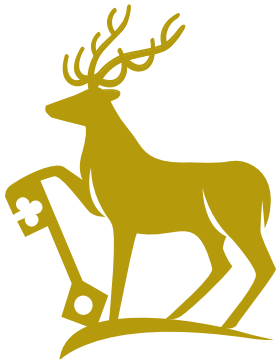

UNIVERSITY OF SURREY

# SMART FLASH

An all in one illumination system for DSLR cameras for photometric image capture and processing

Jackson, William (UG - Elec Electronic Eng)
Wj00084@surrey.ac.uk

## Abstract

This paper proposes a novel addition to a standard DSLR camera to allow the capture of images suitable for photometric computation. The proposed solution mounts to the camera in two stages. The first component is an 8-source light which is mounted to the lens of the DSLR which provides the differential illumination at $45^o$ for photometric capture. The second system is mounted to the hot shoe of the DSLR and contains a raspberry pi 4, 5v battery, and LED power circuit, furthermore this system can calibrate its light sources using a planar mirror/ChArUco board combination. Most current photometric capture methods are costly and static in order to provide high accuracy normal map calculation. The system aims to provide the ability for mobile photometric capture and basic normal map generation while also keeping the cost comparably lower than standard photometric systems.



## Acknowledgments:

I would like to thank my supervisor for this project, Dr Jean-Yves Guillemaut, whose advice and support proved invaluable throughout this project and the unprecedented circumstances in which it was completed.

## Table of Contents

# Table of Figures:

# Introduction:

This project aims to deliver a “Smart flash” module which will allow the capture of images appropriate for computation using photometric stereo. Photometric Stereo is a method, which builds on the “shape from shading” method proposed in 1970 [1] , used to recover surface orientation from an image based on the shadows created from controlled illumination. Unlike binocular stereo, which uses multiple cameras and a single light source, photometric stereo only uses a single fixed camera with multiple light sources located at various angles around the lens. Due to the lack of complex equipment photometric stereo can be simple to implement in its most basic form, with some implementations going as far as to use a laptop screen and webcam as the illumination and capture methods.

Photometric stereo has a variety of potential applications in the worlds of fingerprint scanning and facial recognition due to its ability to obtain low error scans of rough surfaces. By using 3D scans, you allow the potential for higher matching accuracy when compared to using 2D data, one can also verify that the data being scanned is a 3D object and not a 2D image.

The Smart Flash module described in this report aims to be a low-cost alternative to conventional photometric scanners by making use of common commercial parts such as DSLR cameras and Raspberry Pis. The project aimed to be completed across a single academic year in conjunction with other modules leading to compromises needing to be taken when allocating time and prioritising parts of the project which would provide the most function.

The Smart Flash created in this project mounts to a DSLR camera in two 3D printed parts. The first mount attaches to the lens of the camera and houses the 8 LEDs used for a single capture cycle, however the number of LEDs used can be reduced to improve cycle time and reduce computation time. The second mount attaches to the hot shoe on top of the DSLR and houses a battery, LED power circuit, and Raspberry Pi, which is used for LED control and processing of the captured images, the two modules are connected using a 9-pin ribbon cable from the LED power circuit to the routing circuit of the LED lens mount. This produced system addresses most of the following objectives of this project:

1) Design and build a mount for a DSLR camera which should include:
    - A lens mount to house the 8 LEDs for photometric capture.
    - A hot shoe mount to hold the Raspberry Pi, Battery and LED power circuit.
2) Programmed a control system for the DSLR which should:
    - Initiate a synchronised capture cycle using the DSLR and LEDs.
    - Have separate branches for calibration and capture stages.

3) Integrated a basic photometric stereo algorithm which should:
   - Be calibratable using an integrated system.
   - Produce a normal map based off the data captured.

The report produced outlines the following sections:

- A related works section which discusses variations of basic photometric stereo including calibration, unique and novel capture platforms, and real-world applications.
- A system Development section which includes:
  - The steps taken to design hardware which can produce differential lighting conditions for photometric stereo.
  - A description of the software produced which can control the LED lighting cycle, calibrating the lighting directions of the LEDs, and producing a normal map based of captured data.
- A list of References.
- An appendices section which includes project administration (GANT charts, Hazard assessment), code and results for the proof of concept prototype, and full code for the final solution.

# Related Works:

## Basic Photometric Stereo:

Photometric stereo is a technique which was first put forward by Woodham in 1980 [2], which builds on the idea of shape from shading proposed by Horn in 1970 [1]. He describes a method in which the surface normal and reflectivity of an object can be recovered from images taken by a fixed camera under changing light sources in non-planar angles around the camera.

The photometric stereo algorithm proposed by Woodham has many assumptions, for instance, the equations assume the object can be described by a Lambertian model which allows the algorithm to assume that the intensity of light observed at a point is proportional to the angle between the surface normal and the light source direction. Furthermore, basic photometric images need to be captured in dark environments where the only illumination is from light sources used in the capture device, this simplifies calculations as intensities observed in a single image can be assumed to only be as a result of the known illumination vector. The final assumption is that the image captured follows the orthographic projection model

These assumptions exist to make calculations easier however result in errors in the produced normal and reflectance maps as a result of real-world effects such as the near-light case discussed later. Research into photometric stereo techniques allow methods to be generated which reduce the amount of assumptions made.

Basic photometric calculations start with the assumption that at the same point the reflectance map and intensity are directly related. It has been found that the equations for reflectance and intensity are nonlinear, therefore, to solve for a single solution at least three illumination directions are required:

$$I_1(x, y) = R_1(x, y) \quad (1)$$

$$I_2(x, y) = R_2(x, y) \quad (2)$$

$$I_3(x, y) = R_3(x, y) \quad (3)$$

All these intensities at a single point can then be represented as a single column vector:

$$\tilde{I} = [I_1 \quad I_2 \quad I_3] \quad (4)$$

Furthermore, a single column vector of measured illumination vectors can be constructed from a separate calibration stage:

$$\widetilde{l_1} = [l_{11} \quad l_{12} \quad l_{13}] \tag{5}$$

$$\widetilde{l_2} = [l_{21} \quad l_{22} \quad l_{23}] \tag{6}$$

$$\widetilde{l_3} = [l_{31} \quad l_{32} \quad l_{33}] \tag{7}$$

$$\tilde{L} = \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \tag{8}$$

These matrixes can be combined using the orthographic projection model to form the equation:

$$\tilde{I} = k(\tilde{L} \cdot \breve{n}) \tag{9}$$

Where $\tilde{n}$ is the surface normal at a point (x,y).

This can be rearranged to provide the surface normal:

$$\tilde{n} = \frac{1}{k}\, \tilde{L}^{-1}\tilde{I} \tag{10}$$

This equation can be further simplified to:

$$\tilde{n} = \tilde{L}^{-1}\tilde{I} \tag{11}$$

By doing this method for all illumination directions and all pixels in an image the image can be formulated into a normal and depth map.

In practical terms the intensities can be acquired from the 8bit binary number which represents a single pixel and normalized from the max intensity value of 255 to form a multiplier which attenuates incident light.

## Photometric Calibration:

Conventional photometric stereo techniques can fall into two categories: Calibrated and uncalibrated. Calibrated photometric stereo adds a stage to the algorithm in which all the illumination vectors are experimentally measured for use in calculations. In Woodham's original paper he states that "Once calibrated, however, photometric stereo can be reduced to simple table lookup and/or search operations" [2, p. 142], making the addition of a calibration stage ideal for reducing the computational cost of the algorithm; this stage is important for the proposed solution due to the limited computational power of a Raspberry Pi.

In calibrated photometric stereo, obtaining accurate results for the illumination vectors is vital for producing accurate normal and depth maps, as a result of this many techniques have been proposed for photometric calibration. The most common technique is to use a specular sphere of known radius; in other sources it is suggested [3] [4] that to reduce error in calibration using near-light sources, instead of using a chrome sphere, a flat planed mirror of known distance to the lens can be used.

An example of calibrating practically is found in [5]. The first stage in calibrating using a specular sphere is to capture images where only a single light source is illuminating the sphere. The next step is to observe: the pixel coordinates in the image of the point of highest intensity, "*P*" (which assuming no other light sources will reside at a point on the sphere) and the coordinates of the centre of the sphere, "*C*". Once found, the illumination vector can be calculated using the formulas:

$$L = 2(N.R)N - R \tag{12}$$

Where:

$$\mathrm{R} = [0 \quad 0 \quad 1] \tag{13}$$

$$N_x = P_x - C_x \tag{14}$$

$$N_y = P_y - C_y \tag{15}$$

$$N_z = \sqrt{\left(R^2 - N_x^2 - N_y^2\right)} \tag{16}$$

Uncalibrated photometric stereo is a technique in which the algorithm is entered without knowing the prior illumination conditions. This has its applications in advanced problems such as natural light conditions which are difficult to calibrate for. By removing the calibration, the computational cost of the system is increased as well as the errors, leading researchers to explore uncalibrated techniques which can perform to the level of calibrated photometric stereo. One such technique was proposed in [6] which uses a "divide and conquer" approach which firstly breaks down the images into square patches of pixels. These patches then have their normals estimated using a standard photometric stereo algorithm, these results have a range of values due to an unknown variable which is the rotation ambiguity of each patch. Once the patches have a normal estimation the patches can be connected to solve for the rotational ambiguity.

Calibration in photometric stereo allows the algorithm to be reduced to a simple lookup function, however this can also be applied to uncalibrated methods using a technique proposed in [7] which

suggests using a database of BRDFs which include a wide variety of potential materials. These data sets can provide the normals by intensity matching between the database and measured intensities.

## Near-Light Photometric Stereo

Most traditional photometric stereo techniques require the distance between the light sources and object to be far away so in the context of the calculations they can be considered as infinitely far, this allows the calculations to be simplified as all light rays can be modelled as parallel to one another, allowing the equations to be solved linearly. In near case scenarios, such as the solution that has been proposed, all light sources are at a different angle with respect to the object, causing lighting conditions to vary from point to point. Under near-light sources the illumination assumptions need to be changed as luminance of the light sources will change at the surface due to attenuation with distance. Figure 1 shows how in near-light cases the angle of the incident light differs to that of distant lighting, resulting in different measurements for intensity, causing larger surface normals to be calculated.

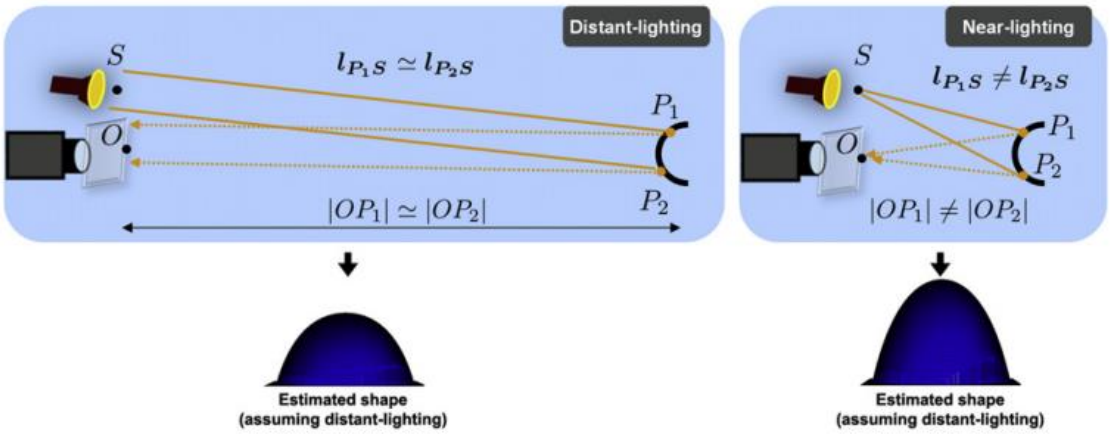


*Figure 1: Effects of distance assumption on estimated shape [8]*

In [3] they propose a photometric solution where the scene is represented as a triangulated 3D mesh where the vertices are positioned at the point of each pixel. This approach reduces complexity in cases where beforehand the surface normals would be represented in 3D derivatives of depth. These vertices are optimized in two stages: Firstly, each vertex normal is optimised using a photometric algorithm coupled with a differential lighting source, in this case a 24 LED ring light. Secondly, the newly computed vertex normals are compared against the original raw image to be further refined. A somewhat similar method is to convert a pixel into a quadrangular facet [9] where the four corners become boundary vertices, this solution can then be solved by mesh deformation techniques [10].

In most cases of near-light photometric stereo the Lambertian reflectance model needs to be reformed to better represent near-light illumination. In many examples [3] [11] [12] the reflectance model gets attenuated by the inverse square law such as Figure 2,which shows how in near-light situations the distance from the light source effects the observed intensity.

$$o_i = E\rho\frac{\boldsymbol{l}_i \cdot (\mathbf{R}_i\boldsymbol{n})}{|\boldsymbol{l}_i|^3} + a,$$

*Figure 2: Equation for modeling near-light intensity [11]*

## Mobile Solutions and Capture Platforms:

Photometric stereo is a technique that requires a large amount of computing power, in [3] the algorithm for near-light photometric stereo is quoted as taking 5 minutes for the processing of 24, 968 x 608 images on an Intel Core-i7 5940. This large processing time makes it difficult for a complete mobile photometric stereo system to be developed which handles the capturing and the processing while maintain high accuracy; however, purely mobile capture solutions are usable without sacrificing on accuracy, provided the algorithm used for post processing is self-calibrating or the capture takes an extra image for calibration later.

Microsoft researchers [11]proposed a mobile solution which combines techniques from binocular stereo and photometric stereo. The solution in Figure 3: Mobile capture method for photometric stereo uses Point Grey DragonFly camera capable of capture images at a 640 X 480 resolution and a single point light source at a fixed position to the camera. The system can use a single light source for photometric stereo through its pairing with binocular stereo [13], in moving the camera for different perspectives they simultaneously provide the differential illumination required for photometric stereo and different perspectives for binocular stereo. Due to the nature of the capture method the algorithm requires a near-light photometric algorithm for accurate results.

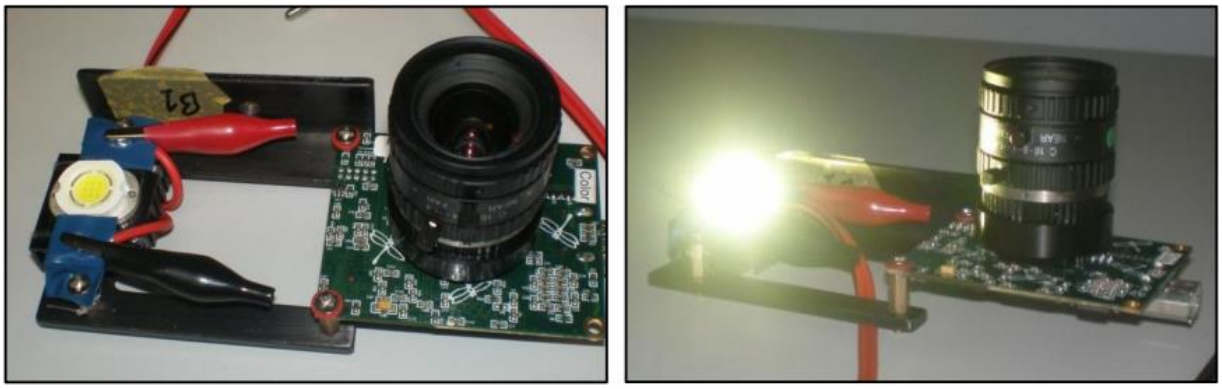

*Figure 3: Mobile capture method for photometric stereo [11]*

The mobile capture method above can be applied easily to a mobile phone, using the flash as an always on point light source. Chia-Kai Yeh et al. [14] proposed a method to convert an iPhone 6 into a photometric capture device using a single 3D printed component as seen in Figure 4. This component plane polarizes light from the camera flash, the device also polarizes incoming light to the camera in the perpendicular plane to the flash. The polarization filters used in the design are to separate diffuse and specular reflectance thus reducing the error in scanning two-layer modeled materials such as skin. In order to scan, the system needs to be moved in an "S" pattern across the surface to provide the differential lighting conditions needed for photometric stereo. In order to feed good data into the uncalibrated photometric algorithm used the captured images need to be normalized, this is achieved by focusing all images around a common point thus matching all potential surface normals.

