University of Strathclyde Glasgow

# Wi-Fi Ultra Low Power Stereo Camera (WULPSC)


**Group I**

| | |
|---|---|
| Aneel Amjad | 201909408 |
| Raheel Amjad | 201909392 |
| Andreas Gavrielides | 201927856 |
| Konstantinos Gkogkalatze | 201922539 |
| Yiannis Michael | 201934829 |

Department of Electronic & Electrical Engineering

19520/EM501

University Of Strathclyde


April 22, 2024

# Abstract

Low-power, remote visual monitoring devices remain a largely unexplored application in the area of stereo vision. An ultra low-power, remote visual monitoring device, which utilises the principles of stereo vision, is a novel monitoring solution. Such a device retains the advantages of contemporary monitoring solutions, but expands the breadth of information that can be gathered about an observed object, by capturing 3D information. With this project, a device of this nature has been successfully implemented and tested. Constituent hardware components have been acquired, or designed and fabricated; the necessary on-board firmware and microcontroller code has been developed and implemented; and a fully functional web interface has been developed, capable of performing the necessary image processing, saving results to a database, completed with an interactive user interface.

Performance evaluation revealed promising results. Depth estimation errors were generally low, particularly in the range of 30cm to 150cm, showcasing the effectiveness of the system in estimating distances accurately. Size estimation, however, showed varying degrees of error. When the line (defect) was perpendicular to the camera, the error was relatively low in the range of 20cm to 200cm, but increased for distances beyond that range. For angled lines, the error was higher in the range of 30cm to 100cm. Notably, the system encountered challenges in estimating sizes for distances outside these ranges, often resulting in inconsistent estimations due to disparity values being significantly off. These findings highlight both the capabilities and limitations of the system, providing valuable insights for further optimization and improvement.

# Contents

# 1 Introduction

The goal of this project is the design and development of a remote, ultra-low power, WiFi, stereo camera device. The device captures images using both of its camera sensors and transmits them over WiFi to the host server. The host server will use the image to infer three-dimensional information about the monitored object. To do so, purpose-written algorithms are used, allowing for the creation of depth maps and even 3D models. The user is able to interact with the device via a website featuring a fully functional graphical user interface. The website allows the user to inspect captured images and to adjust various settings, such as image capture frequency and image brightness.

Motivation for this project stems from the lack of representation of stereo imaging techniques in low-power monitoring solutions. While stereo vision monitoring is not a new idea [1]–[3], this project emphasises the ultra low-power aspect of the device, so that it can be deployed in remote, hard to reach areas for extended periods of time. This helps in reducing or eliminating the need for manual inspection, lowering both costs and risk associated with inspection.

During development, the specific application of pipe monitoring was considered. As such, certain aspects of the device have been developed and optimised for such an application, such as the optimal distance between the device and the monitored object. However, the results of this project can easily be generalised to other industries, including manufacturing [4], infrastructure inspection [5], and agriculture [6], where remote monitoring and defect size calculation are crucial for maintenance and quality control.

This report begins with a section covering the background necessary to understand the nature of the project. Then, design and implementation of all the constituent parts of the project will be covered, including hardware, microcontroller code, back-end and front-end software and image processing algorithms. Sections dedicated to the testing of the project, the discussion of the results and an overview of the budget follow. Finally, a section contemplating further improvements or expansions upon the project.

# 2 Background

## 2.1 Non-linear Distortion

Cameras capture images by concentrating incoming light onto a sensor. For this to be done, a convex lens must be use. Such a lens introduces non-linear distortion to the captured data, making what is a straight line appear curved [7]. Figure 1 illustrates the effect of non-linear distortion.

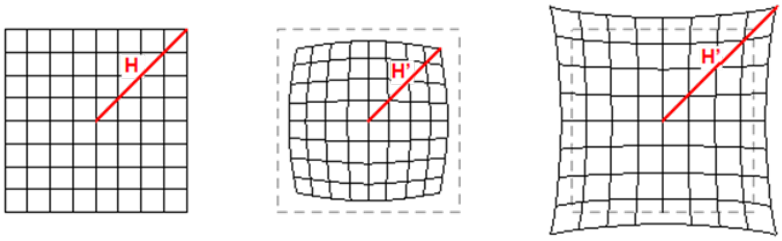



Figure 1: Non-linear distortion: Undistorted (Left), Barrel Distortion (Middle), Pincushion Distortion (Right) [7]

## 2.2 Undistortion

Undistortion is the process of reversing the errors introduced by camera sensors [8]. To accomplish this, certain information about the camera hardware must be known, specifically the focal length, optical center and skew coefficient. With this information, an undistortion algorithm can be employed to minimise distortion. Figure 2 shows how undistortion can fix the non-linear error introduced to a picture of a chessboard.

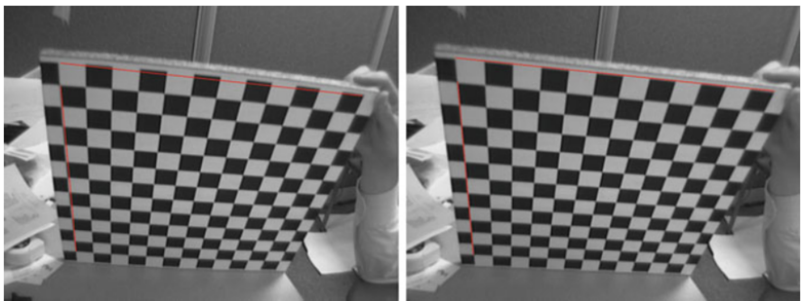


Figure 2: Non-linear correction on a picture of a chess board: Uncorrected (Left) Corrected (Right) [8]

## 2.3 Rectification

Rectification is a transform applied to a pair of stereo images to project them onto a common plane [9], [10]. This is done to eliminate the the difference in the planes on which the camera sensors lie, as even with highly precise equipment, small errors in coplanarity might exist. The result of rectification is that any slanting present in the epipolar line (the line which matches the pixels between the two stereo images) is amended, making the epipolar line straight and parallel to the line connecting the two camera sensors. This process greatly simplifies the process of stereo matching.

## 2.4 Disparity

Disparity is a phenomenon observed in stereoscopic visual systems [11]. It is defined as the shift in the apparent location of an object along the axis connecting the two camera sensors which captured the pictures. Figure 3 illustrates disparity in a pair of stereo images.

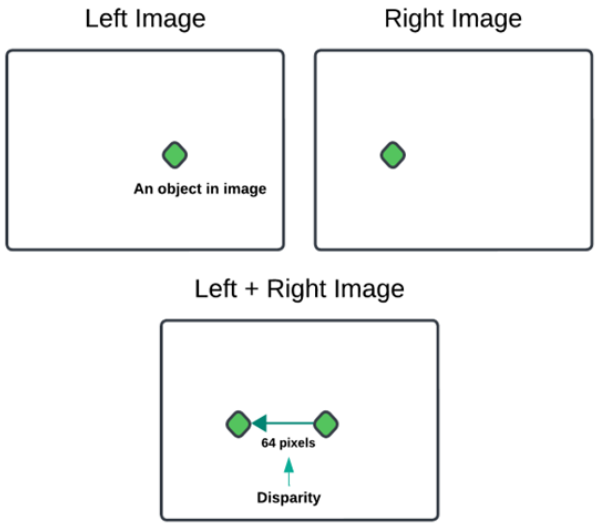



Figure 3: Visualisation of disparity in a stereo configuration

## 2.5 Stereo Matching

Stereo matching plays a crucial role in enabling depth perception from stereo images captured by a stereo camera system [12]. Stereo matching algorithms analyze corresponding points in left and right images to establish depth information. Among these algorithms, block matching is a fundamental technique widely employed for its simplicity and effectiveness. Figure 4 illustrates the block matching process, where the left and right images are divided into small blocks in the same line. These blocks are then compared between the two images to find corresponding features [13]. By measuring the similarity or disparity between these features, stereo matching algorithms can determine the depth of objects in the scene.

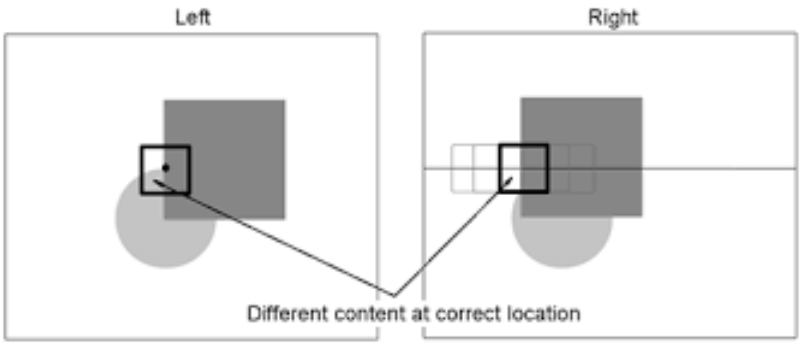



Figure 4: Block Matching [14].

Block matching operates on the principle that corresponding points in the left and right images have similar intensity patterns. However, challenges such as occlusions, textureless regions, and lighting variations can affect the accuracy of stereo matching [15]. To address these challenges, advanced algorithms

such as Semi-Global Matching (SGM) and Weighted Least Squares (WLS) filtering are often employed, enhancing the accuracy and robustness of depth estimation [16].

### 2.5.1 Semi-Global Matching Algorithm

The Stereo Disparity Using Semi-Global Block Matching (SGBM) algorithm is a fundamental technique for calculating depth information from stereo camera images. This algorithm plays a crucial role in distance estimation for various applications [17]. By computing the disparity between left and right images, SGBM enables the extraction of object depth based on the displacement of corresponding points, also known as disparities. The block-based approach adopted by SGBM leverages information from neighboring pixels in multiple directions, optimizing the disparity calculation while ensuring accurate depth estimation. Figure 5 showcases the disparity between the left and right images, highlighting the variations used in the matching process.

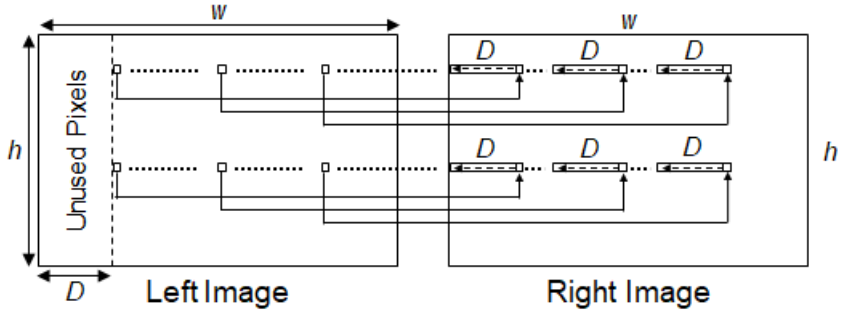



Figure 5: SGBM Disparity Levels [18].

The SGBM algorithm encompasses three major modules: Matching Cost Calculation, Directional Cost Calculation, and Post-processing [18]. In the Matching Cost Calculation module, the algorithm employs the Census Transform to compute the matching cost, which measures the similarity between corresponding points in the left and right images. The Directional Cost Calculation module aggregates minimum cost paths from multiple directions, enhancing disparity smoothness and accuracy. Finally, the Post-processing module applies interpolation and uniqueness functions to refine the disparity map further. Figure 6 illustrates the procedure of the SGBM algorithm, depicting the stages of Matching Cost Calculation, Directional Cost Calculation, and Post-processing. The comprehensive understanding of these modules elucidates the intricate workings of the SGBM algorithm and its significance in stereo disparity computation for various real-world applications.

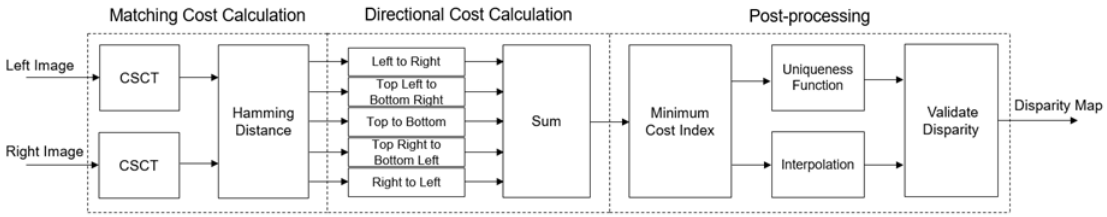



Figure 6: SGBM Block Diagram [18].

### 2.5.2 Weighted Least Squares (WLS) Filter

The Weighted Least Square (WLS) filter is crucial in improving the disparity map obtained from stereo matching algorithms like Semi-Global Block Matching (SGBM). This edge preserving filter is particularly effective in addressing artifacts and noise present in the initial disparity map, resulting in a refined and more accurate depth estimation. By considering the local characteristics of the disparity map and having spatial information, the WLS filter assigns weights to each pixel disparity based on its similarity with neighboring disparities [19].

The WLS filter functions by considering the disparities computed from both the left and right stereo images enhancing the overall disparity map. By adjusting parameters [20] such as the regularization parameter ($\lambda$) and the sigma color parameter ($\sigma$), the WLS filter fine-tunes the balance between smoothing and preserving disparities. The filter performs an adaptive weighted averaging operation over local disparity neighborhoods, removing noise and outliers while preserving disparities corresponding to scene features. The resulting disparity map, after applying the WLS filter, provides more reliable depth estimation and 3D reconstruction.

## 2.6 Similar Triangles

The concept of similar triangles is a fundamental principle for establishing relationships between geometric figures. When two triangles share identical angles, they are considered similar, even if they vary in size [21]. This principle aids in extracting relationships between corresponding sides of these triangles. A common method for identifying these relationships is through the comparison of corresponding sides, knows as the ratio of similarity. This ration remains constant across all corresponding sides of similar triangles. As depicted in Figure 7, where triangles ABC and EGF share congruent angles, the ratio of the lengths of their corresponding sides remains consistent as shown in Equation 1.

$$\frac{AB}{EG} = \frac{BC}{GF} = \frac{AC}{EF} \tag{1}$$

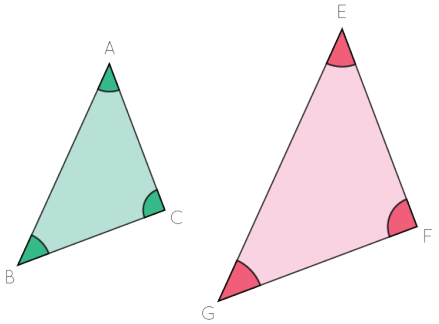



Figure 7: Illustration depicting the principle of similar triangles, showcasing triangles ABC and EGF with congruent angles and proportional sides. [22]

# 3 Literature Review

A variety of relevant literature exists in the field of stereoscopy and monitoring of defects. Some sources are analysed in this section, and any correlations, ambiguities, and gaps are examined.

The first study used a stereo setup to extract 3D information [11]. A deep learning network, CNN (Convolutional Neural Network) was used to identify and quantify defects. The model has been trained on several defect images. The main regions of interest for the study are pipelines with metal loss, corrosion, and cracks. Simple trigonometry was used to measure the defects and estimate their severity. An infrared projector was used in a process known as active stereoscopy since a projector is utilised. This allowed for a more detailed disparity map and a better 3D point cloud. An algorithm automatically generated a bounding box around the region of interest. The camera used was an Intel RealSense D435, with a resolution of 1280 x 720.

Similarly, another source [23] also focused on defect inspection with an Intel RealSense D435i. This study compared a passive stereo camera setup (ZED Camera) and an active stereo setup (Intel RealSense) to evaluate which one performed better. As expected, the active setup achieved better results. The defects observed were on bridges, with UAS (Unarmed Aircraft Systems) carrying the payload. Both sources emphasised the benefits of a stereo vision approach in saving time and labour compared to manual inspection.

The combination of stereo vision and infrared was also observed in another study, which focused on underwater pipeline inspection [24]. Two IR cameras were calibrated for stereo vision, and an RGB camera provided a colour-accurate 3D representation. Accuracies in the range of 1mm were obtained for diameters ranging from 400mm to 700mm. A robot was used to carry the sensor suite, with all the processing done on board. A 120m tether line linked the control station and the robot for communication and control.

Some of the most precise sizes detected were observed in a source that used two Genie Nano M402 monochrome cameras, with a resolution of 4112 x 3008 [25]. The baseline used was 130mm, resulting in a much lower window of effectiveness for object distance. Reflective surfaces were inspected with the combination of stereo vision and blue light projection. Dents, bumps and scratches of lower than 0.1mm were identified.

Another study utilises stereo vision for defect inspection but does not use infrared [26]. It's cited as being hard to utilise, due to the uncontrollable temperature of air and fluids inside pipelines, creating noise. The study used two commercial HD webcams, and all tests were under 50cm in distance from the object being monitored. The baseline used was 4cm. A robot was used to carry the cameras on an axis, allowing them to tilt 90 degrees and capture the perfect angle. The automatic classification of defects was not explored, unlike in the first study [11].

Stereo vision can also be used in manufacturing processes, to act as quality control for parts. Specifically, one study observed [27] used stereo vision to inspect and classify shapes of square/circle parts. The height of the part is also automatically measured, and stereo vision is described as a more powerful tool than other traditional methods involving a single camera. Commercial webcams were used for the sensors, and OpenCV libraries utilised in Python for image processing algorithms. Average height estimation errors ranged from 0.39mm to 0.85mm on objects varying from 20mm to 30mm in height.

A comparison table of all the mentioned sources can be seen below at Table 1. All of the mentioned sources approach a similar problem, the need to reduce manual labour in inspecting defects. Some studies utilised additional sensors such as lasers, to obtain more accurate depth information and higher accuracies. A common theme among all the studies was the use of a high-resolution camera, with the Intel RealSense D345 variants being used in two [11], [23].

There was also a difference in each source's intended application environment. The choice of sensors and approach used varied on the conditions and accuracy required for monitoring. For problems requiring analysis of smaller defects, smaller baselines were used, such as 45.5mm [27] and 130mm [25]. Objects being monitored from longer distances used higher baselines, such as 4cm [26].

However, there is a clear gap in the field of stereo inspection. None of the sources focus on efficient low-power monitoring at inspection sites. The majority of them utilise robotics [23], [24], [26] to maneuver sensors around, and capture photos, with no research involving remote communication. This creates a clear disadvantage in the runtime of the battery-powered devices. In this report, a cloud-based approach is presented, utilising WiFi connectivity to achieve remote processing. With the combination of low-power hardware and lack of onboard processing, device lifetimes in the neighborhoods of years can be achieved.

Table 1: Comparison of relevant literature

| **Study Reference** | **Use Case** | **Size of Defects** | **Camera** | **Resolution** | **Stereo Type** | **Baseline** |
|---|---|---|---|---|---|---|
| [11] | inspection of oil and gas pipelines, corrosion, cracks, leaks | 10-50mm | Intel RealSense D345 | 1280x720 | active, 2x camera + IR projector | n/a |
| [23] | inspection of bridges via drones | n/a | Intel RealSense D345 | 1280x720 | active 2x camera + IR projector | n/a |
| [24] | inspection of underground pipelines | 1mm-500mm | unspecified | 1280x720 | active, 2 IR cameras and RGB camera | n/a |
| [25] | inspection of highly specular surfaces | 0.003m-0.07mm | Genie Nano M4020 | 4112x3008 | active, 2 cameras + laser line projector | 130mm |
| [26] | inspection of sewer pipes | 5mm-15mm | commercial HD webcams | 1920x1080 | passive, 2 cameras | 4cm |
| [27] | inspection of manufactured parts | 20-30mm | commercial cameras | 640x480 | passive, 2cameras | 45.5mm |

# 4 Budget

Budget was a constant consideration whenever a choice had to be made for hardware components. Overall, the budget was managed responsibly, with about £65 left over from the budget total of £500. The pie chat in Figure 8 on the next page shows a breakdown of all the individual purchases made within the scope of the project.

To ensure proper budget management, a cost-benefit analysis was performed before each purchase. A sizeable part of the spending was dedicated to experimental equipment, such as the FPC 24-Pin converter boards and HOBFU Webcams, with the webcam purchase being the largest singular purchase made for the project. Another costly purchase was the battery pack. Due to restocking issues, a cheaper battery pack was unavailable. This necessitated the purchase of a more costly alternative. Using the University fabrication lab also ended being one of the largest expenses at £38, with externally ordering a PCB from JLCPCB costing £26. Only the OV2640 that were shipped with the ESP32-CAMs were used, with extenders being bought to enable more free flexibility when selecting their positions.

To eliminate the possibility of hardware failure sabotaging the progress of the project, spare components were ordered wherever the budget would allow. This increased the cost of some purchases, but the safety provided against time delays was deemed worth the trade-off. Additionally, some components could only be purchased in numbers much larger than were necessary for the project. While this once again helped provide safety against hardware failure, it somewhat increased spending. One notable example of this were the Renesas IDTQS3VH257 MUX/DEMUX ICs, which had a minimum order size of 10 units. A small part of the budget was spent on acquiring miscalleneous items, such as jumper wires. Such items were presumed to be available through the university, but due to the large number required for testing, orders had to be put in for them.

Overall, even though some unfortunate and unforeseen expenses were made throughout the duration of the duration of the project, thanks to careful management of the budget, part of the budget was still left over by the end. Furthermore, if multiple copies of the device were to be produced, the cost per unit would be significantly lower, as duplicates of many parts were purchased, while some other parts were only used in testing. When factoring in the cost of only the parts used, the device cost about £130. If the cheaper battery pack had been purchased, the cost of the device would be much closer to £100. With economies of scale, the cost of the device could be further reduced.

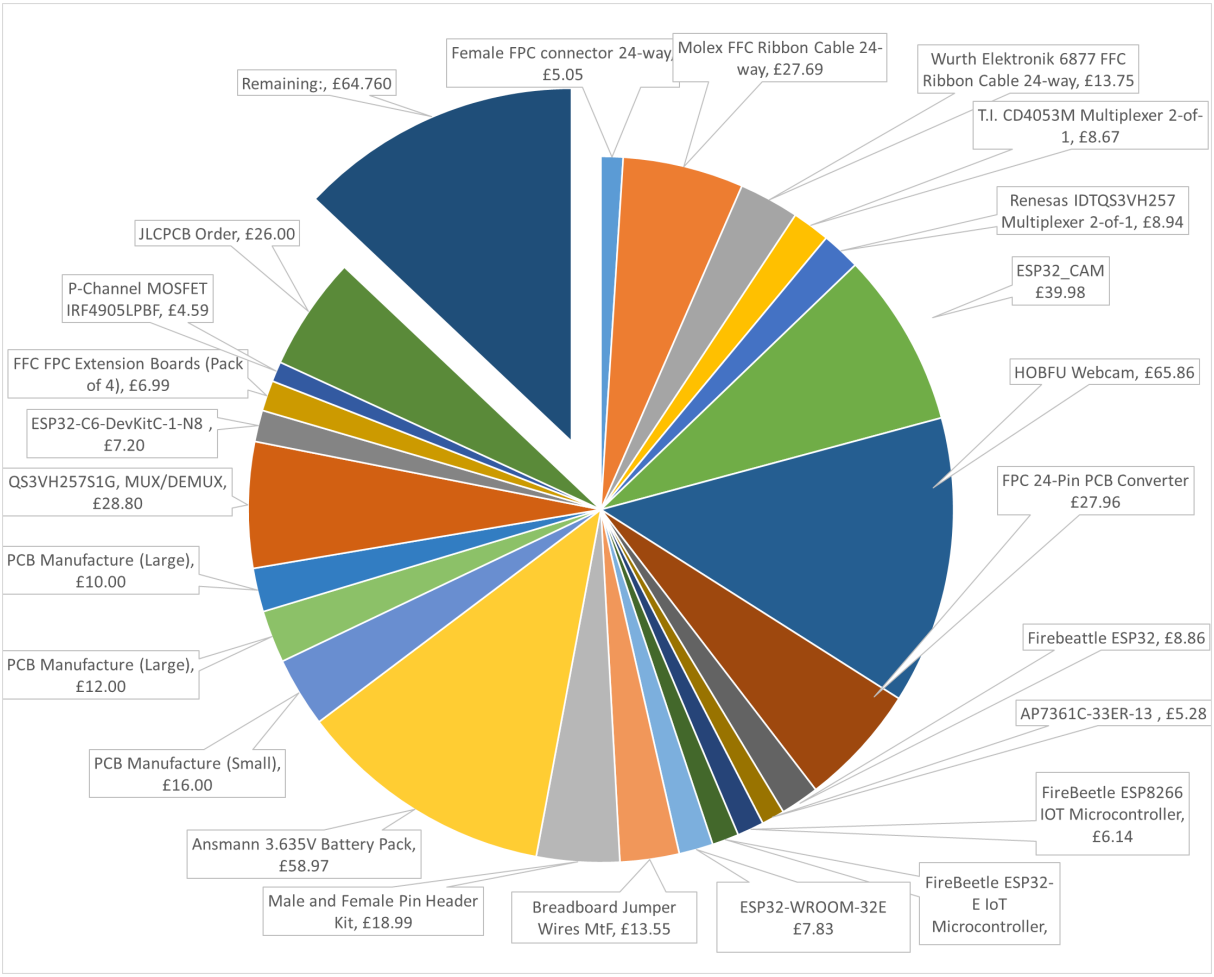



Figure 8: Project Budget Breakdown

# 5 Hardware

This section details all the individual components which comprise the Hardware aspect of the project and their function. These include the microcontrollers, camera sensors, Printed Circuit Boards (PCB), battery module, enclosure, and other miscellaneous components. Additionally, the considerations for each choice are discussed, along with any challenges encountered. The diagram shown in Figure 9 provides a high-level explanation of the interactions between the server, the two microcontrollers, the multiplexing PCB and the Camera sensors.

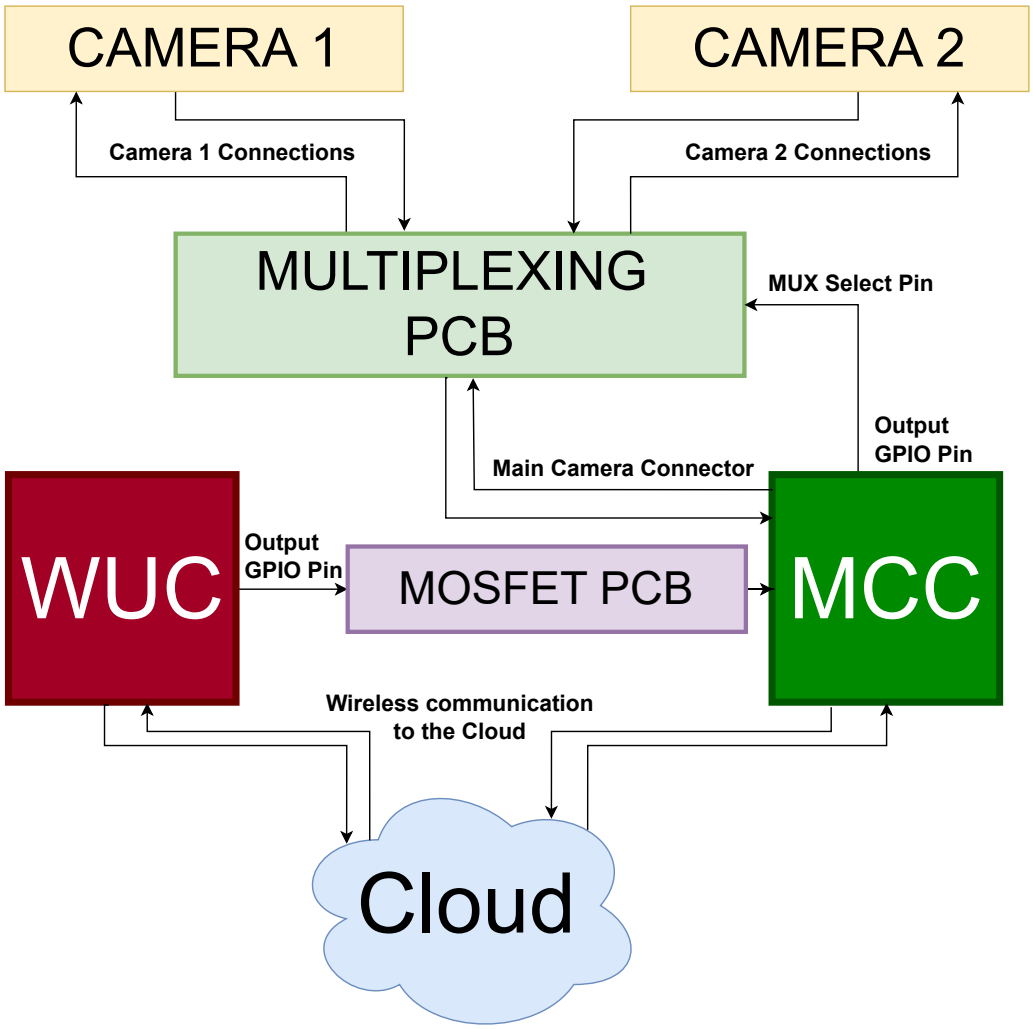



Figure 9: High-Level Hardware Overview [28]

## 5.1 Microcontrollers

Two Microcontrollers are employed to enable the functionality of the device, due to their different attributes and strengths. One microcontroller acts as the Wake-Up Controller (WUC), while the other acts as the Main Camera Controller (MCC).

### 5.1.1 Main Camera Controller (MCC)

The MCC is responsible for controlling the two camera sensors to capture images and then transmit the information to the Host Server. An ESP32-CAM is used for this purpose [29]. The MCC interfaces with the two camera sensors by multiplexing the connections of the cameras, using a custom-designed PCB (Section 5.3.2).

The ESP32-CAM was selected for various reasons. One simple yet crucial reason is the existence of a 24-pin FPC connector [30]. The lack of such a connection would complicate development and also make the multiplexing PCB much bigger and more cumbersome. Another important reason is the extensive catalogue of documentation [31] which facilitates code development. Libraries and APIs are provided by Espressif [32], and example projects can be found online, which helped gain a better understanding of said libraries and APIs. Unfortunately, the ESP32-CAM could not also be used as the WUC, due to its high deep-sleep current draw [28].

### 5.1.2 Wake-UP Controller (WUC)

A Firebeetle 2 ESP32 takes up the role of the WUC. This means that it is responsible for establishing a connection with the Host Server and for sending a signal to the MOSFET PCB (Section 5.3.3) to wake up the MCC.

The Firebeetle microcontroller was chosen due to its low deep-sleep current draw, according to both its data sheet [33] and measurements performed manually (Table 2). The Firebeetle 2 ESP32 could not fill the role of the MCC, due to its lack of a female 24-pin FPC connector.

Table 2: Current Draw and Estimated Battery Life for Each Tested WUC [28]

| **MCU Model** | **Expected Deep Sleep Current Draw** | **Measured Deep Sleep Current Draw** | | **Expected Battery Life (days/ year)** | **Battery Life According to Measured Deep Sleep Current Draw** | |
|---|---|---|---|---|---|---|
| | | **5V @ 180 mA** | **3V3 @ 180 mA** | | **5V @ 180 mA** | **3V3 @ 180 mA** |
| FireBeetle ESP32 | 10 uA | - | 1.25 mA | 41.166/ 114.15 | - | 505.1/ 1.383 |
| FireBeetle 2 ESP32-E | 20 uA | 0.72 mA | 1.61 mA | 20.833/ 57.08 | 578.7/ 1.585 | 392.1/ 1.074 |
| ESP32-WROOM-32E | 5 uA | 1.85 mA | - | 83.333/ 228.31 | 225.2/ 0.617 | - |
| ESP32-S3-WROOM-1 | 8 uA | 4.65 mA | - | 52.083/ 142.69 | 89.60/ 0.245 | - |
| ESP32-WROVER-E | 5 uA | 5.0 mA | - | 83.333/ 228.31 | 69.44/ 0.190 | - |

## 5.2 Camera Sensors

The model of camera sensors used for the device is the OmniVision OV2640 [34]. As mentioned in the interim report [28], the OV2640 was initially chosen due to the convenience of being shipped with the ESP32-CAM. Other camera sensors may boast improvements in certain aspects like resolution or current draw, but they come with trade-offs. Due to the relative ease of sourcing the OV2640, combined with its widespread use in microcontroller-based monitoring projects, it remained the model of camera sensor used throughout the project.

## 5.3 PCBs

As iniitally conceptualised in the interim report [28], a PCB has been designed and fabricated, which facilitates the connections between the MCC and two camera sensors. Additionally, a secondary, smaller PCB has been designed and fabricated to safely power the MCC on and off by utilising a GPIO pin on the WUC.

### 5.3.1 Multiplexers

To implement the multiplexing functionality, dedicated multiplexing IC chips were required. The Renesas IDTQS3VH257 MUX/DEMUX [35] surface-mounted ICs are used to control the bi-directional flow of the signals between the camera sensors and ESP32-CAM.

The main reason for choosing this IC model is it implements both multiplexing and de-multiplexing functionality, meaning that bi-directional communication can be established between the camera sensors and the ESP32-CAM. Another reason is the high bandwidth of up to 500MHz, ensuring that the performance of the overall system would not be affected.

Figure 10 shows the schematic of the IC. Pin 1 is the 'Select' pin, which dictates which of the Inputs is currently active; it is connected to a GPIO pin on the MCC. Pin 8 connects to Ground and pin 16 is the Voltage Common Connector/Voltage Supply, which are supplied through the MOSFET PCB. Pin 15 is the disable pin: if its input is high, all inputs are disabled; it is hard-wired to Ground, as its function is unneeded. The remaining pins are inputs/outputs for multiplexing/de-multiplexing. Camera pins are connected as the inputs and the pins of the ESP32-CAM as the outputs.

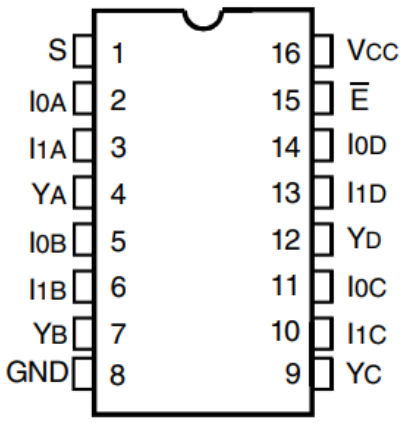



Figure 10: Renesas Multiplexer Schematic [35]

### 5.3.2 Multiplexing PCB

The multiplexing PCB is responsible for powering the components connected to it, including the MCC, the camera sensors, and the MUX/DEMUX ICs. Power to the MUX PCB is supplied by the MOSFET board through the MCC, as will be discussed in Section 5.3.3. The PCB houses the physical FPC connectors for the camera sensors and the ESP32-CAM. The FPC connectors used contain 24 pins and have a 0.5mm pitch. Due to the manner in which the ESP32-CAM interacts with a camera sensor, certain pins need to be connected in a different way to others. Some pins need to be shared between the two camera sensors, where others need to be multiplexed to prevent interference.

To understand what type of connection is needed for each pin, research had to be conducted into existing solutions. However, due to the limited number of existing solutions, manual experimentation had to also be conducted. Because this is a Transistor-Transistor Logic (TTL) circuit, simulations such as those performed for the MOSFET PCB were deemed unnecessary. This necessitated created experimental setups using breadboards before proceeding to PCB fabrication. Figure 11 shows a picture of one experimental setup.

At first, no pre-existing solution was sought out. The first approach was to simply multiplex every single one of the 24 pins. This did not work, as the MCC would fail to initialise the cameras. It was then clear that seeking out a pre-existing solution would be beneficial. The Maix Binocular Camera [36] was the only appropriate solution discovered, and its layout provided a good starting point for testing. However, copying the layout of the Maix Binocular Camera PCB still yielded errors in camera initialisation.

To attempt to understand the issue, the ESP32-CAM [30] was extensively studied. After numerous trials and pouring over the datasheets, two significant changes separated the pin connections of the Maix PCB and the designed PCB. Firstly, whereas in the Maix PCB the reset ('RST') pin is multiplexed, in the designed PCB is it hard-wired to logical high. Secondly, the power down ('`PWDN`') pin in the Maix PCB is hard-wired to logical low, as it is unused; in the designed PCB it is shared between the camera sensors and used to power them down when necessary. The connections between pins in the designed multiplexinng PCB can be seen in Table 3.

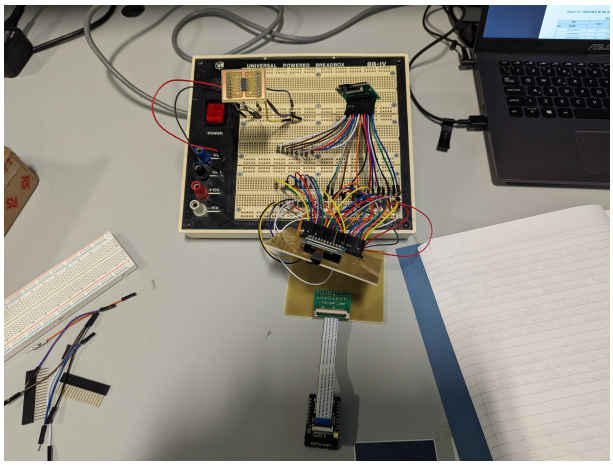


Figure 11: Experimental Breadboard Setup

Table 3: Multiplex PCB Pin Connections

| Pin Number | Pin Name | Connection on PCB |
|---|---|---|
| 1 | - | Not Connected |
| 2 | - | Not Connected |
| 3 | D2 | Multiplexed |
| 4 | D1 | Multiplexed |
| 5 | D3 | Multiplexed |
| 6 | D0 | Multiplexed |
| 7 | D4 | Multiplexed |
| 8 | PCLK | Multiplexed |
| 9 | D5 | Multiplexed |
| 10 | DGND | Shared |
| 11 | D6 | Multiplexed |
| 12 | MCLK | Shared |
| 13 | D7 | Multiplexed |
| 14 | DOVDD | Shared |
| 15 | DVDD | Shared |
| 16 | HSYNC | Multiplexed |
| 17 | PWDN | Shared |
| 18 | VSYNC | Multiplexed |
| 19 | RST | Hard-wired High |
| 20 | SCK | Multiplexed |
| 21 | AVDD | Shared |
| 22 | SDA | Multiplexed |
| 23 | AGND | Shared |
| 24 | - | Not Connected |

#### 5.3.2.1 First Version

Initially the multiplexing functionality was investigated using a breadboard setup. However, as the breadboard setup was difficult and unreliable to work with, as connections were unstable and wire would come loose, the move to a PCB was crucial. Thus, the design for a first version of the PCB was created and fabricated. The schematic is shown in Figure 12, while a picture of the PCB is shown in Figure 13.

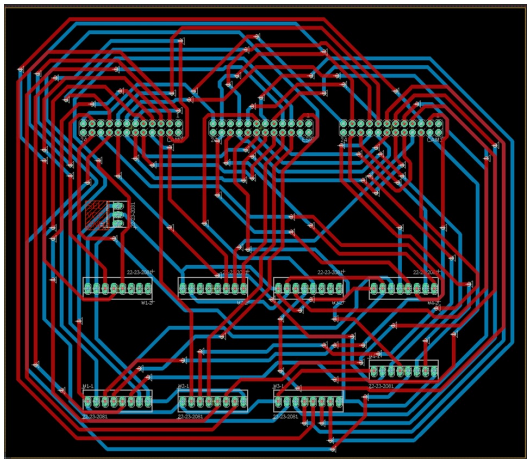


Figure 12: Schematic of Initial Multiplexing PCB

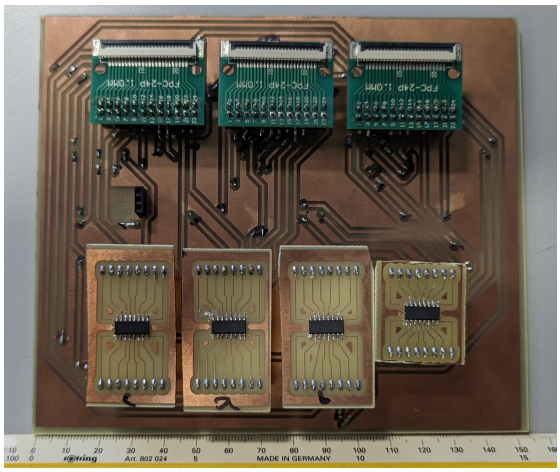


Figure 13: Picture of Initial Multiplexing PCB

This iteration of the PCB design was limited by the capabilities of the university's fabrication laboratories, such as the minimum distance between the traces on a PCB. This meant that the Renesas Multiplexers and the FPC connectors, would be either very difficult or impossible to place directly onto the PCB. Due to these limitations, this PCB uses smaller breakout boards for the Multiplexers and FPC connectors. As a result, the dimensions of the resulting PCB were larger than desirable at about 15.3cm x 13.2cm, which led to the design of the second version PCB.

