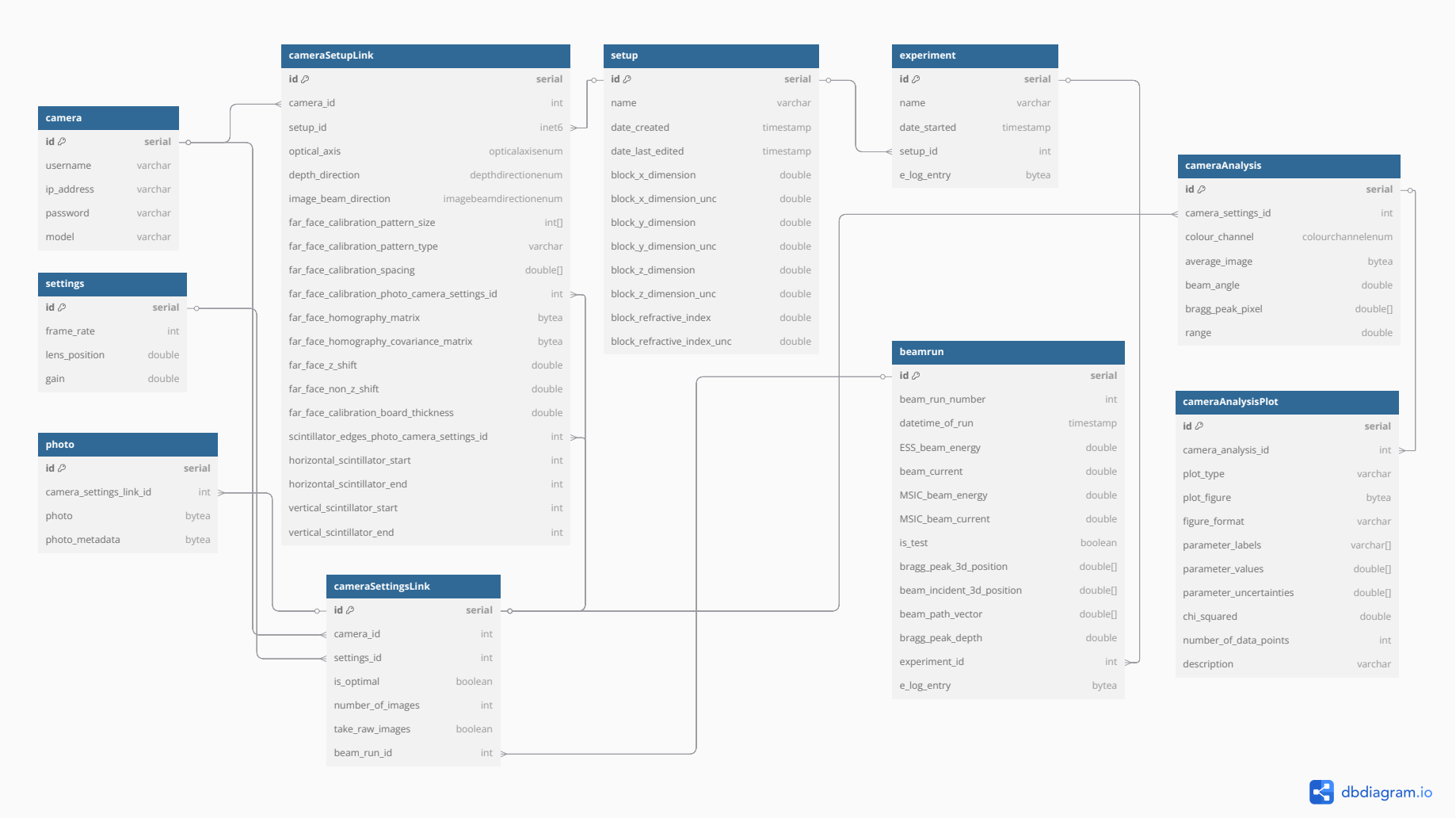



Figure 15: An entity relationship diagram of the database structure used is shown.

