OpenAR 2.1 Documentation

Alpo Ranne & Ville Päivinen

August 2025

Department of Physics and Mathematics

University of Eastern Finland

1 Introduction

In the OpenAR 2.0 documentation, we explored the foundational design of the device, including the possibility of integrating additional sensors to enhance functionality. With OpenAR 2.1 we will exemplify the capabilities of the device with a thermal camera live feed – a real time thermal imaging using a FeatherS3 development board.

This documentation walks you through the steps to create OpenAR 2.1 augmented reality glasses with a thermal camera live feed. The optics and designs used for OpenAR 2.1 mimic OpenAR 2.0 closely. Therefore, reading the documentation for OpenAR 2.0 is required to understand the platform, as this documentation focuses only on what is new. This includes new electronics and some new 3D-printed parts to fit the thermal camera and the new electronics onto the OpenAR 2.1 system.

We will first go through the new electronics, including the thermal camera we used. Then, we will guide you through programming the FeatherS3 development board to make the whole thermal camera circuit work. This part of the documentation will be in-depth, but you are not required to understand all of it, as we provide the code that runs the whole show and an easy-to-follow guide. For those who do want in-depth information: you are welcome! Finally, we will go through the new 3D models and the modifications made to the OpenAR 2.0 platform to fit it with this new tech.

Welcome to OpenAR 2.1 !

2 Thermal Camera Live Feed

Thermal imaging is an interesting application for the OpenAR glasses, allowing the device to visualize heat signatures in real time. In this section, we will demonstrate how to integrate a thermal camera module with the OpenAR platform using the FeatherS3 development board.

The low-resolution image from the thermal camera will be upscaled by, for example, bicubic interpolation to better fill the relatively small field-of-view (FOV) of the glasses. Unfortunately, the FOVs of the camera and the glasses don't match, so we can't get an actual thermal overlay, but rather a picture-in-picture kind of situation.

2.1 Modifications/Additions to OpenAR 2.0

- **Processor:** Processing the thermal camera data in real(-ish) time requires a bit more oomph, so the Blue Pill will have to go in favor of a more powerful board. We'll be using the Unexpected Maker FeatherS3 due to its higher clock frequency, integrated battery charger, Bluetooth and Wi-Fi capabilities, and memory size (both flash and RAM). Note that in addition to the 8 MB of PSRAM, FeatherS3 also features 512 kB of SRAM, which we use exclusively, because it's faster.

  If you know your way around embedded systems, you have a wealth of options to choose from when it comes to development boards. Do note that the program used about 230 kB of RAM, and on this board only around 328 kB of the manufacturer stated 512 kB are at the user's disposal, so it is recommended to steer clear of boards with less than 512 kB of RAM for this application. Flash utilization was about 1.2 MB, so there's plenty of room left to expand the program.

- **Thermal camera:** For the camera, we used a Grove thermal imaging camera (MLX90640) with a $55° \times 35°$ field of view (FOV) and a 32x24 pixels resolution. It was cheap enough for our purposes, the resolution wasn't too high for a cheap MCU to process at an acceptable frame rate, and it was very easy to connect to the FeatherS3 via the I2C connections.

- **Wires:** You will need some type of wires to connect the thermal camera to the board. Whether you choose to purchase a Grove cable or just jumper wires etc. is completely up to you.

2.2 Electronic connections

Since we have a new development board, we will have to re-do the connections. This time it's quite a lot simpler, since FeatherS3 has a built-in battery charger and voltage regulator. Below are the connections between FeatherS3, the display (same one as in OpenAR 2.0), and the thermal camera.