#### 5.3.2.2 Final Version

After ensuring that the initial PCB design facilitated the proper function of the entire system, a second design was made. This design would need to be fabricated by an external fabrication laboratory, and it utilises surface mount components, drastically reducing its footprint to 5.6cm x 9.8cm, thus allowing the final device to be smaller and be deployed in more constricted environments. The schematic of the final PCB can be seen in Figure 14, and a picture of it can be seen in Figure 15.

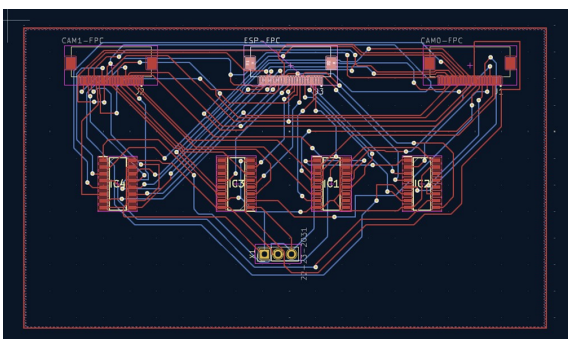



Figure 14: Traces of Final Multiplexing PCB

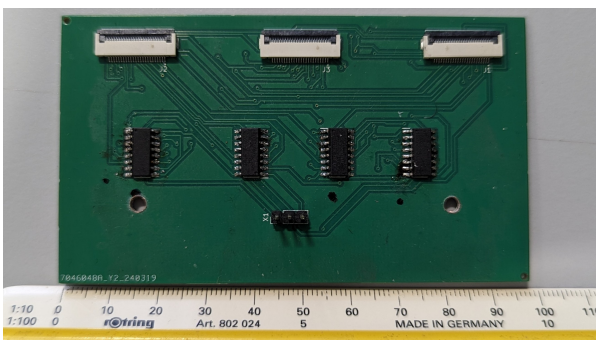



Figure 15: Picture of Final Multiplexing PCB

### 5.3.3 MOSFET PCB

After further research from the time of the interim report, it was decided that a secondary PCB had to be constructed to control the provision of power to the MCC, so that it could be completely powered off when the device was not in active use, and thus reduce its power consumption. As the name suggests, the WUC controls when the MCC gets powered, and when that occurs is discussed later in Section 6.1.2.

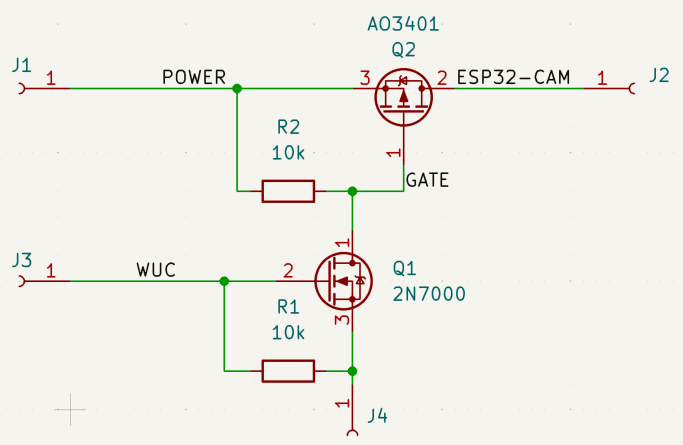



Figure 16: Circuit Schematic the MOSFET PCB in KiCAD.

The circuit diagram for the MOSFET PCB is shown in Figure 16. A common ground connection is in place for both the WUC and MCC to be connected to through their ground pins. The final system uses a custom soldered three-way female-to-female jumper cable to connect all three male headers (MCC, WCU, and MOSFET Board) together. The rest of the connections are detailed in Table 4. The two MOSFETs used in the final version of the board are the AO3401 [37] p-channel MOSFET and the 2N7000 [38] n-channel MOSFET. The AO341 was selected for its very low drain to source resistance ($R_{DS}$) of $47m\Omega$ at the gate threshold voltage ($V_{GS}$) of $-3.3V$ with a maximum drain current ($I_D$) of $2A$. When selecting the 2N7000, only the $V_{GS}$ mattered because the nMOS would short-circuit the gate of the pMOS to ground. Therefore a max $V_{GS}$ of $3V$ was a good match for the GPIOs logical high $3.3V$.

Examinging the circuit in 16, resistor $R1$ is used to pull the GPIO to ground to make sure the pin is zero when below the $V_{GS}$ of the 2N7000. Resistor $R2$ is used for current limiting and to pull the gate terminal of the p-channel MOSFET to be equal to the source terminal, therefore making the pMOS an open circuit. Once the WUC GPIO2 pin is logic high, the n-channel MOSFET connected to that pin becomes a closed circuit, therefore shorting the gate terminal of the p-channel MOSFET. By doing so the p-channel MOSFET becomes a closed circuit and power is able to flow through to the MCC.

There were some issues deciding on a p-channel MOSFET, due to having difficulty finding a through-hole component able to be used for the electronic ratings of the project. The p-channel MOSFETs VP0109 [39] and IRF4905LPbF [40] were experimented with, but the first had too high $R_{DS}$ and the latter had too high $V_{GS}$. A low $R_{DS}$ is required so that the required current can flow through the pMOS to the MCC, and a $V_{GS}$ that includes 3V3 in its range is also required.

Table 4: MOSFET Board Connections to the two microcontrollers

| **Schematic Reference** | Microcontroller Pin**(s)** |
|---|---|
| J1 | WUC $V_{cc}$ Pin |
| J2 | WUC GPIO2 |
| J3 | MCC 3V3 Pin |
| J4 | WCU and MCC Ground pins |

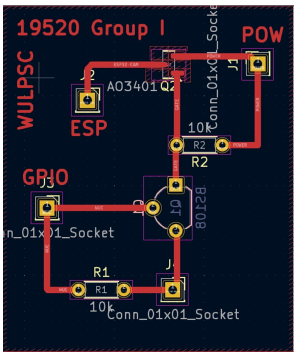



Figure 17: PCB Layout of MOSFET PCB

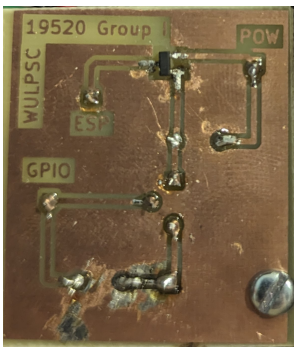



Figure 18: Picture of MOSFET PCB

The PCB layout can be seen in Figure 17, while a picture of the PCB can be seen in Figure 18. The dimensions of the MOSFET PCB are 3.3cm x 3.9cm.

## 5.4 Battery and Power

The device is designed to operate using a rechargeable battery. The battery is directly connected to the WUC, meaning that the WUC is the only device connected directly to the battery. The WUC is setup to use its $V_{cc}$ pin to provide power to the MCC through the MOSFET PCB by setting the specified GPIO pin to logical high. In turn, through the MCC $V_{cc}$ pin the multiplexing PCB is also powered with 3.3 V, therefore powering all of the multiplexers.

The choice of battery for the device is an important one, as it largely affects the safety, longevity and battery life of the device. As outlined in the interim report [28], a Lithium Ion battery pack was desirable, as it is in many ways the safest and most environmentally friendly battery technology. Due to the choice of WUC [33] , a nominal voltage of 3.6V is necessary. An appropriately large capacity is important, as it largely dictates the battery life of the device. The set goal for battery life is a minimum of one year. A desirable capacity of around 10 Ah is chosen based on this goal; the relevant testing is outlined in Section 8.1. Based on all of the above criteria, an Ansmann brand battery pack was chosen (Li-Ion-3.63V-10500mAh-1S3P-PTC/LG, model no: 2447-3034-220) [41], which consists of 3 INR18650 LGC MJ1 cells [42], [43]. To recharge the battery, an appropriate charging station is required.

## 5.5 Enclosure

### 5.5.1 Requirements

To house all of the electronic components, an enclosure meeting certain criteria had to be designed. Firstly, every component would need to be mounted in an adequately secure manner, so they would not be damaged during transportation. Secondly, the enclosure would need to reasonably protect the internal components from the elements, mainly by being resilient to foreseeable external forces and providing reasonable ingress protection. Lastly, to aid in troubleshooting and maintenance, the disassembly and reassembly process for the enclosure had to be as quick and convenient as possible, while minimally sacrificing on the previous two criteria.

### 5.5.2 Modelling

3D printing was the method chosen to create the enclosure, as it offers convenience and accuracy, while at the same time allowing for intricate designs. Autodesk Fusion 360 was the software used to create the 3D model of the enclosure. The 3D printers at the University manufacture labs were used for printing. An image of the model is shown in Figure 19. Blue circles indicate the protrusions to accommodate the camera sensors; behind them are sliders to hold in place the extension PCBs which hold the cameras and connect to the Multiplexing PCB. Green rectangles show the barriers which hold the battery in place in the horizontal plane . Red circles reveal the positions where magnets are to be placed, which will adhere the main body and top panel of the enclosure. Purple and yellow circles indicate the positions where threaded inserts will be inserted; the ones in purple to enable the mounting of the PCBs, and those in yellow to enable the mounting of the FireBeetle 2 ESP32-E microcontroller.

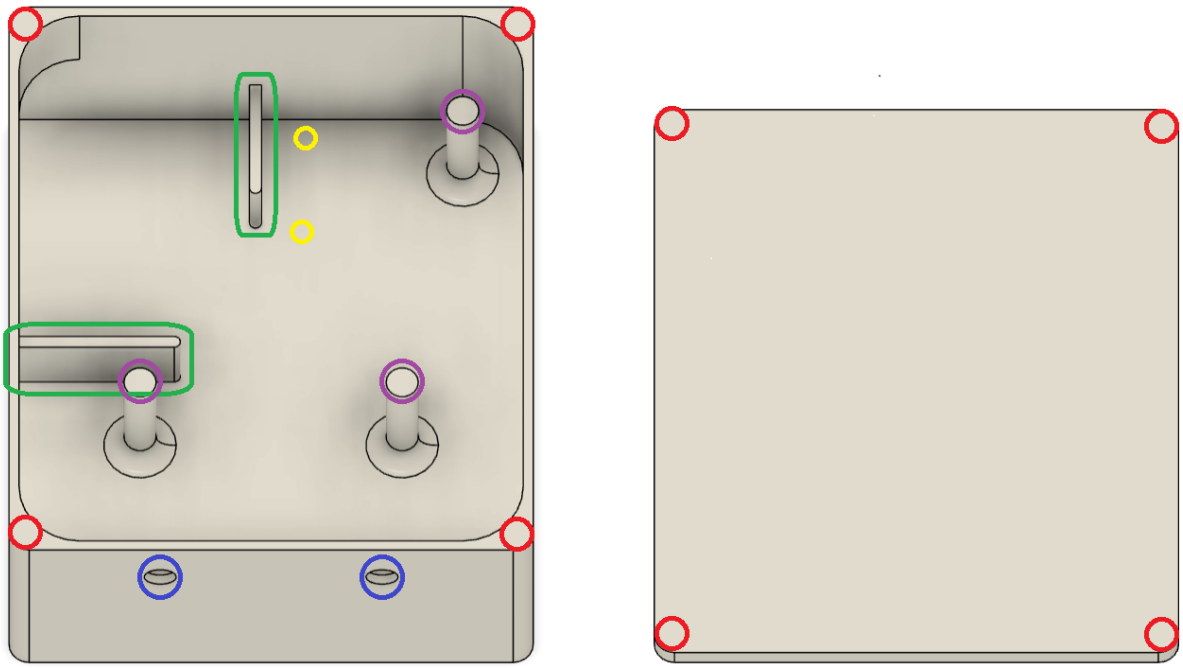


Figure 19: Image of Enclosure 3D Model; main body (left) and top panel (right)

### 5.5.3 Finalised Enclosure

After 3D printing the model, some work needed to be done manually. Most important among these are the threaded inserts (Figure 20). To mount the inserts a soldering iron is used; the insert is placed over the desired location, the soldering iron is used to heat it up, and the plastic around it melts, allowing the insert to slowly sink into place. The inserts are used to mount the two PCBs and the FireBeetle 2 ESP32-E microcontroller. The FireBeetle 2 ESP32-E microcontroller's PCB has pre-drilled holes which allow M2 screws to be used, whereas holes were drilled in the custom-designed PCBs to accommodate screws. The ESP32-CAM microcontroller's PCB has no pre-drilled, nor the clearance to drill any, so some double-sided tape was placed on its micro-SD card slot to secure it in place. Some miscellaneous components were also glued onto the enclosure, including a switch to safely disconnect the battery (Figure 21) and a 2-pin Molex screw terminal (Figure 22). To secure the battery from vertical movement, Velcro was used to secure it to the bottom of the enclosure. Lastly, as rubber gaskets were unavailable through the university's manufacturing labs, a sheet of cork was recommended as a stand-in, and glued to the top part of the enclosure. A picture of the assembled device with dimensions 15cm x 13cm x 6.8cm, can be seen in Figure 23, the final wiring diagram is also displayed at

Figure 20: Picture of Threaded Inserts

Figure 21: Picture of Switch

Figure 22: Picture of Screw Terminal

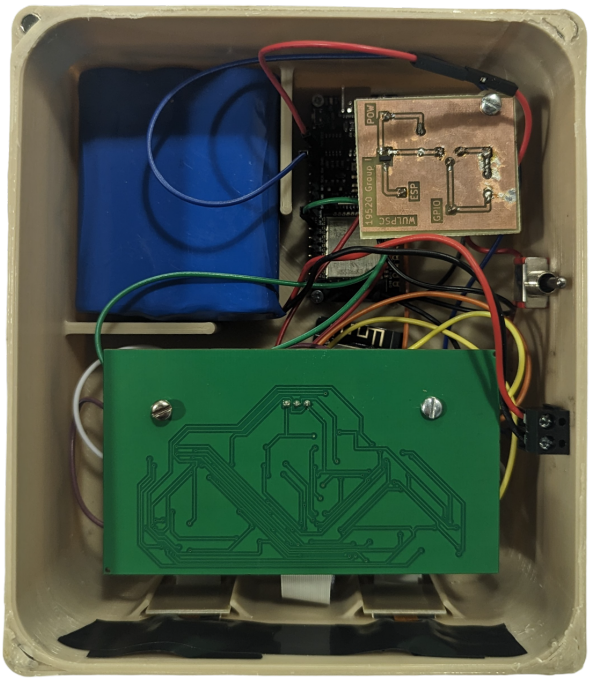


Figure 23: Picture of Assembled Device

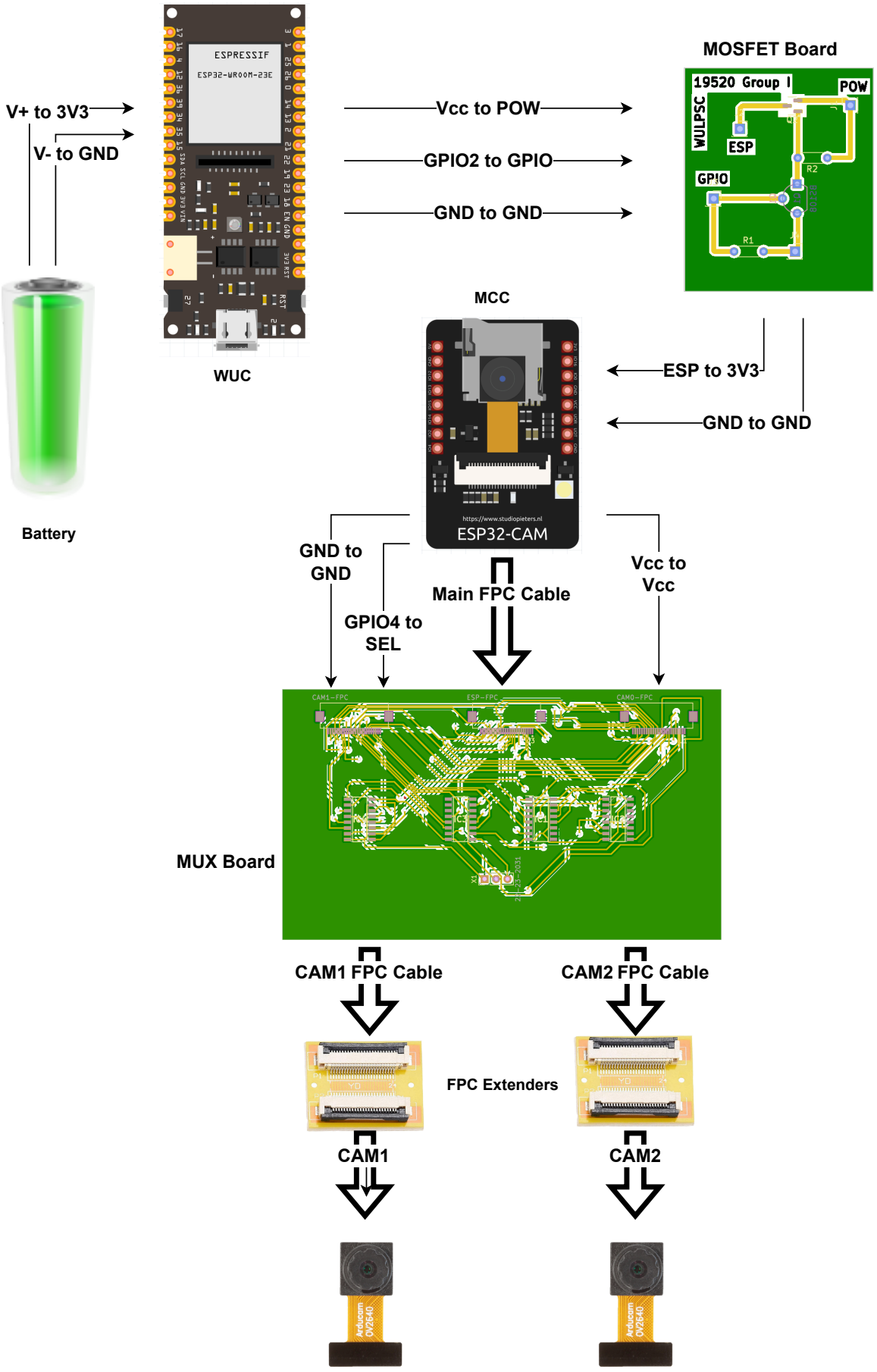



Figure 24: Final Project Wiring Diagram

# 6 Software

Project's software design provided the methods of which the hardware could achieve the project objectives. Therefore, all the different facets of the project's software, from the microcontroller code, to the front-end and back-end of the server-side code. Implementation details and the reasoning behind the choices made are discussed in this section.

## 6.1 Microcontroller Code

As aforementioned, the project used two microcontrollers, the ESP32-CAM [44] for the MCC and the FireBeetle 2 ESP32-E [33] for the WUC. The ESP32 chip present in both allowed the use of the Espressif IoT Development Framework (ESP-IDF) [45] to accomplish each of the microcontroller's software requirements. Code style used was based on the Espressif IoT Development Framework Style Guide [46], so as to make the code look similar to other ESP32 code projects and therefore making the code more easily understood. A range of typical applications scenarios [47] are provided by the manufacturer with examples [48] for each component also provided to aid development and design. For this project, code examples as WiFi STA [49], WiFi Smart Config with EspTouch [50], HTTP Server [51], Non-Volatile Storage (NVS) [52], SD Card [53], GPIO [54], Deep Sleep [55], and ESP Camera [56] were studied, modified and applied to achieve project objectives. With a complex microcontroller chip such as the ESP32, examples in this way must be used so that time is not wasted trying to understand and implement functionalities that have already been documented and tested for developers.

Initially, in this section a brief overview of the code for each of the two modules will be described using the corresponding flowchart. By expanding upon each flowchart block and cross-referencing with the code, a more in-depth description will also be given for how the system accomplishes the corresponding functionality. A description of the design decisions, as well as how the examples were modified to achieve project objectives will also be given. How and why each non-IDF function was implemented will later be described with documentation provided in the project header files. Functions provided by the ESP-IDF will be given a general description and reasoning, but for a more in-depth description the corresponding manufacturer's documentation will be cited. All handwritten code in all `source`, `include` directories and `main.c` is commented, and placed in the Appendix (Listings 2 to 21) and the most recent version of both of the projects is uploaded to the GitHub repository [57].

### 6.1.1 General Code Architecture

Both projects follow similar structures and setups that allow for better code comprehension and code transfer between projects. A system configuration variable is declared as global for both systems to allow access of system settings and status throughout the project. The system configuration variable for the WUC is named `wuc_config` and is declared of the custom structure type `wuc_config_t` which is defined in the `wuc_config.h` header file. Similarly the MCC system configuration variable is named `mcc_config` and is declared of the `mcc_config_t` custom structure type, which is defined in `mcc_config.h`. Directories like `\include` and `\source` contain the header files and source files respectively of the different components of the project. For example, the files `sd.h` and `sd.c` contain the documentation, declaration, and definition of functions and macros used for the SD card functionality in the system. The WUC does not have those files implemented because it doesn't use an SD card, however code used in the WiFi component was easily able to be transferred from the MCC project due to this code architecture. Some other ESP32 projects have a `\components` directory for this type of modular coding style [], but this more traditional method was favoured due to having to implement less CMake configuration files.

By building the code in either of the two main repository folders using ESP-IDF, the rest of the required files are auto-generated into the project directory. An `idf_component.yml` file was created to add dependencies to the projects and allow for more reproducible code, since the versions noted in the file are checked and downloaded at compile time. The `idf_component.yml` for the MCC has the `esp-camera` dependency to automatically download the camera drivers and place them into a default `\managed_components` directory in the project itself. The `CMakeLists.txt` files are used to manage the compile paths of the project, with only the one in `\main` being modified to include all of the source files in `\source` and the header files in `\include`. General all-purpose functions and data types are used that are provided by Espressif, and the more specific ones will be explained in Sections 6.1.2.2 and 6.1.3.2. The ESP Logging library [58] is used to output logs to terminal for debugging and testing. Error codes and Helper functions [59] are also used to aid debugging by creating and checking functions of the `esp_err_t` data type. Depending on the returned value the system has certain behaviours or outputs certain errors to the terminal for debugging and troubleshooting.

### 6.1.2 WUC

The WUC's responsibility is to connect to the main server, wake up the MCC to take the pictures, and enter deep sleep to limit the system's power consumption. Therefore, it should utilise the WiFi, HTTP and Deep Sleep capabilities of the Espressif chips. Through further refining of the system design, the NVS was used to allow the microcontroller to connect to the same network at each boot, and WiFi Smart Config was used to retrieve new network credentials at the first boot. The sleep time should also be modifiable by the user so as to allow different intervals of system wake-up time and active time. In Section 6.1.2 and all subsections, how these capabilities were implemented are described. Power for the WUC is provided by the battery as detailed in Section 5.4

#### 6.1.2.1 Flowchart Overview

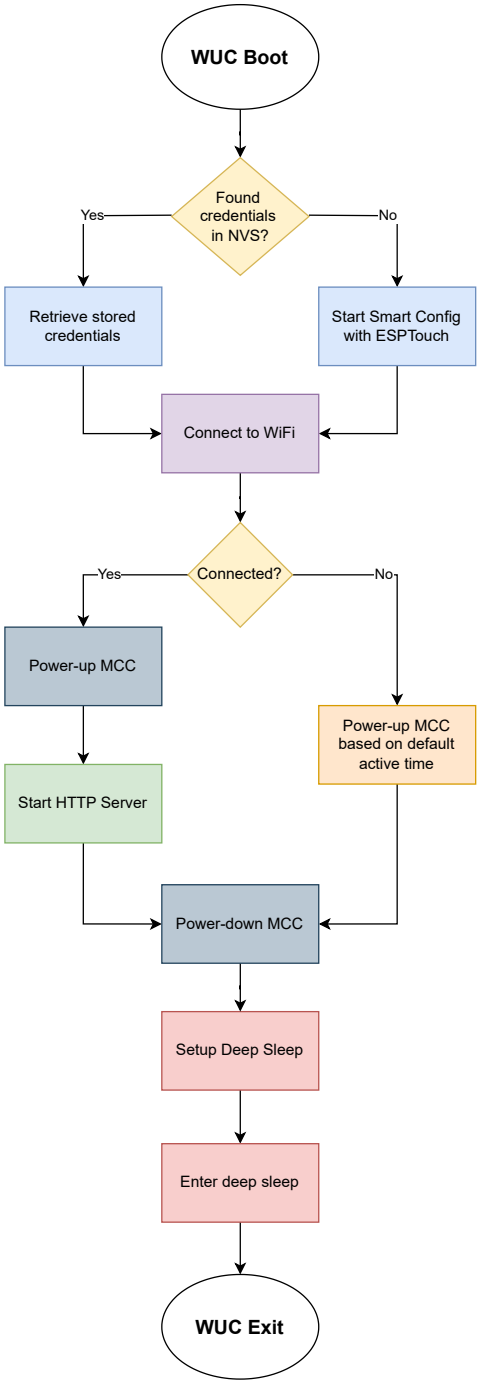



Figure 25: Flowchart for the WUC Functionalities

Using the WUC module flowchart in Figure 25, a brief overview of the system functionality can be visualised. At each boot iteration, the system checks if the WiFi credentials are in the NVS. At the very first boot is when the storage has no credentials saved, therefore Smart Config with EspTouch is executed to retrieve the required credentials. More details regarding Smart Config will be in Section 6.1.2.1.2. After the first boot, the credentials are stored in the system's NVS and are therefore retrieved to connect to the WiFi. Once a WiFi connection is established, the system starts the HTTP server to receive GET or POST requests, and powers up the MCC.

When the user decided to turn the system off, deep sleep setup occurs and then the system sleeps for the specified amount of time. If no connection to WiFi is established, the system executes its hardcoded default cycle, which is to sleep for one day, and to wake up to turn on the MCC for one minute.

#### 6.1.2.1.1 Checking the NVS for credentials

First process taking place within the system checks the NVS for the WiFi credentials. This occurs within the `check_nvs()` function. To retrieve the correct values from storage, the size of the values must be retrieved first in order to allocate the correct space for them to be read back into the program. Retrieving these sizes is done using the `get_nvs_sizes()` function, which is called inside `check_nvs()`. Non-Volatile Storage in the ESP32 works by assigning a "key" that refers to a specific section in the system memory. In the project's case, these sections are referred as `ssid` and `pswd`, and their sizes are retrieved from the ESP32 using the `nvs_get_str()` function by passing these key values with a `NULL` value in its third argument. Their corresponding sizes get stored in the `ssid_required_size` and `pswd_required_size` variables by passing their addresses to the function.

The values from the two size variables are checked directly by `check_nvs()` to tell the system whether it should initialise Smart Config to get new values or to read the credentials from storage. To accomplish this, the sizes are simply checked if they are more than zero, thus indicating that they exist in the NVS. In the case they are zero Smart Config procedure must be started. If they are more than zero the sizes are used to allocate that specific amount of space into the system configuration SSID and Password variables, to be used later during WiFi initialisation.

#### 6.1.2.1.2 Connecting to WiFi

WiFi functionality is achieved by using the credentials stored in the system configuration variable. As mentioned before, Smart Config with EspTouch (Example screenshot shown in Figure 26) is initialised in the instance of first boot where no credentials would be found in the system. To use EspTouch, it has to be installed through its `.apk` file on the GitHub repository [60], or if using an iPhone, it can be installed from the AppStore. The device with EspTouch has to be connected to the desired network, and then the SSID and Password is passed by the app to the ESP to establish the connection. Turning on GPS on the device with EspTouch is also necessary for this process to work. After inputting the network password and pressing confirm, a brief wait time of 10-20 seconds passes while the app and the microcontroller exchange the required packets to establish a connection to the network. If credentials are found in the NVS of the microcontroller, this entire step is skipped and the WiFi connection is established by initialising the ESP WiFi in station mode (STA) and connecting using the read credentials. An important note is that the ESP32 requires a router capable of a 2.4 GHz WiFi connection, as it will not work with 5 GHz connection. Same is true for using the EspTouch where the device that will pass the credentials must have $2.4GHz$ connection capabilities. Some devices have phased out support for the $2.4GHz$ band therefore Smart Config may not work with all devices.

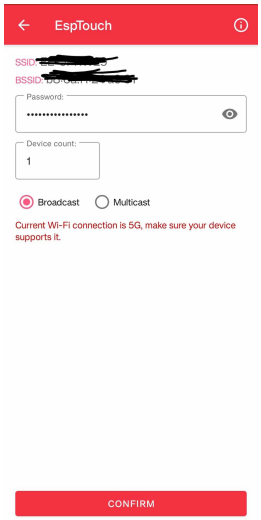



Figure 26: EspTouch Example Screenshot.

#### 6.1.2.1.3 Powering the MCC

Establishing a successful connection to WiFi signals the system that it should power up the MCC and start the HTTP server. MCC power-up occurs by simply setting the GPIO2 pin to HIGH, which allows current to flow through the p-channel MOSFET by shorting its Gate terminal to ground, and powering up the MCC. More info on this is placed in the Hardware Section 5.4. In the case where the WiFi connection is not successfully established, the WUC turns the MCC on for one minute and then goes to sleep for one day. This is to allow for some monitoring to occur in the case of network issues. The active time is still small enough to allow for lots of time for the network to be back up so that regular monitoring can happen again.

#### 6.1.2.1.4 Starting the HTTP Server

Starting the HTTP server is done by declaring a server handle variable as type `httpd_handle_t` and passing assigning to it the `start_webserver()` function call. This function starts the HTTP server and registers to it all of the URI handlers the system requires for proper communication. The URI handlers for the WUC are described in Table 5.

#### 6.1.2.1.5 Setting Up and Entering Deep Sleep Mode

The ESP32 chip series offer two main sleep modes: deep sleep and light sleep, [61] which can be used to reduce power consumption in the devices using their chip models. Multiple wake up sources can be used to exit these sleep modes, such as an internal timer, GPIO pins, and touch pads. In the project's case, deep sleep was the chosen sleep mode due to the ultra-low power consumption required to have a very long lasting battery life. The wake-up source decided on was to use an internal timer which would allow accurate wake-ups in the microseconds with no connection required to anything external to the chip itself. The `deep_sleep_setup()` function calls the correct functions to assign the deep sleep configuration based on the sleep time in the system configuration variable (given in seconds), as well as setting up the timer wake-up source. As soon as the setup is complete, the system enters deep sleep in which the only other method to wake up from other than the timer is with a system reset or a power cycle.

#### 6.1.2.2 WUC In-depth Code Description

In this section a much more detailed description of the handwritten code will be given, examining the different functions at a deeper level than the more descriptive previous sections. The previous sections were necessary to provide a high level understanding of how the system works, what functionalities it can do, and when it executes them. The following section will group the functions based on which header files they belong to, i.e. the functions in `wifi.h` will be presented sequentially. As required by the cited style guide, all header files implement pre-processor header guards such as `#pragma once` to limit multiple definitions and `extern "C"` guards to allow the header to be accessible using C++ code. It would be difficult to fully describe all of the library functions and their parameters, therefore for any further explanation please see the cited examples in Section 6.1 where most of the code was adapted from.

##### 6.1.2.2.1 main.c

The main app file for the WUC module follows the same execution as the flowchart. It initially initialises the NVS for the WiFi, and then checks the NVS for the credentials. Depending on the result from that function, the corresponding WiFi initialisation procedure takes place. If no credentials are found then Smart Config is initialised, else the ones stored in the NVS are used. Following that, if a WiFi connection is established then the HTTP server is initialised and the program enters the main server loop. In this loop the URI handlers can be called as required. In the case of no WiFi connection the system executes a default active time of one minute for the MCC and enters deep sleep for one day. In the expected case were the system connects to the server backend, the active time depends on how long it's connected to the server and the server scheduler. Deep sleep is setup according to the sleep time received by the server and then the system enters deep sleep.

##### 6.1.2.2.2 wifi.h (WUC)

In the `wifi.h` header file is where all of the macros and function definitions are for the WUC WiFi functionalities. The `TEST_CRED` macro is used for testing purposes so that the Smart Config step is skipped. The `ESP_MAXIMUM_RETRY` macro is used to declare the amount of retries to connect to the WiFi access point.

**sc_event_handler():**

Function is used to handle events created by the Smart Config event. It is registered as an event in `wifi_init_sc()`. The events it checks for have multiple names and multiple IDs and can be further viewed in the code. The most notable event is the `SC_EVENT` with event ID `SC_EVENT_GOT_SSID_PSWD` which signifies that the ESP has received credentials through the Smart Config procedure and tries to connect to WiFi using them. These credentials also get stored in the system configuration variable.

**smartconfig_task():**

In the case `sc_event_handler()` detects a `WIFI_EVENT` with WIFI_EVENT_STA_START (which is as soon as WiFi is initialised), this task gets created and is used to tell the system to wait for Smart Config to finish. It does this using the `xEventGroupWaitBits()` function [62] which returns Event Bits based on the bits inputted as an parameter. Please see the cited FreeRTOS documentation for a more detailed explanation. If the bits returned match the `ESPTOUCH_DONE_BIT`, then the task is deleted meaning it has finished.

**wifi_event_handler():**

A typical handler used to check the returned events when connecting to WiFi using a known SSID and Password. It also handles retrying connecting to the network in the case of failure by using the macro defined in `wifi.h`.

**wifi_init():**

A typical ESP32 WiFi initialisation procedure is executed in this function. It requires the stored credentials and therefore no Smart Config procedure is performed here. It copies the credentials stored in the system configuration variable, that were taken from the NVS, to the WiFi configuration variable to be used to establish a connection. A default WiFi initialisation configuration is used and the `wifi_event_handler` is used to handle establishing the connection.

**wifi_init_sc():**

Executes a very similar procedure to `wifi_init()`, however in this function the Smart Config procedure is initialised and no connection is established while in this function. Receiving the credentials and connecting to the network is handles by the `sc_event_handler()`.

#### 6.1.2.2.3 http.h (WUC)

The header file for the HTTP functions contains functionalities that have already been partially described by Table 5 below. A sample cURL command exists at the bottom of the file to show how the sleep time can be changed from the terminal. All of the handlers take a `httpd_req_t *req` parameter which is used in the handlers to check the contents of the request as well as send a response back. The response is of `const char` which means it is a string of text being sent back to the person making the request. The `GET` and `POST` handlers are declared using the standard method described by the documentation, with their URI, methods, and handlers defined in `http.c`.

Table 5: WUC URI handlers with what functions they call and their description

| URI Handler | URI | Method | Function Call | Description |
|---|---|---|---|---|
| `uri_get` | `/uri` | HTTP GET | `get_handler` | Used to test correct functionality of the HTTP server on the WUC. Sends a string as a response to the request. |
| `exit_get` | `/exit` | HTTP GET | `exit_handler` | Used to exit the main server loop by setting the system configuration variable to true (`wuc_config.exit = true`). Terminates the server and allows the program to continue to setup and enter deep sleep. |
| `reset_get` | `/reset` | HTTP GET | `reset_handler` | Used to reset the WiFi credentials stored in the system by setting the system configuration variable to true (`wuc_config.reset = true`). Deletes the current NVS contents and restarts the system. Skips the deep sleep steps. |
| `set_sleep_time_post` | `/sleep` | HTTP POST | `set_sleep_time_handler` | Used to set the duration of deep sleep in seconds by passing the number as a string in the body of the POST request. The value is set in the system configuration variable and is used in the deep sleep setup. |

**set_sleep_time_handler():**

Since the simple URI handlers are adequately described in Table 5, just this handler is described in more detail since it has a lot more functionality. This handler is designed to handle a text string of data as the content of the request. It first checks the length of the data to make sure it's not a very large string being sent by accident or maliciously. The size of the received content plus one, is used to allocate the string in the local memory which is then set by the `httpd_req_recv` function. The reason for adding one to the content size is so that the null string terminator `'\0'` can be appended at the end of the variable so that the string value is valid and logical. Using `sscanf()` the request content is then placed in the system configuration variable by passing its address and using the long unsigned integer `%lu` format specifier. Finally, a response is sent back to signify the request has been completed.

**start_webserver():**

To start the HTTP server, this function is called, which also registers the URI handlers for the server. Here is also where the server port 19520 used in the project is set. If the server initialisation is successful and not `NULL`, a server handler is returned by the function.

**stop_webserver():**

Simply used to safely terminate the http server.

#### 6.1.2.2.4 wuc_config.h

Functions that are required specifically by the WUC module are defined in this file. The system configuration variable is defined in Table 6 below, with explanations for the function of each variable. The functions `setup_mcc_power_switch()`, `mcc_powerup()`, and `mcc_powerdown()` simply setup the GPIO pin used by the WUC to turn on or off the MCC.

Table 6: WUC System Configuration Variable

| **Structure Member Name** | **Type** | **Description** |
|---|---|---|
| exit | bool | Default set to false, set to true by the `/exit` URI handler. Used to exit the main server loop and enter deep sleep. |
| reset | bool | Default set to false, set to true by the `/reset` URI handler. Used to reset the system to accept new WiFi credentials via Smart Config. |
| sleep_time_sec | uint32_t | Default set to 86400 (roughly one day, in seconds). Set by the `/sleep` URI handler. Used to store how long the system be in deep sleep for. |
| active_time_sec | uint32_t | Default set to 60 seconds. Not set by any URI handlers and is hardcoded default value. Used to store how long the WUC will have the MCC active for. |
| ssid | char[33] | Default set to an empty string. Size of 33 due to the ESP32 SSID variable size being 32, plus one to add the null terminator character to create a valid string. Variable gets set either by the Smart Config procedure or by retrieving the stored credentials from the NVS. |
| pswd | char[65] | Default set to an empty string. Size of 65 due to the ESP32 PSWD variable size being 64, plus one to add the null terminator character to create a valid string. Variable gets set either by the Smart Config procedure or by retrieving the stored credentials from the NVS. |
| stored_creds | bool | Default set to false. Variable is used to check whether credentials have been detected in the NVS or not. |

**deep_sleep_setup():**

Executing this function before entering deep sleep sets up the timer wake-up based on the value in the system configuration variable. The code used here is adapted from the standard method of executing a deep sleep timer wake-up on an ESP32. More details regarding this can be viewed by looking at documentation or the cited example.

#### 6.1.2.2.5 wuc_nvs.h

In `wuc_nvs.h`, are the functions that are used for checking and retrieving data from the Non-Volatile Storage. The standard namespace macro for the NVS is also defined in this file. In all functions the NVS is opened and assigned to a handler which is used in all NVS realted functions.

**get_nvs_sizes():**

To check the sizes of the currently occupied space by the credentials in the NVS, the `get_nvs_sizes()` function is called. It initially opens the NVS and looks for the values with the `ssid` or `pswd` keys associated to them. Keys can be thought of as the section labeled as the specific value in the system. It then retrieves the size of the values which is then checked if it is more than zero to signify if Smart Config should be used or if the data should be retrieved in the `check_nvs()` function.

