FACULTY OF ENGINEERING AND DESIGN

IMEE FINAL YEAR MEng PROJECT REPORT

Hybrid Image Recognition Camera for Search and Rescue Drones

Omar Ali

20th May 2020

"I certify that I have read and understood the entry on Plagiarism, duplication of one's own work and Cheating on the Quality Assurance Code of Practice QA53 and in my Departmental Student Handbook. All material in this assignment is my own work, except where I have indicated with appropriate references. I agree that, in line with Regulation 15.3(e), if requested I will submit an electronic copy of this work for submission to a Plagiarism Detection Service for quality assurance purposes."

Author's signature: ……

Supervisor: *Dr. P. Iravani*

Assessor: *Dr J. Taylor*

Abstract

The aim of this project was to develop a lightweight infrastructure for an assistive camera system for the purposes of a Search and Rescue (SAR) drone. The core goals within this project included the development and implementation of Sensor Fusion between visual and thermal images, as well as the implementation of a Region of Interest model, to autonomously identify regions within the image that contains a person to allow the camera system to relocate their position into the centre of the frame.

The Sensor Fusion in the system was a tedious process to implement, but when completed provided a very good overlay of the images. It found that in ideal cold conditions for the thermal camera, the system could very clearly identify people at distances reaching 40m. This was not the case in warmer weather as the system could not differentiate between the person and the environment very well.

The Region of Interest Modelling applied a range of saliency models to use as the primary method to identify people within an image. It was found that although there were many cases these models could pick a person out, this was typically amongst a set of other salient regions that were also identified. It was concluded for this to be implementable, further processing and testing would need to first be completed before it can be deemed reliable.

When fully configured, the overall system was reasonably light, weighing at 660 grams, in quite a compact size. It was evaluated to operate for a minimum operating time of 45 minutes, however it is not in a ready state to be implemented into a drone as work needs to be further developed within the Region of Interest model as well as the implementation of this into the gimbal.

Acknowledgements

I would like to thank my supervisor, Dr P. Iravani, for the continual support and advice through the semester regarding the project. I would also like to thank all the members within the Team Bath Drones group for being a great community to be a part of being there to support my project in times where it was needed. Finally, I would like to show my gratitude to my parents for providing me with support through my time at university.

Table of Contents

Table Of Figures

Page is left intentially blank.

1 Introduction

1.1 Project Background

Drone technology has grown rapidly in the last decade and has become an essential part of a range of industries as directed innovations are developed to improve them [1]. The emergency rescue division [2] has been a particularly interesting sector within drone technology as it requires quite advanced and reliable technologies to function within them within a compact system.

Competitions across the world such as the ERL, UAS IMechE challenge and the Australian UAV Medical Challenge have become more prevalent in the last few years attracting university and research teams to participate to complete a range of disaster relief missions using autonomous drones [3] [4] [5]. These challenges are a great incentive for squeezing out the most innovative solutions to complex problems such as path planning and object detection within an autonomous system in a compact and lightweight form.

This project is part of a greater project to develop an autonomous Search and Rescue (SAR) drone within the Team Bath Drones group. This was inspired by a design specification provided by the 'Search and Rescue Aerial Association' (SARAA) based in Scotland.

Figure 1 Aurora, a drone developed for Search and Rescue

1.2 Project Aim and Objectives

The aim of this project is to develop an infrastructure, for an improved camera system that fuses visual and thermal images as well as looks into the implementation of image recognition models for the purpose of an autonomous camera system. The fusion of images should help the operators more easily identify people that may be lost by observing the images. The image recognition models should be applied to act as an initial step for a functional autonomous camera system.

Table 1 Table of User and System Requirements for the Drone's Camera Sub-System

User Requirements	
UR1	Sensor Alignment - The UAS dual sensors should be auto aligned without landing.
UR2	Image Quality - The UAS image display should be presented clearly.
UR3	Cameras - The UAS should have a combined thermal and visual camera capability.
System Requirements	
SR1	Imaging - The UAS should be capable of identifying a casualty via thermal imaging.
SR2	Assembly - The UAS should require minimal tools to assemble.
SR3	Search Height - The UAS should search at a height of 30-50m.
SR4	Mass - The camera system should retain a sub-system payload limit of 1 kg
SR5	Interface - The cameras shall be capable of displaying images on the System's microprocessor
SR6	Gimbal Stability - The gimbal should stabilise the camera images to allow better analysis of the image.
SR7	Overheating - The components within the camera Sub-System should not overheat.
SR8	Operating Time – The drone should be fully operational for 30 minutes
SR9	Autonomy – The Drone should operate autonomously

The user requirements derived from a specification list previously provided by SARAA, and the system requirements were derived from the desired performance of the drone set by the team.

Taking these into account a set of objectives were formed for the system.

1- Camera Setup and Data Capture
A customised infrastructure should be made to capture the visual and thermal images to be easily post processed. This system should test and capture data in an outdoor environment for post processing analysis, to understand the benefits and limitation of the system.

2- Sensor Fusion
The visual and thermal camera system should go through the procedure of image fusion to produce images with more information. This should be tested and analysed.

3- Region of Interest Modelling
Regions in which a person may be should be identified. If successful, the script should be capable of identifying the location of a person and send a gimbal command to centre the camera system to that location.

4- System Integration into the Drone
An analysis of the system should be done to observe how well it would fit into the final Search and Rescue drone.

This project is intended to contain a system which includes a set of wide-angle visual and thermal cameras to apply coarse searching. This would be followed by a set of narrow angle camera which would apply a finer search to allow full autonomy to the camera system, however this is beyond the scope of this project and is being worked on within a separate project.

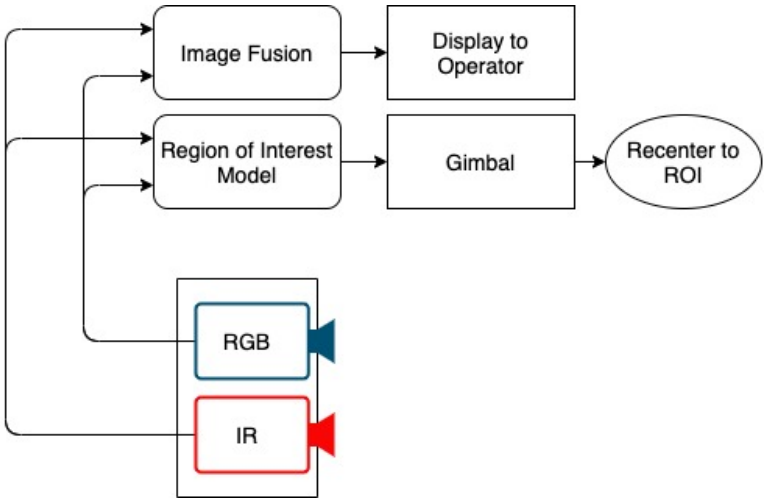


Figure 2 Diagram demonstrating the core operations of the camera system

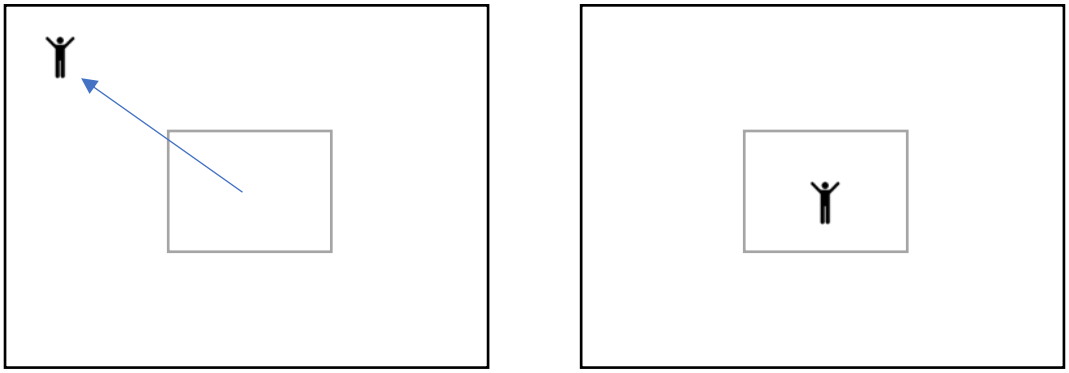


Figure 3 Example of the operating ROI model

2 Literature Review

An initial review of similar work was identified to see what other available systems and implementations exist. This is the initial learning stage to show how different technologies may be able to combine together to create an innovative working package.

The task of identifying humans from a drone has been a computer vision task of interest within the drone community for many years. The techniques used are also ever growing with the development of many object detection algorithms being produced every year. A few platforms of interest have been identified to apply visual and thermal images as well as object detection in an array of scenarios.

In a paper from Linkoping University [6], an impressive camera system was developed to identify the locations of people on the ground from a drone using visual and thermal imagery to complete this task. This applied a 'cascade of boosted classifiers' to allow people to be detected, this however uses quite a computationally expensive system but can afford to do so as the drone used was capable of operating at a take-off weight of 95 kg.

Figure 4 Results for the visual and thermal human [6]

The system still generated some false positive results, but they could always successfully identify the people in the images. The system could also geolocate the positions of the objects identified using the extrinsic parameters of the camera as well as the drone's geometry to the ground.

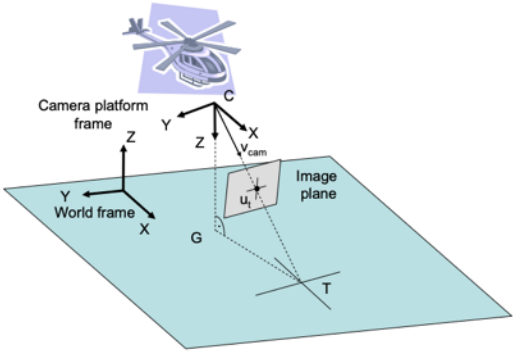


Figure 5 Geometry to geolocate the identified person

Another paper of interest had the goal of detecting fire in the wilderness but also using visual and thermal imagery from a drone. However, this paper uses colour extraction from an image to complete this task, ie relies identifying high concentrations of red regions within the visual

image. In a similar way, image segmentation was also used on the thermal images as a fire would generate a far hotter region in the images than anything else. [7]

Another paper of a group of researchers that developed a sensor fusion system for SAR applications was also identified to be quite useful for its direct applicability [8]. The paper described the outlining procedure the cameras in a fusion system would need to go through to apply fusion. The use of unnatural colours was applied to this to more easily identify the hotter regions in the image.

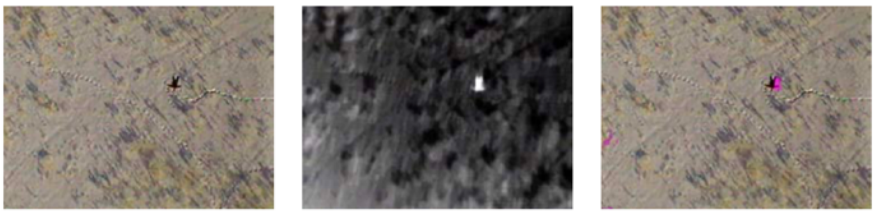


Figure 6: Research paper displaying fusion of person lying in snow [8]

Object detection algorithms were also explored. The 'Yolo' object detection algorithm [9] was of interest as it is considered to perform quite quickly. Furthermore, it comes in a less accurate but much more lightweight form intended to operate on smaller less powerful machines, this is called 'Yolo-tinyv3'. The idea behind this algorithm is that instead of convoluting across an image, like in a typical convolutional neural network, it directly applies analysis on many randomly spaced and sized boxes in an image and does this all in one pass thus making this quite a fast and efficient algorithm. As great as this is compared to some alternatives, this model still requires some computational power and overhead to operate, along with a well-trained model before the system is useable meaning it would require a lot of data, analysis and training before the final implementation.

Other detection systems looked into were a range of saliency models. Visual saliency is a characteristic of the Human Visual System (HVS) which attracts human's eye to interesting points within an image. [10] [11]

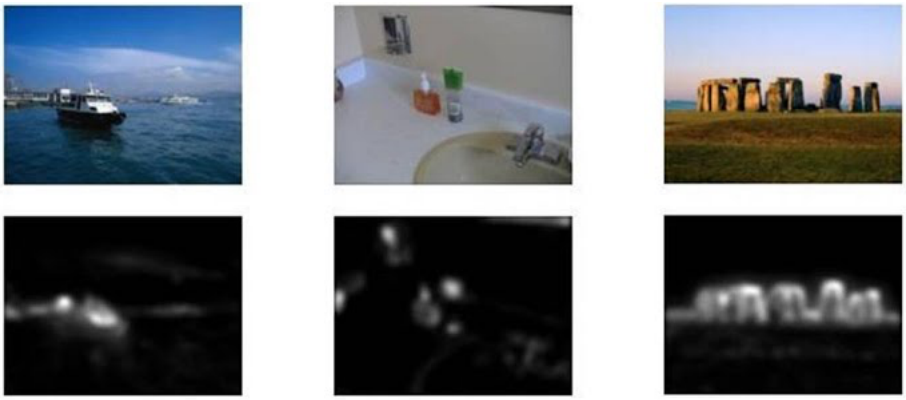


Figure 7: Original image (top), Static Saliency map of the image (bottom) [12]

These are methods of interest due to their ability to look for eye-catching regions in an image. Some of these models perform significantly faster than object detection algorithms meaning they would perform quickly on a smaller computer system. The open source computer vision (OpenCV) programming library contains a few of these models within in

The first of which is uses a ‘Fine Grained’ method which was developed by Montabone and Soto, in a project intended specifically for the purpose of human detection when in different poses whether they are in complete or in partial view of the body [13]. The second is the ‘Spectral-Residual’ method, developed by Xiaodi Hou and Liqing Zhang, which applies a spectral analysis to the image to find outlying features by identifying regions of sharp intensity changes [14]. There are a range of saliency models that use deep neural networks [15] [16] as well as convolutional neural networks [17] [18] to identify regions of interest by generating a saliency map, these provide a directed saliency search by providing the algorithm with a dataset to identify particular structures.

3 System Infrastructure

The system can be broken down into hardware and software. These need to work together to generate a complete camera system package for the SAR drone.

3.1 Hardware Infrastructure

The hardware in system is made up of the microcomputer, cameras and the gimbal module. These are interface together to create the camera system being defined.

3.1.1 Hardware Selection

Due to the system needing to be lightweight and compact, as described by system requirement SR4 ensuring the system stays light, the hardware range was selected accordingly.