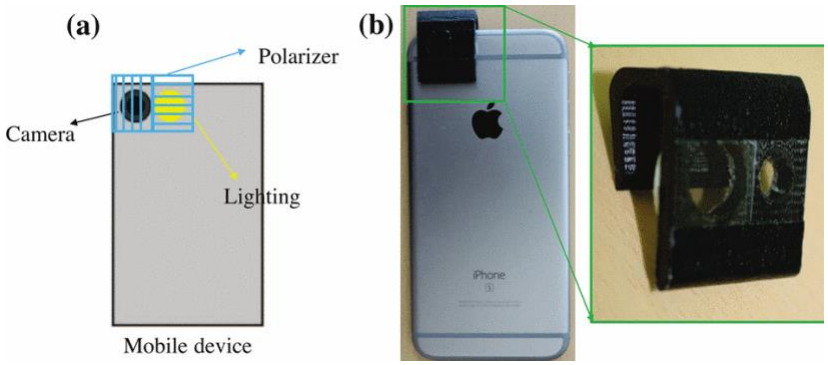


*Figure 4 : Photometric adapter for iPhone 6 [14]*

A further example of a novel all in one mobile photometric solution was put forward in 2014 [15]. This arrangement requires the use of a single laptop for both capture (Utilizing the laptop's webcam) and processing. The illumination for photometric capture is achieved by lighting sections of the screen in different formations to simulate the LED formations of standard photometric capture devices, a scheme for which is seen in Figure 5. By using an LCD screen, it is possible to mitigate the

effects of near point light sources as was proven by Clark [16], who states that (when describing an LCD display light source) “such a light source illuminating a small Lambertian surface patch is equivalent to a single isotropic point light source at infinity, in the absence of shadowing”.

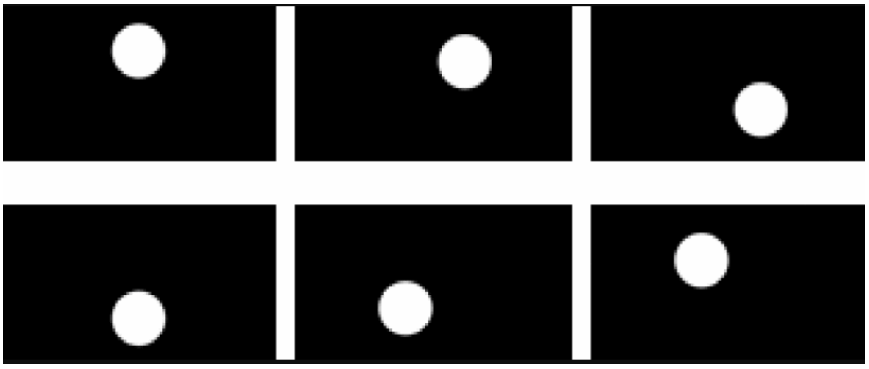

*Figure 5: Illumination conditions for novel photometric capture using a laptop screen [15]*

Most photometric platforms are statically built. In static capture the number of illumination directions is fixed, requiring the system to be prebuilt to accommodate the amount of illumination directions. A drawback of this approach arises when the user wants to reduce the error of the scan beyond that which is capable from the number of light sources used. Hernández- Rodríguez and Castelán [17] proposed a mobile form of capture device in which used a single light source attached to 5 dof robotic arm (Figure 6). In this configuration the robotic arm can move around the capture device in (x,y,z) coordinates, as well as vary the angle of the light source, which provides unlimited potential illumination conditions.

*Figure 6: 5 Dof robotic arm configuration for photometric stereo capture [17]:*

By using a configurable robotic arm, one can ensure that each illumination condition is appropriate for accurate photometric stereo. With this setup it was found that the best lighting conditions for photometric stereo were when firstly, the illumination field of the light source exposes every point on the surface and secondly, when the light source direction was regulated to avoid artifacts from

ambient light. The first condition mentioned is not consistently achievable in static capture because, in order to ensure that a single light source encapsulates the surface all other light sources are changed in the process, resulting in sections of the lighting cycle providing bad data. For fixed capture platforms it has been shown that the slant of the lights can be optimized to reduce the error in calculations for cameras which produce noise [18], and that the angle of this slant is a function of the number of illumination conditions present in the calculation as shown in Figure 7.

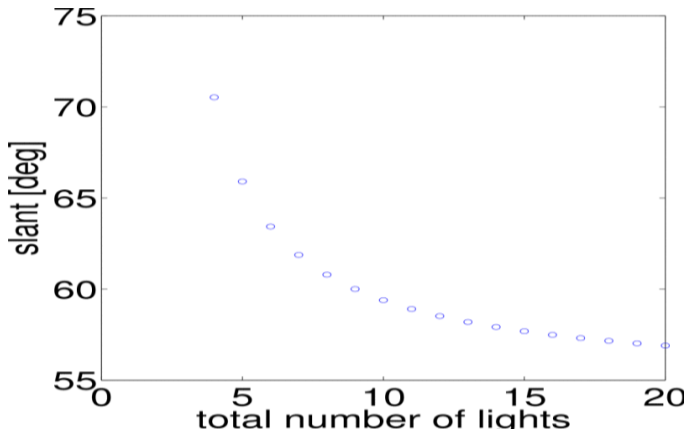


*Figure 7: Graph to show effect of total lights on optimal slant of lighting*

## Applications:

Due to the simple nature of capture and computation photometric stereo has applications in fields dominated by expensive equipment. One example of this is a novel fingerprint scanner proposed in [19]. This system uses a ring of LEDs and a single camera to capture data. The purpose of this system is to scan skin which poses a problem as skin cannot be modelled as a Lambertian surface, instead when exposed to light skin exhibits properties which are both specular and diffuse. As light rays penetrate the skin a portion of them are reflected and a portion are redistributed within and emitted back at the sensor making calculations of the surface normals difficult. To counter this Wuyuan Xie et al. modelled the skin using the Hanrahan-Krueger model of a two-layer surface [20]. The application of photometric stereo in this solution allows the scanner to implement contactless scanning, this improves accuracy over contact scanning by eliminating capture errors which arise from aspects such as dirty contacts and residual prints from previous users [21].

A similar application was proposed in [22] which uses photometric stereo to 3D scan faces for facial recognition systems to improve accuracy, as it has been found that 3D scanning has the potential for higher matching accuracy than 2D scanning [23]. This system improves on standard photometric capture by introducing near infrared light (NIR) as an illumination medium alongside visible light. Hansen et al. states that, when compared to visible light, NIR is less intrusive when scanning while also providing higher scanning accuracy. Similarly, both Wuyuan Xie et al. and Hansen et al. were

required to use skin models to increase accuracy, for this the Oren-Nayar model [24] was chosen. A major contribution is the ability of this system to fulfill a complete capture cycle in 20 ms, this is fast enough to allow the capture of moving objects. To achieve this the setup uses a camera capable of capturing at 200fps which reduces the amount the subject can move between frames. A similar application has also been explored using uncalibrated techniques in [25].

While most photometric stereo applications are applied on scanning singular objects, Liao et al. [26] show how near light photometric stereo can be used to compute an entire indoor scene consisting of multiple objects. The approach suggested improves on standard capture and calibration by having both stages completed at the same time. The capture process put forward details how the scene is computed by using a fixed camera on a video capture setting, a light source is then moved around the room to provide the differential lighting. The calibration for each frame is achieved by placing specular chrome spheres around the environment and using a sphere detection algorithm [27] [28] during the processing phase, after each frame is calibrated the images are then fed through a near-light photometric algorithm. In order to improve accuracy, frames undergo a selection process which evaluates its suitability for photometric stereo when solving for a given pixel as seen in Figure 8. This contribution is ideal for cases which need to reduce the processing cost of the algorithm such as the one proposed in this paper.

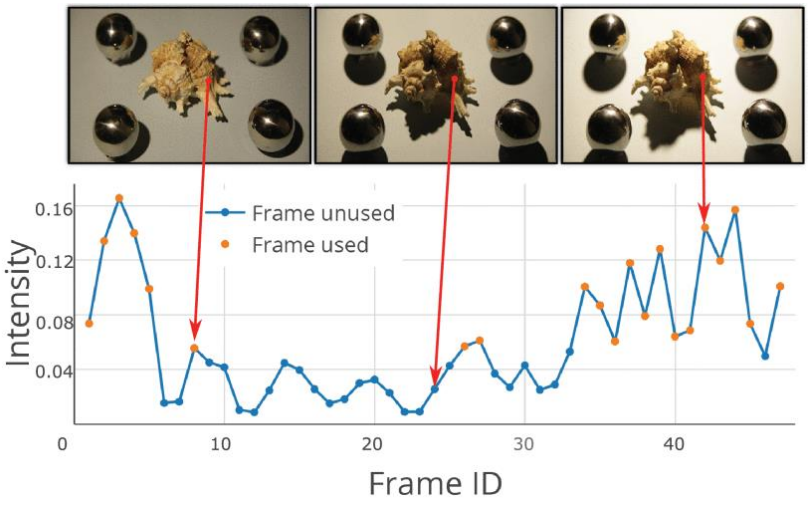


*Figure 8: Frame evaluation algorithm for near-light photometric stereo [26]*

## Summary:

In this section the concept of photometric stereo was introduced and the method for calculating surface normals as proposed by Woodham explained. Various alterations to the basic algorithm proposed were discussed including the differences between calibrated and uncalibrated

photometric stereo, as well as the problems in initial assumptions when working under near-light conditions.

As the solution proposed in this paper is a mobile capture platform, multiple unique capture methods that adapt pre-existing cameras such as smart phones into photometric stereo capture devices were explored. Finally, various applications of photometric stereo are reviewed including the assumptions necessary to bridge the gap between theoretical and real-world calculations.

## System Development

This section steps through the design and development of an all-in-one photometric stereo capture and processing solution which can be adapted to any DSLR Camera.

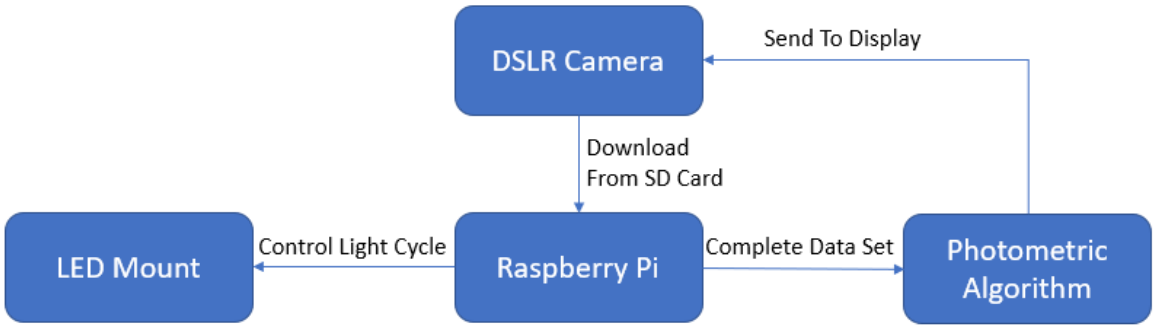


*Figure 9: Block diagram of proposed solution*

Figure 9 shows a block diagram of the final solution. The development for this project can be split into two main parts, the first is the capturing of photometric data which can be considered predominantly hardware focused with a small amount of code required to achieve maximum function. This section is the main priority as the purpose of the project is to provide an all in one solution for photometric capture. The second section of the project is purely software based and focuses on helping users decide on whether captured images would be suitable for further, more accurate photometric computation by providing a basic visualization of the generated depth and normal map. As this section is not the priority of the project it was decided that an open source photometric algorithm [29] would be suitable for providing this pre-visualization instead of developing our own from scratch.

In order to keep a schedule for the project several iterations of GANT charts (Figures 34-36) were created at the start of each semester to help visualize the project workflow and create realistic project expectations based on the time available. Meetings with the project supervisor were scheduled weekly in person, these were then moved to online meetings through the second half of the second semester due to unprecedented circumstances. Before development of the project could be started the project had to go through hazard assessment, the form for which can be found in Figure 37.

The full code used in the development of this project can be found at [30].

## Deciding the Hardware:

### Raspberry Pi:

When deciding the controller to use for the project it was important to consider the processing requirements of the planned algorithm. In the related works section it is mentioned that a photometric algorithm processing 24, 968 X 608 images on an Intel Core-i7 5940 takes 5 minutes to complete making it vital that the chosen controller has the processing power and available memory to successfully complete an operation without triggering a runtime error, because of this the use of Arduinos were not considered however they would potentially be suitable for the lighting cycle portion of this project. The initial proposal for this project was to have the solution mountable to a standard DSLR hot shoe, making size an important factor in deciding the controller.

The fastest commercial controller on the market currently is the Raspberry Pi 4 4GB [31]. This microcontroller is capable of being powered by a 5V battery making it able to be mounted to the hot shoe of a DSLR without the need for a mains power cable while also being small enough to not be overbearing as an all in one solution. The Raspberry Pi 4 also has a vast array of I/O ports including a set of 5V pins which can drive LEDs to provide ample illumination for accurate photometric calculations.

### Lighting Cycle:

It is mentioned in Woodham's initial paper that for photometric stereo to work the algorithm must be provided with a minimum of three lighting directions which are non-planar. To reduce errors, many photometric approaches use more lighting directions than necessary however this increases the computational cost; given the limited processing power of the Raspberry Pi chosen it is necessary to find balance between these two factors. It was decided that the system should incorporate 8 LEDs at $45^o$ positions around the camera as this would provide substantial data for low error scans however can be downscaled to only using 4 LEDs at $90^o$ for circumstances where a faster computing time is a priority.

When choosing the LEDs, it is important to consider the illumination angle. If the angle is too small the illumination across the subject will be non-uniform resulting in poor results. On the other hand, if the angle is too large then the system is wasting energy illuminating areas which are not being captured by the camera. The LEDs used in the prototype are 3.7V white LEDs which produce a luminous flux of 28lm at a viewing angle of $70^o$ [32] which was deemed enough for a wide range of scan sizes and distances. As these LEDs are relatively low power, they will not be suitable for outdoor photometric capture however when powered at 5V they provide ample illumination for indoor conditions.

## Estimated Cost:

Below is a table of estimated costs for all items used to complete this project. Items marked as Preowned are ones which were owned entering the project. 3D printing for the project will be considered as free as the 3D printer used was purchased prior to the start of the project. Assuming anyone interested in this project already own a DSLR camera the total cost of the system is £123.55, including the cost of bulk items such as the 3D printer filament and 24 AWG cables which only need to be purchased once. This price point meets the criteria of the project as a low-cost solution to photometric capture and processing.

| Item | Quantity | Cost | Source |
|---|---|---|---|
| Raspberry Pi 4 4GB | 1 | £44.69 | RS Components |
| Canon EOS 1200D | 1 | £80 | Preowned |
| NPN Transistors | 8 | £2.10 | RS Components |
| 3.7V White LED | 8 | £5.77 | RS Components |
| Raspberry Pi Battery | 1 | £13.99 | Pi Supply |
| Circuit PCB | 5 | ~£5 + £20 Shipping | Elecrow |
| Pi Mounting Screws | 6 | ~£1 | Preowned |
| 24 AWG Cables | 1 | £13 | Preowned |
| 3D Printer Filament | 1 | ~£18 | Amazon |

## Raspberry Pi to Camera Control:

The DSLR being used is a canon EOS 1200D. To control the DSLR the Raspberry Pi is connected via a micro USB data cable. The Pi is also responsible for controlling the lighting cycle via the available GPIO ports and then later responsible for basic image computation with regards to photometric stereo. To control the Raspberry Pi, we will be using a library called “gphoto” [33]. This library exists natively in the Raspberry pi’s command line as well as being importable into python and allows the Raspberry Pi to trigger photo capture using the camera’s current capture settings as well as download photos currently available on the camera’s memory card.

The first step in controlling the DSLR is to kill the gphoto2 process which is launched whenever the Raspberry Pi detects the DSLR has been plugged in, without this step any attempts to trigger capture using the library will result in an error. Figure 10 shows a function used in the final version of the capture cycle, the function searches through all currently running processes on the Raspberry Pi to find the initial “gvsfd-gphoto2” process and terminate it, once complete images can be captured.

```
def killgphoto2Process():
    p = subprocess.Popen(['ps', '-A'],stdout = subprocess.PIPE)
    out, err = p.communicate()
    #search for kill line
    for line in out.splitlines():
        if b'gvfsd-gphoto2' in line:
            pid=int(line.split(None,1) [0])
            os.kill(pid, signal.SIGKILL)
```

*Figure 10: Initial launch kill process function*

When first setting up the ghoto2 library all images captured using this command are saved to the camera's internal RAM instead of the SD card where they can be later accessed. To change this the config was altered in the command line using the command "gphoto2 --set-config capturetarget=1", where option "0" is the camera's RAM and option "1" is the local SD card. The next stage is to capture images, this is accomplished using the command: "gphoto2 --trigger-capture" in the terminal which now saves to the SD card. Once an image is captured it is then saved to the Raspberry Pi's internal memory in the current working directory using the command "gphoto2 --get-all-files". To save storage on the DSLR's SD card, once images are uploaded to the Raspberry Pi the SD card is then cleared using the command: "gphoto2 --folder="/store_00020001/DCIM/100CANON" -R --delete-all-files".

