



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

An investigation into a small footprint heat-source tracking solution

by
Joshua Asare

Supervisor: Gary Jones/Jenifer Prince Devaraj

Project submitted as part of the requirements for the
award of

BEng (Hons) Embedded Electronics Systems

April 2025

Contents

1	Abstract	4
2	List of Figures	5
3	Introduction.....	5
4	Camera technology research and selection	6
4.1	Visible spectrum cameras	6
4.2	Infrared bands	6
4.3	Capturing infrared scenes	7
4.3.1	Quantising light.....	7
4.3.2	Modification of a visible-light camera.....	7
4.3.3	Single wave band infrared sensors.....	8
5	Camera gimbal research and selection.....	12
5.1	Gimbal specification	12
5.2	Data transmission requirements	12
5.3	Data transfer problem	12
5.3.1	Slip rings	12
5.3.2	Wireless data transfer.....	13
5.3.3	WiFi-enabled processors.....	13
5.3.4	Decision to not add additional section	13
6	Infrared Sensor – FLIR Lepton 3.5.....	13
6.1	Dynamic range	13
6.2	Automatic gain control (AGC)	13
6.3	Spectral response	14
7	Microcontroller selection	14
8	Microcontroller programming	15
8.1	Software development environment	15
8.2	Hardware debugging tools	15
8.3	Software Design.....	16
8.3.1	Super-loop.....	16
8.3.2	Reading infrared sensor data.....	17
8.3.3	Version control.....	20
8.3.4	Test and control interface.....	22
8.3.5	Library usage	22
8.3.6	Verification of sensor functionality	22
8.4	Testing tracking stability on stationary and moving objects	23
8.4.1	Test assembly schematic and imagery	23
8.4.2	Test procedure.....	25

8.4.3	Plotting and analysing logged data	26
9	Power consumption.....	27
10	Cost	27
11	Conclusions.....	28
12	Future Improvements	28
13	Bibliography	29
1	Appendix A – pico_controller.c.....	32
2	Appendix B – lepton.c	37
3	Appendix D – servo.c	45
4	Appendix D – st7735.c	46
5	Appendix E – lepton.h	49
6	Appendix F – servo.h.....	53
7	Appendix G – st7735.h	54
8	Appendix H – plot_telem.py	55

1 Abstract

This project presents the development of a "Small Footprint Hot Source Tracking Solution," designed to detect and track hot objects emitting thermal radiation in various environments, such as search and rescue operations; surveillance; pet, captive wildlife and child monitoring, and industrial hazard monitoring. The system comprises of a compact sensor assembly integrated into a rotating mechanism which allows continuous tracking of heat sources through commands sent to a motor. The gimbal assembly was proposed to be stabilized by an inertial measurement unit (IMU), which was expected to facilitate smooth and accurate tracking by compensating for motion and vibrations.

Throughout the project, significant efforts were expended bringing other critical components of the system up function. Time was expended searching for low-cost solution to transfer data between a camera assembly that should be able to rotate more than 360 degrees in the horizontal plane, before concluding this was not possible. The project ended with a solution that was capable of tracking heat-emitting objects in the horizontal plane from $+45^{\circ}$ to -45° which gives the system a total scannable field of view of 132° when combined with the cameras field of view of 42° .

2 List of Figures

Figure 1: The same scene as recorded by a visible (L) and thermal (R) camera [1]	6
Figure 2: Atmospheric transmittance of electromagnetic wavelengths from 0-22 μm [2].....	6
Figure 3: “Typical atmospheric transmission for wavelengths between 1 and 15 μm ” [3]	7
Figure 4: A diagram of the filters inside a Canon 600D DSLR [4].....	7
Figure 5: Transmittance of a UV/IR cut filter [4].....	8
Figure 6: A comparison of a visible (left) and SWIR (right) image of a benign scene [6]	9
Figure 7: A fire spread across a mountain in visible (left) and SWIR (right) [5].....	9
Figure 8: A scene showing workers using a heat gun on a roof recoded in visible (centre) MWIR (left) and SWIR (right) [7].....	10
Figure 9: Pancake and through-bore type slip rings [13].....	12
Figure 10: Three different types of AGC in a simple scene of a mug in an environment [10]	14
Figure 11: The spectral response of the FLIR Lepton focal plane array	14
Figure 12: Pseudocode flow diagram for the simple serial command interface.....	16
Figure 13: Pseudocode flow diagram for searching for the pixel with greatest intensity across a frame	17
Figure 14: A screenshot from Digilent WaveForms [19] software showing the VSYNC pulse output by the FLIR lepton.....	18
Figure 15: The Raspberry Pi Pico reading out a frame (1/4 of the entire image) from the FLIR Lepton	18
Figure 16: I2C bus activity enabling VSYNC output from the FLIR Lepton	18
Figure 17: Packet Header Encoding and Example [10].....	19
Figure 18: Serial printout of the row number received in each of the packets	19
Figure 19: An example of a Git commit which highlights changed code lines	20
Figure 20: A graph of the Git commit history for the software written as a part of this project	21
Figure 21: The schematic of the test assembly	23
Figure 22: The test setup on a breadboard	24
Figure 23: Front and side view of the sensor mounted on a servo	24
Figure 24: Stability test setup	25
Figure 25: A mug filled with hot water	25
Figure 26: A scene containing the hot mug of water as observed by the Infrared Sensor. Dark pixels indicate a hotter, lighter indicate cooler.	25
Figure 27: The X and Y offsets in pixels of the hottest object in the frame, alongside the pan angle the sensor is orientated to look at	26
Figure 28: Example of data recorded over the serial interface	26
Figure 29: Servo unable to find a stable angle where the hottest pixel is in the centre 8 pixels	27
Figure 30: Idle power consumption	27
Figure 31: Active power consumption.....	27
Figure 32: Active power consumption without a screen attached	27
Figure 33: Project Gannt chart.....	28

3 Introduction

Tracking hot objects is of great interest to defence, security, automation and hazard monitoring. Infrared sensors may be mounted on moving platforms such as unmanned aerial vehicles (UAVs) to enhance search and rescue operations by giving operators enhanced situational awareness by highlighting regions of interest.

4 Camera technology research and selection

4.1 Visible spectrum cameras

Cameras that record light in the visible wavelengths are unable to specifically highlight objects that are hotter or cooler than their surroundings. Detecting hot objects relative to their surroundings using a camera that operates in the visible bands is not feasible unless they have other distinct features that can be used to identify them. Identifying using visible wavelengths people often requires machine learning algorithms to ingest large quantities of data during the process of training a model. These models then need to be deployed to hardware that often has a high cost associated with running them, sometimes Field Programmable Gate Arrays (FPGAs)[1].



Figure 1: The same scene as recorded by a visible (L) and thermal (R) camera [1]

4.2 Infrared bands

The infrared spectrum can be split into five distinct bands: Near-infrared, short-wavelength infrared, mid-wavelength infrared, long-wavelength infrared and far-infrared. The presence of ozone, water vapour and carbon dioxide [2] are responsible for absorbing various electromagnetic wavelengths, shown below in Figure 2.

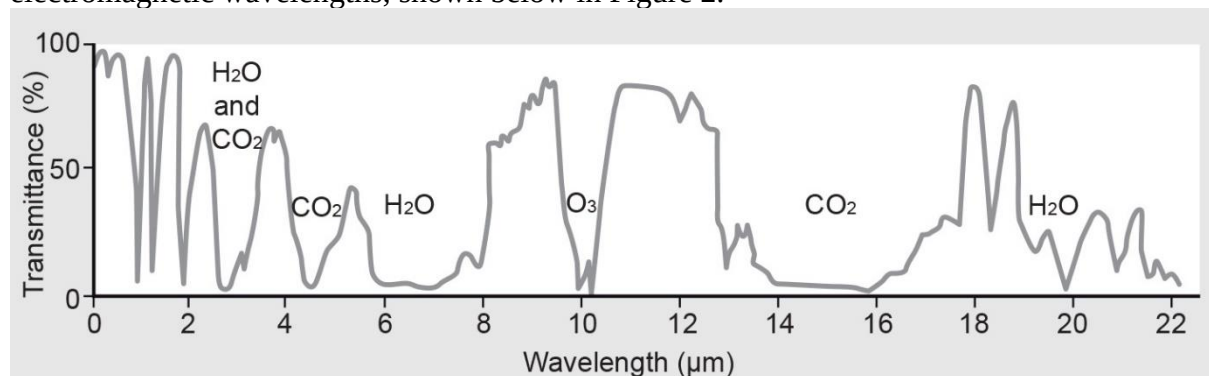


Figure 2: Atmospheric transmittance of electromagnetic wavelengths from 0-22 μm [2]

The three primary infrared bands, short-wavelength infrared, mid-wavelength infrared, long-wavelength infrared span wavelengths of 0.4-2.2 μm .

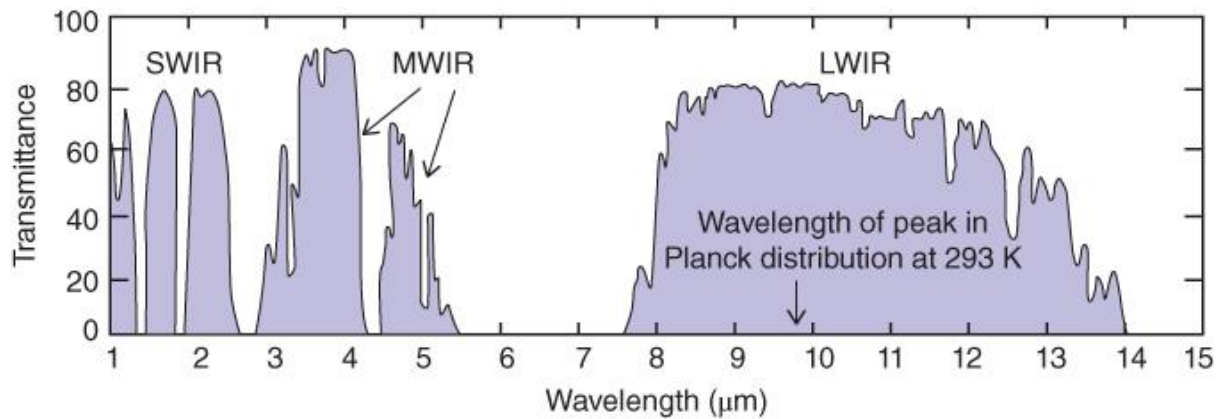


Figure 3: “Typical atmospheric transmission for wavelengths between 1 and 15 μm ” [3]

4.3 Capturing infrared scenes

4.3.1 Quantising light

Both Digital Single-Lens Reflex (DSLR) and Mirrorless cameras use a Complementary Metal-Oxide-Semiconductor (CMOS) based Integrated Circuit (IC) which contains an array with millions of red, green, and blue photodiodes that have their own amplifier on each row along with an ADC to quantise the incident light across their respective photodiodes.

4.3.2 Modification of a visible-light camera

A standard DSLR or mirrorless digital camera may undergo a conversion process to expand the wavelengths from the visible spectrum reaching the sensor in the camera body. This is achieved by through the removal of a cut filter, designed to absorb UV and or IR wavelengths preventing them from reaching the sensor. These filters are housed inside the main body of the camera forward of the sensor. An example of this assembly for a Canon 600D DSLR camera is shown below in Figure 4. An example of a UV/IR cut filters transmittance is shown below in Figure 5.

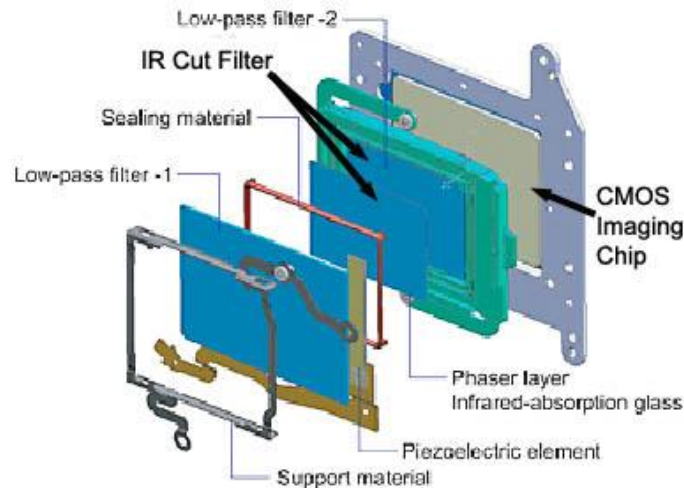


Figure 4: A diagram of the filters inside a Canon 600D DSLR [4]

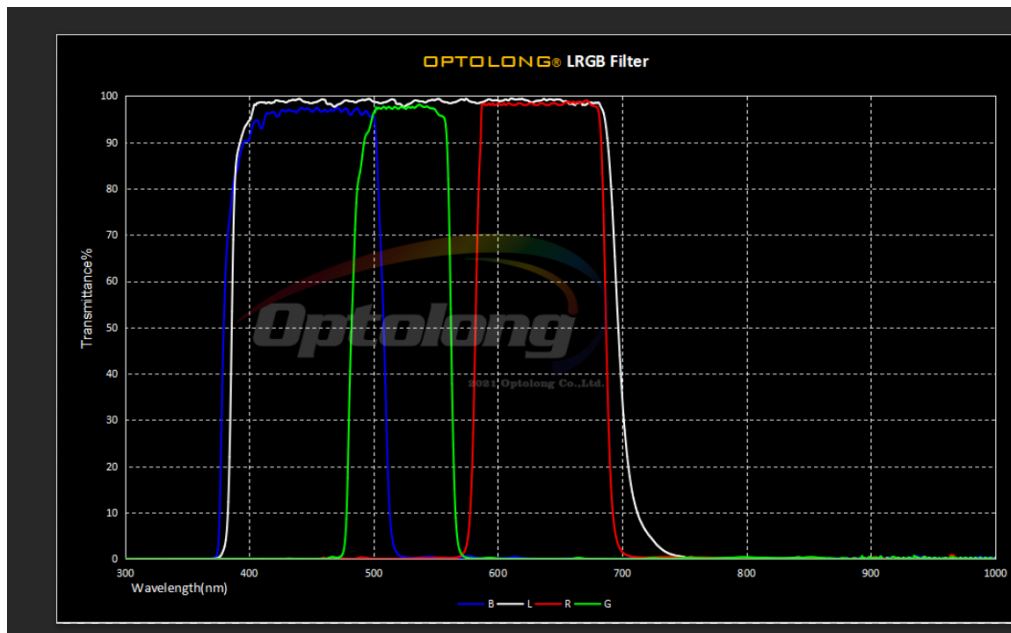


Figure 5: Transmittance of a UV/IR cut filter [4]

A significant challenge associated with utilizing DSLR cameras that provide a clean HDMI output for data acquisition is the necessity for effective data readout in this configuration. While clean HDMI output allows for high-quality video transmission without overlays, it would require this project to implement a system capable of reading and processing the HDMI stream. This introduces an additional layer of complexity, as the software must be designed to handle the HDMI protocol, which can be more demanding than interfacing with sensors that offer direct data output. Consequently, this approach incurs a processing overhead that would significantly increase the footprint and cost of the project. In contrast, sensors specifically designed for data acquisition often provide more straightforward interfaces, enabling direct data access without the need for extensive software processing. Therefore, while the use of clean HDMI output can yield high-quality visual data, the data processing requirements conflict with this project's requirements for a small footprint so this approach has not been selected.

4.3.3 Single wave band infrared sensors

In increasing wavelength, the infrared portion of the electromagnetic spectrum is divided into Short-Wave Infrared (SWIR), Mid-Wave Infrared (MWIR) and Long-Wave Infrared (LWIR). Each of these bands interacts differently with various materials and atmospheric conditions, exhibiting unique absorption and transmittance characteristics at specific wavelengths. As a result, sensors operating within these different bands may excel in certain scenarios while underperforming in others, depending on the specific environmental factors at play.

4.3.3.1 Short Wave Infrared

SWIR covers the 1-2.5 μm wavelengths of the infrared spectrum. Unlike visible cameras, SWIR is capable of imaging “through [small amounts of] rain, mist and fog” shown below in Figure 6 and smoke shown below in Figure 7. This is highly advantageous for firefighting as smoke may obscure the flame to a visible camera [5].