**check_nvs():**

The `check_nvs()` function behaves similarly to the `get_nvs_sizes()` where it checks based on the same key strings. However, in this function as soon as the NVS is opened, it checks the sizes retrieved by the `get_nvs_sizes()` to signify whether to retrieve the stored credentials or to initialise Smart Config. In the case the received sizes are more than zero, they are passed as a parameter to use to copy that specific amount of data over to the system configuration variable. This is handled by the `nvs_get_str()` function where the key, the system configuration variable and address of the size value, is passed as parameters. If the sizes are zero it means they have not been detected in the NVS, therefore Smart Config should be intialised.

**store_creds_to_nvs():**

To store the retrieved credentials no size checks are required. The NVS is simply opened and using the acquired handle, the corresponding keys are assigned to the detected SSID or Password values and they are stored in the NVS using the `nvs_set_str()` function. Since changes were made to the NVS, the data must be committed by using `nvs_commit()` before closing.

### 6.1.3 MCC

The MCC gets powered through the $V_{cc}$ pin of the WUC, however it only gets powered on by setting GPIO2 of the WUC to high, so that the MOSFET circuit closes and current flows through to the MCC. Its main function as the name suggests is to control the cameras by switching between them and to send each image to the server backend to be processed using the stereo imaging algorithm. At each capture the MCC takes an image from one camera, then at the end of the capture it switches to the other camera, meaning for two images the system must be told to capture twice. To accomplish all of the functionalities required from the MCC, the SD card, WiFi, HTTP and Camera components were used in the MCC project. The terminalogy "frame buffer", "picture" or "image" used in this section mean the same thing and are used interchangeably.

#### 6.1.3.1 Flowchart Overview

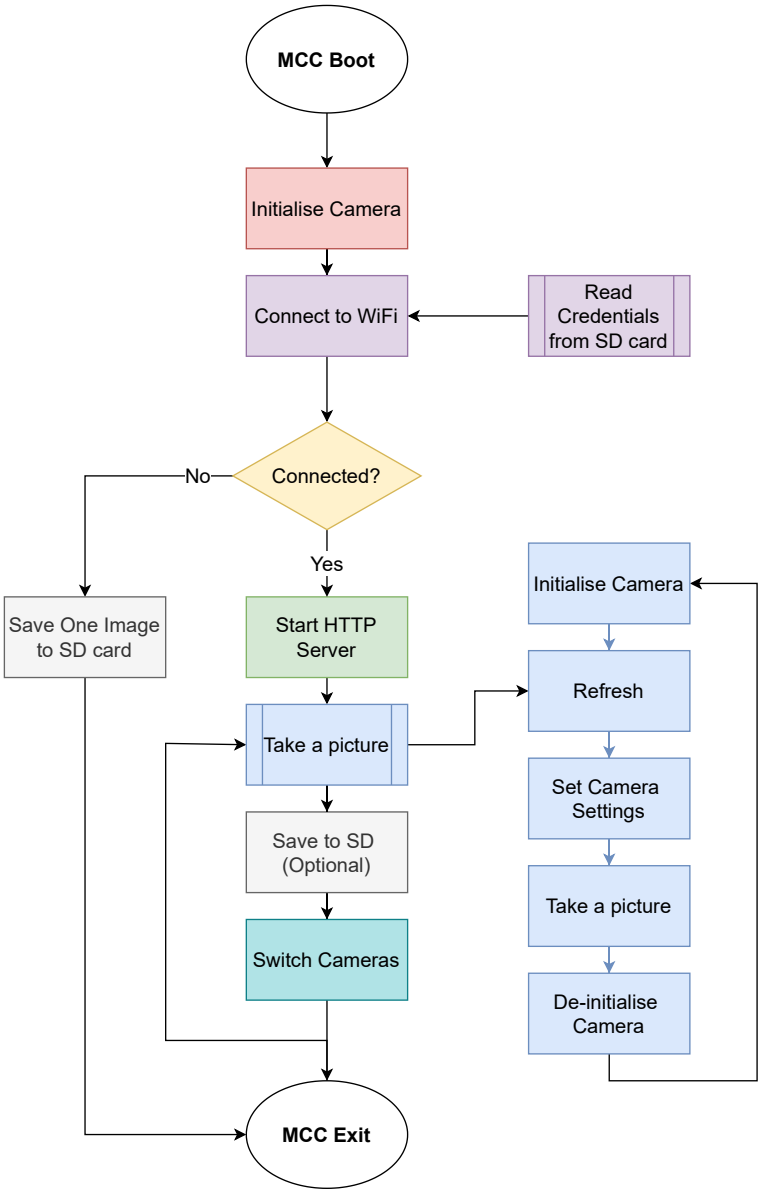



Figure 27: Flowchart for the MCC Functionalities

Figure 27 visualises the functionalities and the program structure of the MCC. First and foremost, one of the cameras is initialised. The reason for this being the first thing is that if the initialisation is unsuccessful, the MCC is essentially useless, and therefore the program exits if initialisation fails. After initialisation, the WiFi credentials are read from the SD card and are used to connect to the WiFi. The format of the WiFi credentials file will be discussed in Section 6.1.3.2.5. If no WiFi connection is established, the system only saves one image to the SD card so as to allow for some partial monitoring to occur. In the normal case were the WiFi connection is successfully established, the MCC starts its HTTP server to handle the incoming requests from the project server backend.

Once the backend requests a picture, a certain process happens to make sure the picture is taken correctly. Due to certain hardware limitations by the OV2640, the image had to be refreshed twice (can be increased but decreasing is not recommended) to remove an obvious green tint to the image. This issue has been documented [63] by many other users online, and refreshing was found by the group as an easy solution to this problem. The camera settings also get set before each photo, due to the de-initialisation step required to safely switch between cameras. Once the picture is finally taken and sent to the server, the camera gets de-initialised and turned off so that safe switching between them happens. Optionally the most recently taken picture can be saved to the SD card. Lastly, the image switches from one camera to the other and initialisation occurs again so that the system is ready for another image capture. Therefore the image taking process must happen twice to receive the two images required for stereo vision. All described processes that are executed under a successful WiFi connection can happen either using the the manual photo capture or the scheduler from the website.

#### 6.1.3.1.1 Initialise Camera

Camera initialisation happens through calling the `init_camera()` function. Error checking occurs to make sure a camera is detected, and if not the program simply exits since the MCC is of no use if no camera can be used. Internally, the camera driver function `esp_camera_init()` is called with the address of the camera configuration passed as a parameter. The camera config will be further explained in 6.1.3.2.4.

#### 6.1.3.1.2 Connecting to WiFi

Connecting to the WiFi is done by reading the credentials from the SD card slot provided by the ESP32-CAM. At this step therefore the SD card is initialised alongside SPI for the credentials to be read from a text file named `wificreds.txt` located in the root directory of the SD card. The required format of this file is described in Section 6.1.3.2.5. The read SSID and Password are then used to connect to the WiFi access point. An important note is that the access point or router must support the 2.4 GHz band, or else the connection will not be established due to the ESP32 chips only supporting that specific band.

#### 6.1.3.1.3 Saving One Image to SD Card

In the case where an WiFi connection is not successfully established, the system takes only one picture from one of the cameras and saves it to the SD card. This implementation is due to the MCC being useless if no WiFi connection is established, and in because in remote locations that may be the case in some scenarios.

#### 6.1.3.1.4 Start HTTP Server

Similarly to the WUC (Section 6.1.2.2.3), an HTTP server is started using `start_webserver()` that uses URI handlers to control certain MCC functionalities. The URI handlers used by the MCC are listed in Table 6.1.3.2.3.

#### 6.1.3.1.5 Take a Picture

Taking a picture with the MCC is a process that requires vital synchronisation between components to make sure nothing goes wrong. Server-side and software delays are implemented to make sure the switching is executed smoothly. The `\pic` URI handler is used to take one picture. Once the request is received, the image is refreshed, meaning a picture is taken successively two times due to the camera adding a green tint to the first image taken after initialisation. Taking a second picture provides reassurance that the camera's sensors are stabilised and that a neutral image with no tint is captured. After this refresh, the image that will be sent over HTTP to the server is captured and is sent.

Two methods are implemented to send an image over HTTP. The reason for two methods is due to the image format and the corresponding size of the image. The system checks if the captured image is formatted as a JPEG and if so it is sent as is over HTTP; if the image is not a JPEG image then it is converted to JPEG and sent in chunks over HTTP. The latter method is used by default in the system, but if SD saving is enabled, the image format is converted to JPEG due to the great reduction in file size, and therefore reducing MCC active time. The captured image before JPEG conversion is YUV422 [64], and it is used because of issues with the changing the camera settings when using the JPEG format. The camera would seemingly ignore the set settings after capture when taking a JPEG photo [ ,comments report similar issues]. Finally, the camera is de-initialised and powered down using the `GPIO32` which is the `CAM_PIN_PWDN` GPIO pin. This is done to make sure switching is done safely and doesn't corrupt or disrupt any of the OV2640's functionalities.

#### 6.1.3.1.6 Optionally Saving to the SD card

As mentioned before, to save to the SD card the image is captured as a JPEG to reduce file size and therefore the total system active time. Writing to the SD card is accomplished by opening a file and writing to it in binary mode. This mode writes the data in a binary format, meaning it doesn't try to write characters and uses the data as is. More details regarding this are given in the Section 6.1.3.2.5.

#### 6.1.3.1.7 Switch Cameras

The switching of cameras happens after taking the image and after de-initialisation of the cameras. Switching between the cameras is done by setting the MCC's `GPIO4` pin to high or low. A delay is done server-side as well as on the microcontroller to ensure correct initialisation is performed after switching. Exiting the main server loop can be done by the `\exit` URI handler and the program exits gracefully by de-initialising every initialised peripheral. The downside to using GPIO4 for this purpose is that it is hardwired internally to turn on a white LED to be used for camera flash. This drawback was unavoidable since the remaining ESP32-CAM GPIOs were unusable due to them being internally wired to other processes like the SPI for the SD card.

#### 6.1.3.2 MCC In-depth Code Description

Very similarly to the previous in-depth section (Section 6.1.2.2) for the WUC, a much more detailed description of the code will be given, by examining the different functions at a deeper level than the previous sections. Organisation of functions and file structure were done in the exact same way, meaning the functions for the WiFi are declared in `wifi.h` with their definitions in `wifi.c`, same goes for the SD card functions, etc. The same style guide was also used for both systems. For any further explanation of the code used, please see the cited examples in Section 6.1 where most of the code was adapted from. Global variables and global pointers are extensively used in the project instead of pointers for no other particular reason other than the fact it was easier to implement and less need to worry about correct addressing or de-referencing. The `extern` [65] is used to declare the variable but to also tell the compiler that this variable is used in the file and defined elsewhere in the program. The main global variables used throughout the program are the frame buffer variable name `fb` and the system configuration variable `mcc_config`.

##### 6.1.3.2.1 main.c

The main app file for the MCC is structured similarly to the flowchart and the WUC main app file. Initially the system configuration file is declared with the default settings, which will be explained later in Section 6.1.3.2.4. The first function execution is the camera initialisation using the `init_camera()` function call. NVS must also be initialised for correct WiFi functionality and the select pin (or `SEL_PIN`) is subsequently setup as an output GPIO at GPIO4 at the initial state of `LOW`. Once the SD card is initialised with `sd_init()`, the WiFi credentials are read from it by calling `read_wifi_credentials()`. Since SD saving is not hardcoded to happen by default, the SD card is de-initialised and can be initialised later through a changing the setting using the `\config` URI handler (see Section 6.1.3.2.5). Establishing a WiFi connection is done by calling the general WiFi initialisation function `wifi_init_general()`, followed by initialising the MCC in WiFi station mode (STA) using `wifi_init_sta()`. Upon successful connection, the HTTP server is started, which is used the same way as the WUC, in the sense that URI handlers are used to call certain functions in the software (see Table 6.1.3.2.3). While in the main server loop, these URI handlers can be called and can either change the system camera settings or change variables to allow for the if-statements in the main loop to be executed. In the case of no WiFi connection then the HTTP server is not started and the system takes only one image instead of two and saves it to the SD card. A graceful exiting of the main app is then implemented by returning the camera frame buffer, de-initialising the camera as well as the SD card.

##### 6.1.3.2.2 camera.h

Written in this header file are the camera pin definitions, all of them being fixed due to the ESP32-CAM model having them all wired internally to its FPC connector. The `REFRESH_NUM` macro is also defined here which is used to set the amount of times the frame buffer is refreshed. The functions for the flash LED (GPIO4) are not used in the program but the functionality exists for future implementations.

**pic_data_output():**

Used to inspect the image data upon image capture. By using the ESP logging functions, the image data is outputted to the terminal. Examples of the output are shown in Figure 28

**init_camera():**

To initialise the cameras, the function `init_camera()` is called. It uses the camera driver function `esp_camera_init()` which handles the initialisation alongside using I2C to get the device address. Error checking also occurs within the function.

**change_pixformat_to_jpeg():**

As mentioned before, when saving to the SD card, the image format must be JPEG to greatly reduce file size and therefore the time the system takes to save the image. Using this function forces the camera to give a JPEG image, therefore disregarding the initial default configuration.

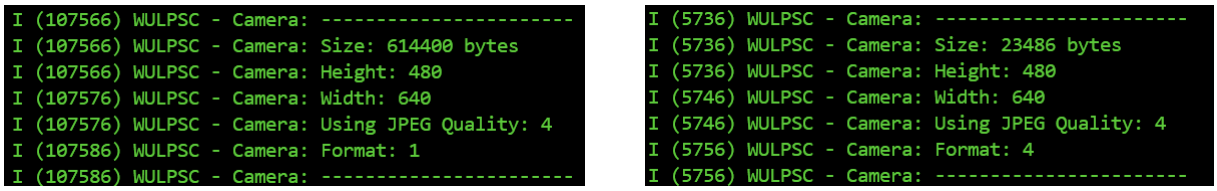



Figure 28: Terminal output to show the potential size of the image in YUV422 (Left, 614400 bytes) and JPEG (Right, 23486 bytes) formats.

**fb_refresh():**

This function is used to refresh the image so that the image received is more balanced and without the aforementioned green tint (as shown in Figure 29). It uses a simple for-loop which captures the frame buffer using the `esp_camera_fb_get()` camera driver function, and the returns this frame buffer using `esp_camera_fb_return()`. The frame buffer must be returned in this way so that it can be used again, and in the loop it is also set to `NULL` to make sure it is empty before taking another image.

Figure 29: Taking an image with no refresh (Left) and with refresh (Right).

#### 6.1.3.2.3 http.h (MCC)

In the exact same manner as the WUC, `http.h` contains all of the function declarations and documentation for the HTTP server functionality. URI handler in Table xxxx below are used to tell the system to execute certain functionalities. A sample cURL command is commented at the bottom of the header file to show how the settings can be changed via the terminal using one of the handlers. As mentioned in the WUC HTTP section, a `httpd_req_t *req` parameter which is used in the handlers to check the contents of the request as well as send a string response back. `GET` and `POST` handlers are declared using the standard method with their source code being in the corresponding `http.c` file. The `start_webserver()` and `stop_webserver()` functions have the exact same purpose as the ones used in the WUC explained in Section 6.1.2.2.3.

Table 7: MCC URI handlers with what functions they call and their description

| URI Handler | URI | Metho | Function Call | Description |
|---|---|---|---|---|
| `uri_get` | `/uri` | HTTP GET | `get_handler` | Used to test correct functionality of the HTTP server on the MCC. Sends a string as a response to the request. |
| `exit_get` | `/exit` | HTTP GET | `exit_handler` | Used to exit the main server loop and exit main by setting the system configuration variable to true (`mcc_config.exit = true`). |
| `pic_get` | `/pic` | HTTP GET | `picture_handler` | Used to refresh the frame buffer, take the actual image that will be sent, and send over the network using HTTP. If the image is not captured as a JPEG, it is sent in chunks. |
| `config _settings_post` | `/config` | HTTP POST | `config_settings_ post_handler` | Receives a JSON object that is then spplit up into the multiple system parameters and placed in the global system variable for system-wide use. |

**jpg_encode_stream():**

Function was provided by the ESP Camera API and is used to convert to JPEG before transmitting it over HTTP in chunks. Limited documentation is provided for the API and why this function must be written in this way. Please see cited examples for more information.

**picture_handler():**

Executed when the `/pic` URI handler is called. As stated in Table 6.1.3.2.3, it initially refreshes the frame buffer and then takes the final image which will be then transmitted. The frame buffer is checked if is not `NULL` so that no empty frame buffer is sent. The response header type is then set to `'image/jpeg'` and if that happens correctly, the header is populated with more of the required HTML data such as the filed name and the image file name. Immediately after, the image is checked for its format, and if it is JPEG it is send as a single data stream using `httpd_resp_send()` with the size of the image and the contents set as parameters. In the case where the image is not of the JPEG format, it is converted into JPEG using `frame2jpg_cb()` and sent in chunks using the `httpd_resp_send_chunk()` function call. The `.pic_done` system configuration variable structure member is set to true so that the system enters the camera switching procedure in the main loop.

**config_settings_post_handler():**

Using this `POST` handler, the system settings can be altered depending on the user specifications. The exact same procedure as in the WUC `POST` handler (see Sectoin 6.1.2.2.3) is used, where the size of the received content is checked and then that size, plus one for the null terminator character `'\0'`, is allocated in memory the `content` character array. This character array is then used as a string that gets passed to the `JSON_config_set()` function that separates the JSON key/value pairs and assigns them to the system configuration variable. The sensor information is then updated using `esp_camera_sensor_get()` and the new camera settings are applied using the `camera_set_settings()` function and passing it the system configuration variable. A response string is sent back to signify correct data transfer.

#### 6.1.3.2.4 mcc_config.h

In this header file, the SD and system configuration data types are defined, detailed in Tables 10 and 8 respectively. The multiplexer select pin macro is also defined here to be GPIO4, which is used to select which camera to use to take a picture. Functions specific to the MCC system are declared in this file and defined in `mcc_config.c`.

**camera_set_settings():**

To set all of the camera settings stored in the system configuration variable, it is passed to the `camera_set_settings()` function call. All of the settings supported by the OV2640 are set using the values of the system config and the ESP Camera API functions. The sensor data stored in `.sensor` of system config is accessed via the arrow operator, and by using its function members the values stored in the system config are applied to the camera sensor. Explanations for each function, its corresponding values and the default values are provided in Table 9.

**JSON_config_set():**

Using JSON objects in C is easily implemented by using cJSON, which is already implemented as a component in ESP-IDF [66]. By including `cJSON.h`, and by passing it a the function a valid JSON object, it can be examined using the provided functions. Initially the input is parsed to a JSON ob-

Table 8: MCC System Configuration Variable

| Structure Member Name | Type | Description |
|---|---|---|
| sd_save | sd_save_t | Default set to `NO_SD_SAVE` (or 0). Type is a `typedef enum` explained in Table 10. |
| flash | bool | Default set to false. Used to set whether the system should use a flash for taking pictures. Unused in the codebase. |
| exit | bool | Default set to false, set to true by the `/exit` URI handler and used to exit the MCC gracefully. |
| pic_done | bool | Default set to false, as it is set true once the `/pic` URI handler finishes. Used to check when an image has been sent. |
| sel_status | bool | Default is set to 0 or false. Used to check the current status of the multiplexer select pin. |
| sensor | *sensor_t | Default is `NULL`. Set only using `esp_camera_sensor_get()`, usually done before setting the settings each time. |
| camera | camera _status _t | Default settings explained under `camera_set_settings` in Table 9. Stores all of the supported camera settings and other values in the `camera_status_t` structure. |

ject pointer called `root`, and by using the `cJSON_GetObjectItem()` function, the corresponding key can be located in the object. Using multiple if-statements allows the inspection of each required key, and if the specific key is found, using the arrow operator its integer value can be extracted by using `cJSON_GetObjectItem(object,"key")->valueint`.

**sys_take_picture():**

Capturing an image is executed by calling this function. Before each picture the current sensor data is acquired by using the `esp_camera_sensor_get())` function, and then the currently stored settings are applied to the camera sensor. A check is executed to check if the camera flash is set but as mentioned before it is not implemented.

**sys_sd_var_setup():**

In the main server loop, the system checks the state of the `sd_save` system config struct member at each iteration. In the case the user selects the SD card option through the frontend, `sd_save` changes from `NO_SD_SAVE` to `SD_SETUP` and `sys_sd_var_setup()` is executed. Table 10, shows the states the `sd_save` member can take and why those values are required. The setup required for the SD card is to change the image format to JPEG, and to create a random number that will be used to name the left and right images. The random number is acquired by executing `esp_fill_random()` and passing the address of a variable to store the value along with its size. The value was then be made into a string using `sprintf()` and `"%u"` to pass the unsigned integer into a string variable.

Table 9: Functions and values used for the camera picture settings. Default values were values found experimentally for optimal room light lever conditions.

| **System Config Struct Member** | **Function** | **Min. Value** | **Max. Value** | **Default Value** | **Description** |
|---|---|---|---|---|---|
| brightness | set_brightness() | -2 | 2 | 0 | Set the brightness of the captured images. |
| contrast | set_contrast() | -2 | 2 | 0 | Set the contrast of the captured images. |
| saturation | set_saturation() | -2 | 2 | 0 | Set the saturation of the captured images. |
| special_effect | set_special_effect | 0 | 6 | 0 | Sets a special effect to the image. 0 - Normal, 1 - Inverted, 2 - Grayscale, 3 - Red Tint, 4 - Green Tint, 5 - Blue Tint, 6 - Yellow Tint. |
| ae_level | set_ae_level() | -2 | 2 | 0 | Set the auto-exposure level of the camera. |
| aec_value | set_ae_level() | 0 | 1200 | 20 | Set the auto-exposure control value of the camera. |
| agc_gain | set_ae_gain() | 0 | 30 | 2 | Set the auto-gain control of the camera. |
| gainceiling | set_gainceiling() | 0 | 6 | 1 | Set the gainceiling of the camera. |
| lenc | set_lenc() | 0 | 1 | 1 | Enable/Disable lense correction |
| agc | set_gain_ctrl() | 0 | 1 | 0 | Enable/Disable automatic gain control |
| aec | set_exposure_ctrl() | 0 | 1 | 0 | Enable/Disable automatic exposure control |
| hmirror | set_hmirror() | 0 | 1 | 0 | Enable/Disable mirroring the image horizontally |
| vflip | set_vlip() | 0 | 1 | 0 | Enable/Disable mirroring the image horizontally |
| aec2 | set_aec2() | 0 | 1 | 1 | Enable/Disable mirroring the image vertically |
| bpc | set_bpc() | 0 | 1 | 1 | Enable/Disable black pixel correction |
| wpc | set_wpc() | 0 | 1 | 1 | Enable/Disable white pixel correction |

Table 10: Description of SD Card Custom Enumeration Type

| Name | Value | Description |
|---|---|---|
| NO_SD_SAVE | 0 | Used to signify no picture saving will take place. |
| SD_SETUP | 1 | Checked in the main server loop and used to execute `sys_sd_var_setup()` to make the SD card ready for saving. |
| SD_SAVE | 2 | Only case it reaches this value is after SD setup is completed. Used to signify that the SD is ready for an image to be saved. |

**sys_sd_save():**

Calling this function only takes place in the case where the `sd_save` struct member is `SD_SAVE`, which can only get set after `sys_sd_var_setup()` is executed in main server loop. It starts by copying the mount point of the SD card, then concatenating using `strcat()` the random number string, the image file extension, as well as whether the image saved is the left or right. To check this, the status of the multiplexer select pin is used, and this allows for synchronisation and consistency between the file names. Finally, the image is saved using the `sd_write_arr()` function call, which is explained is 6.1.3.2.5.

**sys_camera_switch():**

Switching between the two cameras can be considered the most important procedure in the project software design. It allows for gracefully changing between the cameras, reducing risk of sensor damage or data corruption. Initially, the camera is de-initialised using the `esp_camera_deinit()` function provided by the ESP Camera API, then the `CAM_PIN_PWDN` pin is used to to power down the camera through its hardware power down feature. An `XOR` gate is used to change the status of the select pin by `XOR`'ing a HEX value of `0x1` at each function execution. By doing this Table 11 entries two and four will happen each time, meaning it will cycle from a zero to a one and vice versa. Since the value changed is a global variable, its value is saved outside the function and can therefore be used over as iterations as needed.

| A | B | XOR Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 11: XOR Gate Outputs based on the Inputs A and B

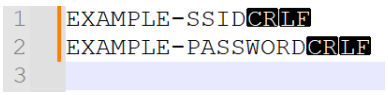



Figure 30: Using Notepad++ to display the CR and LF escape characters.

#### 6.1.3.2.5 sd.h

SPI is used for the SD initialisation, which is one of the two standard methods of initialising the SD card slot. Most of the code used for this functionality of the code is based on the SD SPI example, and adapted for the project's purposes. All of the macro definitions are required by the examples and are used to define the GPIO pins used as well as the mount point for the SD card. The filename for the WiFi credentials `wificreds.txt` is also defined as a macro by concatinating the filename with the mount point macro definition.

**sd_init():**

The initialisation code for the SD card slot is based on the cited SD card SPI example. It initially creates the file system configuration using FAT VFS [67], and initialises the SPI bus. The Sd card slot is then initialised is then mounted via SPI and the card infor is outputted to the terminal.

**sd_write_arr():**

Using the path provided by `sys_sd_save()`, the file at that path in the SD card is open in write binary mode or `"wb"` using the `fopen()` function. In this mode data is written to the file as binary data and not characters, and by using this mode the frame buffer's data can be written to a file, thus saving the image. Each entry in the frame buffer array is treated as a character or one byte in terms of size.

**sd_deinit():**

To de-initialise the SD card is it simply unmounted using the `esp_vfs_fat_sdcard_unmount()` and and the SPI bus is freed using `spi_bus_free(host.slot)`.

**read_wifi_credentials():**

To connect to the WiFi credentials must be read from the `"wificreds.txt"` from the SD card. The file must be in the form shown in 30, where the SSID and Password are in two separate lines. The two global variables to store the credentials are initialised with null terminator strings to make them fully empty until a string is copied to their memory locations. Using the function `fgets()` both lines are read and the Carriage Return (CR) `\r` characters along with the Line Feed (LF) characters `\n` are removed.

An interesting case is created when using different operating system to edit the text file. Windows systems use `CRLF` at the end of lines whereas Unix-based systems like Mac OS and Linux use simply `LF` [68]. For this reason, the characters are removed twice each using the `strchr()` function, since both characters can appear a maximum of four times based on the format (30) of the credentials text file. The `strchr()` function returns the position of the requested characters as a pointer, and then by de-referencing the points, the value where it points to can be replaced. Therefore, the positions where `CR` or `LF` occur are replaced by the null terminator character (`'\0'`).

#### 6.1.3.2.6 wifi.h (MCC)

Again similarly to the WUC, a `wifi.h` file is used to hold the function declarations and macro definitions required by the WiFi connection establishment procedure. Initially, an implementation of WiFi Scanning was used to detect for nearby access points, and the general initialisation function is a remnant of an old implementation. It could very well be copy-pasted into the main WiFi station mode initialisation function, but there was no need since both functions can be called sequentially, and the WiFi would be correctly initialised.

**wifi_init_general():**

Contains most of the standard WiFi function calls required to setup the WiFi component of an ESP32.

**event_handler():**

Handles the WiFi events that are created when using the Wifi on the system. It checks if a successful connection was established and if not then it retries as many times as the value of the `ESP_MAXIMUM_RETRY` macro is defined to (three in this case).

**wifi_init_sta():**

Initialises WiFi in station mode, which is the mode required to connect to a router or an access point. The event group is initially created and the `event_handler()` function is assigned to be executed at any detected WiFi events. The credentials used are the ones read from the SD card but when testing, setting the `SD_CRED` macro to zero is recommended, since it allows to use hardcoded credentials. At successful connection the event bits are set to be the `WIFI_CONNECTED_BIT` and the function returns `ESP_OK` to signify correct execution.

## 6.2 Back-end

### 6.2.1 Motivation

The microcontroller needs to send the images to a remote computer for the stereo image algorithm so that the image processing is all done elsewhere to reduce power consumption on the microcontrollers. This could be achieved in many ways, but in this project the remote computer should be in the cloud so that the images can be accessed anywhere, rather than having the remote computer on the same network as the microcontroller.

### 6.2.2 Overview

The back-end also connects to a cloud database to store past images. The web frontend connects to the back-end and can request images as well as change parameters and force the micro controller to take a picture. The back-end can also analyse an object given its bounding box. The back-end also has a schedule for taking images which can be set by the frontend. There are lots of languages and frameworks for creating back-ends, as OpenCV will be used for the stereo image algorithm, it makes sense to use Python as the back-end language since OpenCV will be used as a Python module. For the back-end framework, the most popular are Django, Flask, and FastAPI. In the end, the latter was chosen as it had the easiest syntax and made most since for this use case, since only HTTP API endpoints were needed and FastAPI is great for this.

### 6.2.3 Host Provider and Database

Next the back-end also needs to be hosted on a computer, fly.io a cloud provider was chosen which lets you host a lot a variety of apps. It has a free tier which should suffice for this case as it allows 160 GB outbound data per month [69], with a size of 100 kB per picture and 10 pictures a day would amount to only 30 MB per month.

MongoDB is chosen for the database, it's easy to create and link with the back-end. It's also hosted on Atlas which is a cloud database provider with a generous free tier allowing 512 MB of storage [70], once again with a size of 100 kB it can host over 5000 images which is more than enough.

### 6.2.4 Endpoints

The back-end as described earlier will have HTTP API end points, which are ways of telling the back-end to do something via its URL. For example, if the back-end was hosted on the local network, you could have `localhost:8000/get_latest_photo`, this URL would return the latest image created from the stereo image algorithm, this end point is known as a GET request. The endpoints can be seen in Figure 31. Starting from the top

**6.2.4.1 /parameters** Returns the parameters currently set as JSON, including the schedule. This is used by the frontend to show the parameters to the user. Figure 32 shows all the current parameters, the schedule parameter is a bit different since it holds a list of strings such as `"09:34"`, `"10:49"`.

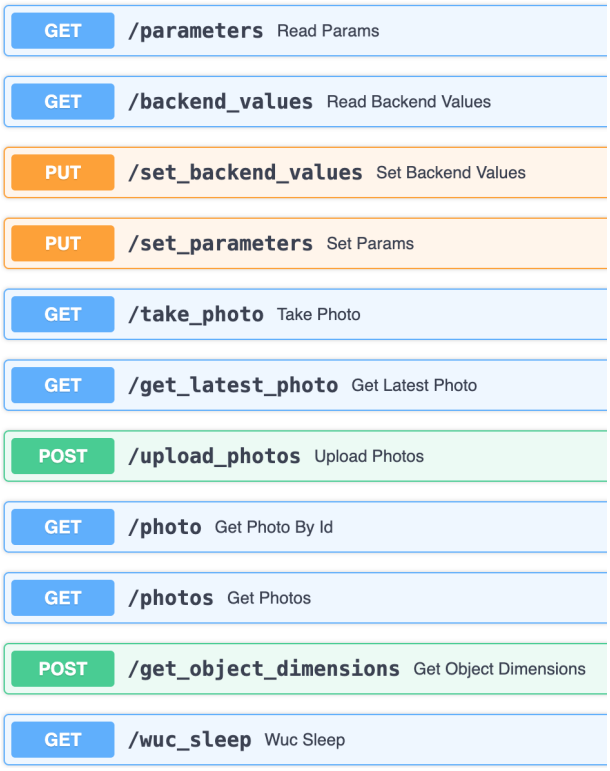



Figure 31: FastAPI Endpoints

All the parameters up to `wpc` are ESP settings. `sleep` is the number of seconds the WUC should sleep for once the `/exit_endpoint` on the WUC is called. `sd_save` tells whether to save the images captured in the MCC on to a sd card, `low_light` will first take two pictures at the start and discard them before taking the main two pictures when capturing. This is to fix any issues with the photo quality which is needed when in low light conditions. Finally, `auto_sleep` changes whether the WUC should sleep when a scheduled image is captured.


```
"brightness": 0,
"saturation": 0,
"contrast": 0,
"special_effect": 0,
"wb_mode": 0,
"ae_level": 0,
"aec_value": 0,
"agc_gain": 0,
"gainceiling": 0,
"lenc": true,
"agc": true,
"aec": true,
"hmirror": true,
"vflip": true,
"aec2": true,
"bpc": true,
"wpc": true,
"sleep": 0,
"sd_save": true,
"low_light": true,
"auto_sleep": true,
"schedule": [
  "string"
]
```


Figure 32: back-end Parameters

**6.2.4.2 /set_parameters** is used to set the parameters by passing in JSON like {`brightness: 100, ....`} in the request body. This is used by the frontend to set the parameters. The parameters are also saved to the database so that on startup the back-end can retrieve the last selected parameters. The microcontroller parameters are also set after using this endpoint.

**6.2.4.3 /wuc_sleep** is used to manually set the WUC to go to sleep with the given sleep seconds set in `set_paramaters`. Normally the back-end will automatically tell the WUC to go to sleep with the scheduled captures.

**6.2.4.4 /back-end_values** is used to get values for the back-end that are not set in the microcontroller currently it only holds the `next_wakeup` timestamp which is when the microcontroller is scheduled to next wake up after it went to sleep.

**6.2.4.5 /take_photo** is used to capture the images from the microcontroller. The microcontroller also has an endpoint `/cam1` and `/cam2` that is called to capture and retrieve the images from the microcontroller and once both images have been sent to the back-end, it then creates the stereo fused image and stores it in the database.

The endpoints for the MCC can be seen in Table 6.1.3.2.3

**6.2.4.6 /upload_photos** This is a test end point used to manually choose what images to upload to combine and fuse without having to get the pictures from the microcontroller.

**6.2.4.7 /photo** A photo ID is passed into this endpoint to retrieve that image, and this is used by the frontend in the detailed photo page to show the left, right and stereo images. It returns the left, right, stereo image and the timestamp of when the photo was taken.

**6.2.4.8 /photos** Returns a list of photos by passing in the offset and limit. This is used by the gallery page of the frontend to display a list of photos.

**6.2.4.9 /get_object_dimensions** Analyses the object given the bounding box coordinates (x1,y1,x2,y2) and the photo ID to analyse. Returns width, height and other details as shown in Figure 33 in cm.


```
{
  "distance": 0,
  "distance_max": 0,
  "distance_min": 0,
  "distance_diff": 0,
  "width": 0,
  "height": 0,
  "length": 0,
  "disparity_diff": 0
}
```


Figure 33: Object analysis data

Figure 34 shows the architecture of the whole system, the back-end is the central component that the database, microcontrollers, and frontend communicate with. One important thing to note is that the WUC needs to be on to access the MCC.

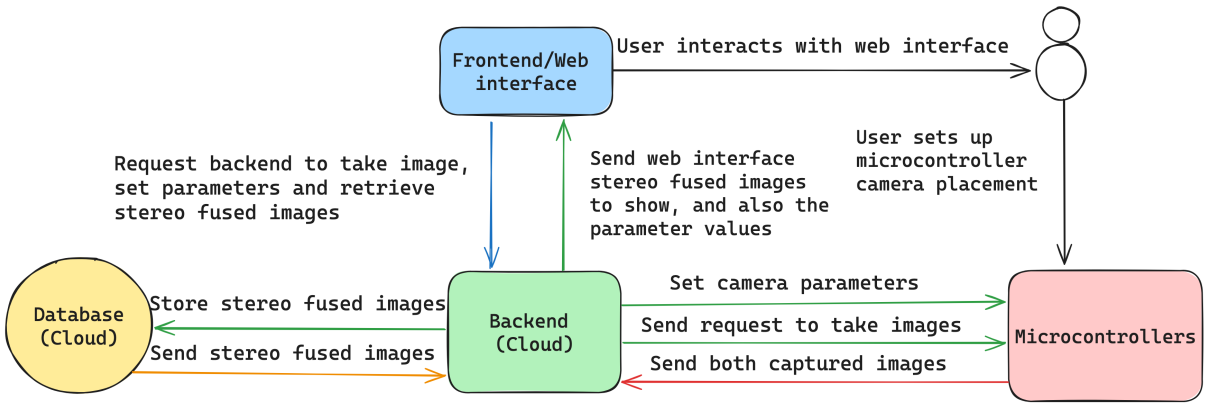



Figure 34: back-end Architecture Diagram

### 6.2.5 Image Processing

All the image processing code is described in Section 7 and is imported into the back-end to be used. A few adjustments had to be made to align the functions so that they could be used in the back-end but most of them are the same.

#### 6.2.5.1 MATLAB Rectification

However, one part that is different is the rectification of the images. This was originally done in MATLAB but to keep it all in Python the MATLAB Engine API For Python was used to call MATLAB functions from python, this worked and is how the stereo rectification is achieved. The engine can be seen here [71]

#### 6.2.5.2 Analysing Optimisation

Originally the `get_object_dimensions` function retrieved the two images from the database, rectified them and fused them together to create the disparity map which is used for the distance calculations. However, alot of extra time was spent having to re-rectify the images. To remedy this problem the code was changed so that when the images are captured, the rectified images are also stored in the database, this way the rectifying step can be skipped when analysing object dimensions. The improvements can be seen in the Table 12 below by benchmarking the time the analyzing function took to complete.

Table 12: Analysing Function Benchmarks.

| Without rectify step (s) | With rectify step (s) |
|---|---|
| 0.65 | 2.00 |
| 0.60 | 1.64 |
| 0.54 | 2.00 |

### 6.2.6 Scheduling

It is also possible to set a schedule for the back-end, so that images are taken at set times in the day. This automatically calls the microcontroller to capture the images and fuses them. To further lower power consumption the WUC sleeps until the next scheduled time arrives, for example if the schedule is 2pm and 3pm, when the time is 2pm the images are captured and then the back-end calls the `/sleep` endpoint of the WUC (seen in and passes in 3600 seconds (about 1 hour) to sleep for and `/exit` to shut down the WUC for the set amount of seconds.

The WUC endpoints that can be called are seen in Table 5. Then by the time 3pm comes around the WUC will have woken up and by proxy the MCC will be ready to capture more images. This way the time the WUC/MCC are on is minimized for low power consumption. This can be turned off with the `auto_sleep` parameter which dictates whether the WUC should go to sleep after a scheduled capture. Once the WUC goes to sleep from either a scheduled capture or from the `/wuc_sleep` endpoint, a timestamp is set called "next_wakeup" which can be retrieved to see when the WUC will next wakeup.

## 6.3 Front-end

To communicate with the back-end, and by proxy the hardware, a front-end is essential. A front-end should contain a variety of features and inputs to allow the user to communicate and access all the functionality provided by the microcontrollers, intuitively and easily. Hence, design requirements were constructed, and were prioritised accordingly using the MoSCoW scheme [72]. They can be seen below in Table 13.

Table 13: Prioritised Design Requirements Using The MoSCoW Scheme.

| ID | MoSCoW | Description |
|---|---|---|
| 1 | Must Have | User can change a range of camera and back-end parameters |
| 2 | Must Have | User can request a photo to be taken on-demand |
| 3 | Must Have | User can view previously captured photos |
| 4 | Must Have | Users can analyse a photo to obtain information about defects |
| 5 | Should Have | User can monitor the state of the system (awake, inactive) |
| 6 | Must Have | User can view the stereo depth map of a photo |
| 7 | Must Have | User can add scheduled times for captures |
| 8 | Must Have | User can manually set the WUC to sleep |
| 9 | Can Have | User can view the remaining battery life |

### 6.3.1 Design

#### 6.3.1.1 Design Requirement Analysis

Starting with requirement one, it's expected that there should be functionality present to manipulate the various camera parameters from the MCC. Some variables include the saturation, brightness, and contrast of the photo. Additionally, back-end parameters are accessible through the front-end, including settings such as the ability to turn off automatic sleep, and sleep time, both of which are described in Section 6.2.6.

