# Raspberry Pi Beam Profiler

# Users Guide

James Keaveney
james.keaveney@durham.ac.uk

Version 1.0.0

Last updated 2018/01/19

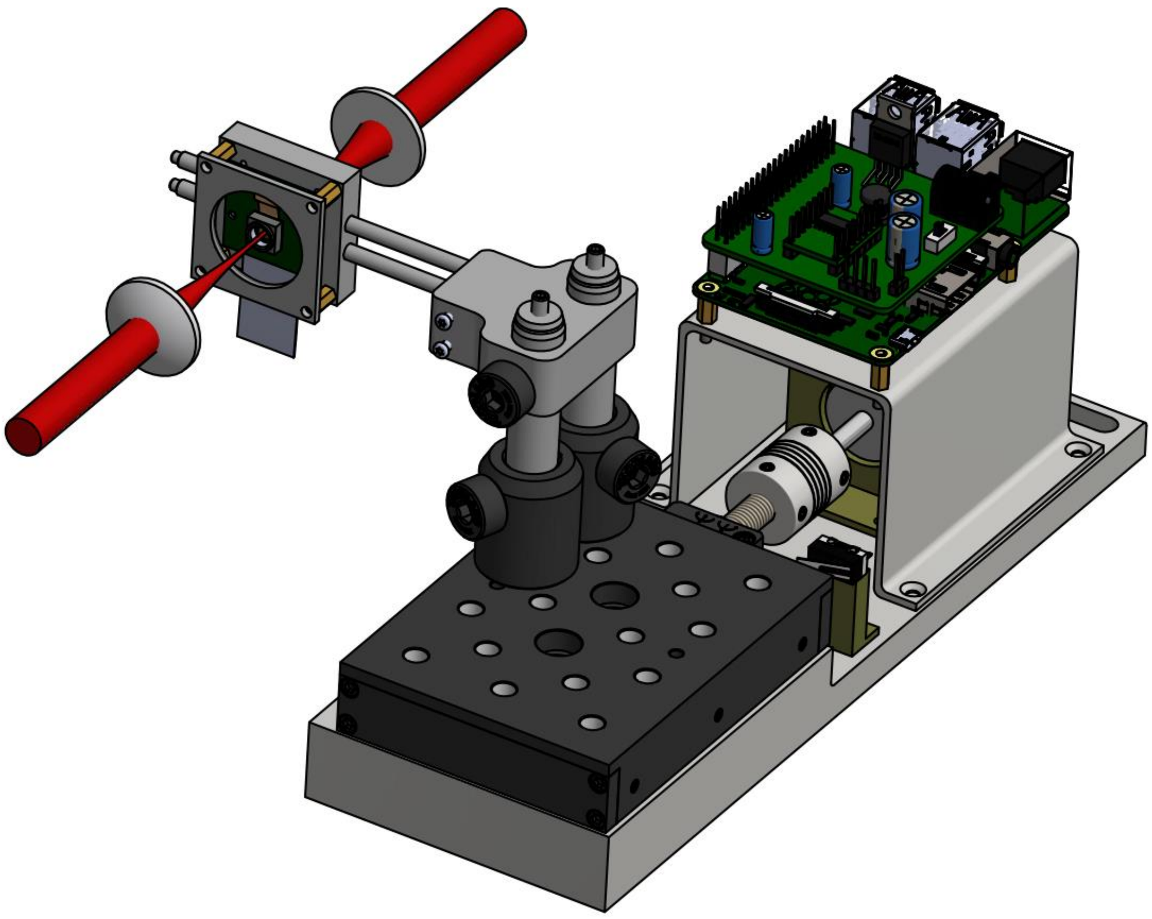
Contents

## 1. Introduction

The Raspberry Pi beam profiler is designed to be a replacement for knife-edging, particularly with one or two fixed-mount lenses where the focal point and size need to be known.

With a conventional knife-edging setup, this task is laborious and time-consuming. With this beam profiler, the idea is to completely automate the process.

It is also possible to perform M-squared measurements of beam quality or further analysis by outputting the raw image data.