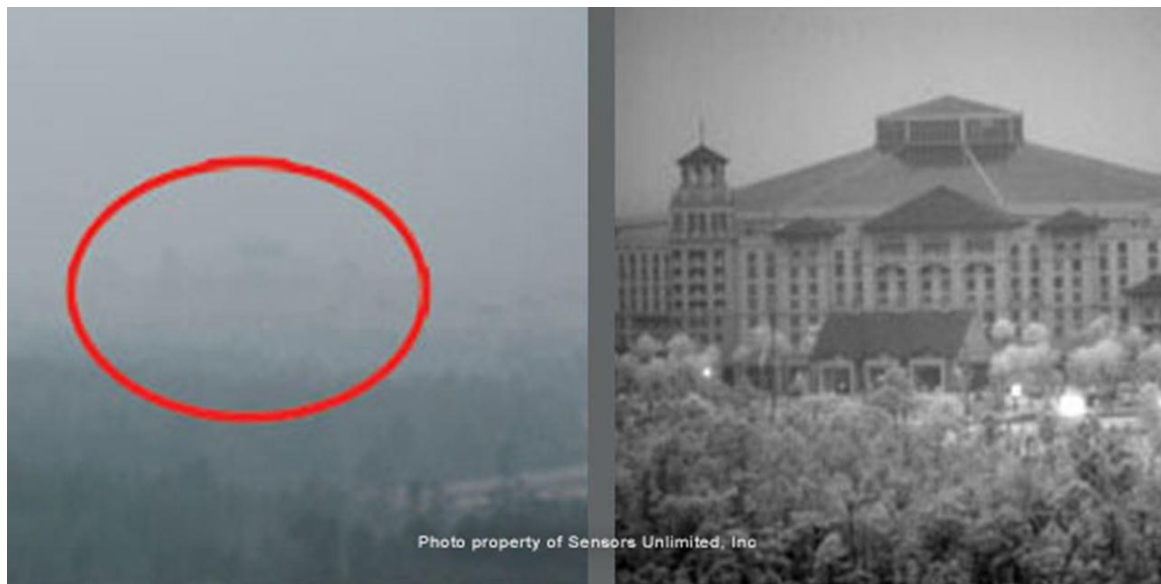


Figure 6: A comparison of a visible (left) and SWIR (right) image of a benign scene [6]



Figure 7: A fire spread across a mountain in visible (left) and SWIR (right) [5]

4.3.3.2 Mid-wave Infrared

MWIR covers the 3-5 μm wavelengths of the infrared spectrum. This band is particularly effective for detecting thermal radiation emitted by objects at moderate temperatures, making it suitable for military surveillance, industrial monitoring, and environmental studies. Figure 8 below demonstrates that MWIR sensors excel in detecting objects that are only slightly warmer than their surroundings compared to SWIR.



Figure 8: A scene showing workers using a heat gun on a roof recorded in visible (centre) MWIR (left) and SWIR (right) [7]

4.3.3.3 Long-wave Infrared

LWIR covers the 8-14 μm wavelengths of the infrared spectrum. LWIR is primarily used to detect heat emitted by solid objects, which is the focus of this project. A significant advantage of LWIR sensors is their ability to see through obscurants like smoke, dust, and fog, which can severely limit visibility for visible light cameras, particularly useful in firefighting, search and rescue, and cluttered environments with non-ideal and uncontrollable conditions. LWIR sensors are also widely used in industrial applications for monitoring equipment temperatures and detecting overheating components, which can prevent equipment failures and enhance safety. Satellite imagery using a fusion of LWIR, SWIR and visible cameras can provide enhanced “during-fire intelligence” [8]. LWIR in this context is capable of differentiating between hot regions of smoke which correspond to lower, warmer smoke, and colder clouds.

4.3.3.4 Comparison of Long Wave Infrared (LWIR) sensors

LWIR sensors with a total pixel count roughly covering different orders of magnitude of ~ 100 , $\sim 10,000$ and $\sim 1,000,000$ are shown below in Table 1. The FLIR Boson offers the highest specification with a greater resolution, frame rate and a slightly greater low-gain dynamic range compared to the FLIR Lepton and Panasonic AMG8833 sensors. However, the cost is 13 times greater than a FLIR lepton and this project does not require the increased resolution or framerate, which is why the FLIR Lepton has been selected as the sensor to take forwards.

Table 1: Comparison of infrared sensors

	FLIR Boson [9]	FLIR Lepton [10]	AMG8833 [11]
Resolution	320 x 256, 640 x 512	160 x 120	8 x 8
Frame rate	60 Hz	8.7 Hz	10 Hz
Interface	MIPI, SPI	MIPI, SPI	I2C
Field of View (Horizontal, Vertical)	Various, lens dependant	57°, 71°	60°, 60°
Wavelengths	Long Wave Infrared (8-14 μm)		Not specified

Temperature range (high gain, low gain)	140°C (dynamic), 500°C (dynamic)	-10 °C to 140 °C, -10 °C to 450 °C	0 °C to 80 °C,
Cost	£1,609.43	£123.72	£39.84

5 Camera gimbal research and selection

5.1 Gimbal specification

5.2 Data transmission requirements

Data must be able to move from the sensor which should be mounted in a rotating 2 axis gimbal assembly, to a fixed base. For the sensor to be able to rotate through more than 360°, standard point-to-point connections cannot be used as they are prone to twisting and breaking.

5.3 Data transfer problem

5.3.1 Slip rings

Slip rings are electromechanical devices that enable the transmission of electrical power and signals from a stationary to a rotating structure. They consist of a circular conductive ring and a set of brushes that continuously maintain contact with the ring as it rotates [12] which allows for a continuous electrical connection. Slip rings are commonly used in wind turbines, rotary joints, and industrial machinery, where rotating components require power and or data transfer. They facilitate the operation of devices such as motors, sensors, and cameras mounted on gimbals, allowing for uninterrupted functionality as they continue to rotate past 360 degrees, where a typical setup using cables would start to snag, twist, separate or possibly break from over-rotation. Another advantage of slip rings is their ability to provide this reliable connection in dynamic environments, reducing the risk of wear and tear associated with traditional connectors. In situations where a system does not need to frequently rotate, a connector could be un-plugged and plugged back in, however this is not an approach this project can take. There are some drawbacks related to the lifespan of friction-based slip rings, including potential friction and wear on the brushes, which can lead to maintenance issues and reduced lifespan.

Slip rings come in various designs, including conductive, optical, and hybrid types, depending on the specific requirements of the application, two examples of conductive slip rings, namely pancake and through bore, are shown below in Figure 9.

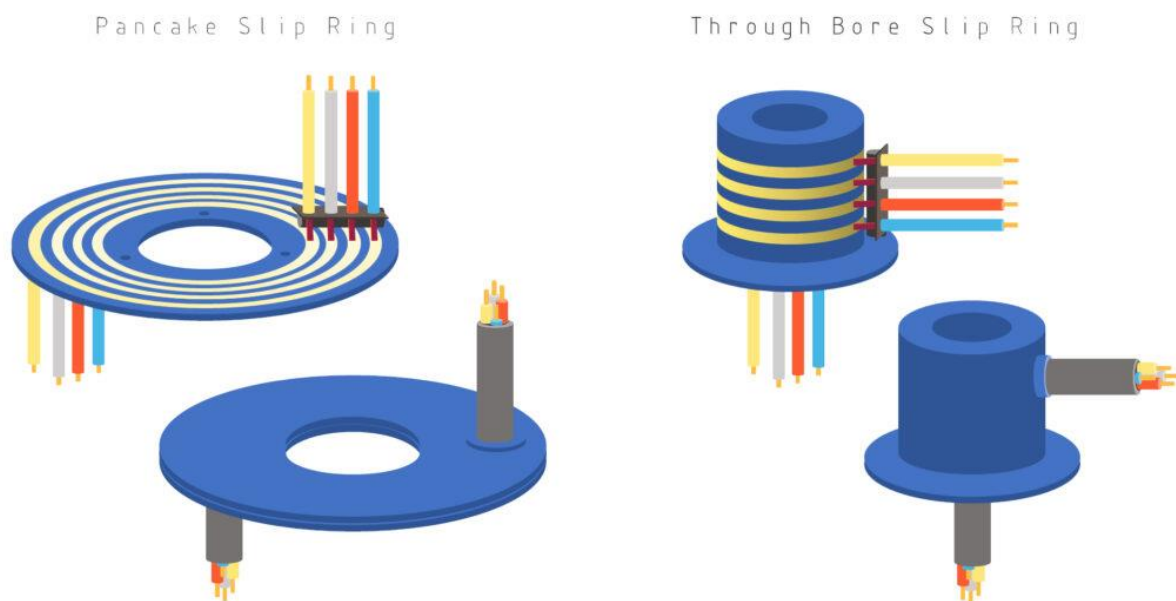


Figure 9: Pancake and through-bore type slip rings [13]

5.3.2 Wireless data transfer

More recent than the invention of slip rings, battery power and wireless data transfer can be used to exchange information across continually rotating boundaries. This has potential to prolong the life of a system and eliminate a point of failure as there are fewer physical boundaries that rub against one another. A disadvantage of this is that the sensor assembly must be charged and powered in addition to the gimbal, separating the system into two parts that must operate independently. As there is still an electrical connection between the two subsystems, it is possible to transfer power across this interface.

5.3.3 WiFi-enabled processors

The integration of WiFi-enabled processors, such as the Espressif Systems ESP32 and its predecessor ESP8266, has become increasingly popular in Internet of Things (IoT) applications. These processors are equipped with built-in WiFi and Bluetooth stacks which make them ideal for low footprint wireless communication.

5.3.4 Decision to not add additional section

Integration of any of the components above, wireless data transfer or slip ring data transfer, would add additional tasks to the Gantt chart which would delay on the delivery of other features this project requires, which have already been delayed due to this research.

6 Infrared Sensor – FLIR Lepton 3.5

This following section will refer to the output of the 160 x 120 array as an “image”.

6.1 Dynamic range

The FLIR Lepton is stated to have a scene dynamic range of -10°C to +140 °C [10]. Dynamic range [14] is important for determining the suitability of this infrared sensor as it defines the range over which the sensor can present information. A low dynamic range means that a given sensor is not able to simultaneously capture a wide range of intensities, which is acceptable when the desired intensities are close to one another but becomes problematic if detail is required as these intensities move further apart.

For the application of monitoring and tracking the position of a human, whose body temperature is normally “around 37C” [15], the expected temperature of a human body lies within the dynamic range of the FLIR Lepton.

6.2 Automatic gain control (AGC)

It is not always desired to have a high dynamic range across the image where the image does not have a wide range of intensities across pixels. The temperature of the environment is estimated to have varied from 11-17°C throughout the duration of this project. This estimation is based on thermostat and ambient outside temperature.

Automatic gain control allows for the minimum and maximum intensities to be stretched to increase contrast in the displayed image. The engineering datasheet [10] provides an example of the options available, for this project option c has been chosen by enabling AGC through a CCI command.

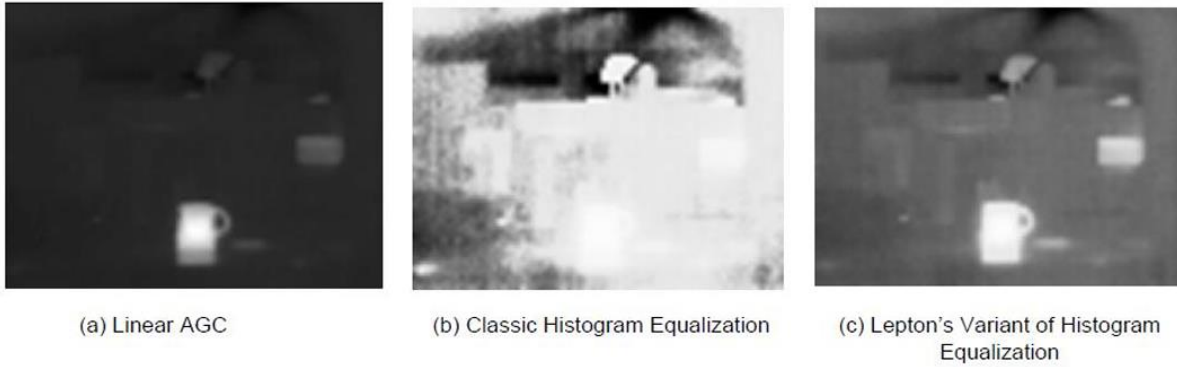


Figure 10: Three different types of AGC in a simple scene of a mug in an environment [10]

Throughout this project, only objects well below 100°C have been examined so the AGC inside the FLIR Lepton has not been set, instead the RAW14-formatted [10] pixels have been used by themselves to search for hot objects. Objects hotter than this fall outside of the risks that were assessed during the project proposal and completion of the ethics form, so they have not been explored for this project.

6.3 Spectral response

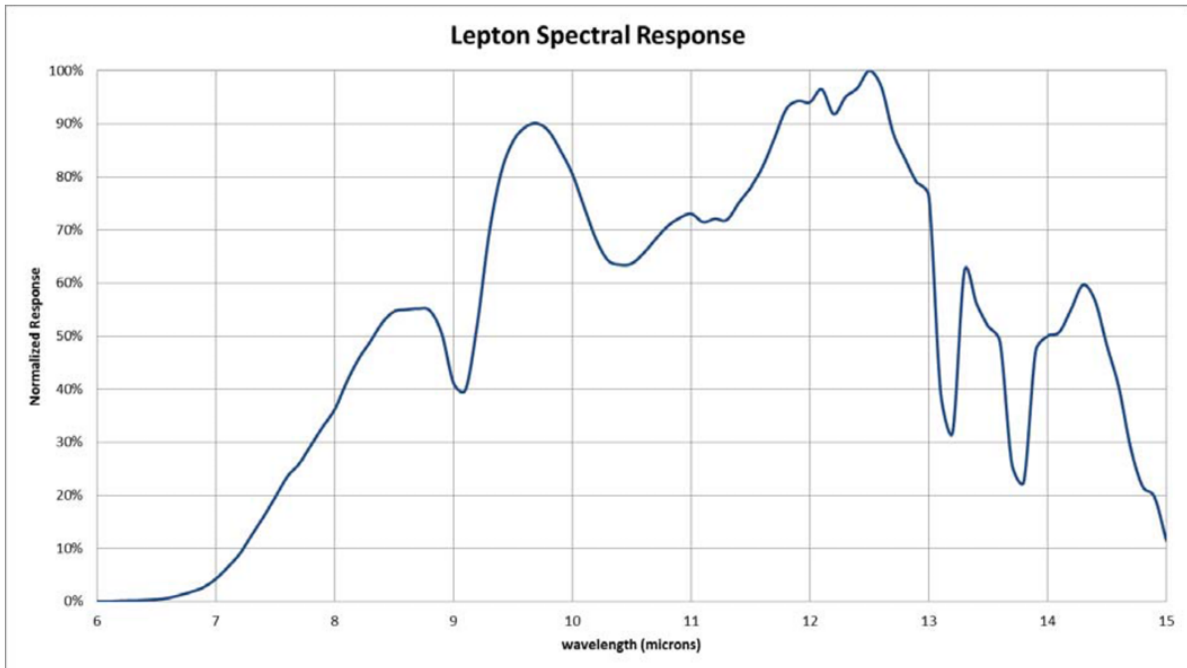


Figure 11: The spectral response of the FLIR Lepton focal plane array

7 Microcontroller selection

The microcontroller should include at least one General Purpose Input/Output (GPIO) pin dedicated to receiving the Vertical Synchronization (VSYNC) signal from the FLIR Lepton. This is essential for synchronizing the timing of the data capture on the microcontroller with the output and availability of data from the camera. Additionally, the microcontroller must feature two separate Serial Peripheral Interface (SPI) interfaces. This is because the FLIR Lepton requires both a dedicated SPI and I2C interface [10]. While one of these SPI interfaces will be utilized to communicate with the FLIR Lepton camera, the second interface will be used to interface with an LCD display, which is crucial for visualising the output of the infrared sensor. To accommodate future improvements and telemetry output to log data to

analyse, the microcontroller should also include a Universal Asynchronous Receiver-Transmitter (UART) channel. The design should incorporate a timer-based mechanism for periodically sending data to a host that logs or makes use of the telemetry data. This approach enables error printing and potentially diagnostics which would enable the system to correct for errors that build up over time. An example of this is periodically performing a Flat Field Correction (FFC).

Table 2: Comparison of Microcontrollers: STM32F103, Raspberry Pi Pico, and ESP32

	STM32F103	Raspberry Pi Pico	ESP32
Core	ARM Cortex-M3	ARM Cortex-M0+	Dual-core Xtensa 32-bit
Clock Speed	Up to 72 MHz	Up to 133 MHz	Up to 240 MHz
Flash Memory	64 KB to 128 KB	2 MB	4 MB (varies by module)
RAM	20 KB to 32 KB	264 KB	520 KB
GPIO Count	Up to 37	26	34
ADC Channels	10-bit, up to 10 channels	3 (12-bit)	12 (12-bit)
DAC Channels	None	None	2 (8-bit)
SPI Interfaces	3	2	4
I2C Interfaces	2	2	2
UART Interfaces	3	2	3
USB	No	Yes (Device only)	Yes (Device and Host)
Wi-Fi	No	No	Yes
Bluetooth	No	No	Yes
Power Supply	2.0V to 3.6V	1.8V to 5.5V	2.2V to 3.6V
Development Environment	STM32CubeIDE, Keil, etc.	MicroPython, C/C++	Arduino, ESP-IDF, MicroPython

The Raspberry Pi Pico was chosen for several compelling reasons that align with the project's requirements and objectives. It is a relatively inexpensive microcontroller which cost £3.60. A second Raspberry Pi Pico was purchased to run alongside

8 Microcontroller programming

8.1 Software development environment

The integrated development environment chosen to development of firmware for the Raspberry Pi Pico is Visual Studio Code [16]. The Raspberry Pi Foundation endorses and provides a software extension for Visual Studio Code to enable development of C/C++ programs for the Raspberry Pi Pico.