# D CRISP Setup Creation Flowchart

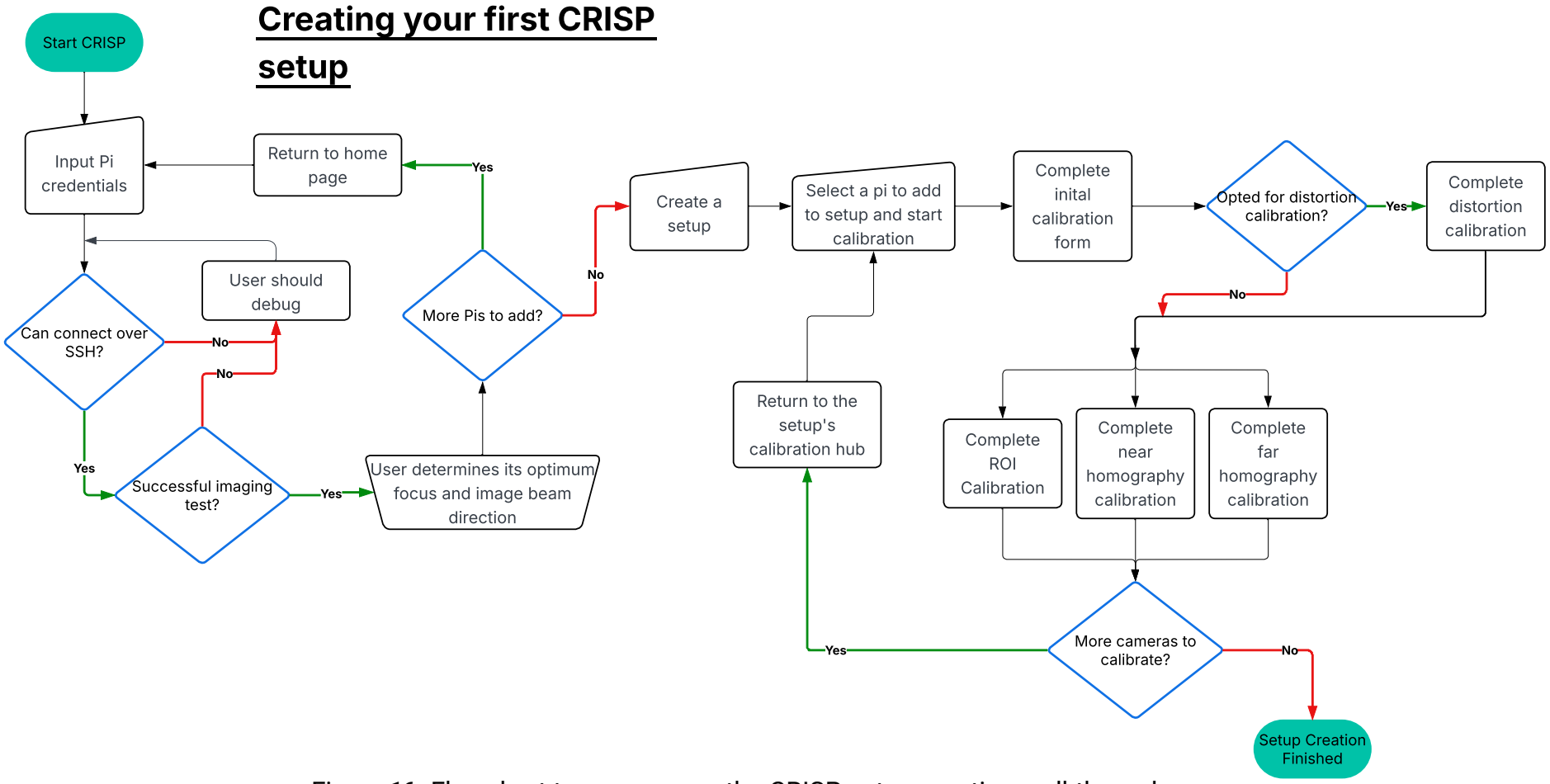



Figure 16: Flowchart to accompany the CRISP setup creation walkthrough.