Microcomputer

Within the university's drone lab, there were a selection of microcomputers available. Of particular interest were a couple Raspberry Pi 4Bs and a single operating Nvidia Jetson Nano.

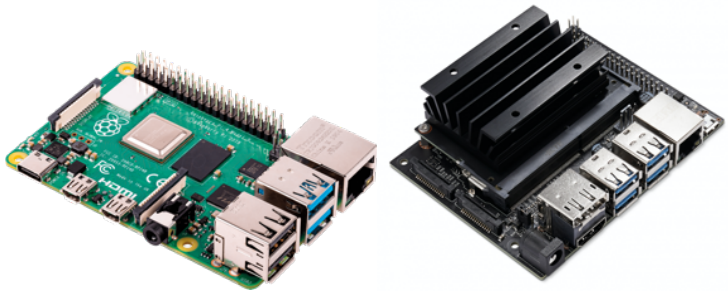


Figure 8 Image of the Raspberry Pi and the Jetson Nano Respectively

Both the Raspberry Pi and the Jetson Nano provide a similar port selection range, with the Jetson Nano allowing for more USB 3 ports thus having potential for a greater data transfer bandwidth if needed. The main benefit of the Jetson Nano is its dedicated graphics card, allowing it to more efficiently complete parallel processing applications which are of great benefit in image processing and machine learning applications. However, the Raspberry Pi provides better CPU performance with its Quad-Core ARM Cortex-A72 64-bit @ 1.5Ghz, as opposed to the Jetson's older generation Quad-Core ARM Cortex-A57 64-bit @ 1.42Ghz, making the Raspberry Pi better as a multipurpose microcomputer which would do a better job of interfacing different hardware and software together. [19]

With regards to the camera interfaces, both systems have a similar IO selection. The main difference with capturing image data would be in the way the cameras were called within the code. The port selection relevant to camera interfaces include a single camera CSI input port, the USB ports and the GPIO interface.

Taking these factors into consideration along with the fact that a similar project to this one is being completed with the focus of object classification and object detection, it was decided that this project would be completed on a Raspberry Pi as it is a more versatile overall system.

Primary Visual Camera (Wide FOV)

The primary camera available to the project was the ArduCam UC-350 Rev.B Sensor board which uses a Sony IMX219 8-megapixel CMOS sensor in combination with the ArduCam M2504ZH05S 45° field of view low distortion lens [20] [21].

Figure 9 Interfacing the ArduCam Module to the Raspberry Pi

This camera interfaces using the Raspberry Pi Camera CSI ribbon cable and uses the 'PiCamera' library to access the camera. This camera is intended to act as the wide angle FOV Camera for the system.

Secondary Visual Camera (Narrow FOV)

The Secondary Visual camera in the system is the Logitech C270 HD WebCam with 60° FOV Lens capturing images of up to 720P at 30fps. [22]

Figure 10 Showing the disassembled WebCam and it's interfacing with the Rapsberry Pi

This will interface with the Raspberry Pi via USB and can be accessed using the 'OpenCV' library. This camera was used due to availability in the circumstances rather than for ideality and is intended to be used as a placeholder for a narrow FOV camera.

Thermal Camera

The thermal camera used in the project is the Seek Compact Pro mobile thermal camera with an FOV of 32° with a resolution of 320x240 pixels. [23]

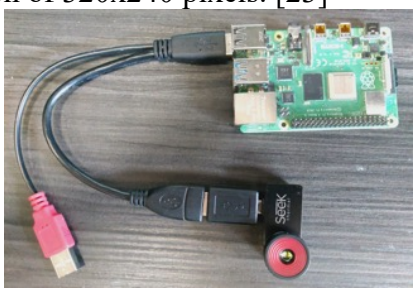


Figure 11 Seek Pro Camera Interfaced with the Raspberry Pi

Although this camera was developed to operate on an android phone through a micro-USB connector, a range of scripts have been developed to access the camera from other platforms. The 'seekpro.py' python script generated by Victor Couty [24] has been used, and modified, to access the thermal camera from the Raspberry Pi. This thermal camera was used in this

project due to the high-resolution thermal images being produced at a very compact and relatively cheap package.

Gimbal

A gimbal was also purchased with the intended use of controlling the direction the cameras are pointing based on the region of interesting modeling.

Figure 12 SToRM-32 Gimbal System

This gimbal uses the SToRM32 gimbal controller. This has the capability to interface with the Pixhawk 2 flight controller, typically used on the drones developed within the University's Drone Lab, by communicating with it via MAVLINKv2 commands. This was not fully implemented into the project however it is available for the sake of analysis.

3.1.2 Hardware Interface

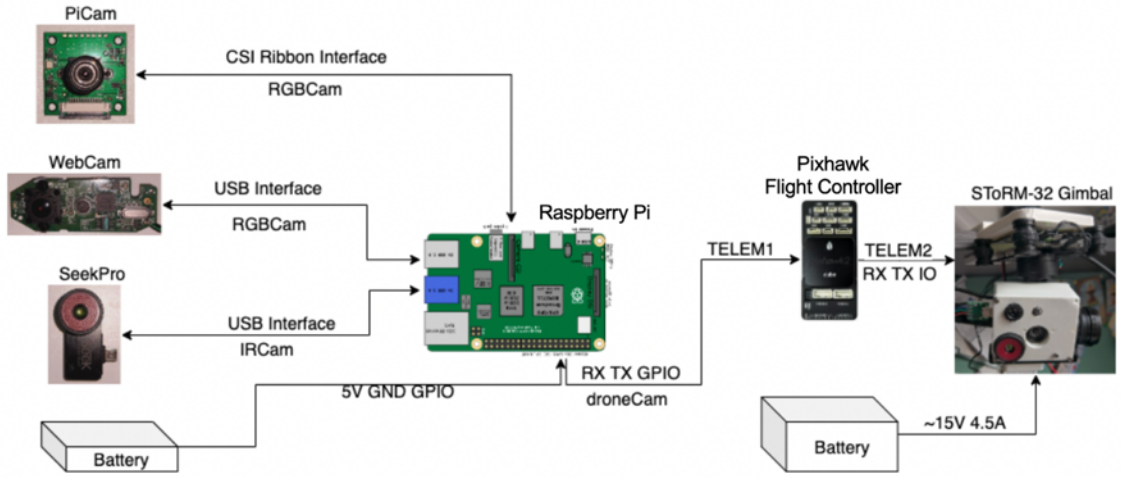


Figure 13 Hardware interface of the wide-angle camera system

3.2 Software Infrastructure

3.2.1 System Operating System

The Raspberry Pi, in this project, uses Raspbian Buster (version 4.19) as the operating system. This is the official, and most up to date, OS for the Raspberry Pi. The installation procedure for this can be found in the appendices section 11.1, this followed a very standard procedure for setting up the Raspberry Pi for computer vision applications.

3.2.2 Programming Language

Python 3 (version 3.7) was the programming language of choice within this project. Reasons for this include some prior experience with the language as well as its extensive documentation. It is also a higher-level programming language compared to C or C++. The main benefit of C or C++ is the speed and efficiency they have over python, however, at this stage the system is acting as a proof of concept thus there is more of an emphasis developing an operational system.

3.2.3 Programming Libraries for the project

There is a list of essential python libraries necessary to get the developed code in this project to function. The installation procedure for these libraries can be found in the appendices section 11.4.

NumPy

NumPy is a very versatile library used for data processing and analytics. It is mainly used in this project for its ability to store and manipulate arrays, which is important in image processing applications where all images are stored in arrays and it would be necessary to modify them in an intuitive and easy way. [25]

OpenCV

OpenCV is the open source computer vision library which is very heavily used to manipulate and process image data. It is a standard library used for imaging application and will very commonly be used in any python projects that require any image data acquisition. [26]

Pillow (PIL)

Pillow is another image processing library, but it is much more lightweight in comparison to that of OpenCV. It contains basic image processing functions, such as reading the image, as well as cropping and resizing functions. [27]

PAIRML (Saliency)

The saliency library in python is an experimental library with a range of saliency models. This will be covered in more detail later on in the project. [28]

PiCamera

The PiCamera library is what is used to interface with cameras that use the CSI camera interface. It allows the settings of the camera to be modified and images to be captured from it. [29]

PyUSB

This allows the user to more easily capture and send information using the USB protocol through python. [30]

3.2.4 Software Interface

There were 5 main operational scripts that were developed and modified over the course of this project. Some others can be found in the appendices, section 13, however they do not contribute to the running of the system beyond some side implementations.

droneCam.py – The interfacing code to capture and post process all the scripts together

RGBCam.py – Visual camera capture script

PiCam() – To capture images from the ArduCam module
WebCam() To capture images from the Logitech WebCam

IRCam.py – Thermal camera capture script

SeekPro() - To capture images from the SeekPro

SaliencyCV.py – Applies the OpenCV Saliency Models for the ROI Model

findSaliencyFineGrained() – Applying the Fine-Grained Saliency Model
findSaliencySpectralResidual() - Applying the Spectral-Residual Saliency Model
contourProcessing() – Identifies region of interest location to send to the gimval

SaliencyTF.py – Applies the PAIRML saliency models for the ROI Model

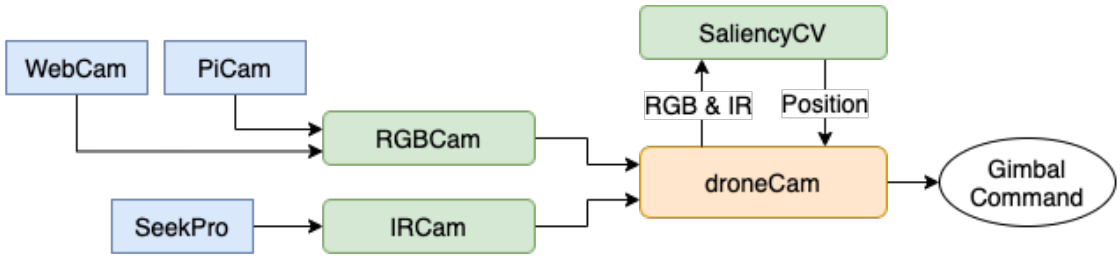


Figure 14 Interface of the system's software

All the scripts used for this project can be found in the Section 13. These are also available on a GitHub repository developed specifically for this Project.[1]

[1]SARCam GitHub Repository https://github.com/Olseda20/SARCam.git

4 Camera Setup and Data Capture

This section aims to ensure the cameras in the system are correctly setup to capture and display image data. Beyond this, the testing of the system was done in outdoor environments to check for image quality as well as capturing data for future processing in the project. The system requirements this section aims to satisfy include UR2, UR3, SR1 and SR5.

4.1 Setup of the Camera system

This section will show how the camera data is captured in the scripts as well as performing an analysis on the image data captured to identify the strengths and weaknesses of visual and thermal imagery for the case of search and rescue.

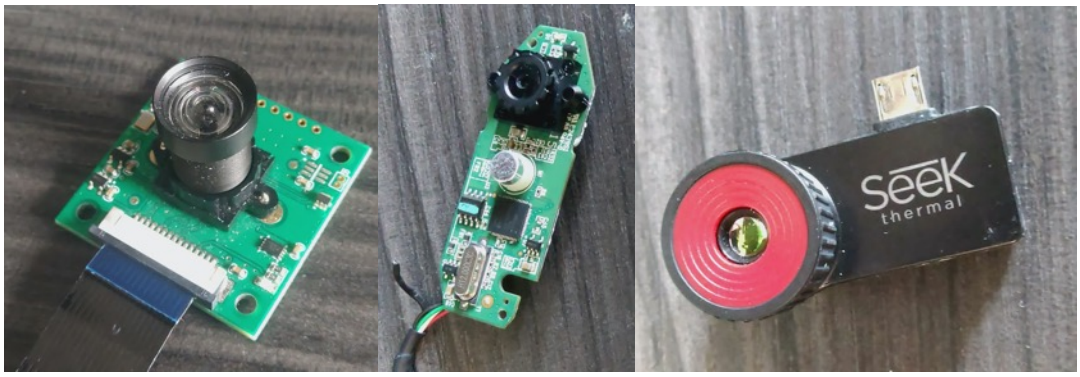


Figure 15 ArduCam Module, Logitech WebCam and Seek Pro Compact

A detailed explanation showing how the code was setup to capture relevant data from the cameras can be found in section 11. This code was developed to allow the cameras to be easily imported into any script, so all the files are within the same directory. From here onwards, the ArduCam module will be described as the PiCam, the Logitech WebCam as the WebCam and the Seek Pro Compact as the SeekPro.

Code Excerpt 1 – Showing how to directly implement the cameras into to display an image

```
import cv2   #to process the image data
import RGBCam#For visual cameras capture
import IRCam #For thermal camera capture
#Accessing the camera classes
PiCam = RGBCam.PiCam()
WebCam = RGBCam.WebCam()
SeekPro = IRCam.SeekPro()
#Capturing the image data from each camera
WideImg = PiCam.frameCapture()
NarrowImg = WebCam.read()
IRImg = SeekPro.rescale(SeekPro.get_image())
#Displaying the images
cv2.imshow('WideImg', WideImg)
cv2.imshow('NarrowImg', NarrowImg)
cv2.imshow('IRImg',IRImg)
```

By placing the highlighted region, into a *'while'* loop and using the *'cv2.imwrite'* function, lots of images can be captured for post proccing.

4.2 Thermal Camera Issues and Limitations:

A few initial problems and observations were made with the thermal camera identifying a clipping issue it had causing failure in image capture when exposed to colder temperatures causing the cold regions to be white and all other regions to be black.