## 2. Setting up the hardware

### a. What you will need:

In addition to the main beam profiler assembly, you will need the following parts:

- Raspberry Pi – we recommend the Model 3B version (or whichever is the latest), since it has by far the most processing power
- A micro-SD card with at least 8 GB capacity. We recommend getting a card with a fast read speed, which will maximise system performance.
- HDMI to HDMI / DVI cable, for connecting a monitor
- Monitor with either DVI or HDMI input. Recommended resolution of at least 1440 x 900.
- Keyboard and Mouse (both USB)
- USB memory stick for retrieving data
- DC power supply, with a barrel jack connector, capable of supplying 8-12 V with more than 2A to power the Raspberry Pi and stepper motor

The RPi's local storage is a Micro-SD card. This should be installed, located near the bottom of the board, on the reverse side, as it stands in the beam profiler. We recommend using the Raspbian operating system (see section 3 below).

### b. What goes where?

We recommend manufacturing the PCB designed for this purpose, in which case the PCB pushes on to the Raspberry Pi via the 2x20 GPIO header connection. However, it is not required, and the DRV8825 breakout board can be used by itself with only a few GPIO pin connections (note in this case the raspberry pi will have to be powered separately via the Micro-USB connector on the RPi board itself). For manually connecting the pins, the connection diagram is shown schematically below, along with the pin numbering scheme of the Raspberry Pi (Model B+, Model 2B, Model 3B).

> ***CAUTION**: Incorrect wiring could lead to short circuits which WILL damage the Raspberry Pi. Make sure it's right before turning on, and never connect or disconnect anything while the Raspberry Pi is powered up.*

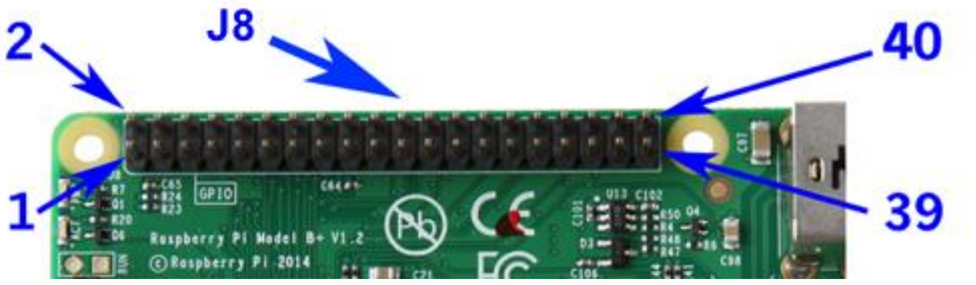


Figure 1: Raspberry  Pi Model B+ GPIO Header pin numbering. The odd-numbered pins are along one side, and pin 2 is located nearest the edge of the RPi board.
Image from http://pi4j.com/pins/model-b-plus.html

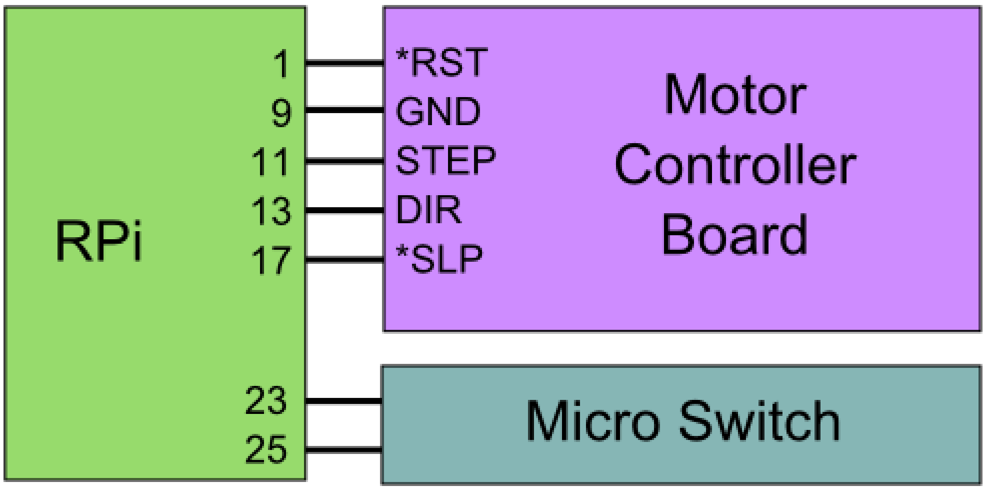


Figure 2: Connection diagram. Pin numbers on the RPi correspond to those in figure 1. The pins on the Motor controller are labelled accordingly.