8.2 Hardware debugging tools

A second Raspberry Pi Pico was flashed with a program provided by Raspberry Pi, debugprobe [17], to program and debug the primary device. The debugger uses Serial Wire

Debug, a debugging protocol developed by ARM [18] for microcontrollers that implement ARM microcontroller IP.

8.3 Software Design

8.3.1 Super-loop

The program runs a main loop which is responsible for writing out to the LCD display and the UART-over-USB interface back to the laptop monitoring the telemetry output from the microcontroller about the state of the system.

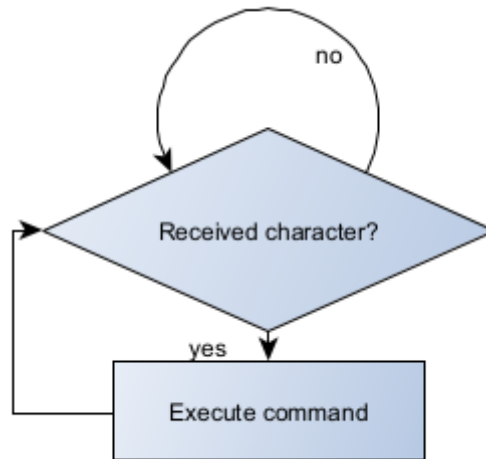


Figure 12: Pseudocode flow diagram for the simple serial command interface

This super-loop performs a search across the image frame for the pixel with the greatest intensity to designate it as the desired location in the frame for tracking. The pseudocode for this loop is shown below in Figure 13, with the implementation in c available in Appendix A – pico_controller.c.

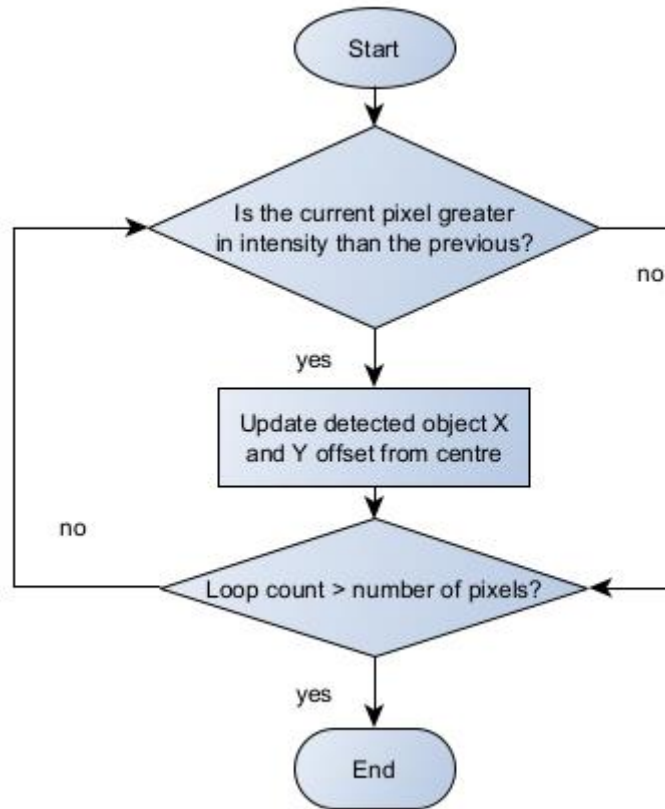


Figure 13: Pseudocode flow diagram for searching for the pixel with greatest intensity across a frame

8.3.2 Reading infrared sensor data

The FLIR Lepton 3.5 captures raw data across the infrared array of the scene internally and applies pre-processing before sending it out over SPI in a 14-bit format. The FLIR Lepton's SPI implementation is slave only with no input from the master device and the recommended clock speed is ~20 MHz.

8.3.2.1 VoSPI

Section "4.2.2.3 VoSPI Stream" and "4.2.3 VoSPI Protocol – Lepton FS1, 3.1R, 3.5, and UWFOV" in the FLIR Lepton engineering datasheet [10] describes the format of VoSPI data that is sent to the host. VoSPI is a specific implementation of SPI mode 3 which does not use the MOSI line. This section describes that due to export restrictions, instead of sending out a unique frame at 27 Hz, they are instead sent out at below 9 Hz with two repeats sent out (This project measured this to be 8.7 Hz) to comply with these export restrictions.

Despite updated frames being transmitted at 9 Hz, the host on the SPI bus must still read out the repeated frames otherwise the FLIR Lepton will decide that it has lost synchronisation with the host and start transmitting discard packets, which contain a header of 0x0FF0.

The total amount of data read out for a single frame is 3x greater than required due to the export restrictions described in the section above. While a complete image frame is made up of 19,200 words / 38,400 bytes, (160 x 120, 14-bit pixels), a total of 57,600 words / 115,200 bytes must be read out. This protocol proved difficult to use as the processor spends a significant amount of time reading out data over SPI. This made up for approximately 50% of the processing.

8.3.2.2 VSYNC

The VSYNC interrupt pin is critical for synchronising the order that rows of pixels are read out from the infrared sensor. This output from the sensor is disabled by default and must be

enabled using the OEM_GPIO_MODE_SELECT command. After power on, the command is issued with a value of 5 which enables the pulse shown below in Figure 14. Figure 16 shows the I2C bus activity that issues the command and checks to make sure the VSYNC output has been enabled by reading back the register that it had written to. The following functions responsible for this are available in Appendix B – lepton.c:

- ``cci_set_OEM_GPIO_MODE_SELECT``
- ``cci_get_OEM_GPIO_MODE_SELECT``

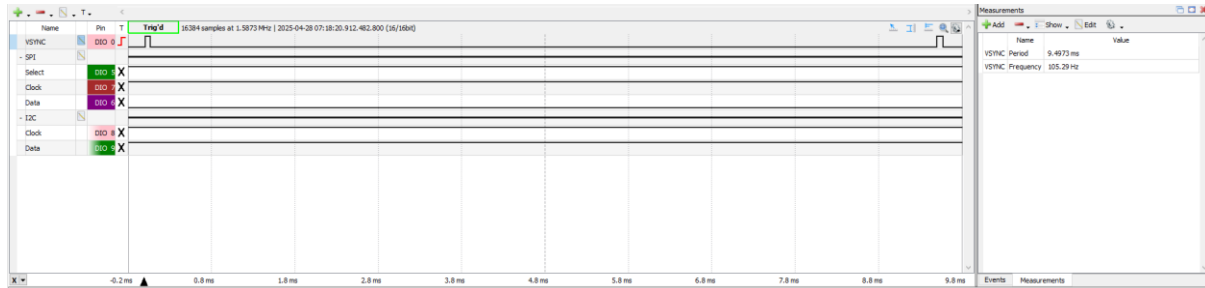


Figure 14: A screenshot from Digilent WaveForms [19] software showing the VSYNC pulse output by the FLIR lepton

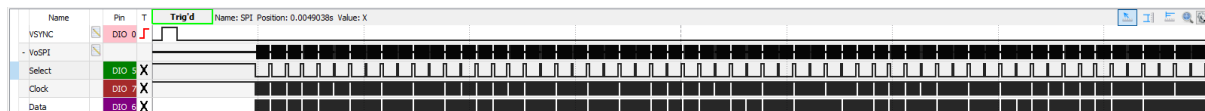


Figure 15: The Raspberry Pi Pico reading out a frame (1/4 of the entire image) from the FLIR Lepton

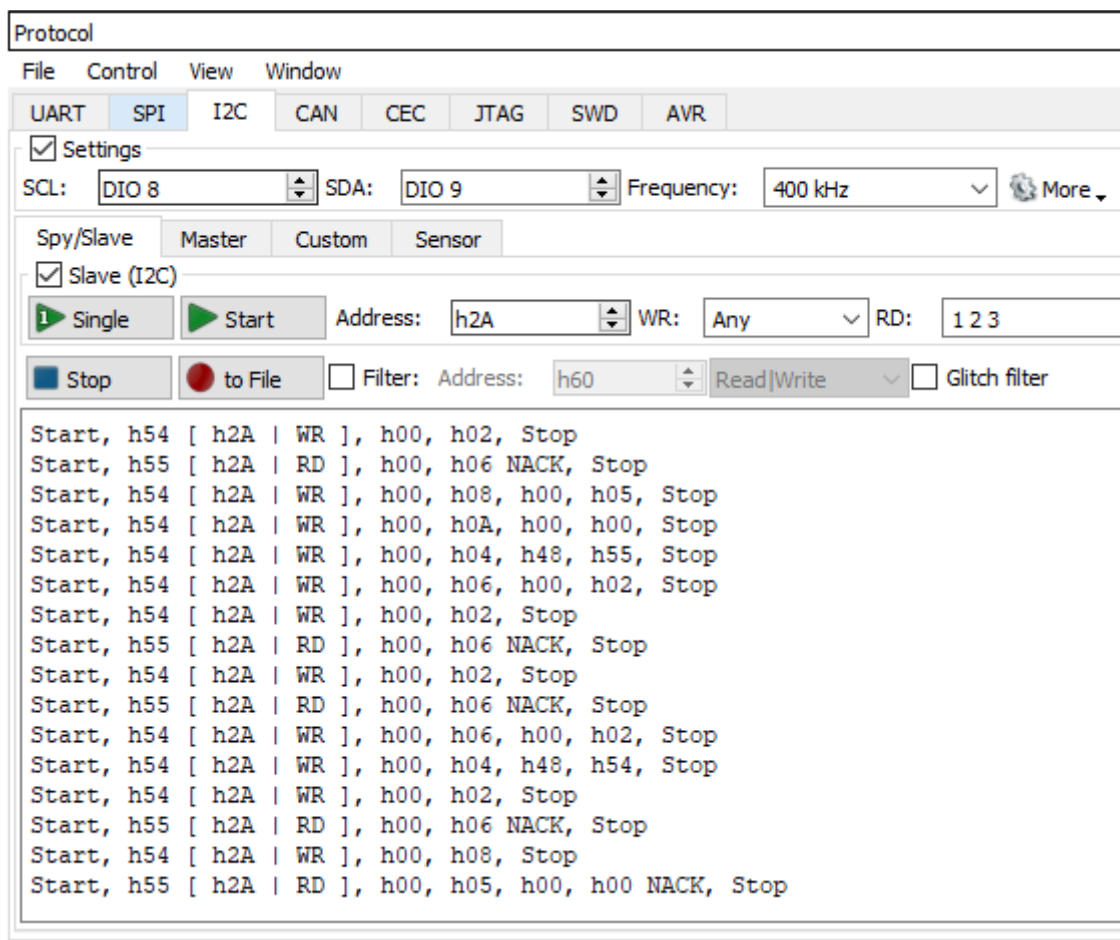
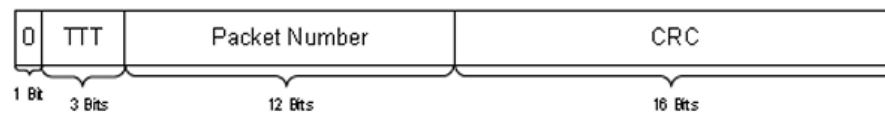


Figure 16: I2C bus activity enabling VSYNC output from the FLIR Lepton

8.3.2.3 Out of order row readout

VoSPI Packets, which are bursts of data containing the values of 80 pixels in a row, that were received from the FLIR lepton appeared to be out of order as the first received row was consistently 52 across multiple tests. The FAQ [20] and engineering datasheet [10] provided by FLIR suggested using the CCI interface to issue the following command, to add a delay to the output of the VSYNC pin adjusting the delay: OEM_GPIO_VSYNC_PHASE_DELAY. At the maximum delay of +3 which corresponds to + 1.5 ms, the only effect this had was the starting row that was read out became 54 instead, meaning the data was still being read out too fast. The negative and positive delays were tested but the frames still did not read out starting at 0. Attempts to “Resync”, also described in the engineering datasheet and FAQ, were also unsuccessful.

To work around this, logic was implemented to examine the packet header (described below in Figure 17), which contains the packet ID, which encodes the row number, to place the rows into the correct location in memory.



(a) Generic Encoding of the packet header

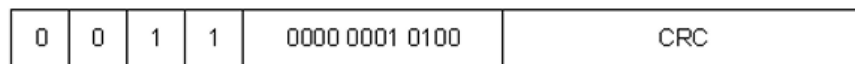


Figure 17: Packet Header Encoding and Example [10]

```

ID: 52 ID: 53 ID: 54 ID: 55
ID: 56 ID: 57 ID: 58 ID: 59
ID:  0 ID:  1 ID:  2 ID:  3
ID:  4 ID:  5 ID:  6 ID:  7
ID:  8 ID:  9 ID: 10 ID: 11
ID: 12 ID: 13 ID: 14 ID: 15
ID: 16 ID: 17 ID: 18 ID: 19
ID: 20 ID: 21 ID: 22 ID: 23
ID: 24 ID: 25 ID: 26 ID: 27
ID: 28 ID: 29 ID: 30 ID: 31
ID: 32 ID: 33 ID: 34 ID: 35
ID: 36 ID: 37 ID: 38 ID: 39
ID: 40 ID: 41 ID: 42 ID: 43
ID: 44 ID: 45 ID: 46 ID: 47
ID: 48 ID: 49 ID: 50 ID: 51

```

Figure 18: Serial printout of the row number received in each of the packets

8.3.3 Version control

Version control is a critical aspect of software development, enabling teams to manage trackable and auditable changes to source code over time. In this project, Git [21] was employed as the version control system, providing a robust framework for tracking modifications and maintaining a history of the project's evolution. Git's distributed nature allows multiple developers to work on the same project simultaneously without interfering with each other's progress, which for future work with more than a single collaborator would be essential. The use of Git in this project aligns with the principles outlined in the ISO/IEC 27001 standard [22], which emphasizes the importance of version control systems. Figure 20 below shows a graph of the commit history to this project. The use of git was invaluable as several bugs were introduced during development which were solved with comparisons to previous, known working versions.

```
264 void receive_frame() {
265     voispi_packet_t packets[120];
266     memset(&packets, 0, sizeof(packets));
267
268     for (int i = 0; i < 120; i++) { // Lepton 3.5 has 120 packets per f
269         uint8_t rx[164] = {0};
270         spi_read_blocking(FLIR_SPI_PORT, 0, rx, sizeof(rx));
271         memcpy(&packets[i], rx, sizeof(voispi_packet_t));
272         printf("Received packet %d\n", packets[i].id);
273         // memcpy(&frame_buffer[packets[i].id * 80], &packets[i].payload[
274     }
275
269 void receive_frame() {
270     voispi_packet_t packets[120];
271     memset(&packets, 0, sizeof(packets));
272+    gpio_put(FLIR_SPI_CS, 0);
273     for (int i = 0; i < 120; i++) { // Lepton 3.5 has 120 packets per f
274         uint8_t rx[164] = {0};
275         spi_read_blocking(FLIR_SPI_PORT, 0, rx, sizeof(rx));
276         memcpy(&packets[i], rx, sizeof(voispi_packet_t));
277         printf("Received packet %d\n", packets[i].id);
278         // memcpy(&frame_buffer[packets[i].id * 80], &packets[i].payload[
279     }
280+    gpio_put(FLIR_SPI_CS, 1);
```