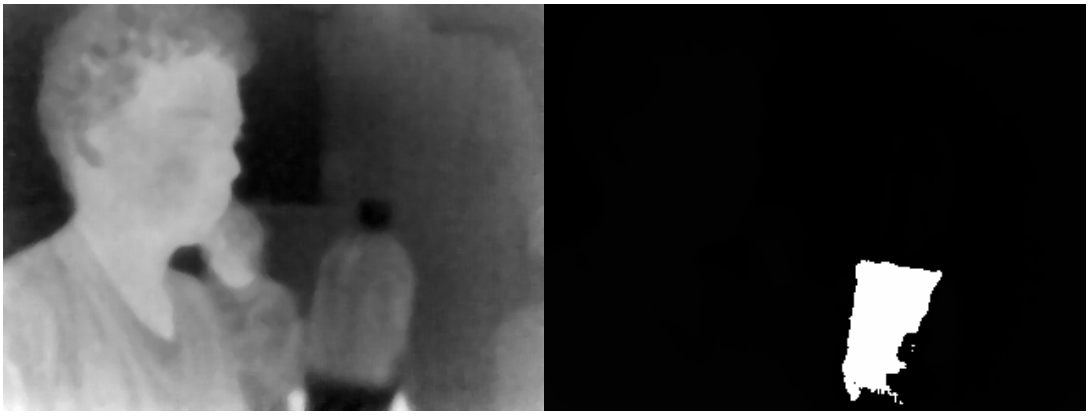


Figure 16 Showing the effects of bad clipping in the SeekPro code

Furthermore, some further testing was done on the SeekPro to show how the temperature of the camera can affect the thermal images produced. It was found that the lens produced hot regions along the edges as well as an increased amount of noise when the thermal camera was warm. This was accepted as a flaw of the thermal camera and will slightly affect all the results generated by the thermal images

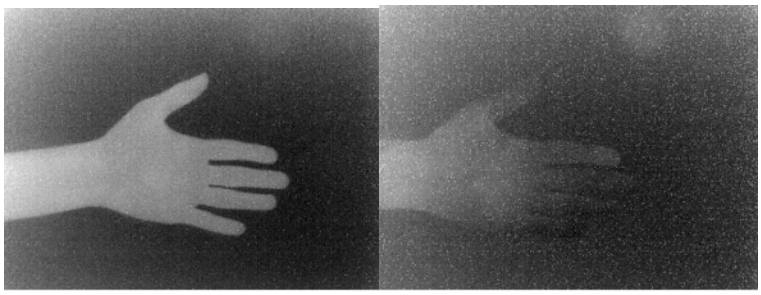


Figure 17 Extreme example of the bad performance degradation of the SeekPro due to heat exposure

More details on both of these observations can be found in the appendices at section 12.2.

4.3 Field Testing with the Cameras

The camera system was eventually taken outdoors in order to capture data for the future image processing applications. The data capture was useful to get a better idea of the limitations of the cameras in outdoor environments.

Two field tests were done through the course of this project one on a cold day and the other on a hot day. The conditions of the cold day were cloudy and slightly windy with a temperature of 12°C. The conditions of the hot day were sunny with a temperature of 20°C.

A range of images were taken at 10-meter intervals on each day to capture ranged data, this was estimated by equating 10m to 14 steps, by moving along that distance to gauge the cameras' performance.

Figure 18 PiCam & SeekPro image at 10m (left cold day, right hot day)

Figure 19 PiCam & SeekPro image at 20m (left cold day, right hot day)

Figure 20 PiCam & SeekPro image at 30m (left cold day, right hot day)

Figure 21 PiCam & SeekPro image at 40m (left cold day, right hot day)

The images being produced by the PiCam vary in clarity depending on the lighting conditions and the distance being captured. The images in the cold weather show that the person is visible up to about 20m, any further and it is quite difficult to identify the location of the person by eye. The images were improved in the hot weather and there is a clearer outline of the person at 30m, however, any further than this and the person becomes too unclear to identify.

The images produced by the SeekPro in the cold weather generated some really good quality thermal images allowing it to capture the human at a further distance. At 30m the human is still clear enough to identify different temperatures differentials across the body and at 40m, although small, there is still a noticeable hotspot at the position of the human. However, in hot weather, performance of the camera performed very poorly, and the person is not really identifiable beyond 20m.

The 45° FOV of the PiCam compared to the 32° FOV of the SeekPro can be noticed quite clearly in the images captured. The narrower FOV that the SeekPro provides means that the regions in image appear larger at the expense of the lost capture area.

A table to area coverage was generated to show how much area within an image a person is covering.

Table 2 Table to show area covered by the person per at a given distance in a 320x240 pixel image using the PiCam and the SeekPro

Distance	PiCam height (px)	PiCam Width (px)	PiCam image covered	SeekPro height (px)	SeekPro width (px)	SeekPro image covered
10	60	19	1.5%	100	32	4%
20	26	6	0.2%	43	13	0.7%
30	17	4	0.09%	29	8	0.3%
40	15	3	0.06%	15	5	0.1%

If it is assumed that a full pixel needs to be covered in order for a temperature to be evaluated by the thermal camera, a person would still continue to be identified at distances of at least 40 m in a thermal image under the right conditions, the thermal camera would still be able to pick up the thermal temperature of the person as they would fully cover the pixels of at least 7x2 squares. This would be enough to generate a heatmap from the person. This however does approach the point of reaching noise and is likely that beyond this it would be very difficult to tell if the person is within the image.

4.4 Discussion – Camera Setup

The field tests done showed the of the benefits of visual and thermal camera systems different outdoor conditions. The visual camera on the hot day was slightly out of focus however and beyond this, at 40m the person was in a different place to the prior images thus making it more difficult to know how much the setting affected the performance. In all the tests, the cameras PiCam captured images at 320x240, however ideally, this would be recaptured at a higher resolution, this would come at the expense of some performance as well as some additional post processing complications.

The PiCam generated great outdoor visual images with lots of contrast during the hot weather making the person quite identifiable up to 30m. In the colder weather, it had quite poor image performance and was not able to clearly identify the person past 20m. The dark clothing choices may have caused the performance to be even worse as is further worsens the contrast of the person.

The SeekPro performed in almost the opposite way to the PiCam. In the hot day, the SeekPro struggled to create much of a temperature differential between the person and the environment generating quire poor images to analyse. However, on the cold day, the images we excellent, very clearly identifying the position of the person up to 40m, and potentially even beyond this.

These results show how well a visual and thermal camera system can complement each other in these environments, where the visual camera could be primarily used in bright and warm conditions, and the thermal camera could be primarily used in darker colder conditions.

However, it is important to realise these are somewhat extreme examples and the performance in the other scenarios, such as warm but cloudy weather, or cold but sunny, need to also be investigated to see how the performance would vary.

5 Sensor Fusion

This section will cover all the relevant steps necessary to successfully apply fusion, between the visual and thermal images to the camera system. This section aims to cover the user and system requirements set by UR1 UR3 and SR1. These requirements are to ensure the camera system contains a well-functioning visual and thermal camera as well as having alignment within the images.

5.1 Camera Calibration

Camera Calibration is the act of removing lens distortion from an image. The distortion in an image causes some sections of the image to appear larger in size and other sections of the image to appear smaller in size. The distortion is important to eliminate because ideally both cameras should capture as similar of an image as possible to allow the sensor fusion to be best represented.

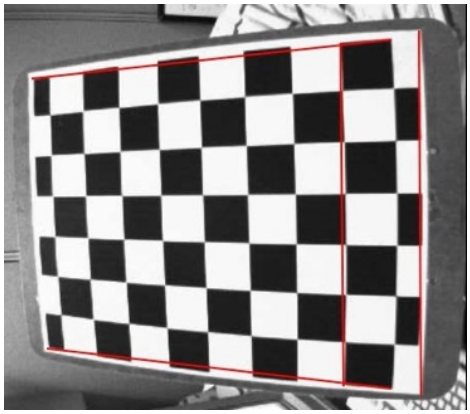


Figure 22 Example of a very distorted calibration board [31]

It can be seen in Figure 22 that the boxes on the checkboard pattern are not in a straight line even though they are in real life. Eliminating as much of this distortion as possible is important before applying a transformation matrix to align the images on top of each other for overlay.

There are 2 types of distortion that occur in cameras, radial and tangential distortion. [31]

Radial distortion is caused by flawed radial curvature of a lens and is described by:

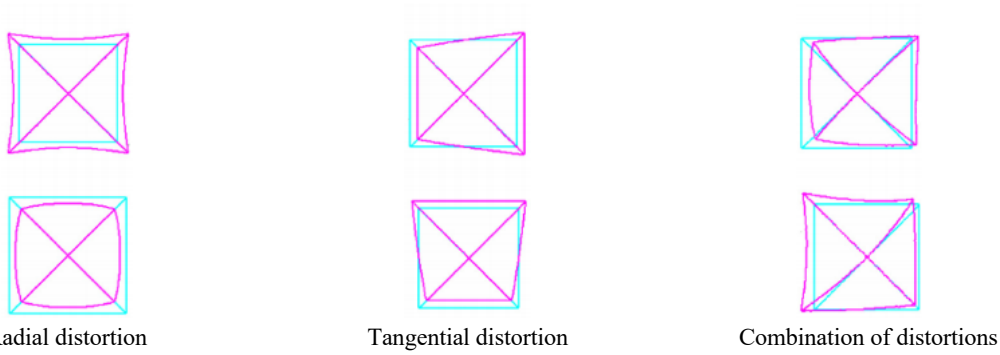
$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Where k_1, k_2 & k_3 are the radial distortion coefficient and r is the distance from point (x, y) from the centre of the image.

Tangential distortion is caused by a slight misalignment in camera assembly causing one side of the image to appear larger than another and is described by:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Where p_1 and p_2 are the tangential distortion coefficients. [32]



Radial distortion Tangential distortion Combination of distortions

Figure 23 The visual effects of distortion in an image [32]

To calibrate the camera images, the distortion coefficients need to be estimated.

$$Distortion\ coefficients = (k_1\ k_2\ p_1\ p_2\ k_3)$$

Furthermore, intrinsic parameters such as the camera focal length (f_x, f_y) and optical camera centre (c_x, c_y) need to be identified to create the camera matrix.

$$Camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

These matrices will be used for the camera calibration

Extrinsic parameters such as rotation and translation of the camera tend to be a common additional point of tuning when it comes to camera calibration, but explicitly applying these is unnecessary since the cameras will apply go through a procedure of perspective transformation later in the section. [31]

OpenCV has some functions implemented for image calibration which allow it to identify the size of specific structures, this includes a checkerboard pattern which can be seen in Figure 24. By providing the functions with the inputs of the size of the boxes and the number of rows and columns there are in the image, the function can generate estimations of the distortion coefficients and camera matrix by analysing many images of the pattern from different angles and distances.

Calibration Board Image Capture

Calibration boards for this process need to be developed before the evaluation of the distortion and camera matrices.

The visual camera calibration board is simply a 50mm square checkerboard pattern, printed out on an A3 sheet of paper and glued onto carboard. It is desirable to have a different number of squares along the horizontal and vertical axis of the checkerboard to make the orientation clear to the calibration script.

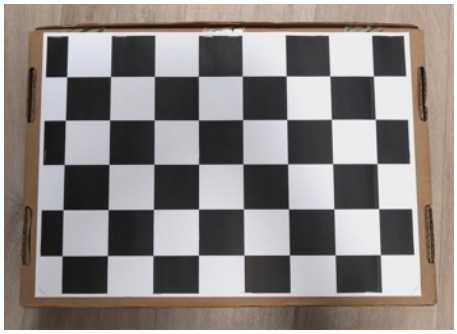


Figure 24 Calibration board for the visual camera

Multiple images of this board were taken from a range of angles and distances, these were saved together in a folder.

Code Excerpt 2 Using imwrite function to save PiCam images into a relevant folder

```
##Saving Image Data
timestr = strftime("%d%m%Y-%H%M%S") #Capturing the current time and date
cv2.imwrite('CameraCalibration/Images/PiCam/{timestr}piimg.png', WideCam) # saving image
time.sleep(0.3) #pause script prevent generating an overwhelming amount of data
```

Figure 25 Sample of PiCam images taken of the calibration board

The thermal camera calibration board was trickier as there needed to be a way to show contrasting thermal temperatures on the board since there are no thermal differences between the white and black regions of the visual calibration board.

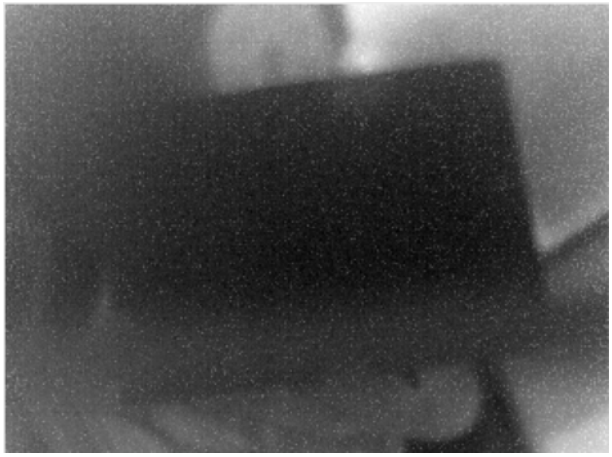


Figure 26 Thermal image of the visual image calibration board

It was noticed that, metals act like thermal mirrors, i.e. by pointing a thermal camera at a sheet of metal, the temperatures of objects around the room are visible on it.

Figure 27 Reflection of a laptop temperature using an aluminium sheet

Using this property, a thermal calibration board was made by arranging and sticking squares of 50mm black paper onto a sheet of 300mm x 300mm aluminium to create a checkboard pattern. A grid of 6x5 squares were arranged to allow the calibration algorithm to identify an orientation more easily.

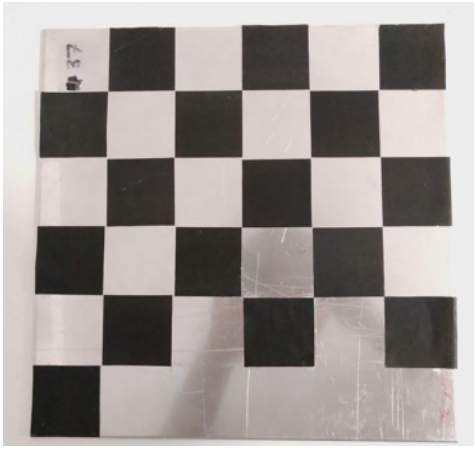


Figure 28 Calibration board for thermal camera

By doing this, the aluminium sheet can be heated up, by being placed on-top of a radiator or inside an oven. This should generate a large temperature differential between the paper and the room temperature allowing for clear distinctions between paper and the aluminium.