Connect jumper wires between the pins of the RPi and Motor controller, as shown in figure 2. Connect the stepper motor via the A1,A2,B1,B2 pins of the motor controller board. Finally, connect the micro-switch to the RPi.

The motor controller is a Polou breakout board (product page here), consisting of a DRV8825 chip (datasheet is here). This is capable of driving up to 2.2A per coil. The current is actively controlled, and a potentiometer controls the current limit. If a different stepper motor is used, this current limit can be reset for the new motor (follow the instructions on the Polou website).

## Mounting the CCD

The CCD assembly should be placed in the holder with the ribbon cable pointing down. This is not a necessity, but the images displayed on the software will be rotated 180 degrees. The CCD can face either direction in the translation (z-) axis, in case multiple beams need to be profiled that travel in different directions, for example a counter-propagating setup for 2-photon EIT.

A combination of any of Thorlabs' standard ½" posts (part # TR*xxx/*M) and 16 mm cage rods (part # SR*x*) can be used to set the x- and y-position of the camera mount.

## 3. Setting up the Raspberry Pi

### a. Installing Raspbian

The operating system we recommend is called Raspbian, a version of Debian Linux. These instructions assume this operating system is used.

Download the latest version of Raspbian from
https://www.raspberrypi.org/downloads/raspbian/

Install the operating system to the SD card following the installation guide at
https://www.raspberrypi.org/documentation/installation/installing-images/

Once all other cables are connected, plug in the power cable. There is no On/Off switch on the RPi, so it should start immediately. There are some indicator lights that should light up as soon as you power-up.

The device should boot to a desktop environment. The operating system is a variant on debian, a linux distribution. If you are not familiar with this type of operating system, google it!

### b. Installation of python modules:

After Raspbian is installed, you will need to install some additional python modules which are required to run the beam profiler.

These can be installed via a terminal window through

*sudo apt-get install <module1> <module2> ...*

The modules that are needed are

*python-matplotlib*
*python-scipy*
*python-wxgtk3.0*

You should also make sure that the following modules are up-to-date:

```
python-numpy
python-picamera
python-rpi.gpio
```

All installed modules can be updated by doing the following:

```
sudo apt-get update
sudo apt-get upgrade
```

### c. Enable the camera module

The camera module is not enabled by default. To enable it, open a terminal and type

```
sudo raspi-config
```

Go to Interfacing Options -> Camera -> Enable Camera. Move down to 'Finish' and reboot the Raspberry Pi.

### d. Disable red LED on camera module

To avoid additional noise on the camera, you should disable the red LED. To do this, edit the /boot/config.txt file, using

```
sudo nano /boot/config.txt
```

and add the line

```
disable_camera_led=1
```

to the end of the file. Save and exit (Ctrl+X, then follow instructions), then reboot the Raspberry pi (sudo reboot).

### e. Installation of beam profiler software

Download the GitHub repository, either by navigating to the GitHub repository and downloading the zip file or by using the git command line tool (git should be installed already, but if not then ‘sudo apt-get install git’ in a terminal window). Running

*git clone http://github.com/jameskeaveney/RPi-Beam-Profiler*

will download the repository to the current directory.

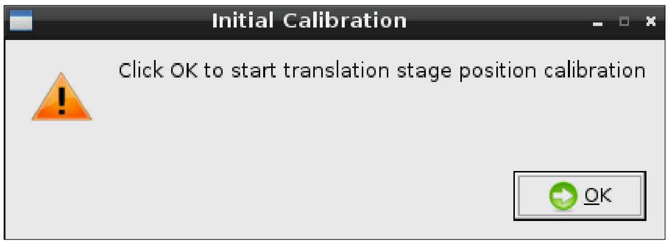
Navigate to the *\software\RPiBeamProfilerApp* sub-directory, and type

*sudo python beamprofiler_guiX.Y.py*

where X.Y is the version number.

## 4. Software Guide

On starting the software, the program will try to calibrate the translation stage. The stage is moved backwards (towards the motor) until the microswitch is triggered. This stops the stage moving and is used as the 0.00 mm reference point. The user is presented with a dialog box to confirm this is happening. When the dialog box disappears the program is ready to continue.


