

Multi-Spectral Tracking Rig

Group Project

Embedded Electronic Systems

Word count 5,649

Contents

1	List of tables	4
2	List of figures	5
3	Introduction – Jack.....	6
4	Background Research & Literature Review	7
4.1	Ultraviolet Radiation – Jack.....	7
4.2	Infrared Radiation – Jack.....	8
4.3	ROCK Pi 5 Model B – Jack.....	9
4.4	Lepton Camera – Jack	9
4.5	Digital Image Processing – Josh	9
4.6	Parallel Manipulators v Pan Tilt units - Ethan	9
4.7	Stepper Motors - Ethan.....	9
4.8	Stepper Motor Drivers - Ethan.....	10
4.9	Pan Tilt – Robbie	11
4.10	3D Printing – Ethan	11
4.11	Limit switches – Robbie	11
5	Development Methodology.....	12
5.1	Jack.....	12
5.2	Algorithm Development – Josh.....	12
6	Design.....	13
6.1	Overall –Jack	13
6.2	PTU Design – Robbie	13
6.3	PTU Hardware – Ethan.....	13
6.4	Algorithm Design – Josh.....	14
7	Implementation	15
7.1	3D Printing – Ethan	15
7.2	Calibration – Robbie.....	15
7.3	Standoffs – Robbie	15
7.4	Circuit – Jack.....	15
7.5	Procurement – Jack.....	15
7.6	Software – Josh	15
7.7	Firmware - Ethan.....	18
8	Results and Analysis	19
8.1	Testing write-up – Robbie.....	19
8.2	Tracking reliability – Josh	19

9	Evaluation.....	20
9.1	Jack.....	20
10	Conclusions and Recommendations.....	21
10.1	Jack.....	21
10.2	Robbie	21
10.3	Josh	22
11	Reflection	23
11.1	Ethan	23
11.2	Jack.....	23
11.3	Joshua	23
11.4	Robbie	24
12	References	25
13	Bibliography	26
14	Glossary.....	28
15	List of abbreviations.....	29
16	List of symbols.....	30
	Appendix 1 – UV Radiation	31
	Appendix 2 – Project Plan – Jack.....	32
17	Annex 1 – Motor controller functionality – Ethan	33
18	Annex 2 – Motor controller functionality 2 – Ethan	34
19	Annex 3 – Alternative PTU Designs – Robbie#.....	35
20	Annex 4 – Split Rings - Robbie.....	35
21	Annex 5 – Edge Detection	35
22	Annex 6 – Micropython Source Code – Ethan	36

1 List of tables

Table 1 - UV Classifications	7
------------------------------------	---

2 List of figures

Figure 1 CMOS Sensor Diagram ^[4]	8
Figure 2 Microbolometer Diagram ^[8]	8
Figure 3 NEMA 11 stepper motor internal view	10
Figure 4 PTU hardware connections	14
Figure 5 An image of an oscilloscope probing a 25MHz signal	16
Figure 6 Software flow chart.....	17
Figure 7 Demand chunking illustration	18
Figure 8 Function to move both axis at same time.....	18
Figure 9 Tracking demand reading and conditioning	33
Figure 10 moveDegrees function.....	34
Figure 11 xSteps function.....	34
Figure 12 Slip Ring.....	35
Figure 13 Sobel operator and threshold applied to an infrared image of a drone.....	36

3 Introduction – Jack

As a group we were tasked by Threat Evaluation Group (TEG) to produce a Multi-Spectral Tracking Rig (MSTR). The Statement of Requirement (SoR) outlined that the final product must be able to track objects in the Infrared (IR) and/or Ultra-Violet (UV) spectrum, and utilise a rate table to be able to point at the tracked targets or perform a search pattern.

Throughout this project we aim to answer the question: How can a Multi-Spectral Tracking Rig capability be developed in a cost effective and time efficient manner? This is important to our organisation as we are ultimately funded by the tax payer.

In the following chapters we will go on to research subjects relevant to the production of our tracking rig in order to help us better understand what technology, manufacturing practices and existing examples are available for us to employ in the production of this capability. We will then walk through the design, manufacture and testing processes to explain and evaluate our final product.

4 Background Research & Literature Review

4.1 Ultraviolet Radiation – Jack

UV (Ultraviolet radiation) is a region of frequencies in the electromagnetic spectrum. UV has shorter wavelengths and is higher frequency than visible light spanning 400 nm down to 10nm (750THz to 30PHz), divided into three major bands on Earth.

UV-A has the longest wavelength (315-400 nm) and makes up 95% of UV penetrating the atmosphere. UV-A can also penetrate into deeper layers of the skin and is associated with tanning and skin damage.

UV-B has a medium wavelength (280-315 nm) with the capability to penetrate the atmosphere. However, the majority of solar UV-B is filtered by the atmosphere through absorption – as a result it only makes up the remaining 5%. This type of UV radiation is “very biologically active but cannot penetrate beyond the superficial skin layers”, responsible for delayed tanning and sun burn.

UV-C is the shortest UV wavelength (100-280 nm), but is completely absorbed by the atmosphere.^[1]

Table 1 - UV Classifications

Name	Abbreviation	Wavelength (nm)	Notes
Ultraviolet A	UV-A	315-400	Long-wave UV, Blacklight, not absorbed by the ozone layer: soft UV
Ultraviolet B	UV-B	280-315	Medium-wave UV, mostly absorbed by the ozone layer: intermediate UV; Dorno radiation
Ultraviolet C	UV-C	100-280	Short-wave UV, Germicidal, Ionizing at shorter wavelengths, completely absorbed by the ozone layer and atmosphere: hard UV
Near Ultraviolet	N-UV	300-400	Visible to birds, insects and fish
Middle Ultraviolet	M-UV	200-300	
Far Ultraviolet	F-UV	122-200	Ionizing at shorter wavelengths
Extreme Ultraviolet	E-UV	10-121	Entirely ionizing by some definitions; completely absorbed by the atmosphere
Vacuum Ultraviolet	V-UV	100-200	Strongly absorbed by atmospheric oxygen, though 150-200 nm wavelengths can propagate through nitrogen.

As UV is highly reactive with most materials, specialised materials for passing and capturing UV photons are used. These properties of these materials make them highly UV transparent, for example; UV-transparent fused silica and calcium fluoride; UV-absorbent CMOS (Complementary Metal Oxide Semiconductor) sensors. CMOS sensors contain rows of photodiodes coupled with individual amplifiers to amplify the electric signal generated by the photodiodes.^[3]

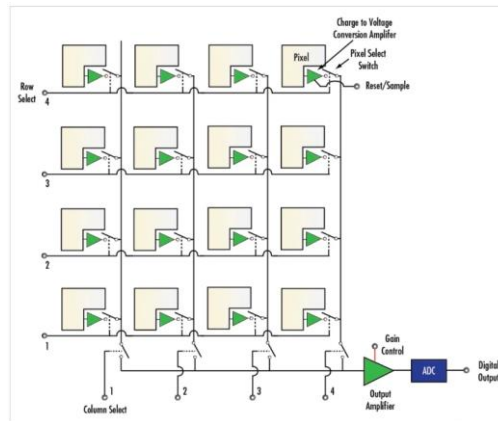


Figure 1 CMOS Sensor Diagram [4]

4.2 Infrared Radiation – Jack

IR (Infrared radiation) is a region of frequencies in the electromagnetic spectrum, IR has longer wavelengths and is lower frequency than Visible Light spanning 1 mm down to 700nm (300GHz up to 430THz). This region is divided into five major bands.

