CRISP GUI Guide

Version 1.0.0

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

[1]lewisdean22@outlook.com

[2]robindefreitas123@gmail.com

July 27, 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.

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. To run the application, the containerisation software Docker must first be installed (see the DockerDocs); 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. 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. 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. 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).

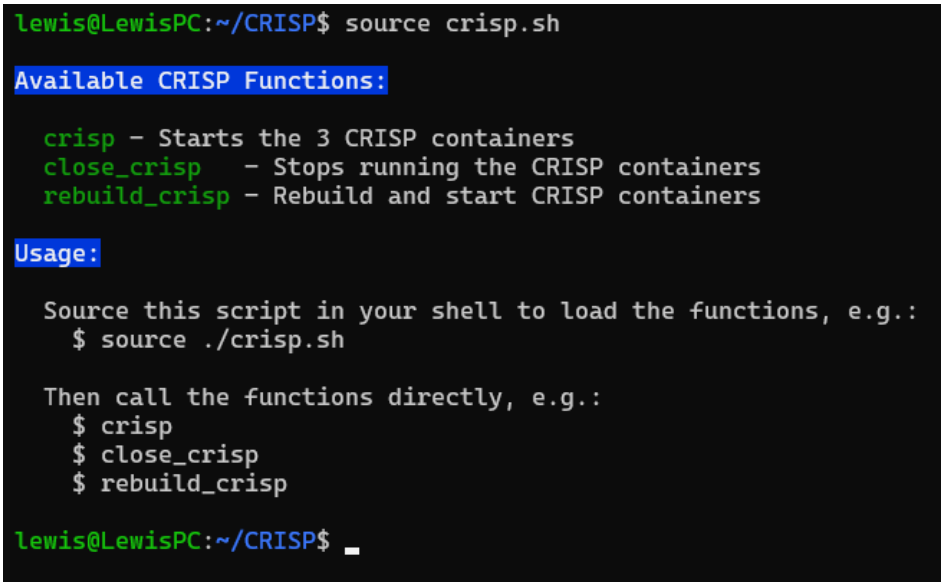


Figure 1: The command line guide printed upon sourcing `crisp.sh`.

1.2 Database

When the application is begun, a connection to a PostgreSQL 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 or the graphical interface PGAdmin. The structure of the database is shown in appendix C. **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 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 3

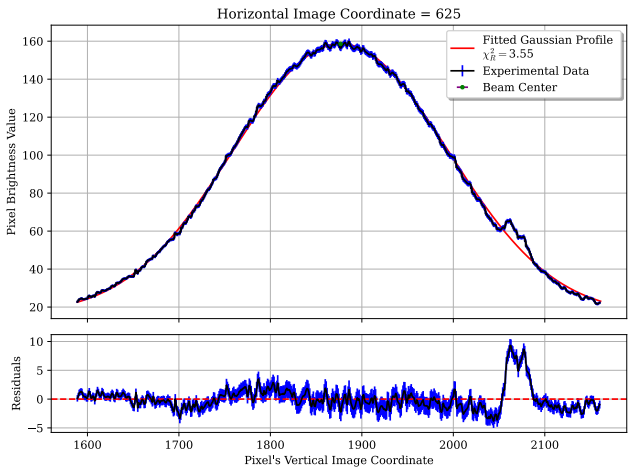
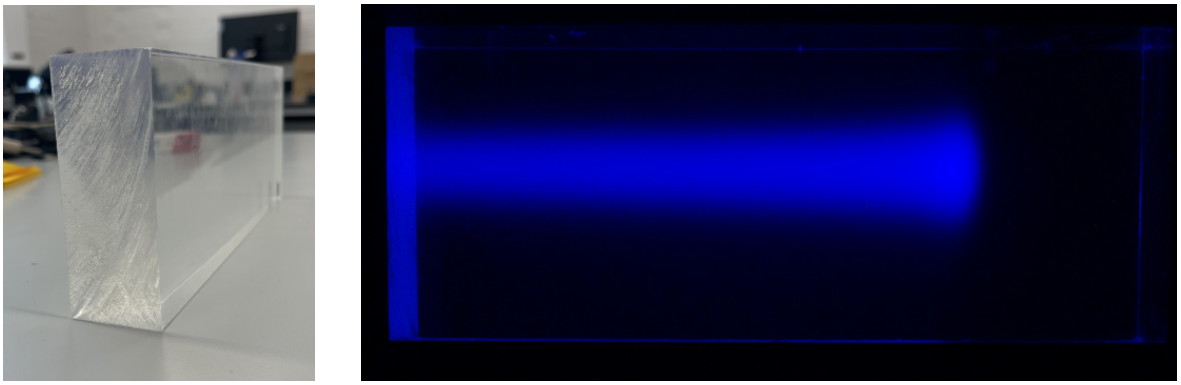