FeatherS3		Display
GND	→	GND
3V3	→	VCC
SCK (IO36)	→	SCL
MO (IO35)	→	SDA
IO5	→	RES
IO6	→	DC
IO14	→	BLK

FeatherS3		MLX90640
GND	→	GND
3V3	→	VCC
SDA (IO8)	→	SDA
SCL (IO9)	→	SCL

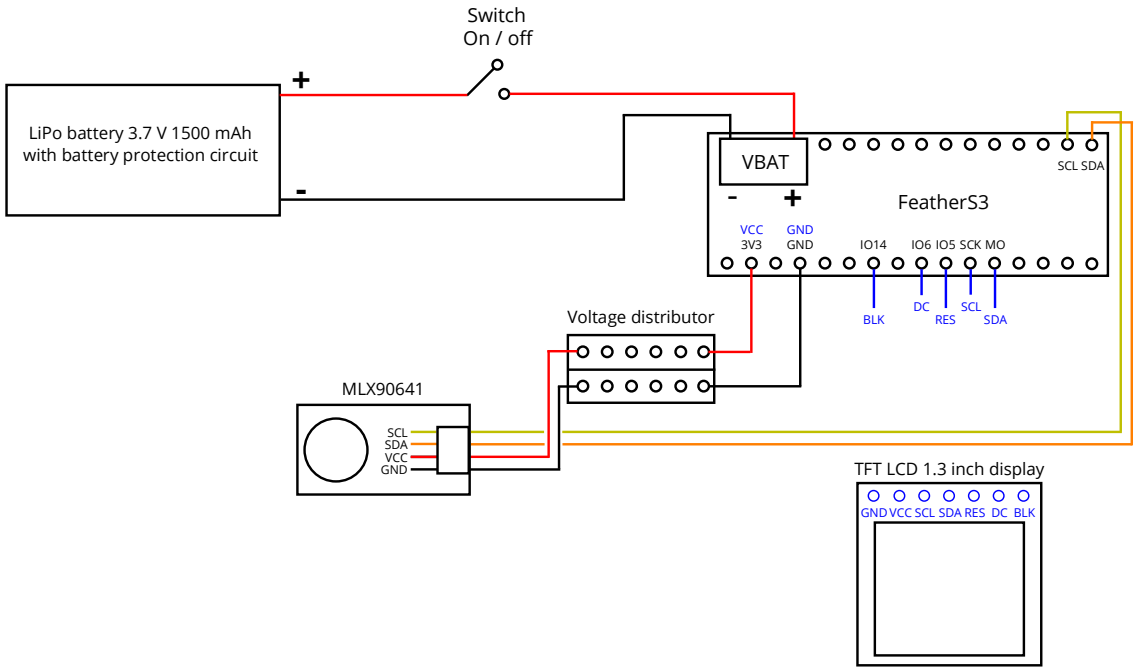

Image 2.1. Circuit diagram for OpenAR 2.1. Zoom to enlarge.

The voltage distributor in the image above also powers the LCD display. This is not shown to keep the image simpler.