To make files easier to import during the processing phase all files captured during a capture cycle are formatted to a uniform scheme which identifies the time the capture cycle was initiated and the position in the lighting cycle the image was taken in, the process for this is shown in Figure 11. In the final version of this project this system is altered so that the files are just renamed to their position in the capture cycle, the folder they are stored in becomes the identifying factor and is named with the date and time that the capture cycle is started.

```
def renameFiles():#Renames files to a known set for later processing
    ID = 0
    for filename in os.listdir("."):
        if len(filename) < 13:
            if filename.endswith(".JPG"):
                os.rename(filename,(shot_time +"-"+str(ID)+".JPG"))
                print("Renamed the JPG")
            elif filename.endswith(".CR2"):
                os.rename(filename, (shot_time +"-"+str(ID) + ".CR2"))
        ID += 1
```

*Figure 11: Function to rename files in a capture cycle*

It is important to set the exposure time of the camera to less than one second as any longer than this in the current configuration the gphoto library will attempt to trigger a second capture while the shutter is still open resulting in an error.

## Lighting Control:

The LEDs used for the capture circuit are controlled by the Raspberry Pi's GPIO ports. In total it has been decided that 8 LEDs are to be mounted at $45^{o}$ around the camera lens. This number of LEDs has been chosen because they will provide enough data for scenarios where high accuracy depth maps need to be calculated, for less demanding situations the system can downscale to 4 LEDs to decrease processing time.

Initial designs for LED control were centred around a 3 to 8 decoder which would be powered by the Pi's 5V pin and controlled by 3 GPIO ports. This design was considered because it would allow the system to power 8 high power LEDs through a single IC package which would keep the footprint of the design low and reduce the amount of exposed cabling from the GPIO ports. The design was later changed to using 8 GPIO pins connected to the gates of separate NPN transistors. When the gates of the transistors are turned on the current will be allowed to flow through the LED from the 5V rail to the collector of the transistor and then from the emitter to ground. This method was favoured as it allowed the activation of multiple LEDs at once, something that is not possible with decoder, which can allow further photometric techniques to be explored.

Figures 12 and 13 show the schematic and full connections for the proof of concept design respectively; this design was extended to 8 LEDs in the final version however in order to simplify the first prototype's development this was downscaled to 4 LEDs to reduce the time of a full capture cycle. Development for this project started with a first-generation Raspberry Pi B+ which completed a 4 LED capture cycle (which included capturing and renaming the files) in 57 seconds. To improve this time, the program was streamlined to only rename the files once the capture cycle was complete which reduced the capture time to 35 seconds. Once development on the Raspberry Pi 4 began it was evident that the limiting factor was the capture speed of the camera when using the gphoto2 library. This was deduced because adding the renaming function to the capture cycle did not noticeably change the cycle time of 15 seconds. It was found that by switching the camera to manual focus the time taken for a full capture cycle was further reduced as it removed the need for the camera to autofocus each time a trigger command was given. After further consideration it was decided that the default camera setting should remain as manual focus as it guarantees that each pixel captured will consistently represent the same point from picture to picture which is vital for the photometric algorithm to produce accurate results.

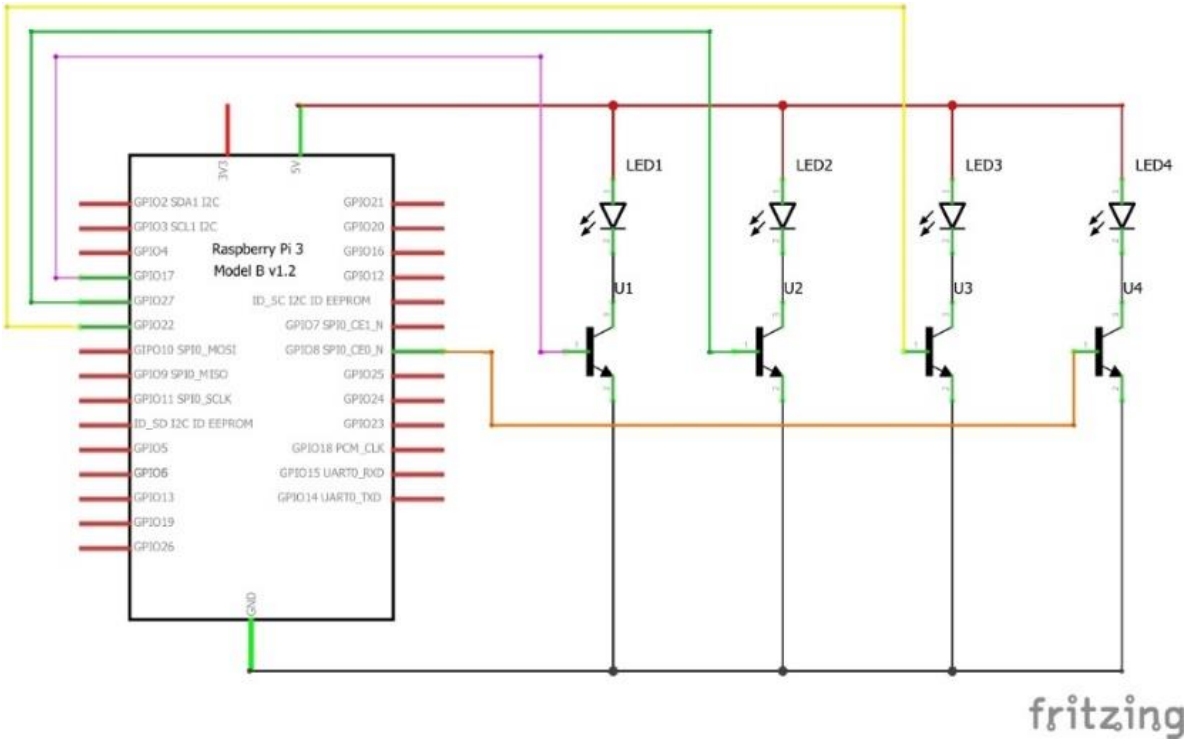


*Figure 12: Schematic for light control of four LEDs*

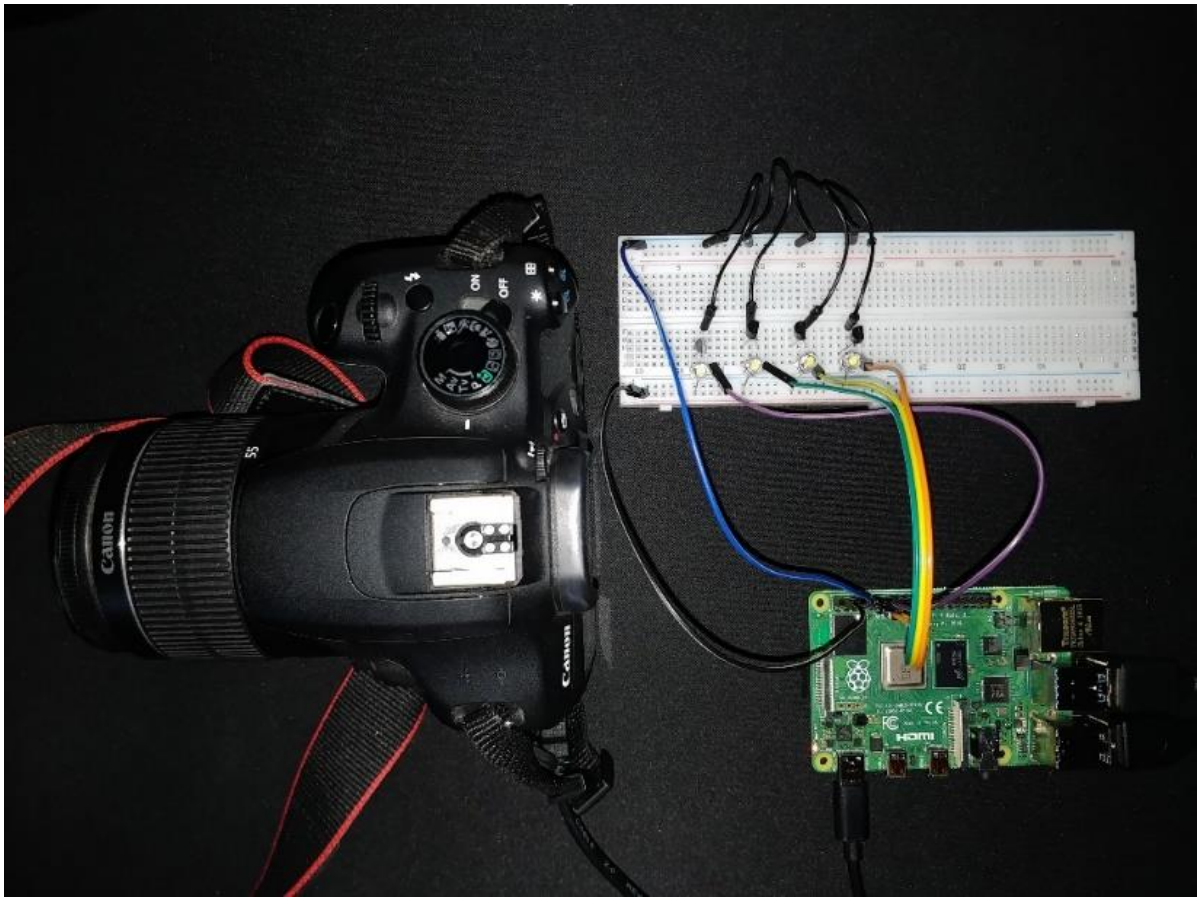


*Figure 13: Photograph of example connections for prototype*

### LED Power Circuit:

In order to make the system suitable for photometric capture it is important to have LED's which are as bright as possible to ensure that varying surface normals provide a large difference in pixel intensity which should reduce error during the processing phase. The Raspberry Pi used has the capacity for 5V and 3.3V output through their respective dedicated power pins. The GPIO pins used to control the LEDs are capped at 3.3V output which conventionally would be suitable for signal LEDs however as illumination intensity is a priority it is desirable to drive the LEDs at 5V. Driving LEDs well above their specified forward current for an extended period of time makes them susceptible to failure, however in the context of a camera flash which is only active for a fraction of a second the subsequent increase in illumination is desirable. Traditionally an LED power circuit would include a current limiting resistors to reduce the power which would be dissipated through the LED thus reducing the chance of it burning out, it was decided that this resistor should be removed from the design as it limits the brightness of LED. The resistor is also unlikely to reduce the burnout rate of the LEDs as for each capture cycle the LEDs are active for such a short amount of time that the potential build-up of heat was considered superfluous.

In preliminary tests it was found that if the LED was left on for an extended period in the current configuration then the LEDs would burn out and start smoking, causing a potential fire hazard due to the sustained high current. To reduce the time an LED is active for, the program sets GPIO outputs to low the moment an image is captured using the gphoto library, this however lead to a potentially dangerous bug in which in the event of a capture failure the LED would permanently be left in the high state causing a potential burn out. This bug was fixed with the addition of an except statement which, in the event of an error from the gphoto capture, will turn all GPIO ports used to low before exiting the program. Similarly, an unfixable bug occurs when booting and shutting down the Raspberry Pi in which some of the GPIO ports will start in the on state which would cause the LEDs to be on for over a minute from boot until the program is run. At the time of writing the only way around this bug is to connect the lens mount only once the capturing algorithm has been started which boots all the GPIO ports to low when first run, and to disconnect the lens mount before powering down the Raspberry Pi.

It was decided that the power circuit should be manufactured on a PCB, while this does not necessarily improve the function of the project it simplifies the mounting process and neatly wraps the project into an all in one solution.

The software used to design the PCB was EAGLE by Autodesk [34], a licence for which was provided by the university. This software provides the ability error check PCB designs as well as ensure that all

designs conform with rules specified by the manufacturing house which reduces the likelihood of the design being rejected by the manufacturer thus saving on manufacturing time which was vital as the cheapest place found to manufacture PCBs including shipping costs was by Elecrow, a company based in Shenzhen, China.

When designing the PCB, it was important to consider how it would be mounted to the system, in order to provide the sturdiest construction, holes where placed in the design to allow the board to be mounted above the Raspberry Pi using M2.5 standoff screws. The design only includes two screw holes on one side as in its current state the board is not large enough to mount to all four, and if the design width were to be increased in size then the board would block any access to the GPIO ports.

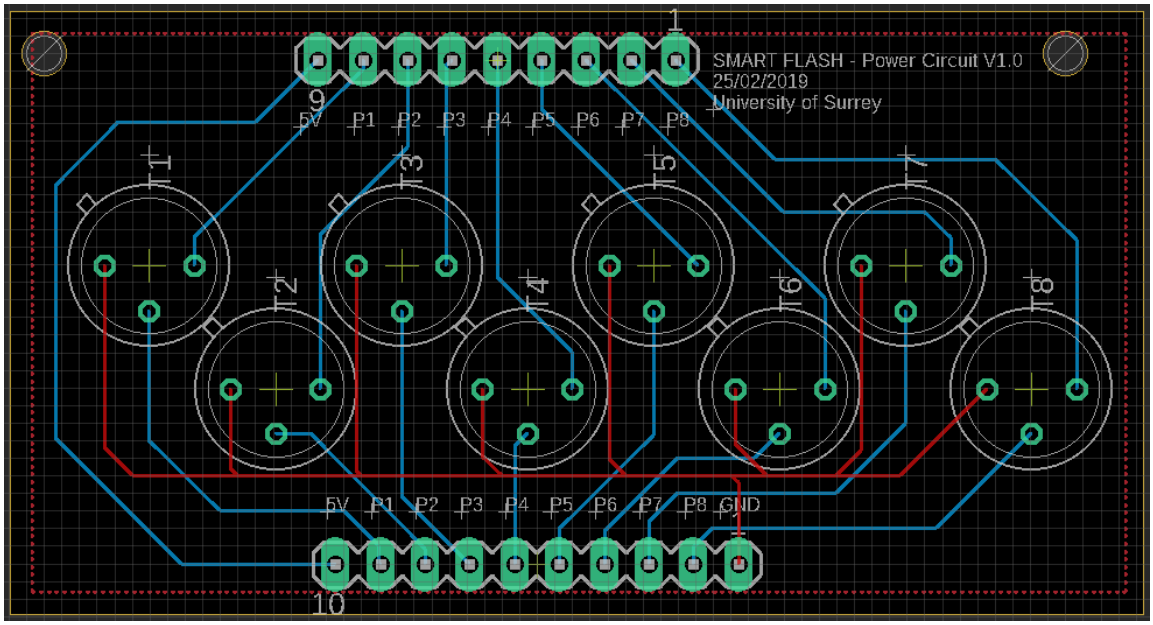


*Figure 14: Board design for a BJT power circuit*

Figure 14 shows the final board design used for this project. The board design consists of two layers to reduce the need for long traces, with the top layer's traces (red) used exclusively for a universal ground connection. The board features two pinout schemes, the 10-pin line (bottom) is used to connect the board to the Raspberry Pi's GPIO ports via a ribbon cable, these pins are responsible for 5V, GND and base control for the 8 NPN transistors. The 9-pin line (top) connects the board to the lens mount and is responsible for the 5V power line and the 8 connectors to the collector ports of the transistors. Figure 15 shows the final assembly of the board, the transistors used for this project are BC337 NPNs by ON Semiconductor [35], these were chosen for their through-hole designs while also providing the lowest likelihood of failure due to their high maximum specification for both forward current and collector bias when compared to the requirements of this project.