Figure 2: A plot showing the effect of a scratch on the scintillator is shown. The effect can be seen as a deviation from the model around pixel 2075.



(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.

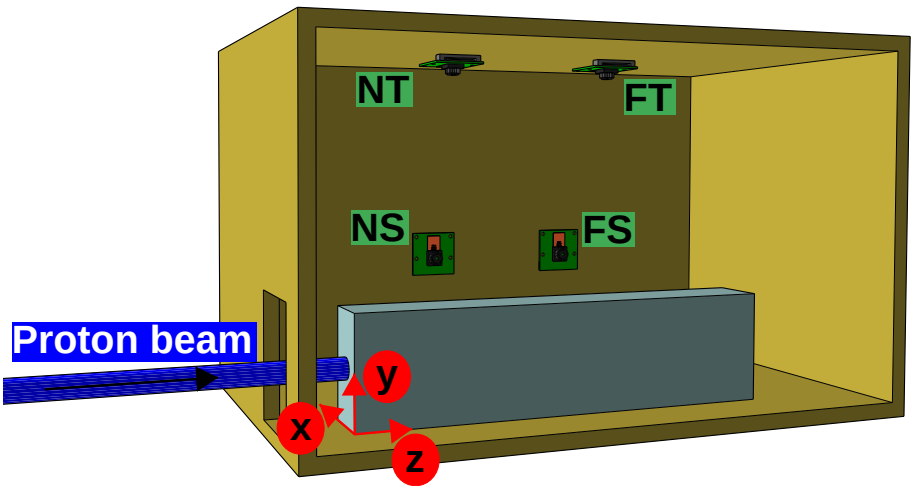


Figure 4: Example schematic for a CRISP box setup.

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-

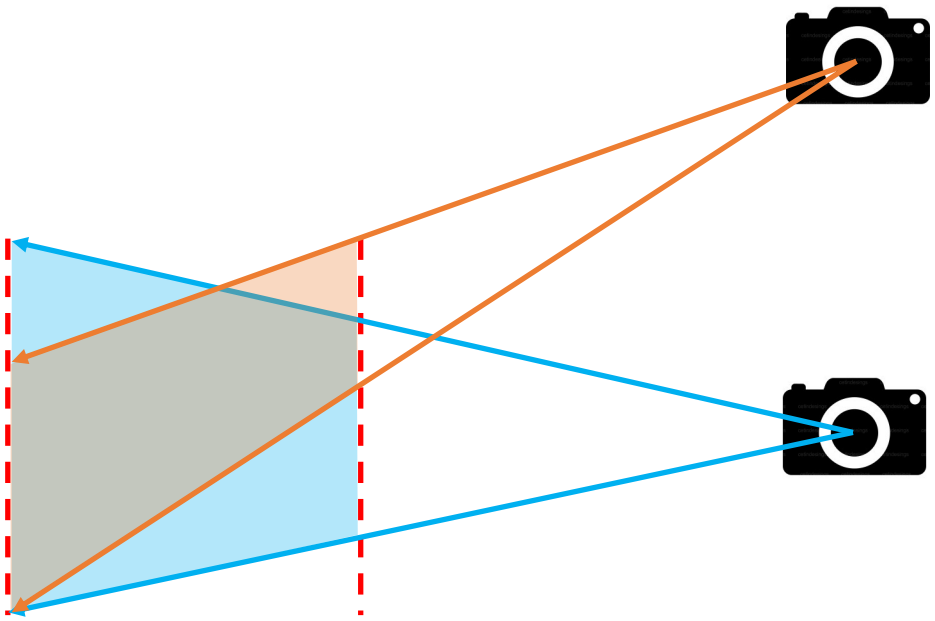


Figure 5: The calibrated regions are shown as shaded regions between two red-dashed calibration planes.

tion. 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. 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.