Sliders are the appropriate visual design element of choice for interaction with half of these values, as they are real numbers. Switches comprise the remaining design elements for controlling boolean parameters. Additionally, the user must be able to request a photo. While the scheduled image capturing will take place, there is functionality to allow the user to manually request a photo ad hoc, if need be, to assess the current environment. Therefore, a button is adequate to provide this function. Also present, is an 'apply' button to allow the user to apply the on-screen parameter changes to the MCC.

The user should also be able to view previously captured photos. This includes both the original photo saved to the back-end database and the output depth map. The photo should also display the capture time. A gallery page is implemented to allow viewing of the various photos. Additionally, it would be useful to be able to view the battery life of the microcontroller from the interface. However, this requirement could not be carried out during the scope of the project and could serve as an objective

for future work. While not essential, this is a requirement which would prove helpful. Furthermore, relating to the status of the MCC, as per requirement 6 it would be helpful for the user to observe the current 'state' of the MCC, whether it is active or asleep (as determined by the WUC). This would ensure that capturing is indeed taking place, evidenced by a change in the state, and would prove useful for debugging purposes too.

For the design process, low-fidelity prototypes were created to establish the general navigation flow and layout of the various pages/elements. Then, a detailed high-fidelity prototype was created in Figma, which added colours, fonts, and shapes at a level of detail very similar to the intended final implementation. The prototype is also interactable, allowing the user to click through the various pages and use the functionality.

#### 6.3.1.2 High Fidelity Prototype Sketches

A preliminary layout for the web interface was established as part of the design phase and can be seen in the interim report [28].

### 6.3.2 Implementation

For the implementation, React is the framework of choice. React allows for rapid prototyping and requires little effort to get started with a design. JavaScript is used as the underlying language behind React. All code can be referred to in the additional data upload for the submission. A component-based approach is used to design and implement pages, with these components being reusable across the pages. It is used to create a 'single page' application, which means any time a new page is loaded, the current web page will be rewritten with the new information, as opposed to the conventional approach of loading an entire new page. For the level of complexity posed by the design requirements, this approach is effective and allows for a range of benefits such as; optimisation, easier debugging, better performance, and less complex implementation [73]. To interface with the back-end, endpoints are used as described in Section **??**, linking functionality from the front-end to API calls to endpoints exposed by the back-end.

#### 6.3.2.1 Latest Photo Component

As seen in Figure 35, the 'last photo' component showcases the latest photo captured by the microcontroller. The final implementation differs from the design, as it shows the rectified photo instead of the depth map. This is required, as it makes analysing the photo for defects easier, and more details can be discerned from the photo. If needed, the depth map can still be viewed via the 'previous' and 'next' buttons shown in Figure 35.

With the addition of the analyse button, the user can obtain dimension information on an area of choice via mouse input. Figure 36 shows this process, a bounding box is drawn over the 'defect' with two clicks, enabling the analyse button. A 5x5 grid is created, to represent the process the algorithm undergoes when analysing defects across high disparities, as mentioned in Section 7.3.2.

Upon clicking, the button sends a request to the back-end via the 'get_object_dimension' route, which returns data such as the width, height, and distance of the defect, shown in Figure 37. This processing is carried out in the back-end as described in Section 6.2.4.9. This information can be used to determine how critical a defect is, allowing for easy remote monitoring and potential action to be taken if necessary.

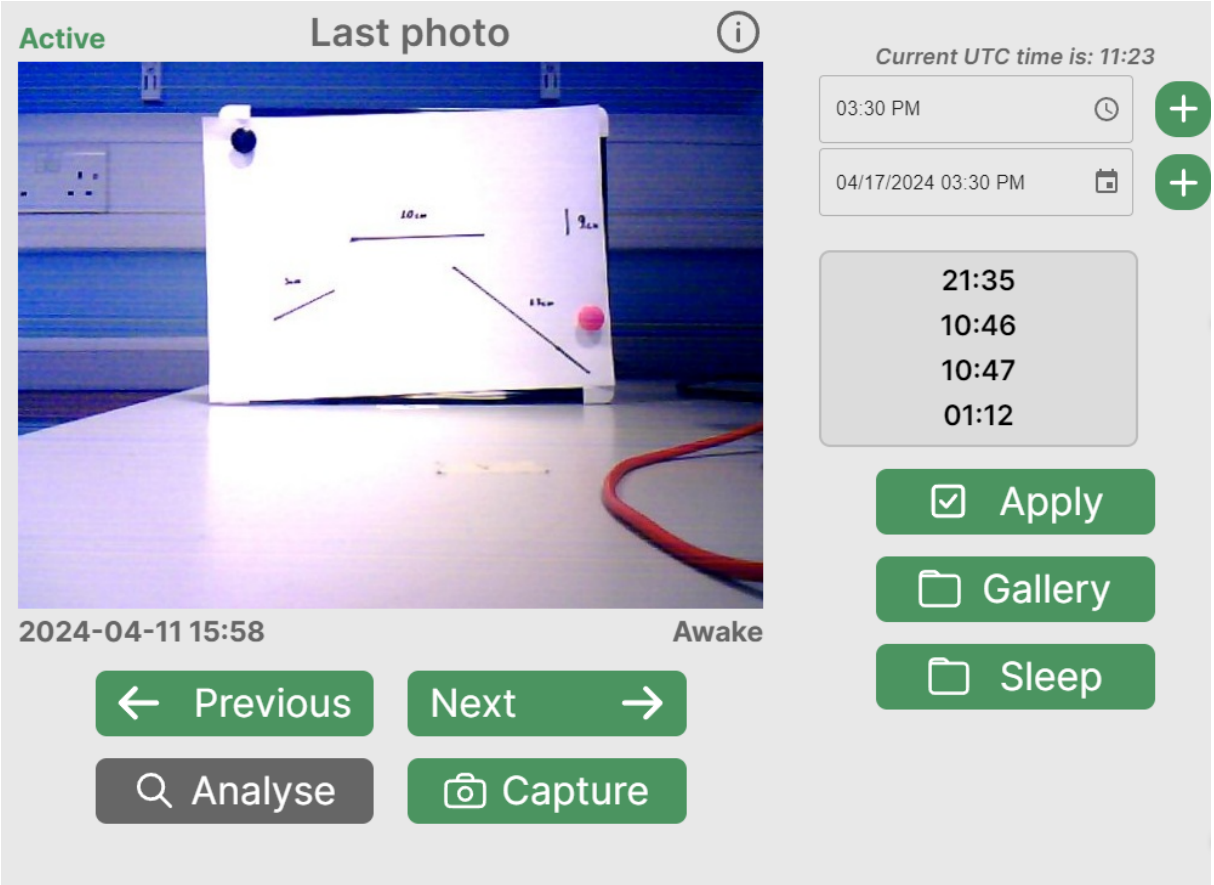



Figure 35: Screenshot of latest photo component

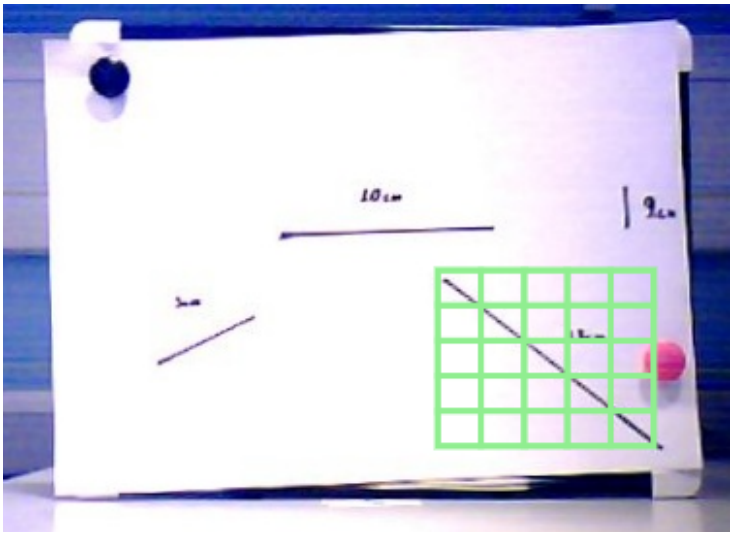


Figure 36: Screenshot of bounding box selection

Additional information can also be seen surrounding the photo in Figure 35, such as the date it was taken, and the next 'wake up' timestamp on the bottom right, obtained from the back-end as part of the 'automatic sleep' mode described in Section 6.2.6. This informs the user with contextual information regarding the state of the microcontrollers and eases the role of supervising the system remotely.

The capture button as seen in Figure 35 allows the user to send a request ('take_photo') to the back-end to take a photo on-demand. If successful, a notification as seen in Figure 38 acknowledges the request

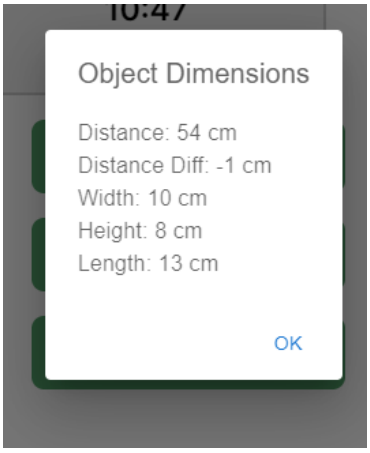



Figure 37: Screenshot of obtained object dimensions

did not fail, and a response object sends back the updated photo which can be seen in real-time on the last photo frame. Similarly, an error notification is also displayed if a connection cannot be made.

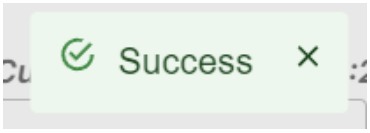



Figure 38: Screenshot showing a successful notification

Below the gallery button in Figure 35, the user can choose to hibernate the system manually. This results in the 'wuc_sleep' route provided by the back-end being called, which sets the system to sleep for a predetermined time set in the parameters. Upon clicking sleep, a dialog box is displayed to ensure the user wants to proceed with shutting the system down, as there is no way of waking it up before the timer ends remotely. A dialog box also informs the user of the calculated wake-up time, based on the sleep time set. This can be seen in Figure 39.

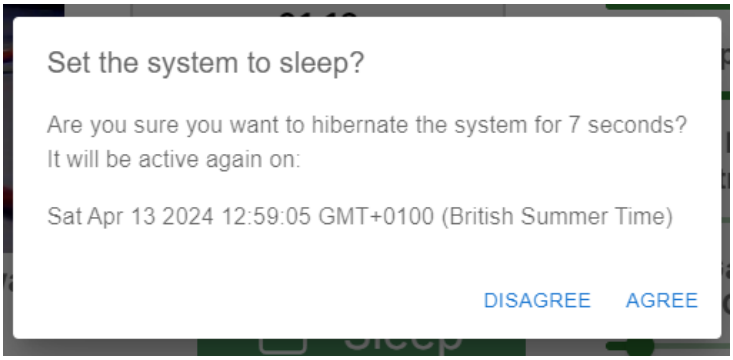



Figure 39: Dialog box prompting confirmation for sleep

#### 6.3.2.2 Settings & Parameters

The time picker component follows the design, as seen in Figure 40. Upon clicking the display box, users can input whatever time they want and add it via the plus button. In addition to the time picker established in the initial design requirements, a date-time picker was also added. This allows for greater control over scheduled captures, as the user may only require captures on specific dates that don't repeat daily. This reduces human intervention, as the user does not need to manually bother with removing and adding times to achieve the same effect, they can select dates and times in advance to be monitored, and leave certain days blank.

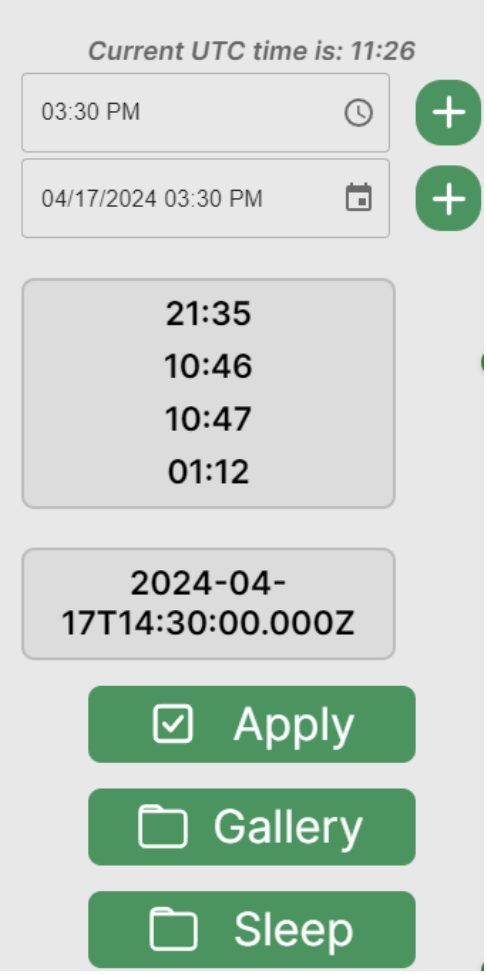



Figure 40: Screenshot showing the time picker component

Similarly, while the design for the parameter settings remains the same, more have been added to allow a greater range of customization on the photo, shown in Figure 41. Upon clicking, the apply button opens up a dialog box, informing the user of all the changes they've made to the schedule and the parameter. This ensures no mistakes are made, and upon accepting this dialog, the request is sent to the back-end with all the updated parameters. By default, the parameters are always synced to the back-end, so that there isn't any inconsistency with what is stored on the micro-controller and displayed on the website.

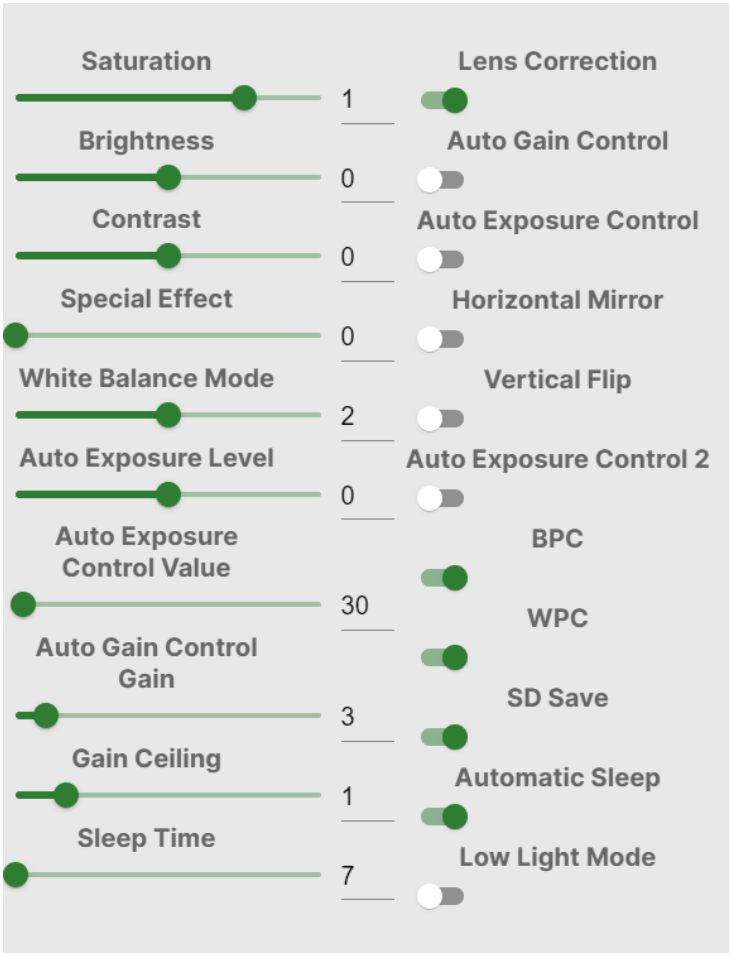



Figure 41: Screenshot showing customisable parameters and settings

#### 6.3.2.3 Gallery & Detailed Photo Pages

Implementation for the gallery page adheres exactly to the design, a grid of thumbnails is displayed showing each photo, along with a date underneath for reference, as seen in Figure 42. A page selector allows the user to browse through older photos. While not in the scope of the project, future work could involve adding more complex sorting and filtering functionality, making finding a single photo out of hundreds easier. Clicking any thumbnail redirects the user to the detailed photo page, which builds on the design with some extra functionality. While the user can swap through versions of the photo and see detailed parameter information, they can also analyse the photo for defects as established in the 'last photo' component. This feature is essential, as it allows any previously captured photos to be analysed regardless of whether they are on the main page or not.

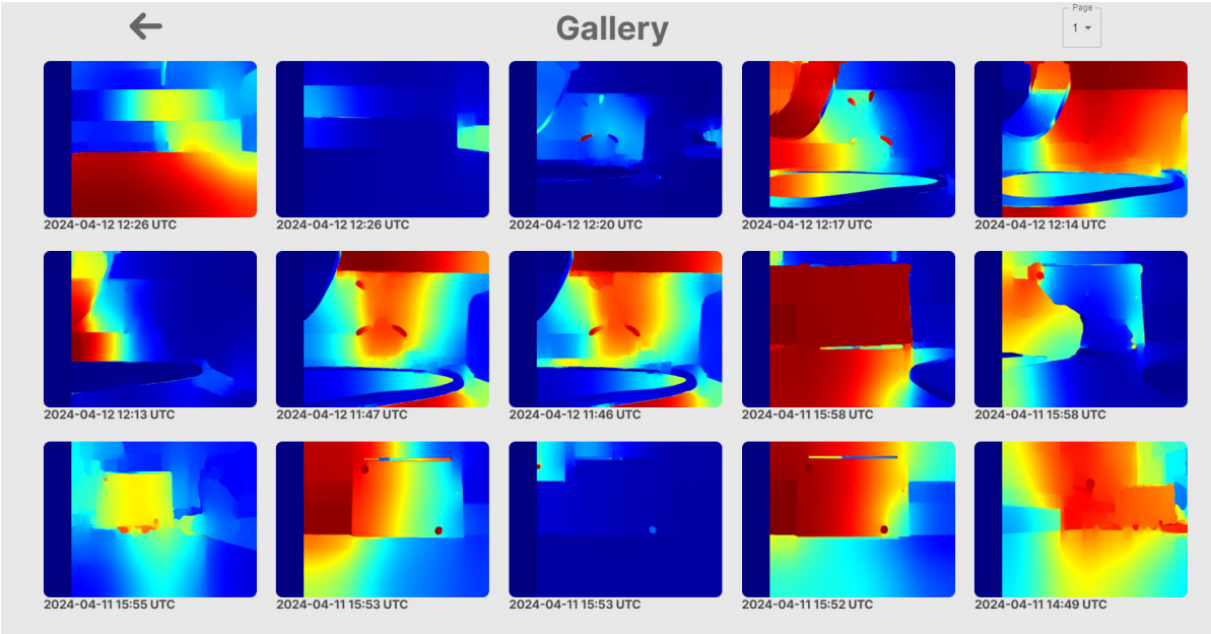



Figure 42: Gallery page

# 7 Image Processing

In the Image Processing section, the essential steps for utilizing the stereo images captured by our ultra-low-power Wi-Fi stereo camera are covered. Calibration and rectification , ensures that the cameras are accurately aligned and calibrated. The method of taking pictures of a checkerboard pattern at different distances and angles to calibrate the system effectively is explained. Moving on, the Disparity Map section discusses algorithms like Semi-Global Matching (SGM) and Weighted Least Squares (WLS) filtering, which are utilized to calculate depth from the stereo images. Then, in the Distance and Line Estimation part, the procedure for estimating distances between objects and detecting lines in the scene to understand space and locate objects better is described. Those procedures can be observed on Figure 43 below.

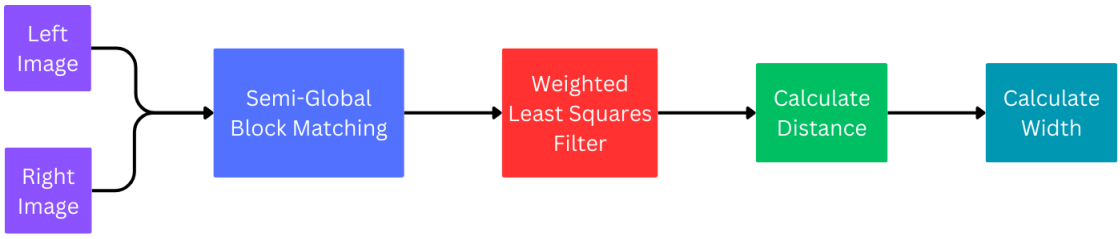



Figure 43: Stereo Processing Layout for Distance & Line Measurement

Finally, in 3D Reconstruction, the process of creating a three-dimensional view of the scene using the depth map and other processed data is outlined. Each of these sections is crucial for optimizing the utilization of our IoT ultra-low-power Wi-Fi stereo camera for remote inspection.

## 7.1 Calibration & Rectification

In the calibration and rectification process, the accuracy of our stereo camera system is ensured through careful adjustment and alignment. Initially, pictures of a checkerboard pattern placed at various distances from the cameras are taken. These pictures assist in configuring each camera's fundamental settings, such as lens focus and correction of lens distortions. Subsequently, using these images, the correct positioning of both cameras relative to each other is ensured, aiding in accurate depth measurement. Following this, image adjustments are made to ensure the neat alignment of corresponding points, simplifying the calculation of distances between objects. These steps play a crucial role in achieving precise depth measurements and facilitating 3D reconstructions.

### 7.1.1 Calibration Images Acquisition

Images of a checkerboard pattern are carefully captured to facilitate accurate calibration of the stereo camera system. The checkerboard, measuring 10 squares by 7 squares with each square spanning 25mm, serves as a calibration reference. A systematic approach is followed to ensure comprehensive calibration across different distances and orientations. Initially, three images are captured at a distance of 50 cm, with the checkerboard positioned straight in each image to capture the entire field of view. Similarly, four images are captured at 75 cm, five images at 100 cm, six images at 125 cm, and finally, six images at 150 cm. Additionally, different orientations of the checkerboard are considered, covering a wide

range of scenarios encountered in real-world applications. It is worth noting that guidelines provided by MATLAB are followed to ensure the effectiveness and reliability of the calibration process, maintaining consistency and accuracy throughout, thus enhancing the performance of the stereo camera system for depth estimation and 3D reconstruction.

### 7.1.2 Stereo Calibration & Rectification

In this section, meticulous attention was paid to ensuring the accuracy of our stereo camera system. The reprojection error, depicted in Figure 44, serves as a metric for evaluating the calibration process. With a mean error of only 0.13 pixels, it showcases the precision achieved in aligning the two camera views. This minimal error demonstrates the effectiveness of the calibration procedure in the subsequent depth estimation and 3D reconstruction processes.

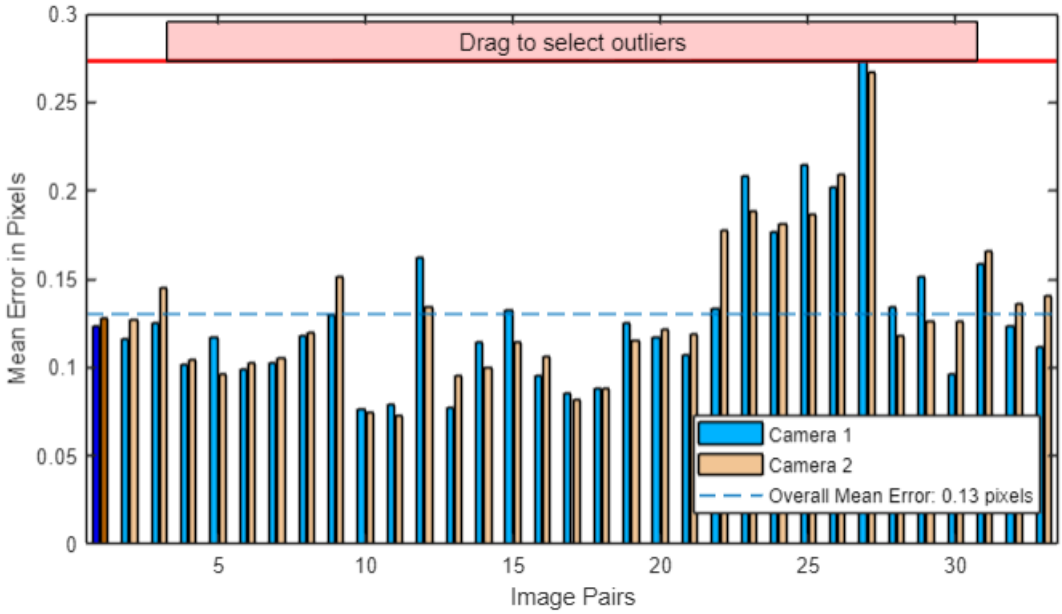



Figure 44: Reprojection errors of the two cameras during calibration

Additionally, the successful detection and reprojection of points on the chessboard, as illustrated in Figure 45, underscore the robustness of our calibration. Across all images, the points were accurately detected and reprojected, further validating the calibration's integrity and reliability.

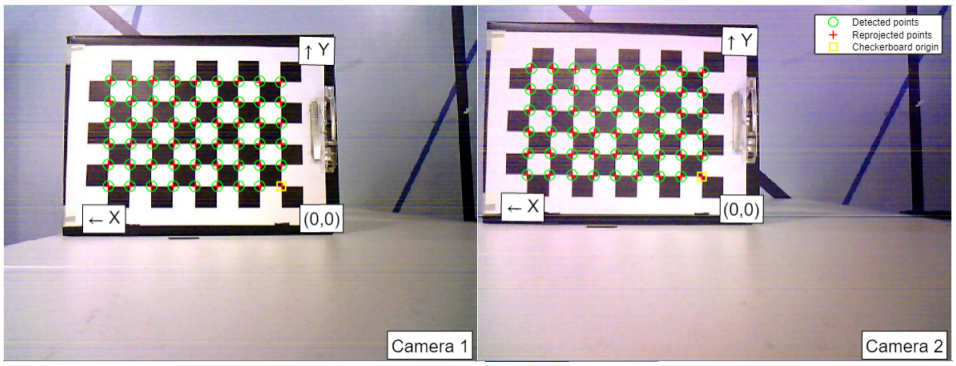



Figure 45: Successful detection and reprojection of points on the chessboard

Moreover, the rectification process, shown in Figure 46, further enhances the quality of our stereo

images. The rectified image of the chessboard displays precise alignment and distortion correction, essential for accurate depth perception and object localization. By rectifying the images, we ensure consistent geometrical properties between the left and right camera views, facilitating seamless correspondence matching and disparity calculation. It is worth noting that the calibration process and parameter extraction were meticulously conducted using MATLAB's stereo camera calibrator app, ensuring standardized and precise calibration across all experiments. These efforts collectively contribute to the robustness and accuracy of our stereo camera system for subsequent image processing tasks.

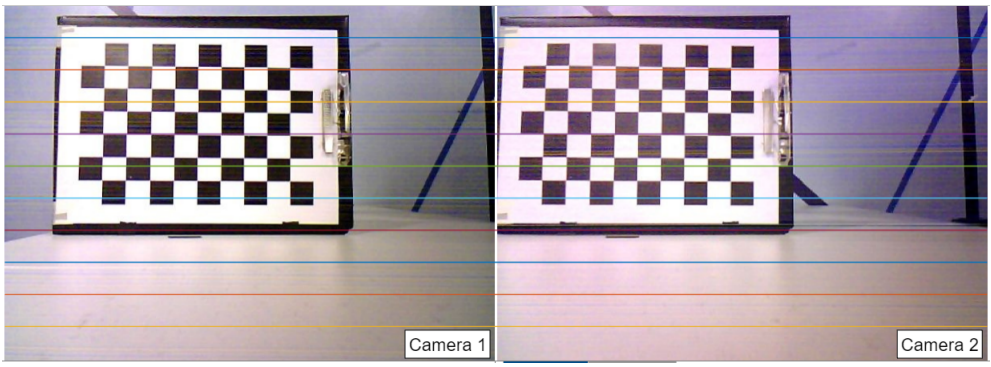



Figure 46: Rectified image of the chessboard

## 7.2 Disparity Map

In this section, the creation of a disparity map from the stereo images taken by our ultra-low-power Wi-Fi stereo camera is examined. Depth understanding, crucial for tasks such as object detection, scene building, and navigation, is facilitated by a disparity map. The methods employed to create the disparity map, with a focus on the Semi-Global Matching (SGM) algorithm and the Weighted Least Squares (WLS) filter, are explained. These techniques aim to translate differences between stereo images into clear depth estimates, thus enhancing our camera system's ability to perceive and understand the 3D world.

### 7.2.1 Semi-Global Matching Algorithm

In applying the Semi-Global Matching (SGM) algorithm between the left and right rectified images, the team began by selecting optimal parameters using a trackbar for each parameter. This approach aimed to achieve the best parameters yielding the most accurate disparity map. The Figure 47 was provided to illustrate the improvement of the disparity map achieved through parameter adjustments. Among the parameters utilized were minDisparity, numDisparities, and blockSize. For instance, the blockSize parameter, set to 1, determined the size of the window used for matching. Additionally, numDisparities was set to 16*4, indicating the total number of disparity levels to be computed. Despite these efforts, the resulting disparity map still exhibited noticeable artifacts and inaccuracies.

However, despite parameter optimization, the disparity map remained suboptimal, revealing numerous artifacts and incomplete object visibility between the left and right cameras. These discrepancies were attributed to the utilization of low-quality OV2640 cameras, resulting in inferior image quality. Furthermore, fluctuations in lighting conditions between the left and right cameras also contributed to disparity map inaccuracies. To address these issues and enhance the disparity map quality, the team proceeded to

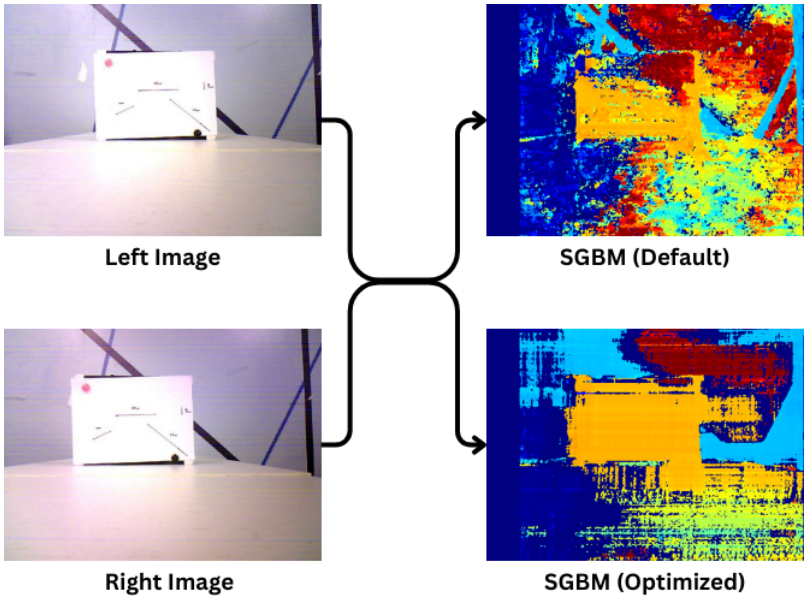



Figure 47: SGBM Disparity Map Comparison before & after optimization

implement Weighted Least Squares (WLS) filtering. This technique, discussed in the subsequent section, aimed to mitigate artifacts and improve the overall accuracy of the disparity map.

### 7.2.2 Weighted Least Squares (WLS) Filter

The procedure begins by creating a disparity map using the Semi-Global Matching (SGM) algorithm for both the left and right images. Subsequently, a right matcher is created to compute the disparity map for the right image. The DisparityWLSFilter is then initiated to perform the Weighted Least Squares (WLS) filtering. This filter aims to refine the initial disparity map by incorporating information from both the left and right images, thereby improving the accuracy of the depth estimation. Parameters such as lambda ($\lambda$) and sigma ($\sigma$) are set to control the filtering process. Finally, the filtered disparity map is obtained by applying the WLS filter to the disparity maps computed earlier, resulting in a more accurate representation of depth values.

The weighted least square filter was employed to enhance the disparity map generated by the SGBM algorithm, thereby achieving a more precise disparity estimation for improved depth perception. Figure 48 illustrates the disparity map's enhancement following the application of the WLS filter. While notable improvements are observed, particularly in the object of interest, some errors persist, particularly in the background and on the flat surface where the object rests. These errors may stem from variations in lighting between the left and right images, as well as the low quality of the cameras utilized. Additionally, reflections present in the images may interfere with the block matching process, contributing to discrepancies in the disparity map.

## 7.3 Distance & Line Estimation

In the next two sections, two key algorithms are introduced to estimate distances between objects and measure lines within the scene. The Distance Estimation Algorithm begins with the creation of a bounding box manually drawn by the user on the left image, serving as a mask to extract the corresponding region

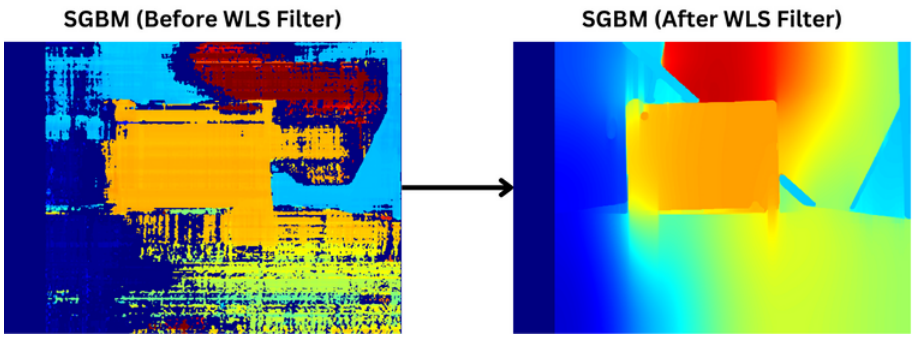



Figure 48: SGBM Disparity Map Comparison before & after WLS filtering

from the disparity map. This process enables the retrieval of the disparity value corresponding to the object of interest. Also, geometric principles are employed to calculate the distance from the camera to the object using the stereo camera setup. Meanwhile, the Line Measurement Algorithm focuses on drawing a line on the object in the left camera image, simulating defect size measurement that would typically be detected by a CNN model. However, due to time constraints and the unavailability of defect and crack datasets, the implementation of the CNN model and subsequent defect detection was postponed. Thus, the line measurement serves as a substitution for defect size estimation, providing valuable insights into the object's dimensions and potential defects.

### 7.3.1 Distance Estimation Algorithm

Depth calculation from disparity involves using the Equation 2, derived from the principle of similar triangles as depicted in Figure 49. When observing an object from two different viewpoints, the resulting images form similar triangles with the object. By analyzing these triangles alongside the baseline distance (B) and focal length (f), a relationship between disparity ($\delta$) and depth (D) can be established. This formula indicates that depth is directly proportional to the baseline distance and focal length, while inversely proportional to the disparity. Consequently, as the disparity decreases, indicating closer objects, the depth value increases, signifying greater distance from the camera.

$$D = \frac{f \times B}{\delta} \tag{2}$$

Additionally, a mask is utilized to extract the specific disparity value from the disparity map, based on the bounding box created by the user from the left image. This bounding box defines the region of interest within the disparity map, allowing for precise selection of the object's disparity value. Once the disparity value is obtained, the distance to the object can be calculated using the known baseline distance (5.5cm) and focal length, which was determined through the calibration process. By integrating these parameters into the depth calculation formula, accurate distance estimations can be derived, giving precise spatial understanding and object localization.

### 7.3.2 Line Measurement Algorithm

The Line Measurement Algorithm uses the depth calculation to accurately measure the dimensions of a defect present in the scene. By applying the principle of similar triangles once more, shown on Figure

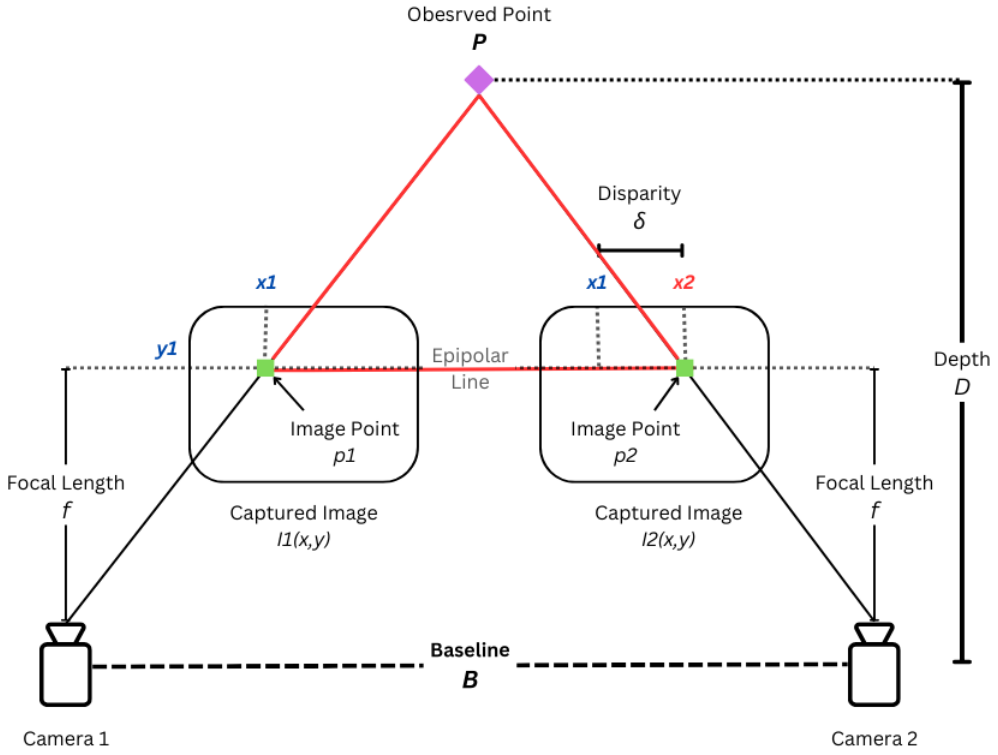



Figure 49: Illustration showing the principle of similar triangles for depth calculation.

50, we utilize the Equation 3 to determine the defect's width in centimeters. The first small triangle consists of the defect's width in pixels, the focal length , and the depth. The second triangle represents the same defect, but with its width now measured in centimeters and the depth also in centimeters. Here, $Wpx$ represents the width of the defect in pixels, a known quantity derived from the bounding box drawn by the user. With the depth (D) of the defect and the baseline distance (B) between the cameras already established, along with the focal length (f), this formula facilitates the conversion of pixel measurements into real-world dimensions. A similar approach is employed to get the height of the defect, enabling comprehensive dimensional analysis. Additionally, utilizing the Pythagorean theorem, the length of the defect can be calculated by considering the width and height obtained through the depth-based measurements.

$$W_{\text{cm}} = \frac{D \times W_{\text{px}}}{f} \tag{3}$$

However, this methodology is applicable only when the defect line is perpendicular to the cameras' view. In cases where there is an angle between the camera and the defect line, resulting in a difference between the maximum and minimum disparity of a specific value (e.g., 15), indicating a significant distance variation along the line, the bounding box is subdivided horizontally and vertically into smaller bounding boxes. For each smaller bounding box, the distance is calculated individually, allowing for the determination of the width and height, which are then summed to obtain the total dimensions. Then, the Pythagorean theorem is employed again to compute the length of the defect line accurately, considering the geometric properties of the scene. This approach ensures precise defect measurement even in scenarios involving non-perpendicular defect lines.