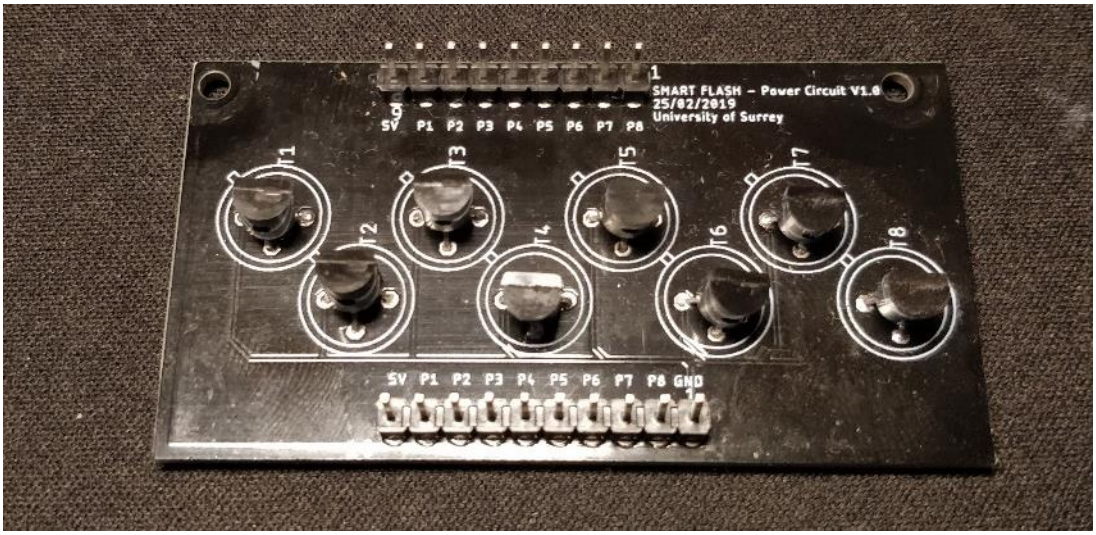


*Figure 15: Soldered version of BJT power circuit*

## Hardware Mounting:

As stated previously, the final solution should be an all-in-one mount which ideally would be mountable to any DSLR camera. The hot shoe is a mounting feature which is shared with almost all DSLR cameras, traditionally it can be used to add a more powerful flash than the one integrated with the camera however for the purposes of this project it can provide a stable mounting platform for a Raspberry Pi. It was decided that the easiest way for the entire system to be mounted on the DSLR was to break the mount into two parts. The first would mount to the hot shoe and house the Raspberry Pi, LED power circuit and battery pack. The second mount would screw into the to the camera lens' filter and house the 8 LEDs.

As not all lenses have the same diameter of filter mount it would be difficult to make an adaptable solution which would fit all. In photography step-up and step-down rings can be used to allow any size filter to be fitted to any diameter of camera lens.

All 3D printed parts for this project were designed in Tinkercad due to its shallow learning curve when compared to other 3D modeling software such as Solidworks while also being robust enough to produce designs complex enough for the purposes of this project. All 3D object files were then put into Creality Slicer to convert them to printable GCODE and add the necessary raft and printing supports, all files were then printed on an Ender 3D with a printing thickness of 0.15mm in white PLA.

## Hot Shoe Mount:

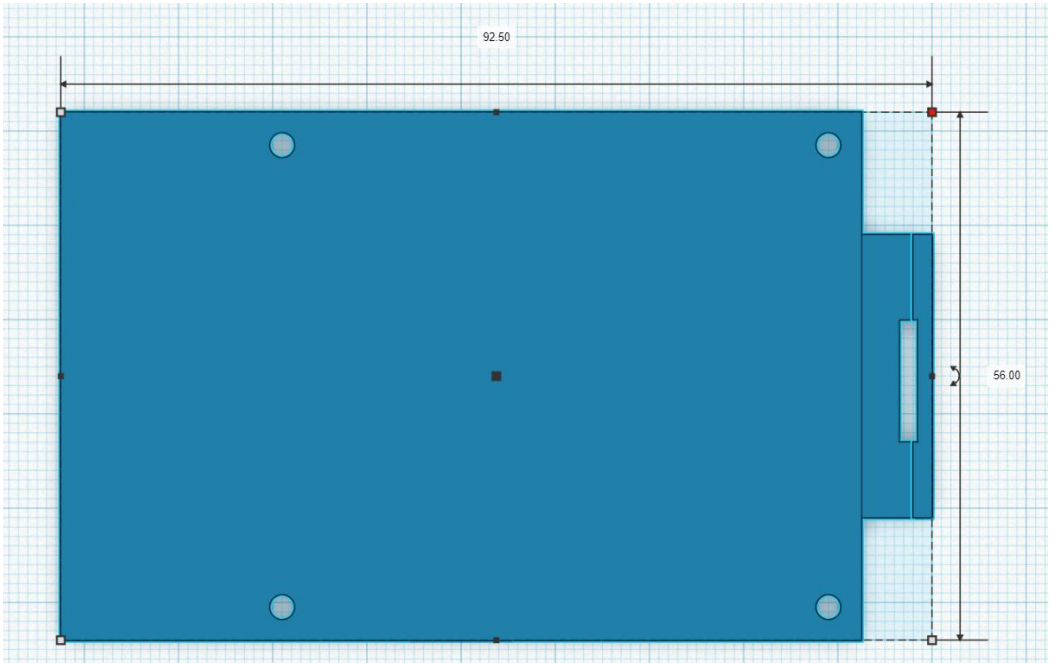


*Figure 16: Above view of hot shoe mount*

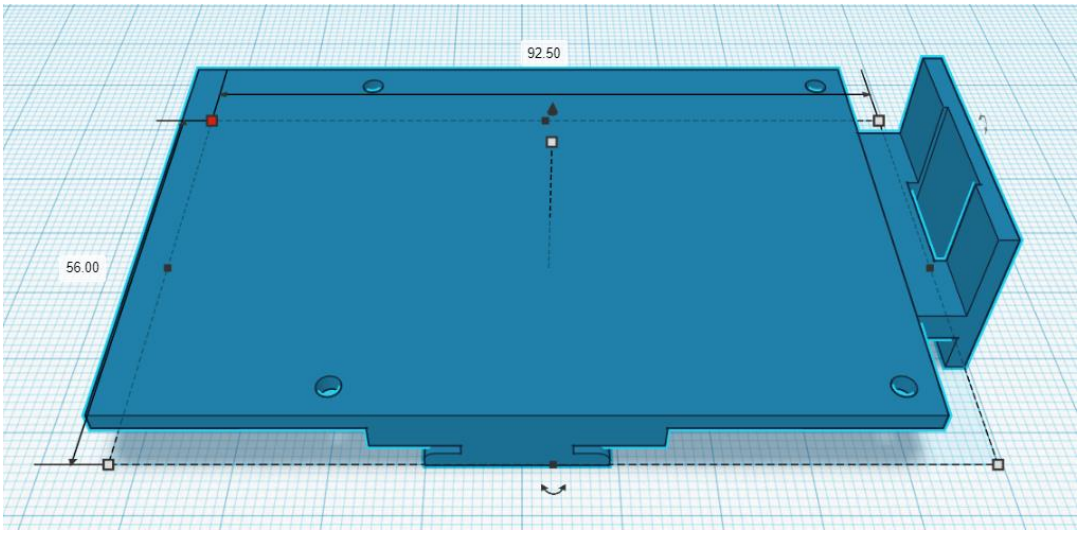


*Figure 17: Side View of Hot Shoe Mount*

Figures 16 and 17 show a render of the final hot shoe mount. The final design includes a tab to the right side of the Raspberry Pi which will allow the battery pack to be mounted using a Velcro tie, this was necessary as at the time of writing the more convenient and discrete PiJuice HAT [36] power solution was out of stock. In this design the mounting holes for the Raspberry Pi are offset slightly to ensure that none of the board is overhanging when mounted. The size of the male hot shoe mount did not need to be completely accurate to provide a solid mount as the female mount of the camera features a spring system to ensure that any mount does not come loose, this feature is ideal for

objects being printed on low-end 3D printers such as the one used in this project where tolerances cause a discrepancy between printed and designed values.

Figure 18 shows a final assembly of the hot shoe mount, this features the Velcro attached battery pack which will power the Raspberry Pi through a USB A to type-C power cable. This battery pack only produces 2A of the maximum rated 3A for the Raspberry Pi 4 however tests have shown that the battery is able to power the Pi while under the load of the program without shutting down due to a lack of power.

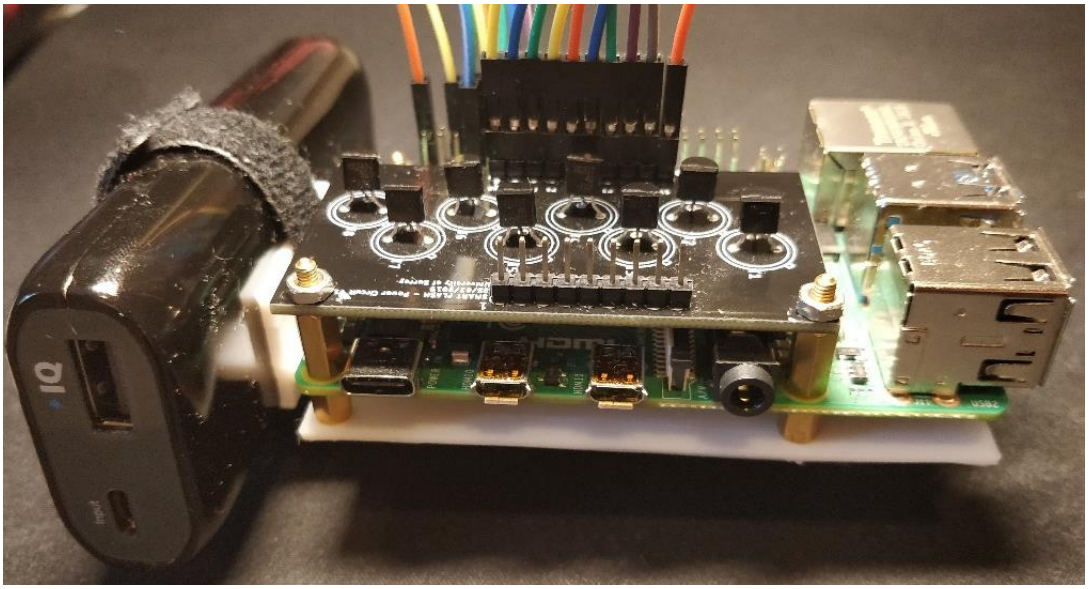

*Figure 18: Assembled hot shoe mount*

### Lens Mount:

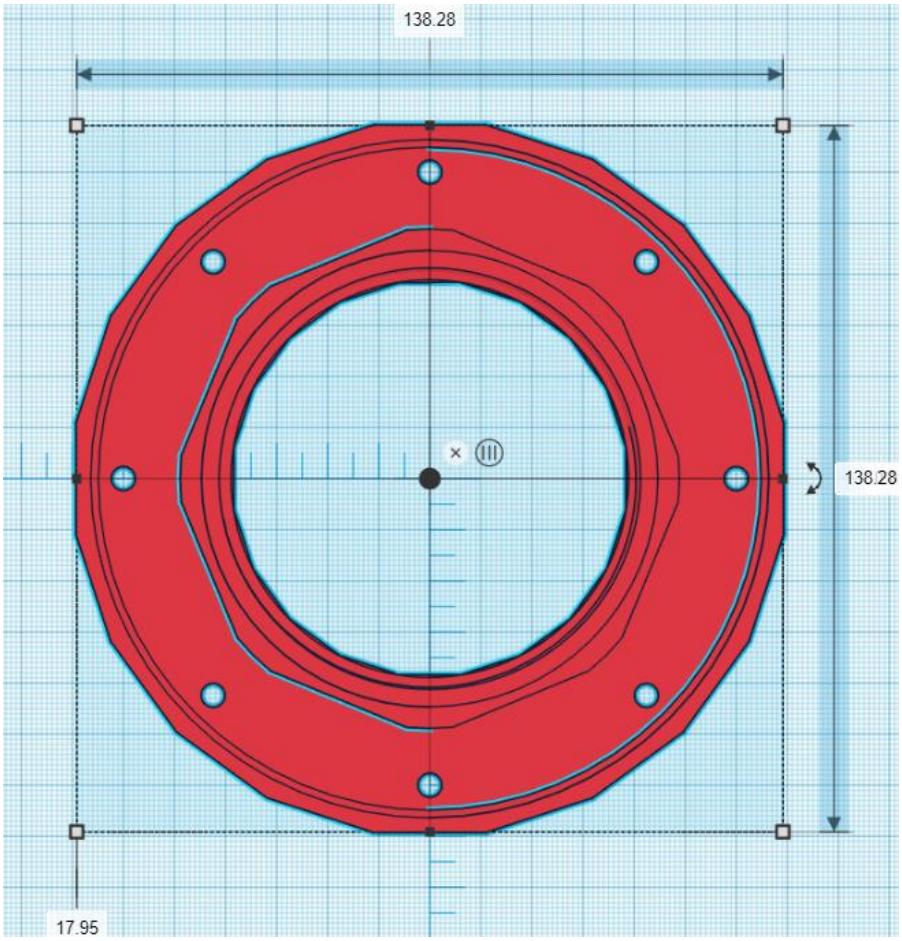


*Figure 19: Above view of lens mount*

Figure 19 Shows the final design of the lens mount used to collect data. This design features 8 holes for LEDs to be placed in, each at $45^{\circ}$ to each other. The design is hollowed out to allow channels for cables to be ran through. Figure 20 shows a second part of the system which is placed on the back of the lens mount to hide the cables and house the routing circuit which takes an input from the 9-pin ribbon cable.

At the time of printing the only available printing filament was white PLA, this be posed a potential problem in that when a light is shone through white PLA it appears translucent. Due to the geometry of the design it was unknown whether an active LED would be in any way visible to the camera, once printed however this flaw did not prove to have any effect on the captured data. The translucent spots did result in a shift in the pixel locations of specular spots when trying to calibrate the light sources however this effect can be reduced by increasing the average bright area which the calibration program is looking for. If the project were to be redone the mounts should be printed in matte black PLA.

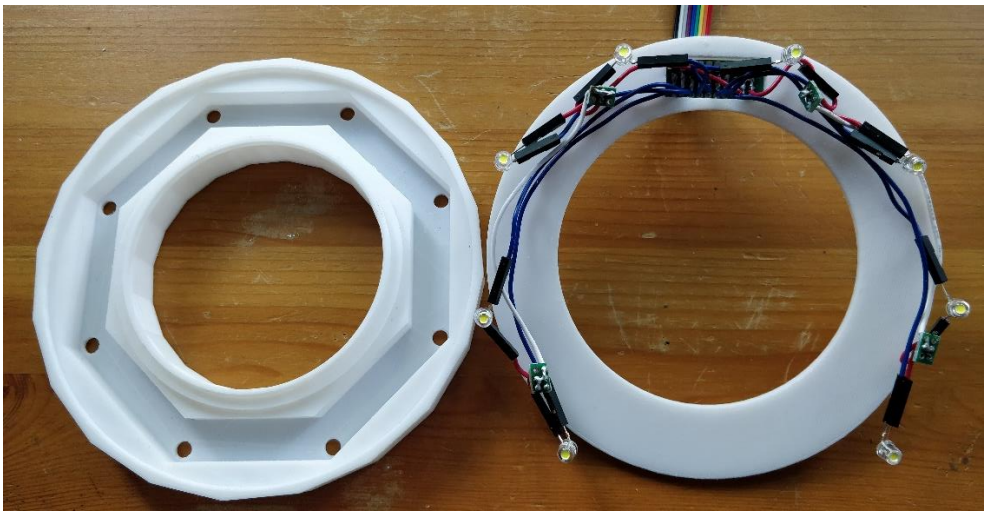

*Figure 20: Assembly of lens mount*

Due to the limited remaining time at assembly it was not possible to manufacture a PCB for the routing circuit so in order to get the best function possible the circuit had to be soldered onto a prototyping board. To simplify the repair process, for example if one of the LEDs were to burn out, the LEDs were connected using Dupont connectors instead of soldered onto the wires. These connectors provided little change to the resistance of the circuit while also providing a strong enough connection to the LED contacts that they are unlikely to come loose through regular use. Figure 21 shows the final assembly of the separate project mounts on to the DSLR camera.

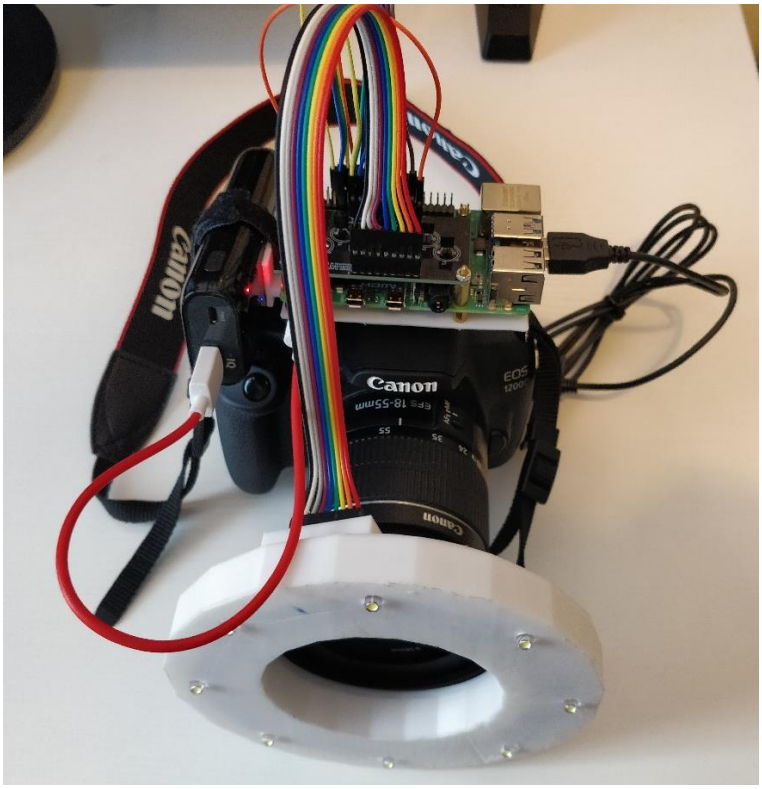


*Figure 21: Final assembly of hardware*

## Photometric Stereo:

As stated previously, a photometric stereo algorithm with a public use license will be used to process the images taken by the DSLR's capture cycle. The program we will be using is "RobustPhotometricStereo" by Yasuyuki Matsushita [29], this program provides multiple methods for solving for surface normals such as the standard least-squares method or more advanced solutions such as Robust Principal Component Analysis [37].