- Near Infrared
- Short-wave Infrared
- Medium-wave Infrared
- Long-wave Infrared
- Far Infrared

Short, Medium, and Long-wave Infrared are the bands that can be used in imaging applications. Typically, thermal imagers are set in Medium or Long wave bands as these frequencies sit within the atmospheric window of thermal IR.^[5] Short-wave IR imagers use reflected IR to image objects, allowing them to be used in both day and night settings.^[6]

Long-wave IR imagers use microbolometers to detect and measure the incident IR radiation to produce a heat map of the image on the lens. A bolometer is an extremely sensitive digital thermometer which uses electrical resistance that varies with temperature. A microbolometer is a specific type of bolometer used as a detector in a thermal camera, comprised of an array of microscopic Vanadium Oxide or amorphous silicon heat sensors on top a corresponding grid of silicon. IR radiation strikes the Vanadium Oxide sensor and changes its electrical resistance. This resistance change is measured and processed into temperature values.^[7]

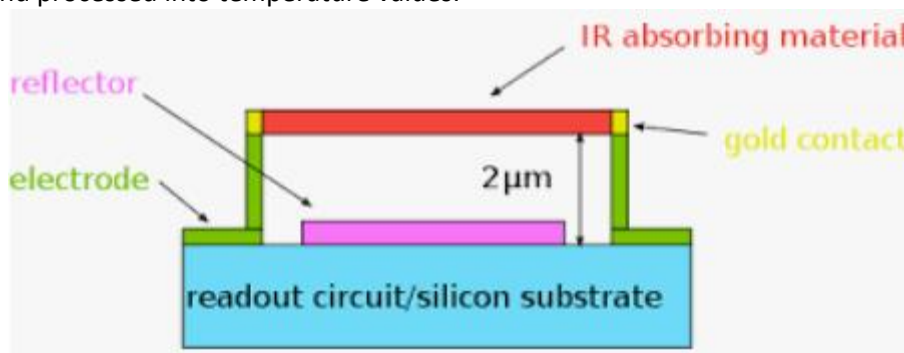


Figure 2 Microbolometer Diagram [8]

4.3 ROCK Pi 5 Model B – Jack

The Rock SBC has been chosen over other options because of its processing power and ease of use. The Rock Pi 5B sports a Rockchip RK3588 SoC and a Quad-core ARM Cortex-A76 MPCore and ARM Cortex-A55 MPCore processors.^[9]

The SBC also has an embedded Multi-Media Card (eMMC) module. This allows for compact and fast storage solutions.

4.4 Lepton Camera – Jack

When sourcing the camera for this project we had to consider properties that would affect the image produced and impact image processing. Resolution impacts the clarity of the image but also increases the processing requirements of our computer. Sensitivity would allow for a greater dynamic range to the images making a threshold more difficult to apply due to the less sudden change in colour. Radiometry is the process of compensating for other sources of radiation and atmospheric effects that interfere with the image produced.^[11]

The FLIR Lepton thermal imaging camera is a LWIR radiometric-capable camera with a resolution of 160 x 120, within the scope of the brief, providing sufficient resolution to make out objects with a low processing overhead. This camera supports SPI video output which enables us to communicate with the computer we have purchased.^{[12], [13]}

4.5 Digital Image Processing – Josh

To make the most out of the image data returned by the FLIR Lepton camera, additional image processing will be performed. Digital Image Processing is the process of modifying an image with filters, convolutions or logical statements to produce an output image containing data in a format that satisfies a range of requirements. These requirements could be to: convert the image to greyscale, where different constants are applied to the red, green and blue channels of each pixel and put in to a new, grayscale image; Remove noise from the image to improve the output of image processing algorithms; Use a convolution to detect the edges of shapes in areas of high contrast.

Images contain a wealth of data that isn't immediately accessible by a direct visual representation of the image as pixels in a 2D-array [1], hence the requirement for image processing. By applying a filter or other function to an image, extra detail can be extracted.

4.6 Parallel Manipulators v Pan Tilt units - Ethan

A parallel manipulator is able to move in 6-axis. This means they can rotate in 3-axis and move the platform in 3-axis. It achieves this by using 6 linear actuators. Linear actuators function by extending and retracting a rod to move the point attached to the end of the rod [2]. Whilst this has many benefits in larger scale use cases, this is very disadvantageous for tracking application which require a large FoR (Field of Regard).

4.7 Stepper Motors - Ethan

Stepper motors were used for this project due to their suitability for our requirements. They are dc powered motors which use a sequence of electromagnet coils to rotate a ferrous shaft one step at a time. The angle of this step depends on the orientation and spacing of the coils. An example of the coils and shaft in a stepper motor can be seen below in Figure 3.

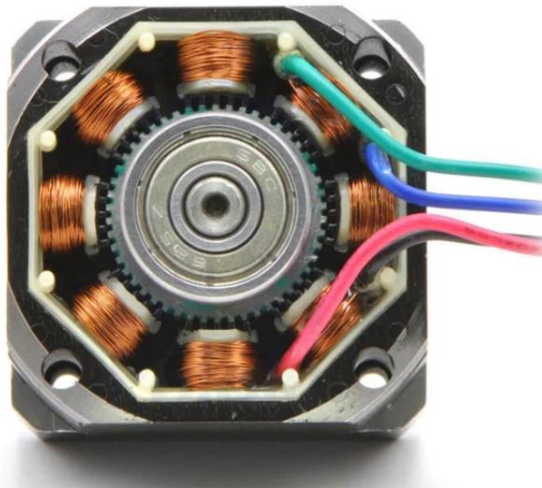


Figure 3 NEMA 11 stepper motor internal view

The ability for the stepper motor to move one defined distance at a time means there is a large amount of control for the angle the motor turns. This functionality is crucial for CNC machines such as routers or 3D printers which rely on the precise movement of a tool head to create a part with very small tolerances.

Stepper motors can achieve smooth movement through the use of microstepping, where the input signal for the stepper motor is split into several smaller steps. The chosen stepper motor can microstep from $\frac{1}{2}$ to $\frac{1}{256}$ of 1.8° . This means the stepper motors output runs a lot smoother and allows for greater accuracy in finer movements.

4.8 Stepper Motor Drivers - Ethan

A stepper motor driver takes a logic signal from a microcontroller and creates a suitable alternating signal for oscillating the coils in the stepper motor. The chosen stepper motor for this project is the TMC2208 [6]. This motor driver utilises the TMC2208 IC [7] and uses 3 logic inputs from a microcontroller to manipulate the stepper motor. These inputs are “step”, “direction” and “enable”. When the step pin is driven high, the stepper motor moves a single step. Therefore, by toggling the step pin at a certain rate the motor will rotate at that rate. The direction pin determines the direction the stepper motor will turn. The enable pin determines if the stepper motor shaft is held in position or is loose and can turn freely.

4.9 Pan Tilt – Robbie

Pan Tilt units are designed to point a camera in a three-dimensional space by rotating up and down in the y axis and rotates around a base allowing them to track in real time an object in those planes of movement.

A design [9] for a pan tilt unit from isaac879 for a DSLR used 3D printing to form a two axis of rotation model that allowed him to get an improved FoR that doesn't overlap with the base model. The design can be easily modified and due to the 3D printing it can be made at a low cost in a short period of time.

4.10 3D Printing – Ethan

3D printing is the deposition of thin layers of filament on top of one another to create a structure. This is done by heating the filament and feeding it through a nozzle that can move around in all three axis allowing for a shape to be made. There are different types of filament including PLA, ABS, PETG which all have different properties suitable for different applications.

4.11 Limit switches – Robbie

Limit switches can be used to ensure that disc cannot turn indefinitely, which could cause damage to wiring looms. I have found two software and one physical limit switch solution. The first stops the disc from rotating if an angle limit is exceeded in software. The second uses a calibrated hall-effect sensor and magnets to stop rotation when a calibrated value is met. Finally, a physical barrier ensuring that rotation is impossible in the direction of the switch could be used.

5 Development Methodology

5.1 Jack

To achieve the project brief laid out in the SoR we intended to use a single board computer and IR camera to capture and process the images and a PTU to perform the tracking operations. A single board computer was chosen to perform the image processing as it is a powerful and compact method of computing allowing it to be mounted directly into the final product, it also ensures the user can set up and control the system in the prototyping stages of the product development. We decided on IR imaging as this is an easy method of Electro-Optical tracking as it can be done through pixel contrast unlike visible light cameras which require more complex processes to track objects. IR was the preferred imaging method as UV imagers are difficult to source and typically have stipulations such as being unable to be used outside.

The PTU was 3D printed and controlled by stepper motors to allow for ease of manufacture and precise control for positioning the camera to point at the target. 3D printing allowed for rapid prototyping and manufacture, the PTU was produced and modified at speed and with ease. The material costs were also reduced as additive manufacture is generally cheaper than traditional manufacturing methods.

The FLIR Lepton 3.5 camera was chosen as it is small and lightweight, making it easy to mount to the PTU. It also had a breakout board, allowing us to use only the desired features of the camera to reduce complexity and processing power needed. It has a lower resolution than most other cameras we were looking at, allowing us to reduce the amount of processing required, while still meeting the demands of the SoR

5.2 Algorithm Development – Josh

MATLAB has been chosen as the software package to design and verify the behaviour of image processing algorithms using a set of test images and videos. Testing image processing algorithms in a software based solution instead of experimenting in hardware allows for a wider range of scenarios to be presented to the algorithms than would be seen in hardware. It also allows for raw data captured from hardware testing to be replayed over and over, changing the algorithms processing it to compare performance and characteristics of subtle changes.