Initial Calibration

Click OK to start translation stage position calibration

Initial calibration dialog

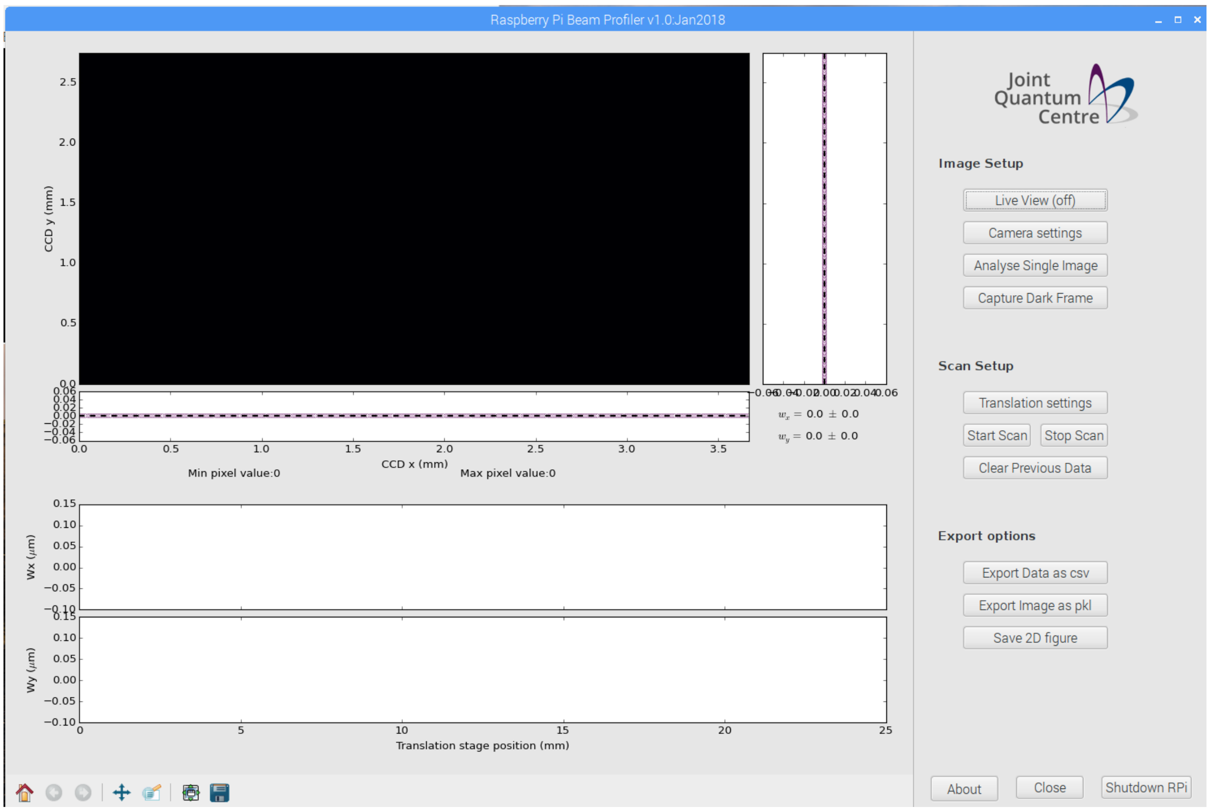
### a. The front panel

The front panel of the program is shown below. On the left side of the screen, the current image is shown, along with the x-axis and y-axis slices (the 2d image array is integrated along one axis to give the slice in the other axis). The y-axis is the vertical axis (direction of the ribbon cable).

Bottom left are the fitted beam waists in x- (top) and y-axes (bottom), as a function of translation stage position (z-axis). This will dynamically update when a translation stage scan is run.

This is a plot using matplotlib, and as such is interactive – any axis can be panned or zoomed using the controls on the toolbar at the bottom left of the screen.

On the right side of the front panel are the various controls, which will be detailed separately below.



The front panel, as it appears on starting the program.