In order to provide an output normal map, the program requires three inputs, the first is the file directory which contains the dataset of photographs taken under differential lighting conditions. The second input is an image file containing a white mask of the area in the images which the algorithm will solve for. The final file is a text file containing the unit vectors of all lighting directions used to capture the dataset, this is acquired using a lighting calibration stage. A final optional file can be added to the input containing the ground truth of the dataset, this file is used to calculate the error in calculated normals and is mainly used for rendered dataset and so for the purposes of this project will remain unused, however could potentially be applied to 3D printed objects using the 3D object file to obtain the ground truth.

## Calibration:

In order to simplify the processing required for photometric stereo it was decided that a calibration system should be incorporated. Traditionally calibration is achieved through identifying bright spots on a chrome sphere as discussed in the related works section; however, as the purpose of this project is to apply photometric stereo to non-specialist equipment it was decided that the calibration of the light sources should be achieved through the use of a planar mirror. For this technique we will refer to [4] which describes how light sources can be calibrated using a planar mirror in either one or two orientations, to keep things simple we will only be using one mirror orientation. This paper specifies that in order to calibrate light sources we must first acquire the intrinsic and extrinsic properties of the camera, in this case they are the camera matrix "**A**", the translation vectors "**t**" and, the rotation vectors "**R**". To get these values the paper implies a calibration scheme using a chessboard, however, to improve calibration accuracy we will be using an improved scheme utilizing a ChArUco board. A ChArUco board combines a chessboard design with identifiable ArUco markers to improve on accuracy and reduce the probability of known bugs with chessboard calibration occurring such as Z axis flipping. It is important that during calibration the ChArUco board be on the same plane as the mirror, as by calibrating for the board you also get the normal of the mirror allowing us to use the reflection equation (12) mentioned in the Photometric Calibration section of the literature review.

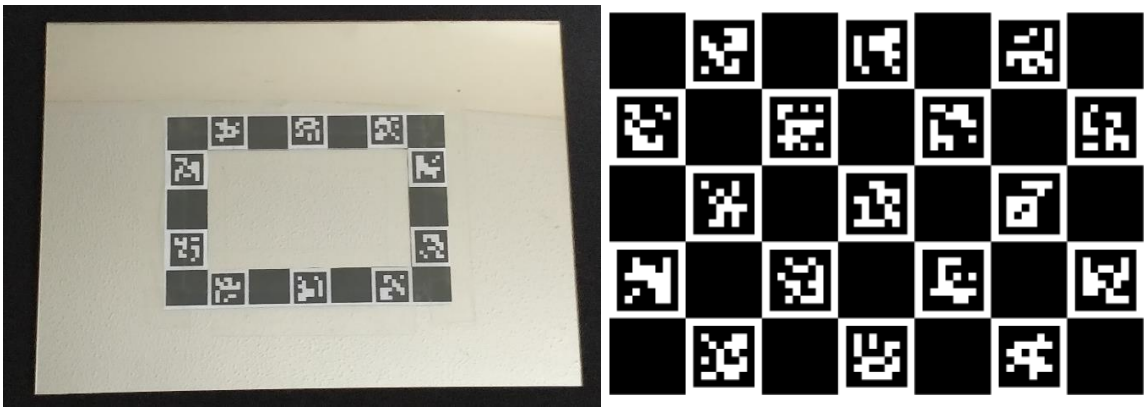

*Figure 22: Early prototype of calibration board vs full board*

Figure 22 shows an early design of the calibration board used, it features only the outline boarder markers of the complete board which for ChArUco should be sufficient for accurate calibration, these centre markers were removed as it would allow for more accurate calibration for specular spots inside the border than if a full board were exclusively in the corner of the mirror. Early calibration results with this board provided distortion coefficients which resulted in corrected images being distorted beyond recognition as shown in Figure 23, it is unknown what caused this problem however the problem was subverted by firstly pinning the board down to be perfectly flat against the mirror and secondly only taking calibration images which were not at a large angle to the normal of the mirror. The second fix is used to correct a problem where the second surface mirror allows the camera to see behind the paper ChArUco board which causes errors in corner point detection.

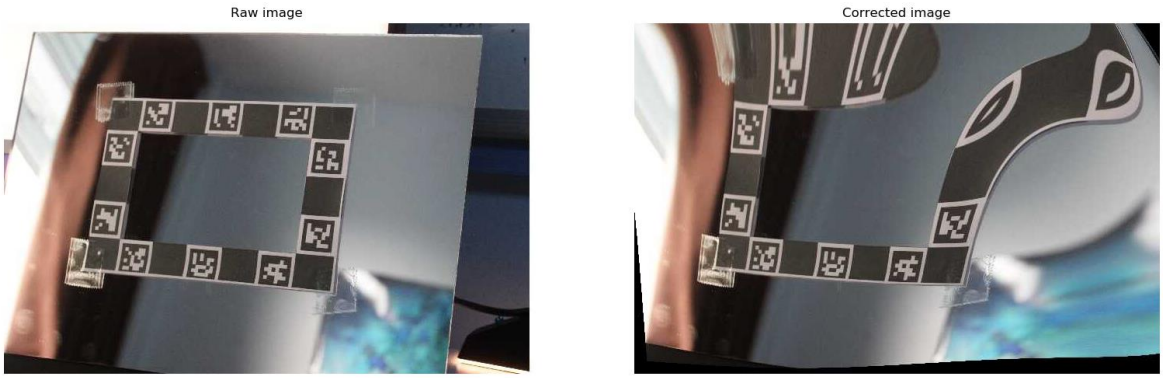


*Figure 23: Errors which occurred using early calibration design*

To calibrate the camera two functions have been written: read_chessboards() and calibrate_camera(). The function read_chessboards() takes in a folder containing all of the calibration images, it then returns the size of the image in pixels, and the pixel locations of the chess board corners and ArUco IDs. These variables are then fed into the second function calibrate_camera() which returns all the properties of the camera including the camera matrix, rotation vectors, translation vectors and, the distortion coefficients. These values can then be

applied into the technique mentioned in [4]. In order to simply the overall calibration stage, the images fed into the camera calibration functions to work out extrinsic properties of the camera are the same ones analysed to locate the position of the specular bright spots.

```
def pixelList(images):
    maxList = []
    for im in images:
        x = cv2.imread(im)
        gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (131,131), 0)
        (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(gray)
        maxList.append(maxLoc)

    return maxList
```

*Figure 24: Function to find the brightest spot in an image*

The next step in the calibration is to locate the position of the different LEDs, to do this we assume that when an LED is on it will be the brightest spot in the image. Figure 24 shows the function used in the program to find the brightest spot in an image, this function takes in an array (images) which contains the directory of each of the eight calibration images and returns an array containing the corresponding pixel coordinates of the brightest spot in the image. To improve the likelihood of a correct pixel location the image undergoes two prerequisite processes, firstly the image is converted to greyscale, so the pixel intensity is only a function of a single colour. Secondly, the image undergoes a gaussian blur to reduce the effect of noise on the final coordinate. The resulting processed image can then be analysed to find the locations and values of the points of highest and lowest intensity. An example of this process working is shown in Figure 25 which locates the point of highest intensity using the function cv2.minMaxLoc(). As the gphoto library only initiates a capture based on the current settings we can reduce the size of the specular spot on the mirror by setting a low camera exposure rate during the calibration capture cycle which should provide a more accurate representation of the LED's location, this exposure should then be increased when capturing the images of an object to be processed.

*Figure 25: Circled location of brightest spot in image using pixelList()*

### *Calibration Algorithm:*

The final part of the calibration scheme involves the vector calculations described in [4]. The first step taken is to find the camera matrix **A** which represents the camera's intrinsic properties, to do this we must first take photographs of the calibration board from various angles and feed these into the two functions read_chessboards() and calibrate_camera() as mentioned before. The next step in calibration is to find the camera's extrinsic properties, the rotation matrix **R** and the translation vector **t,** as these properties are extrinsic, they are likely to change with varying camera positions and so it is important to calculate these for every new calibration dataset. The function calibrate_camera() does not give the rotation matrix only the rotation vector, fortunately OpenCV has the function cv2.Rodrigues() [38] which can convert a 3x1 rotation vector to a 3x3 rotation matrix or vice versa. Using the intrinsic and extrinsic properties of the camera we can form a 3x3 homography matrix **H** [39] as shown in equation 17.

$$H = A[r_1 \quad r_2 \quad t] \tag{17}$$

Where $r_1$ and $r_2$ are the first two columns of the rotation matrix **R**.

The homography matrix helps describe how a 2D image point can be mapped into world coordinates which will allow us to find the 3D vectors of the lighting directions. Conventionally equation (17) would include the third column of the rotation matrix however as we are projecting onto a mirror, we can assume that all the Z values are the same. Now that we have the homography matrix 3D world coordinates can be computed by the equation:

$$H^{-1}\tilde{m} = \tilde{M} \tag{18}$$

Where $\tilde{m}$ and $\tilde{M}$ represent the homogeneous coordinates of the 2D image and 3D world coordinates respectively. To make the image coordinates to homogeneous we can use cv2.convertPointsToHomogeneous() which takes in a 2xN array and appends an extra value to each

coordinate available. The world coordinates of the specular spots $P_w$ can then be computed into the camera coordinate system using the equation:

$$P = RP_w + t \tag{19}$$

In the camera coordinate system, as **C** is the origin, the vector $\overrightarrow{CP}$ and the point P are equal.
The surface normal can be computed simply in the camera coordinate system from equation:

$$n = r_1 \times r_2 \tag{20}$$

Now that the vector $\overrightarrow{CP}$ and the mirror normal $n$ have been calculated the vectors of the incident light can be calculated from equation 12. These equations are easily implemented in python using the NumPy [40] library which includes functions such as np.transpose() and np.dot() to handle all matrix calculations

### *Improved Calibration Board:*

In order to improve the results obtained from calibration a second ChArUco board (Figure 26) was produced to address the faults of the original prototype and improve on usability. Due to the size of the old ChArUco board the area of mirror which could show the light spots was small making it difficult to line up the board so that all spots can be shown in a single capture cycle. The second improvement was to add a white boarder on either side of the ChArUco markers so that corners of the chess board can be more accurately identified.

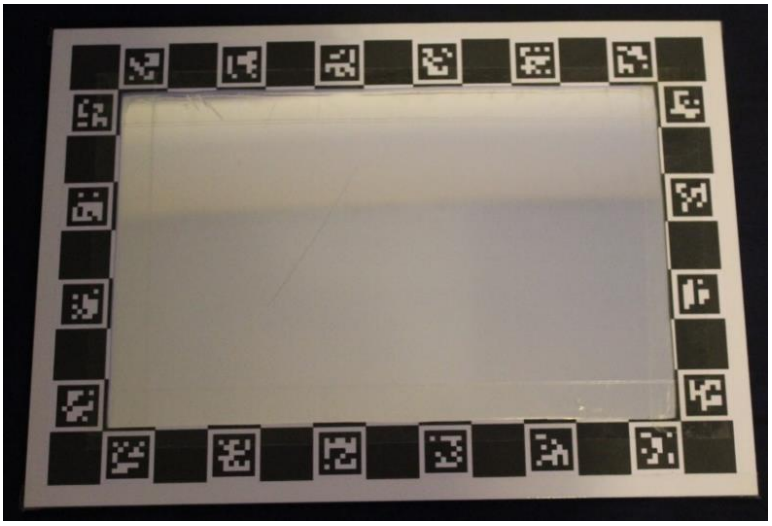

*Figure 26: Second iteration of calibration board*

We can quickly verify if the calibration scheme is working by taking two calibration cycles, the first will show the lights to the left side of the optical centre and the second will show them to the right. The results for this test are shown in Figure 27. As shown the X values for the vectors (first column) switch polarity as the lights swap sides around the optical centre.

```
CAMERA CALIBRATION                    CAMERA CALIBRATION
[[0.16334728 0.19122441 0.96785892] [[-0.1187741   0.20032286  0.97250371]
 [0.09372192 0.27188451 0.95775519]  [-0.15095644  0.16988446  0.97383336]
 [0.12955019 0.16152118 0.97832901]  [-0.13811711  0.27176555  0.95240073]
 [0.08462878 0.16922255 0.98193773]  [-0.2068775   0.25086134  0.94565865]
 [0.13715547 0.26564603 0.95426441]  [-0.17675221  0.27590527  0.9447936 ]
 [0.05721516 0.20307146 0.97749087]  [-0.11483475  0.24303814  0.96319543]
 [0.16436503 0.23127296 0.95890404]  [-0.21585581  0.20839493  0.95392758]
 [0.06207209 0.24366791 0.96787035]] [-0.19043438  0.17369332  0.96621187]]
```

*Figure 27: Example lighting directions from calibration*

To reduce costs the mirror of the calibration board is made from acrylic Perspex and is categorized as a second surface mirror. If accuracy is a priority, then the mirror should be replaced with a first surface mirror however this will come at a penalty to cost, as this system aims to be a budget solution to photometric capture the current second surface mirror was deemed sufficient.

## Working Pipeline:

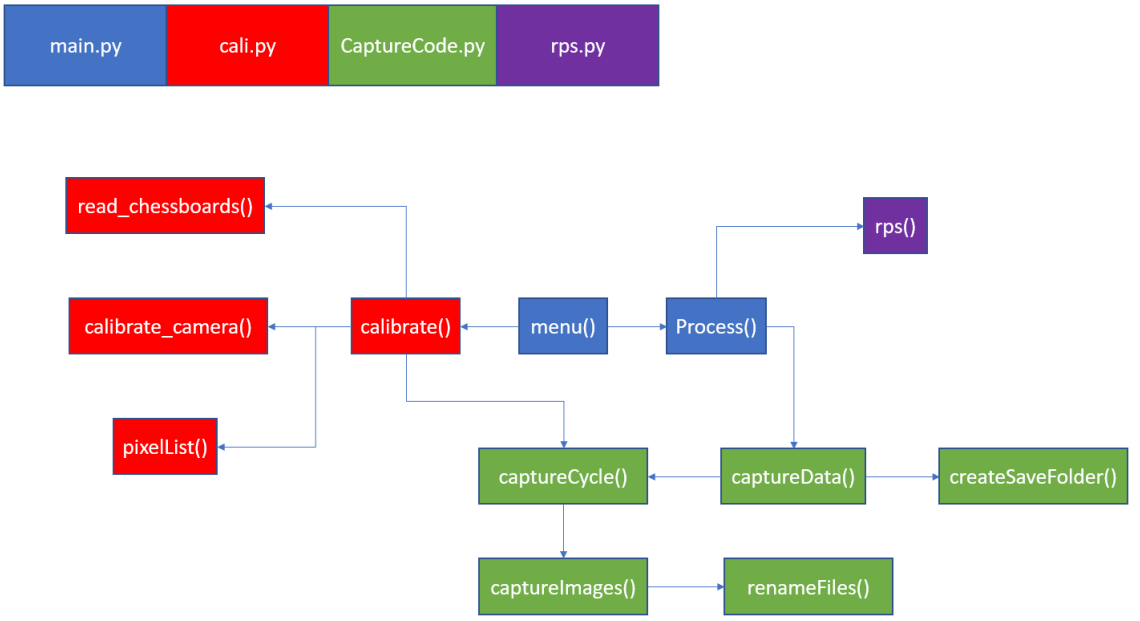


*Figure 28: Function hierarchy diagram*

Figure 28 shows the function hierarchy diagram which operates the current version of the prototype, the current way to access this is by connecting the Raspberry Pi to WiFi and using VNC Viewer on a separate machine to access the Pi's command line which displays the output of the function menu(). When at the menu() function there are two paths, the first function calibrate() takes you through the setup to calibrate the lighting directions of the camera through the use of the ChArUco/mirror board. The second path goes through the capture and processing of photometric images. Once completed an output folder is created which is named after the date and time which the process was started, this folder contains the 8 captured images in the form "POSITION.JPG", an

npy file containing the a matrix of the estimated normal map and a png image of the generated normal map.