Figure 19: An example of a Git commit which highlights changed code lines

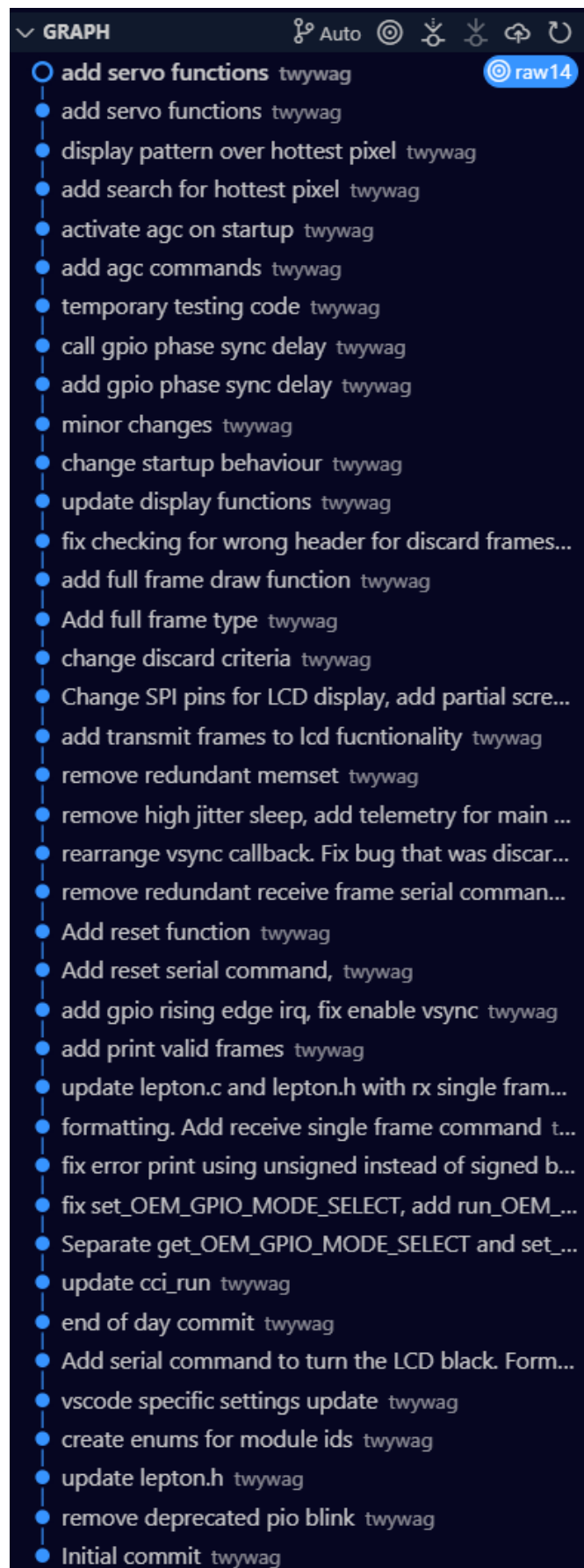


Figure 20: A graph of the Git commit history for the software written as a part of this project

8.3.4 Test and control interface

The microcontroller program implements a simple serial command interface within the main loop of the program checking if data is available to read from the UART interface. When a character is received, it echoes back the character and executes a series of commands or a particular function. These are primarily used to test the functionality of the individual components. These commands include stopping the program and resetting the sensor, filling the LCD with a specific colour, drawing patterns or lines on the LCD, running a servo test, querying the sensor status, running a flat-field correction, getting or setting GPIO modes, scanning the I2C bus, and retrieving various sensor information. This interactive command system allows the user to control and test different hardware features of the device in real time via serial input. In a production device, this test and control interface would be disabled, and the user would only have access to a telemetry output, described towards the end of this chapter.

8.3.5 Library usage

The main libraries used for this project are the hardware abstraction layers from Raspberry Pi to interact with their SDK to use low level functionality of the RP2040 microcontroller without directly interacting with the registers that control their functionality. These allowed for the configuration of the system clocks, UART, GPIO, SPI and I2C cores.

The program also includes custom libraries for the FLIR Lepton's Camera Control Interface (CCI) and Video over Serial Peripheral Interface (VoSPI), in addition to lightweight libraries to interface with the ST7735 LCD display and SG90 servo motors.

Table 3: A list of libraries used in the project

Library	Description
pico/stdlib.h	Standard Pico SDK functions (delays, sleep, etc.)
hardware/gpio.h	GPIO pin initialization and control
hardware/spi.h	SPI communication
hardware/i2c.h	I2C communication
stdio.h	Standard input/output functions (e.g., printf)
string.h	Memory and string manipulation (e.g., memset, memcpy)
lepton.h	Project-specific header for FLIR Lepton sensor functions
servo.h	Project-specific header for servo control
st7735.h	Project-specific header for ST7735 LCD display functions

8.3.6 Verification of sensor functionality

The FLIR Lepton sensor presented numerous issues when attempting communication in line with documentation provided directly from FLIR. A web-hosted FAQ [23] highlighted there is a minimum wait time after power on of 700 ms from releasing reset at power-up, but increasing this to a total of 1.6 s was found to be required to get no errors from the sensor's status register. The first of these issues related to the module not responding to the SYS_RUN_FFC command. This command is responsible for triggering the correction of fixed pattern noise, which infrared and thermal imaging sensors are subject to. Sample variations due to manufacturing tolerances, and non-uniformity [10] in material properties are two sources of fixed pattern noise, however these may be corrected before a sensor module is delivered to a customer through measurement and calibration processes.

8.4 Testing tracking stability on stationary and moving objects

8.4.1 Test assembly schematic and imagery

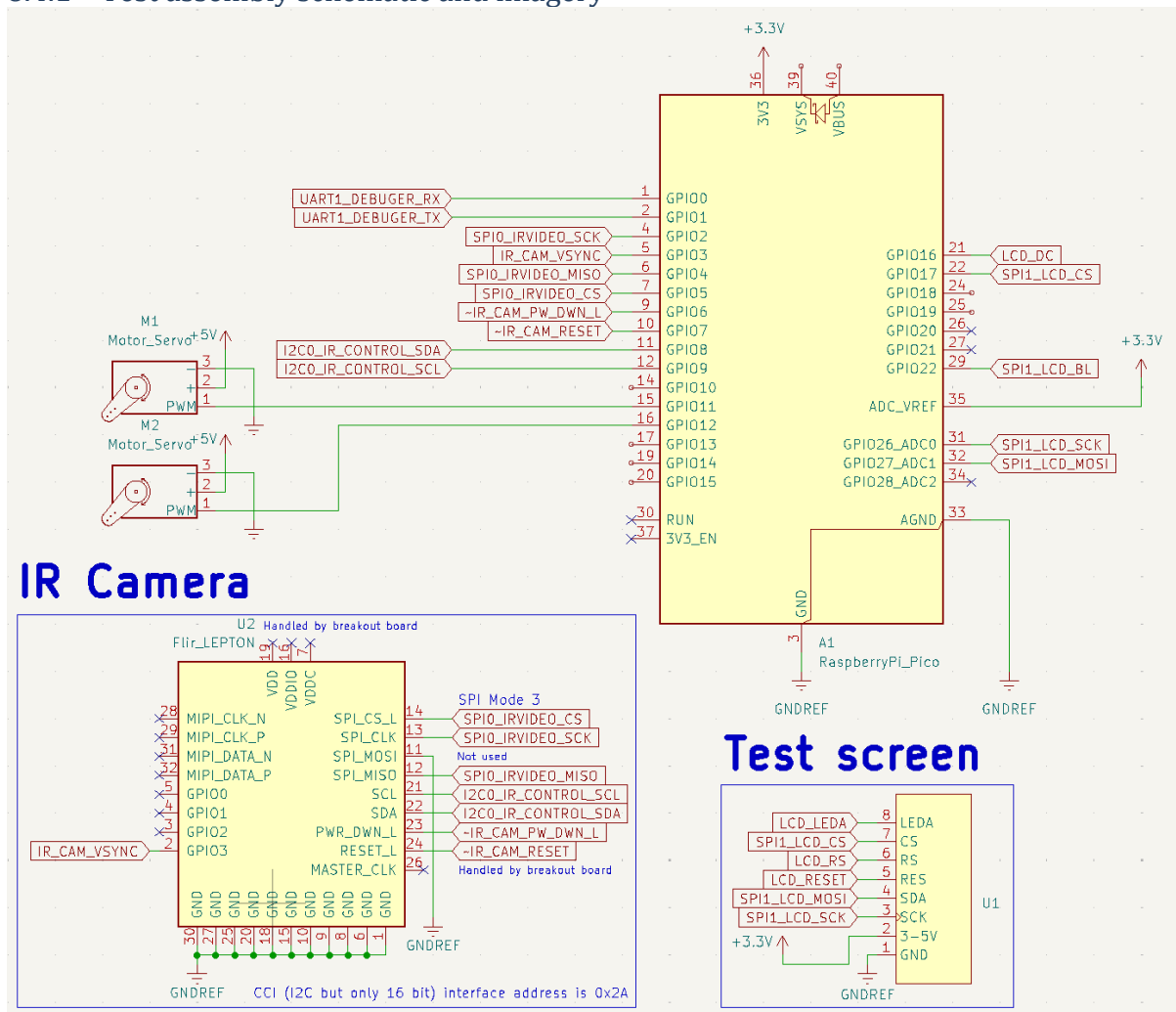


Figure 21: The schematic of the test assembly

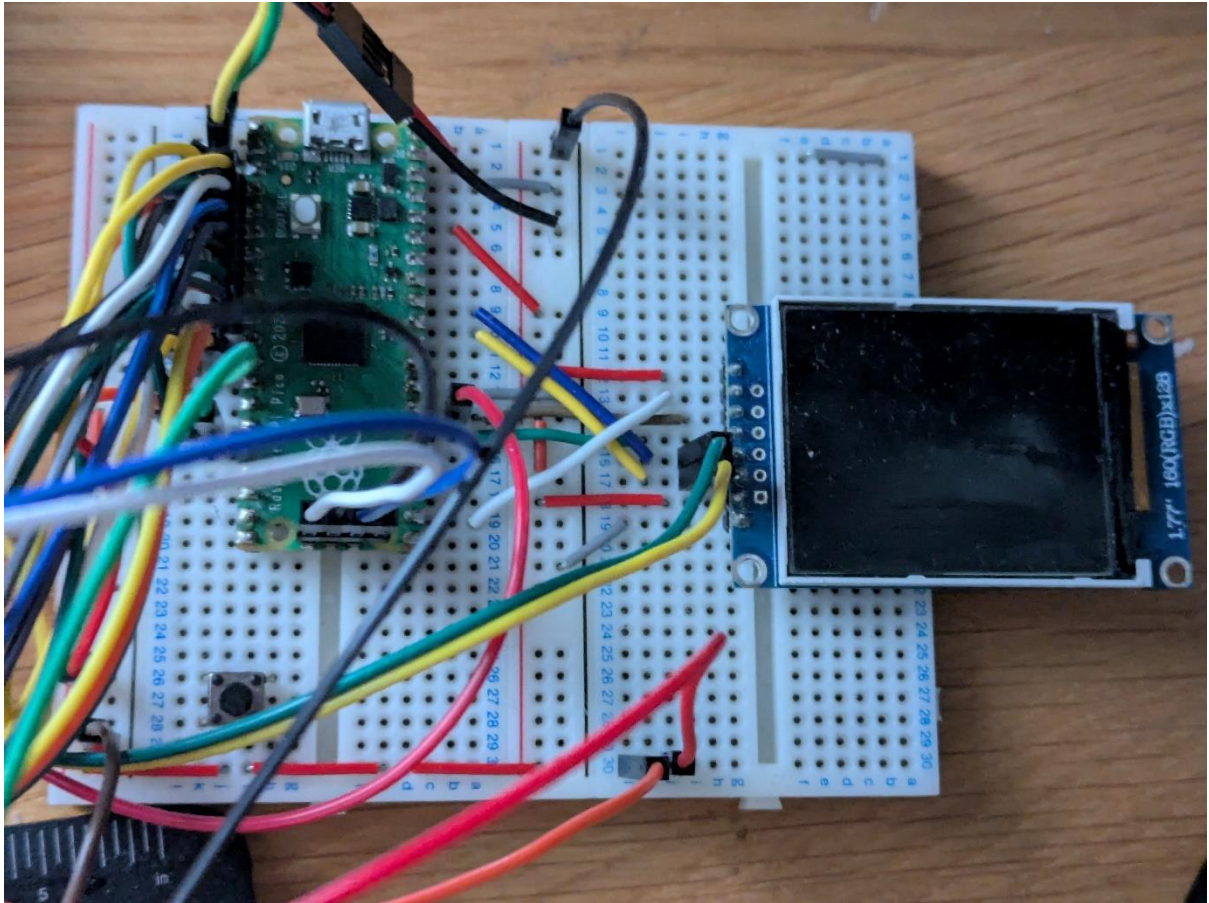


Figure 22: The test setup on a breadboard

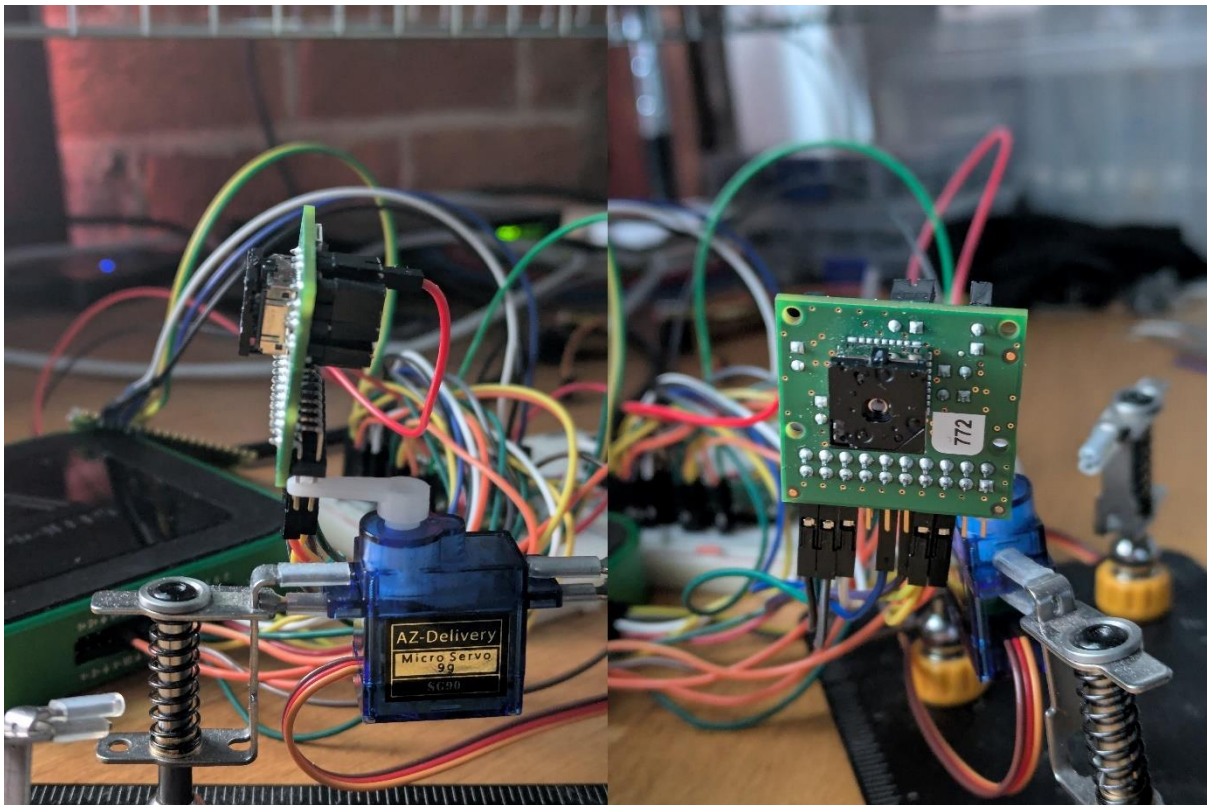


Figure 23: Front and side view of the sensor mounted on a servo

8.4.2 Test procedure

A test of the stability of the system when using different step sizes to move the servo by 1, 2, 3 and 4 degrees at a time was performed to ensure that when the brightest pixels were in the centre of the frame, the servos would not add a lot of jitters to the system. This test was performed by introducing a single hot object into the scene, in this case a mug filled with freshly boiled hot water. Then the source was moved from left to right while data was being logged. This simple test setup is shown below in Figure 24.

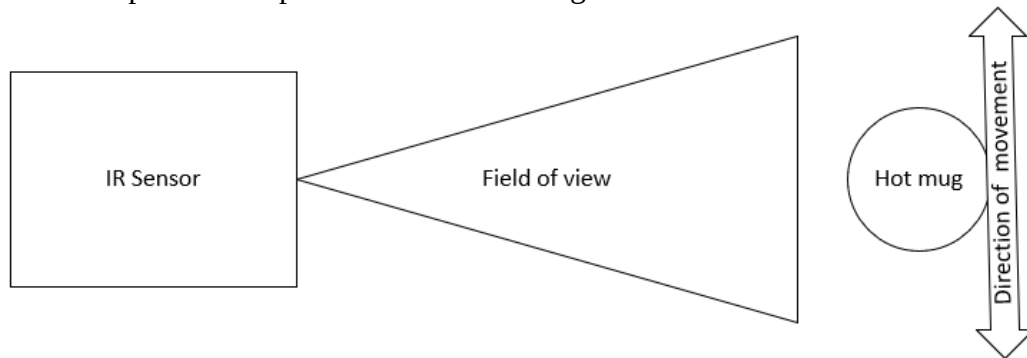


Figure 24: Stability test setup



Figure 25: A mug filled with hot water



Figure 26: A scene containing the hot mug of water as observed by the Infrared Sensor. Dark pixels indicate a hotter, lighter indicate cooler.