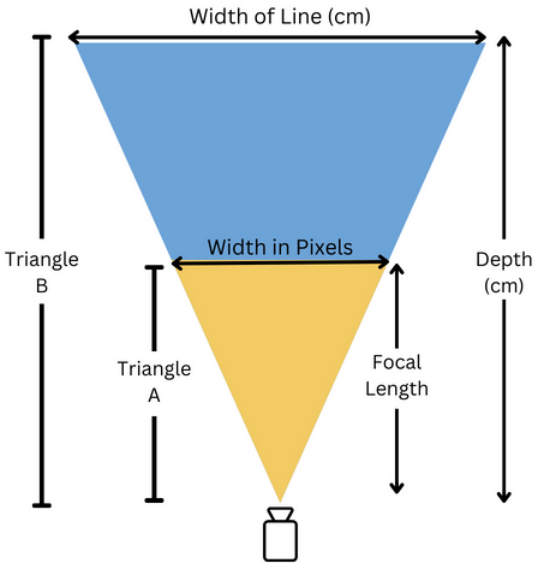



Figure 50: Illustration showing the principle of similar triangles for line calculation.

## 7.4 3D Reconstruction

In the analysis of metallic objects with features like holes and varied textures, generating accurate 3D models is essential for defect detection and comprehensive remote monitoring. The process of 3D reconstruction involves several steps. Initially, a disparity map is computed from the stereo images, capturing the differences in pixel positions between corresponding points in the left and right images. Using the disparity map and camera parameters such as the focal length and baseline distance, a transformation matrix (Q) is constructed to convert pixel disparities into 3D coordinates. These coordinates are then adjusted for visualization, including reflections and colour extraction from the original images. By filtering the resulting 3D points based on minimum disparity and dimensional constraints, a refined set of 3D points and their corresponding colours are obtained. Finally, the data is formatted into a PLY (Polygon File Format) file, which represents the 3D model with vertex coordinates and associated colour information. The Figure 51 shows the left image of the object alongside its corresponding 3D model. This 3D model not only facilitates defect analysis of complex objects like pipes but also enables remote monitoring systems to accurately assess object conditions and identify potential defects through the pass of time.

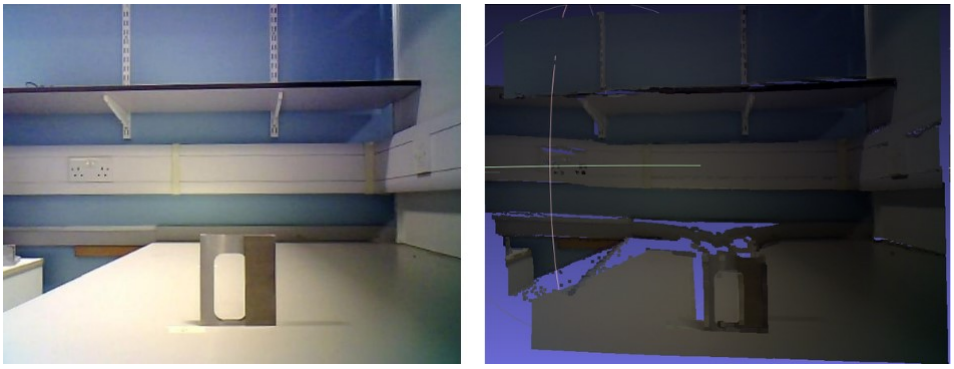


Figure 51: The left image of the metal object (left) and its corresponding 3D model generated from the disparity map (right).

# 8 Testing & Results

## 8.1 Current Draw and Battery Life

Due to obvious time limitations, the battery life of the device could not be accurately tested by recording the endurance of a full charge. Therefore the time period for which a single charge can last has to be estimated. To do this, current draw was measured for two states: deep sleep and active. The device is considered to be in deep sleep state when only the WUC has power and is itself in deep sleep mode. The device is considered to be in active state at all other times.

Current draw was measured for both states using a multi-meter. To get an average value, the display of the multimeter was recorded with a smartphone camera in slow motion; the current values seen in the frames of the video were recorded, and an average was calculated. Seven scenarios were considered when estimating battery life:

- constantly in active state
- constantly in deep sleep state
- in active state for X minutes per day, deep sleep state for the rest of the day
  (x = 1, 2, 3, 4, 5)

The device typically remains in the active state for less than 30 seconds for each image capture. Thus, for each scenario, $2 \cdot x$ image captures are performed. The average current for a 24-hour period was calculated for each scenario using Equation (4) below.

$$Avg.\ Current\ Draw\ = \frac{ActiveCurrent \cdot x + DeepSleepCurrent \cdot (24 \cdot 60 - x)}{24 \cdot 60} \tag{4}$$

The estimated battery life in days was then calculated using Equation (5) where Battery Capacity is in Ampere hours and Avg. Current Draw in Amperes. The findings of the testing are summarised in Table 14.

$$Battery\ Life\ = \frac{BatteryCapacity}{Avg.CurrentDraw \cdot 24} \tag{5}$$

A note should be made that the deep sleep current value of the FireBeetle 2 ESP32-E is lower than that seen in Table 2. This could be due to a variety of reasons due the amount of components interacting in real time and the initial testing scenario not fully replicating the final system. Each component is affected by temperature and by the outputs of all of the other preceding components. Furthermore, both microcontroller units are considered active devices, with their power consumption changing based on which internal components are used at the time. The testing method used for 2 was simply measuring the deep sleep current draw with a timer as a wake-up source, therefore no other external GPIO connections were used like in the final system, which could affect power consumption. Using a bench power supply

to replace a battery is also not an accurate method for replicating the final system. This is due to the battery's nominal voltage being the average voltage supplied by the battery and not the actual supplied voltage. Some more technical reasons for the discrepancy would be the inaccuracies of the lab equipment such as the power supply used for the tests and the ammeter used to measure the current draw. All of the listed reasons are why there is a difference between the initially tested deep sleep power consumption and the final one.

Table 14: Current Draw and Estimated Battery Life

| **Activity State** | **Avg. Current Draw** | **Battery Life** |
|---|---|---|
| **Active** | 172 mA | 2.55 Days |
| **Deep Sleep** | 0.642 mA | 681 Days |
| **1 min Active/Day** | 0.761 mA | 575 Days |
| **2 min Active/Day** | 0.880 mA | 497 Days |
| **3 min Active/Day** | 0.999 mA | 438 Days |
| **4 min Active/Day** | 1.119 mA | 391 Days |
| **5 min Active/Day** | 1.239 mA | 353 Days |

## 8.2 MOSFET Board

To test the MOSFET board initially a simulation was created on LTspice, shown in Figure 52. The 2N7000 model was used from the built in libraries whereas the AO3401 model was inputted using the following SPICE directive,

.model AO3401 VDMOS Rg=7.8 Rd=0m Rs=0.2m Vto=-1.2 Kp=17 Cgdmax=400p Cgdmin=80p Cgs=645p Cjo=147p Is=10p Rb=0.2m Pchan mfg=Alpha_&_Omega ksubthres=.1

and the DC operating point was used as the simulation mode, which uses DC voltage sources with their values remaining as they were set.

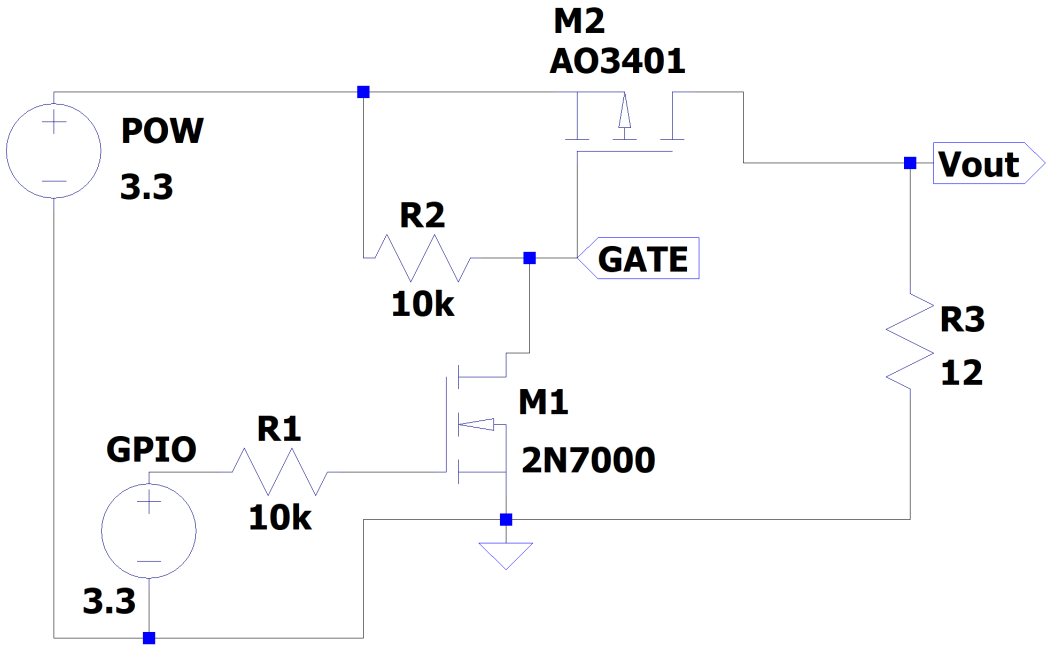



Figure 52: LTspice simulation circuit diagram

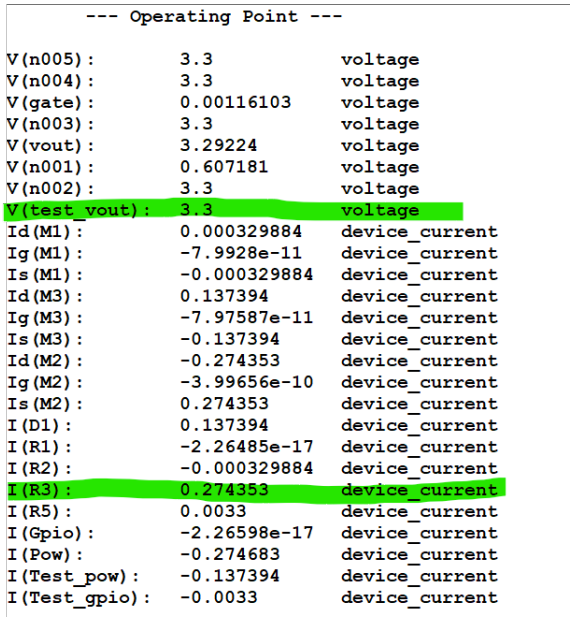
POW is the $V_{CC}$ pin and GPIO is GPIO2 of the WUC. $R3$ is set to 10Ω to simulate the ESP32-CAM. This values was deduced by using the ESP32-CAM Datasheet [30] and performing some calculations. Since at 5 V the microcontroller uses 180 mA, the power dissipated is $P = VI = 5 * 0.180 = 0.9$ W. To find the current required for 3V3 operation, the equation used is re-arranged in Equation (6) below,

$$I = \frac{P}{V} = \frac{0.9}{3.3} = 0.273\ mA \tag{6}$$

Using another form of the power equation $P = \frac{V^2}{R}$, the resistance of the microcontroller can also be found under 3V3 conditions by rearranging in Equation (7) below,

$$P = \frac{V^2}{R} \rightarrow R = \frac{V^2}{P} = \frac{3.3^2}{0.9} = 12.1\ \Omega \tag{7}$$

By using the data in Equations (6) and (7) it is now known what current draw to expect from the active system and how that can be simulated via a resistance $R3$ of 12Ω. Simulating the circuit gives us the desired results as shown in Figure 53 therefore showing correct functionality of the MOSFET Board.



--- Operating Point ---

| | | |
|---|---|---|
| V(n005): | 3.3 | voltage |
| V(n004): | 3.3 | voltage |
| V(gate): | 0.00116103 | voltage |
| V(n003): | 3.3 | voltage |
| V(vout): | 3.29224 | voltage |
| V(n001): | 0.607181 | voltage |
| V(n002): | 3.3 | voltage |
| V(test_vout): | 3.3 | voltage |
| Id(M1): | 0.000329884 | device_current |
| Ig(M1): | -7.9928e-11 | device_current |
| Is(M1): | -0.000329884 | device_current |
| Id(M3): | 0.137394 | device_current |
| Ig(M3): | -7.97587e-11 | device_current |
| Is(M3): | -0.137394 | device_current |
| Id(M2): | -0.274353 | device_current |
| Ig(M2): | -3.99656e-10 | device_current |
| Is(M2): | 0.274353 | device_current |
| I(D1): | 0.137394 | device_current |
| I(R1): | -2.26485e-17 | device_current |
| I(R2): | -0.000329884 | device_current |
| I(R3): | 0.274353 | device_current |
| I(R5): | 0.0033 | device_current |
| I(Gpio): | -2.26598e-17 | device_current |
| I(Pow): | -0.274683 | device_current |
| I(Test_pow): | -0.137394 | device_current |
| I(Test_gpio): | -0.0033 | device_current |

Figure 53: Highlighted in green are the important values from the simulation. $V_{OUT}$ is 3.3 V at the expected current of 0.274 mA.

## 8.3 Microcontroller Code

### 8.3.1 WUC Testing

Testing the WUC code was done by isolating the project code into separate projects and modifying it so that only the specific test case is tested. The SSID loaded currently unto the system upon project submission was "pap" and the Password is "12345678". Therefore to use the project, a network with those credentials has to exist within the vicinity. Test cases and outcomes are stated below in Table 15.

Table 15: Testing Cases for the WUC along with the Expected and Observed Outcomes.

| **Test ID** | **Test Case** | **Expected Outcome** | **Observed Outcome** | **Comments** |
|---|---|---|---|---|
| W1 | Connecting to WiFi with hard-coded credentials. Used SSID: "pap", Paswd: "12345678". | Connection Successful. | Connection Successful. | May fail to connect once or twice but due to the implementation of the retries, the conenction is eventually established. Does not work on a 5 GHz access point. |
| W2 | Connecting to WiFi using Smart Config. Connected to a mobile hotspot with SSID: "pap", Paswd: "12345678" and used EspTouch to establish a WiFi connection to the WUC. | Connection Successful. | Connection Successful. | Takes around 10-20s for the packets to be detected and for connection to be established. |
| W3 | Connecting to WiFi using the credentials in stored the NVS. | Connection Successful. | Connection Successful. | Credentials are accessed very quickly with no issues. |

| | | | | |
|---|---|---|---|---|
| W4 | Using Smart Config to retrieve network credentials and store it in the NVS. Tested by executing Smart Config and restarting the system. | Credentials are saved and Smart Config is not executed at second boot. | Credentials are saved and Smart Config is not executed at second boot. | - |
| W5 | Setting GPIO2 to high to turn on the MCC. Used a multimeter connected to the WUC's GPIO2 and GND connections. MCC is powered through the MOSFET board. | GPIO2 Succesfully goes into logical high state. MCC turns on. | GPIO2 Succesfully goes into logical high state. | - |
| W6 | Setting GPIO2 to low to turn off the MCC. Used a multimeter connected to the WUC's GPIO2 and GND connections. MCC is powered through the MOSFET board. | GPIO2 Succesfully goes into logical low state and MCC turns off. | GPIO2 Succesfully goes into logical low state | - |
| W7 | Starting the HTTP server and using `/get` to test if it is functional. Used the IPs detected by the hotspot, e.g. http://192.168. 145. 45: 19520/get | "URI GET Response" outputted on the browser page. | "URI GET Response" outputted on the browser page. | Takes longer to appear when using an unstable connection. |

| | | | | |
|---|---|---|---|---|
| W8 | Using the `/reset` to reset the currently stored WiFi credentials | Credentials reset and system boots into Smart Config. | Credentials reset and system boots into Smart Config. | Requires access to the currently used network to reset to input the new network. |
| W9 | Using the `/exit` to exit the main server loop | Server loop is exited and system enters deep sleep. | Server loop is exited and system enters deep sleep | Uses the `.sleep_time_sec` system config variable member for the sleep duration. |
| W10 | Using the `/sleep` to set the sleep time. Example cURL command used to sleep for 10 seconds: "curl -ContentType 'text/plain' -Body "10" -Method Post http://192.168. 145: 45:19520/sleep" | After exiting the main loop, WUC sleeps for 10 seconds. | After exiting the main loop, WUC sleeps for 10 seconds | Stores the value from the cURL command into the `.sleep_time_sec` system config variable member. Must call `/exit` to enter deep sleep. |

### 8.3.2 MCC Testing

To test the MCC, the same network was used as the WUC, SSID: "pap", Password: "12345678". Test cases and outcomes are shown below in Table 16.

Table 16: Testing Cases for the MCC along with the Expected and Observed Outcomes.

| Test ID | Test Case | Expected Outcome | Observed Outcome | Comments |
|---|---|---|---|---|
| M1 | Single camera initialisation | Succesfull initialisation | Succesful initialisation | - |
| M2 | Setting GPIO4 high or low for switching between the two cameras. Multimeter connected to MCC GPIO4 and GND pins. | Successful change of GPIO state. | Successful change of GPIO state. | GPIO4 is connected to the flash LED, therefore at each switch the flash LED turns on for the duration the second camera is active. |
| M3 | Initialising the SD card and reading the WiFi text file. "`wificreds.txt`" | Successfully read text file. | Successfully read text file. | Removing of the escape characters has not been tested on Linux or MacOS systems where they use `LR` instead of `CLRF`. |
| M4 | Connecting to WiFi with the read credentials. | Connection established. | Connection established. | Connection may fail at first try but the system retries and eventually connects. Does not work on a 5 GHz access point. |
| M5 | Starting the HTTP server and using `/get` to test if it is functional. Used the IPs detected by the hotspot, e.g. http://192.168. 145. 45: 19520/get | "URI GET Response" outputted on the browser page | "URI GET Response" outputted on the browser page | Takes longer to appear when using an unstable connection. |
| M6 | Using the `/exit` to exit the main server loop | Server loop is exited and system returns from main | Server loop is exited and system returns from main | - |

| | | | | |
|---|---|---|---|---|
| M7 | Using the `/pic` to take a picture | Picture taken and sent to the browser | Picture taken and sent to the browser | Using `/pic` over an unreliable connection may lead to the transmission failing in the case of strict firewall rules or a poor connection. Port forwarding the device on the access point is recommended! |
| M8 | Using the `/pic` twice to take two pictures | Both pictures taken and sent to the browser | Both pictures taken and sent to the browser | Same comments as above. |
| M9 | Using the `/config` to send new settings to the camera | Camera settings changed | Camera settings changed | See Section 8.3.3 for results from this test. |
| M10 | Saving both images to the SD card | Both images are saved successfully | Both images saved successfully | When the sensors are set to take a photo using JPEG, the settings do not have an affect. Therefore the auto-adjust functionalities are in effect in this case to adjust automatically to the current scene. |

### 8.3.3 Images taken with the MCC

In Figure 55, showcases some of the results from testing the MCC camera settings; each images differs by one setting. The base image in Figure 54 is taken with the default settings stated in Table 9.

Figure 54: Base image with default settings.

Figure 55: Settings used from left to right, `vflip = true, hmirror = true, ae_level = 2, ae_level = -2` .

Figure 56: Settings used from left to right, `aec_level = 0, aec_level = 200, aec_level = 1000, aec2 = false`.

Figure 57: Settings used from left to right, `aec = true, agc_gain = 10, agc_gain = 30, agc = true`

Figure 58: Settings used from left to right, `bpc = false`, `brightness = 2`, `brightness = -2`, `contrast = 2`

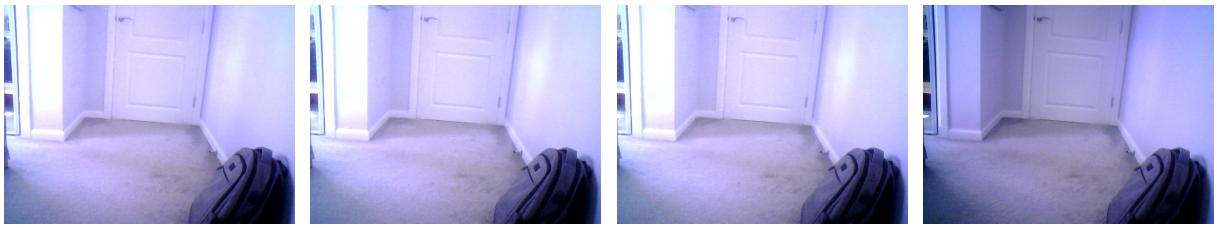


Figure 59: Settings used from left to right, `contrast = -2`, `gainceiling = 3`, `gainceiling = 6`, `lenc = false`

Figure 60: Settings used from left to right, `saturation = -2`, `saturation = 2`, `special_effect = 1`, `special_effect = 2`

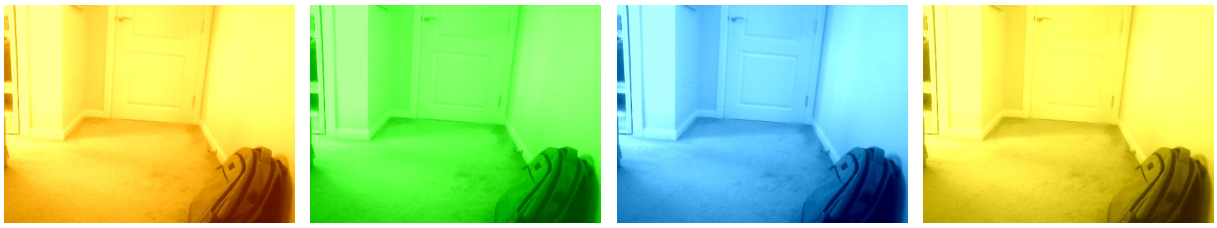


Figure 61: Settings used from left to right, `special_effect = 3`, `special_effect = 4`, `special_effect = 5`, `special_effect = 6`

Figure 62: Settings used from left to right, `wb_mode = 0`, `wb_mode = 1`, `wb_mode = 2`, `wb_mode = 3`, `wb_mode = 4`

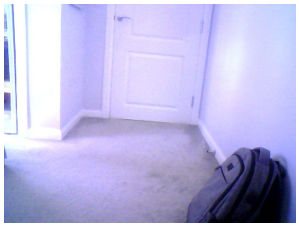


Figure 63: Settings used, `wpc = false`

### 8.3.4 Light Level Comparison

A more direct comparison to lighting conditions can be seen in Figures 64 and 65.

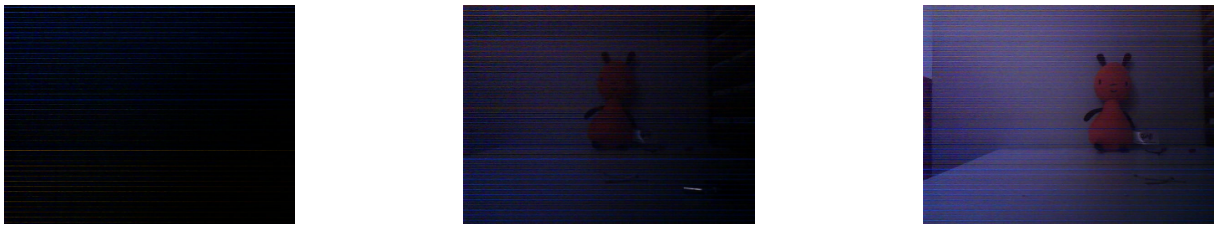


Figure 64: AEC Values from left to right, 10, 200, 600. Object is placed at very dark conditions.

Figure 65: AEC Values from left to right, 10, 200, 600. Object is placed at very illuminated conditions.

### 8.3.5 Results

Overall, all of the test cases executed in Tables 15 and 16 provide a sufficient exploration of the capabilities of the WUC and MCC, and therefore the system as a whole. The WUC can wake up the MCC and then the MCC can take the two required pictures needed to start the stereo vision process. Settings for the cameras can be changed over the network, and the device can adapt to a variety of scenery such as very dark or very bright. Implementation of extra features such as offline functionalities and SD card saving allow the device to automatically adapt in situations where the network may be down by saving pictures automatically to the SD card. The device can also be connected to new networks without requiring the developer to reset the on-board credentials, thus allowing WiFi provisioning.

## 8.4 Website Testing

The website was tested by going through each of the separate test cases and recording the results. Test cases and outcomes are shown below in Table 17.

Table 17: Testing Cases for the website along with the Expected and Observed Outcomes.

| Test ID | Test Case | Expected Outcome | Observed Outcome | Comment |
|---|---|---|---|---|
| S1 | Clicking capture button. | Success notification appears after a few seconds, and photo updates. | Success notification appears after a few seconds, and photo updates | Takes some time because of delay. |
| S2 | Going to gallery page. | Takes you to gallery page. | Takes you to gallery page | Takes time to load from DB. |
| S3 | Analysing a photo via input. | Dialog box showing dimensions appears. | Dialog box showing dimensions appears. | - |
| S4 | Applying parameter changes | Confirmation box appears and clicking it applies changes | Confirmation box appears and clicking it applies changes | - |
| S5 | Applying parameter changes with asleep MCC. | Confirmation box appears and clicking it should cause error. | Confirmation box appears and clicking it should cause error. | Non MCC settings still get changed. |
| S6 | Clicking sleep button while MCC is awake. | Confirmation box appears and clicking sleep causes success notification. | Confirmation box appears and clicking sleep causes success notification. | - |
| S7 | Clicking sleep button while MCC is asleep. | Confirmation box appears and clicking sleep causes error. | Confirmation box appears and clicking sleep causes error. | - |

### 8.4.1 Results

The test cases in Table 17 showcase most of the functionality of the website, and all the outcomes observed from the tests are what was expected.

## 8.5 Image Processing Testing & Results

### 8.5.1 Depth Estimation for Various Baselines

In this subsection, the camera system was tested with various baseline distances to determine the optimal configuration. Different baseline distances were employed, and the resulting depth estimations analyzed to identify the baseline that yielded the most accurate results.

#### 8.5.1.1 Testing

Determining the optimal baseline distance for our system involved an approach for achieving a maximum depth estimation of up to 2 meters while minimizing the impact of the camera's low resolution (640x480) on size measurement accuracy. At the start, an initial range for the baseline was established using an [74], yielding a range of 4cm to 8cm. Then, a series of tests were conducted to assess the performance of the system across different baseline distances. The results of these tests, as depicted in Table 18, showcased the varying depth ranges achieved at baseline distances of 4cm, 5.5cm, and 7cm.

| Baseline Distance (cm) | Minimum Depth (cm) | Maximum Depth (cm) |
|---|---|---|
| 4 | 25 | 100 |
| 5.5 | 30 | 150 |
| 7 | 50 | 200 |

Table 18: Depth range achieved at different baseline distances.

#### 8.5.1.2 Results

In the results, it is evident that the choice of baseline distance significantly impacts the depth estimation capabilities of the camera system. As illustrated in Table 18, baseline distances of 5.5cm and 7cm exhibit broader depth ranges compared to a baseline distance of 4cm. Specifically, the baseline distance of 7cm allows for a maximum depth estimation of up to 200cm, meeting the desired requirement of estimating depths up to 2 meters. However, it is important to note that while increasing the baseline distance expands the depth range, it may also introduce trade-offs in terms of size measurement accuracy, particularly for objects closer to the camera. Thus, careful consideration must be given to balancing depth estimation requirements with size measurement precision when selecting the optimal baseline distance for the camera system. The choice of 5.5cm as the optimal baseline distance ensures that our system can effectively estimate depths up to 2 meters while maintaining the required accuracy for defect size measurement with our low-resolution camera.

### 8.5.2 Testing Depth Estimation at Different Distances

This subsection focuses on evaluating the depth estimation capabilities of the camera system at different object distances while maintaining a constant baseline. The camera system was tested with objects placed at various distances, ranging from being close and farther to the camera.

#### 8.5.2.1 Testing

To evaluate the depth estimation capabilities of our system, tests were conducted at various object

distances ranging from 20cm to 200cm. Each test involved placing an object at a specific distance from the camera system and recording the depth estimation provided by our algorithms. The distances selected cover a wide range commonly encountered in real-world scenarios, allowing us to assess the system's performance across different spatial conditions. The results obtained from these tests are summarized in Table 19.

| Object Distance (cm) | Depth Estimation (cm) |
|---|---|
| 20 | 14 |
| 30 | 27 |
| 50 | 51 |
| 75 | 74 |
| 100 | 98 |
| 125 | 118 |
| 150 | 140 |
| 200 | 180 |

Table 19: Depth estimation for an object at different distances.

#### 8.5.2.2 Results

The depth estimation results, as depicted in Table 19, indicate a generally accurate performance of our system across varying object distances. At distances 30cm to 100cm, the depth estimations closely align with the actual distances, with errors ranging from 1cm to 3cm. However, as the distance increases beyond 125cm, slight discrepancies between the estimated and actual distances become more noticeable, with errors ranging from 7cm to 20cm. These errors, as detailed in Table 20, reflect the difference between the estimated and actual distances for each object distance tested. The highest error occurs at a distance of 20cm and 200cm, where the depth estimation has an error of 6cm and 20cm, accordingly, from the actual distance.

| Object Distance (cm) | Depth Estimation Error (cm) |
|---|---|
| 20 | 6 |
| 30 | 3 |
| 50 | 1 |
| 75 | 1 |
| 100 | 2 |
| 125 | 7 |
| 150 | 10 |
| 200 | 20 |

Table 20: Depth estimation errors for different object distances.

The graph shown in Figure 66 illustrates the comparison between the actual distances and the corresponding estimated distances obtained from our depth estimation algorithms. Overall, the graph shows a good alignment between the actual and estimated distances across the tested range. However, slight dis-

crepancies are noticeable, particularly for distances beyond 125cm, where the estimated distances tend to slightly overestimate the actual values. Despite these deviations, the graph underscores the effectiveness of our depth estimation algorithms in accurately estimating defect distances within a reasonable margin of error.

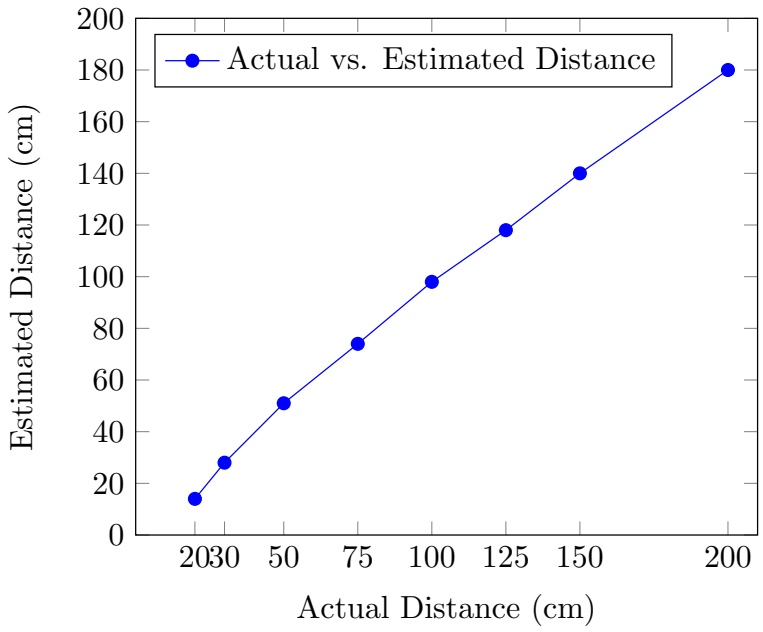



Figure 66: Actual vs. Estimated Distance

### 8.5.3 Size Measurement Testing

In this section, the camera's ability to measure the size of lines was tested. Lines of known dimensions were placed within the camera's field of view, and their sizes were measured using the system's algorithms. The measured sizes were compared against the ground truth to evaluate the system's precision and reliability in size measurement.

#### 8.5.3.1 Testing

For the size measurements, tests were conducted involving lines of different lengths (2cm and 10cm) placed at various distances from the camera system (ranging from 20cm to 200cm). For each distance, measurements were taken for both perpendicular and angled lines. The perpendicular lines were positioned parallel to the camera's focal plane, while the angled lines were positioned at approximately 45 degrees relative to the camera. This arrangement aimed to simulate real-world scenarios where objects might not be perfectly aligned with the camera's axis. The measurements were obtained using our depth estimation algorithms, and the results were recorded in Table 21.

#### 8.5.3.2 Results

The results from the size measurement testing, as depicted in Table 21, reveal insights into our system's performance in measuring the length of lines at various distances and orientations. For lines perpendicular to the camera, the measured lengths show consistent results, with minimal error across the tested distances. However, for lines angled to the camera, differences are observed, particularly for shorter

| Object Distance (cm) | Perpendicular to Line | | Angled to Line (cm) | |
|---|---|---|---|---|
| | 2cm | 10cm | 2cm | 10cm |
| 20 | 1.6 | 9.4 | N/A | N/A |
| 30 | 1.8 | 9.7 | 1.1 | 8.2 |
| 50 | 2.1 | 10.1 | 1.2 | 8.5 |
| 75 | 2.0 | 9.9 | 1.3 | 8.7 |
| 100 | 1.9 | 9.8 | 0.9 | 7.2 |
| 125 | 1.4 | 9.1 | N/A | N/A |
| 150 | 1.3 | 9.0 | N/A | N/A |
| 200 | 1.2 | 8.9 | N/A | N/A |

Table 21: Length measurement of lines at different distances.

distances. This is likely attributed to factors such as suboptimal calibration, low-resolution cameras, and varying lighting conditions, which can affect the accuracy of the depth estimation and consequently impact the size measurement. Additionally, inconsistencies in the length measurements at certain distances, indicated by "N/A" entries, further highlight the challenges encountered in maintaining measurement accuracy under specific conditions.

The error analysis, detailed in Table 22, provides further insights into the discrepancies between the measured lengths and the actual sizes of the lines. The errors are relatively low for lines perpendicular to the camera, with deviations ranging from 0.1cm to 1.1cm. However, for lines angled to the camera, the errors are more pronounced, particularly at shorter distances, where deviations range from 0.8cm to 1.8cm. These errors are consistent with the challenges highlighted earlier.

| Object Distance (cm) | Perpendicular Error (cm) | Angled Error (cm) |
|---|---|---|
| 20 | 0.4 - 0.6 | N/A |
| 30 | 0.2 - 0.3 | 0.9 - 1.8 |
| 50 | 0.1 | 0.8 - 1.5 |
| 75 | 0 - 0.1 | 0.7 - 1.3 |
| 100 | 0.1 - 0.2 | 1.1 - 1.8 |
| 125 | 0.6 - 0.9 | N/A |
| 150 | 0.7 - 1 | N/A |
| 200 | 0.8 - 1.1 | N/A |

Table 22: Error in length measurement at different distances.

The graph in Figure 67 illustrates the comparison between errors in length measurement for lines perpendicular and angled to the camera at various object distances. It is evident that the errors fluctuate differently with changes in object distance for both types of lines. The errors for lines perpendicular to the camera remain relatively consistent across different distances, whereas the errors for angled lines show more variability. This discrepancy suggests that the angle of the line relative to the camera plays a significant role in measurement accuracy. These insights aid in understanding the performance of our measurement system under different conditions and guide potential improvements for more accurate

results.

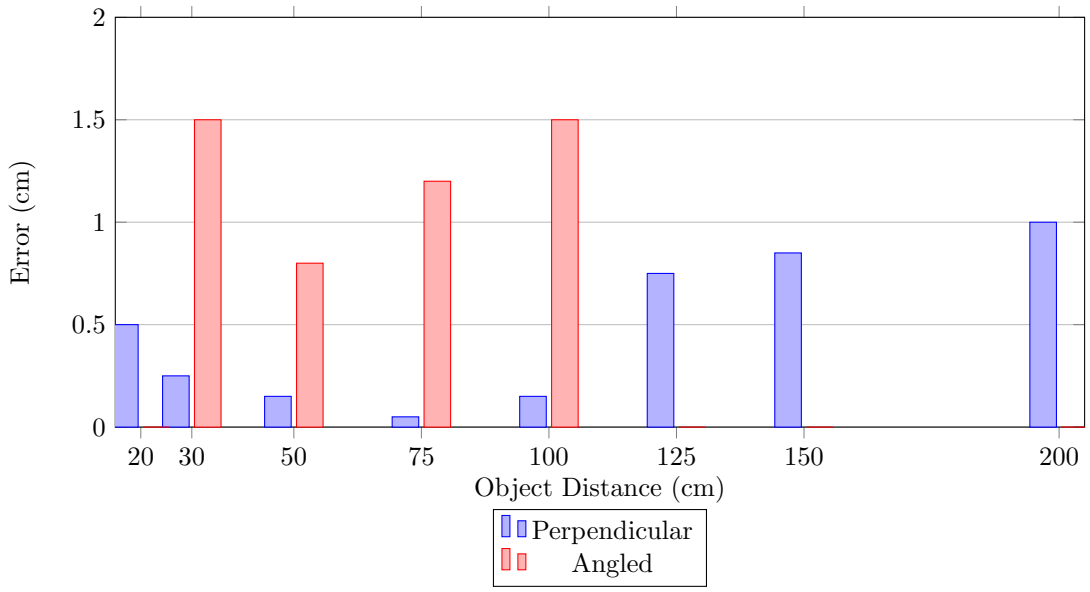



Figure 67: Error in length measurement for lines perpendicular and angled to the camera.

### 8.5.4 Discussion of Depth and Size Estimation

In the discussion of the testing and results, it's crucial to interpret the findings within the context of the system's capabilities and limitations. The depth estimation testing revealed promising results, with the system accurately estimating distances up to 2 meters with minimal error, particularly within the range of 30cm to 100cm. However, as distances increased beyond 125cm, slight differences showed, indicating a higher error rate. Similarly, the size measurement testing demonstrated consistent performance in measuring the lengths of lines perpendicular to the camera, with errors ranging from 0.1cm to 0.6cm. Nonetheless, challenges arose when measuring angled lines, especially at shorter distances, resulting in errors of up to 1.8cm. These findings underscore the importance of considering factors such as camera resolution, calibration accuracy, and object orientation when evaluating the system's performance. While the results indicate overall effectiveness, it is important to acknowledge the limitations of the system, including potential inaccuracies in depth estimation at longer distances and the impact of object orientation on size measurement precision. These insights provide valuable guidance for refining the system's algorithms and addressing key areas for improvement to enhance overall performance and reliability in real-world applications.

# 9 Future Work

## 9.1 Improvements

### 9.1.1 Camera Sensors

One area where experimentation could have been performed is the model of camera sensor used. The choice to use the OV2640 was made early in the project, as it strikes a balance between resolution and power consumption. However, thanks to the development of the MOSFET board, the power to the camera sensors is completely cut off during deep sleep, making their deep sleep power draw a non-factor. For this reason, higher resolution cameras could provide noticeable improvement in image quality and thus accuracy of results, for little trade-off.

## 9.2 Firmware Improvements

As with every project involving firmware, it is nearly impossible to think of every case that can cause the hardware to malfunction. More testing can be done with more edge-cases that will lead to more bugs surfacing, and therefore by fixing them, a more robust system. Some minor currently known bugs is that the system must be rebooted to exit SD saving mode, even when turned off from the front-end. The ESP32s also only work over a 2.4 GHz network, therefore a separate module could be implemented to allow it to use the 5 GHz band as well. Lastly, in a market product, there would only be one ESP32 chip controlling everything, which would limit power consumption and the need for cross-microcontroller synchronisation from the server.

### 9.2.1 Enclosure

Due to limitations in the available equipment and resources, a few restrictions were placed onto the enclosure. Firstly, as previously mentioned, rubber gaskets were unavailable, limiting the ingress protection of the device. Similarly, protective glass covers for the camera sensors were difficult to source, and would be one of the biggest priorities to amend in a future revision. Lastly, the magnetic closing mechanism would be swapped out for screws in a version of the device meant for deployment, exchanging the convenience in testing and debugging for superior ingress protection.