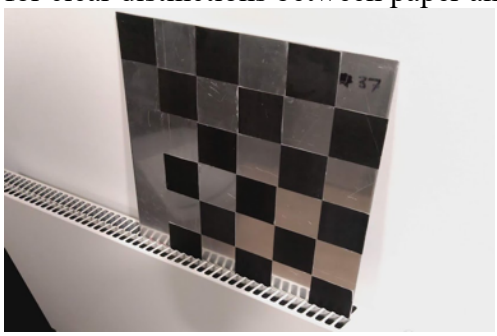


Figure 29 Heating up of the thermal calibration board

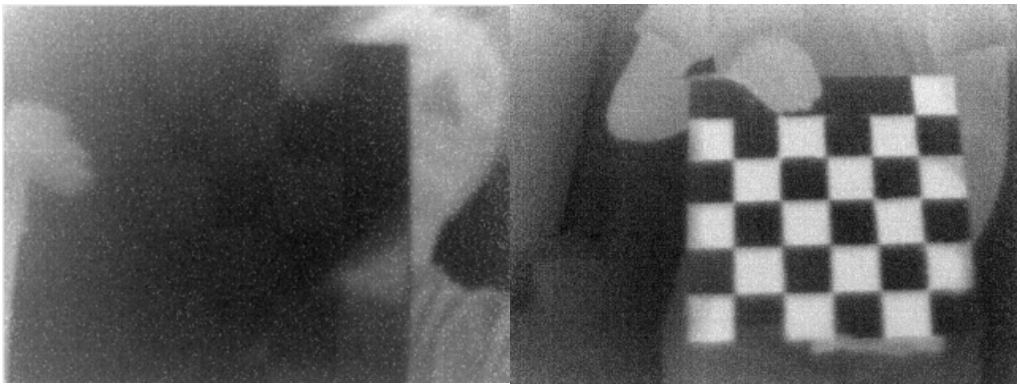


Figure 30 Thermal image of the calibration board before and after heating

A clear 6x5 checkerboard pattern is now generated and can be used for the camera calibration. In a similar way to the PiCam images, a set of thermal images were captured using the SeekPro class and saved into a folder.

Code Excerpt 3 Using imwrite function to save SeekPro images into a relevant folder

```
##Saving Image Data
timestr = strftime("%d%m%Y-%H%M%S") #Capturing the current time and date
cv2.imwrite('CameraCalibration/Images/SeekPro/{timestr}thermimg.png',IRImg) #saving image
time.sleep(0.3) #pause script prevent generating an overwhelming amount of data
```

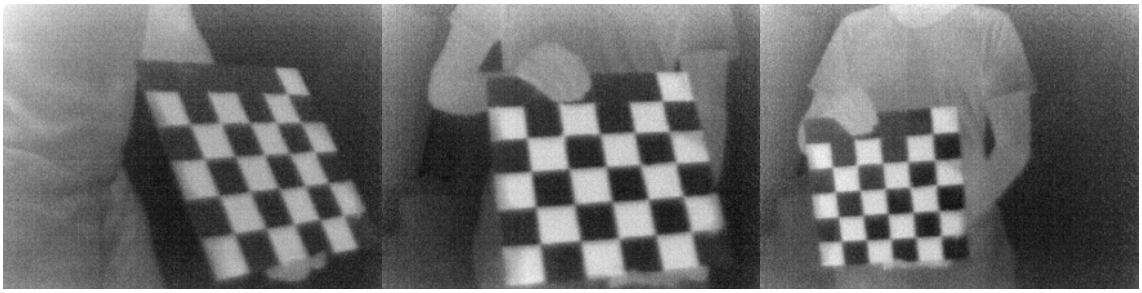


Figure 31 Samples of SeekPro images captured for calibration

Now that images for calibration have been captured, the calibration algorithm can be implemented to generate the relevant calibration matrices.

Calibration of images

There is a standardised approach to generating the calibration code described in the official OpenCV documentation [31]. This uses the images of the calibration board to identify the shapes and sizes of each square, which then identifies the distortion matrix as well as the camera matrix. This can later be used to remove the distortion from every image generated from the camera. The code that was used for this procedure can be found in the appendices at section 13.6.

There are a set of input parameters at the beginning of the script, used to generate a pattern for pattern recognition on the images.

Code Excerpt 4 Setting the input parameters of the image

```
#-- SET THE PARAMETERS
nRows = 4 # number of box intersections on the rows
nCols = 5 # number of box intersections on the columns
dimension = 50 # size of the boxes
workingFolder = "./CameraCalibration/Images/SeekPro" # path to image file
imageType = 'png' #format of the image
```

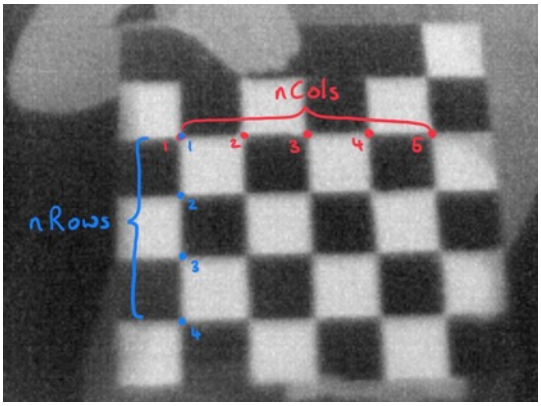


Figure 32 Showing how nCols and nRows should be evaluated

If the input parameters are correctly placed, once the code is run, the algorithm should correctly identify the calibration board's pattern in this way.

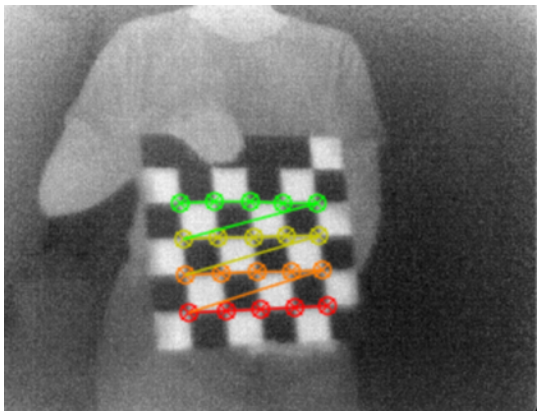


Figure 33 Successful pattern recognition for calibration

However, some modifications were made to the standardised code to allow the user to accept or reject the pattern recognised due to occasionally bad pattern recognition.

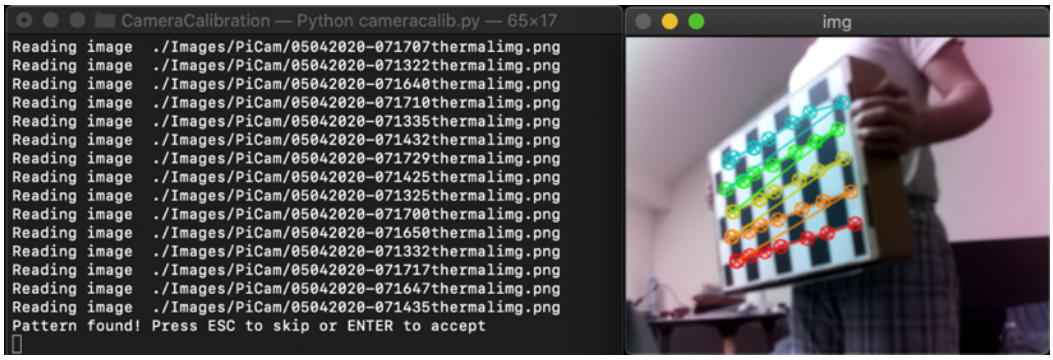


Figure 34 Example of bad pattern recognition from visual camera calibration attempt

For the calibration to work well, the working folder must contain more than 10 photos clear enough for calibration to achieve any reasonable results. The script can now be run and once completed, the script outputs the results and saves them into two text files in the same directory as the working folder containing the distortion coefficient and camera matrices.

```
Found 15 good images
Image to undistort:  ./Images/SeekPro/05042020-100841thermalimg.png
ROI:  9 7 302 230
Calibrated picture saved as calibresult.png
Calibration Matrix:
[[549.37517974   0.         186.13778055]
 [  0.         553.23471501 193.11098121]
 [  0.           0.           1.        ]]
[[-2.13274607e-01 -1.46747335e+00  3.60407600e-03 -8.28859067e-03
   4.02022661e+00]]
Disortion:  [[-2.13274607e-01 -1.46747335e+00  3.60407600e-03 -8.28859067e-03
   4.02022661e+00]]
total error:  0.09557893793606827
```

Figure 35 Results of the thermal image calibration

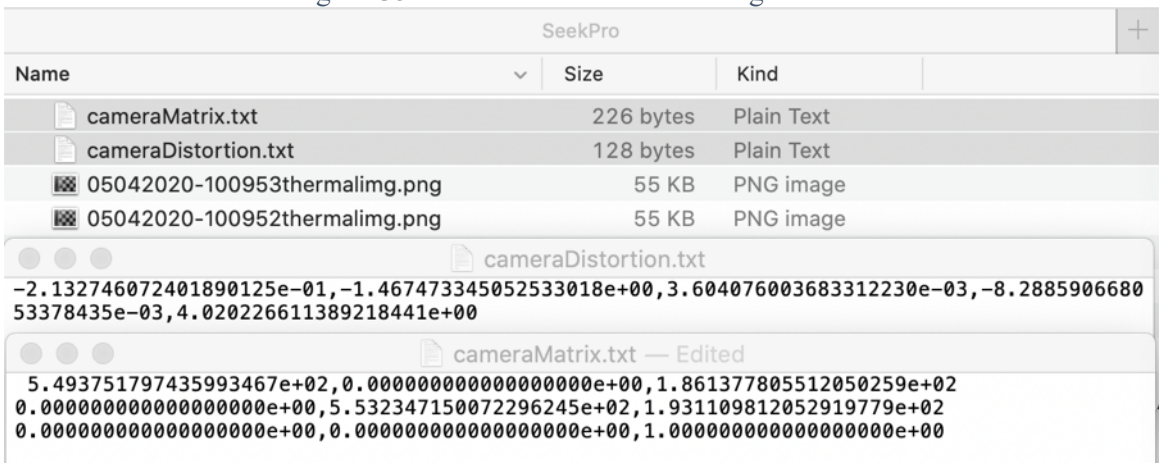


Figure 36 Showing the files generated by the script as well as their contents

Finally, to implement the calibration to apply onto the images generated by the camera, the files are loaded into a NumPy array then apply the *'cv2.getOptimanlNewCameraMatrix'* function from within the OpenCV library to generate the corrected image.

Code Excerpt 5 – Generating the calibration matrix for the thermal camera

```
Calibration for the thermal camera
SeekProCalib =
np.loadtxt('./CameraCalibration/Images/SeekPro/cameraMatrix.txt',delimiter=',')

SeekProDist =
np.loadtxt('./CameraCalibration/Images/SeekPro/cameraDistortion.txt',delimiter=',')

SeekProUnDisMat, roiSeekPro =
cv2.getOptimalNewCameraMatrix(SeekProCalib,SeekProDist,(RESOLUTION),1,(RESOLUTION))
```

The corrected image can finally be generated using the *'cv2.undistort'* function.

Code Excerpt 6 Removing the distortion from the image

```
#Revoming image distortion
IRCorrected = cv2.undistort(IRImg, SeekProCalib, SeekProDist, None, SeekProUnDisMat)
```

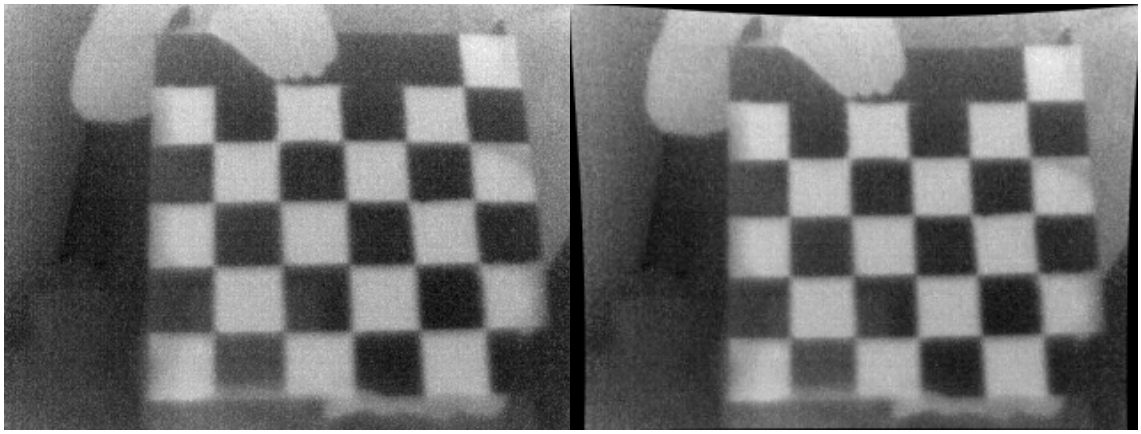


Figure 37 Displaying the thermal image before and after image calibration

5.2 Perspective Transformation

Perspective transformations is a form of homography matrices that allow images to realign depending on the points provided to it. As the name suggests, this changes the perspective of the image allowing for a view from a particular direction.

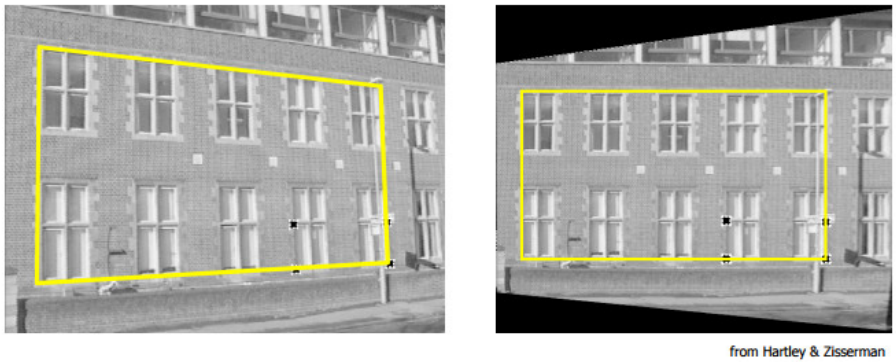


Figure 38 Perspective to perceive the building as perpendicular to the image [33]

This can be done if at least 4 original positions and 4 desired positions are selected within an image.

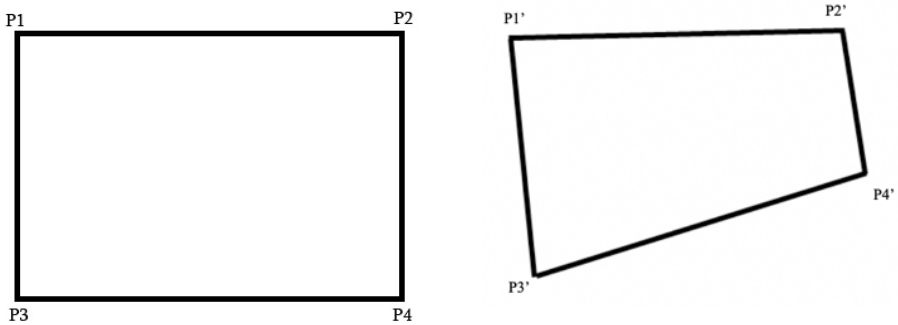


Figure 39 Showing 4 source and destination points for a perspective transformation

A 3x3 transformation matrix needs to be evaluated to successfully shift the points' position.

$$H_1 = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{12} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

Taking points P_1 and P_1' for example.