The software running on the SBC will be written in C++ as the libraries available to display images on a screen are available in C++. Compared to an interpreted language like Python, a compiled language such as C++ offers greater performance and greater control over hardware peripheral devices, with access to CPU register and ports.

6 Design

6.1 Overall –Jack

The design requirements set out in the SoR were for this work package to produce a MSTR capable of tracking a source in both IR and UV bands, with the capability to train the boresight onto a target through the use of a rate table or similar method of aiming the camera to the target.

6.2 PTU Design – Robbie

The pan tilt unit would be modelled off of isaac879 design. With limited time and access to a 3d printer it was though best to take his design and modify it to our own needs. The plan is to use PETG as the filament for the print because it has a high strength and durability allowing for a better print when parts are moving. The modifications to the design are as follows to make the PTU more efficient:

1. A longer base to allow for greater space tot fit the SBC and PSU PCB using standoffs.
2. Control commands coming from the Image Processing Board instead of being operator controlled.
3. The IR camera would be mounted on top, facing towards the user using a C bracket so the camera is pointing at the object it wants to track.
4. Limit switches will be used to: Calibrate to the origin point this is used every time it would be turned on; restrict the PTU from performing a full 360 degree rotation.

6.3 PTU Hardware – Ethan

Whilst the mechanical design for the PTU was an existing design posted to Thingiverse [10], the rest of the hardware and firmware controlling the PTU was designed from scratch. The motor controller is a Raspberry Pi Pico with a Micropython interpreter installed using the Thonny IDE. Micropython is a subset of the python programming language used on microcontrollers that have less available device memory [11]. Thonny is used to create firmware and flash it onto the motor controller. This allows for rapid prototyping of firmware during development [12].

The Pico was used to send a digital signal to the motor drivers and manipulate the PTU head whilst also taking in commands from the user and tracking demand from the camera control system.

Each motor driver had 3 digital logic input which are connected to the GPIO pins of the Pico. The drivers' logic circuits are powered using a 3.3V output from the Pico. The stepper motors themselves are connected to the drivers and are powered using a 12V regulator on the power supply board. A diagram of these connections can be seen below in Figure 4.

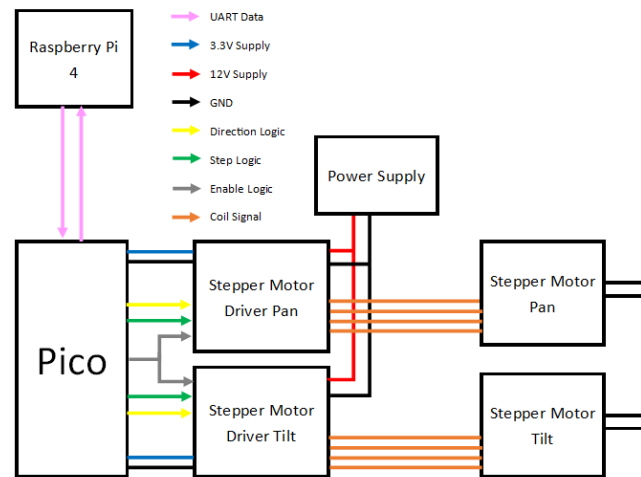


Figure 4 PTU hardware connections

6.4 Algorithm Design – Josh

The image processing board uses a simplified algorithm to ignore all pixel values below a certain threshold, this presents a final image with less clutter in the background. The algorithm checks if the pixel is below a threshold and sets it to zero.

The main body of the program running on the image processing board that is responsible for displaying the processed images on the display attached is the function void

LeptonThread::run() which opens the SPI port with a 20MHz clock on the raspberry pi to read in the SPI packets to a buffer. These packets are parsed and converted to a grayscale image with values ranging from 0-255.

Once the entire image has been read in as a grayscale image, a cut-off is placed at all values below 239, (Max value 255 – 16) to display only the brightest pixels to the screen. Then the brightest pixel is located by searching through every pixel and updating location variables if the currently selected pixel is greater in intensity than the previously selected pixel.

7 Implementation

7.1 3D Printing – Ethan

The model was planned to be printed in PETG because of its higher strength and stiffness, coming from increased adhesion between layers from the higher print temperature, for moving parts [13].

This was the first time I attempted to print in PETG and the test prints and calibration with failed. This could be as a result of: incorrect extruder or bed temperature causing too little layer adhesion; too much airflow cooling the filament too rapidly. We decided to print the parts in PLA which we have sufficient experience in using. PLA is more brittle than PETG but much easier to print with. Meaning the parts printed in PLA will be more likely to break but are accurately printed and able to fit together for assembly.

The pan tilt unit was mostly printed with grey eSun PLA+ and used 406g of filament over a total print time of 36 hours. This comes to a total material cost of £8.

7.2 Calibration – Robbie

The system needs to be calibrated each time it is restarted to give itself an origin. Photoresistors and limit switches were both considered but magnets and hall-effect sensors were used instead. When the hall-effect sensors read a maximum value, each of the axis is at its respective zero point. These were selected as they can also function as a soft limit switch when the U-mount is upside down.

7.3 Standoffs – Robbie

I mounted the SBC and PCB to the PTU with standoffs to the base of the U-mount. The SBC was mounted in the inside of the U-mount with the PCB fixed to the reverse side. Standoffs were used in areas that had been drilled. The PCB and SBC were mounted in different orientations to accommodate for the U-mount being used upside down to offer a greater range of motion.

7.4 Circuit – Jack

I made a circuit board to power the motors and send signals between the PTU, camera, and SBC, this is necessary to provide a single, central point for all of the peripherals and the computer to simplify the product and reduce the distance of and amount of wires running between the peripherals and the computer. I decided to skip prototyping on a veroboard as the signal lines had been tested directly between the SBC and test equipment, and I used application circuits for each of the voltage regulators, recommended by the manufacturer of the ICs.

7.5 Procurement – Jack

We attempted to procure the IR camera through appropriate work processes, this however did not work out as the procurement team took too long to react to our order, during the 4 weeks it took them to place the order for the camera I had contacted the procurement team – to check where the order was in the process – and then bought and received the same camera within the week.

The camera software had been written for raspberry pi so we found it difficult to set up for the SBC as such we swapped out the Rock for the Raspberry Pi for greater compatibility.

7.6 Software – Josh

I adapted stable open source software [14] to add image processing capabilities and UART communication the program.

An oscilloscope was used to verify the clock on one of the pins coming out from the FLIR Lepton camera before connecting it to the image processing board. Once this had been confirmed at 25MHz, the camera was connected to the image processing board and the SPI lines were probed to confirm that data being sent over the MISO line.



Figure 5 An image of an oscilloscope probing a 25MHz signal

Integrating the IR camera with the Rock board presented greater difficulty than originally anticipated. This is a result of not examining the complexity of hardware compatibility between the FLIR Lepton camera and the SBC.