### 9.2.2 Security

Depending on the chosen application for the device, the importance of transmission security varies. Given that the device communicates with the host server over Wi-Fi, several potential vulnerabilities could be exploited. One type of attack that the device's transmissions could be vulnerable to is a 'Man in the middle' attack, where a bad actor listens in on the transmissions and attempts to decrypt the data in the present or in the future. Another potential future security measure would be encrypting the binaries on the actual chip itself, therefore making it hard to access the codebase to an outside physical attacker. While security has been a consideration throughout the project, it was not a major one and did not dictate the outcome of any decisions. This means that there is potential for additional security measures to be investigated and implemented in the future.

### 9.2.3 Front-End Authentication

Following from transmission, security could also be implemented at the front-end level. Currently, the front-end is hosted on a cloud service, with a publicly accessible URL. With more time, an entire authorization and authentication procedure could be set up, so that usernames and passwords are stored securely in a database. Users would then log in and be able to access the various controls the site provides. Bcrypt is an example of a JavaScript library that assists with hashing passwords, a potential secure implementation would involve bcrpyt being used in conjunction with a database to store the hashed information. Alternatively, third-party authorization could be utilised, such as OAuth 2.0 [75]. With this, any service that supports OAuth can be used to log in to the website, such as Google, Facebook, or Microsoft. The added benefit is the lack of developer overhead in managing a secure authorization protocol, as the work complexity is reduced. Additionally, the user experience is improved as there is no need to remember a new set of credentials.

## 9.3 Expansions

### 9.3.1 Pattern Projection

One possible expansion to the hardware and image processing of the project would be the transition to active stereoscopy, where a particular kind of random light pattern is projected onto the surface of the monitored object. This approach can yield great improvements in results, as 'texture' is projected onto the monitored object, providing more visual information to process. The improvement in results can be especially impressive in poor lighting conditions, as the projected pattern also 'lights up' the monitored object.

This approach does present two drawbacks however. Firstly, the power consumption of the device during the active state would increase; this might not be a big issue however. Secondly, the type of random light projection must be chosen carefully , and might even need to vary depending on the application the device is used for. Image processing will likely need to be more complex to account for the random light pattern. A minor consideration is the physical aspect of a light projector, as a partial or complete redesign of the device enclosure might be necessitated.

### 9.3.2 Artificial Intelligence Image Analysis

Artificial Intelligence is advancing at an astounding rate and is being applied to many different sectors. Artificial intelligence is already quite well established in image processing and image recognition. This could be leveraged to have a trained neural network automatically analyse the captured images to detect changes in the observed object, assisting the operator responsible for analysing the images, and minimising the room for errors.

### 9.3.3 Multiple Accounts

In the future the backend could be extended to host multiple accounts, authentication is discussed in the previous Section 9.2.3, however multiple accounts could also be used so that there would be an account linked for each WULPSC device. After logging in to an account in the website, it would show the images

for the linked device. This way it would be possible to mass produce WULPSC devices if they were being commercialised. There would also be separate stereo parameters for each account since the calibration would be different for every device.

### 9.3.4 Auto Calibration

Currently all calibration is done manually by taking the images directly from the microcontroller and putting them into MATLAB to perform the calibration. For the future there should be a way through the website to calibrate the cameras. This would work by having a button to click to re-calibrate and then the website would give instructions for the procedure to calibrate with picture examples. The user would hold the checkerboard in front of the cameras, the picture would be taken and then the user could decide to keep that picture if it was correct or discard and take another one. After going through the whole procedure and taking all the photos, the stereo parameters would be created and used. This would benefit the user a lot making it more convenient and simple to calibrate the stereo algorithm.

### 9.3.5 Brightness Balance

Enhancing the brightness balance between the left and right images could significantly improve the disparity map's accuracy.Having uniform brightness levels across both images is important for stereo matching algorithms, as it minimizes errors caused by intensity differences. One approach to address this issue is through the Gray-Scale method [76], which involves adjusting the pixel intensities to ensure consistency between the images. Another method is the Colour method [77], where the images are converted to the HSV colour space, allowing adjustments of the brightness while preserving hue and saturation information. Those techniques, can generate an enhanced disparity map, leading to a more reliable and precise depth estimation.

# 10 Conclusion

To conclude, the aim of this project was the design and implementation, of a remote, ultra-low power WiFi, stereo camera device. The motivation for this device was the need for low-power monitoring of remote areas, such as pipes and other locations where defects may appear over time, as a clear gap existed in the relevant literature for this area.

The developed solution involved the use of two OV2640 sensors. An ESP32-CAM and FireBeetle 2 ESP32 were used, with the former as the Main Camera Controller and the latter as the Wake-Up Controller. This setup allowed for very low power consumption when in deep sleep, and a suitable enclosure was built to house all of the hardware. Communication between the system and the back-end utilised WiFi, allowing for remote processing, removing the need for power-expensive onboard processing. A user-friendly interface was developed, allowing for easy management and analysis of captured photos, along with photo scheduling functionality to aid remote inspection, and changing of the camera settings to make the system more adaptable to different kinds of situations.

Assuming an active time of 1 minute per day (2-3 captures), the battery is predicted to last for about 1.5 years, surpassing the low power objective set out at the start. Regarding distance estimation, at a baseline of 5.5cm, the system was evaluated at a range of distances, and the highest error was 6.7% at a distance of 150cm, excluding 20cm and 200cm as both are out with the specified minimum and maximum depths for the chosen baseline. Additionally, for line measurement, the highest error for objects perpendicular to the camera was 0.8 - 1.1 cm at a distance of 200cm. Furthermore, the highest error was 1.1 - 1.8 cm for angled objects at a distance of 100cm.

Overall, all major project objectives outlined in the statement of intent have been completed. The objectives being: selecting an appropriate microcontroller with WiFi and camera, develop optimised firmware to enable the user to send data to the microcontroller via the internet, develop optimised firmware to enable the microcontroller to send data to the host computer via the internet, provision the host device with WiFi connectivity, implement firmware to capture synchronised images from the two cameras, design an algorithm to fuse the two images into a single homographic image, creating a robust HTTP API backend to store the captured images and control the camera parameters, constructing a web interface for users to remotely manage the camera system and monitor the images, select an appropriate battery and evaluate the system's power consumption and lastly to evaluate 3D scene reconstruction accuracy. Two PCBs were created to house the components and allow for easy connection of the microcontroller development boards, and a casing was created to house the system, therefore completing all minor objectives as well. An optional objective was also completed where WiFi provisioning was also implemented by changing the WiFi credentials text file on the MCC and by calling `/reset` on the WUC to boot to Smart Config.

Discussed in detail in Section 9, several avenues exist for future expansion of the project's scope, like implementing neural networks and deep learning for automatic defect detection. Additionally, research could be carried out into integrating active stereoscopy to improve disparity map quality, along with the use of higher resolution sensors, as their increased power draw would be negligible for most of the run-time due to the MOSFET board.

# References


[1] L. Puglia, M. Vigliar, and G. Raiconi, "Real-time low-power FPGA architecture for stereo vision," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 11, pp. 1307–1311, Nov. 2017, Conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs, ISSN: 1558-3791. DOI: 10.1109/TCSII.2017.2691675. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7893762 (visited on 04/21/2024).

[2] E. Sola-Thomas and M. H. Imtiaz, "An ultra-low-power design of smart wearable stereo camera," in *SoutheastCon 2021*, ISSN: 1558-058X, Mar. 2021, pp. 1–8. DOI: 10.1109/SoutheastCon45413.2021.9401833. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9401833 (visited on 04/21/2024).

[3] B. Jian, Y. Ling, X. Liu, and Y. Xie, "Design of a stereo monitoring system based on wireless sensor networks," *Journal of Physics: Conference Series*, vol. 1802, no. 3, p. 032 042, Mar. 2021, Publisher: IOP Publishing, ISSN: 1742-6596. DOI: 10.1088/1742-6596/1802/3/032042. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1802/3/032042 (visited on 04/21/2024).

[4] N. Kurniati, R.-H. Yeh, and J.-J. Lin, "Quality inspection and maintenance: The framework of interaction," *Procedia Manufacturing*, Industrial Engineering and Service Science 2015, IESS 2015, vol. 4, pp. 244–251, Jan. 1, 2015, ISSN: 2351-9789. DOI: 10.1016/j.promfg.2015.11.038. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978915011543 (visited on 04/21/2024).

[5] F. Orellana, P. J. V. D'Aranno, S. Scifoni, and M. Marsella, "SAR interferometry data exploitation for infrastructure monitoring using GIS application," *Infrastructures*, vol. 8, no. 5, p. 94, May 2023, Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2412-3811. DOI: 10.3390/infrastructures8050094. [Online]. Available: https://www.mdpi.com/2412-3811/8/5/94 (visited on 04/21/2024).

[6] J. Zhang and D. Li, *Method of surface defect detection for agricultural machinery parts based on image recognition technology*. Mar. 23, 2023. DOI: 10.21203/rs.3.rs-2725077/v1.

[7] "Camera lens distortion - image engineering." (), [Online]. Available: https://www.image-engineering.de/library/image-quality/factors/1062-distortion (visited on 04/21/2024).

[8] S. Giancola, M. Valenti, and R. Sala, "State-of-the-art devices comparison," in Jun. 2018, pp. 29–39, ISBN: 978-3-319-91760-3. DOI: 10.1007/978-3-319-91761-0_3.

[9] "Epipolar (stereo) rectification." (Nov. 13, 2015), [Online]. Available: https://web.archive.org/web/20151113084918/http://www.diegm.uniud.it/fusiello/demo/rect/ (visited on 04/21/2024).

[10] D. Oram, "Rectification for any epipolar geometry," *Proc. British Machine Vision Conf*, Apr. 2002. DOI: 10.5244/C.15.67.

[11] A. S. Saragih, F. Aditya, and W. Ahmed, "Defect identification and measurement using stereo vision camera for in-line inspection of pipeline," in *2022 Advances in Science and Engineering Technology International Conferences (ASET)*, Feb. 2022, pp. 1–5. DOI: 10.1109/ASET53988.2022.9735082. [Online]. Available: https://ieeexplore.ieee.org/document/9735082 (visited on 04/19/2024).

[12] "Stereo matching - an overview — ScienceDirect topics." (), [Online]. Available: https://www.sciencedirect.com/topics/engineering/stereo-matching (visited on 04/20/2024).

[13] Z. Li, X. Liu, N. Drenkow, *et al.*, "Revisiting stereo depth estimation from a sequence-to-sequence perspective with transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, pp. 6177–6186, ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.00614. [Online]. Available: https://ieeexplore.ieee.org/document/9711118/ (visited on 04/20/2024).

[14] H. Hirschmüller, "Semi-global matching – motivation, developments and applications,"

[15] V. H. Borisagar and M. A. Zaveri, "Disparity map generation from illumination variant stereo images using efficient hierarchical dynamic programming," *The Scientific World Journal*, vol. 2014, pp. 1–12, 2014, ISSN: 2356-6140, 1537-744X. DOI: 10.1155/2014/513417. [Online]. Available: http://www.hindawi.com/journals/tswj/2014/513417/ (visited on 04/20/2024).

[16] J. Kern, J. Silva, and C. Urrea, "Development of an algorithm that allows improving disparity maps in environments contaminated with suspended particles," *International Journal of Electrical and Electronics Engineering*, vol. 10, no. 4, pp. 169–174, Apr. 30, 2023, ISSN: 23488379. DOI: 10.14445/23488379/IJEEE-V10I4P117. [Online]. Available: https://www.internationaljournalssrg.org/IJEEE/paper-details?Id=484 (visited on 04/20/2024).

[17] F. Bethmann and T. Luhmann, "SEMI-GLOBAL MATCHING IN OBJECT SPACE," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-3-W2, pp. 23–30, Mar. 10, 2015, Conference Name: PIA15+HRIGI15 – Joint ISPRS conference (Volume XL-3/W2) - 25–27 March 2015, Munich, Germany Publisher: Copernicus GmbH, ISSN: 1682-1750. DOI: 10.5194/isprsarchives-XL-3-W2-23-2015. [Online]. Available: https://isprs-archives.copernicus.org/articles/XL-3-W2/23/2015/ (visited on 04/20/2024).

[18] "Stereo disparity using semi-global block matching - MATLAB & simulink - MathWorks united kingdom." (), [Online]. Available: https://uk.mathworks.com/help/visionhdl/ug/stereoscopic-disparity.html (visited on 04/20/2024).

[19] "Weighted least square filter for improving the quality of depth map on FPGA," *International Journal of Networking and Computing*,

[20] "OpenCV: Filters." (), [Online]. Available: https://docs.opencv.org/3.4/da/d17/group__ximgproc_filters.html (visited on 04/20/2024).

[21] "Similar triangles." (), [Online]. Available: https://www.varsitytutors.com/hotmath/hotmath_help/topics/similar-triangles (visited on 04/21/2024).

[22] "Aa-similar-triangle-1622102459.png (PNG image, 717 × 607 pixels)." (), [Online]. Available: https://d138zd1ktt9iqe.cloudfront.net/media/seo_landing_files/aa-similar-triangle-1622102459.png (visited on 04/21/2024).

[23] Y. Benkhoui, T. El Korchi, and L. Reinhold, "UAS-based crack detection using stereo cameras: A comparative study," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, Atlanta, GA, USA: IEEE, Jun. 2019, pp. 1031–1035, ISBN: 978-1-72810-333-4. DOI: 10.1109/ICUAS.2019.8798311. [Online]. Available: https://ieeexplore.ieee.org/document/8798311/ (visited on 04/19/2024).

[24] A. Gunatilake, L. Piyathilaka, A. Tran, V. K. Vishwanathan, K. Thiyagarajan, and S. Kodagoda, "Stereo vision combined with laser profiling for mapping of pipeline internal defects," *IEEE Sensors Journal*, vol. 21, no. 10, pp. 11 926–11 934, May 15, 2021, ISSN: 1530-437X, 1558-1748, 2379-9153. DOI: 10.1109/JSEN.2020.3040396. [Online]. Available: https://ieeexplore.ieee.org/document/9270040/ (visited on 04/19/2024).

[25] A. Dawda and M. Nguyen, "Defects detection in highly specular surface using a combination of stereo and laser reconstruction," in *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, Wellington, New Zealand: IEEE, Nov. 25, 2020, pp. 1–6, ISBN: 978-1-72818-579-8. DOI: 10.1109/IVCNZ51579.2020.9290660. [Online]. Available: https://ieeexplore.ieee.org/document/9290660/ (visited on 04/19/2024).

[26] P. Huynh, R. Ross, A. Martchenko, and J. Devlin, "Anomaly inspection in sewer pipes using stereo vision," in *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, Kuala Lumpur, Malaysia: IEEE, Oct. 2015, pp. 60–64, ISBN: 978-1-4799-8996-6. DOI: 10.1109/ICSIPA.2015.7412164. [Online]. Available: http://ieeexplore.ieee.org/document/7412164/ (visited on 04/19/2024).

[27] W. Koodtalang, T. Sangsuwan, and B. Noppakaow, "A design of automated inspections of both shape and height simultaneously based on stereo vision and PLC," in *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, Oct. 2018, pp. 1290–1294. [Online]. Available: https://ieeexplore.ieee.org/document/8571609 (visited on 04/20/2024).

[28] A. Amjad, R. Amjad, A. Gavrielides, K. Gkogkalatze, and Y. Michael, "Internet of things (iot) ultra low power wi-fi stereo camera," Univeristy of Strathclyde, Tech. Rep., 2024.

[29] "ESP32-CAM ESP32 development board with camera," Hobby Components. (), [Online]. Available: https://hobbycomponents.com/other-dev-boards/1129-esp32-cam-esp32-development-board-with-camera (visited on 04/17/2024).

[30] *Esp32-cam datasheet*. [Online]. Available: https://loboris.eu/ESP32/ESP32-CAM%20Product%20Specification.pdf.

[31] "ESP32-CAM video streaming and face recognition with arduino IDE — random nerd tutorials." (Dec. 10, 2019), [Online]. Available: https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/ (visited on 04/17/2024).

[32] *Espressif/esp-idf*, original-date: 2016-08-17T10:40:35Z, Apr. 21, 2024. [Online]. Available: https://github.com/espressif/esp-idf (visited on 04/21/2024).

[33] *FireBeetle 2 ESP32-E IoT Microcontroller (Supports Wi-Fi & Bluetooth)*, en. [Online]. Available: https://www.dfrobot.com/product-2195.html (visited on 04/13/2024).

[34] "OV2640: Specs, camera, datasheet & alternative (2022 report)," Arducam. (), [Online]. Available: https://www.arducam.com/ov2640/ (visited on 04/19/2024).

[35] "IDTQS3vh257 by renesas electronics corporation datasheet — DigiKey." (), [Online]. Available: https://www.digikey.in/en/htmldatasheets/production/272426/0/0/1/idtqs3vh257 (visited on 04/21/2024).

[36] "Maix binocular camera - waveshare wiki." (), [Online]. Available: https://www.waveshare.com/wiki/Maix_Binocular_Camera (visited on 04/19/2024).

[37] *AO3401.pdf*. [Online]. Available: https://aosmd.com/sites/default/files/res/datasheets/AO3401.pdf (visited on 04/21/2024).

[38] *ONSM-S-A0013669918-1.pdf*. [Online]. Available: https://4donline.ihs.com/images/VipMasterIC/IC/ONSM/ONSM-S-A0013670144/ONSM-S-A0013669918-1.pdf?hkey=6D3A4C79FDBF58556ACFDE234799DDF (visited on 04/21/2024).

[39] *VP0109C082313.pdf*. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/VP0109C082313.pdf (visited on 04/21/2024).

[40] *0900766b814b7a89.pdf*. [Online]. Available: https://docs.rs-online.com/12e8/0900766b814b7a89.pdf (visited on 04/21/2024).

[41] "2447-3034-20-520 — ansmann 3.635v lithium-ion rechargeable battery pack, 10.5ah - pack of 1 — RS." (), [Online]. Available: https://uk.rs-online.com/web/p/rechargeable-battery-packs/1445692?gb=s (visited on 04/21/2024).

[42] Y.-D. Youngdungpo-Gu, "PRODUCT SPECIFICATION," [Online]. Available: https://www.nkon.nl/sk/k/Specification%20INR18650MJ1%2022.08.2014.pdf.

[43] *PRODUCT SPECIFICATION*. [Online]. Available: https://www.nkon.nl/sk/k/Specification%20INR18650MJ1%2022.08.2014.pdf.

[44] *Diymore 2pcs ESP32-CAM-MB WiFi Bluetooth Development Board, ESP32 Dual-core Wireless Development Board with OV2640 Camera TF Card Module : Amazon.co.uk: Electronics & Photo*. [Online]. Available: https://www.amazon.co.uk/diymore-ESP32-CAM-MB-Bluetooth-Development-Dual-core/dp/B08P1NMPLL (visited on 04/13/2024).

[45] *Get Started - ESP32 - — ESP-IDF Programming Guide v5.1.3 documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1.3/esp32/get-started/index.html (visited on 04/13/2024).

[46] *Espressif IoT Development Framework Style Guide - ESP32 - — ESP-IDF Programming Guide latest documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/contribute/style-guide.html# (visited on 04/13/2024).

[47] *Product Overview - ESP32 - — ESP Hardware Design Guidelines latest documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-hardware-design-guidelines/en/latest/esp32/product-overview.html (visited on 04/13/2024).

[48] *Esp-idf/examples at master · espressif/esp-idf*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples (visited on 04/13/2024).

[49] *Esp WiFi station*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/wifi/getting_started/station (visited on 04/21/2024).

[50] *Esp-idf/examples/wifi/smart_config at master · espressif/esp-idf*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/wifi/smart_config (visited on 04/13/2024).

[51] *Esp http server*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/protocols/http_server/simple (visited on 04/21/2024).

[52] *Esp-idf/examples/storage/nvs_rw_blob at master · espressif/esp-idf*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/storage/nvs_rw_blob (visited on 04/13/2024).

[53] *Esp SD SPI*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/storage/sd_card/sdspi (visited on 04/21/2024).

[54] *Esp-idf/examples/peripherals/gpio at master · espressif/esp-idf*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/peripherals/gpio (visited on 04/21/2024).

[55] *Esp-idf deep sleep*, en. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/system/deep_sleep (visited on 04/21/2024).

[56] *Espressif/esp32-camera*, original-date: 2018-11-13T10:08:16Z, Apr. 2024. [Online]. Available: https://github.com/espressif/esp32-camera (visited on 04/13/2024).

[57] ymich9963, *Ymich9963/WULPSC*, original-date: 2023-11-24T18:41:31Z, Apr. 2024. [Online]. Available: https://github.com/ymich9963/WULPSC (visited on 04/13/2024).

[58] *Logging library - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/log.html#logging-library (visited on 04/14/2024).

[59] *Error Code and Helper Functions - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/esp_err.html#error-code-and-helper-functions (visited on 04/14/2024).

[60] EspressifApp, *EspressifApp/EsptouchForAndroid*, original-date: 2015-04-22T06:18:45Z, Apr. 2024. [Online]. Available: https://github.com/EspressifApp/EsptouchForAndroid (visited on 04/14/2024).

[61] *Sleep Modes - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html#sleep-modes (visited on 04/14/2024).

[62] *xEventGroupWaitBits() - Wait for a bit (flag) or bits in an FreeRTOS event group*, en-US. [Online]. Available: https://www.freertos.org/xEventGroupWaitBits.html (visited on 04/18/2024).

[63] *Ov2640 green tint - Google Search*. [Online]. Available: https://www.google.com/search?client=firefox-b-d&q=ov2640+green+tint (visited on 04/19/2024).

[64] *Understanding YUV data formats*, en-GB, 2016. [Online]. Available: https://www.flir.co.uk/support-center/iis/machine-vision/knowledge-base/understanding-yuv-data-formats/ (visited on 04/19/2024).

[65] H. Sankar, *What is C extern Keyword?* en, Nov. 2022. [Online]. Available: https://www.scaler.com/topics/c-extern/ (visited on 04/20/2024).

[66] *Esp-idf/components/json/README at master · espressif/esp-idf*, en. [Online]. Available: https://github.com/espressif/esp-idf/blob/master/components/json/README (visited on 04/20/2024).

[67] *FAT Filesystem Support - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/fatfs.html#using-fatfs-with-vfs (visited on 04/21/2024).

[68] A. Hovhannisyan, *CRLF vs. LF: Normalizing Line Endings in Git — Aleksandr Hovhannisyan*, en-US, Apr. 2021. [Online]. Available: https://www.aleksandrhovhannisyan.com/blog/crlf-vs-lf-normalizing-line-endings-in-git/ (visited on 04/21/2024).

[69] *Fly.io Resource Pricing · Fly Docs*. [Online]. Available: https://fly.io/docs/about/pricing/ (visited on 04/21/2024).

[70] *Manage Clusters - MongoDB Atlas*. [Online]. Available: https://www.mongodb.com/docs/atlas/manage-clusters/ (visited on 04/21/2024).

[71] *Call MATLAB from Python - MATLAB & Simulink - MathWorks United Kingdom*. [Online]. Available: https://uk.mathworks.com/help/matlab/matlab-engine-for-python.html?s%5C_tid=CRUX%5C_lftnav (visited on 04/19/2024).

[72] "Making your UX life easier with the MoSCoW," The Interaction Design Foundation. (Nov. 18, 2015), [Online]. Available: https://www.interaction-design.org/literature/article/making-your-ux-life-easier-with-the-moscow (visited on 04/19/2024).

[73] "Pros and cons of single-page applications," Spiceworks Inc. (), [Online]. Available: https://www.spiceworks.com/tech/devops/articles/what-is-single-page-application/ (visited on 04/19/2024).

[74] "Lens, focal length and stereo baseline calculator – allied vision technologies: Nerian." (), [Online]. Available: https://nerian.com/support/calculator/ (visited on 04/19/2024).

[75] D. Hardt, "The OAuth 2.0 authorization framework," Internet Engineering Task Force, Request for Comments RFC 6749, Oct. 2012, Num Pages: 76. DOI: 10.17487/RFC6749. [Online]. Available: https://datatracker.ietf.org/doc/rfc6749 (visited on 04/19/2024).

[76] L. Maurya, V. Lohchab, P. Kumar Mahapatra, and J. Abonyi, "Contrast and brightness balance in image enhancement using cuckoo search-optimized image fusion," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 7247–7258, Oct. 1, 2022, ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2021.07.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821001750 (visited on 04/21/2024).

[77] "FotoForensics." (), [Online]. Available: https://fotoforensics.com/tutorial.php?tt=coloradjustment (visited on 04/21/2024).

# Appendix

## Code

Listing 1: Stereo Algorithm Code

```
import cv2
import numpy as np
import math
# import matlab.engine

# Global variables
start_point = None
end_point = None
drawing = False

# Mouse callback function
def draw_bbox(event, x, y, flags, param):
    global start_point, end_point, drawing

    if event == cv2.EVENT_LBUTTONDOWN:
        if not drawing:
            start_point = (x, y)
            drawing = True
        else:
            end_point = (x, y)
            drawing = False

# Function to apply mask to disparity map
def apply_mask(disparity_map, mask):
    masked_disparity = cv2.bitwise_and(disparity_map, disparity_map, mask=mask)
    return masked_disparity

def getDisparityMap(left_nice, right_nice):
    leftMatcher = cv2.StereoSGBM_create(
    minDisparity=min_disp,
    numDisparities=num_disp,
    blockSize=window_size,
    P1=8 * 3 * window_size**2,
    P2=32 * 3 * window_size**2,
    disp12MaxDiff=1,
    uniquenessRatio=15,
    speckleWindowSize=0,
    speckleRange=2,
    preFilterCap=63,
    mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY)

    right_matcher = cv2.ximgproc.createRightMatcher(leftMatcher)
    left_disp = leftMatcher.compute(left_nice, right_nice)
    right_disp = right_matcher.compute(right_nice,left_nice)

```

```
    # Now create DisparityWLSFilter
    wls_filter = cv2.ximgproc.createDisparityWLSFilter(leftMatcher)
    wls_filter.setLambda(lmbda)
    wls_filter.setSigmaColor(sigma)
    filtered_disp = wls_filter.filter(left_disp, left_nice, disparity_map_right=right_disp)

    disparity_map = cv2.normalize(src=filtered_disp, dst=None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
      dtype=cv2.CV_8UC1)
    return disparity_map

def getDistance(disparity):
    distance = (avgFocalLength) * (baseline / (disparity / 4.0))
    return distance


def getWidth(boxPoints, distance):
    pixelWidth = boxPoints[1][0] - boxPoints[0][0]
    return abs((distance * pixelWidth) / avgFocalLength)


## Changed height condition
def getHeight(boxPoints, distance):
    if disparity_diff < 10:
        pixelHeight = boxPoints[2][1] - boxPoints[1][1]
    else:
        pixelHeight = boxPoints[1][1] - boxPoints[0][1]
    return abs((distance * pixelHeight) / avgFocalLength)

# Variables Needed
baseline = 0.055  # meters
lmbda = 8000  # 80000
sigma = 2.0

Focal_Lengths = [576.933515171272, 577.044426870670]
# Focal_Lengths = [602.0259, 602.4632]

avgFocalLength = (Focal_Lengths[0] + Focal_Lengths[1]) / 2
window_size = 1
min_disp = 0
nDispFactor = 4
num_disp = 16 * nDispFactor - min_disp

# Rectification using Matlab Engine
# eng = matlab.engine.start_matlab()
# eng.eval('load("stereoParams_11_04.mat")', nargout=0)
# eng.workspace['I1'] = eng.imread("left_testing/img_1.jpg")
# eng.workspace['I2'] = eng.imread("right_testing/img_2.jpg")
# t = eng.eval('rectifyStereoImages(I1, I2, stereoParams_11_04)', nargout=3)
# eng.imwrite(t[0], "left_test.png", nargout=0)
# eng.imwrite(t[1], "right_test.png", nargout=0)

```

```
# Load left image and create a window
left_image = cv2.imread('pap_l.png')
right_image = cv2.imread("pap_r.png")
left_image_gray = cv2.cvtColor(left_image, cv2.COLOR_BGR2GRAY)
right_image_gray = cv2.cvtColor(right_image, cv2.COLOR_BGR2GRAY)

# Median Blur
left_image_gray = cv2.medianBlur(left_image_gray, 11)
right_image_gray = cv2.medianBlur(right_image_gray, 11)

cv2.namedWindow('Left Image')
cv2.setMouseCallback('Left Image', draw_bbox)

# Load disparity map
disparity_map = getDisparityMap(left_image_gray, right_image_gray)
disparity_map_rgb = cv2.applyColorMap(disparity_map, cv2.COLORMAP_JET)

while True:
    # Display left image
    img = left_image.copy()

    if start_point is not None and end_point is not None:
        # Draw the bounding box
        cv2.rectangle(img, start_point, end_point, (0, 255, 0), 2)

        # Create a mask from the bounding box
        mask = np.zeros_like(disparity_map)
        cv2.rectangle(mask, start_point, end_point, 255, -1)

        # Apply mask to disparity map
        masked_disparity = apply_mask(disparity_map, mask)

        # Store coordinates of bounding box in format (top-left, top-right, bottom-right, bottom-left)
        x_tl, y_tl = start_point
        x_br, y_br = end_point
        box_coordinates = [(x_tl, y_tl), (x_br, y_tl), (x_br, y_br), (x_tl, y_br)]

        non_zero_elements = masked_disparity[masked_disparity != 0]  # Extract non-zero elements
        disparity = non_zero_elements.mean()

        disparity_min = non_zero_elements.min()
        disparity_max = non_zero_elements.max()
        disparity_diff = disparity_max - disparity_min

        # Condition for line being perpedincular to the camera or not
        if disparity_diff < 15:
            distance = getDistance(disparity)
            width = getWidth(box_coordinates, distance)
            height = getHeight(box_coordinates, distance)
            length_est = math.sqrt(width**2 + height**2)
            total_width = 0
```

```
147
148
149         ## Condition for when disparity difference is higher than 10 (Changed)
150         else:
151             # Split bounding box into 5 sections horizontally and 5 sections vertically
152             horizontal_sections = np.linspace(start_point[0], end_point[0], 6).astype(int)
153             vertical_sections = np.linspace(start_point[1], end_point[1], 6).astype(int)
154
155             section_widths = []
156             section_heights = []
157             total_width = 0
158             total_height = 0
159
160             for i in range(5):
161                 for j in range(5):
162                     section_start_x = horizontal_sections[i]
163                     section_end_x = horizontal_sections[i+1]
164
165                     section_start_y = vertical_sections[j]
166                     section_end_y = vertical_sections[j+1]
167
168                     # Define section bounding box
169                     section_bbox = [(section_start_x, section_start_y), (section_end_x, section_end_y)]
170
171                     # Apply mask to disparity map for section
172                     section_mask = np.zeros_like(disparity_map)
173                     cv2.rectangle(section_mask, section_bbox[0], section_bbox[1], 255, -1)
174                     section_masked_disparity = apply_mask(disparity_map, section_mask)
175
176                     # Calculate average disparity for section
177                     section_non_zero_elements = section_masked_disparity[section_masked_disparity != 0]
178                     section_disparity = section_non_zero_elements.mean()
179
180                     # Calculate distance for section
181                     section_distance = getDistance(section_disparity)
182
183                     # Calculate width for section
184                     section_width = getWidth(section_bbox, section_distance)
185                     section_widths.append(section_width)
186
187                     total_width += section_width
188
189                     # Calculate height for section
190                     section_height = getHeight(section_bbox, section_distance)
191                     section_heights.append(section_height)
192
193                     total_height += section_height
194
195                     width = total_width/5
196                     height = total_height/5
197                     length_est = (math.sqrt(width**2 + height**2))
```

```
198
199                 # Display bounding box for section
200                 cv2.rectangle(img, section_bbox[0], section_bbox[1], (255, 0, 0), 2)
201
202                 distance = 0
203
204
205         # Create a new image to display distance and overlay text
206         results_image = np.zeros_like(img)
207         cv2.putText(results_image, f'Distance: {distance*100:.2f} cm', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
208                 (255, 255, 255), 2)
209         cv2.putText(results_image, f'Width: {width*100:.2f} cm', (10, 190), cv2.FONT_HERSHEY_SIMPLEX, 1,
210             (255, 255, 255), 2)
211         cv2.putText(results_image, f'Height: {height*100:.2f} cm', (10, 230), cv2.FONT_HERSHEY_SIMPLEX, 1,
212             (255, 255, 255), 2)
213         cv2.putText(results_image, f'Length: {length_est*100:.2f} cm', (10, 270), cv2.FONT_HERSHEY_SIMPLEX, 1,
214             (255, 255, 255), 2)
215         cv2.putText(results_image, f'Disparity Difference: {disparity_diff:.2f} pxs', (10, 310), cv2.
        FONT_HERSHEY_SIMPLEX, 1,
216         (255, 255, 255), 2)
217
218         # Display the distance image
219         cv2.imshow('Results', results_image)
220
221     cv2.imshow('Left Image', img)
222     cv2.imshow('Disparity Mape', disparity_map_rgb)
223
224     key = cv2.waitKey(1)
225     if key == 27:  # ESC key
226         break
227
228 cv2.destroyAllWindows()
```

Listing 2: WUC Code - main.c

```
#include "wifi.h"
#include "http.h"
#include "wuc_nvs.h"
#include "wuc_config.h"

/* Tag used for logging to terminal through the serial interface */
static const char *TAG = "WULPSC – Wake Up Module";

/* Default initialisation for the system configuration */
wuc_config_t wuc_config = {
    .exit               = false,
    .reset              = false,
    .sleep_time_sec     = 86400,    /* 1 day default sleep time= 24 * 60 * 60 * 10^6 microseconds */
    .active_time_sec    = 60,       /* 1 minute default active time */
    .ssid               = "",
    .pswd               = "",
    .stored_creds       = false,
};

void app_main(void)
{
    ESP_LOGI(TAG, "———————WUC Booted———————");

    /* Variable for function returns */
    esp_err_t ret;
    esp_err_t wifi_ret;

    /* HTTP Server handle */
    httpd_handle_t server;


    /* Initialise the Non Volatile Storage, required for WiFi */
    ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
        if(ret != ESP_OK) {
            ESP_LOGE(TAG, "Error with NVS initialisation");
            return;
        }
    }

    /* Check the NVS for credentials */
    check_nvs();

/* Used for testing the WUC, set the credentials in wifi.h*/
#if TEST_CRED
    memcpy(wuc_config.ssid, ESP_WIFI_SSID, sizeof(ESP_WIFI_SSID));
    memcpy(wuc_config.pswd, ESP_WIFI_PASS, sizeof(ESP_WIFI_PASS));
    wuc_config.stored_creds = true;
```

```
#endif


    /**
     * If stored credentials are detected, initialise the WiFi with them,
     * else start smart config to get the credentials
     * */
    if(wuc_config.stored_creds){
        wifi_ret = wifi_init();
    } else {
        wifi_ret = wifi_init_sc();

        /* Poll for smart config to finish */
        while(strcmp(wuc_config.ssid, "") == 0 && strcmp(wuc_config.pswd, "") == 0){vTaskDelay(10/
       portTICK_PERIOD_MS);}

        /* Store the new credentials */
        store_creds_to_nvs();
    }

    /* Setup GPIO pin for MMC power-up */
    setup_mcc_power_switch();

    /* If WiFi initialisation and connection was succesful */
    if (wifi_ret == ESP_OK) {
        /* Power-up MCC */
        mcc_powerup();

        /* Start the HTTP server */
        server = start_webserver();
        if (server == NULL) {
            ESP_LOGE(TAG,"Error Server is NULL");
            return;
        }

        /* Main server while-loop, call /exit to enter deep sleep or /reset to reset the WiFi credentials */
        ESP_LOGI(TAG, "Entering webserver");
        while (server) {
            vTaskDelay(10/portTICK_PERIOD_MS);
            if (wuc_config.exit) {
                ESP_LOGI(TAG, "Server Stopping");
                stop_webserver(server);
                server = NULL;
            }
            if (wuc_config.reset) {
                nvs_flash_erase();
                esp_restart();
            }
        }
    } else {
        /* If no WiFi connection, still allow the MCC to power up for partial monitoring */
```

```
101         ESP_LOGW(TAG, "WiFi not succesfully connected, turn on MCC based on the active time...");
102         mcc_powerup();
103         vTaskDelay(wuc_config.active_time_sec*1000/portTICK_PERIOD_MS);
104     }
105
106     /* Setup deep sleep */
107     deep_sleep_setup();
108
109     /* Power down the MCC */
110     mcc_powerdown();
111
112     /* Enter deep sleep */
113     ESP_LOGI(TAG,"Entering deep sleep\n");
114     esp_deep_sleep_start();
115
116     ESP_LOGI(TAG, "Ended app_main()");
117     return;
118 }
```

Listing 3: WUC Code - http.h

```
#pragma once

#include <string.h>
#include "sys/param.h" // contains MIN() macro
#include "esp_http_server.h"
#include "esp_log.h"
#include "wuc_config.h"

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief GET Handler, only used for testing
 *
 * @param req HTTP Request
 *
 * @return ESP_OK on success
*/
esp_err_t get_handler(httpd_req_t *req);

/**
 * @brief GET handler to notify WUC to shut down
 *
 * @param req HTTP Request
 *
 * @return ESP_OK on success
*/
esp_err_t exit_handler(httpd_req_t *req);

/**
 * @brief GET handler to notify WUC to reset the NVS to accept new credentials
 *
 * @param req HTTP Request
 *
 * @return ESP_OK on success
*/
esp_err_t reset_handler(httpd_req_t *req);

/**
 * @brief POST handler to receive the specified sleep time
 *
 * @param req HTTP Request
 *
 * @return ESP_OK on success
*/
esp_err_t set_sleep_time_handler(httpd_req_t *req);

/**
 * @brief Initialise the HTTP server
```

```
 *
 * @return Handle to the HTTP server
*/
httpd_handle_t start_webserver(void);


/**
 * @brief Call to stop the HTTP server
 *
 * @param server handle to server
*/
void stop_webserver(httpd_handle_t server);

/* Example cURL command

curl -ContentType 'text/plain' -Body "SLEEP_TIME_SECONDS" -Method Post http://IP_ADDRESS_HERE:19520/
    sleep

*/

#ifdef __cplusplus
}
#endif
```