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. Convention must be followed for pinpointing via homographies to be implemented correctly.

2.6 Calibration boards

Get some example calibration images. Also get 3D rendered boards to demonstrate how they wrap around the scintillator's volume.

3 CRISP Usage Walkthrough

CLICK HERE to help make flowchart of CRISP UX

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

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. Complete a real run
10. Perform analysis

3.1 Open CRISP

First, use the `crisp` command (or Docker compose) 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.

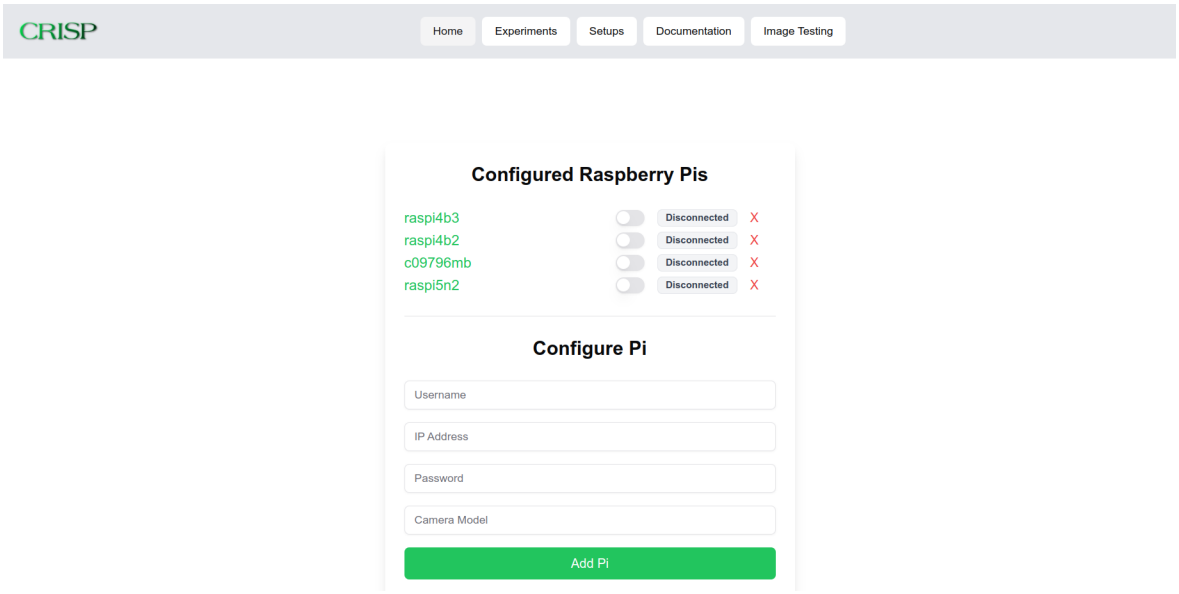


Figure 6: CRISP homepage is shown

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 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. 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.