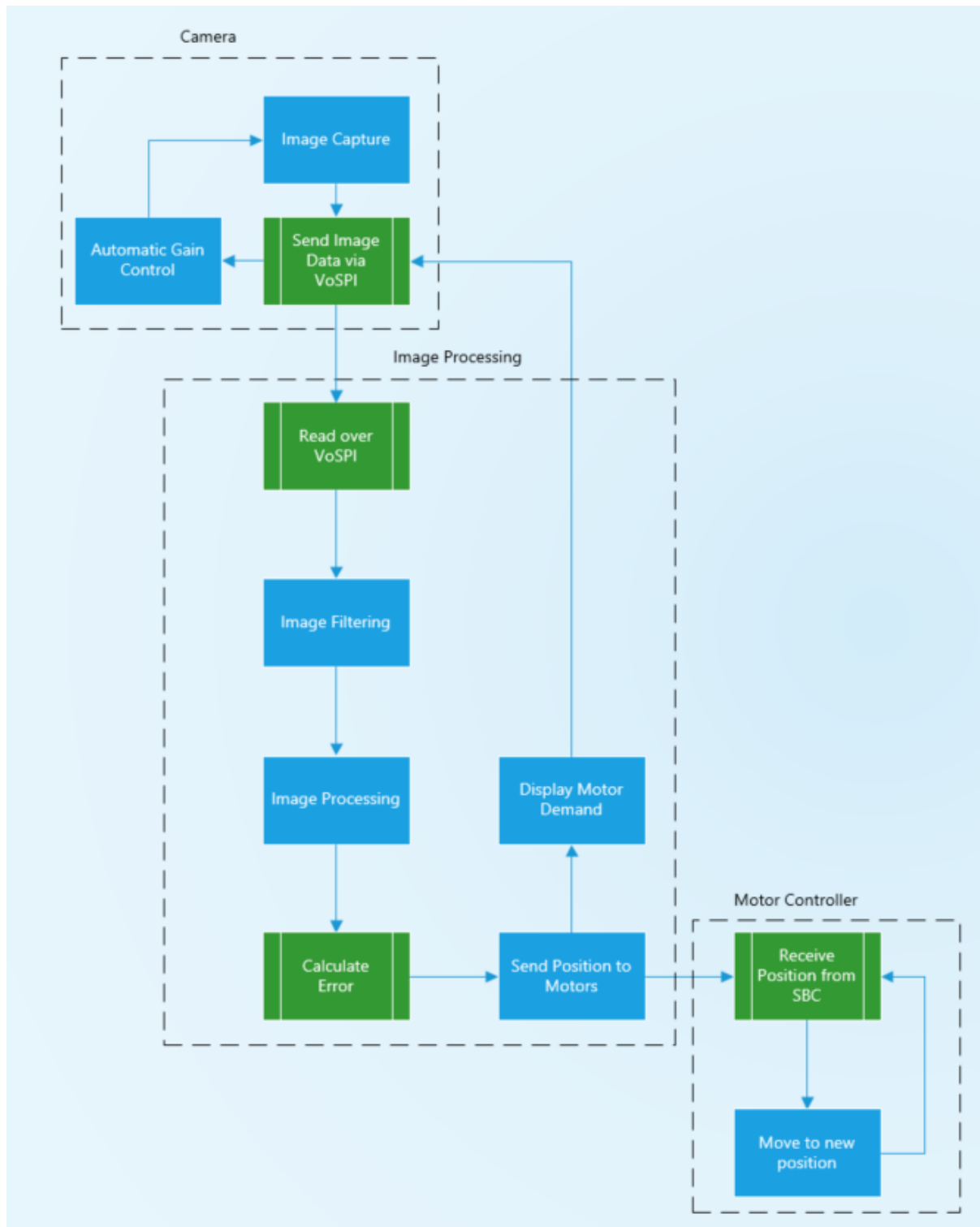


Figure 6 Software flow chart

7.7 Firmware - Ethan

The function of the firmware is to take the error between the boresight and the target sent from the Raspberry Pi 4 via UART and move the PTU so the boresight is over the target. The data comes in from the Pi4 in the form of an X and Y component in pixels.

Both axis were moved at the same time by separating the total distance into smaller chunks and moving in each axis separately to create a smooth rotation. An example is illustrated below in Figure 7.

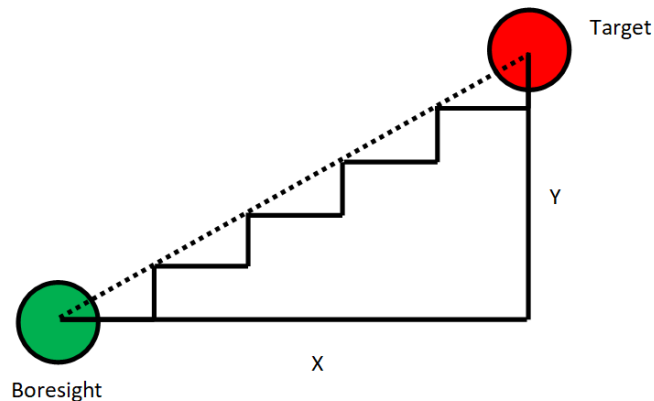


Figure 7 Demand chunking illustration

The value of how many chunks was chosen during testing as it was found that too high a value caused too much vibration and caused errors where one chunk would have a demand less than one step and too low of a value would be too rough for tracking.

For each chunk of the demand, a number of steps each axis is to move in is generated and this is looped over for the total number of chunks. The function can be seen below in Figure 8, including software limit switches.

```
#-----Move Pan & Tilt at same time function-----
#Divides both movements into x chunks. Alternates moving each axis by a chunk until full distance travelled
def both_moveDegrees(pan_move2, tilt_move2):
    if -180 > (pan_location+pan_move2) < 180:
        pan_move2 = 0
    if -180 > (tilt_location+tilt_move2) < 180:
        tilt_move2 = 0

    if pan_move2 and tilt_move2 >= 45:
        chunks = 500
    else:
        chunks = 30

    pan_steps2 = pan_moveDegrees(pan_move2)
    pan_share = pan_steps2/chunks

    tilt_steps2 = tilt_moveDegrees(tilt_move2)
    tilt_share = tilt_steps2/chunks

    if debug: print("Debug: pan_share:%s steps:%s tilt_share:%s steps:%s" % (pan_share,pan_steps2,tilt_share,tilt_steps2))
    for i in range(chunks):
        pan_xSteps(pan_share)
        tilt_xSteps(tilt_share)
```

Figure 8 Function to move both axis at same time

8 Results and Analysis

8.1 Testing write-up – Robbie

Testing the rotation speed of the pan tilt unit would determine what speed an object would have to go, to get out of line of site of the camera. To find out what speed the pan tilt unit would have to rotate at, the average speed and height of a drone would be used. A drone moving at 20m/s laterally across the horizon of the FoV (Field of View) of the camera at a height of 120m would move at 9.5 degrees per second, as calculated below:

$$\tan^{-1}(20\text{ms}^{-1}/120\text{m}) = 9.5^{\circ}/\text{sec}$$

To test the speed of the pan tilt unit a photoresistor would be used. When it detects light a timer would start and stop when dark again. Using a stopwatch, the PTU completed a 180 rotation in 7.5 seconds giving it a rotation speed of 24 degrees per second. This is 2.5x faster than required to follow a drone. The test was done again but for the Y axis. The pan tilt unit took 10 seconds to do a full 360 this give a rotation speed of 36 degrees per second, which is more than required to track a moving drone.

8.2 Tracking reliability – Josh

When tracking small objects much brighter than their surroundings, the tracking performed exceedingly well and sent out over UART an accurate and precise pixel location of the target. However for larger objects that took up a significant portion of the FoV, such as a hand close to the lens of the camera, the target location became erratic. This is because there were multiple points on the target at roughly the same temperature, and this caused the target point to jump to a different location in each frame.

9 Evaluation

9.1 Jack

Our statement of requirement was to produce a multi-spectral tracking rig comprising of a tracking algorithm and a pan and tilt unit to aim the camera. The intention was to produce the MSTR capability in a cost effective and time efficient manner. We were able to produce the MSTR in around 3 months for an approximate total cost of £13,024 – £664 material costs and £12,360 man hours – this works out to be cheaper than some COTS PTUs available to buy and works out to be cheaper than procuring a COTS PTU from industry and developing the system around that. Cheaper PTUs can be purchased at a lower cost than industry costs, however it would be less capable than both the industry PTU and our own PTU we printed.

COTS PTUs can be purchased for approximately £10,000+ however these are supplied with user interfaces and sometimes include inertial stabilisation and other high level capabilities which add cost to the unit. In contrast, cheaper COTS PTUs would not have the capabilities required and may not support the communication interfaces we require.

We were unable to conduct many tests on the Tracking Rig due to rushed production, which resulted in only being basic functionality tests of the PTU and the camera, while more complex tests involving the camera directing the PTU were performed at a later date. From these tests we were able to conclude that the PTU operated as intended and expected, it proved to have greater rate capability than the result expected.

The camera tests were conducted to establish the output from the camera, this concluded that we would receive a standard maximum value from the camera due to the AGC. This ensured seamless demands to the PTU despite the temperature of the hottest source in the FoV. We were unable to perform conclusive test on the MSTR because the PTU was not produced, and the camera feed was first passed through the algorithm during the final project week. As such we can say the MSTR should be able to track a heat source in its FoV and the PTU should correctly train the camera boresight onto the heat source.

If we were to produce this next time we would need to produce the PTU much earlier as this can then remain inert but available until it is ready for integration with the SBC and the camera, this would have allowed us to focus on aspects of the project such as the tracking algorithm and the motor control.

10 Conclusions and Recommendations

10.1 Jack

Throughout this project we have worked to understand PTU design, control, object detection, and image processing. From our research, a design for the MSTR was developed which laid out methods of detecting and tracking objects in the FoV of the camera and how the boresight of the camera would then be trained onto the object, the output from the design stage was a set of image processing methods which could be used to identify and maintain track on an object; and a PTU as the method of training the camera onto the target.