8.4.3 Plotting and analysing logged data

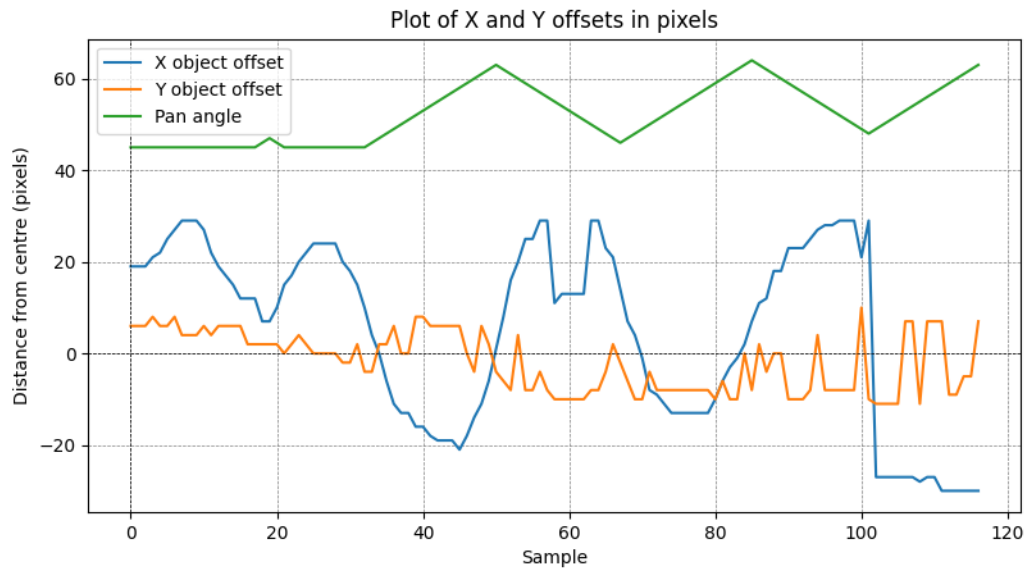


Figure 27: The X and Y offsets in pixels of the hottest object in the frame, alongside the pan angle the sensor is orientated to look at

Figure 27 above shows plots of three fields of data logged using the serial interface of the Raspberry Pi Pico over USB. The application Tera Term [24] was used as it has a logging feature that records data displayed in the terminal to a file. The file used to generate the plot displayed above is detailed in Appendix H – plot_telem.py.

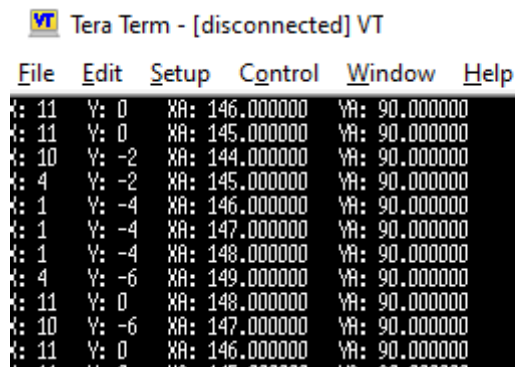


Figure 28: Example of data recorded over the serial interface

The gradient of green line in plotted in Figure 27 is either 0, positive 1 or negative 1 as this is the step size for the servo that was chosen after testing with larger step sizes. Larger step sizes caused the hottest pixel to continuously move out of the central 8x8 (+/- 4) pixels every time a step was performed. This resulted in the plot shown below in Figure 29 which appears to show a constant oscillation despite the hot object being stationary when using a step size of 2.

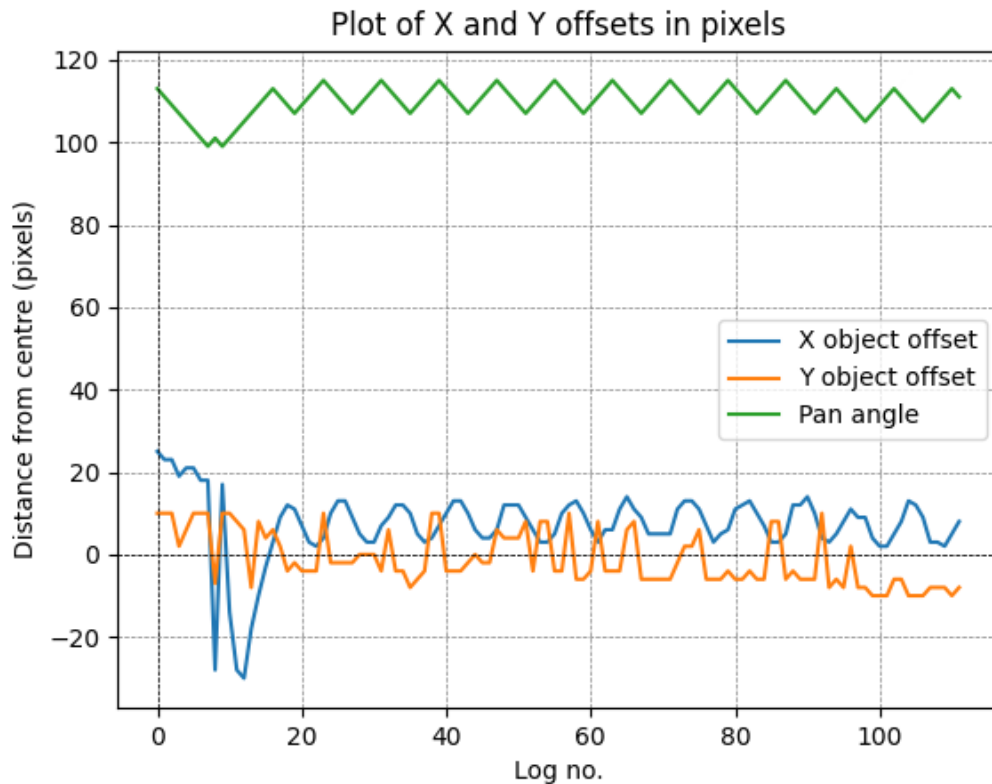


Figure 29: Servo unable to find a stable angle where the hottest pixel is in the centre 8 pixels

9 Power consumption

Figure 30, Figure 31, and Figure 32 show power consumption of the USB power supply powering the logic analyser that provided a +ve and -ve supply rail for the microcontroller and servo. Unfortunately, the power consumption of the microcontroller, infrared sensor and display were not tested on their own, so only rough measurements are shown below. It is suggested that this is thoroughly analysed in the future.

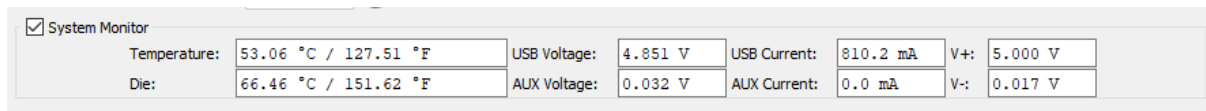


Figure 30: Idle power consumption

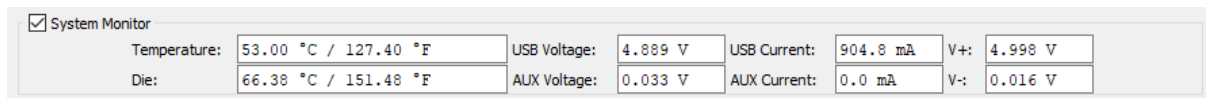


Figure 31: Active power consumption

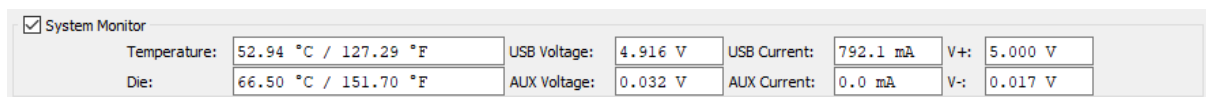


Figure 32: Active power consumption without a screen attached

10 Cost

The total cost for purchased components this project used was £151.29. After future improvements are applied and if this tracking solution were to be sent into production, the

cost of the display would be reduced to 0, but there would be costs associated with the integration of an inertial measurement unit and 3D printed enclosure. These low costs still satisfy the requirement for a small footprint.

Table 4: Cost of components purchased for this project

Component	Quantity	Cost
Raspberry Pi Pico	2	£3.80 [25]
Servo Motor	2	£5.99 [26]
FLIR Lepton 3.5	1	£123.72 [27]
ST7735 SPI Display	1	£7.99 [28]
Total		£151.29

11 Conclusions

This project was able to successfully track a hot object within a reduced field of view of an infrared sensor in an article that has a small footprint in size, cost and possible in power consumption too. The project was unable to investigate the power consumption of the tracking solution as it was overlooked in the planning phase. Additionally, the integration of an inertial measurement unit and slip ring solutions were not implemented due to time constraints.

The simple tracking algorithm used was able to keep track of the hottest object within a scene and record the angle at which it was looking back to a host that was logging the data, satisfying the requirement of a telemetry output so that this system can be integrated into other applications. This can provide additional situational awareness in firefighting if installed on a small remote-controlled device such as a drone; indicate where a hazard is in an industrial setting; or keep track of heat-emitting pets or children for welfare monitoring.

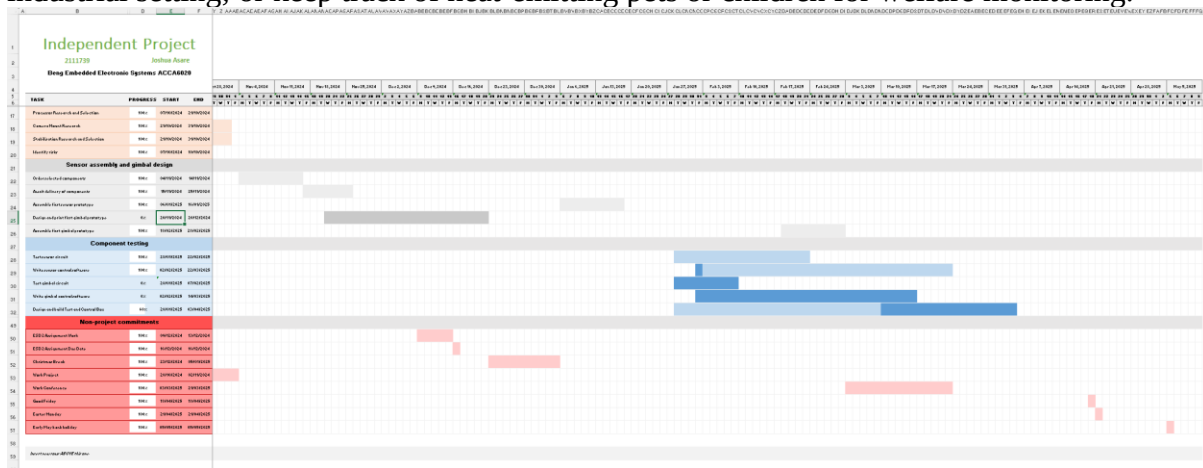


Figure 33: Project Gantt chart

12 Future Improvements

It is recommended that future work investigates camera stabilisation to reduce the influence of the movement and rotation of the assembly the camera is attached to on the tracking of a hot object. By utilizing Inertial Measurement Units (IMUs), which provide real-time data on orientation, acceleration, and angular velocity, these sensors can provide an estimation of the orientation of the system so movements caused by user motion or environmental factors can be removed from the tracking loop. When combined with stepper motors, which offer precise control over positioning and can hold their position without the need for continuous power, unlike the servo motors used in this project, the system can dynamically respond to changes in orientation and stabilize the camera effectively. These two improvements would lead to a

smoother video capture and clearer images in an environment where the sensor is not in a fixed position. Future advancements may include the development of more sophisticated algorithms for image processing which could implement more sophisticated algorithms than simply searching for a pixel with the greatest intensity. The current implementation of tracking is relatively slow, and the motors used can move much faster. It is suggested that a different algorithm with a non-linear speed is designed to move the motors faster when the desired object is further away, but slow when it is closer to the centre of the field of view to avoid the jittering issue described in this report.

13 Bibliography

- [1] Teledyne FLIR, “Visible vs. Thermal Security Cameras – Which One Do You Need,” Teledyne FLIR, [Online]. Available: <https://www.flir.com/discover/security/visible-vs-thermal-security-cameras/>. [Accessed 10 December 2024].
- [2] University of Twente., “Living Textbook | Atmospheric absorption and transmission | By ITC, University of Twente,” [Online]. Available: <https://ltb.itc.utwente.nl/491/concept/79556>. [Accessed 10 December 2024].
- [3] H. Asahi and Y. Horikoshi, III–V Semiconductors for Infrared Detectors, Wiley Telecom, 2019.
- [4] Optolong, “LRGB Filter Kit -Optolong Optics Co.,Ltd.,” Optolong, [Online]. Available: <https://www.optolong.com/cms/document/detail/id/49.html>. [Accessed 24 February 2025].
- [5] Sony Semiconductor Solutions Corporation, “Utilization of the SWIR Image Sensor in Firefighting | Other | Sony Semiconductor Solutions Group,” Sony Semiconductor Solutions Group, [Online]. Available: <https://www.sony-semicon.com/en/solution-proposal/swir-for-firefighting.html>. [Accessed 20 January 2025].
- [6] Sensors Unlimited, Sensors Unlimited, 12 August 2014. [Online]. Available: <https://www.sensorsinc.com/applications/general/aircraft-landing-systems/>. [Accessed 20 January 2025].
- [7] Infiniti Electro-Optics, , “MWIR, Visible & SWIR Camera Comparison from Multi-Sensor EO/IR PTZ Camera with LRF,” Infiniti Electro-Optics, , [Online]. Available: <https://www.infinitioptics.com/video/mwir-vs-swir-vs-visible-comparison>. [Accessed 16 January 2025].
- [8] NV5 Geospatial, “Using SWIR and LWIR Imagery to Analyze Forest Fires - NV5 Geospatial,” NV5 Geospatial, 15 January 2015. [Online]. Available: <https://www.nv5geospatialsoftware.com/Learn/Blogs/Blog-Details/using-swir-and-lwir-imagery-to-analyze-forest-fires>. [Accessed 20 January 2025].
- [9] OEMCameras, “Teledyne FLIR BOSON 320 4.3mm | OEMCameras.com,” OEMCameras, [Online]. Available: <https://www.oemcameras.com/product/flir-boson-320x256-4mm-htm/>. [Accessed 16 January 2025].
- [10] FLIR, “FLIR LEPTON® Engineering Datasheet,” 8 October 2019. [Online]. Available: <https://www.flir.com/globalassets/imported-assets/document/flir-lepton-engineering-datasheet.pdf>. [Accessed March 24 2025].
- [11] Kanazu Factory Engineering Department, “Infrared array sensor "Grid-EYE",” 29 July 2019. [Online]. Available: https://api.pim.na.industrial.panasonic.com/file_stream/main/fileversion/8452. [Accessed 16 February 2025].