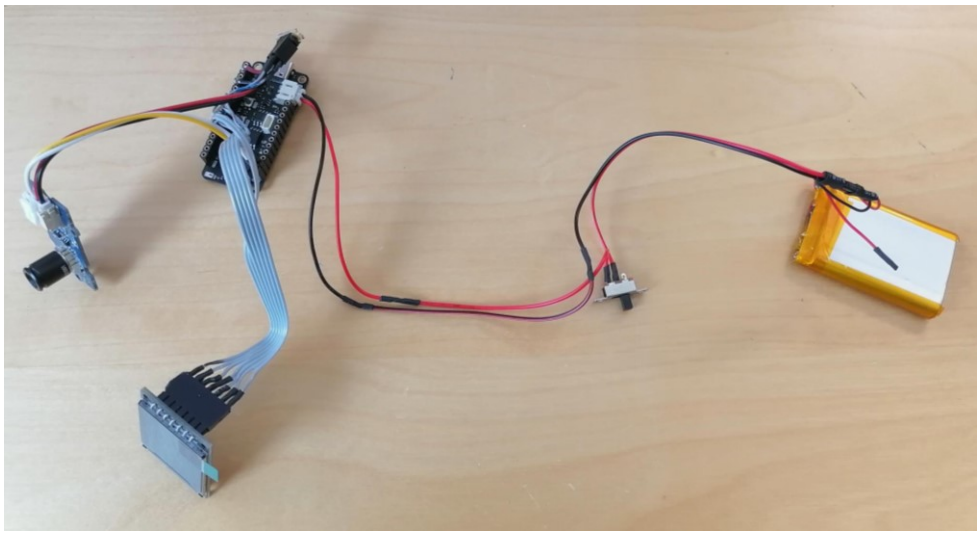

Image 2.2. OpenAR 2.1 circuit.

2.3 Programming the FeatherS3 Development Board

This section will be split into two parts: a guide for beginners that is more straightforward and a guide for intermediate users who already have some understanding of programming ESP-based development boards. For the beginner guide, we will show you how to upload the thermal camera code we have made onto your own FeatherS3 board using Visual Studio. And for intermediate users we will talk about the process behind how we ended up using Visual Studio and how the thermal camera program works. Reading both parts is, of course, recommended as it gives you a better understanding of OpenAR 2.1.

For beginners

Download VS Code once you have built your OpenAR 2.1 circuit (image above) and follow the instructions below:

1. Open VS Code and download PlatformIO IDE extension. The shortcut for the extension menu is CTRL+SHIFT+X. Search for PlatformIO and install the extension.

2. Inside VS Code go to file → open folder and browse to the OpenAR_Thermal_Camera folder and open it. PlatformIO will start configuring the project which may take some time.

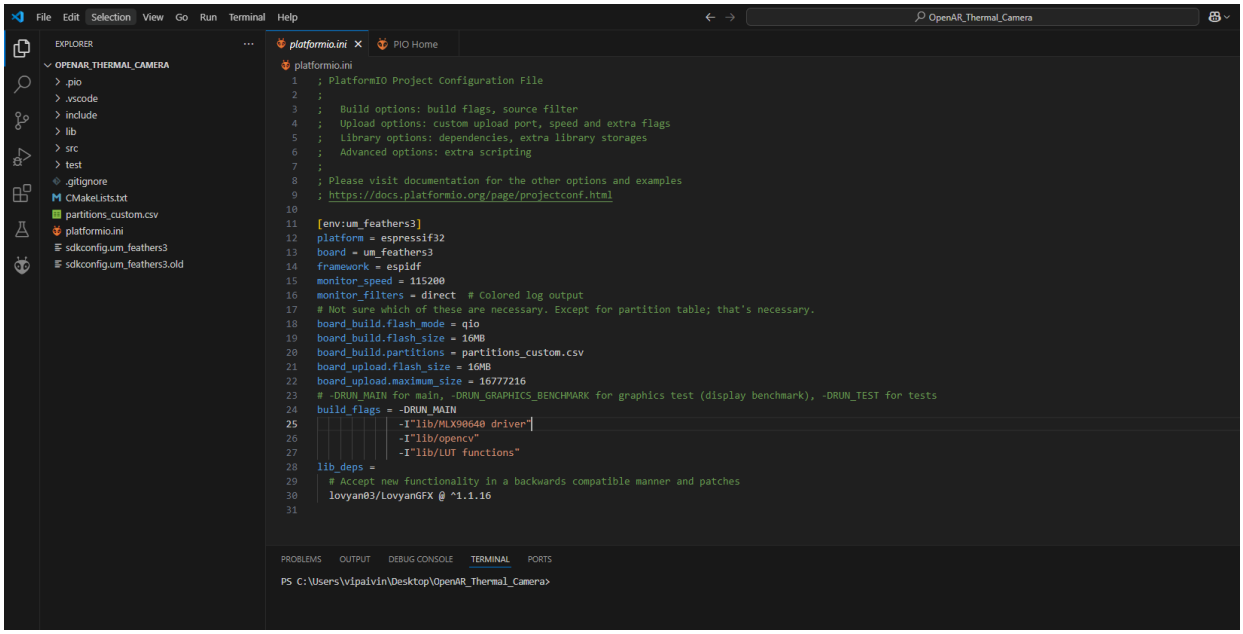

Image 2.3. VS Code environment open with the OpenAR_Thermal_Camera code.