Listing 4: WUC Code - http.c

```
#include "http.h"

const char* TAG = "WULPSC – WUC HTTP";

/* System configuration defined in main */
extern wuc_config_t wuc_config;

/* Our URI handler function to be called during GET /uri request */
esp_err_t get_handler(httpd_req_t *req)
{
    ESP_LOGI(TAG, "Entered get handler");

    /* Send a simple response to test functionality */
    const char resp[] = "URI GET Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

esp_err_t exit_handler(httpd_req_t *req)
{
    /* Set to true to exit the main server loop and enter deep sleep*/
    wuc_config.exit = true;

    /* Send response to signify it has been executed */
    const char resp[] = "Exited server loop.";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

esp_err_t reset_handler(httpd_req_t *req)
{
    /* Set reset to true to reset the system from the main loop */
    wuc_config.reset = true;

    /* Send response to signify it has been executed */
    const char resp[] = "Erased flash and restarted system.";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}


esp_err_t set_sleep_time_handler(httpd_req_t *req)
{

    /* Max character length */
    int MAX = 511;

    /* Make sure bytes dont go over max length */
    size_t recv_size = req->content_len;
    if (recv_size >= MAX){
```

```
51          ESP_LOGE(TAG, "Over %d bytes sent", MAX);
52          return ESP_FAIL;
53      }
54
55      ESP_LOGI(TAG, "Size %u bytes", recv_size);
56
57      /* Allocate the size of the received string */
58      char* content = malloc(sizeof(char) * recv_size + 1);
59
60      /* Set the received data to the content variable */
61      int ret = httpd_req_recv(req, content, recv_size);
62
63      /* Add a null terminator character to signify the end of the string */
64      content[recv_size] = '\0';
65
66      ESP_LOGI(TAG, "Data %s", content);
67      if (ret <= 0) {  /* 0 return value indicates connection closed */
68          /* Check if timeout occurred */
69          if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
70              /* In case of timeout one can choose to retry calling
71               * httpd_req_recv(), but to keep it simple, here we
72               * respond with an HTTP 408 (Request Timeout) error */
73              httpd_resp_send_408(req);
74          }
75          /* In case of error, returning ESP_FAIL will
76           * ensure that the underlying socket is closed */
77          return ESP_FAIL;
78      }
79
80      /* Set the received conent to the system config */
81      sscanf(content, "%lu", &wuc_config.sleep_time_sec);
82      ESP_LOGI(TAG, "Sleep time set to %lu", wuc_config.sleep_time_sec);
83
84      /* Send a response to */
85      const char resp[] = "Sleep parameter set";
86      httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
87      return ESP_OK;
88  }
89
90  /* GET handlers */
91  httpd_uri_t uri_get = {
92      .uri      = "/uri",
93      .method   = HTTP_GET,
94      .handler  = get_handler,
95      .user_ctx = NULL
96  };
97
98  httpd_uri_t exit_get = {
99      .uri      = "/exit",
100     .method   = HTTP_GET,
101     .handler  = exit_handler,
```

```
    .user_ctx = NULL
};

httpd_uri_t reset_get = {
    .uri      = "/reset",
    .method   = HTTP_GET,
    .handler  = reset_handler,
    .user_ctx = NULL
};

/* POST Handlers */
httpd_uri_t set_sleep_time_post = {
    .uri      = "/sleep",
    .method   = HTTP_POST,
    .handler  = set_sleep_time_handler,
    .user_ctx = NULL
};

/* Function for starting the webserver */
httpd_handle_t start_webserver(void){
    /* Generate default configuration */
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 19520;

    /* Empty handle to esp_http_server */
    httpd_handle_t server = NULL;

    /* Start the httpd server */
    if (httpd_start(&server, &config) == ESP_OK)
    {
        /* Register URI handlers */
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &exit_get);
        httpd_register_uri_handler(server, &reset_get);

        /* Register POST handlers */
        httpd_register_uri_handler(server, &set_sleep_time_post);

    }
    /* If server failed to start, handle will be NULL */
    return server;
}

void stop_webserver(httpd_handle_t server)
{
    if (server) {
        httpd_stop(server);
    }
}
```

Listing 5: WUC Code - wifi.h

```
#pragma once

#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "esp_log.h"
#include "esp_event.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_netif.h"
#include "esp_vfs_dev.h"
#include "esp_smartconfig.h"
#include "driver/uart.h"
#include "wuc_config.h"

/* Credentials used to test the WUC without the smart config process */
#define TEST_CRED 0
    #if TEST_CRED
        #define ESP_WIFI_SSID      "test-ssid"
        #define ESP_WIFI_PASS      "test-pass"
    #endif

/* Maximum retries to connect to the WiFi. Only used when stored credentials are detected */
#define ESP_MAXIMUM_RETRY  5

/* Bits used in the WiFi event group */
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief Event handler for smart config using ESPTouch. Parameters are required for smart config to take place.
*/
void sc_event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data);

/**
 * @brief Task that gets created for smart config to be executed
*/
void smartconfig_task(void * parm);

/**
 * @brief Event handler for the WiFi initialisation with only using the stored credentials. Parameters are the required
      event handler parameters.
*/
void wifi_event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data);

```

```
50 /**
51  * @brief Initialise WiFi with the stored credentials. Only uses the credentials stored in the NVS. For the credentials to
        change
52  * the flash must be erased.
53  *
54  * @return ESP_OK on success
55 */
56 esp_err_t wifi_init(void);
57 
58 /**
59  * @brief Initialise WiFi with smart config. It requires the ESPTouch app downloaded to pass the credentials from the
60  * connected device with the app to the ESP32.
61  *
62  * @return ESP_OK on success
63 */
64 esp_err_t wifi_init_sc(void);
65 
66 #ifdef __cplusplus
67 }
68 #endif
```

Listing 6: WUC Code - wifi.c

```
#include "wifi.h"

/* FreeRTOS event group to signal when we are connected & ready to make a request */
static EventGroupHandle_t s_wifi_event_group;


/* Initialisations required for WiFi events */
static const int CONNECTED_BIT = BIT0;
static const int ESPTOUCH_DONE_BIT = BIT1;
extern wuc_config_t wuc_config;

/* Variable to store retries */
static int s_retry_num = 0;

static const char *TAG = "WUC - WiFi";

void smartconfig_task(void * parm)
{
    EventBits_t uxBits;
    ESP_ERROR_CHECK( esp_smartconfig_set_type(SC_TYPE_ESPTOUCH) );
    smartconfig_start_config_t cfg = SMARTCONFIG_START_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_smartconfig_start(&cfg) );

    /* Task loop */
    while (1) {
        uxBits = xEventGroupWaitBits(s_wifi_event_group, CONNECTED_BIT | ESPTOUCH_DONE_BIT, true, false,
       portMAX_DELAY);

        if(uxBits & CONNECTED_BIT) {
            ESP_LOGI(TAG, "WiFi Connected to ap");
        }

        /* If the ESPTOUCH_DONE_BIT is detected, the task has finished */
        if(uxBits & ESPTOUCH_DONE_BIT) {
            ESP_LOGI(TAG, "smartconfig over");
            esp_smartconfig_stop();
            vTaskDelete(NULL);
        }
    }
}

void sc_event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data)
{
    /* Check what events happen and act accordingly */
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        xTaskCreate(smartconfig_task, "smartconfig_task", 4096, NULL, 3, NULL);
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        esp_wifi_connect();
        xEventGroupClearBits(s_wifi_event_group, CONNECTED_BIT);
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
```

```
        xEventGroupSetBits(s_wifi_event_group, CONNECTED_BIT);
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_SCAN_DONE) {
        ESP_LOGI(TAG, "Scan done");
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_FOUND_CHANNEL) {
        ESP_LOGI(TAG, "Found channel");
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_GOT_SSID_PSWD) {
        ESP_LOGI(TAG, "Got SSID and password");

        smartconfig_event_got_ssid_pswd_t *evt = (smartconfig_event_got_ssid_pswd_t *)event_data;
        wifi_config_t wifi_config;
        uint8_t rvd_data[33] = { 0 };

        /* Copy SSID and PSWD to global variables */
        memcpy(wuc_config.ssid, evt->ssid, sizeof(evt->ssid));
        memcpy(wuc_config.pswd, evt->password, sizeof(evt->password));

        /* Initialise the config to all zeros and assign the credentials */
        bzero(&wifi_config, sizeof(wifi_config_t));
        memcpy(wifi_config.sta.ssid, wuc_config.ssid, sizeof(wifi_config.sta.ssid));
        memcpy(wifi_config.sta.password, wuc_config.pswd, sizeof(wifi_config.sta.password));

        wifi_config.sta.bssid_set = evt->bssid_set;
        if (wifi_config.sta.bssid_set == true) {
            memcpy(wifi_config.sta.bssid, evt->bssid, sizeof(wifi_config.sta.bssid));
        }

        ESP_LOGI(TAG, "SSID:%s", wifi_config.sta.ssid);
        ESP_LOGI(TAG, "PASSWORD:%s", wifi_config.sta.password);

        if (evt->type == SC_TYPE_ESPTOUCH_V2) {
            ESP_ERROR_CHECK( esp_smartconfig_get_rvd_data(rvd_data, sizeof(rvd_data)) );
            ESP_LOGI(TAG, "RVD_DATA:");
            for (int i=0; i<33; i++) {
                printf("%02x ", rvd_data[i]);
            }
            printf("\n");
        }

        /* Disconnect, assign config and reconnect */
        ESP_ERROR_CHECK( esp_wifi_disconnect() );
        ESP_ERROR_CHECK( esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
        esp_wifi_connect();
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_SEND_ACK_DONE) {
        xEventGroupSetBits(s_wifi_event_group, ESPTOUCH_DONE_BIT);
    }
}

void wifi_event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
    {
```

```
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
    {
        if (s_retry_num < ESP_MAXIMUM_RETRY)
        {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else
        {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
    {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

esp_err_t wifi_init_sc()
{
    ESP_LOGI(TAG, "Initialising WiFi SC. Open ESPTouch...");

    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_t *sta_netif = esp_netif_create_default_wifi_sta();
    assert(sta_netif);

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_wifi_init(&cfg) );

    ESP_ERROR_CHECK( esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &sc_event_handler, NULL) )
      ;
    ESP_ERROR_CHECK( esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &sc_event_handler, NULL)
      );
    ESP_ERROR_CHECK( esp_event_handler_register(SC_EVENT, ESP_EVENT_ANY_ID, &sc_event_handler, NULL) );
    ESP_ERROR_CHECK( esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK( esp_wifi_start() );

    return ESP_OK;
}

esp_err_t wifi_init()
{
    esp_err_t ret;
    s_wifi_event_group = xEventGroupCreate();
```

```
150
151     ESP_ERROR_CHECK(esp_netif_init());
152     ESP_ERROR_CHECK(esp_event_loop_create_default());
153     esp_netif_t *sta_netif = esp_netif_create_default_wifi_sta();
154     assert(sta_netif);
155
156     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
157     ESP_ERROR_CHECK( esp_wifi_init(&cfg) );
158
159     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
160     ESP_ERROR_CHECK(esp_wifi_start());
161
162     esp_event_handler_instance_t instance_any_id;
163     esp_event_handler_instance_t instance_got_ip;
164     ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler
        , NULL, &instance_any_id));
165     ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler
        , NULL, &instance_got_ip));
166
167     wifi_config_t wifi_config = {
168         .sta = {
169             .ssid       = "",       // initialise the values with empty string due to a bug
170             .password   = "",
171         }
172     };
173
174     /* Copy the strings from the text file to the config */
175     memcpy(wifi_config.sta.ssid, wuc_config.ssid, sizeof(wuc_config.ssid));
176     memcpy(wifi_config.sta.password, wuc_config.pswd, sizeof(wuc_config.pswd));
177
178     ESP_LOGI(TAG, "Detected SSID: |%s|", wifi_config.sta.ssid);
179     ESP_LOGI(TAG, "Detected Password: |%s|", wifi_config.sta.password);
180
181     // set credentials and try to connect
182     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
183     ret = esp_wifi_connect();
184
185     if(ret != ESP_OK)
186     {
187         ESP_LOGE(TAG, "Error with Wi—Fi connection");
188         return ESP_FAIL;
189     }
190
191     /* Waiting until either the connection is established (WIFI_CONNECTED_BIT) or connection failed for the
        maximum
192     * number of re—tries (WIFI_FAIL_BIT). The bits are set by event_handler() (see above) */
193     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group, WIFI_CONNECTED_BIT | WIFI_FAIL_BIT, pdFALSE,
        pdFALSE, portMAX_DELAY);
194
195     /* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event actually happened.
        */
```

```
196     if (bits & WIFI_CONNECTED_BIT) {
197         ESP_LOGI(TAG, "connected to Access Point SSID:%s password:%s", wifi_config.sta.ssid, wifi_config.sta.password);
198         return ESP_OK;
199     } else if (bits & WIFI_FAIL_BIT) {
200         ESP_LOGW(TAG, "Failed to connect to SSID:%s, password:%s", wifi_config.sta.ssid, wifi_config.sta.password);
201         return ESP_FAIL;
202     } else {
203         ESP_LOGE(TAG, "UNEXPECTED EVENT");
204         return ESP_FAIL;
205     }
206 
207     return ESP_OK;
208 }
```

Listing 7: WUC Code - wuc_config.h

```
#pragma once

#include <stdbool.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "soc/soc_caps.h"
#include "esp_sleep.h"
#include "esp_log.h"
#include "driver/rtc_io.h"
#include "esp_timer.h"
#include "wuc_config.h"

#ifdef __cplusplus
extern "C" {
#endif

/* GPIO pin the MCC is connected to */
#define MCC_PIN         GPIO_NUM_2

/* WUC System Configuration structure. */
typedef struct{
    bool exit;                  // to check to exit the main server loop and enter deep sleep
    bool reset;                 // to reset the system to accept new WiFi credentials
    uint32_t sleep_time_sec;    // to store how long the system will sleep
    uint32_t active_time_sec;   // to store the default active time
    char ssid[33];              // to store the WiFi SSID
    char pswd[65];              // to store the WiFi Password
    bool stored_creds;          // to check if credentials are stored in the NVS
}wuc_config_t;

/**
 * @brief Setup the deep sleep configuration based on the sleep_time_sec in the config structure. Happens at every boot.
 *
 * @return ESP_OK on success
*/
esp_err_t deep_sleep_setup();

/**
 * @brief GPIO setup to power on the MCC
 *
 * @return ESP_OK
*/
esp_err_t setup_mcc_power_switch();

/**
 * @brief Call to power up the MCC.
 *
```

```
 * @return ESP_OK on success
*/
esp_err_t mcc_powerup();

/**
 * @brief Call to power down the MCC
 *
 * @return ESP_OK on success
*/
esp_err_t mcc_powerdown();

#ifdef __cplusplus
}
#endif
```

Listing 8: WUC Code - wuc_config.c

```
#include "wuc_config.h"

static const char *TAG = "WUC - System Config";

/* Declaration required to output time slept */
static RTC_DATA_ATTR struct timeval sleep_enter_time;

extern wuc_config_t wuc_config;

esp_err_t setup_mcc_power_switch(){
    return gpio_set_direction(MCC_PIN, GPIO_MODE_OUTPUT);
}

esp_err_t mcc_powerup()
{
    ESP_LOGI(TAG, "Powering up MCC...");
    return gpio_set_level(MCC_PIN, 1);
}

esp_err_t mcc_powerdown()
{
    ESP_LOGI(TAG, "Powering down MCC...");
    return gpio_set_level(MCC_PIN, 0);
}

esp_err_t deep_sleep_setup()
{
    /* Register RTC timer */
    ESP_LOGI(TAG,"Enabling timer wakeup, %lud s", wuc_config.sleep_time_sec);

    /* Enable RTC timer wakeup */
    ESP_ERROR_CHECK(esp_sleep_enable_timer_wakeup(wuc_config.sleep_time_sec * 1000000));
    ESP_LOGI(TAG, "Entering sleep for %ld", wuc_config.sleep_time_sec / 1000000 );

    /* Get sleep time in ms to output to temrinal */
    struct timeval now;
    gettimeofday(&now, NULL);
    int sleep_time_ms = (now.tv_sec - sleep_enter_time.tv_sec) * 1000 + (now.tv_usec - sleep_enter_time.tv_usec) / 1000;

    switch (esp_sleep_get_wakeup_cause()) {
        case ESP_SLEEP_WAKEUP_TIMER: {
            ESP_LOGI(TAG,"Wake up from timer. Time spent in deep sleep: %dms", sleep_time_ms);
            break;
        }
        case ESP_SLEEP_WAKEUP_UNDEFINED:
        default:
            ESP_LOGI(TAG,"Not a deep sleep reset");
    }

    vTaskDelay(10 / portTICK_PERIOD_MS);
```

```
51
52      /*
53          Isolate GPIO12 pin from external circuits. This is needed for modules
54          which have an external pull-up resistor on GPIO12 (such as ESP32-WROVER)
55          to minimize current consumption.
56      */
57      rtc_gpio_isolate(GPIO_NUM_12);
58
59      /* Get deep sleep enter time */
60      gettimeofday(&sleep_enter_time, NULL);
61
62      return ESP_OK;
63  }
```

Listing 9: WUC Code - wuc_nvs.h

```
#pragma once

#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "nvs.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "wuc_config.h"

#define STORAGE_NAMESPACE "storage"

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief Used to check the size (in bytes) allocated for the credentials in the NVS.
 *
 * @return ESP_OK on success
*/
esp_err_t get_nvs_sizes();

/**
 * @brief Check the NVS storage for the  Wifi credentials. If a size of 0 was detected by get_nvs_sizes(), then they are
      assumed to not exist.
 *
 * @return ESO_OK on success
*/
esp_err_t check_nvs();

/**
 * @brief For storing the new credentials to the NVS. Sizes of the credentials is not required. Called after smart config is
      executed
 * to get the credentials.
 *
 * @return ESP_OK on success
*/
esp_err_t store_creds_to_nvs();

#ifdef __cplusplus
}
#endif
```

Listing 10: WUC Code - wuc_nvs.c

```
#include "wuc_nvs.h"

static const char* TAG = "WUC - NVS";
extern wuc_config_t wuc_config;
size_t ssid_required_size = 0;  // value will default to 0, if not set yet in NVS
size_t pswd_required_size = 0;

esp_err_t get_nvs_sizes()
{
    ESP_LOGI(TAG, "Getting the credential sizes from the NVS...");

    nvs_handle_t my_handle;
    esp_err_t err;

    /* Open NVS storage */
    err = nvs_open(STORAGE_NAMESPACE, NVS_READWRITE, &my_handle);
    if (err != ESP_OK) {ESP_LOGE(TAG, "Error opening NVS."); return err; }

    /* Get the SSID variable size */
    err = nvs_get_str(my_handle, "ssid", NULL, &ssid_required_size);
    ESP_LOGI(TAG, "SSID required size: %d", ssid_required_size);

    /**
     * Check what was returned by looking for the SSID. NVS not found error is acceptable because it
     * means it has a size of 0, meaning it has not been stored yet!
     */
    if (err != ESP_OK && err != ESP_ERR_NVS_NOT_FOUND) {
        ESP_LOGE(TAG, "Error getting SSID size.");
        return err;
    }

    /* Get Password variable size */
    err = nvs_get_str(my_handle, "pswd", NULL, &pswd_required_size);
    ESP_LOGI(TAG, "Password required size: %d", pswd_required_size);

    /**
     * Check what was returned by looking for the Password. NVS not found error is acceptable because it
     * means it has a size of 0, meaning it has not been stored yet!
     */
    if (err != ESP_OK && err != ESP_ERR_NVS_NOT_FOUND) {
        ESP_LOGE(TAG, "Error getting Password size.");
        return err;
    }

    /* Close the NVS storage */
    nvs_close(my_handle);
    return ESP_OK;
}

esp_err_t check_nvs()
```

```
{

    ESP_LOGI(TAG, "Checking NVS for the credentials...");

    nvs_handle_t my_handle;
    esp_err_t err;

    /* Call the function to get the SSID and Password sizes*/
    get_nvs_sizes();

    /* Open NVS storage */
    err = nvs_open(STORAGE_NAMESPACE, NVS_READWRITE, &my_handle);
    if (err != ESP_OK) {ESP_LOGE(TAG, "Error opening NVS."); return err; }

    /* Check if either of them is 0 */
    if(ssid_required_size == 0 || pswd_required_size == 0){
        /* Do smart config to get credentials */
        wuc_config.stored_creds = false;
    } else {
        /* Read stored SSID */
        err = nvs_get_str(my_handle, "ssid", wuc_config.ssid, &ssid_required_size);
        ESP_LOGI(TAG, "Read SSID from NVS: %s", wuc_config.ssid);

        /* Error checking */
        if (err != ESP_OK) {
            ESP_LOGE(TAG, "Error getting string. Error: %s", esp_err_to_name(err));
            return err;
        }

        /* Read stored password */
        err = nvs_get_str(my_handle, "pswd", wuc_config.pswd, &pswd_required_size);
        ESP_LOGI(TAG, "Read Password from NVS: %s", wuc_config.pswd);

        /* Error checking */
        if (err != ESP_OK) {
            ESP_LOGE(TAG, "Error getting string.");
            return err;
        }

        /* Set to true as to not do the smart config process*/
        wuc_config.stored_creds = true;
    }

    /* Close the NVS storage */
    nvs_close(my_handle);
    return ESP_OK;
}

esp_err_t store_creds_to_nvs()
{
    ESP_LOGI(TAG, "Saving credentials to NVS...");
```

```
    nvs_handle_t my_handle;
    esp_err_t err;

    /* Open NVS storage */
    err = nvs_open(STORAGE_NAMESPACE, NVS_READWRITE, &my_handle);
    if (err != ESP_OK) return err;

    /* Save SSID to NVS storage and check for errors */
    err = nvs_set_str(my_handle, "ssid", wuc_config.ssid);
    if (err != ESP_OK) return err;

    /* Save Password to NVS storage and check for errors */
    err = nvs_set_str(my_handle, "pswd", wuc_config.pswd);
    if (err != ESP_OK) return err;

    /* Commit changes to NVS */
    err = nvs_commit(my_handle);
    if (err != ESP_OK) return err;

    /* Close NVS storage */
    nvs_close(my_handle);
    return ESP_OK;
}
```

Listing 11: MCC Code - main.c

```
#include "camera.h"
#include "sd.h"
#include "wifi.h"
#include "http.h"
#include "mcc_config.h"

/* TAG to be used when outputting logs to terminal */
static const char *TAG = "WULPSC";

/* Empty camera buffer to be used throughout the project */
camera_fb_t* fb = NULL;

/* System configuration default settings. The values set here are for normal room light level. */
mcc_config_t mcc_config = {
    .exit           =  false,
    .flash          =  false,
    .sd_save        =  NO_SD_SAVE,
    .pic_done       =  false,
    .sel_status     =  0,
    .camera = {
        .brightness     = 0,    // from -2 to 2
        .contrast       = 0,    // from -2 to 2
        .saturation     = 0,    // from -2 to 2
        .sharpness      = 0,    // from -2 to 2, NOT SUPPORTED
        .denoise        = 0,    // from 0 to 255, NOT SUPPORTED
        .special_effect = 0,    // from 0 to 6
        .wb_mode        = 3,    // from 0 to 4
        .ae_level       = 0,    // from -2 to 2
        .aec_value      = 20,   // from 0 to 1200
        .agc_gain       = 2,    // from 0 to 30
        .gainceiling    = 1,    // from 0 to 6
        .lenc           = true,
        .agc            = false,
        .aec            = false,
        .hmirror        = false,
        .vflip          = false,
        .aec2           = true,
        .bpc            = true,
        .wpc            = true
    }
};

void app_main(void)
{
    esp_err_t ret; // variable for function returns
    esp_err_t wifi_ret;
    httpd_handle_t server;

    /* Initialise the camera */
    ret = init_camera();
```

```
    if(ret != ESP_OK) return;

    /* Initialise NVS for WiFi*/
    ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ESP_ERROR_CHECK(nvs_flash_init());
    }

    /**
     * Setup the GPIO pin to change the value of the SEL pin of the MUXs, initialised to 0
     * */
    gpio_set_direction(SEL_PIN, GPIO_MODE_OUTPUT);
    gpio_set_level(SEL_PIN, 0);
    vTaskDelay(10/portTICK_PERIOD_MS);
    ESP_LOGI(TAG, "SEL pin set to %d", SEL_PIN);

/* Set SD_CRED definition to 0 to use the hardcoded credentials in wifi.h */
#if SD_CRED
    /* Initialise SD card */
    sd_init();

    /* Read the WiFi credentials from the SD card */
    read_wifi_credentials();

    if (mcc_config.sd_save == NO_SD_SAVE) {
        sd_deinit();
    }
#endif

    /* Initialise WiFi */
    wifi_init_general();
    wifi_ret = wifi_init_sta();

    /* Only create a HTTP server if a WiFi connection was established */
    if (wifi_ret == ESP_OK) {
        server = start_webserver();
        if (server == NULL) {
            ESP_LOGE(TAG,"Error Server is NULL");
            return;
        } else {
            ESP_LOGI(TAG, "Started server...");
        }

        while(server) {
            vTaskDelay(10/portTICK_PERIOD_MS);
            if (mcc_config.exit) {
                ESP_LOGI(TAG, "Server Stopping");
                stop_webserver(server);
                server = NULL;
            }
```

```
            if (mcc_config.sd_save == SD_SETUP) {
                sd_init();
                sys_sd_var_setup();
                mcc_config.sd_save = SD_SAVE;
            }
            if (mcc_config.pic_done) {
                if (mcc_config.sd_save == SD_SAVE) {
                    sys_sd_save(fb);
                }

                /* Return frame buffer and make sure it is empty */
                esp_camera_fb_return(fb);
                fb = NULL;
                sys_camera_switch();

                /* Reset for next picture */
                mcc_config.pic_done = false;
                ESP_LOGI(TAG, "SWITCHED");
            }
        }
    } else {
        /**
         * Save only one image if no WiFi so that some monitoring can still occur,
         * SD card will always be required due to the WiFi credentials
         * If SD saving is not enabled, enable it in order to allow for partial monitoring
         * */

        fb = fb_refresh(fb);
        fb = sys_take_picture();

        if(!mcc_config.sd_save){
            sys_sd_var_setup();
        }
        sys_sd_save(fb);
    }

    /* Return frame buffer */
    esp_camera_fb_return(fb);
    esp_camera_deinit();
    ESP_LOGI(TAG, "Returned frame buffer and de-initialised camera before exit.");

    /* Check if SD is initialised to de-initialise it*/
    if(mcc_config.sd_save == SD_SAVE){
        sd_deinit();
    }

    ESP_LOGI(TAG,"DONE!!!!!!!!!!");
    return;
}
```

Listing 12: MCC Code - http.h

```
#pragma once

#include "esp_http_server.h"
#include "esp_err.h"
#include "mcc_config.h"
#include "esp_log.h"
#include "camera.h"
#include "sd.h"

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief Used as a callback to encode the frame buffer to JPEG when the pixel format is not
 * JPEG and must be converted for efficient data transmission. Function was provided by Esspressif.
 *
 * @param arg Used to set the JPEG chunking variable
 * @param index To check length
 * @param data Image data
 * @param len Size of the data
 *
 * @return len or the size of the data
*/
static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len);

/**
 * @brief Handler to take a picture and send it via HTTP.
 *
 * @param req pointer to HTTP request
 *
 * @return ESP_OK on success
*/
esp_err_t picture_handler(httpd_req_t *req);

/**
 * @brief A simple HTTP handler. Used to test server connection.
 *
 * @param req pointer to the HTTP request
 *
 * @return ESP_OK on success
*/
esp_err_t get_handler(httpd_req_t *req);

/**
 * @brief Start the HTTP server
 *
 * @param void
 *
 * @return handle to the HTTP server
```

```
*/
httpd_handle_t start_webserver();

/**
 * @brief Stop the HTTP server
 *
 * @param server handle to the HTTP server
*/
void stop_webserver(httpd_handle_t server);

/**
 * @brief Handler to change the camera configuration and system settings
 *
 * @param req pointer to the HTTP request
 *
 * @return ESP_OK on success
*/
esp_err_t config_settings_post_handler(httpd_req_t *req);


/* Sample cURL command to use with the terminal to set the system settings,

curl -ContentType 'application/json' -Body ' {"brightness":0, "contrast":0, "saturation":0, "sharpness":0, "denoise":0,
      "special_effect":0,"wb_mode":3,"ae_level":0,"aec_value":200,"agc_gain":2,"gainceiling":6,"lenc":1,"agc":0,"aec":0, "
      hmirror":0,"vflip":0,"aec2":0, "bpc":1, "wpc":1, "sd_save":0}' -Method Post http://IP_ADDRESS_HERE:19520/
      config

*/

#ifdef __cplusplus
}
#endif
```

Listing 13: MCC Code - http.c

```
#include "http.h"

static const char *TAG = "WULPSC – http protocol";
extern camera_fb_t * fb;
extern mcc_config_t mcc_config;

typedef struct {
        httpd_req_t *req;
        size_t len;
} jpg_chunking_t;


esp_err_t get_handler(httpd_req_t *req)
{
    ESP_LOGI(TAG, "Entered get handler");

    /* Send a simple response. Mainly used for testing */
    const char resp[] = "URI GET Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len)
{
    jpg_chunking_t *j = (jpg_chunking_t *)arg;
    if (!index) {
        j->len = 0;
    }

    if (httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK) {
        return 0;
    }

    j->len += len;
    return len;
}

esp_err_t picture_handler(httpd_req_t *req){
    esp_err_t res = ESP_OK;

    /* Refresh the image and take a new picture */
    fb = fb_refresh(fb);
    fb = sys_take_picture();

    /* Check if the frame buffer is ok to send */
    if (!fb) {
        ESP_LOGE(TAG, "Camera capture failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }
```

```
51      ESP_LOGI(TAG,"Took new picture.");
52
53      /* Output image information to the terminal */
54      pic_data_output(fb);
55
56      ESP_LOGI(TAG,"Picture received. Trying to send now.");
57
58      /* Set the content type header to a JPEG image */
59      res = httpd_resp_set_type(req, "image/jpeg");
60      if (res == ESP_OK) {
61          res = httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.jpg");
62      }
63
64      /* Check if the response is as expected and if yes then try to send the image */
65      if (res == ESP_OK) {
66          /* If the image is JPEG it gets sent as one batch of data */
67          if (fb->format == PIXFORMAT_JPEG) {
68              res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
69              ESP_LOGI(TAG, "Response Sent");
70          } else {
71              /* If the image is anything but JPEG, it gets converted to JPEG and sent in chunks */
72              jpg_chunking_t jchunk = {req, 0};
73              res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk)?ESP_OK:ESP_FAIL;
74              httpd_resp_send_chunk(req, NULL, 0);
75              ESP_LOGI(TAG, "Response Sent in chunks");
76          }
77      }
78      /* To switch cameras */
79      mcc_config.pic_done = true;
80      return res;
81  }
82
83  esp_err_t config_settings_post_handler(httpd_req_t *req)
84  {
85      /**
86      * Buffer for recieving content/params, content is expected to be a JSON string
87      * such as {saturation: 2, contrast: 0, brightness: 1, ...}
88      */
89      int MAX = 511;
90      char *content;
91
92      /* Make sure bytes dont go over max length*/
93      size_t recv_size = req->content_len;
94      if (recv_size >= MAX) {
95          ESP_LOGE(TAG, "Over %d bytes sent", MAX);
96          httpd_resp_send_500(req);
97          return ESP_FAIL;
98      }
99      content = malloc(sizeof(char) * recv_size + 1);
100     ESP_LOGI(TAG, "Size %d bytes", recv_size);
101
```

```
    int ret = httpd_req_recv(req, content, recv_size);
    ESP_LOGI(TAG, "Data %s", content);
    if (ret <= 0) {  /* 0 return value indicates connection closed */
        /* Check if timeout occurred */
        if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
            /* In case of timeout one can choose to retry calling
             * httpd_req_recv(), but to keep it simple, here we
             * respond with an HTTP 408 (Request Timeout) error */
            httpd_resp_send_408(req);
        }
        /* In case of error, returning ESP_FAIL will
         * ensure that the underlying socket is closed */
        return ESP_FAIL;
    }
    /* Add null terminating byte to convert content buffer into string */
    content[recv_size] = '\0';


    JSON_config_set(content);
    mcc_config.sensor = esp_camera_sensor_get();
    camera_set_settings(mcc_config);

    /* Send a response to */
    const char resp[] = "Paramaters set";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;

    /* Sample cURL command is in http.h */
}

esp_err_t exit_handler(httpd_req_t *req){
    mcc_config.exit = 1;
    /* Send a response to */
    const char resp[] = "Exited safely";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

/* GET Handlers */
httpd_uri_t uri_get = {
    .uri      = "/uri",
    .method   = HTTP_GET,
    .handler  = get_handler,
    .user_ctx = NULL
};

httpd_uri_t pic_get = {
    .uri      = "/pic",
    .method   = HTTP_GET,
    .handler  = picture_handler,
    .user_ctx = NULL
```

```
};

httpd_uri_t exit_get = {
    .uri      = "/exit",
    .method   = HTTP_GET,
    .handler  = exit_handler,
    .user_ctx = NULL
};

/* POST Handlers */
httpd_uri_t config_settings_post = {
    .uri      = "/config",
    .method   = HTTP_POST,
    .handler  = config_settings_post_handler,
    .user_ctx = NULL
};

httpd_handle_t start_webserver(void)
{
    /* Generate default configuration */
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 19520;

    /* Empty handle to esp_http_server */
    httpd_handle_t server = NULL;

    /* Start the httpd server */
    if (httpd_start(&server, &config) == ESP_OK) {
        /* Register URI handlers */

        /* GET */
        httpd_register_uri_handler(server, &uri_get);   // simple handler for testing
        httpd_register_uri_handler(server, &pic_get);   // to get the image
        httpd_register_uri_handler(server, &exit_get);   // to get the image

        /* POST */
        httpd_register_uri_handler(server, &config_settings_post); // change camera settings
    }
    /* If server failed to start, handle will be NULL */
    return server;
}

void stop_webserver(httpd_handle_t server)
{
    if (server) {
        /* Stop the httpd server */
        httpd_stop(server);
    }
}
```

Listing 14: MCC Code - camera.h

```
#pragma once

#include "esp_camera.h"
#include "esp_log.h"
#include "mcc_config.h"

/* Camera pin definitions for ESP32-CAM */
#define CAM_PIN_PWDN 32
#define CAM_PIN_RESET -1 // Reset is hardwired to HIGH, based on ESP32-CAM schematic
#define CAM_PIN_XCLK 0
#define CAM_PIN_SIOD 26
#define CAM_PIN_SIOC 27
#define CAM_PIN_D7 35
#define CAM_PIN_D6 34
#define CAM_PIN_D5 39
#define CAM_PIN_D4 36
#define CAM_PIN_D3 21
#define CAM_PIN_D2 19
#define CAM_PIN_D1 18
#define CAM_PIN_D0 5
#define CAM_PIN_VSYNC 25
#define CAM_PIN_HREF 23
#define CAM_PIN_PCLK 22
#define CAM_PIN_FLASH 4

/**
 * Amount of times to refresh the frame buffer before taking a picture.
 * It was found that a green tint was in the images when only one was taken, this is to remedie that.
*/
#define REFRESH_NUM 2

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief Used to set up the flash LED on the ESP
 *
 * @return ESP_OK on success
*/
esp_err_t setup_flash();

/**
 * @brief Turn on the flash
 *
 * @return ESP_OK on success
*/
esp_err_t turn_on_flash();

/**
```

```
51  * @brief Turn off the flash
52  *
53  * @return ESP_OK on success
54 */
55 esp_err_t turn_off_flash();
56
57 /**
58  * @brief Output data from the picture taken
59  *
60  * @param fb pointer to the camera frame buffer
61 */
62 void pic_data_output(camera_fb_t *fb);
63
64 /**
65  * @brief Initialise the camera
66  *
67  * @return ESP_OK on success
68 */
69 esp_err_t init_camera();
70
71 /**
72  * @brief Used to change the pixel format outside the initialisation. Only gets executed when the user selects
73  * saving to the SD card
74  *
75  * @return ESP_OK on succcess
76 */
77 esp_err_t change_pixformat_to_jpeg();
78
79 /**
80  * @brief Refresh the frame buffer by taking a pic a discarding it
81  *
82  * @param fb - frame buffer
83  *
84  * @note It was observed that for an updated picture with the correct settings, the picture had to be taken twice, this
        function simplifies that process.
85 */
86 camera_fb_t* fb_refresh(camera_fb_t * fb);
87
88 #ifdef __cplusplus
89 }
90 #endif
```

Listing 15: MCC Code - camera.c

```
#include "camera.h"

/* Log tag for camera source code */
static const char *TAG = "WULPSC - Camera";

/* System config defined in main*/
extern mcc_config_t mcc_config;

/* Declare the camera configuration */
camera_config_t camera_config = {
    .pin_pwdn       = CAM_PIN_PWDN,
    .pin_reset      = CAM_PIN_RESET,
    .pin_xclk       = CAM_PIN_XCLK,
    .pin_sccb_sda   = CAM_PIN_SIOD,
    .pin_sccb_scl   = CAM_PIN_SIOC,

    .pin_d7 = CAM_PIN_D7,
    .pin_d6 = CAM_PIN_D6,
    .pin_d5 = CAM_PIN_D5,
    .pin_d4 = CAM_PIN_D4,
    .pin_d3 = CAM_PIN_D3,
    .pin_d2 = CAM_PIN_D2,
    .pin_d1 = CAM_PIN_D1,
    .pin_d0 = CAM_PIN_D0,

    .pin_vsync  = CAM_PIN_VSYNC,
    .pin_href   = CAM_PIN_HREF,
    .pin_pclk   = CAM_PIN_PCLK,

    /* Typically used 20 MHz, reduced to 10 MHz to improve synchonisation*/
    .xclk_freq_hz   = 10000000,
    .ledc_timer     = LEDC_TIMER_0,
    .ledc_channel   = LEDC_CHANNEL_0,

    .pixel_format   = PIXFORMAT_YUV422, /* YUV422,GRAYSCALE,RGB565,JPEG */
    .frame_size     = FRAMESIZE_VGA,    /* QQVGA-UXGA, For ESP32, do not use sizes above QVGA when not
      JPEG. */

    .jpeg_quality   = 4,                /* 0-63, for OV series camera sensors, lower number means higher quality */
    .fb_count       = 1,                /* When jpeg mode is used, if fb_count more than one, the driver will work in
      continuous mode. */
    .grab_mode      = CAMERA_GRAB_WHEN_EMPTY,
    .fb_location    = CAMERA_FB_IN_PSRAM,
};

esp_err_t init_camera()
{
    /* Initialise camera */
    esp_err_t err = esp_camera_init(&camera_config);
    if (err != ESP_OK)
```

```
    {
        ESP_LOGE(TAG, "Camera Init Failed");
        return err;
    }
    return ESP_OK;
}

esp_err_t change_pixformat_to_jpeg()
{
    camera_config.pixel_format = PIXFORMAT_JPEG;
    return ESP_OK;
}

esp_err_t setup_flash()
{
    ESP_ERROR_CHECK(gpio_set_level(CAM_PIN_FLASH,0)); //initialise to 0 so to not blink twice
    return gpio_set_direction(CAM_PIN_FLASH,GPIO_MODE_OUTPUT);
}

esp_err_t turn_on_flash()
{
    return gpio_set_level(CAM_PIN_FLASH,1);
}

esp_err_t turn_off_flash()
{
    return gpio_set_level(CAM_PIN_FLASH,0);
}

void pic_data_output(camera_fb_t *fb)
{
    ESP_LOGI(TAG, "-------------------------");
    ESP_LOGI(TAG, "Size: %zu bytes", fb->len);
    ESP_LOGI(TAG, "Height: %zu", fb->height);
    ESP_LOGI(TAG, "Width: %zu", fb->width);
    ESP_LOGI(TAG, "Using JPEG Quality: %d", camera_config.jpeg_quality);
    ESP_LOGI(TAG, "Format: %d", fb->format); // see sensor.h for format
    ESP_LOGI(TAG, "-------------------------");
}

camera_fb_t* fb_refresh(camera_fb_t * fb)
{
    for(int i = 0; i < REFRESH_NUM; i++){
        fb = esp_camera_fb_get();
        esp_camera_fb_return(fb);
        fb = NULL;
        ESP_LOGI(TAG, "Refreshed camera...");
    }

    return fb;
}
```