### Testing:

To setup the system for capture we must first calibrate the lighting direction. Firstly, we setup the camera system on a secure tripod to ensure that the same feature locations of an object remain consistent from image to image. We then setup the improved ChArUco/mirror calibration board at roughly the distance we will expect to capture the object, ensuring that all ChArUco markers are visible in the image through the viewfinder as well as all LEDs being visible in the mirror of the board. To reduce the background interference in results we zoom in on the board so that it covers the image as much as possible, this benefits the processing later as we can then replace the calibration board with a blank background which will cover most of the image.

The first set of tests carried out by the processing portion of the pipeline used a white blank image to represent the image mask. It is noted that by using a plain white image as the mask, results will include the image background in processing thus the accuracy of the results will suffer. However, the purpose of these firsts tests are to see if the photometric algorithm in conjunction with our lighting system is capable of simply detecting the geometry of a shape rather than the accuracy of these results, later tests use a mask with a black border in an attempt to reduce the amount of background present while not covering any of the image subject. To keep comparisons consistent all tests are conducted under dark indoor lighting conditions with the photometric algorithm set to use the least squares method of solving.

The first object used to test how the system handles basic geometry is a plain piece of paper folded in a V shape, which is tested under the configuration in Figure 29. For these early tests we will be using objects with uniform colour to be considered as a control variable. As shown by the results in Figure 30 the algorithm is able to detect the centre line of the fold in the paper as well as the orientation of the folded sides as shown by the colour gradient, with green tones indicating that the paper is oriented to the left and the purple tones showing its pointed to the right.

After the data was captured and analysed it was noticed that two of the lights produced a slightly different colour to the others, potentially affecting results. To rectify this all the LEDs were swapped out to new ones from a separate packet of 10.

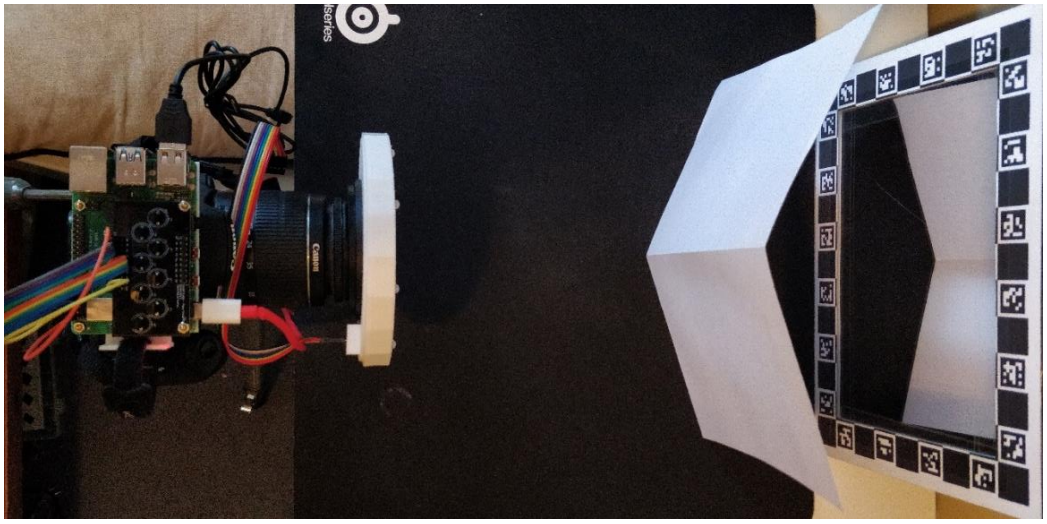

*Figure 29: Geometry test configuration*

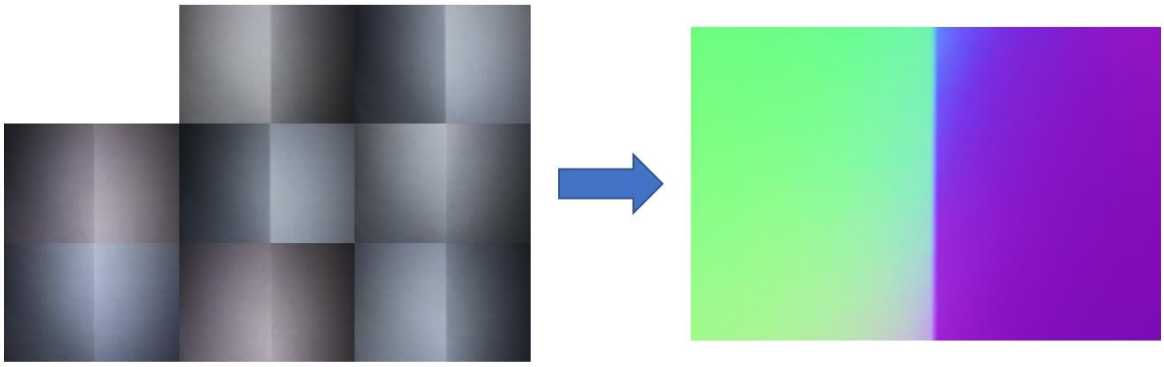

*Figure 30: Results of early geometry test*

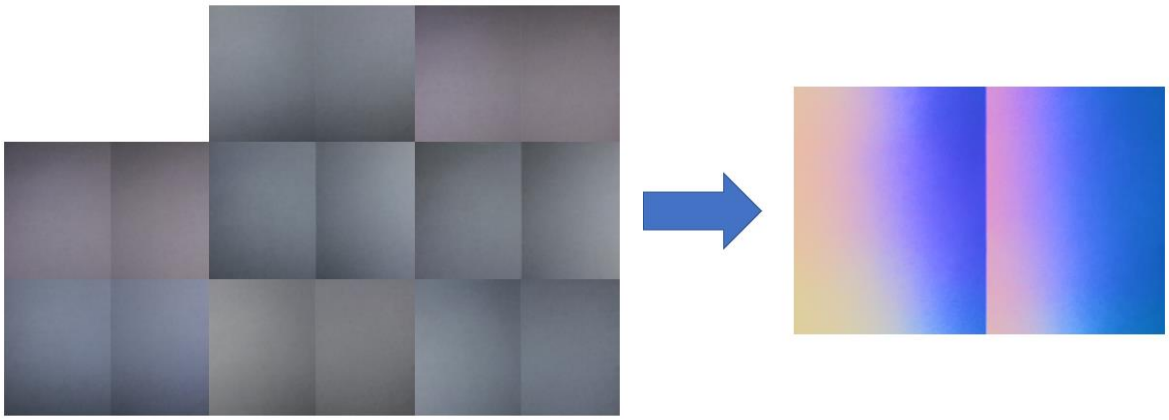

*Figure 31: Test with inverted paper arrow*

The flaws in this system are exposed when the paper is reversed and the arrow the paper forms points towards the right-hand side of Figure 29. In this configuration it is more difficult for the algorithm to discern the shadow caused by the geometry as light incident on one side of the paper will reflect and brighten the opposite side of the sheet resulting in very similar intensities. The algorithm does manage to discern that there is a fold between the two planes however it interprets the image as the two planes pointing in the same direction as shown by Figure 31. The next test to be conducted observes how the system handles objects which do not have uniform colouring, this could potentially cause errors as the algorithm will struggle to distinguish whether the difference of intensity observed is due to the lighting conditions or the colour at that point. This test will be on a can of deodorant which has simple cylindrical geometry but a colourful design pattern.

*Figure 32: Normal map generation for object with colour*

The results in Figure 32 show that the system still manages to identify the overall shape of an object however wherever there is colour on the bottle design the algorithm identifies it as a change in shape, this is emphasised on the logo of the bottle where the normal map is split in two and shows separate colours to its surroundings. The results are not improved by the coating of product which becomes partially specular under high light intensity as shown in the reference images of Figure 32. The results in Figure 32 also highlight the importance of generating an accurate mask which follows the outline of the object to be photographed, as shadows from the object are projected onto white background they are interpreted as part of the objects geometry.

In the related works section, it was discussed how photometric stereo can be applied to scan fingerprints while the developed solution can't achieve that level of accuracy the next test is designed to show how the solution can detect details. The object used is a silicon case with a matt uniform colour and an engraved logo where the lines have a width and depth of <1mm. Figure 33 shows the results of this test, as the shape of the object is convex and the coating doesn't exhibit any specular behaviors the normal map of the object can be easily interpreted from the produced

image. The image farthest to the right of Figure 32 shows the normal map representation of the logo engraved on the case. As shown, the engraved section appears in a darker shade than its surroundings showing that the system can detect small details in an object's geometry.

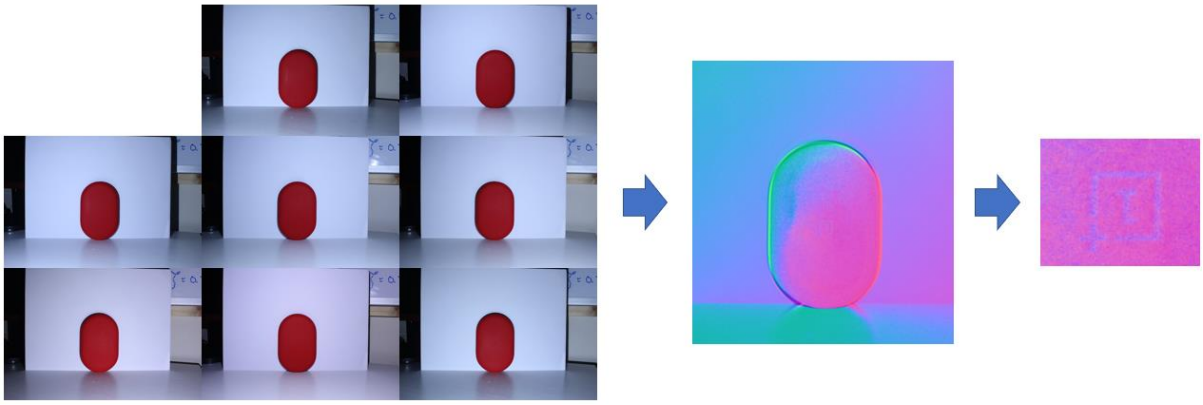

*Figure 33: Test for detail in object*

## Summary:

This section describes the steps taken to produce an all-in-one photometric capture and processing solution making developments in both hardware and software. The hardware parts of this section describe the design process around creating a lighting mount which can be adaptable to any DSLR camera through the hot shoe and filter mounts, as well as the design of an electronic power circuit used to power the LEDs. The software parts of this section describe firstly, how a Raspberry Pi can be used to control the designed lighting system, and secondly the methods used to calibrate the system by using a mirror and ChArUco board combination.

Finally, various tests were designed to evaluate the situations in which the solution developed would be most adequate. These results show that the proposed solution is most appropriate for capturing convex shapes of uniform colour, however the results show that the solution produces adequate results for a previsualization of more complex objects which are either specular or feature non uniform coloring.

## Final Conclusions and Future Work:

This report introduced an all-in-one solution for the capturing and processing of images using photometric stereo. In order to produce this solution, developments had to be made in both hardware and software. The hardware portion includes a lens mount which holds 8 LEDs to provide differential lighting conditions, and a hot shoe mount which holds a Raspberry Pi, LED power circuit and battery pack. The software side of the project includes the control of the LEDs and DSLR to synchronise the LEDs with image capture and utilizing a ChArUco/mirror board combination to calibrate the lighting direction of the LEDs. These two sections when combined can create a normal map of an object based off captured image data. As this project uses common commercial items it provides a cheap alternative to conventional photometric systems and achieves its job of providing a basic previsualisation of a normal map which can quickly help to determine whether a scene is suitable for photometric computation.

The timeframe for this project was a single academic year which resulted in certain parts of the project receiving a higher priority than others. It was decided that the aspects of the project which related to the capturing of photometric images should be put before the onboard processing as it would still allow us to test whether our solution was adequate for photometric capture by replacing the onboard processing with exporting images to a 3$^{rd}$ party desktop software. Fortunately, these contingencies were not required and both pillars of the project were completed before the deadline. The time delays experienced in this project were minimal, these were extended PCB wait times due to importing from China and a lack of part availability, however these did not affect the overall function of the solution due to generous time allocation for development in the planning stage as well as suitable alternatives being sourced such as using a USB mobile charger instead of a dedicated Raspberry Pi battery pack.

Had more time been given for development the project should aim to integrate a mask generation scheme which would create a new mask for each new data set, this could improve the results of the normal map and could potentially be achieved through simply using a green screen background. The current system is unable to display the normal map on the screen of the DSLR due to a bug in the current version of the gphoto library, future iterations should aim to fix this however it still produces a downloadable normal map file for testing data. Finally, to improve usability a hardware control system could be implemented so that the Raspberry Pi's functions can be executed without the need for a secondary computer.

To conclude, this project has produced an all-in-one capture system which can calibrate its own light sources and producing a normal map based on captured input data and has subsequently met most of the objectives formulated at the start of the project.

# References


[1] B. K. P. Horn, "Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View," MIT, Cambridge, 1970.

[2] R. J. Woodham, "Photometric method for determining surface orientation from multiple images," *Optical Engineering,* vol. 19, no. 1, pp. 139-144, 1980.

[3] C. Liu, S. G. Narasimhan and A. W. Dubrawski, "Near-Light Photometric Stereo using Circularly Placed Point Light Sources," IEEE, Pittsburgh, 2018.

[4] H.-L. Shen and Y. Cheng, "Calibrating light sources by using a planar mirror," *Journal of Electronic Imaging,* vol. 20, no. 1, 2011.

[5] C. S. Verma and M.-J. Wu, "Photometric Stereo," University of Wisconsin-Madison, [Online]. Available: http://pages.cs.wisc.edu/~csverma/CS766_09/Stereo/stereo.html. [Accessed 29 12 2019].

[6] Z. Mo, B. Shi, F. Lu, S.-K. Yeung and Y. Matsushita, "Uncalibrated Photometric Stereo Under Natural Illumination," in *IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, 2018.

[7] F. Lu, Y. Matsushita, I. Sato, T. Okabe and Y. Sato, "From Intensity Profile to Surface Normal: Photometric Stereo for Unknown Light Sources and Isotropic Reflectances," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 37, no. 10, pp. 1999-2012, 2015.

[8] C. Tsiotsios, A. J. Davison and T.-K. Kim, "Near-lighting Photometric Stereo for unknown scene distance," *Image and Vision Computing,* vol. 57, pp. 44-57, 2014.

[9] W. Xie, C. Dai and C. C. L. Wang, "Photometric Stereo with Near Point Lighting: A Solution by Mesh Deformation," IEEE, Boston, 2015.

[10] W. Xie, Y. Nie, Z. Song and C. C. L. Wang, "Mesh-Based Computation for Solving Photometric Stereo With Near Point Lighting," *IEEE Computer Graphics and Applications,* vol. 38, no. 3, pp. 73-85, 2019.

[11] T. Higo, Y. Matsushita, N. Joshi and K. Ikeuchi, "A Hand-held Photometric Stereo Camera for 3-D Modeling," in *12th International Conference on Computer Vision*, Kyoto, 2009.

[12] T. Papadhimitri and P. Favaro, "Uncalibrated Near-Light Photometric Stereo," BMVA Press, Bern, 2014.

[13] F. Duan, "Consistent depth maps estimation from binocular stereo video sequence," *Journal of Shanghai Jiaotong University,* vol. 21, no. 2, p. 184–191, 2016.

[14] C.-K. Yeh, F. Li, G. Pastorelli, M. Walton, A. K. Katsaggelos and O. Cossairt, "Shape-from-Shifting: Uncalibrated Photometric Stereo with a Mobile Device," in *017 IEEE 13th International Conference on e-Science*, Auckland, 2017.

[15] L. Bi, Z. Song and L. Xie, "A novel LCD based photometric stereo method," in *4th IEEE International Conference on Information Science and Technology*, Shenzhen, 2014.

[16] J. J.Clark, "Photometric stereo using LCD displays," *Image and Vision Computing,* vol. 28, no. 4, pp. 704-714, 2010.

[17] F. Hernandez-Rodriguez and M. Castelan, "A Mobile Data Acquisition Platform for Photometric Stereo," in *Electronics Robotics and Automotive Mechanics Conference*, Cuernavaca, 2009.

[18] O. Drbohlav and M. J. Chantler, "On optimal light configurations in photometric stereo," in *Tenth IEEE International Conference on Computer Vision*, Beijing, 2005.

[19] W. Xie, Z. Song and R. C. Chung, "Real-time three-dimensional fingerprint acquisition via a new photometric stereo means," *Optical Engineering,* vol. 52, no. 10, p. 103103, 2013.