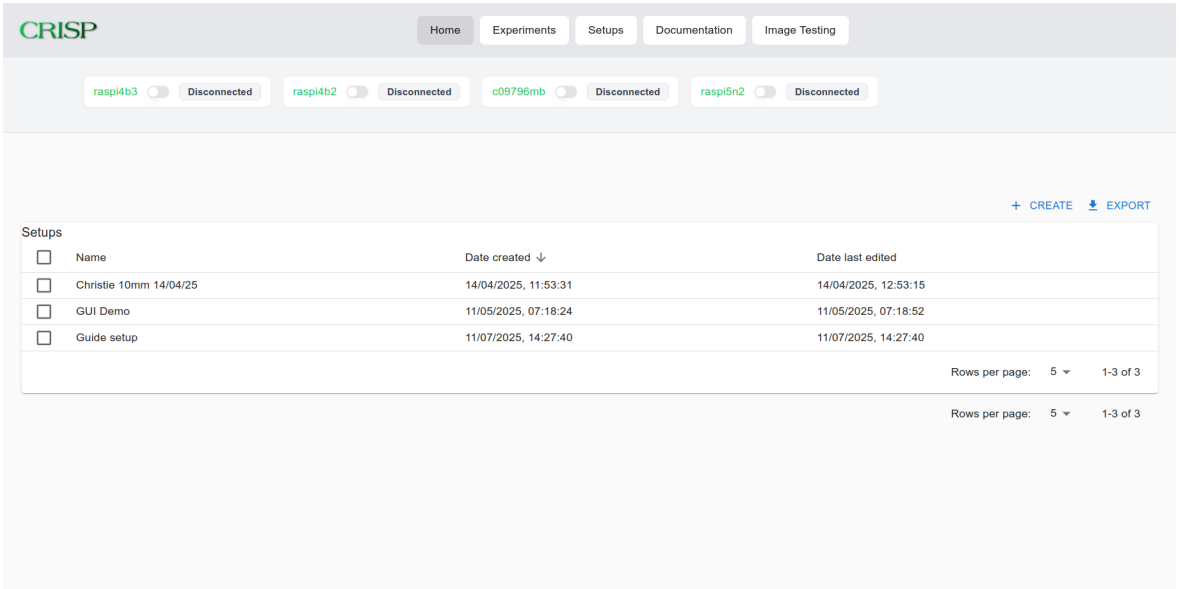


Figure 7: The landing page of the setup tab can be seen to show a list of the current, existing setups as well as a create new setup button.

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.

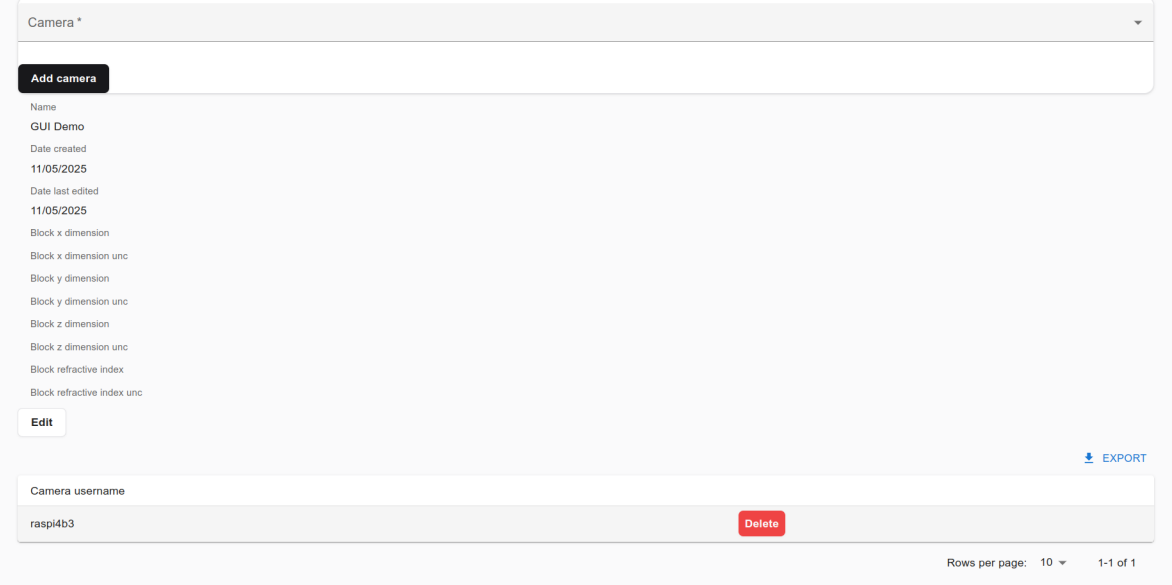


Figure 8: Summary page of a specific setup is shown.

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 8. 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 9) 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.