# E CRISP Experimental Pipeline Flowchart

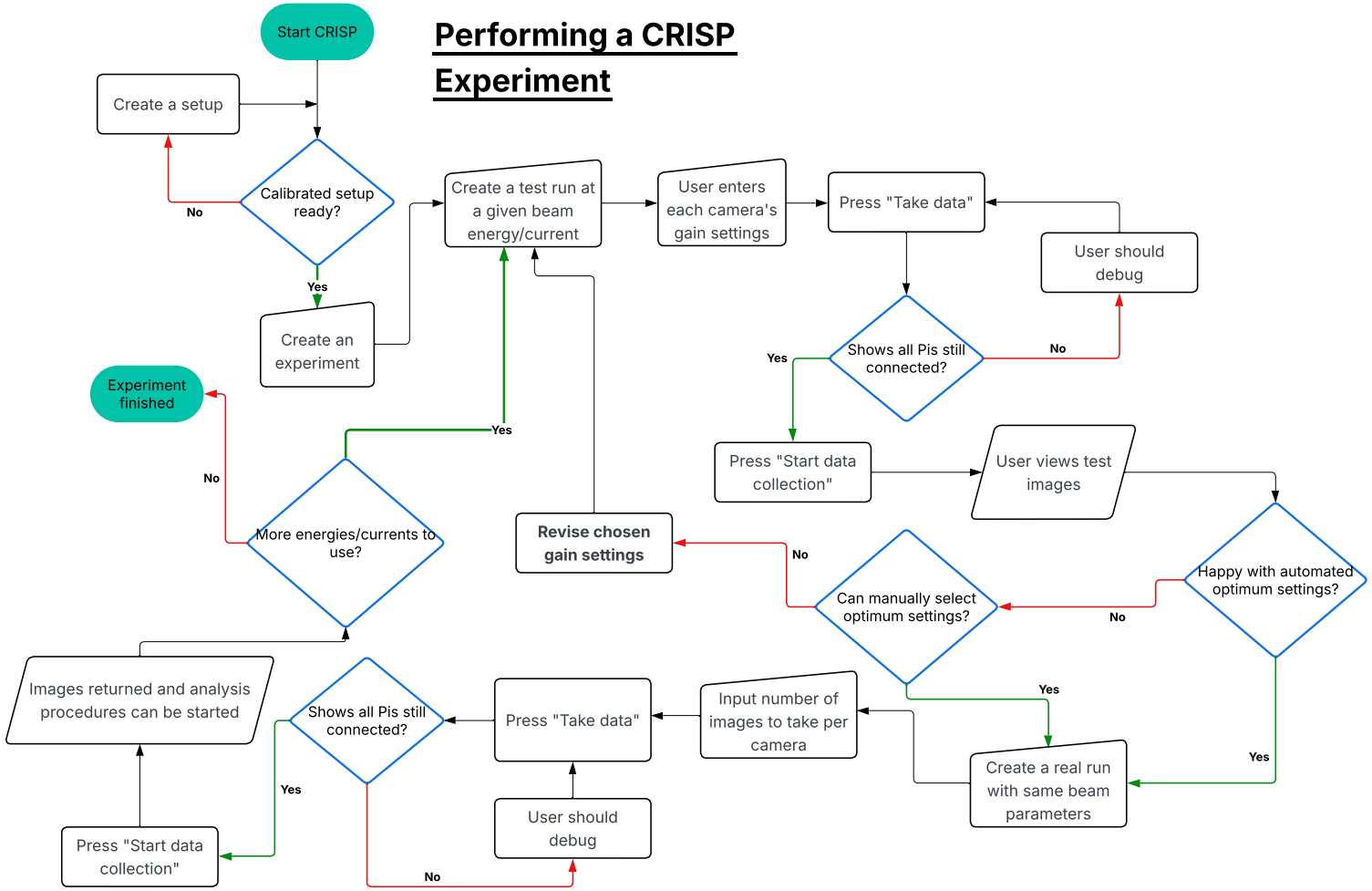



Figure 17: Flowchart to accompany the CRISP experiment walkthrough.

# Index of URLs in document


1. https://github.com/rdef12/CRISP
2. https://github.com/rdef12/CRISP
3. https://docs.docker.com/
4. https://learn.microsoft.com/en-us/windows/wsl/install
5. https://www.docker.com/products/docker-desktop/
6. https://docs.docker.com/compose/
7. https://docs.docker.com/
8. https://www.postgresql.org/
9. https://www.postgresql.org/docs/current/app-psql.html
10. https://www.pgadmin.org/
11. https://www.postgresql.org/docs/8.1/backup.html
12. https://docs.arducam.com/Raspberry-Pi-Camera/Native-camera/16MP-IMX519/
13. https://www.realvnc.com/en/
14. https://calib.io/pages/camera-calibration-pattern-generator
15. https://www.postgresql.org/
16. https://fastapi.tiangolo.com/
17. https://nextjs.org/
18. https://marmelab.com/react-admin/
19. https://www.docker.com/
20. https://docs.arducam.com/Raspberry-Pi-Camera/Native-camera/PiCamera2-User-Guide/
21. https://opencv.org/
22. https://numpy.org/
23. https://scipy.org/
24. https://github.com/flome/pybragg