[20] P. Hanrahan and W. Krueger, "Reflection from layered surfaces due to subsurface scattering," in *SIGGRAPH*, Anaheim, 1993.

[21] F. Alonso-Fernandez, J. Fierrez, J. Ortega-Garcia, J. Gonzalez-Rodriguez, H. Fronthaler, K. Kollreider and J. Bigun, "A Comparative Study of Fingerprint Image-Quality Estimation Methods," *IEEE Transactions on Information Forensics and Security,* vol. 2, no. 4, pp. 734-743, 2007.

[22] M. F. Hansen, G. A. Atkinson, L. N. Smith and M. L. Smith, "3D face reconstructions from photometric stereo using near infrared and visible light," *Computer Vision and Image Understanding,* vol. 114, no. 8, pp. 942-951, 2010.

[23] A. F. Abate, M. Nappi, D. Riccio and G. Sabatino, "2D and 3D Face Recognition: A Survey," *Pattern Recognition Letters,* vol. 28, no. 10, pp. 1885-1906, 2007.

[24] M. Oren and S. k. Nayar, "Generalization of the Lambertian model and implications for machine vision," *International Journal of Computer Vision,* vol. 14, no. 3, pp. 227-251, 1995.

[25] M. E. Angelopoulou and M. Petrou, "Uncalibrated flatfielding and illumination vector estimationfor photometric stereo face reconstruction," *Machine Vision and Applications,* vol. 25, no. 5, pp. 1317-1332, 2014.

[26] J. Liao, B. Buchholz, J.-M. Thiery, P. Bauszat and E. Eisemann, "Indoor Scene Reconstruction Using Near-Light Photometric Stereo," *IEEE Transactions on Image Processing,* vol. 26, no. 3, pp. 1089-1101, 2017 .

[27] Y. Lei and K. C. Wong, "Ellipse detection based on symmetry," *Pattern Recognition Letters,* vol. 20, no. 1, pp. 41-47, 1999.

[28] Tsuji and Matsumoto, "Detection of Ellipses by a Modified Hough Transformation," *IEEE Transactions on Computers,* Vols. C-27, no. 8, pp. 777-781, 1978.

[29] Y. Matsushita, "RobustPhotometricStereo," Osaka University, 22 February 2019. [Online]. Available: https://github.com/yasumat/RobustPhotometricStereo. [Accessed 30 March 2019].

[30] W. Jackson, "PhotometricCameraControl," 08 March 2020. [Online]. Available: https://github.com/WillJackson99/PhotometricCameraControl. [Accessed 18 May 2020].

[31] "Raspberry Pi 4 Model B Specifications," Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. [Accessed 3 1 2020].

[32] "3.7 V White LED 5mm Through Hole, Nichia NSDW570GS-K1-B-P9-P11," RS Components, [Online]. Available: https://uk.rs-online.com/web/p/leds/8294275/?sra=pstk. [Accessed 2 1 2020].

[33] H. Figuière and H. U. Niedermann, "gPhoto," [Online]. Available: http://www.gphoto.org/. [Accessed 09 12 2019].

[34] "EAGLE," Autodesk, [Online]. Available: https://www.autodesk.co.uk/products/eagle/overview. [Accessed 3 March 2020].

[35] "ON Semi BC33740TA NPN Transistor, 800 mA, 45 V, 3-Pin TO-92," RS Components, [Online]. Available: https://uk.rs-online.com/web/p/bjt-bipolar-transistors/6711119/. [Accessed 7 February 2020].

[36] "PiJuice HAT - A Portable Power Platform For Every Raspberry Pi," Pi Supply, [Online]. Available: https://uk.pi-supply.com/products/pijuice-standard. [Accessed 18 May 2020].

[37] Z. Lin, M. Chen and Y. Ma, "The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices," Cornell University Library, arXiv.org, 2013.

[38] "Camera Calibration and 3D Reconstruction," Opencv dev team, 31 December 2019. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[39] "Basic concepts of the homography explained with code," OpenCV, [Online]. Available: https://docs.opencv.org/master/d9/dab/tutorial_homography.html. [Accessed 23 April 2020].

[40] "NumPy," NumPy Developers, 2020. [Online]. Available: https://numpy.org/. [Accessed 1 May 2020].

# Appendices:

## Project Planning:

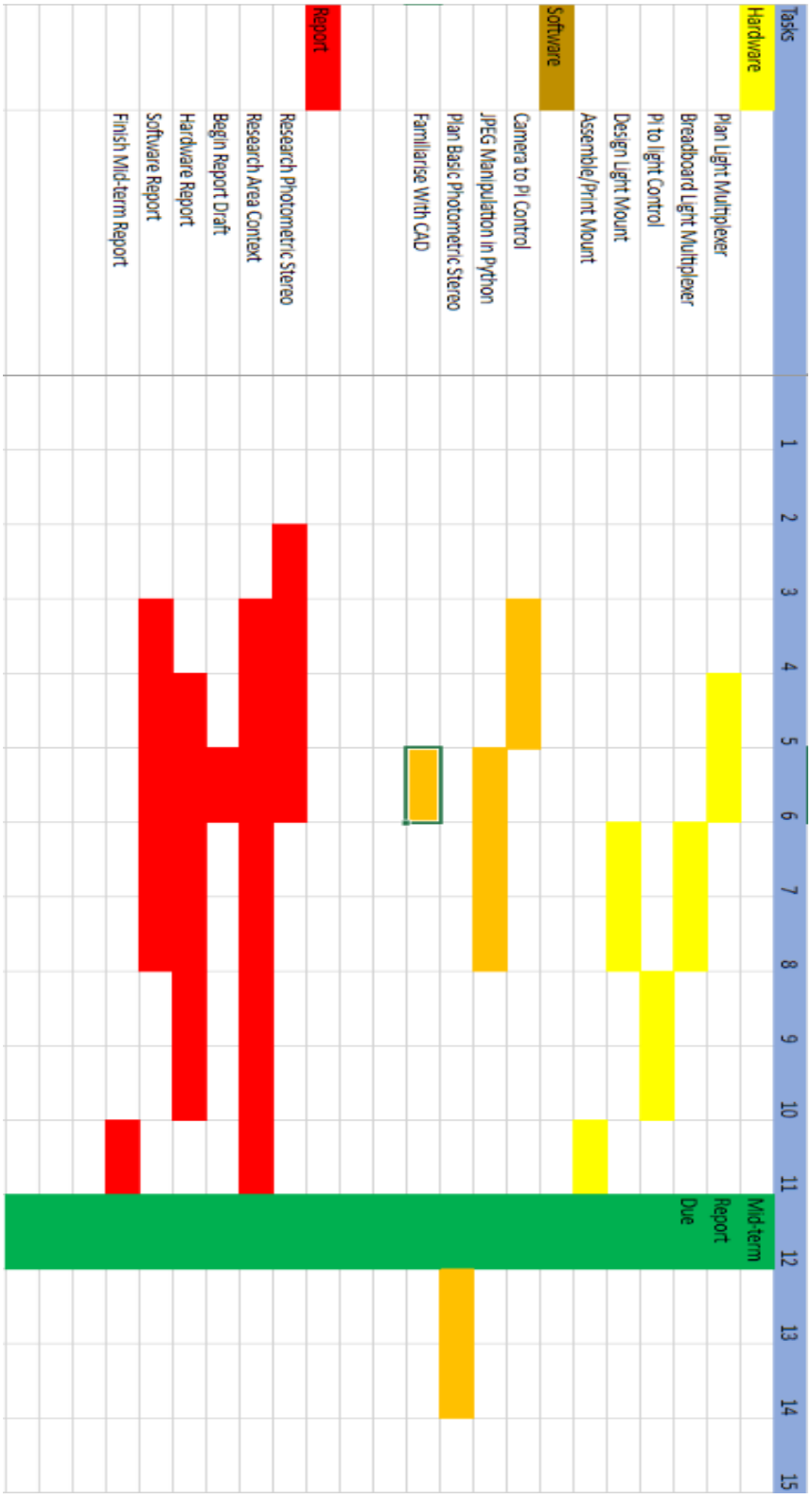


*Figure 34: GANT Chart Semester 1 V1*

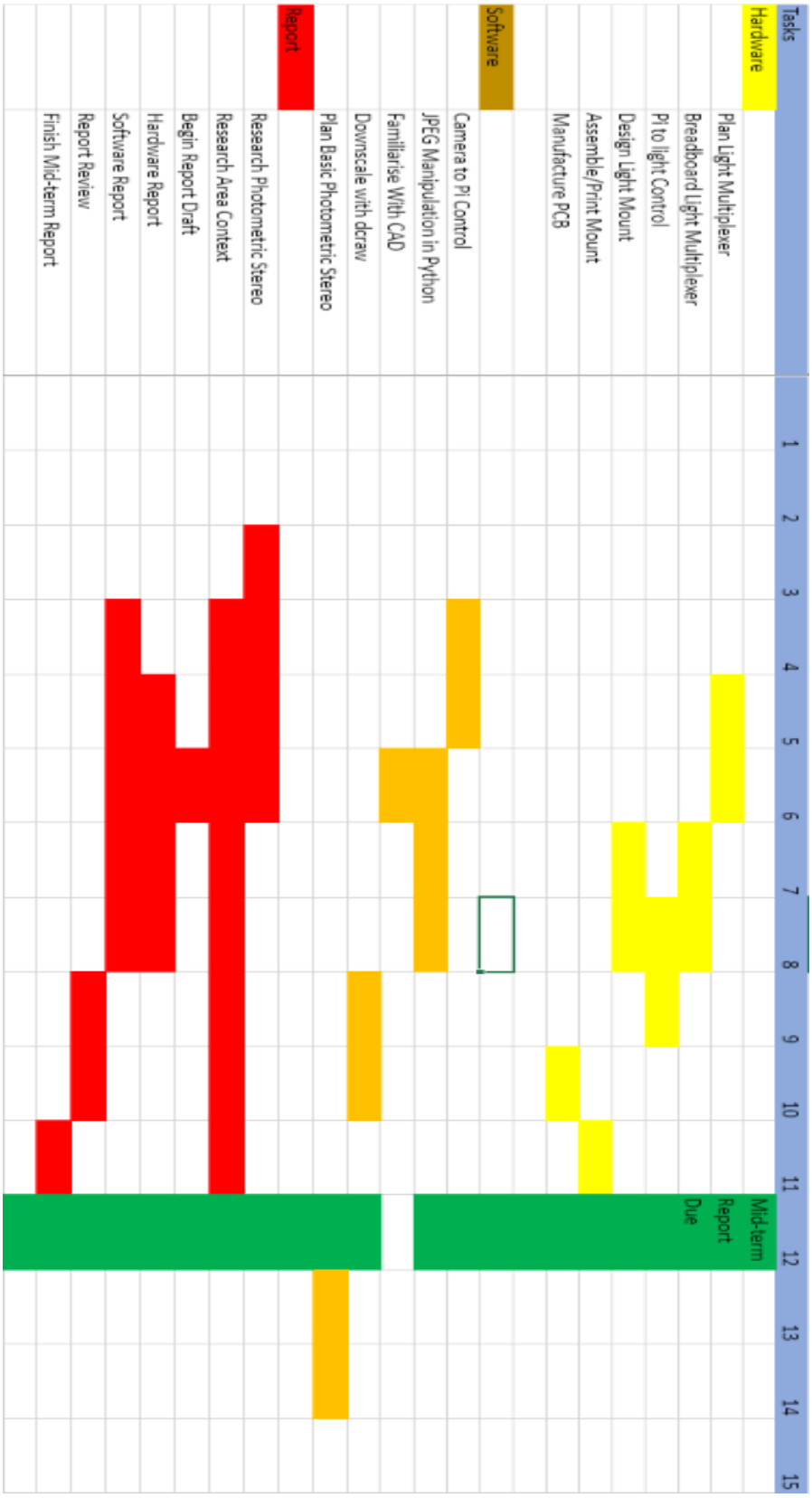


*Figure 35: GANT Chart Semester 1 V2*

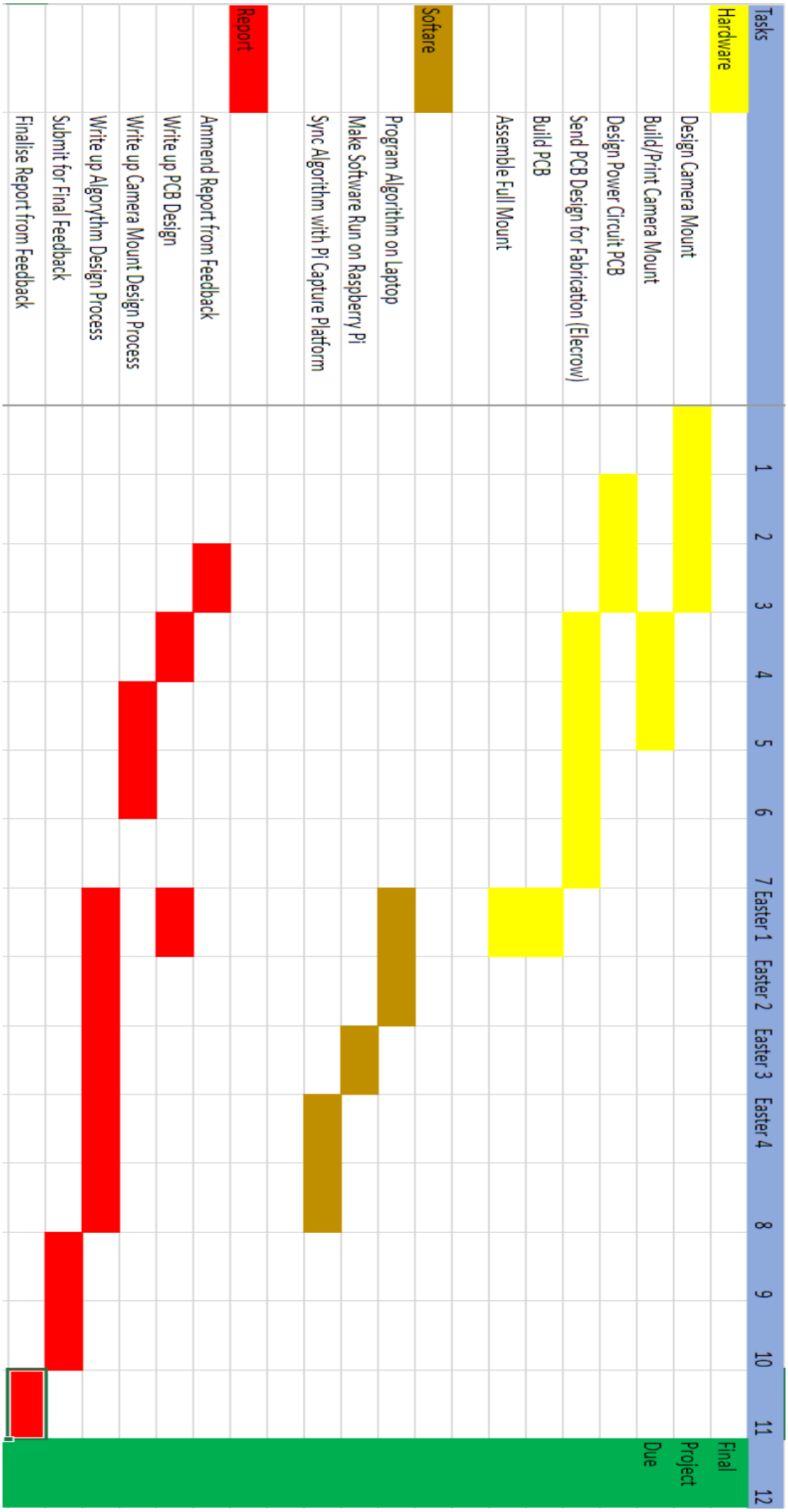


*Figure 36: GANT Chart Semester 2*

**SCHOOL OF ELECTRONICS AND PHYSICAL SCIENCES**

Undergraduate Project Hazards Assessment Form
(to be completed jointly by student/academic supervisor)

Student William SACKSON Project No. 1jg-4096

Project Title SMART FLASH

Location(s) for experimental work (list all) FEPS LAB, HOME WORK

Complete the table below. Up to date risk assessments must be available which cover the hazards of a project, and necessary procedures and precautions explained to the student.