The implementation of these designs was mostly successful with the camera capable of identifying where the object of interest was in the FoV and translating it into a set of coordinates to be read by the PTU, we were also able to print a PTU that was capable of performing demands given by the user. At the time of writing this report a test was conducted to determine if the error signal from the camera could drive the PTU as expected, consistently, this test however was inconclusive as of July 6th the PTU did receive and interpret the error signal demands but did not move in a predictable manner and was unable to reproduce the same movements across several repeated tests.

Despite the tests indicating that the MSTR does not function correctly at present, given time this can be resolved. As such we have managed to successfully achieve our aim of producing a tracking capability in a cost effective and time efficient manner as this product to date costs £664 and took approximately 3 months to produce.

My contributions to this project included securing workspace for us to carry out project work, booking meetings to discuss the progress made and next steps. I also designed and populated the communications circuit board that hosted all of the data flow and power required to operate the MSTR, I also wrote elements of the report such as the introduction, evaluation and Conclusion.

In future iterations of this product there would be potentiometers or an arrow pad on the PTU to allow for manual control, and a dedicated payload mount. I also designed in the capability to send data to display screens, which could be utilised. The PTU printed is capable of performing 360° rotations in both pan and tilt axis, we set limits for 180° range of motion as we had not been able to implement the slip ring into the PTU, if the slip ring was implemented the PTU could rotate through 360° in both axis.

To allow this unit to become multi-spectral we would need to implement a UV imager, we could use the existing code with a different set of rules to track a source in the UV band. To further improve the imaging capabilities of the MSTR we could merge the IR and UV images received from each camera with a visible light camera to give a single output capable of tracking across multiple bands of the EM Spectrum using contrast and edge tracking capability.

10.2 Robbie

Although the PTU is functional, a few improvements could be made. For example a more custom U mount could have been made. To not only mount everything more comfortably, but also allow for slip rings to be used so the u mount could do a full 360 without any cables wrapping round the mount. Other features that could have been added include limit switches, so it can't look down and track its self. One last improvement, the PTU could have been printed in PETG giving more durability and strength.

10.3 Josh

This project did not meet the requirement of adding multispectral tracking, this is work that is recommended to be carried forward. Implementing this functionality will involve further image processing and

UV multispectral implementation, combining and scaling images, bias towards one or another

11 Reflection

11.1 Ethan

My part in this project was to design and build the electrical hardware of the PTU and the firmware on the Raspberry Pi Pico microcontroller running it. Robbie found an appropriate mechanical design for the PTU and purposefully avoided looking at their firmware so I could come up with my own implementation. This led to lots of testing with the microcontroller and stepper motors to figure out how we could operate the PTU.

I had access to a 3D printer with a large print area and printed the PTU parts. Most parts were printed in PETG but due to calibration errors later on PETG parts were failing to print correctly and PLA was used instead. I assembled and integrated the printed parts with the electrical hardware I had constructed separately.

Unfortunately, this project was very rushed and if I was able to spread the research and testing of the hardware over a longer period of time rather than a couple weeks I would have been able to foresee some of the challenges which arose and come up with a better way of doing controlling the stepper motors in the firmware as a colleague pointed out once I was finished.

11.2 Jack

My contribution to the group project consisted of meeting and workspace bookings as well as the communication circuit board. I also worked on the camera tracking aspects of the project. I worked to develop the tracking logic and motor control signals that aimed the camera. The circuit board was produced in time for the testing phase of the project, allowing me to contribute something to the project outside of tracking code and PTU development which meant I wasn't delayed by any work that was not completed and no other work was dependant on this portion of the project.

The project would have benefited from more regular meetings, having a dedicated time slot in our calendars which would have enabled us to keep track of the work flow and work with greater efficiency through the project timeline. I spent my time balancing my project contributions against other commitments while I was waiting for other deliverables to be produced, I could have used this time to better manage the project as a whole and provide the leadership that was needed. This would have sped up the process of procuring the camera and the printing of the PTU parts.

Issues arose from the lack of leadership, and no continuous progression as a result. The communication between each group member was poor and in some instances nothing was communicated. It became apparent that one of us should have been assigned the role of leading the group project, and a project manager role should have been assigned as well; this would have helped to keep track of workloads and progression throughout the project.

11.3 Joshua

My contribution consisted of integrating the FLIP Lepton IR camera with the Raspberry Pi and motor controller by writing image searching algorithms and UART TX functions for the Raspberry Pi, UART TX functionality for the Pico, and editing this report. This included taking raw images from the IR camera and applying a simple image filter over them, enhancing the object of interest against the background.

Throughout development, there should have been more frequent isolated and integrated testing of systems to ensure compatibility, both with software and hardware, to reduce the burden later along the timeline where most testing was done all at once.

Integrating the image processing board with the motor controller was difficult as it had to be done towards the end of the project at the same time as the report was being written as the motor controller board didn't have UART receiving functionality at the start of the testing phase. A key lesson learned here is to establish communication between each member of the team to adjust deadlines to account for sub-deliverables throughout the project that are a bottleneck to further sub-deliverables. In this instance, the motors should have been tested with the motor controller board and the communication between the two tested before the PTU had been printed.

The issues mentioned above could be resolved by assigning a project manager with responsibility to ensure the delivery of the project by conducting regular check-ins with team members and identifying any shortfalls in time, resources or delivery between the current and expected state of the project.

11.4 Robbie

My contribution to the project was finding a suitable PTU to use and then modifying it so the camera and PCB's could mount and be functional. This meant using standoffs, drilling holes and making everything secure without anything colliding whilst the PTU was rotating.

Issues with the time line of the delivery of work made it difficult to add extra features that although aren't strictly necessary for the PTU to work they would have been nice to have, for example if slip rings were used and wiring loom was implemented the PTU would be more stable and less likely to break after repeated use.

The issues of the time in which work was delivered was due to a lack of management from us, as a group leader or manager wasn't set, leading to tasks being delayed due to other commitments. In addition one day a week dedicated to this would have helped us move along faster and get objectives done in a timely manner, this would have given us more time to write the report as everything was pushed to the last week of the project.

12 References

- [1] 'Radiation: Ultraviolet (UV) radiation'. [https://www.who.int/news-room/questions-and-answers/item/radiation-ultraviolet-\(uv\)](https://www.who.int/news-room/questions-and-answers/item/radiation-ultraviolet-(uv)) (accessed Jun. 12, 2023).
- [2] 'UV Light - Stanford Solar Center'. <http://solar-center.stanford.edu/about/uvlight.html> (accessed Jun. 12, 2023).
- [3] 'Canon : Canon Technology | Canon Science Lab | CMOS Sensors', *Canon Global*. https://global.canon/en/technology/s_lab/light/003/05.html (accessed Jun. 12, 2023).
- [4] 'Introduction to CMOS Image Sensors - Technical Articles'. <https://www.allaboutcircuits.com/technical-articles/introduction-to-cmos-image-sensors/> (accessed Jun. 12, 2023).
- [5] gd-admin, 'What is NIR/SWIR/MWIR/LWIR/FIR Spectral Range?', <https://www.viewsheen.com/>. <https://www.viewsheen.com/news/nir-swir-spectrum/> (accessed Jun. 12, 2023).
- [6] 'Ranges of Infrared Radiation', *Xenics*. <https://www.xenics.com/infrared-technologies/> (accessed Jun. 12, 2023).
- [7] 'Microbolometer | Infiniti Electro-Optics'. <https://www.infinitioptics.com/glossary/microbolometer> (accessed Jun. 12, 2023).
- [8] 'Microbolometer', *Wikipedia*. May 31, 2023. Accessed: Jun. 12, 2023. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Microbolometer&oldid=1157843316>
- [9] 'RK3588 Brief Datasheet.pdf'. Accessed: Jun. 12, 2023. [Online]. Available: <https://www.rock-chips.com/uploads/pdf/2022.8.26/192/RK3588%20Brief%20Datasheet.pdf>
- [10] 'Rock5/hardware/5b - Radxa Wiki'. <https://wiki.radxa.com/Rock5/hardware/5b> (accessed Jun. 12, 2023).
- [11] 'Choosing the Right Thermal Imager for Your Integrated Project', 1AD. <https://www.flir.com/discover/cores-components/choosing-the-right-thermal-imager-for-your-integrated-project/> (accessed Jun. 12, 2023).
- [12] '500-0771-01', *Digi-Key Electronics*. <https://www.digikey.co.uk/en/products/detail/flir-lepton/500-0771-01/7606616> (accessed Jun. 12, 2023).
- [13] 'Lepton LWIR Micro Thermal Camera Module | Teledyne FLIR'. <https://www.flir.co.uk/products/lepton?vertical=microcam&segment=oem> (accessed Jun. 12, 2023).