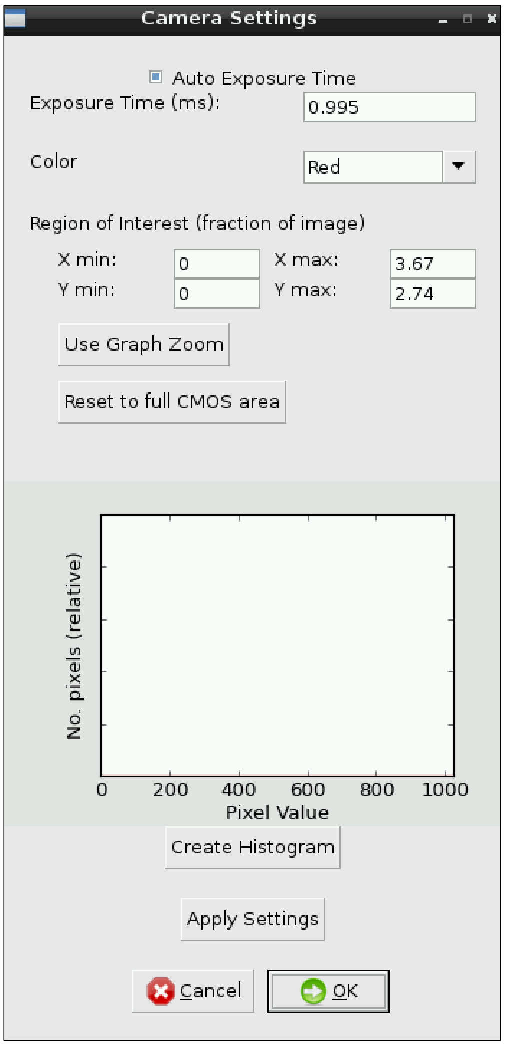
## b. Live View

The first button is called LiveView, and is intended as an alignment aid. It brings up a video feed of the ccd image. Because of the way this preview works, this is a full-color image so may look a bit weird! This is fed through with the default amount of post-processing (there is no option to disable this), so is not representative of the still image quality. The video is also cropped to the central area of the ccd, and is in 16:9 format.

Just to stress the point – the live view is not indicative of the image quality of the still images – these will be much better!

## c. Camera settings

The camera settings sub-menu contains the controls for setting exposure time and area-of-interest. The interface is shown below.



Exposure Time (ms): 0.995

Color: Red

Region of Interest (fraction of image)

X min: 0 X max: 3.67

Y min: 0 Y max: 2.74

Camera settings sub-menu.

The exposure time is controlled either automatically, when the tick-box is selected, or manually through the text entry box (values in milliseconds). The minimum exposure time is 9 or 12 µs (version 1 or 2 sensor), maximum is a few seconds. Auto-exposure works by analysing an image frame and making sure that no pixels are saturated, which is fundamentally different from how normal camera auto-exposure metering works.

The next control is for the color of pixel used. Since the physical sensor has filtered pixels arranged in a Bayer pattern (see section 4), we must select which set of pixels to use, since they have different sensitivity depending on wavelength.

The final control is the region-of-interest (ROI), which essentially crops the image in to a specific area. This makes fitting faster, since there is less data to process, so is recommended for use with small beams.

There are two ways to set these parameters – either by using the text input boxes, or by zooming the graph using the matplotlib toolbar buttons (discussed above). This must be done with the sub-menu closed, so zoom in, open the sub-menu, and then click on 'Use Graph Zoom', which will then update the text boxes with the current axis limits. To go back to the full sensor area, click the 'Reset to...' button.

Clicking 'Apply Settings' will save the current exposure / ROI settings and take an image, without closing the menu.