| Which of these hazards are involved in the project? | Delete as applicable | Location of risk assessments | Hazards & precautions explained |
|---|---|---|---|
| Biohazards | ~~YES~~/NO | | |
| Electrical work | ~~YES~~/NO | | |
| Extreme temperatures | ~~YES~~/NO | ELECTRONICS LAB | SOLDERING IRON TRAINING |
| Flammable or explosive substances | ~~YES~~/NO | | |
| Ionising radiation | ~~YES~~/NO | | |
| Knives / cutting tools | ~~YES~~/NO | | |
| Lasers | ~~YES~~/NO | | |
| Lifting equipment | ~~YES~~/NO | | |
| Machinery | YES/NO | FEPS LAB | LAB TRAINING |
| Manual handling | ~~YES~~/NO | | |
| Microwave radiation | ~~YES~~/NO | | |
| Noise | ~~YES~~/NO | | |
| Non-ionising radiation (eg UV, IR, RF) | ~~YES~~/NO | | |
| Power tools | YES/~~NO~~ | FEPS LAB | LAB TRAINING |
| Pressure vessels/systems | ~~YES~~/NO | | |
| Substances hazardous to health eg chemicals, dusts | ~~YES~~/NO | | |
| Vacuum systems | ~~YES~~/NO | | |
| Working at height | ~~YES~~/NO | | |
| Other: (please describe) | | | |

The student has discussed this project with the academic supervisor, and:

- Understands the nature of hazards associated with the project
- Has been instructed in the precautions and procedures to be used to minimise risk from these hazards and the actions to be taken in the event of an accident
- Has attended all appropriate lectures on laboratory safety and procedures during their undergraduate course
- Will ask the supervisor to clarify any H&S matters of which they are unsure

Supervisor Name JEAN-YVES GUILLEMAZ Signature [signature] Date 24/10/2019

Student Name WILLIAM SACKSON Signature [signature] Date 24/10/19

NOW RETURN THIS FORM TO THE TEACHING LABORATORY MANAGER

6 September 2005

*Figure 37: Undergraduate Project Hazards Assessment Form*

## Prototype Development:

```
from time import sleep
from datetime import datetime
from sh import gphoto2 as gp
import signal, os, subprocess
import RPi.GPIO as GPIO
import time

#kill camera app first

def killgphoto2Process():
    p = subprocess.Popen(['ps', '-A'],stdout = subprocess.PIPE)
    out, err = p.communicate()
    #search for kill line
    for line in out.splitlines():
        if b'gvfsd-gphoto2' in line:
            pid=int(line.split(None,1) [0])
            os.kill(pid, signal.SIGKILL)


shot_date = datetime.now().strftime("%Y-%m-%d")
shot_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
picID = "piShots"

clearCommand = ["--folder","/store_00020001/DCIM/100CANON", \ #Deletes all files on DSLR SD card
                "-R", "--delete-all-files"]

triggerCommand = ["--trigger-capture"]

downloadCommand = ["--get-all-files"]

folder_name = shot_date + picID
save_location = "/home/pi/Desktop/gphoto/" + folder_name

def captureImages():#Starts DSLR capture command
    gp(triggerCommand)
    sleep(1)
    gp(downloadCommand)
    gp(clearCommand)

def createSaveFolder(): #Makes a new save folder each day
    try:
        os.makedirs(save_location)
    except:
        print("failed dir init")
    os.chdir(save_location)

def renameFiles():#Renames files to a known set for later processing
    ID = 0
    for filename in os.listdir("."):
        if len(filename) < 13:
            if filename.endswith(".JPG"):
                os.rename(filename,(shot_time +"-"+str(ID)+".JPG"))
                print("Renamed the JPG")
            elif filename.endswith(".CR2"):
                os.rename(filename, (shot_time +"-"+str(ID) + ".CR2"))
        ID += 1
```

*Figure 38: Proof of concept code part 1*

```
#==================--Program Start--============================================================
killgphoto2Process()
gp(clearCommand)
createSaveFolder()

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIOpins=[17,27,22,18]#Ports controlling LED set
for i in range(0,4):
    GPIO.setup(GPIOpins[i],GPIO.OUT)

for i in range(0,4):#Cycle capture with LED lights
    GPIO.output(GPIOpins[i],GPIO.HIGH)
    captureImages()
    GPIO.output(GPIOpins[i],GPIO.LOW)

renameFiles()
```

*Figure 39: Proof of concept code part 2*

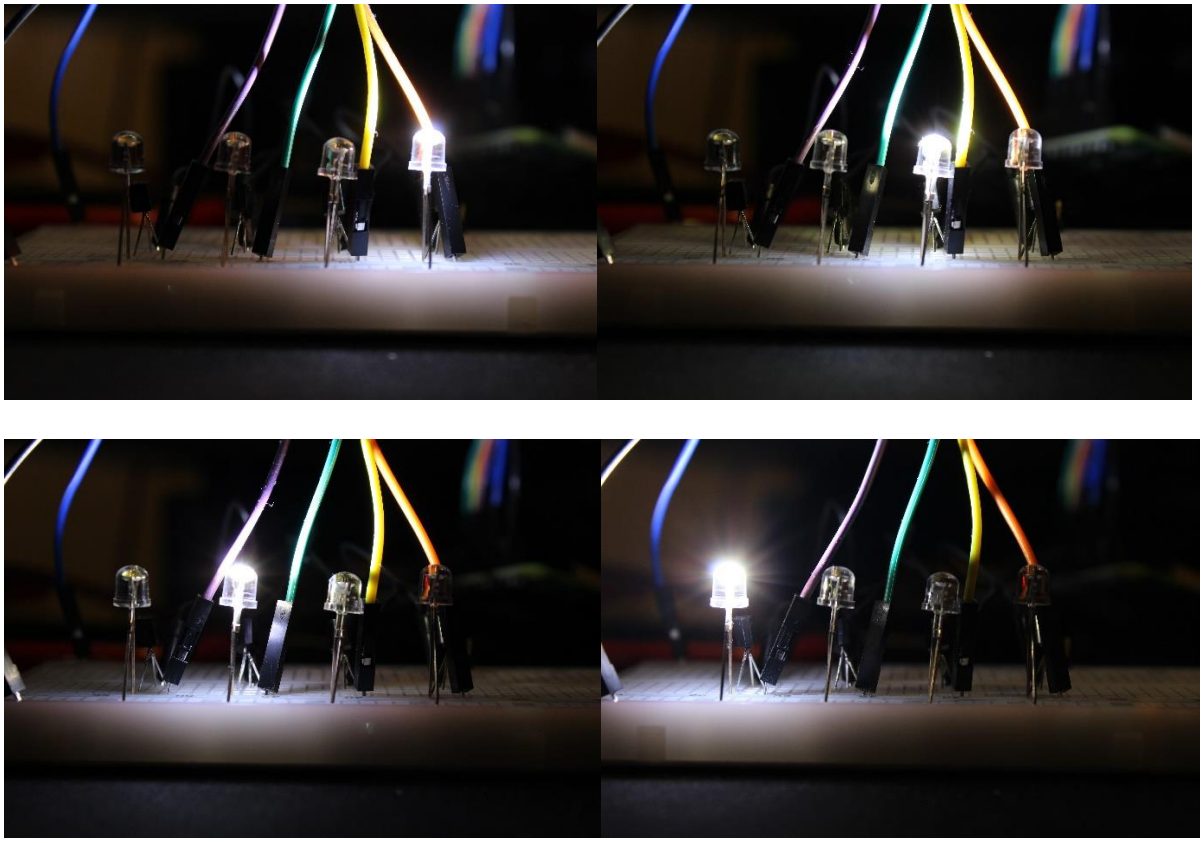

*Figure 40: Example of images taken by camera of capture cycle*

## Final Solution Code:

```
from __future__ import print_function
import cali
import CaptureCode
import numpy as np
import cv2, PIL, os
from cv2 import aruco
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
from time import sleep
from datetime import datetime
from sh import gphoto2 as gp
import signal, os, subprocess
import RPi.GPIO as GPIO
import time
import sh
from rps import RPS
import psutil



GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIOpins = [27,17,22,23,4,14,15,18]
for i in range(0,8):
    GPIO.setup(GPIOpins[i], GPIO.OUT)
    GPIO.output(GPIOpins[i], GPIO.LOW)


def process():
    METHOD = RPS.L2_SOLVER
    DATA_FOLDERNAME = CaptureCode.captureData()
    LIGHT_FILENAME =
'/home/pi/gphoto/PhotometricCameraControl/lights.txt'
    MASK_FILENAME =
'/home/pi/gphoto/PhotometricCameraControl/data/mask2.png'


    rps = RPS()
    rps.load_mask(filename=MASK_FILENAME)    # Load mask image
    rps.load_lighttxt(filename=LIGHT_FILENAME)
    rps.load_images(foldername=DATA_FOLDERNAME,ext='JPG')
    os.chdir(DATA_FOLDERNAME)
    rps.solve(METHOD)
    rps.save_normalmap(filename="./est_normal")
    psutil.disp_normalmap(normal=rps.N, height=rps.height, width=
rps.width)


def menu():
    print("MENU \n 1.Calibrate \n 2.Capture & Process \n 3.EXIT")
    choice = str(input())

    if choice == "1":
        print("Calibrating")
        cali.calibrate()
        menu()

    if choice == "2":
        print("Capturing")
        process()
        menu()

    if choice == "3":
        print("EXITING")

menu()
```

*Figure 41: main.py*

```python
import numpy as np
import cv2, PIL, os
from cv2 import aruco
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd

import CaptureCode

workdir = "./workdir/"
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
board = aruco.CharucoBoard_create(13, 9, 1, .8, aruco_dict)
imboard = board.draw((2000, 2000))
cv2.imwrite(workdir + "chessboard.tiff", imboard)


def read_chessboards(images):
    """
    Charuco base pose estimation.
    """
    print("POSE ESTIMATION STARTS:")
    allCorners = []
    allIds = []
    decimator = 0
    # SUB PIXEL CORNER DETECTION CRITERION
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.00001)

    for im in images:
        #print("=> Processing image {0}".format(im))
        frame = cv2.imread(im)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(gray, aruco_dict)

        if len(corners)>0:
            # SUB PIXEL DETECTION
            for corner in corners:
                cv2.cornerSubPix(gray, corner,
                                 winSize = (3,3),
                                 zeroZone = (-1,-1),
                                 criteria = criteria)
            res2 = cv2.aruco.interpolateCornersCharuco(corners,ids,gray,board)
            if res2[1] is not None and res2[2] is not None and len(res2[1])>3 and
 decimator%1==0:
                allCorners.append(res2[1])
                allIds.append(res2[2])

        decimator+=1

    imsize = gray.shape
    return allCorners,allIds,imsize

def calibrate_camera(allCorners,allIds,imsize):
    """
    Calibrates the camera using the dected corners.
    """
    print("CAMERA CALIBRATION")
    cameraMatrixInit = np.array([[ 1000.,    0., imsize[0]/2.],
                                 [    0., 1000., imsize[1]/2.],
                                 [    0.,    0.,           1.]])

    distCoeffsInit = np.zeros((5,1))
    flags = (cv2.CALIB_USE_INTRINSIC_GUESS + cv2.CALIB_RATIONAL_MODEL +
 cv2.CALIB_FIX_ASPECT_RATIO)
    #flags = (cv2.CALIB_RATIONAL_MODEL)
    (ret, camera_matrix, distortion_coefficients0,
     rotation_vectors, translation_vectors,
     stdDeviationsIntrinsics, stdDeviationsExtrinsics,
     perViewErrors) = cv2.aruco.calibrateCameraCharucoExtended(
                      charucoCorners=allCorners,
                      charucoIds=allIds,
                      board=board,
                      imageSize=imsize,
                      cameraMatrix=cameraMatrixInit,
                      distCoeffs=distCoeffsInit,
                      flags=flags,
                      criteria=(cv2.TERM_CRITERIA_EPS & cv2.TERM_CRITERIA_COUNT, 10000,
1e-9))

    return
 ret, camera_matrix, distortion_coefficients0, rotation_vectors, translation_vectors



def pixelList(images):#Locate the position of bright spots
    maxList = []
    for im in images:
        x = cv2.imread(im)
        gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (131,131), 0)
        (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(gray)
        maxList.append(maxLoc)

    return maxList
```

```python
def calibrate():


#====================================--Intrinsic--=====================================
=========================
    intrinsic = "/home/pi/gphoto/PhotometricCameraControl/workdir/data2/"
    images = np.array([intrinsic + f for f in os.listdir(intrinsic) if f.endswith(".JPG"
) ])
    images.sort()

    allCorners,allIds,imsize= read_chessboards(images)
    ret, mtx, dist, rvecs, tvecs = calibrate_camera(allCorners,allIds,imsize)
#Only using mtx variable

#====================================--Extrinsic--=====================================
=========================
    lights_location = "/home/pi/gphoto/PhotometricCameraControl/workdir/LIGHT/"
    map( os.unlink, (os.path.join( lights_location,f) for f in
 os.listdir(lights_location)) )#Deletes all files in the light direction

    os.chdir(lights_location)
    CaptureCode.captureCycle()

    datadir = lights_location
    images = np.array([datadir + f for f in os.listdir(datadir) if f.endswith(".JPG") ])
    images.sort()
    print(images)


    allCorners,allIds,imsize= read_chessboards(images)
    ret, a, dist, rvecs, tvecs = calibrate_camera(allCorners,allIds,imsize)

#================================================================================
==================
    pixels = np.array(pixelList(images))

    homoPixels = np.empty([3,8])
    homoPixels = cv2.convertPointsToHomogeneous(pixels)

    R = np.empty([3,3])
    cv2.Rodrigues(rvecs[0],R)#The homogeneous coordinates

    N = np.array([np.cross(R[:,0], R[:,1])])#Calculate plane normal

    Z = np.column_stack((R[:,0], R[:,1], tvecs[0]))
    H = np.empty([3,3])
    H = np.dot(mtx,Z)#Get the homography matrix
    HI = np.linalg.inv(H)#Create inverse of homography matrix


    Lights = np.empty((0,3),int)

    for i in range(0,8):e
        m = np.dot(HI,homoPixels[i].transpose())
        m=np.column_stack((m[0]/m[2],m[1]/m[2],1))
#World position of point, normalize around Z
        #print(m)

        P = np.dot(R,m.transpose())
        P = np.add(P,tvecs[0])#Camera coordinates of position

        L = P        #Incident line from camera to point

        L1= np.squeeze(np.asarray(L))
        N1= np.squeeze(np.asarray(N.T))
        Ref = (2*(N1*L1)*N1)-L1 #Get the reflection
        Ref = Ref / np.linalg.norm(Ref)
        Lights = np.append(Lights, np.array([Ref]),axis=0)

    print(Lights)
    os.chdir("/home/pi/gphoto/PhotometricCameraControl/")
    np.savetxt("lights.txt", Lights, fmt="%s")
```

*Figure 42: cali.py*

```
from time import sleep
from datetime import datetime
from sh import gphoto2 as gp
import signal, os, subprocess
import RPi.GPIO as GPIO
import time

#kill camera app first

def killgphoto2Process():
    p = subprocess.Popen(['ps', '-A'],stdout = subprocess.PIPE)
    out, err = p.communicate()
    #search for kill line
    for line in out.splitlines():
        if b'gvfsd-gphoto2' in line:
            pid = int(line.split(None, 1) [0])
            os.kill(pid, signal.SIGKILL)



clearCommand = ["--folder","/store_00020001/DCIM/100CANON", \
                "-R", "--delete-all-files"]

triggerCommand = ["--trigger-capture"]

downloadCommand = ["--get-all-files"]


def captureImages(i):
    try:
        gp(triggerCommand)
        sleep(0.2)
        GPIO.output(GPIOpins[i], GPIO.LOW)
        sleep(0.25)
        gp(downloadCommand)
        gp(clearCommand)
    except:
        for i in range(0,8):
            GPIO.output(GPIOpins[i], GPIO.LOW)

def createSaveFolder(save_location):
    try:
        os.makedirs(save_location)
    except:
        print("failed dir init")
    os.chdir(save_location)

def renameFiles():
    ID = 0
    for filename in os.listdir("."):
        if len(filename) < 13:
            if filename.endswith(".JPG"):
                os.rename(filename,(str(ID)+".JPG"))
                print("Renamed the JPG")
            elif filename.endswith(".CR2"):
                os.rename(filename,(str(ID)+".CR2"))
        ID += 1

#==============================--Program Start--=======================
=========
killgphoto2Process()
GPIOpins = [27,17,22,23,4,14,15,18]
#createSaveFolder()

def captureCycle():
    gp(clearCommand)
    GPIOpins = [27,17,22,23,4,14,15,18]

    for i in range(0,8):
        GPIO.output(GPIOpins[i], GPIO.HIGH)
        captureImages(i)

    renameFiles()

def captureData():

    shot_time = datetime.now().strftime("%Y-%m-%d@%H:%M:%S")
    picID = "piShots"
    folder_name = shot_time + picID
    save_location = "/home/pi/gphoto/PhotometricCameraControl/data/" +
 folder_name
    createSaveFolder(save_location)
    captureCycle()

    return(save_location + '/')
```

*Figure 43: CaptureCode.py*