13 Bibliography

- [1] G. Anbarjafari, "1. Introduction to Digital Image Processing | Digital Image Processing," University of Tartu, January 2014. [Online]. Available: <https://sisu.ut.ee/imageprocessing/book/1>. [Accessed 15 06 2023].
- [2] Progressive Automations, "How do motion simulators and linear actuators work together," [Online]. Available: <https://www.progressiveautomations.com/blogs/how-to/how-do-motion-simulators-and-linear-actuators-work-together>.
- [3] PiHut, "NEMA 11 Bipolar Stepper Motor Image," [Online]. Available: <https://thepihut.com/products/nema-11-bipolar-stepper-motor-200-steps-rev-3-8v-0-67-a-phase>.
- [4] Oozenest, "NEMA 17 Stepper Motors," [Online]. Available: <https://ooznest.co.uk/product/nema17-stepper-motors/>.
- [5] telecointercon, "145 NEMA motor sizes, what does it mean?," [Online]. Available: <https://www.telcointercon.com/blog/145-nema-motor-sizes-what-does-it-mean>.
- [6] Oozenest, "TMC2208 SilentStepStick Stepper Motor Driver," [Online]. Available: <https://ooznest.co.uk/product/tmc2208-silentstepstick-stepper-motor-driver/>.
- [7] Trinamic, "TMC2208," [Online]. Available: <https://www.trinamic.com/products/integrated-circuits/details/tmc2208-la/>.
- [8] Ring, "Pan-Tilt Mount For Ring Stick Up Cam," Ring, [Online]. Available: <https://en-uk.ring.com/products/pan-tilt-mount>. [Accessed 12 06 2023].
- [9] isaac789, "3D Printed DSLR Camera Pan Tilt Mount (Arduino/Stepper Driven) 2020," 23 May 2020. [Online]. Available: <https://youtu.be/uJO7mv4-OPY>. [Accessed 12 June 2023].
- [10] isaac879, "DSLR Camera Pan Tilt Mount (Stepper Motor Driven)," [Online]. Available: <https://www.thingiverse.com/thing:4316563>.
- [11] Micropython, "Micropython," [Online]. Available: <https://micropython.org/>.
- [12] Thonny, "Thonny Homepage," [Online]. Available: <https://thonny.org/>.
- [13] "PETG vs PLA: The Main Differences | All3DP," All3DP, [Online]. Available: <https://all3dp.com/2/petg-vs-pla-3d-printing-filaments-compared/>. [Accessed 2023 06 12].
- [14] M. Narvaro, "Code for getting started with the FLIR Lepton breakout board," [Online]. Available: https://github.com/groupgets/LeptonModule/tree/master/software/raspberrypi_video. [Accessed 17 06 2023].
- [15] J. Roith, "Johannes' personal website," [Online]. Available: <https://gnu-linux.ocks/>. [Accessed 17 06 2023].

- [16] isaac879, “DSLR Camera Pan Tilt Mount (Stepper Motor Driven),” 23 May 2020. [Online].
Available: <https://www.thingiverse.com/thing:4316563>. [Accessed 12 June 2023].

14 Glossary

Item

This is the description of the glossary item.

15 List of abbreviations

ABS	Acrylonitrile Butadiene Styrene
COTS	Commercial Off The Shelf
FLIR	Forward Looking Infrared
FoR	Field of Regard
FoV	Field of View
GCC	Gnu Compiler Collection
GDB	Gnu Debugger
IC	Integrated Circuit
IDE	Integrated Development Environment
LWIR	Long-wave Infrared
MSTR	Multi Spectral Tracking Rig
MWIR	Mid-wave Infrared
NEMA	National Electrical Manufacturers Association
PCB	Printed Circuit Board
PETG	Polyethylene Terephthalate Glycol
PLA	Polylactic Acid
PSU	Power Supply Unit
PTU	Pan-Tilt Unit
SBC	Single Board Computer
SWIR	Short-wave Infrared
TEG	Threat Evaluation Group
UV	Ultraviolet

16 List of symbols

ξ Xi

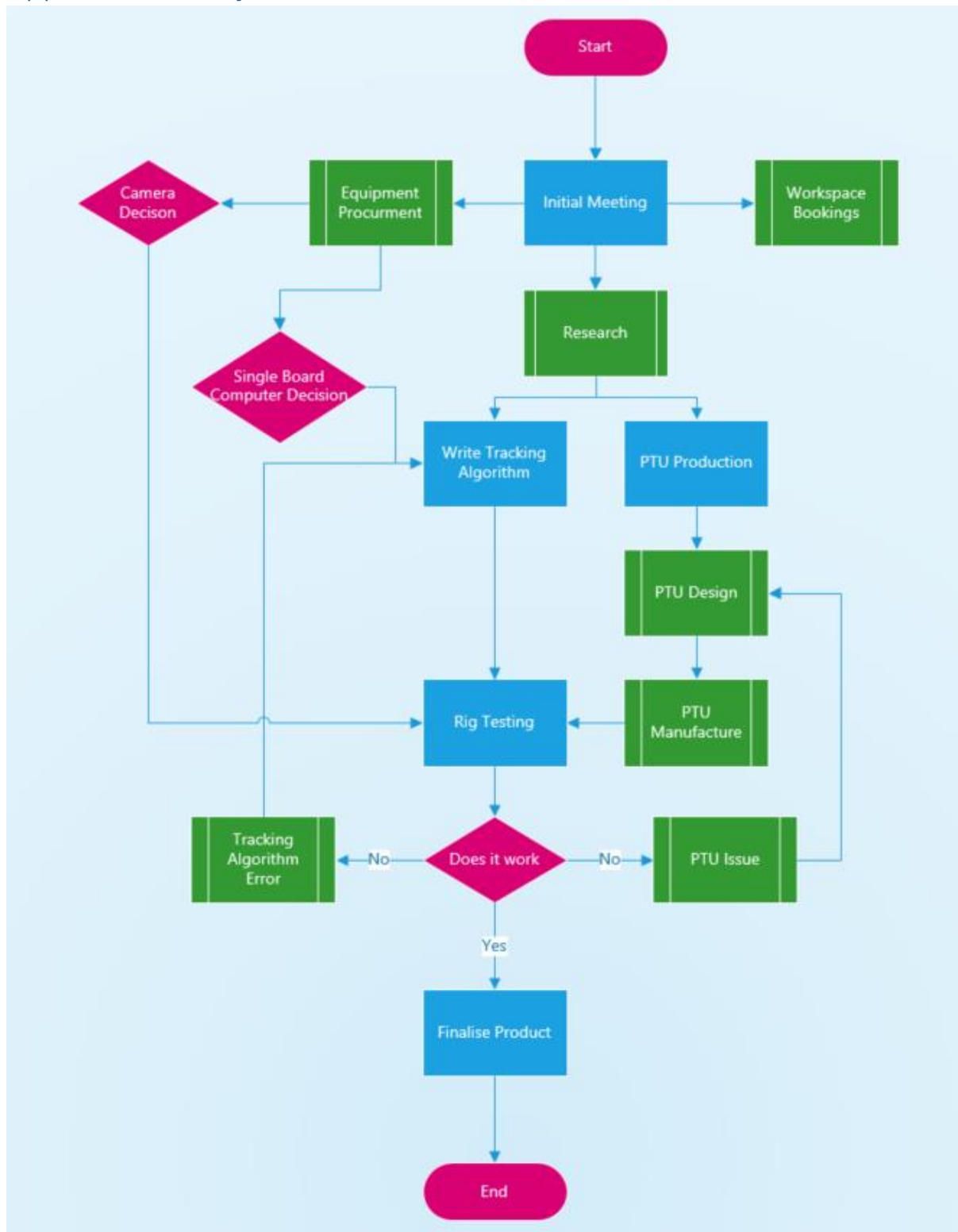
Appendix 1 – UV Radiation

When UV radiates through a vacuum, a different set of bands are used; loosely following a similar trend.