- [12] G. Dorsey, "Fretting possibility in slip rings: A review," in *2017 IEEE Holm Conference on Electrical Contacts*, Denver, 2017.
- [13] J. Shepard, "Rotary slip ring connectors — what are they used for? - Electrical Engineering News and Products," EEWorld Onlin, 13 February 2023. [Online]. Available: <https://www.eeworldonline.com/rotary-slip-ring-connectors-what-are-they-used-for-faq/>. [Accessed 27 January 2025].
- [14] J. Polak, "What is Dynamic Range in Photography," 27 April 2022. [Online]. Available: <https://photographylife.com/what-is-dynamic-range>. [Accessed 24 March 2025].
- [15] NHS, "Hypothermia - NHS," NHS, 9 June 2023. [Online]. Available: <https://www.nhs.uk/conditions/hypothermia/>. [Accessed 24 March 2025].
- [16] Microsoft, "Visual Studio Code," Microsoft, [Online]. Available: <https://code.visualstudio.com/>. [Accessed 26 January 2025].
- [17] Raspberry Pi Ltd, "GitHub - raspberrypi/debugprobe: Firmware source and binaries for the RP2040-based Debug Probe accessory," Raspberry Pi Ltd, 9 January 2025. [Online]. Available: <https://github.com/raspberrypi/debugprobe>. [Accessed 25 January 2025].
- [18] "ARM Debug Interface v5 Architecture Specification," ARM, 8 February 2006. [Online]. Available: <https://developer.arm.com/documentation/ih0031/a/The-Serial-Wire-Debug-Port--SW-DP-/Introduction-to-the-ARM-Serial-Wire-Debug--SWD--protocol>. [Accessed 25 January 2025].
- [19] Digilent, "WaveForms - Digilent Reference," Digilent, 2015. [Online]. Available: <https://digilent.com/reference/software/waveforms/waveforms-3/start>. [Accessed 29 November 2024].
- [20] FLIR Systems Inc, "FLIR OEM - Lepton SPI Driver Debugging," FLIR Systems Inc, 13 December 2024. [Online]. Available: https://flir.custhelp.com/app/answers/detail/a_id/5812/~flir-oem---lepton-spi-driver-debugging. [Accessed 10 March 2024].
- [21] "Git," [Online]. Available: <https://git-scm.com/>. [Accessed 21 April 2025].
- [22] International Organization for Standardization, "ISO/IEC/IEEE 90003:2018 - Software engineering — Guidelines for the application of ISO 9001:2015 to computer software," 2018. [Online]. Available: <https://www.iso.org/standard/74348.html>. [Accessed 21 April 2025].
- [23] FLIR Systems Inc, "FLIR OEM - Lepton Power Up I2C Commands," FLIR Systems Inc, 12 December 2024. [Online]. Available: https://flir.custhelp.com/app/answers/detail/a_id/3247/related/1. [Accessed 10 March 2025].
- [24] TeraTermProject, "GitHub - TeraTermProject/teraterm," [Online]. Available: <https://github.com/TeraTermProject/teraterm>. [Accessed 3 October 2024].
- [25] Pimoroni, "Raspberry Pi Pico," [Online]. Available: <https://shop.pimoroni.com/products/raspberry-pi-pico?variant=32402092294227>. [Accessed 12 March 2025].
- [26] Amazon, "AZDelivery SG90 Micro Servo," [Online]. Available: <https://www.amazon.co.uk/AZDelivery-SG90-Micro-Servo-Parent/dp/B07CZYK379?th=1>. [Accessed 12 March 2025].

- [27] Digi-Key, "FLIR Lepton 3 Radiometry," [Online]. Available: <https://www.digikey.co.uk/en/product-highlight/f/flir/lepton-3-radiometry>. [Accessed 12 March 2025].
- [28] Amazon, "'AZDelivery 1.77 inch SPI TFT Display 128x160 pixels ST7735S 2.7V - 3.3V 50mA ST7735 Compatible with Arduino and Raspberry Pi Including E-Book! : Amazon.co.uk: Computers & Accessories'," [Online]. Available: <https://www.amazon.co.uk/dp/B078JBBPXK>. [Accessed 12 March 2025].
- [29] Altium, "Altium Designer," Altium, [Online]. Available: <https://www.altium.com/altium-designer>. [Accessed 10 October 2024].
- [30] Number One Systems, "Easy-PC Schematic and PCB Layout," Number One Systems, [Online]. Available: <https://numberone.com/layout>. [Accessed 10 October 2024].
- [31] KiCad, "KiCad EDA - Schematic Capture and PCB Design Software," KiCad, [Online]. Available: <https://www.kicad.org/>. [Accessed 10 October 2024].
- [32] Dassault Systèmes - SolidWorks Corporation, "SOLIDWORKS 3D CAD | SOLIDWORKS," Dassault Systèmes - SolidWorks Corporation, [Online]. Available: <https://www.solidworks.com/product/solidworks-3d-cad>. [Accessed 10 October 2024].
- [33] FreeCAD, "FreeCAD: Your own 3D parametric modeler," FreeCAD, [Online]. Available: <https://www.freecad.org/>. [Accessed 10 October 2024].
- [34] Autodesk, "Autodesk Fusion | 3D CAD, CAM, CAE, & PCB Cloud-Based Software | Autodesk," Autodesk, [Online]. Available: <https://www.autodesk.com/products/fusion-360>. [Accessed 10 October 2024].
- [35] F. J. Bartos, "Slip rings safely send power and data across rotary interfaces," *Control Engineering*, vol. 49, no. 5, pp. 72-72, 2002-05.
- [36] X. Liu, L. Zhao, C. Yan, S. Xu and L. Wang, "Line-of-Sight Stabilization and Reconstruction of Yaw-Pitch Seeker Installed on Rotating Platform," in *Proceedings of the 42nd Chinese Control Conference*, Tianjin, 2023.
- [37] X. Liu and B. Mo, "Line-of-sight Estimation for Missile with Roll-Pitch Seeker," in *Proceedings of the 37th Chinese Control Conference*, Wuhan, 2018.
- [38] NaturalPoint, Inc., "TrackIR5 - In Depth | TrackIR," NaturalPoint, Inc., [Online]. Available: <https://www.trackir.com/trackir5/>. [Accessed 26 October 2024].
- [39] NaturalPointKevin, "TrackIR Explained," NaturalPointKevin, 14 February 2007. [Online]. Available: https://youtu.be/_AO0F5sLdVM. [Accessed 26 October 2024].
- [40] I. Penton Media, "3D prototypes created with fused-deposition modeling," *Machine Design*, vol. 64, no. 4, p. 100, 1992-02.
- [41] H. B. Srivastava, Y. Limbu, R. Saran and A. Kumar, "Airborne Infrared Search and Track Systems," *Defence Science Journal*, vol. 57, no. 5, pp. 739-753, 2007.
- [42] D.-S. Lee and J.-H. Jo, "Pan-Tilt IR Scanning Method for the Remote Measurement of Mean Radiant Temperatures at Multi-Location in Buildings," *Remote Sensing*, vol. 13, no. 11, p. 2158, 2021-06.
- [43] National Institute of Standards and Technology, "Infrared Imaging | NIST," National Institute of Standards and Technology, 14 May 2017. [Online]. Available: <https://www.nist.gov/mml/mmsd/security-technologies-group/infrared-imaging>. [Accessed 10 February 2025].
- [44] Jens Hashagen, "SWIR Applications and Challenges: A Primer | Sep 2014 | Photonics Spectra," Photonics, September 2014. [Online]. Available: <https://www.photonics.com/Article.aspx?AID=56646>. [Accessed 10 February 2025].

- [45] G. Honis, “Step 16 for 600D Modification -Camera Disassembly Continued - by Gary Honis,” 1 November 2016. [Online]. Available: <https://garyhonislegacy.com/600Dreinstallsteps1a.html>. [Accessed 10 February 2025].
- [46] Thales, “ARTEMIS IRST – 360° Naval InfraRed Search and Track system,” Thales, [Online]. Available: <https://www.thalesgroup.com/en/markets/defence-and-security/naval-forces/above-water-warfare/artemis-irst-360deg-naval-infrared>. [Accessed 17 October 2024].
- [47] Espressif Systems, “ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems,” Espressif Systems, [Online]. Available: <https://www.espressif.com/en/products/socs/esp32>. [Accessed 10 February 2025].
- [48] Raspberry Pi, ““Silicon - Raspberry Pi Documentation”,” Raspberry Pi, [Online]. Available: <https://www.raspberrypi.com/documentation/microcontrollers/silicon.html#rp2040>. [Accessed 17 February 2024].
- [49] Espressif, ““SPI Master Driver - ESP32 - — ESP-IDF Programming Guide v5.4.1 documentation”,” Espressif, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/spi_master.html. [Accessed 28 February 2025].

1 Appendix A – pico_controller.c

```
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include "hardware/spi.h"
#include "pico/stdlib.h"
#include <stdio.h>
#include <string.h>

#include "lepton.h"
#include "servo.h"
#include "st7735.h"

extern size_t print_counter;
extern size_t discarded;
extern size_t received;
extern size_t spi_speed;
extern vospi_packet_t frame[60];

#define SEARCHY 23
#define SEARCHX 60

// Flash the LCD display on and off to test functionality
static void lcd_test(size_t interval, size_t repeats) {
    printf("LCD test: %u blinks @ %.3f s intervals \n", repeats,
           interval / 1000.0f);
    if (repeats < 1) {
        return;
    }
}
```

```

    }
    for (size_t i = 0; i < repeats; i++) {
        lcd_fill(LCD_WHITE);
        sleep_ms(interval);
        lcd_fill(LCD_RED);
        sleep_ms(interval);
        lcd_fill(LCD_GREEN);
        sleep_ms(interval);
        lcd_fill(LCD_BLUE);
        sleep_ms(interval);
        lcd_fill(LCD_BLACK);
        sleep_ms(interval);
    }
}

// Scan all valid I2C addressess
static void i2c_scan() {
    printf("\nScanning for I2C devices\n");
    printf("  0 1 2 3 4 5 6 7 8 9 A B C D E F\n");
    for (int addr = 0; addr < (1 << 7); ++addr) {
        if (addr % 16 == 0) {
            printf("%02x ", addr);
        }
        int ret;
        uint8_t rx;
        if ((addr & 0x78) == 0 || (addr & 0x78) == 0x78) {
            ret = PICO_ERROR_GENERIC;
        } else {
            ret = i2c_read_timeout_per_char_us(i2c0, addr, &rx, 1, false, 1000);
        }
        printf(ret < 0 ? "." : "@");
        printf(addr % 16 == 15 ? "\n" : " ");
    }
    printf("Done.\n");
    return;
}

int main() {
    stdio_init_all();
    servo_init(11);

    // Initialise SPI @ 20 MHz
    lcd_init();
    lcd_fill(LCD_BLACK);

    lepton_init();
    printf("SPI Speed %d MHz\n", spi_speed);
    gpio_init(PICO_DEFAULT_LED_PIN);

```

```

gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
for (size_t i = 0; i < 2; i++) {
    gpio_put(PICO_DEFAULT_LED_PIN, 0);
    sleep_ms(75);
    gpio_put(PICO_DEFAULT_LED_PIN, 1);
    sleep_ms(250);
}

printf("#####\n      Start      \n#####\n");
sleep_ms(600);
// cci_set_AGC_ENABLE_STATE();
cci_get_OEM_SOFTWARE_VERSION();
cci_set_OEM_GPIO_VSYNC_PHASE_DELAY();
cci_get_OEM_GPIO_MODE_SELECT();
cci_set_OEM_GPIO_MODE_SELECT(OEM_GPIO_MODE_VSYNC);
size_t PRINTMAX = 240;
uint16_t full_frame[4][160 * 30] = {0};
size_t previous_frame_index = 0;
float x_angle = 90.0f;
float y_angle = 90.0f;
while (true) {
    // Simple serial command interface
    // x: exit
    if (uart_is_readable(uart0)) {
        char c = getchar();
        // print out the received character
        printf("%c\n", c);
        if (c == 'x') {
            printf("====Stopping====\n");
            print_counter = PRINTMAX;
            lepton_reset();
        } else if (c == '1') {
            printf("Filling LCD with black\n");
            lcd_fill(LCD_BLACK);
        } else if (c == '2') {
            printf("Drawing checkered pattern\n");
            lcd_draw_checkered_pattern(0xF800, 0x07E0);
        } else if (c == '3') {
            printf("Drawing diagonal line from x,y = 0,0 to x,y=120,120\n");
            for (size_t i = 0; i < 120; i++) {
                lcd_write_pixel(i, i, 0x8888);
            }
        } else if (c == '4') {
            lcd_test(100, 1);
        } else if (c == '9') {
            printf("Running Servo Test\n");
            servo_test();
        }
    }
}

```

```

else if (c == 'c') {
    cci_get_CAM_STATUS();
} else if (c == 'f') {
    run_SYS_RUN_FFC();
} else if (c == 'g') {
    cci_get_OEM_GPIO_MODE_SELECT();
} else if (c == 'G') {
    cci_set_OEM_GPIO_MODE_SELECT(OEM_GPIO_MODE_VSYNC);
    print_counter = 0;
} else if (c == 'p') {
    ;
} else if (c == 'r') {
    uint8_t rx[] = {0, 0};
    cci_read(STATUS_REG, rx, 1);
    printf("%02x%02x | Error status: %d Boot: %d\n", rx[0], rx[1],
        (uint8_t)rx[0], (rx[1] & 0b10) >> 1);
} else if (c == 's') {
    i2c_scan();
} else if (c == 'S') {
    cci_get_FLIR_SERIAL_NUMBER();
} else if (c == 'u') {
    cci_get_CAM_UPTIME();
} else if (c == 'v') {
    cci_get_OEM_SOFTWARE_VERSION();
}
}
// Main packet processing loop
size_t segment = 0;
// Scan for the segment in the packet with column id 20
for (size_t packet_index = 0; packet_index < 60; packet_index++) {
    // Scan for the 20th packet because they aren't transmitted starting at
0
    uint16_t nnn = frame[packet_index].id & 0x0fff;
    uint16_t packet_no = frame[packet_index].id & 0x00ff;
    uint16_t ttt = frame[packet_index].id >> 12;
    // This is the packet that contains the frame index
    if (packet_no == 0x14) {
        segment = ttt;
    }
}
if (segment == 0) {
    memset(full_frame, 0, sizeof(full_frame));
    previous_frame_index = 0;
    // Check to see if we are receiving a duplicate frame
} else if (previous_frame_index == segment - 1) {
    previous_frame_index = segment;
    for (size_t i = 0; i < 60; i++) {
        if (segment > 4) {

```

```

        break;
    }
    for (size_t j = 0; j < 80; j++) {
        full_frame[segment - 1][60 * i + j] =
            (frame[i].payload[2 * j] << 8) | frame[i].payload[2 * j + 1];
    }
}
}

// We have received a full frame, transfer it to the LCD
if (segment == 4) {
    for (size_t y = 3; y < SEARCHY + 3; y++) {
        for (size_t x = 60; x < SEARCHX + 60; x++) {
            uint16_t pixel = full_frame[0][y * 120 + x];
            lcd_write_pixel(y, x, pixel);
        }
    }
    // uint16_t search_frame[SEARCHY][SEARCHX];
    // Search for hottest pixel
    // Search over a subset of the frame
    uint16_t hottest = 0;
    uint16_t hottest_xy[2] = {0};
    for (uint16_t y = 3; y < SEARCHY + 3; y++) {
        for (uint16_t x = 60; x < SEARCHX + 60; x++) {
            uint16_t current_pixel = full_frame[0][y * 60 + x];
            if (current_pixel >= hottest) {
                hottest = current_pixel;
                hottest_xy[0] = x;
                hottest_xy[1] = y;
            }
        }
        // printf("%d ", hottest);
    }
}
for (size_t i = 3; i < SEARCHY + 3; i++) {
    lcd_write_pixel(i, hottest_xy[0], LCD_MAGENTA);
}
for (size_t i = 60; i < SEARCHX + 60; i++) {
    lcd_write_pixel(hottest_xy[1], i, LCD_MAGENTA);
}
int16_t track_x = hottest_xy[0] - 90;
int16_t track_y = hottest_xy[1] - 14;
if ((track_x) < 8){
    x_angle++;
} else if (track_x > 4) {
    x_angle--;
}
if (x_angle > 180) {
    x_angle = 180;
}

```

```

    }
    if (x_angle < 0) {
        x_angle = 0;
    }
    servo_set_angle(11, x_angle) ;
    printf("X: %d\tY: %d\tXA: %02f\t YA: %02f\n", hottest_xy[0] - 90,
hottest_xy[1] - 14, x_angle + 45, y_angle);
    }
}
// Enter endless while loop to prevent execution of undefined code
while (true) {
}
}

```

2 Appendix B – lepton.c

```

#include <stdint.h>
#include <stdio.h>
#include <string.h>

#include "hardware/i2c.h"
#include "hardware/spi.h"
#include "pico/stdlib.h"

#include "lepton.h"

/*
FLIR Lepton Software IDD describes how the CCI (16 bit I2C interface)
is used to communication with the FLIR Lepton.
Page 11 roughly says:

1) Poll sys reg to check busy bit is clear
2) Poll FFC status to check system is ready
3) Write to data regs or block data regs
4) write the number of data words - remember this is a 16 bit interface
5) Write the command ID to the command register
6) Repeat no.1
7) Read success code from status reg
8) Read response from data regs or block data buffer

Start up command sequence:
Wait 950 ms after power on, clocks applied, and RESET_L de-asserted (high)
If shutter (3.5 does have one), wait for 5 seconds instead. This it to perform
FFC. Read status reg bit 2, which checks if the camera has booted 0: not
booted
yet 1: booted Read status reg bit 0, which checks if the interface is busy 0:
not busy 1: busy Poll SYS FFC Status command to verify the system is ready to