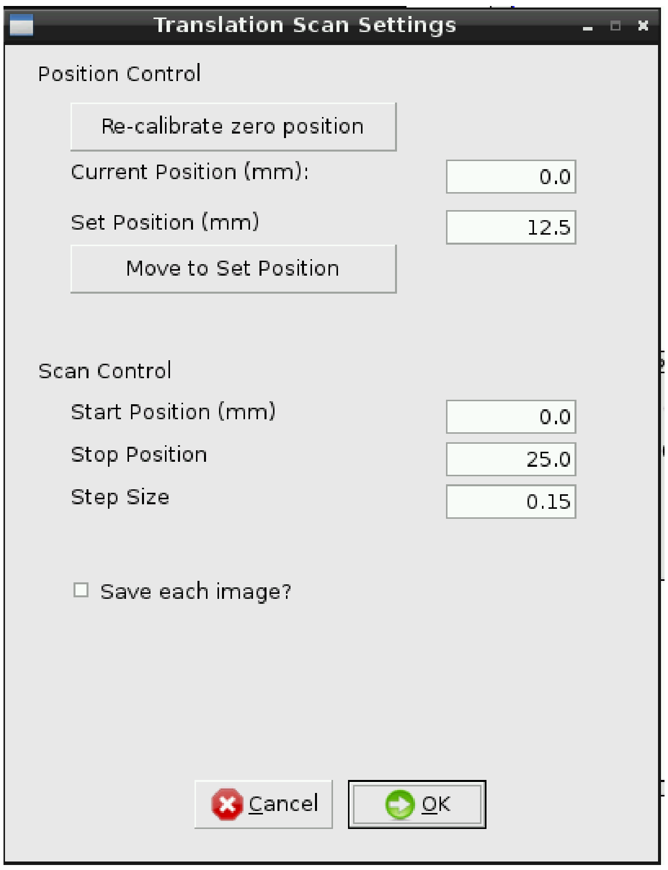
> *NOTE: None of these parameters are saved until the 'Apply' button is pressed. Closing the window without pressing apply will NOT save any changes.*

Finally, there is also the option to look at a histogram of the current image, which may be useful for a quick look at the distribution of pixel values (e.g. checking for saturation when in manual exposure mode)

## d. Translation stage scan settings

The translation scan settings sub-menu controls the position of the translation stage, the start/stop positions of a scan and the step size. The sub-menu interface is shown below.

The translation stage is designed to move a maximum of 25 mm, however the stage becomes difficult to move near the edge of it’s movement range so this is limited (in software) to 0 - 23 mm. The step size can be very small, down to 1 micron, however this may not be enough to overcome static friction so results may be variable!


Translation Scan Settings

Position Control

Re-calibrate zero position

Current Position (mm): 0.0

Set Position (mm) 12.5

Move to Set Position

Scan Control

Start Position (mm) 0.0

Stop Position 25.0

Step Size 0.15

☐ Save each image?

Cancel

OK

Translation settings sub-menu.

There is the option to re-calibrate the zero position, which re-runs the routine that happens on starting the program (see above for details). The current position text box is read-only, and displays (strangely enough) the position relative to the zero of the translation stage. The set position box and button allows movement to a specific position.

The scan control inputs control the starting and final positions of the stage, and the step size.

The ‘save each image’ tick box will automatically save an image at each position of the translation stage. When the start scan button is pressed and this option is selected, the user will be prompted to select a location and filename for the images. A number is appended to the filename for each image, with 0 corresponding to the start of the translation scan. They are saved as pickled (binary) data files – see the section on exporting data for more information.

> *WARNING: If multiple scans are run, the ‘save each image’ option will overwrite existing files with the same name.*

**Running a scan**

To run a scan, click the ‘Start Scan’ button on the front panel. The stage will move to the start position, capture an image and fit x- and y-axis waists. To cancel the scan running, click the ‘Stop Scan’ button. Be aware it may take a little while to respond if an image acquisition / analysis is in progress.

By default the data does not clear between scans, the idea being that the first scan would be fairly coarsely spaced to find the rough focus position and then a finer scan would be performed around the focal point. In this case all the data is still useful, so the plot does not clear. If the data is not wanted, click the ‘Clear previous data’ button to start with a blank canvas.

## e. Export options

The final 3 buttons on the main panel are for exporting data. Data can be saved direct to the RPi’s SD card, but space is fairly limited so a better alternative is to use a USB flash drive.

‘Export data as csv’ exports the two curves of beam waist against position, in order of increasing position coordinate. The output is in csv (comma separated value) format and is therefore readable by spreadsheet programs as well as Python etc. The data is arranged in columns as follows:

*z-position (mm), x-waist (µm), x-error (µm), y-waist (µm), y-error (µm)*

The ‘Export image as pkl’ button exports the current image as a python pickle file, which is a binary output format for quickly reading in to python again. Saving the image as a 2D csv file is computationally expensive for such a little computer… (the file sizes are ~4 MB, which takes over a minute to write). If csv files are required, they can be written using a separate python script called *pkl_to_csv.py* located in the /beamprofiler directory

The script *pkl_to_image.py* reads in a particular image and produces a matplotlib plot similar to the one in the beamprofiler app. The plot is automatically saved in both png and pdf format. Run this from the command line specifying the file to be read. For example,

```
python pkl_to_image.py imagefile.pkl
```

will read in the file *imagefile.pkl* and produce two new files, imagefile.png and imagefile.pdf.