- Near Ultraviolet (NUV) Light (300-400nm)
- Middle Ultraviolet (MUV) Light (200-300nm)
- Far Ultraviolet (FUV) Light (100-200nm)
- Extreme Ultraviolet (EUV) Light (10-100nm)

EUV can only travel through a vacuum and is completely absorbed by the atmosphere because it ionizes the upper atmosphere, creating the ionosphere. Earth's thermosphere is heated mainly by EUV waves from the Sun.^[2]

Appendix 2 – Project Plan – Jack



17 Annex 1 – Motor controller functionality – Ethan

The demand reading and conditioning function waits until data is being received on the UART pin and reads a packet lengths worth of data into a buffer. The data is received in byte format so must first be decoded into a string for conditioning. Before any information is taken from the packet it is first checked that the first character of the packet is to be expected, this prevents false information from being acted on and causing errors in tracking.

The packet is received in the format of 'x---,y---' so it is split at the comma into a x demand and y demand. The respective demand is then multiplied by the degrees per pixel and if that demand is more than 2 degrees the mount is moved to meet the demand. The demand must be more than 2 degrees to prevent jittering around a point.

```
elif command == "3":
    tracking = True
    #Read information from RX pin and decode it into a string. Split the string in to x and y demand
    #Packet format is x---,y---
    #Demand is received as pixels and is converted using 0.35625 degrees per pixel calculated from resolution and FoV of camera.
    #The mount is only moved if the tracking demand is greater than 2 degrees
    while tracking:
        print("""Tracking
        Current Location :%s°, %s°
        """, pan_location, tilt_location)
        #uart.write("x%3d,y%3d"%(30,-40))
        utime.sleep(0.1)
        if uart.any():
            buffer = uart.read(9)
            buffer = buffer.decode('utf-8')
            print("buffer =", buffer)
            uart.write('recv\r\n')
            if buffer[0] == 'x':
                pan_move, tilt_move = buffer.split(",")
                pan_move = float(pan_move[1:])*0.3526
                tilt_move = float(tilt_move[1:])*0.3526

                if fabs(pan_move) and fabs(tilt_move) >= 2:
                    if debug: print("Pan demand: %s , Tilt demand: %s" %(pan_move, tilt_move))
                    both_moveDegrees(pan_move, tilt_move)
```

Figure 9 Tracking demand reading and conditioning

By taking FoV of the camera and its resolution, it was calculated that one pixel is equal to 0.35625 degrees. When this value is multiplied by the amount of pixels, an error in degrees is obtained. The Micropython function for this part of the firmware can be seen in figure 3 below.

By changing the delay between each round of driving the motor high and low, the speed of the motor can be controlled.

In order to control how far the motor rotates it is as simple as only sending a certain amount of step commands. The correct amount of steps can be calculated using the degrees per step of the motor, microstepping setting of the driver and gear ratio of the mount. For this system, the degrees per step of the motor is 1.8°; the microstepping is set to 1/8 and the gear ratio for the mount is 144:17 for pan and 64:21 for tilt. Therefore, $(1.8 \div 8) \div (144 \div 17) = 0.0265^\circ$ per step for pan and $(1.8 \div 8) \div (64 \div 21) = 0.0738^\circ$ per step for tilt.

18 Annex 2 – Motor controller functionality 2 – Ethan

The moveDegrees function is the function which is used to convert the demand in degrees into a demand in steps. This function can be seen in figure 6.

```
#Sets direction of the x-axis rotation and calculates how many steps to achieve a certain degree turn
def pan_moveDegrees(pan_move2):
    global pan_location
    pan_delta = pan_move2 - pan_location
    if pan_delta > 0:
        pan_dir.high()
    if pan_delta < 0:
        pan_delta = -pan_delta
        pan_dir.low()
    #degrees to steps - 0.0266 degrees per step
    pan_steps = pan_delta / 0.0266
    return pan_steps
    #pan_xSteps(pan_steps)
```

Figure 10 moveDegrees function

This function is a set of two, one for pan and one for tilt. Each function has the corresponding degrees per step for the axis. This function could be further simplified into one function with a further input defining which axis to be used. The current function takes into the demand in degrees and sets the direction of the motor based on whether it is negative or not. It then simply divides the demand by the amount of degrees per step to calculate the amount of steps and returns it.

The xSteps functions is the function which takes the steps demand and oscillates the step signal the required amount of times to rotate the stepper motors by the required angle. The function can be seen in figure 7.

```
#Calls OneStep function a certain amount of times depending on how many degrees to turn
def pan_xSteps(pan_steps3):
    pan_report = 0
    #print location every 100 steps
    for i in range(pan_steps3):
        pan_report += 1
        if pan_report == 100:
            pan_report = 0
            print("Pan at "+str(round(pan_location,0))+")")
        pan_OneStep()
```

Figure 11 xSteps function

The function also includes a report variable which iterates each time a step is made. Every 100 steps this variable then activates a if statement and prints out the current location of the axis. This is so the user can track the movement of the axis without being spammed every step.

This function calls the oneStep function for every step. This function contains the commands to toggle the step pin and also iterates the location global variable every step so the location of the mount is accurately tracked without using feedback.

19 Annex 3 – Alternative PTU Designs – Robbie#

A simple pan tilt unit design comes from Ring [8], controlled via a mobile application and rotating around a central point. This is impractical for our application.

20 Annex 4 – Split Rings - Robbie

Split rings are used so a disc can rotate around without a wiring loom, made up of two components, the first is either a metal tape or a metallic ring. The second is a brush which goes around connecting to the ring. It allows for to data and power to be transferred without a wire demonstrated below in .

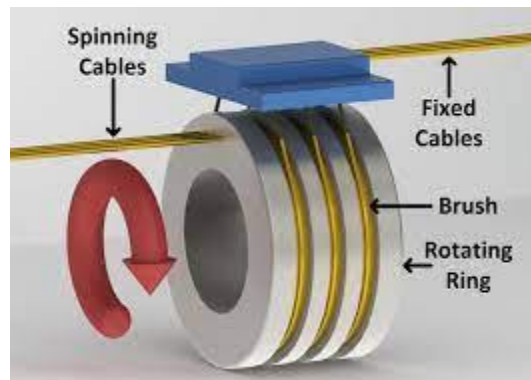


Figure 12 Slip Ring

Although research and development plans went into slip rings there wasn't enough time to implement this feature as it isn't essential for the PTU to function, as 360 degrees of rotation isn't a key feature of the U mount. To implement the slip rings a custom U mount would have had to been drawn up allowing for the modification, however due to them being STL files and the limit time it was deemed not necessary to do.

21 Annex 5 – Edge Detection

Edge detection is a type of post-processing effect which highlights regions of an image where there is contrast in pixels in an area. An example is shown below in using a "Sobel operator", which is comprised of two 3x3 kernels as follows:

Equation 1 Horizontal Sobel Operator

$$\begin{matrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{matrix}$$

Equation 2 Vertical Sobel Operator

$$\begin{matrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

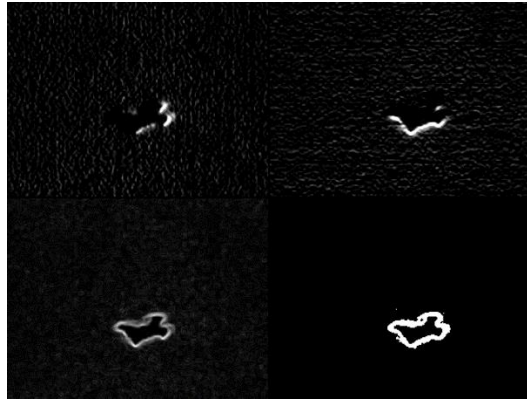


Figure 13 Sobel operator and threshold applied to an infrared image of a drone

22 Annex 6 – Micropython Source Code – Ethan

```
from machine import Pin,UART
import utime
from math import fabs

#Set up pico pins
pan_dir = Pin(15, Pin.OUT)
pan_step = Pin(14, Pin.OUT)
motor_en = Pin(16, Pin.OUT)
tilt_dir = Pin(9, Pin.OUT)
tilt_step = Pin(8, Pin.OUT)

#Initialise UART for receiving data from camera control
uart = UART(1, baudrate=9600, tx=Pin(4), rx = Pin(5))
uart.init(bits = 8, parity = None, stop = 1)

#Initialise pico pins
motor_en.high()
pan_dir.high()
tilt_dir.high()

#motor_te.high()

#pan location relative to middle of FOV in Degrees. One motor step is 1.8
degrees, motor driver microstepping means 1 step is 0.225 degrees
#for pan axis, gear ratio is 144:17 so 1 step is 0.0266 degrees
#for tilt axis, gear ratio is 64:21 so 1 step is 0.0775 degrees
pan_location = 0
tilt_location = 0

#motor speed percentage multiplier, 0.001 seconds per step is 100%
speed = 100
delay = 0.001

#Toggles debug outputs from certain actions
debug = False
```

```
#Tracks movement of the motors to store their location for movement to be
accurate
def pan_trackLocation():
    global pan_location
    if pan_dir.value() == 1:
        pan_location += 0.0266
    else:
        pan_location -= 0.0266
    #if debug: print(pan_dir.value(), pan_location)

def tilt_trackLocation():
    global tilt_location
    if tilt_dir.value() == 1:
        tilt_location += 0.0775
    else:
        tilt_location -= 0.0775
    #if debug: print(tilt_dir.value(), tilt_location)

#Toggles stepper motors allowing them to move freely or not
def motor_Toggle():
    motor_en.toggle()
    print(motor_en.value())
    #if motor_en.value() == 1:
    #    motor_en.low()
    #    # print("Stepper Motors Enabled")
    #else:
    #    motor_en.high()
    #    # print("Stepper Motors Disabled")

#Sets the speed to a certain percentage based on the time delay between
step commands
def setSpeed():
    global speed
    global delay
    # 100% is 0.001 seconds per step
    delay = 0.001 / (speed / 100)

#Sets motor control pins to low
def motor_Reset():
    pan_dir.low()
    pan_step.low()
    tilt_dir.low()
    tilt_step.low()
```

```
#-----Pan Functions-----
-----
#Move x-axis by 1 step - Used by all other pan functions
def pan_OneStep():
    pan_trackLocation()

    pan_step.high()
    utime.sleep(delay)
    pan_step.low()

#Calls OneStep function a certain amount of times depending on how many
degrees to turn
def pan_xSteps(pan_steps3):
    pan_report = 0
    #print location every 100 steps
    for i in range(pan_steps3):
        pan_report += 1
        if pan_report == 100:
            pan_report = 0
            print("Pan at "+str(round(pan_location,0))+")")
            pan_OneStep()

#Sets direction of the x-axis rotation and calculates how many steps to
achieve a certain degree turn
def pan_moveDegrees(pan_move2):
    global pan_location
    pan_delta = pan_move2 - pan_location
    if pan_delta > 0:
        pan_dir.high()
    if pan_delta < 0:
        pan_delta = -pan_delta
        pan_dir.low()
    #degrees to steps - 0.0266 degrees per step
    pan_steps = pan_delta / 0.0266
    return pan_steps
    #pan_xSteps(pan_steps)

#-----Tilt Functions-----
-----
#Move y-axis by 1 step - Used by all other tilt functions
def tilt_OneStep():
    global delay
    tilt_trackLocation()
    utime.sleep(delay*(speed/100))
    tilt_step.high()
    utime.sleep(delay*(speed/100))
    tilt_step.low()

#Iterates OneStep function a certain amount of times depending on how many
degrees to turn
```

```
def tilt_xSteps(tilt_steps3):
    tilt_report = 0
    #print location every 100 steps
    for i in range(tilt_steps3):
        tilt_report += 1
        if tilt_report == 10:
            tilt_report = 0
            print("Tilt at "+str(round(tilt_location,0))+""")
            tilt_OneStep()

#Sets direction of the y-axis rotation and calculates how many steps to
achieve a certain degree turn
def tilt_moveDegrees(tilt_move3):
    tilt_delta = tilt_move3 - tilt_location
    #print(tilt_location)
    #print("fff", tilt_delta)
    if tilt_delta >= 0:
        tilt_dir.high()
    else:
        tilt_delta = -tilt_delta
        tilt_dir.low()
    #print("ff", tilt_delta)
    #degrees to steps - 0.225 degrees per step
    tilt_steps = tilt_delta / 0.0775
    #print("####",tilt_steps)
    return tilt_steps
    #tilt_xSteps(tilt_steps)

#-----Move Pan & Tilt at same time function-----
#Divides both movements into x chunks. Alternates moving each axis by a
chunk until full distance travelled
def both_moveDegrees(pan_move2, tilt_move2):
    if -180 > (pan_location+pan_move2) < 180:
        pan_move2 = 0
    if -180 > (tilt_location+tilt_move2) < 180:
        tilt_move2 = 0

    if pan_move2 and tilt_move2 >= 45:
        chunks = 500
    else:
        chunks = 30

    pan_steps2 = pan_moveDegrees(pan_move2)
    pan_share = pan_steps2/chunks

    tilt_steps2 = tilt_moveDegrees(tilt_move2)
    tilt_share = tilt_steps2/chunks

    if debug: print("Debug: pan_share:%s steps:%s tilt_share:%s steps:%s"
% (pan_share,pan_steps2,tilt_share,tilt_steps2))
```

```
    for i in range(chunks):
        pan_xSteps(pan_share)
        tilt_xSteps(tilt_share)

#-----Main Code-----
#-----
while True:

    command = input("""

    1 [x-degrees] [y-degrees]. Move to defined location
    2 [x-degrees] [y-degrees]. Move relative to current location
    3. Start Tracking
    4. Move to position 0
    5. Reset position to 0
    6. Toggle stepper motors
    7. Change motor speed
    9. Toggle debug mode
    Enter Command: """)

    if command[0] == "1":
        target = command.split()
        pan_move = float(target[1])
        tilt_move = float(target[2])
        #pan_moveDegrees(pan_move)
        #tilt_moveDegrees(tilt_move)

        both_moveDegrees(pan_move,tilt_move)
        print("X-axis current Location: "+str(round(pan_location,2))+""")
        print("Y-axis current Location: "+str(round(tilt_location,2))+""")

    elif command[0] == "2":
        target = command.split()
        pan_relative = int(target[1])
        tilt_relative = int(target[2])
        pan_move = int(pan_location) + pan_relative
        tilt_move = int(tilt_location) + tilt_relative
        both_moveDegrees(pan_move,tilt_move)
        #pan_moveDegrees(pan_move)
        #tilt_moveDegrees(tilt_move)
        print("X-axis current Location: "+str(round(pan_location,2))+""")
        print("Y-axis current Location: "+str(round(tilt_location,2))+""")

    elif command == "3":
        tracking = True
        #Read information from RX pin and decode it into a string. Split
the string in to x and y demand
        #Packet format is x---,y---
        #Demand is received as pixels and is converted using 0.35625
degrees per pixel calculated from resolution and FoV of camera.
```

```

#The mount is only moved if the tracking demand is greater than 2
degrees
while tracking:
    print("""Tracking
    Current Location : %s°, %s°
    """ %(pan_location, tilt_location))
    #uart.write("x%3d,y%3d"%(30, -40))
    utime.sleep(0.1)
    if uart.any():
        buffer = uart.read(9)
        buffer = buffer.decode('utf-8')
        print("buffer =", buffer)
        uart.write('recv\r\n')
        if buffer[0] == 'x':
            pan_move, tilt_move = buffer.split(",")
            pan_move = float(pan_move[1:])*0.3526
            tilt_move = float(tilt_move[1:])*0.3526

            if fabs(pan_move) and fabs(tilt_move) >= 2:
                if debug: print("Pan demand: %s , Tilt demand: %s"
%(pan_move, tilt_move))
                both_moveDegrees(pan_move, tilt_move)
        elif command == "4":
            pan_move = 0
            tilt_move = 0
            both_moveDegrees(pan_move, tilt_move)
            print("X-axis current Location: "+str(round(pan_location, 2))+ "°")
            print("Y-axis current Location: "+str(round(tilt_location, 2))+ "°")
        elif command == "5":
            pan_location = 0
            tilt_location = 0
            print("location reset to 0")
            print("x-axis current Location: "+str(round(pan_location, 2))+ "°")
            print("y-axis current Location: "+str(round(tilt_location, 2))+ "°")
        elif command == "6":
            #motor_Toggle()
            motor_en.toggle()
        elif command == "7":
            speed = int(input("Enter speed percentage: "))
            setSpeed()
            print("Set motor speed to "+str(speed)+"%")

        elif command == "9":
            if debug == True:
                debug = False
                print("Debug mode is now off")
            else:
                debug = True
                print("Debug mode is now on")
        else:

```

Josh Asare – 2111739, Jack Heeks – 2111738, Ethan Wilson – 2114187, Robbie Young – 2111907

```
print("Command not recognised")
```