```

```

process commands
It seems that a total delay of ~1.6 seconds is actually required?
*/

size_t discarded = 0;
size_t received = 0;
size_t print_counter = 0;
size_t spi_speed = 0;
size_t i2c_speed = 0;

size_t dbg = 0;
vospi_packet_t frame[60];

void lepton_cci_init() {
    // Initialise the I2C port (CCI) at 400 KHz
    i2c_init(CCI_PORT, 400 * 1000);
    gpio_set_function(CCI_SDA, GPIO_FUNC_I2C);
    gpio_set_function(CCI_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(CCI_SDA);
    gpio_pull_up(CCI_SCL);
}

static void lepton_vospi_init() {
    // Initialise SPI at 24 MHz to receive VoSPI frames
    // VoSPI does NOT use MOSI.
    spi_speed = spi_init(FLIR_SPI_PORT, 24 * 1000 * 1000);
    gpio_set_function(FLIR_SPI_MISO, GPIO_FUNC_SPI);
    gpio_set_function(FLIR_SPI_SCK, GPIO_FUNC_SPI);
    spi_set_format(FLIR_SPI_PORT, 8, SPI_CPOL_1, SPI_CPHA_1, SPI_MSB_FIRST);

    gpio_init(FLIR_SPI_CS);
    gpio_set_dir(FLIR_SPI_CS, GPIO_OUT);

    gpio_init(FLIR_VSYNC_PIN);
    gpio_set_dir(FLIR_VSYNC_PIN, GPIO_IN);

    gpio_put(FLIR_SPI_CS, 1);

    // Set up VSYNC Interrupt
    gpio_set_irq_enabled_with_callback(FLIR_VSYNC_PIN, GPIO_IRQ_EDGE_RISE, true,
                                      &receive_frame_callback);
}

void vospi_resync() {
    gpio_put(FLIR_SPI_CS, 0);
    sleep_ms(950);
    gpio_put(FLIR_SPI_CS, 1);
}

```

```

// Initialise the GPIO pins for the FLIR Lepton
void lepton_init() {
    gpio_init(FLIR_PW_DOWN_PIN);
    gpio_set_dir(FLIR_PW_DOWN_PIN, GPIO_OUT);
    gpio_init(FLIR_RESET_PIN);
    gpio_set_dir(FLIR_RESET_PIN, GPIO_OUT);
    lepton_reset();

    lepton_cci_init();
    lepton_vospi_init();
    return;

    // Enable the VSYNC pin on the FLIR Lepton
}

void lepton_reset() {
    gpio_put(FLIR_PW_DOWN_PIN, 0);
    gpio_put(FLIR_RESET_PIN, 0);
    sleep_ms(200);
    gpio_put(FLIR_PW_DOWN_PIN, 1);
    sleep_ms(200);
    gpio_put(FLIR_RESET_PIN, 1);
}

static void wait() {
    uint8_t status_reg[] = {0, 0};
    do {
        cci_read(STATUS_REG, status_reg, 1);
    } while (status_reg[1] & 0b1);
    printf("%d\n", (int8_t)status_reg[0]);
}

/* Implements the Lepton Command Word (I2C Register) pp 20 FLIR Lepton
Software
IDD Lepton command word format:
15      14      13 12      11 10 9 8      7 6 5 4 3 2      1 0
Undefined OEM Bit Undefined Module ID Public Module CMD IDs Command
Type
*/

void cci_read(uint8_t reg, uint8_t *data_ptr, uint8_t word_len) {
    uint8_t reg_addr[] = {0, reg};
    i2c_write_blocking(CCI_PORT, CCI_ADDR, reg_addr, 2, false);
    i2c_read_blocking(CCI_PORT, CCI_ADDR, data_ptr, word_len << 1, false);
    // printf("cci_read: ");
    // for (uint8_t i = 0; i < word_len << 1; i++) {
    //     printf("%02x", data_ptr[i]);

```

```

    // }
    // printf("\n");
}

void cci_write(uint8_t reg, uint8_t *data_ptr, uint8_t word_len) {
    uint8_t total_length = 2 + (word_len << 1);
    uint8_t write_buffer[total_length];
    memset(write_buffer, 0, total_length);
    write_buffer[0] = 0;
    write_buffer[1] = reg;
    for (uint8_t i = 2; i < total_length; i++) {
        write_buffer[i] = data_ptr[i - 2];
    }
    i2c_write_blocking(CCI_PORT, CCI_ADDR, write_buffer, total_length, false);
    if (dbg == 0) {
        return;
    }
    printf("cci_write: %02x%02x ", write_buffer[0], write_buffer[1]);
    for (uint8_t i = 2; i < total_length; i++) {
        printf("%02x", write_buffer[i]);
    }
    printf("\n");
}

void cci_get(uint8_t *cmd_id, uint8_t cmd_len, uint8_t *res_ptr) {
    uint8_t len[] = {0, cmd_len};
    cmd_id[1] |= CCI_GET_TYPE;
    uint8_t status_reg[] = {0, 0};
    do {
        cci_read(STATUS_REG, status_reg, 1);
    } while (status_reg[1] & 0b1);
    cci_write(DATA_LENGTH_REG, len, 1);
    cci_write(COMMAND_REG, cmd_id, 1);
    do {
        cci_read(STATUS_REG, status_reg, 1);
    } while (status_reg[1] & 0b1);
    cci_read(DATA_0_REG, res_ptr, cmd_len);
    // printf("%02x%02x | Error status: %i Boot: %d\n", status_reg[0],
    // status_reg[1],
    // (int8_t)status_reg[0], (status_reg[1] & 0b10) >> 1);
}

void cci_set(uint8_t *cmd_id, uint8_t cmd_len, uint8_t *attribute,
             uint8_t attribute_len) {
    uint8_t len[] = {0, cmd_len};
    cmd_id[1] |= CCI_SET_TYPE;
    uint8_t status_reg[] = {0, 0};
    do {

```

```

    cci_read(STATUS_REG, status_reg, 1);
    if (status_reg[0] != 0) {
        printf("Error: %d\n", (int8_t)status_reg[0]);
    }
} while (status_reg[1] & 0b1);

cci_write(DATA_LENGTH_REG, len, 1);
sleep_ms(10);
cci_write(COMMAND_REG, cmd_id, 1);
sleep_ms(10);
cci_write(DATA_0_REG, attribute, attribute_len);
do {
    cci_read(STATUS_REG, status_reg, 1);
    if (status_reg[0] != 0) {
        printf("Error: %d\n", (int8_t)status_reg[0]);
    }
} while (status_reg[1] & 0b1);
}

void cci_run(uint8_t *cmd_id) {
    uint8_t len[] = {0, 0};
    cmd_id[1] |= CCI_RUN_TYPE;
    wait();
    cci_write(DATA_LENGTH_REG, len, 1);
    cci_write(COMMAND_REG, cmd_id, 1);
    wait();
}

void cci_get_FLIR_SERIAL_NUMBER(void) {
    uint8_t rx[] = {0, 0, 0, 0, 0, 0, 0, 0};
    uint16_t cmd_16bit = SYS_MODULE | SYS_FLIR_SERIAL_NUMBER;
    uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
    cci_get(cmd_8bit, 4, rx);
    printf("SYS_FLIR_SERIAL_NUMBER: %02x %02x %02x %02x\n", rx[0], rx[1], rx[2],
        rx[3], rx[4], rx[5], rx[6], rx[7]);
}

void cci_get_CAM_UPTIME(void) {
    uint8_t rx[] = {0, 0, 0, 0};
    uint16_t cmd_16bit = SYS_MODULE | SYS_CAM_UPTIME;
    uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
    cci_get(cmd_8bit, 2, rx);
    uint64_t uptime = (rx[2] << 24) | (rx[3] << 16) | (rx[0] << 8) | rx[1];
    float uptime_s = uptime / 1000.0f;
    printf("SYS_CAM_UPTIME: %.3f seconds\n", uptime_s);
}

void cci_get_CAM_STATUS(void) {

```

```

uint8_t rx[] = {0, 0, 0, 0, 0, 0, 0, 0};
uint16_t cmd_16bit = SYS_MODULE | SYS_CAM_STATUS;
uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
cci_get(cmd_8bit, 4, rx);
printf("SYS_CAM_STATUS: %02x %02x %02x %02x %02x %02x %02x %02x\n", rx[0],
        rx[1], rx[2], rx[3], rx[4], rx[5], rx[6], rx[7]);
}

void cci_get_OEM_GPIO_MODE_SELECT(void) {
    uint8_t rx[4];
    memset(rx, 0, sizeof(rx));
    uint16_t cmd_16bit = OEM_BIT | OEM_MODULE | OEM_GPIO_MODE_SELECT;
    uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
    cci_get(cmd_8bit, 2, rx);
    printf("OEM_GPIO_MODE_SELECT: %02x %02x %02x %02x\n", rx[3], rx[2], rx[1],
        rx[0]);
}

void cci_get_OEM_SOFTWARE_VERSION(void) {
    uint8_t rx[] = {0, 0, 0, 0, 0, 0, 0, 0};
    uint16_t cmd_16bit = OEM_BIT | OEM_MODULE | OEM_SOFTWARE_VERSION;
    uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
    cci_get(cmd_8bit, 4, rx);
    printf("OEM_SOFTWARE_VERSION: %02x %02x %02x %02x %02x %02x %02x %02x\n",
        rx[0], rx[1], rx[2], rx[3], rx[4], rx[5], rx[6], rx[7]);
}

void cci_set_OEM_GPIO_MODE_SELECT(oem_gpio_mode_e mode) {
    uint8_t tx[4];
    uint8_t cmd_8bit[2];
    uint8_t len[2] = {0, 2};
    memset(tx, 0, 4);
    memset(cmd_8bit, 0, 2);

    // tx[3] = mode;
    // tx[2] = mode;
    tx[1] = mode;
    // tx[0] = mode;
    uint16_t cmd_16bit =
        OEM_BIT | OEM_MODULE | OEM_GPIO_MODE_SELECT | CCI_SET_TYPE;
    cmd_8bit[0] = (cmd_16bit >> 8) & 0xFF;
    cmd_8bit[1] = cmd_16bit & 0xFF;
    wait();
    cci_write(DATA_0_REG, tx, 1);
    cci_write(DATA_1_REG, tx + 2, 1);
    cci_write(COMMAND_REG, cmd_8bit, 1);
    cci_write(DATA_LENGTH_REG, len, 1);
    wait();
}

```

```

    // cci_set(cmd_8bit, 2, tx, 2);
}

void cci_set_OEM_GPIO_VSYNC_PHASE_DELAY(void) {
    uint8_t tx[4] = {0, 3, 0, 0};
    uint8_t len[2] = {0, 2};
    uint16_t cmd_16bit =
        OEM_BIT | OEM_MODULE | OEM_GPIO_VSYNC_PHASE_DELAY | CCI_SET_TYPE;
    uint8_t cmd_8bit[2] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};

    wait();
    cci_write(DATA_0_REG, tx, 1);
    cci_write(DATA_1_REG, tx + 2, 1);
    cci_write(COMMAND_REG, cmd_8bit, 1);
    cci_write(DATA_LENGTH_REG, len, 1);
    wait();
}

void cci_set_AGC_ENABLE_STATE(void) {
    uint8_t tx[4] = {0, 1, 0, 0};
    uint8_t len[2] = {0, 2};
    uint16_t cmd_16bit = AGC_MODULE | AGC_ENABLE_STATE | CCI_SET_TYPE;
    uint8_t cmd_8bit[2] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};

    wait();
    cci_write(DATA_0_REG, tx, 1);
    cci_write(DATA_1_REG, tx + 2, 1);
    cci_write(COMMAND_REG, cmd_8bit, 1);
    cci_write(DATA_LENGTH_REG, len, 1);
    wait();
}

void cci_get_AGC_ENABLE_STATE(void) {
    uint8_t rx[] = {0, 0, 0, 0};
    uint16_t cmd_16bit = AGC_MODULE | AGC_ENABLE_STATE;
    uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
    cci_get(cmd_8bit, 2, rx);
    printf("AGC_ENABLE_STATE: %02x %02x %02x %02x\n", rx[0], rx[1], rx[2],
rx[3]);
}

void run_SYS_RUN_FFC() {
    uint16_t cmd_16bit = SYS_MODULE | SYS_RUN_FFC;
    uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
    cci_run(cmd_8bit);
}

void run_OEM_BIT_TEST() {

```

```

uint16_t cmd_16bit = OEM_BIT | OEM_MODULE | OEM_BIT_TEST;
uint8_t cmd_8bit[] = {(cmd_16bit >> 8) & 0xFF, cmd_16bit & 0xFF};
cci_run(cmd_8bit);
}

void lepton_cci_ping() {
    uint8_t data[] = {0, 0};
    printf("Ping: %d %d\n", data[0], data[1]);
}

void vospi_sync() {
    gpio_init(FLIR_VSYNC_PIN);
    gpio_set_dir(FLIR_VSYNC_PIN, GPIO_IN);
}

// Packet structure from the FLIR Lepton 3's VoSPI
// ID: 2 bytes
// CRC: 2 bytes (Calculated over the entire packet, with ID and CRC set to 0)
// PAYLOAD: 160 bytes

void print_packet(vospi_packet_t *packet) {
    printf("Packet ID: %04x, CRC: %04x\n", packet->id, packet->crc);
    for (size_t i = 0; i < sizeof(packet->payload); i++) {
        printf("%02x ", packet->payload[i]);
        if ((i + 1) % 16 == 0) {
            printf("\n");
        }
    }
}

void receive_frame_callback(uint gpio, uint32_t event_mask) {
    // for (volatile size_t _ = 0; _ < 5000; _++){
    uint8_t rx[164] = {0};
    // Lepton 3.5 has 60 packets per frame, but 4 frames per image
    for (size_t i = 0; i < 60; i++) {
        gpio_put(FLIR_SPI_CS, 0);
        int rx_count = spi_read_blocking(FLIR_SPI_PORT, 0, rx, sizeof(rx));
        gpio_put(FLIR_SPI_CS, 1);

        uint16_t id = (rx[0] << 8) | rx[1];
        if ((id & 0xFF0) == 0xFF0) {
            // This is a DISCARD packet, so we discard it
            discarded++;
        } else {
            frame[i].id = rx[0] << 8 | rx[1];
            frame[i].crc = rx[2] << 8 | rx[3];
            memcpy(&frame[i].payload, &rx[4], sizeof(rx) - 4);
            received++;
        }
    }
}

```

```

    }
    print_counter++;
}

```

3 Appendix D – servo.c

```

#include "hardware/pwm.h"
#include "pico/stdlib.h"

#define SERVO_PIN 11
#define MIN_PULSE_WIDTH 500
#define MAX_PULSE_WIDTH 2500
#define PWM_PERIOD 40000

void servo_init(uint gpio) {
    gpio_set_function(gpio, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(gpio);

    pwm_set_clkdiv(slice_num, 64.0f); // Set clock divider
    pwm_set_wrap(slice_num, 40000); // Set PWM period (20ms for 50Hz)
    pwm_set_enabled(slice_num, true);
}

void servo_set_angle(uint gpio, float angle) {
    if (angle < 0.0f)
        angle = 0.0f;
    if (angle > 180.0f)
        angle = 180.0f;

    uint slice_num = pwm_gpio_to_slice_num(gpio);
    uint16_t pulse_width =
        MIN_PULSE_WIDTH +
        (uint16_t)((angle / 180.0f) * (MAX_PULSE_WIDTH - MIN_PULSE_WIDTH));
    uint16_t duty_cycle = (pulse_width * PWM_PERIOD) / PWM_PERIOD;

    pwm_set_chan_level(slice_num, pwm_gpio_to_channel(gpio), duty_cycle);
}

void servo_test() {
    servo_init(SERVO_PIN);
    for (float angle = 0; angle <= 180; angle += 10) {
        servo_set_angle(SERVO_PIN, angle);
        sleep_ms(50);
    }
    for (float angle = 180; angle >= 0; angle -= 10) {
        servo_set_angle(SERVO_PIN, angle);
        sleep_ms(50);
    }
}