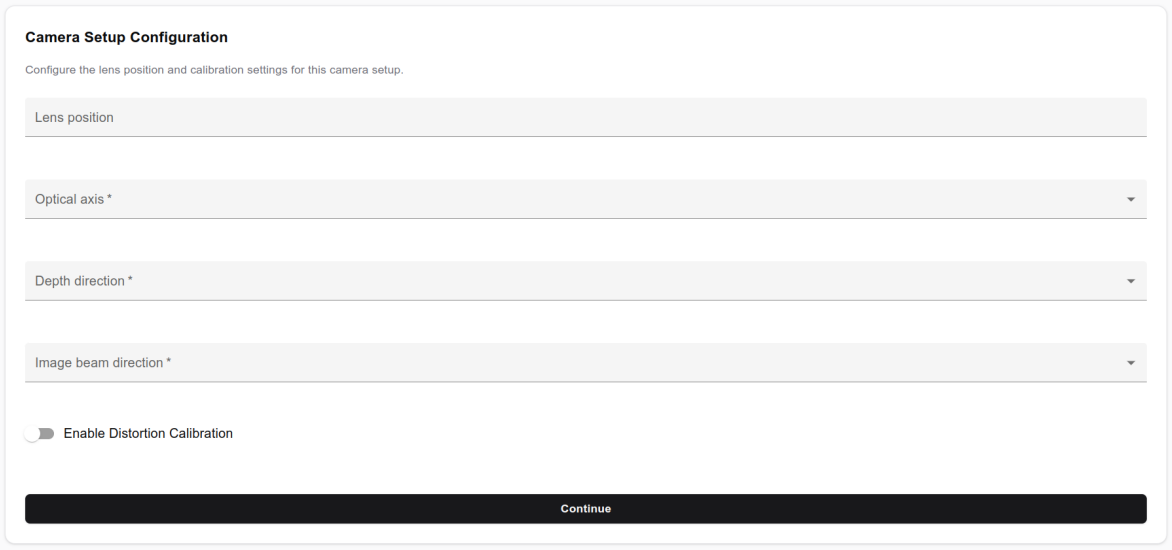


Figure 9: The form which asks the user to define the camera's position (optical axis, depth direction and image beam direction - as defined in section 2.5) is shown. Lens position and enabling distortion calibration must also be input here.

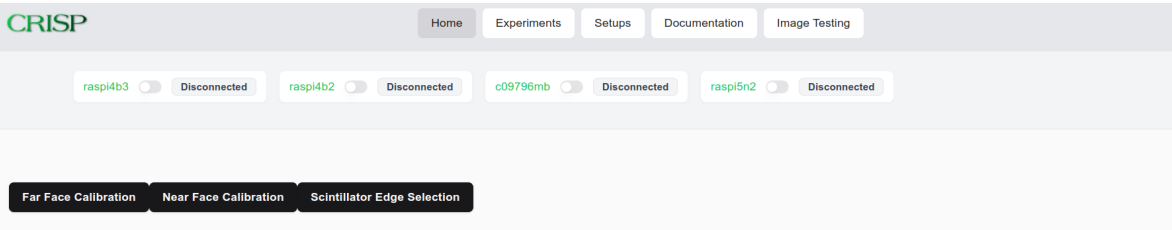


Figure 10: Page which provides links to the 3 necessary calibrations - near face calibration, far face calibration and scintillator edge selection.

Having submitted the form, you will land on the calibration hub page shown in figure 10. 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

3.6.2 Scintillator edge selection

3.7 Create an experiment

3.8 Complete a test run

3.9 Complete a real run

3.10 Perform analysis

4 CRISP Codebase

4.1 CRISP Tech Stack

- Docker
- PostgreSQL
- FastAPI
- Next.js
- React Admin

4.2 Beam Run Imaging Script

Imaging script:

- created to run with Python 3.9.
- Logging
- Arg parser (some hardcoded options like bit depth and maybe format could be changed by future users)
- Selection of colour channel (see in Experiment section)
- "Raw images" - list image settings which are kept simple
- Images of a given pi stored as a tarball which is then sent over Ethernet and saved to the CRISP database. ı