3. PlatformIO automatically downloads a folder that we don't need as we already have it included. Building the code now would lead to fatal errors, so we need to delete it. Inside VS Code browse to:

   .pio → libdebs → LovyanGFX → src → lgfx → v1 → platforms, and delete the folder named opencv.

4. Build the code by clicking the checkmark icon in the bottom left corner of the VS Code window. You will get some yellow warning messages, but those do not affect the functionality of OpenAR 2.1. You should not get any red error messages, and the build should finish successfully after some time.

5. Connect the FeatherS3 to your computer using a USB-C cable and click the upload button (it looks like an arrow pointing to the right). The upload should be successful, but if you get an error message, try uploading the code again, but this time once you see the message "Looking for upload port…" in the terminal window, press down and hold the BOOT button on the FeatherS3. While holding the BOOT button, press and then release the RESET button, and then finally release the BOOT button. This sequence should allow VS Code to upload the program to your FeatherS3.

6. Once the upload has finished successfully, disconnect the USB-C cable from FeatherS3 and turn it on. The thermal camera program should now work!

Note: If you want to change the minimum and maximum temperatures that are drawn onto the display, simply open the main.cpp in the “src” folder. On line 34 and 35 you can find MIN_TEMP and MAX_TEMP variables. Here you can change the values to the desired temperature.

For intermediate users

Programming an ESP32-based board like the FeatherS3 is straightforward, thanks to a wide variety of frameworks and tools available. On the flip side, choosing from the variety of methods can be a bit daunting at first. For this project, we will need to write and upload code to the board, so here’s a quick overview of the options.

Frameworks

An ESP board can be programmed using several different firmware and programming languages. In C/C++ languages, the options are:

- **Arduino framework:** Perfect for beginners or quick prototyping with a simple and user-friendly API. Lightweight and has an extensive library ecosystem.
- **ESP-IDF (Espressif IoT Development Framework):** For more control over the device, we can opt for the official framework provided by Espressif. It is a more heavyweight and powerful option with a (lot) steeper learning curve, but it allows to optimize performance, utilize multiple core development boards (like the two cores on FeatherS3), and overall tailors the device to specific needs better.

Since our application is performance critical, we will be opting for the latter option. The board can also be programmed using CircuitPython, JavaScript, LUA, etc.

Integrated Development Environments (IDEs)

- **Arduino IDE:** The simplest option with built-in support for the Arduino framework with no native support for ESP-IDF. Sufficient for beginners and basic applications.
- **Espressif IDE:** Eclipse based IDE specifically designed for the ESP-IDF framework. No support for Arduino framework or non-ESP development boards.
- **Visual Studio Code with PlatformIO:** Supports both the Arduino and ESP-IDF frameworks, supports various embedded boards beyond ESP, built-in dependency management system with tons of libraries, etc. Surprisingly straightforward to use despite the wealth of advanced features available.

I, the programmer, dabbled around with both the Arduino and the ESP-IDF frameworks, so VS Code with PlatformIO was the natural choice.

Program Overview

Now that the IDE is set up, it’s time to start writing some code. Our goal is to process the raw data from the thermal camera in real-time and display it as a live feed on the OpenAR device. While this may sound a little complex, the program structure is actually quite simple.

The hard lifting is mainly handled by the display driver and the thermal camera driver, which handle the software-hardware communication. For the display, we use LovyanGFX, and for the camera we

use Melexis' own driver. The I2C functions need to be implemented by the user, which has been done for the ESP-IDF framework by linkineo.