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{12} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

By identifying the transformation matrix, by solving the matrix using all 4 corners, the transformation matrix can be applied to all the other points within the image to move them to the new corrected position.

Camera Enclosure Development

Before the computation of the perspective transformation is applied, the cameras need to remain held together quite rigidly. This was done by developing a 3D printed enclosure for the cameras to be held together.

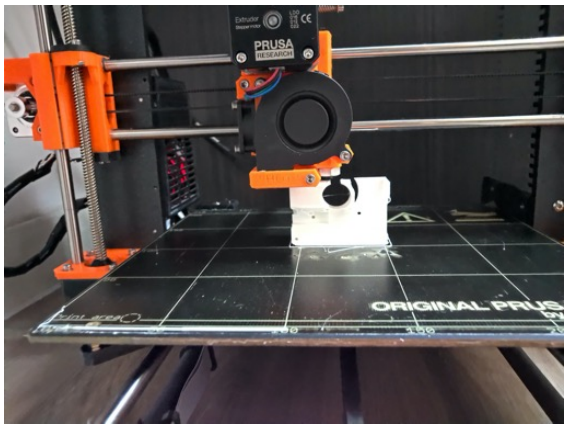


Figure 40 3D printing an enclosure to encapsulate all the cameras

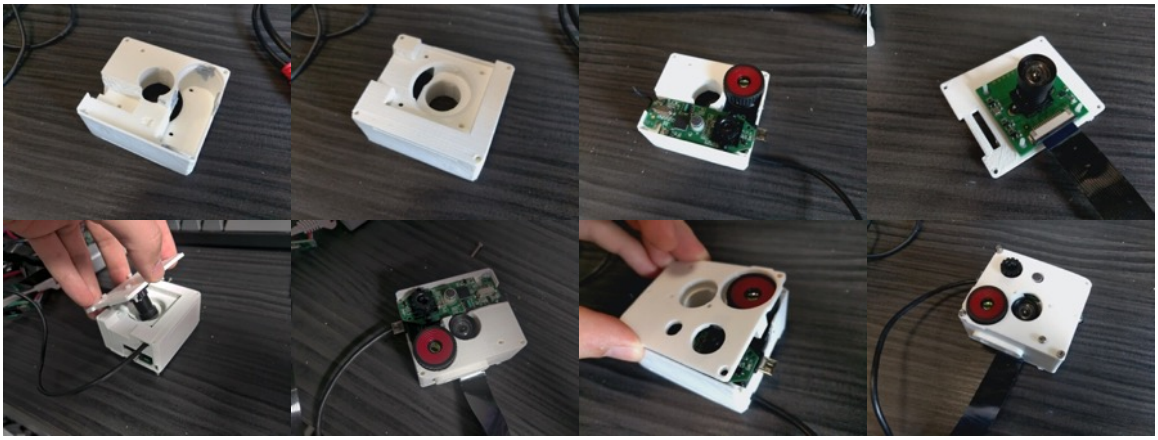


Figure 41 Assembly of the cameras into the enclosure

Once this was completed, multiple pictures were then taken using the SeekCam and the PiCam in their held configuration. It was important to capture several hot spots and distinct regions for the thermal camera to more easily be aligned with the visual camera.[2]

[2] All the photos taken during the field used this camera enclosure

Images were taken of myself standing in multiple orientation to captures many edges of the body, i.e. elbows, knees and fingers. The calibration board that was previously developed was also used as it captures quite clear edges when reasonably heated.

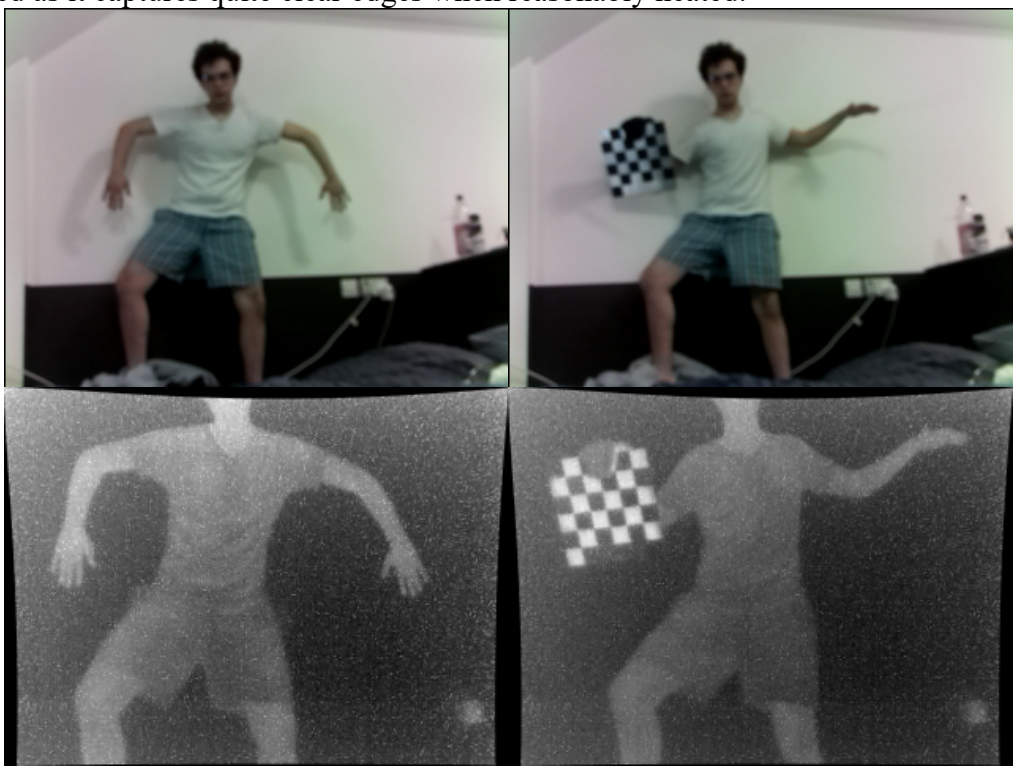


Figure 42 Corresponding images captured from the PiCam and SeekPro

To correspond these images to align with each other, MATLAB was used. It contains a useful function called the control point selection tool (cpselect) which allows points on images to be selected to generate two arrays with corresponding points.

The images captured should be imported into MATLAB and passed through the *'cpselect'* function as the fixed and moving points. The fixed points indicate the positions on the base image that should not move, and the moving points indicate the positions on the overlaying image that will contort to overlay. In this case, the PiCam images will be the fixed points and the SeekPro images will be the moving points.

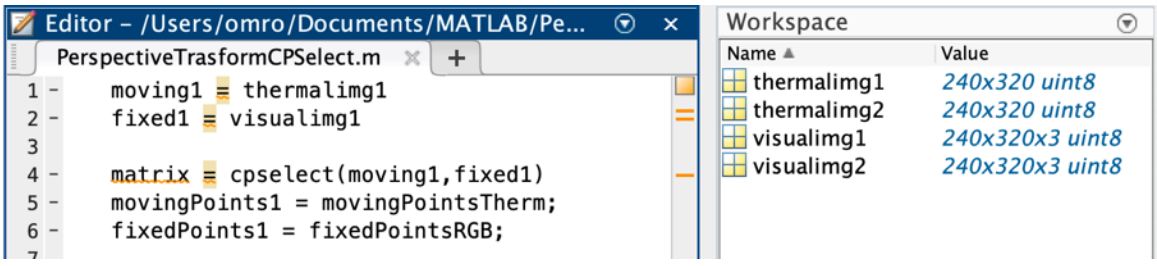


Figure 43 Generating the fixed and moving points from the cpselect function

When this is run, a selection interface shows up where corresponding points between the two images should be selected and saved as the movingPoints1 for the thermal image and the fixedPoints1 for the visual image.

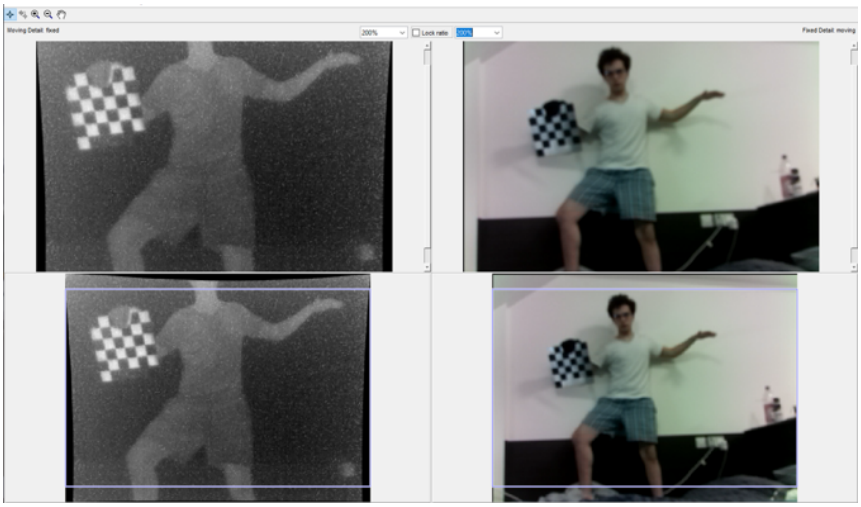


Figure 44 Before selecting relevant points on the cpselect GUI

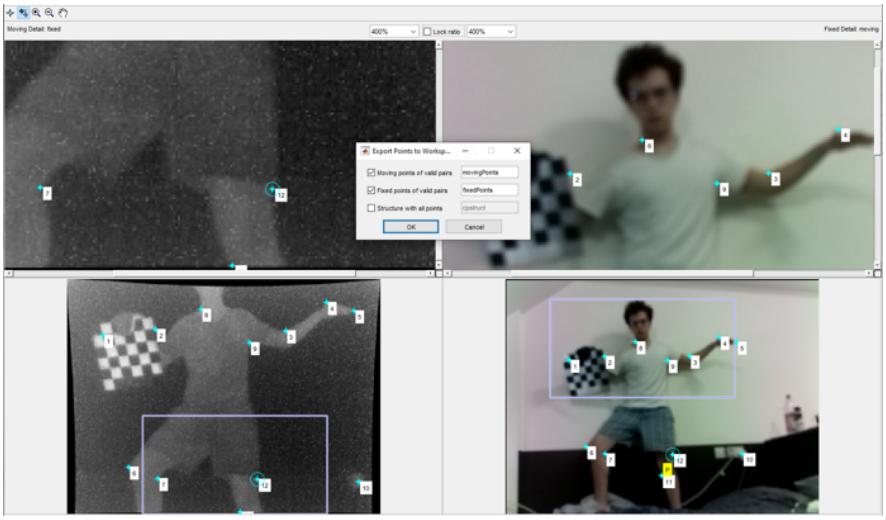
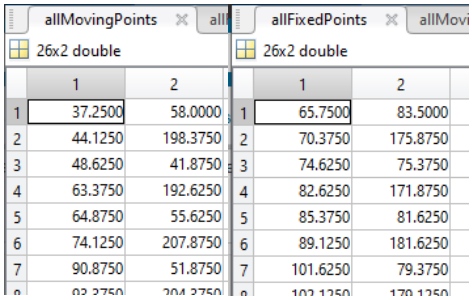


Figure 45 After selecting relevant points and saving the points into the relevant variables

A second set of points were generated in a similar way to and saved as movingPoints2 and fixedPoints2. These arrays were concatenated and sorted to be in ascending order of the X-axis position.

```
allMovingPoints = sortrows(vertcat(movingPoints1,movingPoints2))
allFixedPoints  = sortrows(vertcat(fixedPoints1,fixedPoints2))
```

Figure 46 Concatenating and sorting the first and second set of fixed and moving points



allMovingPoints

26x2 double

	1	2
1	37.2500	58.0000
2	44.1250	198.3750
3	48.6250	41.8750
4	63.3750	192.6250
5	64.8750	55.6250
6	74.1250	207.8750
7	90.8750	51.8750

allFixedPoints

26x2 double

	1	2
1	65.7500	83.5000
2	70.3750	175.8750
3	74.6250	75.3750
4	82.6250	171.8750
5	85.3750	81.6250
6	89.1250	181.6250
7	101.6250	79.3750

Figure 47 All the moving and fixed points ascending order in the x-axis

Finally, this is saved into a text file in an easily readable format for python.

```
>> dlmwrite('movingPoints.txt',allMovingPoints)
>> dlmwrite('fixedPoints.txt',allFixedPoints)
```

movingPoints - Notepad

File Edit Format View Help

37.25,58
44.125,198.38
48.625,41.875
63.375,192.63
64.875,55.625
74.125,207.88

fixedPoints - Notepad

File Edit Format View Help

65.75,83.5
70.375,175.88
74.625,75.375
82.625,171.87
85.375,81.625
89.125,181.63

Figure 48 Saving and displaying the fixed and moving points in text files

These files should be moved into the main working directory on the Raspberry Pi.

These points can now be loaded into the python script and processed into a perspective transformation matrix '*tform*' using the *'cv2.findHomography'* function which takes the list of moving points and fixed points to generates a matrix to warp the moving points onto the fixed points.

Code Excerpt 7 Generating the perspective transformation matrix from the saved points

```
#Loading in the moving and fixed points to generate the perspective transformation
movingPoints = np.loadtxt('/home/pi/SARCam/movingPoints.txt', delimiter=',')
fixedPoints  = np.loadtxt('/home/pi/SARCam/fixedPoints.txt', delimiter=',')
#Generating the perspective transformation matrix using the moving and fixed points
tform, status = cv2.findHomography(movingPoints, fixedPoints)
```

After capturing a thermal image 'IRImg', the image should pass into the *'cv2.warpPerspective'* function with the previously generated 'tform' matrix as well as the resolution of the output image.

Code Excerpt 8 Applying the transformation matrix for the image overlay

```
#Applying Perspective transformation matrix to overlay thermal image onto visual
IRWarp = cv2.warpPerspective(IRImg,tform,(RESOLUTION))
```

Figure 49 Thermal image before and after the perspective transformation

5.3 Colourmap Implementation

The implementation of a colourmap is necessary in the process of image fusion. The thermal images being greyscale, it contains only a single channel of 8-bit information and is thus unable to be overlaid onto the visual image which contains three channels of 8-bit information, Red, Green & Blue.