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

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

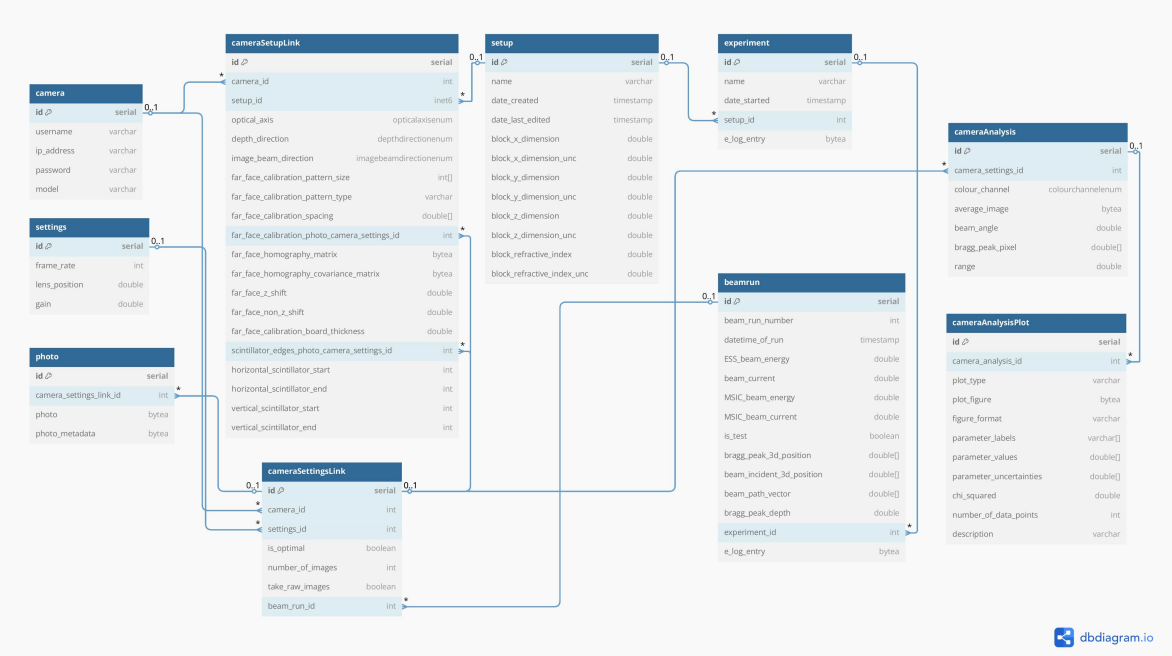


Figure 11: An entity relationship diagram of the database structure used is shown.

D System Architecture

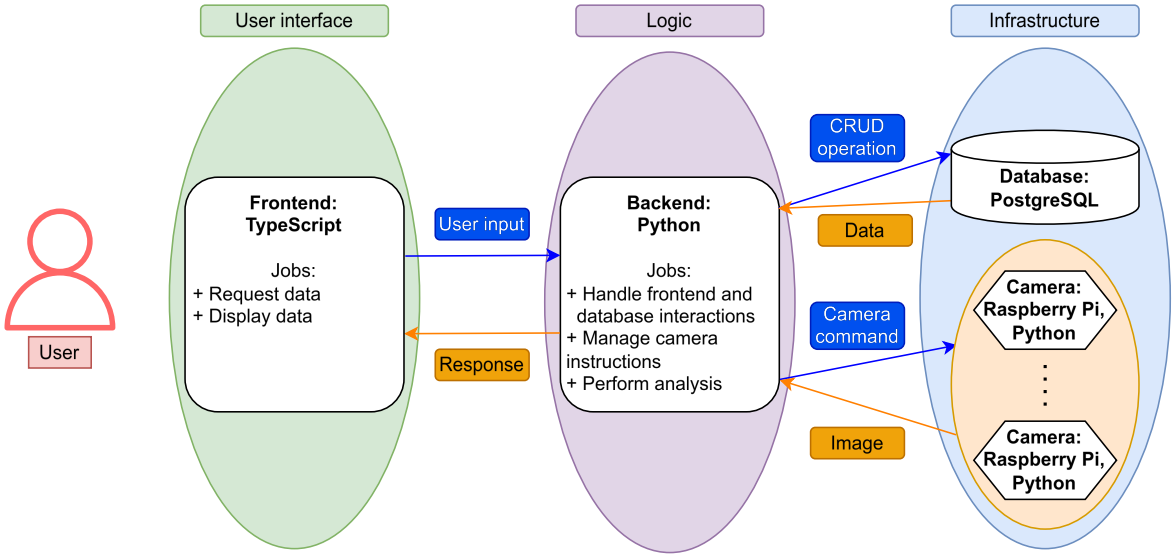


Figure 12: A schematic diagram of CRISP's system structure is shown. A user interface layer can be seen to send user inputs to to a logic layer which can subsequently call Create, Read, Update or Delete (CRUD) operations on a database in the infrastructure layer which returns data. The logic layer can 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.