You could download these through the above links, but I did quite a lot of performance optimization for the thermal camera driver, and some changes were also necessary for LovyanGFX to co-operate with our setup, so I recommend using the drivers included in my project folder.

For more experienced folk, the details of the program are probably quite easy to understand by reading through the code. For the uninitiated, here we will just provide a high-level overview of what the program does:

1. **Initialization:** The program starts by including the necessary libraries, allocating framebuffers, configuring I2C and initializing the peripherals (camera and display). Calibration constants and configuration parameters are also extracted from the thermal camera's EEPROM.
2. **Data capture:** Each frame begins with reading the raw frame data from the MLX90640. This includes the 768 (32x24) pixel values along with some calibration values etc.
3. **Temperature calculation:** The raw frame data is fed into the manufacturer's driver function, along with the EEPROM parameters. The output of this function is a 32x24 array of temperature values.
4. **Interpolation:** Since the resolution of the camera is far lower than the display, the raw 32x24 frame is upscaled to closer match the display resolution using a bicubic interpolation algorithm. To do this, we include an ESP-IDF-friendly clone of OpenCV. This also gives us the freedom to choose bilinear interpolation for more performance at the cost of image quality, or Lanczos interpolation for the opposite compromise.
5. **Color mapping:** To get a displayable version of the temperature frame, we define a colormap (turbo in our case) with 256 color values in a format that the display driver understands, and calculate a corresponding index (0-255) for each temperature value in the frame.
6. **Display output:** The RGB image is pushed to the display via SPI. This process is optimized by leveraging FeatherS3's DMA (direct memory access) capabilities.

Steps 2-6 loop until the power is switched off. As a final touch, a crosshair is marked at the center of the frame and the central temperature (in Celsius) is displayed above the frame.

Image 2.3. Thermal camera image of a hand.

3 3D-printed parts

OpenAR 2.1 features some new 3D printable parts. Mainly holders for the thermal camera and the FeatherS3 development board. This time we have less electronics, thanks to the in-built charger and voltage regulator on the FeatherS3 board, so this means less parts to 3D print. The optical parts are identical to OpenAR 2.0, so it is recommended to follow the 2.0 build guide first. Below is a list of the new and/or modified 3D-printed parts used for OpenAR 2.1.

- Modified OpenAR 2.0 headband
- Modified optics outer shell to fit the thermal camera
- Modified headband covers
- A casing for the thermal camera (consists of three parts)
- Thermal camera holder
- A small mount for the FeatherS3 board
- Locking mechanism for the headband covers

More information about the parts can be found within the “guide for optics models” and “guide for headband models” folders inside the OpenAR 2.1 documentation. It is also recommended to view the f3d files included in the documentation, as the models in the Fusion360 projects are assembled as the OpenAR 2.1 parts would be. This can give a better visual representation of an assembled OpenAR 2.1. Below are a few images of the unique parts made for 2.1 version of OpenAR.