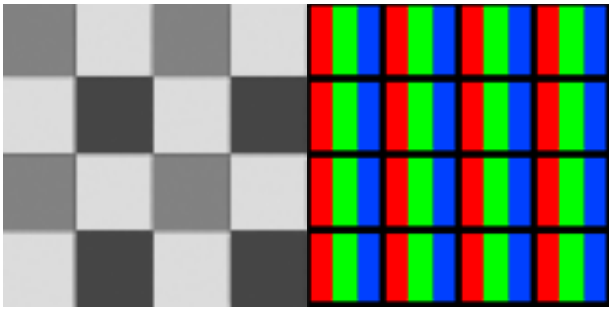


Figure 50 Single channel greyscale and 3-channel RGB

The two ways this could be done is by increasing the number of channels in the thermal data by applying a colourmap or reducing them on the visual image. Since information is important, it would be more ideal to retain all the information on the visual image and add pseudo information to the thermal image. A colormap converts greyscale single channel images into pseudo coloured three-channel image. This would allow the thermal image to be in the same data structure as the visual images as well as provide a clearer representation of temperature to the user as the RGB spectrum can generate high amounts of data range to interpret.

```
greyscale: array([[147, 147, 140, ...,  77,  80,  83],
  special variables
  [0:145] : [array([147, 147, 140...ype=uint8), array([148, 145, 142...ype=uint8.
  dtype: dtype('uint8')
  max: 231
  min: 13
  shape: (145, 200)
  size: 29000
colormap: array([[[52,  0, 93],
  special variables
  [0:145] : [array([[52,  0, 93],...ype=uint8), array([[55,  0, 93],...ype=uint8…
  dtype: dtype('uint8')
  max: 222
  min: 0
  shape: (145, 200, 3)
  size: 87000
```

Figure 51 displaying the array structure of a greyscale image, and a coloured image

For each singular value in the greyscale image, it has been mapped onto a 3-channel RGB value. OpenCV contains a predefined set of colourmaps within its library.

Class	Scale
COLORMAP_AUTUMN	
COLORMAP_BONE	
COLORMAP_COOL	
COLORMAP_HOT	
COLORMAP_HSV	
COLORMAP_JET	
COLORMAP_OCEAN	
COLORMAP_PINK	
COLORMAP_RAINBOW	
COLORMAP_SPRING	
COLORMAP_SUMMER	
COLORMAP_WINTER	

Figure 52 List the colourmaps available on the OpenCV library [34]

Colormaps can easily be implemented by using the cv2.applyColorMap function

```
colourMapImg = cv2.applyColorMap(greyscaleImg, cv2.COLORMAP_HOT)
```

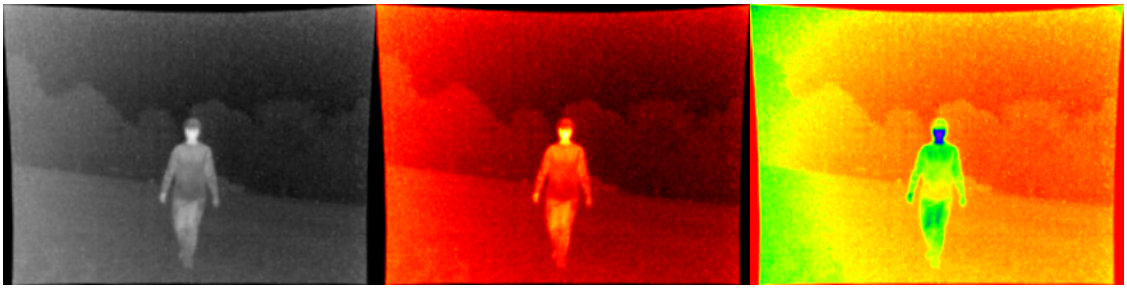


Figure 53 Applying the 'hot' and 'rainbow' colourmaps to a thermal image

Code Excerpt 9 Applying the transformation matrix for the image overlay

```
#Applying a colourmap to the calibrated & transformed thermal image
IRColor = cv2.applyColorMap(IRtform, cv2.COLORMAP_HSV)
```

A custom colourmap was also developed through the process. This was done by specifying a 3x256 array to map the greyscale values to using an online colourmap tool.

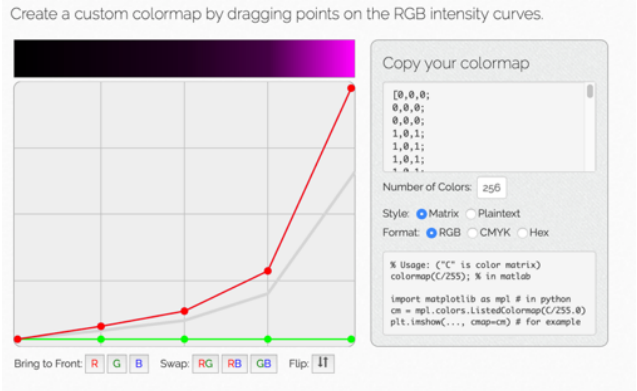


Figure 54 Generating a customised colourmap array using an online generator[3]

[3] The colourmap generator can be found at: https://jdherman.github.io/colormap/

By copying and saving the colormap data into a text file with a relevant name, such as *'purpColCustom.txt'* in this case, the file can be imported and run through the *'customColorMap'* function developed. This loops through every point in the image replacing its greyscale value with the relevant intensity values from the colourmap.

Code Excerpt 10 implementation of the custom colormap

```
#Importing the custom colourmap from the working directory
purpMap = np.loadtxt('purpColCustom.txt', delimiter=',').astype(int)
def customColorMap(img, LUT = purpMap):
    #generating a new template for image
    newImg = np.zeros((img.shape[0],img.shape[1],3)).astype(np.uint8)

#looping around the y and x axis to replace greyscale points with an RGB colormap
    for y in range (0,img.shape[0]):
        for x in range (0, img.shape[1]):
            newImg[y][x] = LUT[img[y][x]]
    return newImg
```

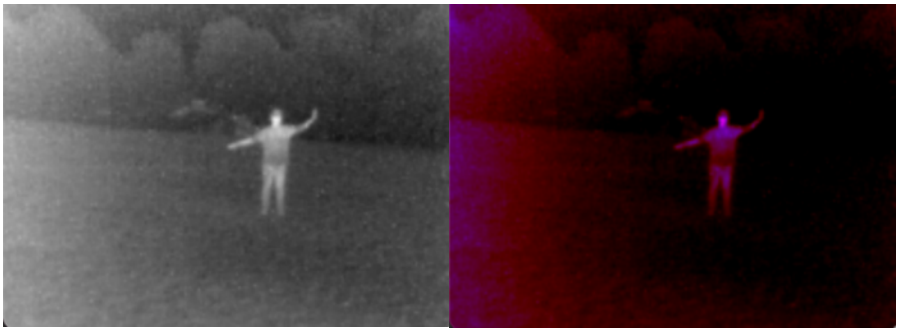


Figure 55 Application of the custom purple colormap

5.4 Fusion

Finally, image fusion is applied using the *'cv2.addWeighted function'* which overlays two images on top of each other. It contains an alpha and beta value to act as a transparency mixer for the background and the foreground image respectively.

This was implemented into an *'imageFusion'* function within the *'droneCam'* script function to fuse the images together.

Code Excerpt 11 imageFusion function script to fuse two images

```
def imageFusion(background, alpha, foreground, beta):
    if background.shape == foreground.shape:
        ## Overlays a background and foreground image onto each other
        fusedImg = cv2.addWeighted(background, alpha, foreground, beta, 0)
        return fusedImg
    else :
        print('[ERROR] Shape of background image and foreground are not the same')
        print(f'Background shape is :{background.shape}')
        print(f'Foreground shape is :{foreground.shape}')
```

Figure 56 Samples of fusion where the visual image is the background the thermal image is the foreground. Left: alpha = 0.5, beta = 1, Middle: alpha = 1, beta = 0.5, Right: alpha = 1, beta = 1

5.5 Analysis - Sensor Fusion

5.5.1 Camera Enclosure Development and Analysis

An iterative process was followed through the development of the camera case, initially it was built to encapsulate just the PiCam and the SeekPro

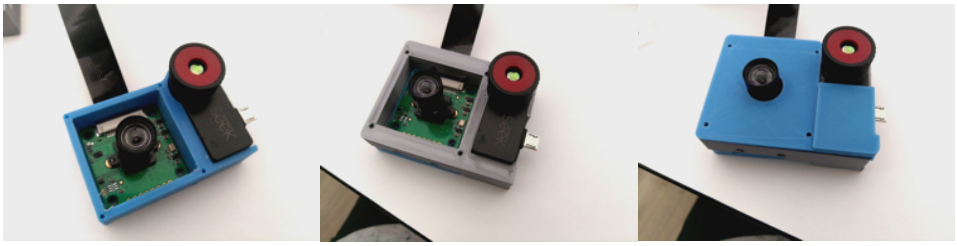


Figure 57 Iterations of the PiCam and SeekPro enclosure

The first and second enclosures were the simplest. They were designed to allow the cameras to slot into place. There was no further support for the SeekPro camera and so it would frequently fall out of the case. The third iteration introduced a front plate for the cameras to better protect the PiCam's PCB as well as supporting the thermal camera a bit better. Some holes were placed into the bottom of the third iteration with the intention of placing it onto the purchased gimbal, however, it was quickly identified that this case was too wide to be placed onto the gimbal.

Figure 58 Final camera enclosure design, before, during and after assembly

The final iteration of the enclosure was developed to hold a third camera as well as reducing the width of the enclosure, this was done by placing the slot for the SeekPro over the position of the PiCam's PCB along with a slot for the WebCam to be placed on top of that. This design came at the expense of thickness of the enclosure; however, this does not restrict the movement of the gimbal when installed. The groove at the bottom was developed to sit within the seating slot of the gimbal.

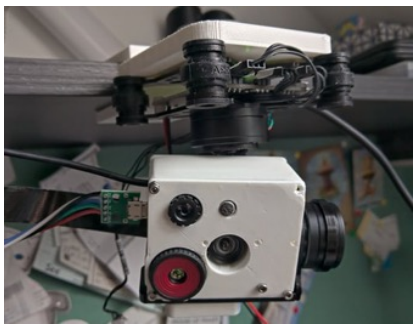


Figure 59 Camera enclosure installed onto the gimbal

This just fits well enough within the gimbal, however since in operation the camera should be placed horizontally, the tight tolerance should not cause any operational issues.

5.5.2 Camera Calibration Analysis

Only the PiCam and the SeekPro cameras went through the procedure of calibration as these would produce the images that would be fused together. There is no benefit in calibrating the WebCam images at this stage since there is nothing for it to overlay onto, it was thus dismissed from the calibration procedure immediately.

PiCam Calibration

When the PiCam went through the procedure of calibration the relevant matrices that were generated were:

$$\text{PiCam Calibration Parameters: } [0.00457 \quad 1.21 \quad -0.0173 \quad 0.0374 \quad -2.643]$$

$$\text{PiCam Camera Matrix: } \begin{bmatrix} 345 & 0 & 160 \\ 0 & 342 & 126 \\ 0 & 0 & 1 \end{bmatrix}$$

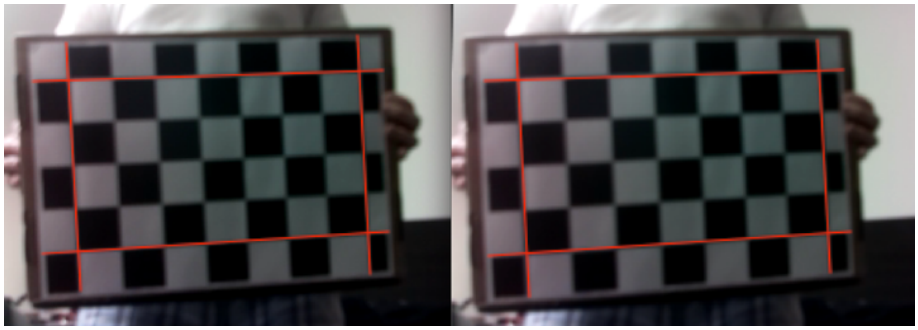


Figure 60 PiCam image before and after calibration

It can be noticed that there is not a significant difference between the image before and after calibration of the image. This is due to the PiCam lens being pre-tuned by the manufacturer to obtain a low distortion lens hence its low distortion label [21]. This makes applying camera calibration to this redundant and just add to the post processing. For these reasons applying calibration to the PiCam was also dismissed.

SeekPro Calibration

A set of 267 thermal images were captured for calibration, however only 15 images were successfully identified by the calibration algorithm. This is due to the temperature of objects around the room such as computers and monitors being reflected by the calibration board in some images, affecting the contrast between the squares on some of the sections.

It can be seen how much more contrast and uniformity there is on the calibration board within the images in Figure 62 compared to those on Figure 61.