The pickled image files can be read into another python script using the following:

```
import cPickle as pickle
IMG = pickle.load(open(<FILENAME>,’rb’))
```

where *IMG* is a numpy 2d-array and *<FILENAME>* is the filename (including path location if necessary) of the *.pkl* file.

Finally, ‘Save 2D figure’ saves the left-hand side of the front panel (everything apart from the button bar) in the same way as matplotlib saves figures. This has the same function as the save icon on the matplotlib toolbar.

## 5. Image analysis methods

The images are fitted to 1D Gaussian functions of the form

$$I(x) = A\exp(-\frac{2(x-c)^2}{w^2})$$

Where *x* is the horizontal or vertical coordinate, and *w* is the 1/e$^2$ radius.

After the translation stage scan, the profile is fit to a function of the form

$$w_{x,y}(z) = w_0\sqrt{1+\left(\frac{(z-z_c)}{z_R}\right)^2}$$

Fitting is done via the *curve_fit* method which is part of *scipy.optimize*. Errors in all fit parameters are found from the covariance matrix (square-root of the diagonal elements).

## f. Program information

The program is written entirely in Python, and uses the following modules:

*wxpython* – GUI

*matplotlib* – plotting

*numpy, scipy* – numeric, arrays, fitting routines

*RPi.GPIO* – raspberry pi specific module for interfacing with the GPIO (general purpose input/output) pins

*picamera* – raspberry pi specific module for interfacing with the camera

The program is entirely open-source. It is therefore possible (and encouraged!) to add further functionality to the software yourself. Should you do so, please fork the GitHub repository to make your improvements publicly available.

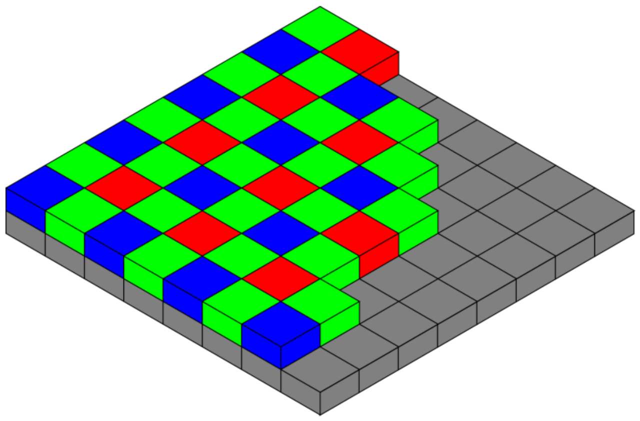
## 6. **Hardware specifications**

The camera is the PiNOIR camera, which is a normal CMOS camera, but crucially without an infra-red filter, which allows us to use the full wavelength sensitivity range of the silicon sensor – should be roughly 300 – 1100 nm (not tested!). It's therefore bounded by the usual limits of silicon, and is more sensitive in the region of around 750 nm.

### *A. Camera specifications:*

Sensor area: 3.67 x 2.54 mm
Total number of pixels: 2592 x 1944 (v1), 3280 x 2464 (v2)
Pixel pitch: 1.4 µm (v1), 1.12 µm (v2)

Bayer pattern:



Bayer Pattern. Image from Wikipedia.

Like all colour sensors, the raw data only has ¼ the number of total pixels for red and blue, and ½ the number of total pixels for green. Since we want the raw data, we have two options – either interpolate back to the full resolution of the camera, or use just the R, G or B pixels and lose a factor of 2 in resolution.

### *b. Translation Stage Specifications*

The translation stage is a Thorlabs PT1/M, with a custom 0.5 mm pitch screw thread. This means that one revolution moves the stage by 500 µm. The stepper motor (datasheet here, model number ST4209S1006-B) has a single step resolution of 0.9 degrees, so in theory the system has a spatial step size of 500 x 0.9 / 360 = 1.25 µm.