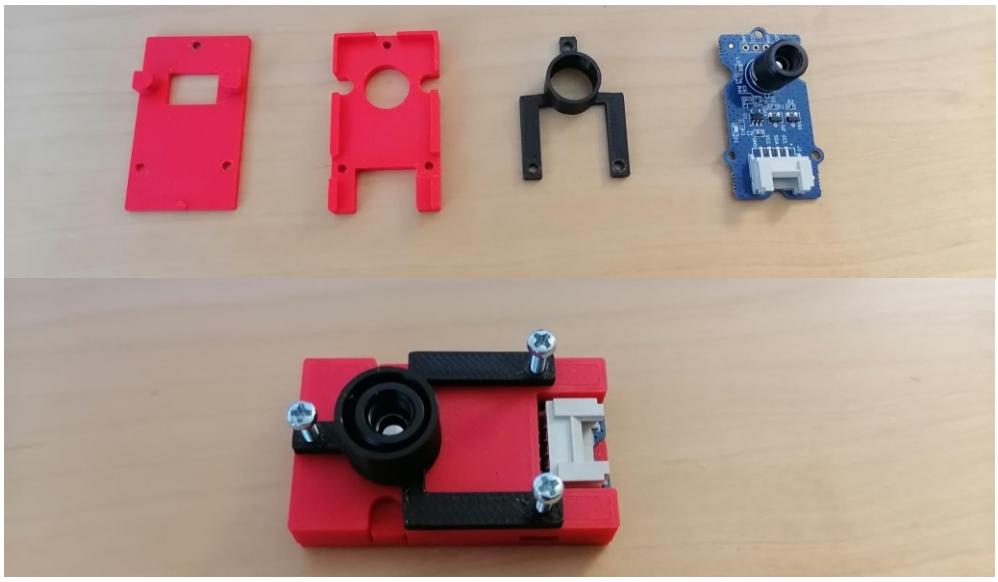

Image 3.1. The thermal camera casing disassembled and assembled.

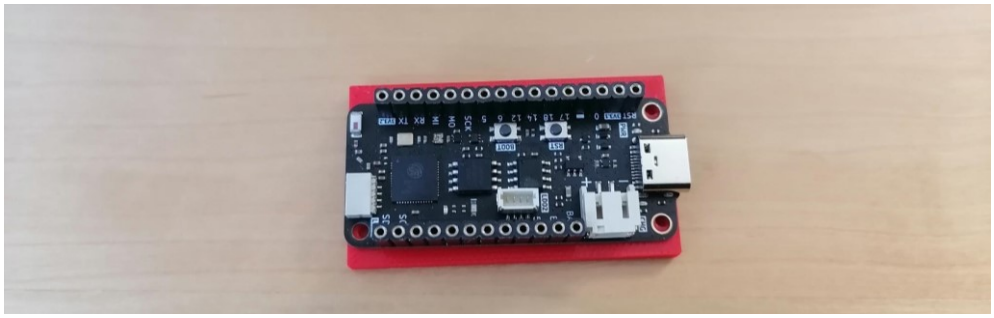

Image 3.2. The mount of FeatherS3 development board.

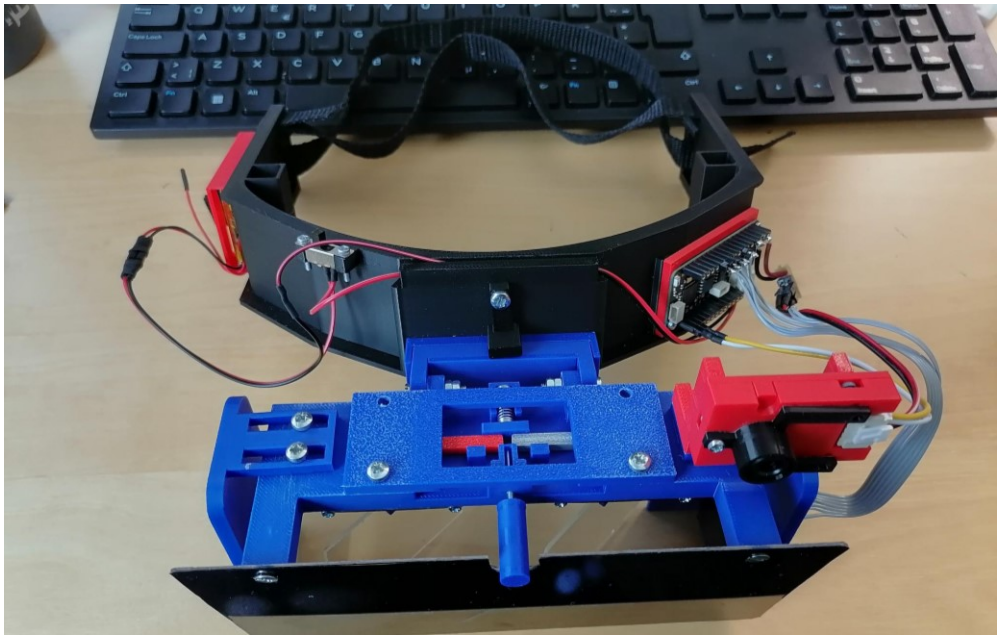

Image 3.3. Electronics on the OpenAR 2.1 headband and optical setup.