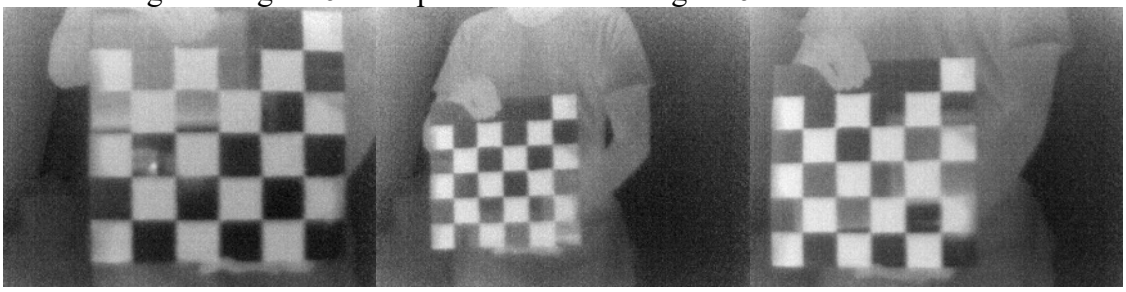


Figure 61 SeekPro images that were not successful in the calibration algorithm

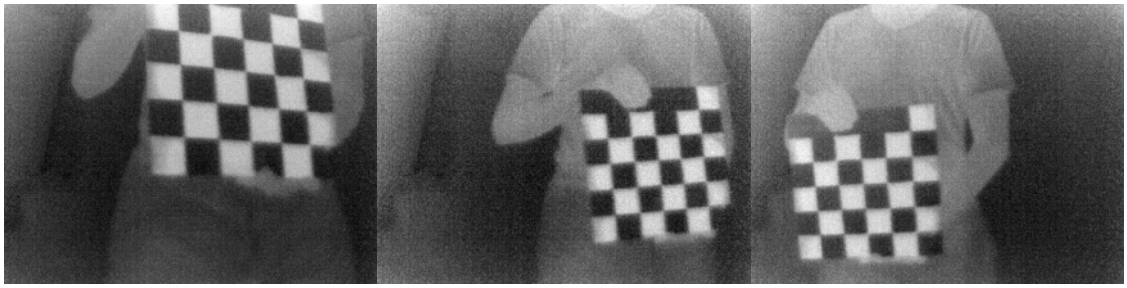


Figure 62 SeekPro images that were successful in the calibration algorithm

When the algorithm was completed the matrices for calibration were evaluated to be:

$$\text{SeekPro Calibration Parameters: } (-0.213 \quad -1.47 \quad 0.00360 \quad -0.00829 \quad 4.021)$$

$$\text{SeekPro Camera Matrix: } \begin{bmatrix} 549 & 0 & 187 \\ 0 & 556 & 193 \\ 0 & 0 & 1 \end{bmatrix}$$

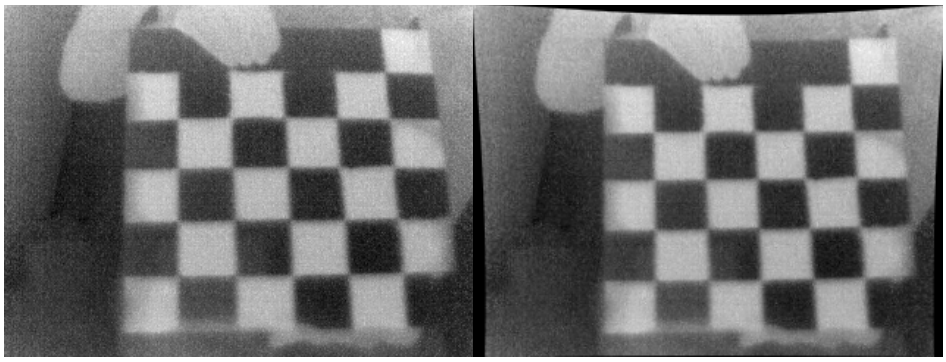


Figure 63 SeekPro image before and after calibration

Before calibration, there is a slight noticable amount of radial distortion along the sides of the board, this is corrected after the calibration achieving much more linear edges along the calibration board. Beyond this, there is also some tangential distortion which made the lower regions of the imgae appear less stretched and more uniform.

5.5.3 Image Fusion Analysis

The perspective transformation matrix to place the SeekPro images onto the PiCam images was identified and applied to the calibrated thermal images.

$$perspective\ transformation\ matrix = \begin{bmatrix} 6.56\text{x}10^{-1} & -0.0196\text{x}10^{-2} & 42.9 \\ -1.35\text{x}10^{-2} & 0.0619\text{x}10^{-1} & 47.9 \\ -4.01\text{x}10^{-9} & -1.48\text{x}10^{-4} & 1 \end{bmatrix}$$

Figure 64 Applying the perspective transformation onto the thermal image

An indoor test was then completed to observe how well the calibration and perspective transformation was done and thus the success of the sensor fusion. This was done by heating up the thermal camera calibration board and standing with it at distances of 0.5m intervals to see how well the thermal performs. This used the OpenCV rainbow colourmap with an alpha value of 1 and a beta value of 0.6.

Figure 65 Sensor fusion test from 0.5m to 3m at 0.5m intervals. Alpha = 1, Beta = 0.6, Colourmap = Rainbow

In Figure 65, the hot areas in the thermal image are observed as the blue regions. At distances less than 1m, the fusion of the images is not very good, this is due to the large parallax error there is in the cameras at this distance. As the distance increases beyond 1m, the error significantly reduces, and the image overlay performs well, consistently overlaying the hot blue regions of the thermal image onto the black regions on the visual image.

The OpenCV colourmaps were tested to observe what kind of overlays would be suitable to be placed ontop of a visual image.

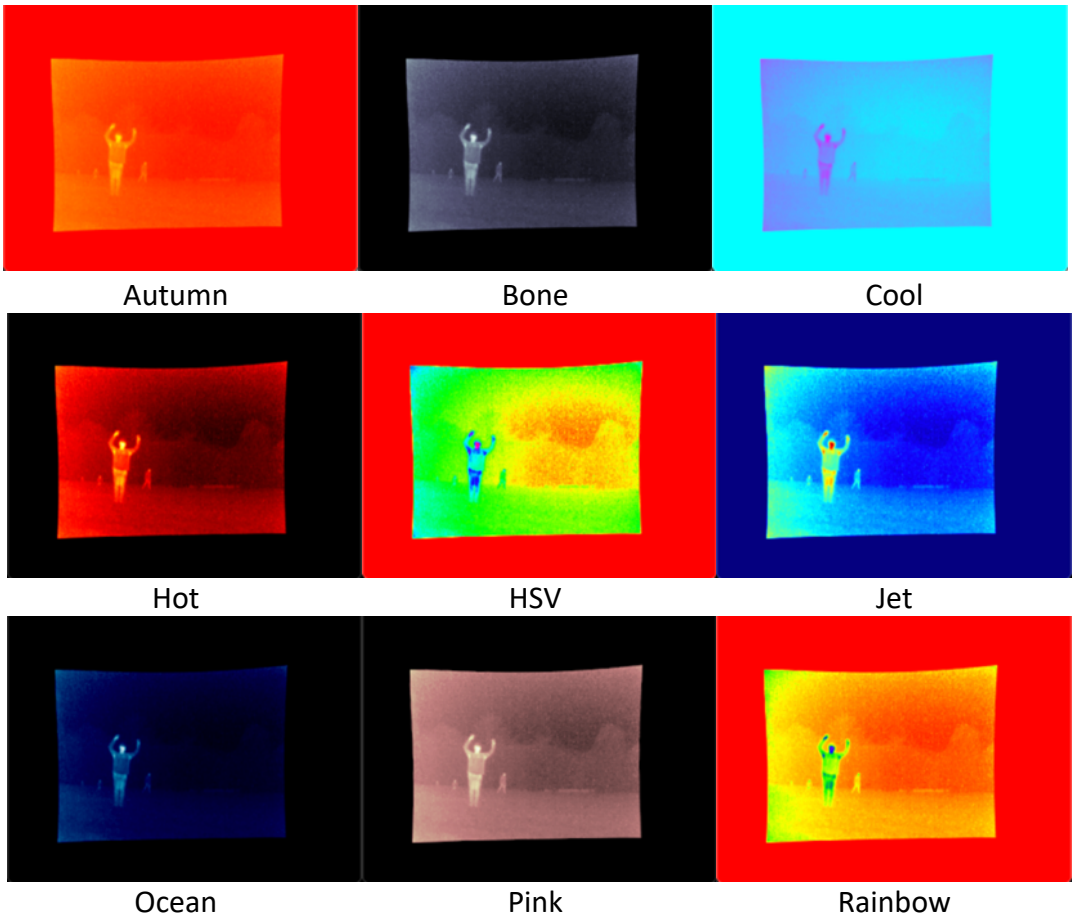


Figure 66 Applying a range of OpenCV Colourmaps to a sample thermal image

Rainbow, HSV and Jet are multicolour colourmaps making these quite good at distinguishing between temperature ranges. The use of a colour for low temperatures is undesirable however, since it will blow out the rest of the visual image when an overlay does occur and they are not the temperatures that need to be identified. The Hot, Bone and Pink colourmaps uses black when the temperature is low however, it would be desirable if the colour range began to develop at a higher temperature range to reduce the less important information within the image.

Taking these factors into account, a customised colormap was made to identify only the hottest regions in an image and ignore the coldest regions. Purple was used as it is quite an unnatural colour and would not typically be seen in the wilderness.

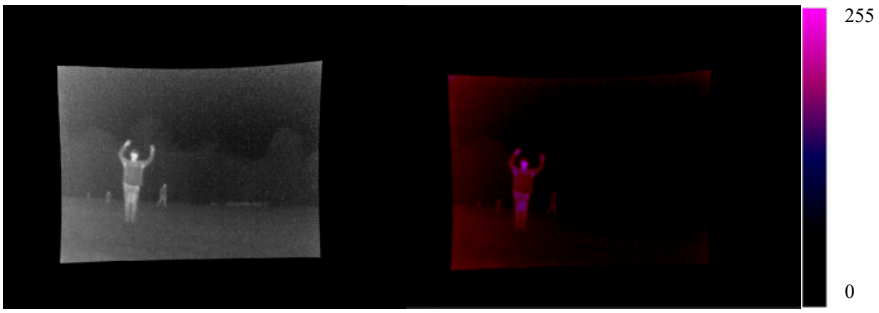


Figure 67 Application of a Custom Colourmap

The colourmap generated does a good job of removing a lot of the landscape regions within the thermal image. Furthermore, the heat of the people in the background can still be seen showing that this does a good job of identifying the hotspots at close and far distance. The heating of the camera system has caused the SeekPro lens to warm and caused the sides of the image to slip into the regions causing some purple to show on the boundary the image.

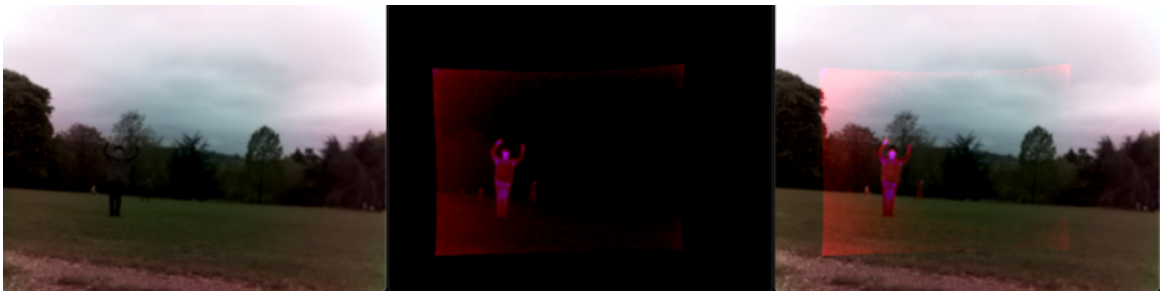


Figure 68 PiCam image, SeekPro image and Fused image

Once the image fusion is applied, the benefits of this method can really be observed. In the PiCam image, it is slightly difficult to identify the person upfront due to the bad lighting and can see no other people beyond that. The thermal image captures the heat from the person at the front as well as two people that are far back. Once fused, it is much easier to identify all of the people within the image whilst still having lots of visual information about the surrounding area. The Alpha and Beta values were both set to 1 as there is no particular benefit to setting either to a low value in this kind of condition.
This can be seen more clearly in Figure 69.

Figure 69 Fused PiCam and SeekPro Image with custom colourmap

Field Testing Data

More samples were taken from the field test on the cold day and processed to apply the customised colourmap to evaluate how well the image fusion would work at different distances, these images were taken from the samples on the cold day as this is when the visual imaging is poor and the thermal imaging is good.