Listing 16: MCC Code - mcc_config.h

```
#pragma once

#include "esp_camera.h"
#include "esp_err.h"
#include "cJSON.h"
#include "esp_random.h"
#include "sd.h"
#include "camera.h"

/* Pin to change MUX */
#define SEL_PIN  4

#ifdef __cplusplus
extern "C" {
#endif

/* States used for checking the SD configuration */
typedef enum{
    NO_SD_SAVE,
    SD_SETUP,
    SD_SAVE
}sd_state_t;

/* System Configuration structure. Stores all system settings */
typedef struct{
    sd_state_t sd_save;
    bool flash;
    bool exit;
    bool pic_done;
    bool sel_status;
    sensor_t* sensor;
    camera_status_t camera;
}mcc_config_t;

/**
 * @brief Takes all camera variables in the system config and sets them
 *
 * @return ESP_OK on success
*/
esp_err_t camera_set_settings();

/**
 * @brief Used for debugging the camera sensor settings. Prints the logs to the terminal
 *
*/
void camera_get_settings();

/**
 * @brief Uses the content received and parses it to the corresponding system config variables
 *
```

```
51  * @param content pointer to the JSON string
52  *
53  * @return variable of type mcc_config, i.e. the system config variable
54 */
55 esp_err_t JSON_config_set(char* content);
56 
57 /**
58  * @brief Take a picture. Used to check if picture should be taken with flash or not
59  *
60  * @return camera frame buffer
61 */
62 camera_fb_t* sys_take_picture();
63 
64 /**
65  * @brief Setup the system to allow efficient saving time of the frame buffers.
66  * Creates the variables for the file names by creating random string of six caracters, one being the null terminator.
67  *
68  * @return ESP_OK on success
69 */
70 esp_err_t sys_sd_var_setup();
71 
72 /**
73  * @brief Save the frame buffers to the SD card based on the file names defined by the sys_sd_var_setup() function.
74  *
75 */
76 esp_err_t sys_sd_save(camera_fb_t* fb);
77 
78 /**
79  * @brief Switch the cameras. Used in the HTTP GET handlers
80  *
81  * @return ESP_OK on success
82 */
83 esp_err_t sys_camera_switch();
84 
85 #ifdef __cplusplus
86 }
87 #endif
```

Listing 17: MCC Code - mcc_config.c

```
#include "mcc_config.h"

const char * TAG = "WULPSC - Config";

extern mcc_config_t mcc_config;

/* 16-bit random number */
uint16_t rnd;

/* String var to store 16-bit random number (up to 65536,  i.e. 5 character bytes + 1 for null terminator)  */
char rnd_str[6] = "\0";

/* 19+1 bytes max length file path */
char path[20] = "\0";

esp_err_t camera_set_settings()
{
    /* Set the values to the system config */
    mcc_config.sensor->set_brightness(mcc_config.sensor, mcc_config.camera.brightness);             // from -2 to 2
    mcc_config.sensor->set_contrast(mcc_config.sensor, mcc_config.camera.contrast);                 // from -2 to 2
    mcc_config.sensor->set_saturation(mcc_config.sensor, mcc_config.camera.saturation);             // from -2 to 2
    // Not supported by OV2640
    // mcc_config.sensor->set_sharpness(mcc_config.sensor, mcc_config.camera.sharpness);               // from -2 to 2 NOT SUPPORTED
    // mcc_config.sensor->set_denoise(mcc_config.sensor, mcc_config.camera.denoise);                   // from 0 to 255 NOT SUPPORTED
    mcc_config.sensor->set_special_effect(mcc_config.sensor, mcc_config.camera.special_effect);     // from 0 to 6
    mcc_config.sensor->set_wb_mode(mcc_config.sensor, mcc_config.camera.wb_mode);                   // from 0 to 4, white balance mode, 0 - sunny, 1 - cloudy, 2 - office, 3 - home

    mcc_config.sensor->set_ae_level(mcc_config.sensor, mcc_config.camera.ae_level);                 // from -2 to 2, valid when set_exposure_control is ON
    mcc_config.sensor->set_aec_value(mcc_config.sensor, mcc_config.camera.aec_value);               // from 0 to 1200, value for automatic exposure control
    mcc_config.sensor->set_agc_gain(mcc_config.sensor, mcc_config.camera.agc_gain);                 // from 0 to 30, automatic gain control
    mcc_config.sensor->set_gainceiling(mcc_config.sensor, mcc_config.camera.gainceiling);           // from 0 to 6, upper limit of gain

    /* Settings with 1/0 enable/disable */
    mcc_config.sensor->set_lenc(mcc_config.sensor, mcc_config.camera.lenc);                         // lens correction
    mcc_config.sensor->set_gain_ctrl(mcc_config.sensor, mcc_config.camera.agc);                     // auto gain control on/off
    mcc_config.sensor->set_exposure_ctrl(mcc_config.sensor, mcc_config.camera.aec);                 /* auto exposure control on/off,
                                                                                                       it is ON by default. If this ON state alone is insufficient for the exposure value, it is a good idea to also turn set_aec2 ON.     */
    mcc_config.sensor->set_hmirror(mcc_config.sensor, mcc_config.camera.hmirror);                   // mirror horizontally
    mcc_config.sensor->set_vflip(mcc_config.sensor, mcc_config.camera.vflip);                       // flip vertically
    mcc_config.sensor->set_aec2(mcc_config.sensor, mcc_config.camera.aec2);                         /* auto exposure control 2, auto exposure control with DSP of camera
```

```
41                                                                  It seems that it will not take effect unless
      set_exposure_ctrl is turned ON. I think if you turn on both set_exposure_ctrl and set_aec2,
42                                                                  you can get an acceptable exposure for
      automatic exposure control.    */
43    mcc_config.sensor->set_bpc(mcc_config.sensor, mcc_config.camera.bpc);                    // black pixel correction
44    mcc_config.sensor->set_wpc(mcc_config.sensor, mcc_config.camera.wpc);                     // white pixel correction
45
46    ESP_LOGI(TAG, "Set camera settings...");
47    return ESP_OK;
48 }
49
50 void camera_get_settings()
51 {
52    ESP_LOGI(TAG, "---SENSOR--DATA---");
53    ESP_LOGI(TAG, "Brightness: %d", mcc_config.sensor->status.brightness);
54    ESP_LOGI(TAG, "Contrast: %d", mcc_config.sensor->status.contrast);
55    ESP_LOGI(TAG, "Saturation: %d", mcc_config.sensor->status.saturation);
56    ESP_LOGI(TAG, "Special Effect: %d", mcc_config.sensor->status.special_effect);
57    ESP_LOGI(TAG, "WB Mode: %d", mcc_config.sensor->status.wb_mode);
58    ESP_LOGI(TAG, "AE Level: %d", mcc_config.sensor->status.ae_level);
59    ESP_LOGI(TAG, "AEC Value: %d", mcc_config.sensor->status.aec_value);
60    ESP_LOGI(TAG, "AGC Gain: %d", mcc_config.sensor->status.agc_gain);
61    ESP_LOGI(TAG, "Gainceiling: %d", mcc_config.sensor->status.gainceiling);
62    ESP_LOGI(TAG, "Lenc: %d", mcc_config.sensor->status.lenc);
63    ESP_LOGI(TAG, "AGC: %d", mcc_config.sensor->status.agc);
64    ESP_LOGI(TAG, "AEC: %d", mcc_config.sensor->status.aec);
65    ESP_LOGI(TAG, "HMirror: %d", mcc_config.sensor->status.hmirror);
66    ESP_LOGI(TAG, "VFlip: %d", mcc_config.sensor->status.vflip);
67    ESP_LOGI(TAG, "AEC2: %d", mcc_config.sensor->status.aec2);
68    ESP_LOGI(TAG, "BPC: %d", mcc_config.sensor->status.bpc);
69    ESP_LOGI(TAG, "WPC: %d", mcc_config.sensor->status.wpc);
70    return;
71 }
72
73 esp_err_t JSON_config_set(char* content)
74 {
75    /* Use the cJSON API to parse the content to a JSON object*/
76    cJSON *root = cJSON_Parse(content);
77    ESP_LOGI(TAG, "Changing settings...");
78
79    /**
80     * Look for the string corresponding to the correct value in the system configuration.
81     * Happens for every variable the user can change through HTTP.
82    */
83    if (cJSON_GetObjectItem(root, "brightness")) {
84    int8_t _brightness = cJSON_GetObjectItem(root,"brightness")->valueint;
85        mcc_config.camera.brightness =  _brightness;
86    ESP_LOGI(TAG, "Brightness=%d",_brightness);
87  }
88    if (cJSON_GetObjectItem(root, "contrast")) {
89    int8_t _contrast = cJSON_GetObjectItem(root,"contrast")->valueint;
```

```
90          mcc_config.camera.contrast = _contrast;
91        ESP_LOGI(TAG, "Contrast=%d",_contrast);
92      }
93        if (cJSON_GetObjectItem(root, "saturation")) {
94        int8_t _saturation = cJSON_GetObjectItem(root,"saturation")->valueint;
95          mcc_config.camera.saturation = _saturation;
96        ESP_LOGI(TAG, "Saturation=%d",_saturation);
97      }
98        if (cJSON_GetObjectItem(root, "sharpness")) {
99        int8_t _sharpness = cJSON_GetObjectItem(root,"sharpness")->valueint;
100         mcc_config.camera.sharpness = _sharpness;
101       ESP_LOGI(TAG, "Sharpness=%d",_sharpness);
102     }
103       if (cJSON_GetObjectItem(root, "denoise")) {
104       uint8_t _denoise = cJSON_GetObjectItem(root,"denoise")->valueint;
105         mcc_config.camera.denoise = _denoise;
106       ESP_LOGI(TAG, "Denoise=%d",_denoise);
107     }
108       if (cJSON_GetObjectItem(root, "special_effect")) {
109       uint8_t _special_effect = cJSON_GetObjectItem(root,"special_effect")->valueint;
110         mcc_config.camera.special_effect = _special_effect;
111       ESP_LOGI(TAG, "Special Effect=%d",_special_effect);
112     }
113       if (cJSON_GetObjectItem(root, "wb_mode")) {
114       uint8_t _wb_mode = cJSON_GetObjectItem(root,"wb_mode")->valueint;
115         mcc_config.camera.wb_mode = _wb_mode;
116       ESP_LOGI(TAG, "WB Mode=%d",_wb_mode);
117     }
118       if (cJSON_GetObjectItem(root, "ae_level")) {
119       int8_t _ae_level = cJSON_GetObjectItem(root,"ae_level")->valueint;
120         mcc_config.camera.ae_level = _ae_level;
121       ESP_LOGI(TAG, "AE Level=%d",_ae_level);
122     }
123       if (cJSON_GetObjectItem(root, "aec_value")) {
124       int16_t _aec_value = cJSON_GetObjectItem(root,"aec_value")->valueint;
125         mcc_config.camera.aec_value = _aec_value;
126       ESP_LOGI(TAG, "AEC Value=%d",_aec_value);
127     }
128       if (cJSON_GetObjectItem(root, "agc_gain")) {
129       uint8_t _agc_gain = cJSON_GetObjectItem(root,"agc_gain")->valueint;
130         mcc_config.camera.agc_gain = _agc_gain;
131       ESP_LOGI(TAG, "AGC Gain=%d",_agc_gain);
132     }
133       if (cJSON_GetObjectItem(root, "gainceiling")) {
134       uint8_t _gainceiling = cJSON_GetObjectItem(root,"gainceiling")->valueint;
135         mcc_config.camera.gainceiling = _gainceiling;
136       ESP_LOGI(TAG, "Gainceiling=%d",_gainceiling);
137     }
138       if (cJSON_GetObjectItem(root, "lenc")) {
139       bool _lenc = cJSON_GetObjectItem(root,"lenc")->valueint;
140         mcc_config.camera.lenc = _lenc;
```

```
    ESP_LOGI(TAG, "LenC=%d",_lenc);
  }
    if (cJSON_GetObjectItem(root, "agc")) {
    bool _agc = cJSON_GetObjectItem(root,"agc")->valueint;
        mcc_config.camera.agc = _agc;
    ESP_LOGI(TAG, "AGC=%d",_agc);
  }
    if (cJSON_GetObjectItem(root, "aec")) {
    bool _aec = cJSON_GetObjectItem(root,"aec")->valueint;
        mcc_config.camera.aec = _aec;
    ESP_LOGI(TAG, "AEC=%d",_aec);
  }
    if (cJSON_GetObjectItem(root, "hmirror")) {
    bool _hmirror = cJSON_GetObjectItem(root,"hmirror")->valueint;
        mcc_config.camera.hmirror = _hmirror;
    ESP_LOGI(TAG, "HMirror=%d",_hmirror);
  }
    if (cJSON_GetObjectItem(root, "vflip")) {
    bool _vflip = cJSON_GetObjectItem(root,"vflip")->valueint;
        mcc_config.camera.vflip = _vflip;
    ESP_LOGI(TAG, "VFlip=%d",_vflip);
  }
    if (cJSON_GetObjectItem(root, "aec2")) {
    bool _aec2 = cJSON_GetObjectItem(root,"aec2")->valueint;
        mcc_config.camera.aec2 = _aec2;
    ESP_LOGI(TAG, "AEC2=%d",_aec2);
  }
    if (cJSON_GetObjectItem(root, "bpc")) {
    bool _bpc = cJSON_GetObjectItem(root,"bpc")->valueint;
        mcc_config.camera.bpc = _bpc;
    ESP_LOGI(TAG, "BPC=%d",_bpc);
  }
    if (cJSON_GetObjectItem(root, "wpc")) {
    bool _wpc = cJSON_GetObjectItem(root,"wpc")->valueint;
        mcc_config.camera.wpc = _wpc;
    ESP_LOGI(TAG, "WPC=%d",_wpc);
  }
    if (cJSON_GetObjectItem(root, "sd_save")) {
        bool _sd_save = cJSON_GetObjectItem(root, "sd_save")->valueint;
        mcc_config.sd_save = _sd_save;
        ESP_LOGI(TAG, "SD_SAVE=%d", _sd_save);
    }
  cJSON_Delete(root);
    return ESP_OK;
}


camera_fb_t* sys_take_picture()
{
    camera_fb_t* fb;

```

```
192     /* Apply settings at each picture. Required due to initialisation and de-initialisation */
193     mcc_config.sensor = esp_camera_sensor_get();
194     camera_set_settings(mcc_config);
195     vTaskDelay(10/portTICK_PERIOD_MS);
196 
197     /* Not implemented, but if a flash LED was used, it would get turned on based on the .flash member state */
198     switch (mcc_config.flash){
199     case false:
200         fb = esp_camera_fb_get();
201         return fb;
202         break;
203     case true:
204         turn_on_flash();
205         fb = esp_camera_fb_get();
206         turn_off_flash();
207         return fb;
208         break;
209     default:
210         fb = NULL;
211         ESP_LOGW(TAG, "Problem with flash value, no image taken...");
212         return fb;
213         break;
214     }
215 }
216 
217 esp_err_t sys_camera_switch()
218 {
219     esp_err_t ret;
220 
221     /* De-initialise */
222     ret = esp_camera_deinit();
223     if (ret != ESP_OK) {
224         ESP_LOGW(TAG, "De-init returned badly");
225     } else {
226         ESP_LOGW(TAG, "De-init OK");
227     }
228     vTaskDelay(100/portTICK_PERIOD_MS);
229 
230     /* Power down for safe de-initialisation */
231     gpio_set_level(CAM_PIN_PWDN, 0);
232     if (ret != ESP_OK) {
233         ESP_LOGW(TAG, "GPIO set level returned badly");
234     } else {
235         ESP_LOGW(TAG, "GPIO set level OK. PWDN is 0");
236     }
237     vTaskDelay(100/portTICK_PERIOD_MS);
238 
239     /* Using an XOR gate to flip the SEL pin for the MUXs*/
240     mcc_config.sel_status ^= 0x1;
241     gpio_set_level(SEL_PIN, mcc_config.sel_status);
242     ESP_LOGI(TAG, "SEL Status: %d at %d", mcc_config.sel_status, SEL_PIN);
```

```
243     vTaskDelay(1000/portTICK_PERIOD_MS);
244
245     /* Power up! */
246     gpio_set_level(CAM_PIN_PWDN, 1);
247     if (ret != ESP_OK) {
248         ESP_LOGW(TAG, "GPIO set level returned badly");
249     } else {
250         ESP_LOGW(TAG, "GPIO set level OK. PSWDN is 1.");
251     }
252     vTaskDelay(100/portTICK_PERIOD_MS);
253
254     /* Initialise */
255     ret = init_camera();
256     if (ret != ESP_OK) {
257         ESP_LOGW(TAG, "Camera init returned badly");
258     } else {
259         ESP_LOGW(TAG, "Camera init OK");
260     }
261     vTaskDelay(100/portTICK_PERIOD_MS);
262
263     return ESP_OK;
264 }
265
266 esp_err_t sys_sd_var_setup()
267 {
268     /* To reduce file size when saving */
269     change_pixformat_to_jpeg();
270
271     esp_fill_random(&rnd, sizeof(rnd));
272     sprintf(rnd_str,"%u", rnd);
273
274     ESP_LOGI(TAG, "SD Ready for saving!");
275     return ESP_OK;
276 }
277
278 esp_err_t sys_sd_save(camera_fb_t* fb){
279     esp_err_t ret;
280
281     /* Set the first part of the path which is the mount point '/sdcard' followed by another '/' */
282     strcpy(path,MOUNT_POINT"/");
283
284     /* Attach the random string obtained from the setup function */
285     strcat(path, rnd_str);
286
287     /* Based on the SEL pin status, attach the L or R as a suffix to separate the two images, as well as the file extension
        */
288     switch (mcc_config.sel_status) {
289     case 0x0:
290         strcat(path, "L.jpg");
291         break;
292     case 0x1:
```

```
293         strcat(path, "R.jpg");
294     default:
295         break;
296     }
297 
298     /* Write frame buffer to SD card */
299     ret = sd_write_arr(path, fb);
300 
301     /* Error check if file was not written correctly */
302     if (ret != ESP_OK){
303         ESP_LOGW(TAG, "File not written correctly");
304         return ESP_FAIL;
305     }
306 
307     return ESP_OK;
308 }
```

Listing 18: MCC Code - sd.h

```
#pragma once

#include <string.h>
#include <sys/unistd.h>
#include <sys/stat.h>
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"
#include "esp_camera.h"
#include "esp_err.h"

/* SD card pins used in SPI communication */
#define SD_DATA0    2
#define SD_DATA3    13
#define SD_CLK      14
#define SD_CMD      15

/* Defined to be used as SPI pins */
#define PIN_NUM_MISO    SD_DATA0
#define PIN_NUM_CS      SD_DATA3
#define PIN_NUM_CLK     SD_CLK
#define PIN_NUM_MOSI    SD_CMD

/* Folder to mount at and WiFi file name */
#define MOUNT_POINT         "/sdcard"
#define WIFICRED_FILE_NAME  MOUNT_POINT"/wificred.txt"
#define MAX_CHAR_LINE       30

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief Initialise the SD card, save the picture and de-initialise
 *
 * @param fb Pointer to the camera frame buffer
 *
 * @return ESP_OK on success
*/
esp_err_t sd_init();

/**
 * @brief Writes all of the image data to a file
 *
 * @param path File path
 * @param fb Pointer to the camera frame buffer
 *
 * @return ESP_OK on success
*/
esp_err_t sd_write_arr(const char *path, camera_fb_t *fb);

```

```
51 /**
52  * @brief De—initialise the SD card, variables used are the globals defined in sd.c
53  *
54  * @return ESP_OK on success
55 */
56 esp_err_t sd_deinit();
57 
58 /**
59  * @brief Read WiFi credentials from the file wificred located on the SD card
60  *
61  * @return ESP_OK on success
62 */
63 esp_err_t read_wifi_credentials();
64 
65 #ifdef __cplusplus
66 }
67 #endif
```

Listing 19: MCC Code - sd.c

```
#include "sd.h"

const static char* TAG = "WULPSC - SD";

/* Default structure initializer for SD over SPI driver */
sdmmc_host_t host = SDSPI_HOST_DEFAULT();

/* Variable to store SD card information */
sdmmc_card_t *card;

/* Directory to mount to */
const char mount_point[] = MOUNT_POINT;

/* Initialise SSID and Password variables of size defined by MAX_CHAR_LINE */
char sd_ssid[MAX_CHAR_LINE];
char sd_pswd[MAX_CHAR_LINE];

esp_err_t sd_write_arr(const char *path, camera_fb_t *fb)
{
    size_t ret;
    ESP_LOGI(TAG, "Opening file |%s|", path);

    /* Open or create file located in the path string. Use binary write to write bytes instead of characters*/
    FILE *f = fopen(path, "wb");
    if (f == NULL) {
        ESP_LOGE(TAG, "Failed to open file for writing");
        return ESP_FAIL;
    }

    /* Write the frame biffer to file */
    ret = fwrite(fb->buf, sizeof(char), fb->len, f);
    ESP_LOGI(TAG,"Wrote %zu elements out of %d requested", ret, fb->len);

    /* Close the file */
    fclose(f);
    ESP_LOGI(TAG, "File written");

    return ESP_OK;
}

esp_err_t sd_init()
{
    esp_err_t ret;

    /* File system configuration to use for the SD card */
    esp_vfs_fat_sdmmc_mount_config_t mount_config = {
        .format_if_mount_failed = false,
        .max_files = 10,
        .allocation_unit_size = 16 * 1024   // for 16 GB
    };
```

```
51
52      ESP_LOGI(TAG, "Initializing SD card using SPI peripheral");
53      /* Initialise SPI bus config */
54      spi_bus_config_t spi_bus_cfg = {
55          .mosi_io_num = PIN_NUM_MOSI,
56          .miso_io_num = PIN_NUM_MISO,
57          .sclk_io_num = PIN_NUM_CLK,
58          .quadwp_io_num = -1,
59          .quadhd_io_num = -1,
60          .max_transfer_sz = 4092,
61      };
62
63      /* Initialise and check if succesful */
64      ret = spi_bus_initialize(host.slot, &spi_bus_cfg, SDSPI_DEFAULT_DMA);
65      if (ret != ESP_OK) {
66          ESP_LOGE(TAG, "Failed to initialize bus.");
67          return ret;
68      }
69
70      /* Initializes the slot without card detect (CD) and write protect (WP) signals */
71      sdspi_device_config_t slot_config = SDSPI_DEVICE_CONFIG_DEFAULT(); // uses GPIO13 for CS
72      slot_config.gpio_cs = PIN_NUM_CS;
73      slot_config.host_id = host.slot;
74
75      /* Finally mount the SD card using the declared SPI bus*/
76      ESP_LOGI(TAG, "Mounting filesystem");
77      ret = esp_vfs_fat_sdspi_mount(mount_point, &host, &slot_config, &mount_config, &card);
78
79      if (ret != ESP_OK) {
80          if (ret == ESP_FAIL) {
81              ESP_LOGE(TAG, "Failed to mount filesystem. "
82                       "If you want the card to be formatted, set the CONFIG_EXAMPLE_FORMAT_IF_MOUNT_FAILED
    menuconfig option.");
83          } else {
84              ESP_LOGE(TAG, "Failed to initialize the card (%s). "
85                       "Make sure SD card lines have pull-up resistors in place.", esp_err_to_name(ret));
86          }
87          return ret;
88      }
89      ESP_LOGI(TAG, "Filesystem mounted");
90
91      /* Card has been initialized, print its properties */
92      sdmmc_card_print_info(stdout, card);
93
94      return ret;
95  }
96
97  esp_err_t sd_deinit()
98  {
99      /* All done, unmount partition and disable SPI peripheral */
100     esp_vfs_fat_sdcard_unmount(mount_point, card);
```

```
    ESP_LOGI(TAG, "Card unmounted");

    /* Deinitialize the bus after all devices are removed */
    spi_bus_free(host.slot);
    return ESP_OK;
}

esp_err_t read_wifi_credentials(){

    /* Initialise arrays with ACII null escape character '\0' */
    memset(sd_ssid, '\0', sizeof(sd_ssid));
    memset(sd_pswd, '\0', sizeof(sd_pswd));

    /* Read the wifi credentials text file */
    const char *path = WIFICRED_FILE_NAME;
    ESP_LOGI(TAG, "Reading file %s", path);
    FILE *f = fopen(path, "r");
    if (f == NULL) {
        ESP_LOGE(TAG, "Failed to open file for reading");
        return ESP_FAIL;
    }

    /**
     * Get the two lines from the file. First line is used for the SSID,
     * second line is used for the Password.
    */
    fgets(sd_ssid, sizeof(sd_ssid), f);
    fgets(sd_pswd, sizeof(sd_pswd), f);
    fclose(f);

    /**
     * Pointer to the character locations.
     * Used to strip the '\n' and '\r' in the strings.
     * If at the pos pointer location an '\r' or '\n' is detected, it is replaced by the
     * null terminator.
    */
    char *pos;
    pos = strchr(sd_ssid, '\r');
    if (pos) {
        *pos = '\0';
    }
    pos = strchr(sd_ssid, '\n');
    if (pos) {
        *pos = '\0';
    }
    pos = strchr(sd_pswd, '\r');
    if (pos) {
        *pos = '\0';
    }
    pos = strchr(sd_pswd, '\n');
    if (pos) {
```

```
152         *pos = '\0';
153     }
154 
155     ESP_LOGI(TAG, "SSID from file: '%s'", sd_ssid);
156     ESP_LOGI(TAG, "Pswd from file: '%s'", sd_pswd);
157 
158     return ESP_OK;
159 }
```

Listing 20: MCC Code - wifi.h

```
#pragma once

#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "esp_log.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "sd.h"

#define SD_CRED    0
   #if !SD_CRED
      #define ESP_WIFI_SSID      "EE-5PRW29"
      #define ESP_WIFI_PASS      "uMLrF4KWmQvD6G4n"
   #endif

#define ESP_MAXIMUM_RETRY  3
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

#ifdef __cpluplus
extern "C" {
#endif

/**
 * @brief General required calls to initialise WiFi functionality for an ESP32.
*/
void wifi_init_general();

/**
 * @brief Used to set the CONNECTED or FAIL bits for the event.
 *
 * @param arg General arguement pointer
 * @param event_base Event type
 * @param event_id Event ID
 * @param event_data Data received from the event.
*/
void event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data);

/**
 * @brief Initialise Station Mode
 *
 * @return ESP_OK on success, ESP_FAIL on failing to connect to the WiFi network
*/
esp_err_t wifi_init_sta(void);

#ifdef __cplusplus
}
#endif
```

Listing 21: MCC Code - wifi.c

```
#include "wifi.h"

static const char *TAG = "WULSC - Wi-Fi";

/* Variable to store amount of retries */
static int s_retry_num = 0;

/* FreeRTOS event group to signal when connected */
static EventGroupHandle_t s_wifi_event_group;
static esp_netif_t *sta_netif;
static wifi_init_config_t cfg;

/* Variables for the string credentials defined in sd.c */
extern char sd_ssid[MAX_CHAR_LINE];
extern char sd_pswd[MAX_CHAR_LINE];

void wifi_init_general()
{
    /* Standard initialisations to enable WiFi on an ESP32 */
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    sta_netif = esp_netif_create_default_wifi_sta();
    assert(sta_netif);
    cfg = (wifi_init_config_t) WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_start());
}

void event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data)
{
    /* Check for each WiFi event and act accordingly */
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        /* Retry as many times as defined by the directive */
        if (s_retry_num < ESP_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
```

```
51 }
52
53 esp_err_t wifi_init_sta(void)
54 {
55     esp_err_t ret;
56
57     /* Initialise events for STA mode */
58     s_wifi_event_group = xEventGroupCreate();
59     esp_event_handler_instance_t instance_any_id;
60     esp_event_handler_instance_t instance_got_ip;
61     ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &event_handler,
       NULL, &instance_any_id));
62     ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &event_handler,
       NULL, &instance_got_ip));
63
64 /* Directive set to use credentials from the SD card. */
65 #if SD_CRED
66     wifi_config_t wifi_config = {
67         .sta = {
68             .ssid       = "",       // Initialise the values with empty string due to a bug
69             .password   = ""
70         }
71     };
72
73     /* Copy the strings from the text file to the config */
74     strcpy((char*)wifi_config.sta.ssid, sd_ssid);
75     strcpy((char*)wifi_config.sta.password, sd_pswd);
76
77 /* If set to 0 use the directives. Mainly used for testing. */
78 #else
79     wifi_config_t wifi_config = {
80         .sta = {
81             .ssid       = ESP_WIFI_SSID, // this works due to initialisation and not value assignment
82             .password   = ESP_WIFI_PASS
83         }
84     };
85 #endif
86
87
88     ESP_LOGI(TAG, "Detected SSID: %s", wifi_config.sta.ssid);
89     ESP_LOGI(TAG, "Detected Password: %s", wifi_config.sta.password);
90
91     /* Set credentials and try to connect. It also tries to connect in the event_handler() */
92     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
93     ret = esp_wifi_connect();
94
95     /* Error checking */
96     if (ret != ESP_OK) {
97         ESP_LOGE(TAG, "Error with Wi-Fi connection");
98         return ESP_FAIL;
99     }
```

```
100
101     /**
102      * Waiting until either the connection is established (WIFI_CONNECTED_BIT) or connection failed for the maximum
103      * number of re-tries (WIFI_FAIL_BIT). The bits are set by event_handler() (see above)
104     */
105     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group, WIFI_CONNECTED_BIT | WIFI_FAIL_BIT, pdFALSE,
          pdFALSE, portMAX_DELAY);
106
107
108     /* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event actually happened.
          */
109     if (bits & WIFI_CONNECTED_BIT) {
110         ESP_LOGI(TAG, "connected to Access Point SSID:%s password:%s", wifi_config.sta.ssid, wifi_config.sta.password);
111         return ESP_OK;
112     } else if (bits & WIFI_FAIL_BIT) {
113         ESP_LOGW(TAG, "Failed to connect to SSID:%s, password:%s", wifi_config.sta.ssid, wifi_config.sta.password);
114         return ESP_FAIL;
115     } else {
116         ESP_LOGE(TAG, "UNEXPECTED EVENT");
117         return ESP_FAIL;
118     }
119 }
```

# User Guide

**Setting up the MCC and WUC**

**MCC:** SD Card of the MCC must have a text file named “wificreds.txt” at the root directory, where the WiFi Access Point SSID and Password must be written in two separate lines, separated by pressing the Enter key. This is signal the end of the lines by entering invisible escape characters. An example file is shown below, CRLF are the invisible escape characters that are inputted by pressing enter.

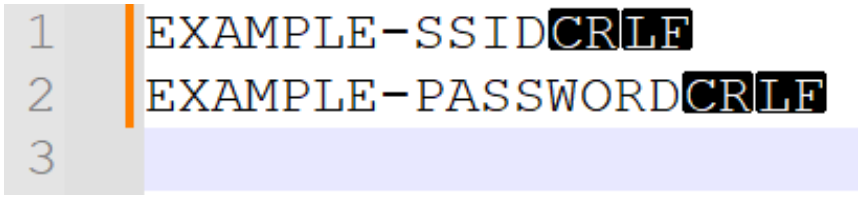



*Figure 1. Example "wificreds.txt" file placed in the root directory of the SD card on the MCC.*

To connect to a new network, new credentials must simply be written in the place of the old ones on the SD card. Currently loaded credentials are,

***SSID: pap***

***Password: 12345678***

**WUC:** At the very first boot of the WUC, the Smart Config procedure with EspTouch is started. A device already connected to the desired network must have EspTouch installed, and have GPS enabled on their device. By inputting the password and pressing confirm, special signals will be sent to allow the WUC to connect to the desired network. The device with EspTouch installed must have 2.4 GHz WiFi capabilities or else Smart Config won't work.

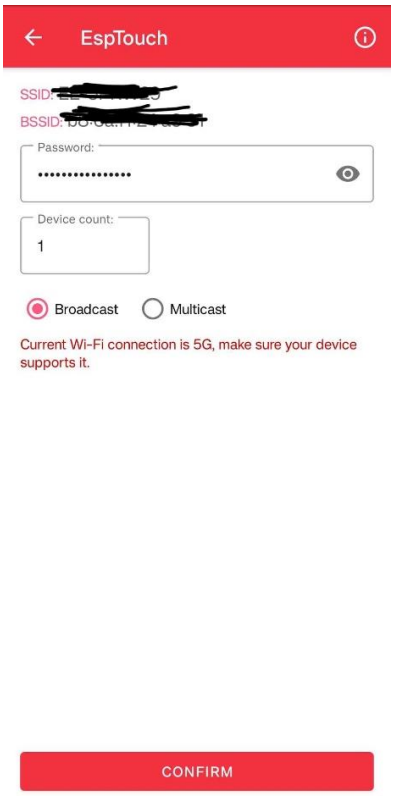



*Figure 2. EspTouch example screen.*

To reset the currently connected network, a device must be connected to the existing network, and be used to call the **/reset** URI handler on the WUC using its IP Address. Opening the browser on the device and typing (replace with actual IP address) **http://192.168.x.xxx:19520/reset**, deletes the existing credentials and reboots the device.

## Configuring and hosting the back-end/front-end

To self-host the backend, clone the backend using git:

git clone https://github.com/DDQuin/stereo-camera-backend.

Then making sure you have python 3.10 installed run

pip install –r requirements.txt

within the terminal at the folder location to install the python libraries required. MATLAB is also required for the backend to work.

A MongoDB Atas account also needs to be setup with a database collection URL. To start the backend, after connecting the MCC and WUC to same network use the command below.

WUC_URL= ”http://192.168.x.x:xxxx” MCC_URL="http://192.168.x.x:xxxx” MONGO="mongodb+srv://" uvicorn main:app --reload --host 0.0.0.0

Put the correct URL’s in the sections with =.

For the frontend, also git clone https://github.com/sawtoise/wulpsc-frontend. While at the folder directory, use the npm install command in the terminal and then npx vite –host to start the frontend.

## Taking a photo

Make sure the hardware is powered on and not sleeping, by checking to see the label underneath the last photo reads 'Awake'

Once the object you want to capture is positioned in-front of the camera, click 'Capture'

Wait a few seconds, and you should see the image appear in real-time. You can analyse the photo for defects by creating a bounding box around an area of choice. Clicking 'Analyse' will then display the dimensions of the object.


'Previous' and 'Next' can be used to look at the stereo depth map, showing the output produced by the image processing algorithm.

## Adding a schedule

The time picker shown can be used to add scheduled times at which photos are to be captured.

Select a time, and then add it by clicking the '+' button. The selected time should appear in a list below.

The back-end will schedule a capture on all the times displayed in the list, and if automatic sleep is enabled, the system will sleep until the next capture.

If automatic sleep is enabled, the next wake-up timestamp will be displayed below the last photo.

## Settings and parameters

Most settings relate to camera parameters and have been fine-tuned for the venue. A brief description of this can be seen in the table below. However, there are some noteworthy settings unrelated to camera quality.

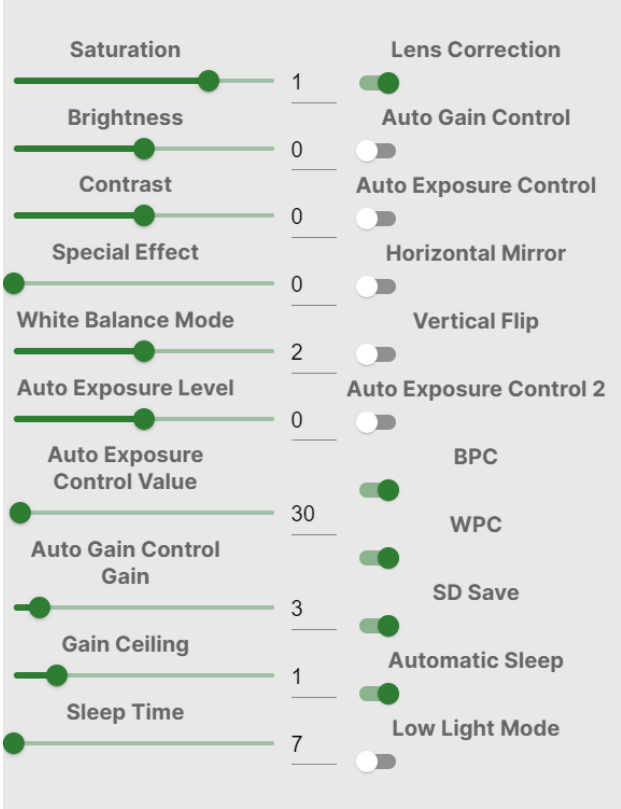
| Parameter | Range | Type | Description |
|---|---|---|---|
| Saturation | -2, 2 | Int | Set saturation |
| Brightness | -2,2 | Int | Set brightness |
| Contrast | -2,2 | Int | Set contrast |
| Special Effect | 0 – no effect<br>1 – Negative<br>2 – Grayscale<br>3 – Red Tint<br>4 – Green Tint<br>5 – Blue Tint<br>6 – Sepia | Int | Set a special effect |
| White Balance Mode | 0 – Auto<br>1 – Sunny<br>2 – Cloudy<br>3 – Office<br>4 – Home | Int | Set white balance mode |
| Auto Exposure Level | -2,2 | Int | Set auto exposure level |
| Auto Exposure Control Value | 0,1200 | Int | Set auto exposure control value |
| Auto Gain Control Gain | 0,30 | Int | Set auto gain control value |
| Gain Ceiling | 0,6 | Int | Set gain ceiling |
| Sleep Time | 1,2628000 | Int | Set sleep time in seconds |
| Lens Correction | True/False | Boolean | Toggle lens correction |
| Auto Gain Control | True/False | Boolean | Toggle auto gain control |
| Auto Exposure Control | True/False | Boolean | Toggle auto exposure control |
| Horizontal Mirror | True/False | Boolean | Toggle horizontal flip |
| Vertical Flip | True/False | Boolean | Toggle vertical flip |
| Auto Exposure Control 2 | True/False | Boolean | Toggle auto exposure control 2 |
| BPC | True/False | Boolean | Toggle black pixel correction |
| WPC | True/False | Boolean | Toggle white pixel correction |
| SD Save | True/False | Boolean | Toggle SD Save mode |
| Automatic Sleep | True/False | Boolean | Toggle automatic sleep |

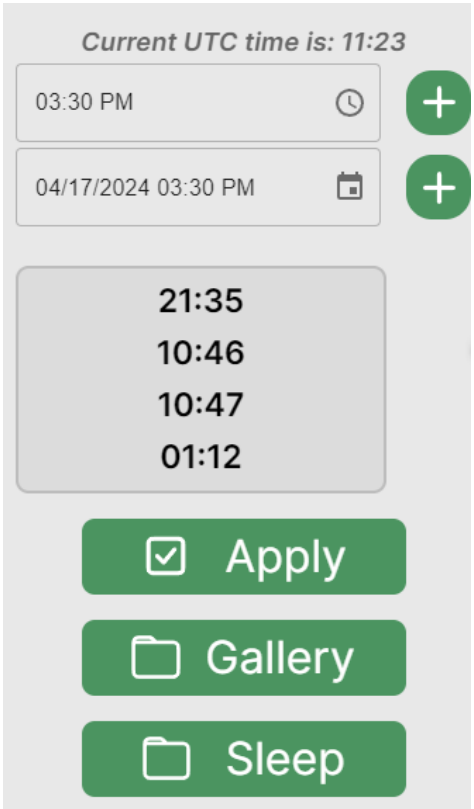
Sleep time sets how long the device will sleep for in seconds if the sleep button is clicked.

Automatic sleep when enabled results in the device sleeping automatically after each scheduled capture until the next, turning this off means the device will always stay awake unless manually set to sleep by the user.

SD Save forces the microcontroller to take 4 photos instead of 2, intended to be enabled when the device has an SD card inserted. This is due to issues with parameters not being applied on the first capture, requiring multiple to ensure the correct parameters are being used.

Clicking apply will result in a dialog box being displayed, showcasing all the changes made before confirmation.




## Manually sleeping

You can manually set the system to sleep after each capture via the sleep button. The time the system will sleep for will be set by the ‘sleep time’ parameter. A confirmation dialog will show up as a prompt, displaying the time the system will sleep for along with the timestamp of when it will next wake up.

## Gallery

The gallery button can be clicked to view old photos. Any photo can be analysed as explained previously, via a bounding box.

**Troubleshooting**

An unstable or weak connection can greatly affect the system, therefore make sure a strong connection to the network exists. Firewalls on the devices used to access the system, or even the actual access points will affect network speed and results. To have the best results using the WULPSC it is recommended to port forward the device IP addresses or even set them to the DMZ of the router. Please refer to your internet service provider for how to do so.