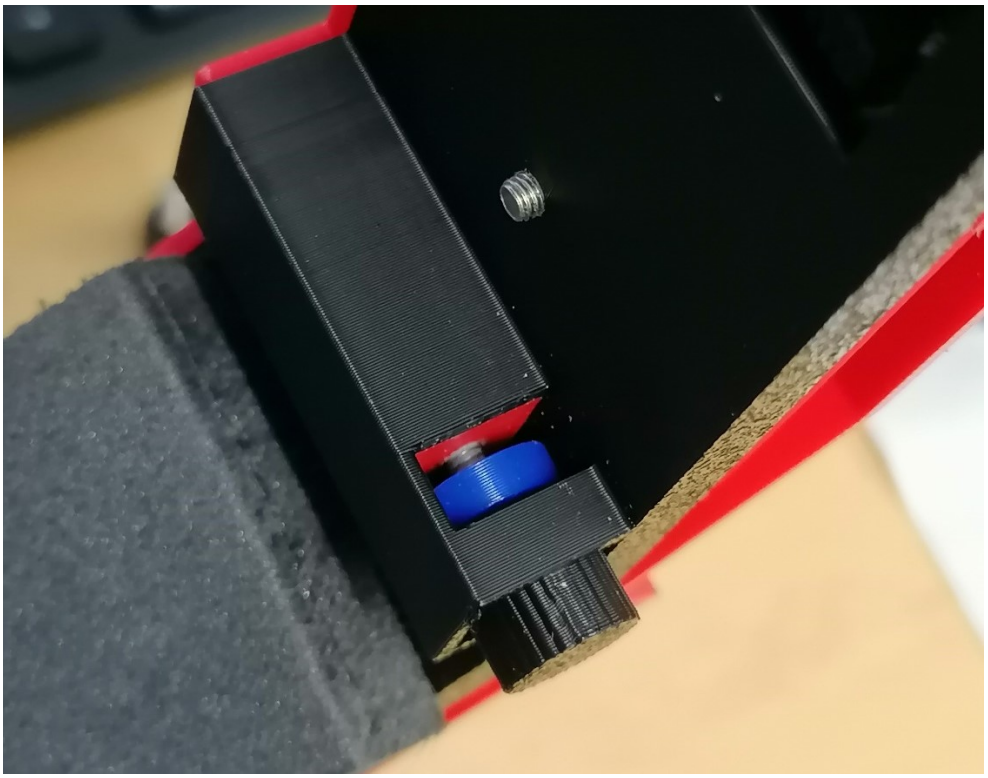

Image 3.4. Locking mechanism for the headband covers. A bolt is attached to the threaded nut (blue). Tightening or loosening the bolt nub (black) releases or locks the threaded headband cover (red).

4 Conclusion

OpenAR 2.1 does not differ all that much from OpenAR 2.0 except for the electronics used, and some new and modified 3D parts to fit them onto the OpenAR platform. It is, however, a bit more expensive as the thermal camera and FeatherS3 development board cost more. OpenAR 2.1 is a prime example of how easy it is to modify the OpenAR platform to suit a completely different function. OpenAR 2.0 version runs a simple demo program to showcase augmented reality, while 2.1 enhances the user's vision by enabling thermal vision. OpenAR 2.0 did most of the heavy lifting, as designing and building 2.1 version took a fraction of the time that 2.0 did.

And that is exactly the philosophy of OpenAR; we design, build, and then share the whole process with everyone for free. Hopefully reading this documentation will inspire you to build some cheap augmented reality glasses yourself! Remember to stay tuned for new and exciting OpenAR updates in the future.