```

```
}  
}
```

4 Appendix D – st7735.c

```
#include "hardware/gpio.h"  
#include "hardware/spi.h"  
#include "pico/stdlib.h"  
  
#include "lepton.h"  
#include "st7735.h"  
  
// Define pins  
#define SPI_PORT spi1  
#define PIN_DC 21  
#define PIN_CS 17  
#define PIN_RST 20  
#define PIN_SCK 26  
#define PIN_MOSI 27  
  
static void lcd_write_command(uint8_t cmd) {  
    gpio_put(PIN_DC, 0); // Logic low indicates a command  
    gpio_put(PIN_CS, 0); // Select the display  
    spi_write_blocking(SPI_PORT, &cmd, 1);  
    gpio_put(PIN_CS, 1); // Deselect the display  
}  
  
static void lcd_write_data(uint8_t *data, size_t len) {  
    gpio_put(PIN_DC, 1); // Logic high indicates data  
    gpio_put(PIN_CS, 0); // Select the display  
    spi_write_blocking(SPI_PORT, data, len);  
    gpio_put(PIN_CS, 1); // Deselect the display  
}  
  
void lcd_init() {  
    // Initialize SPI at 10 MHz  
    spi_init(SPI_PORT, 50 * 1000 * 1000);  
    gpio_set_function(PIN_SCK, GPIO_FUNC_SPI);  
    gpio_set_function(PIN_MOSI, GPIO_FUNC_SPI);  
  
    // Initialize control pins  
    gpio_init(PIN_CS);  
    gpio_set_dir(PIN_CS, GPIO_OUT);  
    gpio_put(PIN_CS, 1);  
    // Used to toggle between command and data  
    gpio_init(PIN_DC);  
    gpio_set_dir(PIN_DC, GPIO_OUT);  
}
```

```

// Used to Reset the display
gpio_init(PIN_RST);
gpio_set_dir(PIN_RST, GPIO_OUT);

// Reset the display in hardware
gpio_put(PIN_RST, 0);
sleep_ms(150);
gpio_put(PIN_RST, 1);
sleep_ms(150);

// Initialize the display
lcd_write_command(ST7735_SWRESET);
sleep_ms(150);
lcd_write_command(ST7735_SLPOUT);
sleep_ms(150);
lcd_write_command(ST7735_COLMOD);
// Set the color mode to 16 bit
uint8_t col_mode = 0x05;
lcd_write_data(&col_mode, 1);

lcd_write_command(ST7735_DISPON);
}

void lcd_set_address_window(uint16_t x0, uint16_t y0, uint16_t x1,
                           uint16_t y1) {
    uint8_t data[4];
    data[0] = 0x00;
    data[2] = 0x00;

    lcd_write_command(ST7735_CASET); // Columns
    data[1] = x0;
    data[3] = x1;
    lcd_write_data(data, 4);

    lcd_write_command(ST7735_RASET); // Rows
    data[1] = y0;
    data[3] = y1;
    lcd_write_data(data, 4);
}

void lcd_write_pixel(uint16_t x, uint16_t y, uint16_t color) {
    if (x >= 128 || y >= 160) {
        return; // Out of bounds, do nothing
    }

    uint8_t data[2];
    lcd_set_address_window(x, y, x, y);
    lcd_write_command(ST7735_RAMWR); // Write to the LCD's RAM

```

```

    data[0] = color >> 8;
    data[1] = color & 0xFF;
    lcd_write_data(data, 2);
}

void lcd_fill(uint16_t color) {
    uint8_t data[2] = {color >> 8, color & 0xFF};

    lcd_set_address_window(0, 0, 127, 159);
    lcd_write_command(ST7735_RAMWR); // Write to the LCD's RAM

    for (int i = 0; i < 128 * 160; i++) {
        lcd_write_data(data, 2);
    }
}

void lcd_draw_frame(uint16_t *buf) {
    lcd_set_address_window(0, 0, 39, 119);
    lcd_write_command(ST7735_RAMWR);
    lcd_write_data(buf, 4800);
}

void lcd_draw_partial_frame(uint16_t *buf) {
    for (int i = 0; i < 60 * 80; i++) {
        // Convert from RAW14 from FLIR to RGB565 for ST7735
        uint16_t pixel = 255 * (buf[i] / ((1 << 14) - 1));
        uint8_t r = pixel * 31;
        uint8_t g = pixel * 63;
        uint8_t b = pixel * 31;

        uint16_t data_16 = (r << 11) | (g << 5) | b;

        uint8_t data[2] = {data_16 >> 8, data_16 & 0xFF};
        lcd_write_data(data, 2);
    }
}

void lcd_draw_checkered_pattern(uint16_t color1, uint16_t color2) {
    for (uint16_t y = 0; y < 160; y++) {
        for (uint16_t x = 0; x < 128; x++) {
            if ((x / 8 + y / 8) % 2 == 0) {
                lcd_write_pixel(x, y, color1);
            } else {
                lcd_write_pixel(x, y, color2);
            }
        }
    }
}

```

```
}
```

5 Appendix E – lepton.h

```
#ifndef LEPTON_H
#define LEPTON_H

#include <stdint.h>

// uint8_t txbuf[16];
// uint8_t rxbuf[16];

#define CCI_ADDR 0x2A
// Module IDs which specify which camera subsystem to access
typedef enum module_id_t {
    AGC_MODULE = 0x0100,
    SYS_MODULE = 0x0200,
    VID_MODULE = 0x0300,
    OEM_MODULE = 0x0800,
    RAD_MODULE = 0x0E00,
} module_id_t;

// AGC Module Command IDs
typedef enum agc_cmd_id_t {
    AGC_ENABLE_STATE = 0x00,
    AGC_POLICY = 0x04,
    AGC_ROI = 0x08,
    AGC_STATISTICS = 0x0C,
    AGC_HISTOGRAM_CLIP_PERCENT = 0x10,
    AGC_HISTOGRAM_TAIL_SIZE = 0x14,
    AGC_LINEAR_MAX_GAIN = 0x18,
    AGC_LINEAR_MIDPOINT = 0x1C,
    AGC_LINEAR_DAMPENING_FACTOR = 0x20,
    AGC_HEQ_DAMPENING_FACTOR = 0x24,
    AGC_HEQ_MAX_GAIN = 0x28,
    AGC_HEQ_CLIP_LIMIT_HIGH = 0x2C,
    AGC_HEQ_CLIP_LIMIT_LOW = 0x30,
    AGC_HEQ_BIN_EXTENSION = 0x34,
    AGC_HEQ_MIDPOINT = 0x38,
    AGC_HEQ_EMPTY_COUNTS = 0x3C,
    AGC_HEQ_NORMALIZATION_FACTOR = 0x40,
    AGC_HEQ_SCALE_FACTOR = 0x44,
    AGC_CALC_ENABLE_STATE = 0x48
} agc_cmd_id_t;

// SYS Module Command IDs
typedef enum sys_cmd_id_t {
    SYS_PING = 0x00,
    SYS_CAM_STATUS = 0x04,
```

```

SYS_FLIR_SERIAL_NUMBER = 0x08,
SYS_CAM_UPTIME = 0x0C,
SYS_AUX_TEMPERATURE_KELVIN = 0x10,
SYS_FPA_TEMPERATURE_KELVIN = 0x14,
SYS_TELEMETRY_ENABLE_STATE = 0x18,
SYS_TELEMETRY_LOCATION = 0x1C,
SYS_EXECUTE_FRAME_AVERAGE = 0x20,
SYS_NUM_FRAMES_TO_AVERAGE = 0x24,
SYS_CUST_SERIAL_NUMBER = 0x28,
SYS_SCENE_STATISTICS = 0x2C,
SYS_SCENE_ROI = 0x30,
SYS_THERMAL_SHUTDOWN_COUNT = 0x34,
SYS_SHUTTER_POSITION = 0x38,
SYS_FFC_SHUTTER_MODE_OBJ = 0x3C,
SYS_RUN_FFC = 0x42,
SYS_FFC_STATUS = 0x44
} sys_cmd_id_t;
// VID Module Command IDs
typedef enum vid_cmd_id_t {
    VID_POLARITY_SELECT = 0x00,
    VID_LUT_SELECT = 0x04,
    VID_LUT_TRANSFER = 0x08,
    VID_FOCUS_CALC_ENABLE = 0x0C,
    VID_FOCUS_ROI = 0x10,
    VID_FOCUS_THRESHOLD = 0x14,
    VID_FOCUS_METRIC = 0x18,
    VID_SBNUC_ENABLE = 0x1C,
    VID_GAMMA_SELECT = 0x20,
    VID_FREEZE_ENABLE = 0x24
} vid_cmd_id_t;

#define SYS_PING 0x00
// VID Module Command IDs
// OEM Module Command IDs
#define OEM_BIT 0x4000
typedef enum oem_cmd_id_t {
    OEM_POWER_DOWN = 0x00,
    OEM_STANDBY = 0x04,
    OEM_LOW_POWER_MODE_1 = 0x08,
    OEM_LOW_POWER_MODE_2 = 0x0C,
    OEM_BIT_TEST = 0x10,
    OEM_MASK_REVISION = 0x14,
    OEM_MASTER_ID = 0x18,
    OEM_FLIR_PART_NUMBER = 0x1C,
    OEM_SOFTWARE_VERSION = 0x20,
    OEM_VIDEO_OUTPUT_ENABLE = 0x24,
    OEM_VIDEO_OUTPUT_FORMAT = 0x28,
    OEM_VIDEO_OUTPUT_SOURCE = 0x2C,

```

```

OEM_VIDEO_OUTPUT_CHANNEL = 0x30,
OEM_VIDEO_GAMMA_ENABLE = 0x34,
OEM_CUST_PART_NUMBER = 0x38,
OEM_VIDEO_OUTPUT_CONSTANT = 0x3C,
OEM_REBOOT = 0x40,
OEM_FFC_NORMALIZATION_TARGET = 0x44,
OEM_STATUS = 0x48,
OEM_SCENE_MEAN_VALUE = 0x4C,
OEM_POWER_MODE = 0x50,
OEM_GPIO_MODE_SELECT = 0x54,
OEM_GPIO_VSYNC_PHASE_DELAY = 0x58,
OEM_USER_DEFAULTS = 0x5C,
OEM_USER_DEFAULTS_RESTORE = 0x60,
OEM_SHUTTER_PROFILE_OBJ = 0x64,
OEM_THERMAL_SHUTDOWN_ENABLE_STATE = 0x68,
OEM_BAD_PIXEL_REPLACE_CONTROL = 0x6C,
OEM_TEMPORAL_FILTER_CONTROL = 0x70,
OEM_COLUMN_NOISE_ESTIMATE_CONTROL = 0x74,
OEM_PIXEL_NOISE_ESTIMATE_CONTROL = 0x78
} oem_cmd_id_t;
// RAD Module Command IDs

// I2C command types
typedef enum cci_cmd_t {
    CCI_GET_TYPE = 0x00,
    CCI_SET_TYPE = 0x01,
    CCI_RUN_TYPE = 0x02,
} cci_cmd_t;

// CCI Registers
typedef enum cci_reg_t {
    STATUS_REG = 0x02,
    COMMAND_REG = 0x04,
    DATA_LENGTH_REG = 0x06,
    DATA_0_REG = 0x08,
    DATA_1_REG = 0x0A,
    DATA_2_REG = 0x0C,
    DATA_3_REG = 0x0E,
    DATA_4_REG = 0x10,
    DATA_5_REG = 0x12,
    DATA_6_REG = 0x14,
    DATA_7_REG = 0x16,
    DATA_8_REG = 0x18,
    DATA_9_REG = 0x1A,
    DATA_10_REG = 0x1C,
    DATA_11_REG = 0x1E,
    DATA_12_REG = 0x20,
    DATA_13_REG = 0x22,

```

```

    DATA_14_REG = 0x24,
    DATA_15_REG = 0x26,
    DATA_CRC_REG = 0x28
} cci_reg_t;

// CCI enums
typedef enum oem_gpio_mode_e {
    OEM_GPIO_MODE_GPIO = 0,
    OEM_GPIO_MODE_I2C_MASTER = 1,
    OEM_GPIO_MODE_SPI_MASTER_VLB_DATA = 2,
    OEM_GPIO_MODE_SPIO_MASTER_REG_DATA = 3,
    OEM_GPIO_MODE_SPI_SLAVE_VLB_DATA = 4,
    OEM_GPIO_MODE_VSYNC = 5
} oem_gpio_mode_e, *oem_gpio_mode_p;

// CCI Error Codes from LEPTON_ErrorCodes.h

#define REG_SYS_TEST_PATTERN 0x4C
// I2C config
#define CCI_PORT i2c0
#define CCI_SDA 8
#define CCI_SCL 9

// SPI config
#define FLIR_SPI_PORT spi0
#define FLIR_SPI_MISO 4
#define FLIR_SPI_CS 5
#define FLIR_SPI_SCK 2
// FLIR Lepton GPIO config
#define FLIR_VSYNC_PIN 3
#define FLIR_PW_DOWN_PIN 6
#define FLIR_RESET_PIN 7

// Function declarations
void lepton_init();
void lepton_reset(void);
void lepton_cci_init(void);
void lepton_cci_ping(void);
void vospi_resync(void);
void vospi_sync(void);
// void receive_frame(uint gpio, uint32_t event_mask);

void cci_read(uint8_t reg, uint8_t *data_ptr, uint8_t len);
void cci_write(uint8_t reg, uint8_t *data_ptr, uint8_t len);
void cci_get(uint8_t *cmd_id, uint8_t cmd_len, uint8_t *res_ptr);
void cci_set(uint8_t *cmd_id, uint8_t cmd_len, uint8_t *attribute, uint8_t attribute_len);
void cci_run(uint8_t *cmd_id);

```

```

// CCI GET commands
void cci_get_CAM_STATUS(void);
void cci_get_FLIR_SERIAL_NUMBER(void);
void cci_get_CAM_UPTIME(void);
void cci_get_OEM_GPIO_MODE_SELECT(void);
void cci_get_OEM_SOFTWARE_VERSION(void);
void cci_get_AGC_ENABLE_STATE(void);
// CCI SET commands
void cci_set_OEM_GPIO_MODE_SELECT(oem_gpio_mode_e mode);
void cci_set_OEM_GPIO_VSYNC_PHASE_DELAY(void);
void cci_set_AGC_ENABLE_STATE(void);
// CCI RUN commands
void run_SYS_RUN_FFC(void);

// VoSPI commands
typedef struct vospi_packet_t {
    uint16_t id;
    uint16_t crc;
    uint8_t payload[160];
} vospi_packet_t;

typedef struct vospi_frame_t {
    vospi_packet_t packet[60];
} vospi_frame_t;

typedef uint16_t full_frame_t[4][60 * 80];

extern vospi_frame_t frames[4];

void receive_frame_callback(uint gpio, uint32_t event_mask);
void print_packet(vospi_packet_t *packet);
#define FLIP_BYTES(x) (x >> 8) | (x << 8)

extern size_t print_counter;
extern size_t discarded;
extern size_t received;
extern size_t spi_speed;
#endif // LEPTON_H

```

6 Appendix F – servo.h

```

#ifndef SERVO_H
#define SERVO_H

void servo_test(void);
void servo_set_angle(uint gpio, float angle);

```

```
void servo_init(uint gpio);  
#endif // SERVO_H
```

7 Appendix G – st7735.h

```
#ifndef ST7735_H  
#define ST7735_H  
  
#include <stdint.h>  
#include <stddef.h>  
  
// ST7735 commands  
#define ST7735_SWRESET 0x01  
#define ST7735_SLPOUT 0x11  
#define ST7735_NORON 0x13  
#define ST7735_INVOFF 0x20  
#define ST7735_INVON 0x21  
#define ST7735_DISPON 0x29  
#define ST7735_CASET 0x2A  
#define ST7735_RASET 0x2B  
#define ST7735_RAMWR 0x2C  
#define ST7735_MADCTL 0x36  
#define ST7735_COLMOD 0x3A  
#define ST7735_FRMCTR1 0xB1  
#define ST7735_DISSET5 0xB6  
#define ST7735_PWCTR1 0xC0  
#define ST7735_PWCTR2 0xC1  
#define ST7735_VMCTR1 0xC5  
#define ST7735_GMCTRP1 0xE0  
#define ST7735_GMCTRN1 0xE1  
  
// Colours  
#define LCD_WHITE 0xFFFF  
#define LCD_BLACK 0x0000  
#define LCD_RED 0xF800  
#define LCD_GREEN 0x07E0  
#define LCD_BLUE 0x001F  
#define LCD_YELLOW 0xFFE0  
#define LCD_CYAN 0x07FF  
#define LCD_MAGENTA 0xF81F  
  
// Function declarations  
void lcd_init();  
void lcd_set_address_window(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1);  
void lcd_write_pixel(uint16_t x, uint16_t y, uint16_t color);  
void lcd_fill(uint16_t color);
```

```

void lcd_draw_checkered_pattern(uint16_t color1, uint16_t color2);
void lcd_draw_frame(uint16_t *buf);
void lcd_draw_partial_frame(uint16_t *buf);
#endif // ST7735_H

```

8 Appendix H – plot telem.py

```

from matplotlib import pyplot as plt

filename = input()
with open(filename, 'r') as f:
    data = f.read()

x_values = []
y_values = []
a_values = []

for line in data.strip().split('\n'):
    buf = line.split('\t')
    x = int(buf[0].split(':')[1])
    y = int(buf[1].split(':')[1])
    a = float(buf[2].split(':')[1])
    x_values.append(x)
    y_values.append(y)
    a_values.append(a)

plt.plot(x_values, label='X object offset')
plt.plot(y_values, label='Y object offset')
plt.plot(a_values, label='Pan angle')
plt.title('Plot of X and Y offsets in pixels')
plt.xlabel('Log no.')
plt.ylabel('Distance from centre (pixels)')
plt.axhline(0, color='black', linewidth=0.5, ls='--')
plt.axvline(0, color='black', linewidth=0.5, ls='--')
plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
plt.legend()
plt.show()

```