Figure 70 Testing image fusion 10m distance

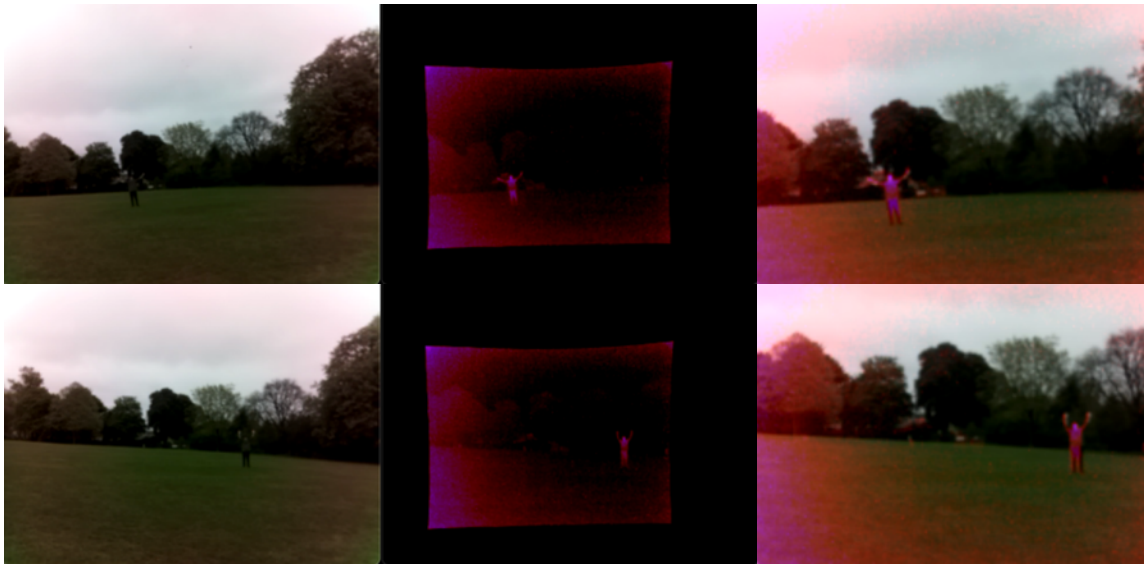


Figure 71 Testing image fusion 20m distance

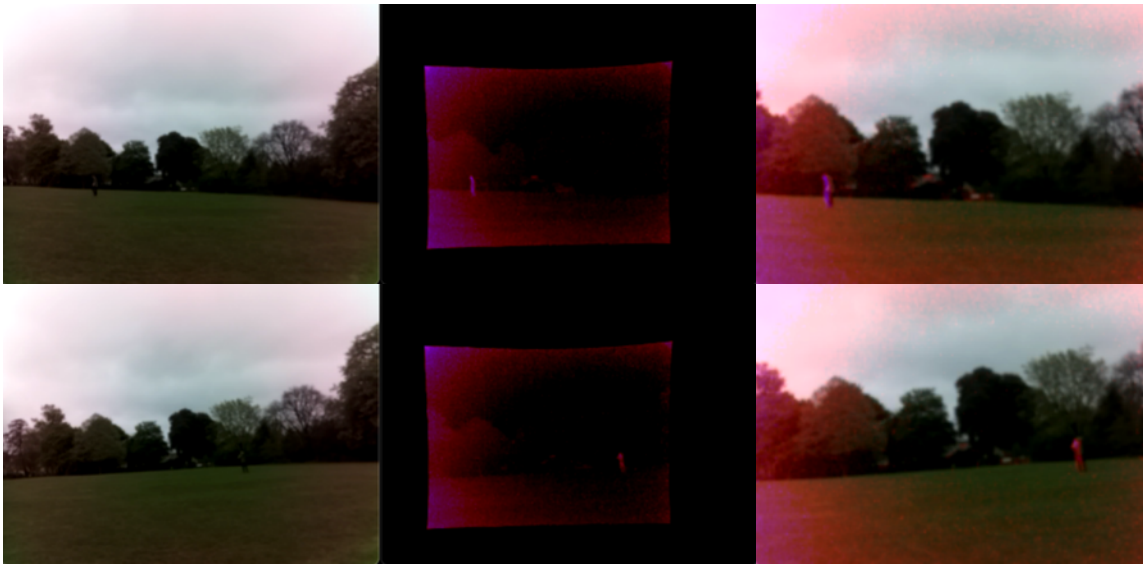


Figure 72 Testing image fusion 30m distance

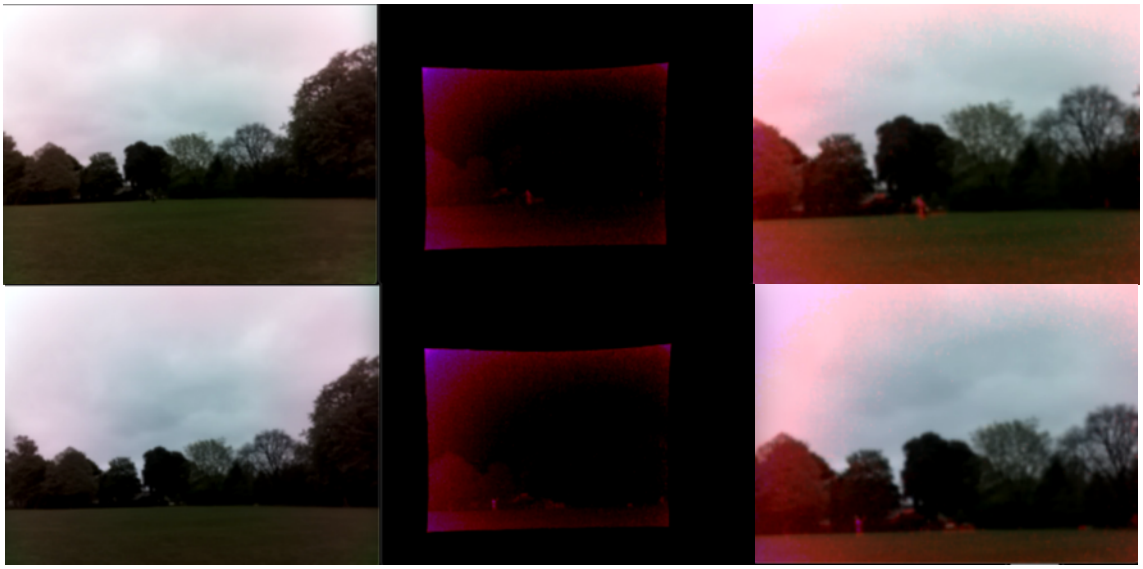


Figure 73 Testing image fusion 40m distance

A sample of the hot day was also processed to show how the fusion would perform in this scenario.

Figure 74 Testing the image fusion on the hot day

In the images from the field test, it can be observed that the person is quite visible in the visual images, only up to 20m, beyond this distance, it starts to be quite difficult to identify the person visually, however, thermally the person in the image is very identifiable, all the way up to 40m. It can be seen that when the images fuse together the person's hotspot from the thermal image overlay and identifies the person quite well in the image. This is a very successful result for this procedure and can be utilised in a very powerful way under the right conditions. Unfortunately in the wrong conditions it can be detrimental as can be seen in the final test, Figure 74, the fusion of the images on the hot day was just overlaid with the purple colour.

5.6 Discussion – Sensor Fusion

In the steps preceding the final fusion of the visual and thermal images, the SeekPro camera went through a calibration procedure followed by a perspective transformation.

In the process of camera calibration, only 15 out of 267 thermal images that were accepted for the calibration procedure, this was likely due to the temperature of the surrounding objects affecting the contrast in the calibration board, this makes it very possible for the calibration of the SeekPro to be even better. The calibration board could potentially be improved by developing a board to use a system that is actively heating and cooling to allow for a very constant contrast among the respective checkboard sections. Despite this the resulting images came out well with a visibly reduced radial as well as some tangential distortion.

The next step in the procedure was the perspective transformation. Although the procedure is a bit tedious, it is easy to follow. The method for obtaining points was not ideal for this process and could be done better with the development of a dedicated large board with a thermally visible pattern on-top of it. Despite this, the perspective transformation was still very successful which was seen in the close-range distance tests and again in the field tests.

It was found that the OpenCV colourmaps were too overwhelming when placed onto the thermal images creating intensive colour for all regions within the image, because of this a custom colourmap was designed and implemented to display only at the hotter regions. The results provided very clear regions of hotspots when the person would stand within the frame, this however did not eliminate the effects generated by the SeekPro's lens warming up and so the edges were purple as well.

Finally, the overlay of the thermal image on-top of the visual image proved to generate some really clear results where, despite not being able to identify the person within the visual image at far distances, a high intensity purple region formed in the same place making it very easy to see the person. The fusion of the images only works conditionally however, only operating well in colder environments due to the large amount of heat across the image in hotter environment. This is not necessarily an issue because the visual images typically produce the clearest results within sunny brighter day making it easier to identify the person from a farther distance in these cases.

6 Region of Interest (ROI) Modelling

The purpose of a ROI model is to autonomously identify potential regions a person would be within an image. The PiCam and SeekPro would feed their image data into the model which would output a directional signal to the gimbal causing it to move and allowing that region to be the new centre of the image. This section aims to tackle SR1 from the perspective of internal system identification as well as SR9 which requires the system to be autonomous.

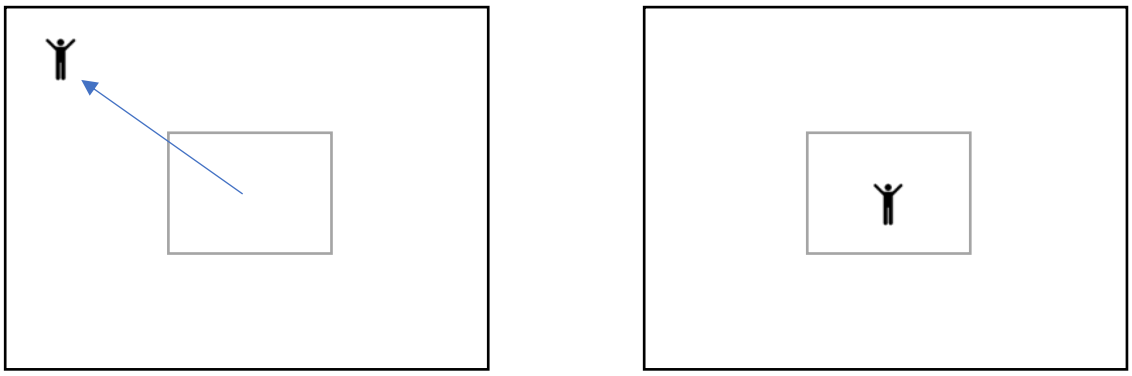


Figure 75 Example of the operating ROI model

Within this project, a set of experimental saliency models were implemented and used to process the data capture from the field tests. These saliency models are built up to try and emulate features of the human's attention system and creates saliency maps that identify the most interesting positions within an image.

Two libraries were used in this project for the saliency work. The first of which was from within the OpenCV library and the other was the PAIRML Saliency library which uses more computationally intensive processes by applying deep as well as convolutional neural networks to create the saliency maps.

6.1 OpenCV Saliency Models

The OpenCV library contains three types of saliency algorithms within it. These are the motion, objectness and static saliencies.

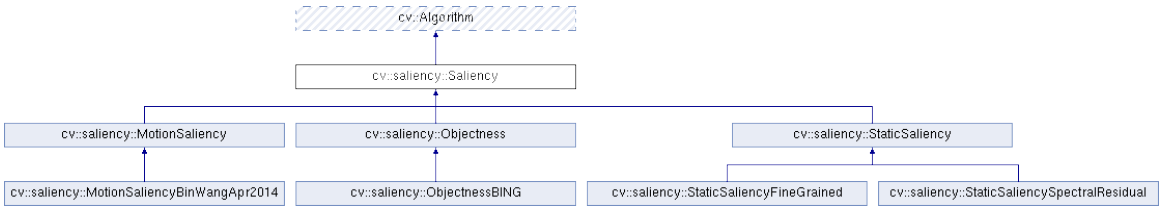


Figure 76 Breakdown of the different saliency models available in OpenCV [35]

Motion saliency

This is used to identify moving objects in an image. In a search and rescue scenario, it is unlikely the person on the ground will be moving very much, especially relative to the image, furthermore the environment will also consistently be changing thus will cause large regions of saliency in flight, these factors have meant this method was dismissed.

Objectness saliency
This identifies as many objects as it can within the image to generate a set of bounding boxes around them. It does not produce a traditional saliency map like the other saliency algorithms analysed.

Static saliency
This identifies the most interesting regions within an image by looking for visually different regions in the image, this tries to emulate the way that the human visual system identifies interesting regions within a static image.

The static saliency models in this library include the Fine-Grained model and the Spectral-Residual model. The Fine-Grained saliency model was developed as a feature extraction mechanism to help with human detection in a range of scenarios. The Spectral-Residual model was developed as a much faster saliency model which analysed the colours of local regions with the rest of the image within the frequency spectrum range, allowing it to quickly identify changes in contrast and colours quite well.

The OpenCV saliency script that was developed contains three classes within it that used both the static saliency models as well as implementing a very basic location identification function.

'*findSaliencyFineGrained':* used to apply the Fine-Grained saliency
'*findSaliencySepctralResidual'*: used to apply the Spectral-Residual saliency
'*contourProcessing*': used to identify the ROI generated by the saliency to redirect the position of the camera system.

Code Excerpt 12 Fine-Grained Salience class

```
class findSaliencyFineGrained():
      def __init__(self):
            ''' Initiating the saliency model to be use for this process'''
            self.saliency = cv2.saliency.StaticSaliencyFineGrained_create()

      def getSaliency(self,img,threshperc):
            ''' function to capture the saliency in a given image and output a
             threshold map based on a given threshold percentage desire'''
            #Grabbing the saliency and converting it into a greyscale integer value
            saliencyMap = np.array(255 * self.saliency.computeSaliency(
                              img)[1],dtype="uint8")
            #Creating a threshold percentatge to grescale conversion
            threshpercconv = np.int((threshperc/100)*255)

            #applying thresholding the top 'threshperc' of the image
            threshMap =  cv2.threshold(saliencyMap,  threshpercconv, 255,
                                       cv2.THRESH_BINARY)[1]
            return saliencyMap, threshMap
```

The '*findSaliencySepctralResidual'* class is identical to this with the slight change in initiation using:

```
self.saliency = cv2.saliency.StaticSaliencySpectralResidual_create()
```

The ‘getSaliency’ function in these classes require an input image as well as a threshold parameter. The outputs of this function generate a saliency map as well as its threshold map.

These functions can be applied by running the relevant saliency function as well as providing an image and threshold to it.

Code Excerpt 13 Running the saliency models to generate a saliency and threshold map

```
import saliencyCV2
threshold = 65
#Generating a saliency map and its threshold map for a given image
(saliencyMap,threshMap) = findSaliencyFineGrained.getSaliency(img,threshold)
(saliencyMap2,threshMap2) = findSaliencySpectralResidual.getSaliency(img,threshold)
```

These functions output the saliency map generated to show the points of interest in the image. The brighter the point in the saliency map the more interesting it is perceived to be by the algorithm, a threshold map is also developed thus a threshold variable is required for the function as well.

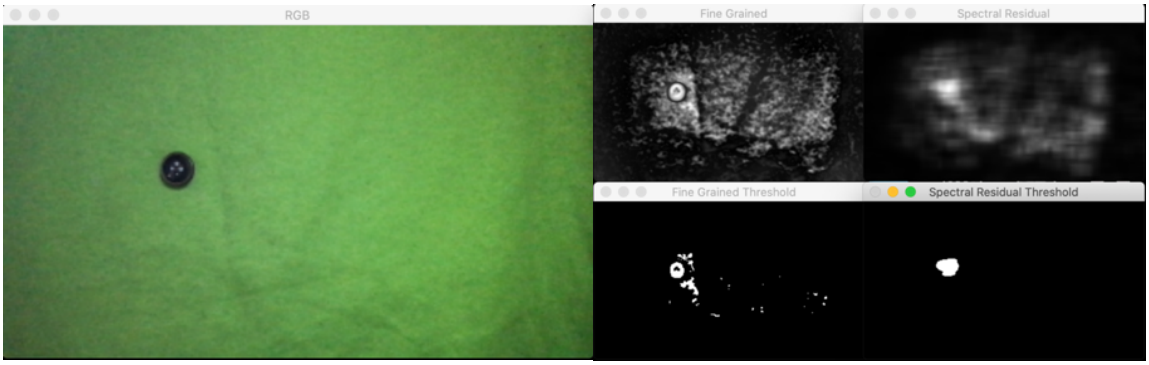


Figure 77 Displaying an RGB image with an example output of the Fine-Grained and Spectral-Residual image with thresholds at 65%

Following on from the implementation of this saliency model, a method of detection needs to be implemented for the future applications onto a gimbal system. This is done by using contouring functions on the images. Contours are made using image segmentation techniques to identify regions enclosed by boundaries [36]. This process can only be done using the thresholded images.

Code Excerpt 14 Generation the location of the bounding boxes in the thresholded images

```
def getContours(self,threshMap, origImg):
      '''Function to find all the thresholded regions ie Contours in a saleint image
      and to identify the possible ROI'''
      # Finding the countors in the image
      cnts = cv2.findContours(threshMap, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
      cnts = imutils.grab_contours(cnts)
      #Sorting and numbering the Contours starting from the bottom of the image
      method = "bottom-to-top"
      (cnts, boundingBoxes) = contours.sort_contours(cnts, method=method)
      #Labelling the image with contour positions
      for (i, c) in enumerate(cnts):
            sortedImage = contours.label_contour(origImg, c, i, color=(240, 0, 159))
      return sortedImage, boundingBoxes
```

Once these contours are found, a list of bounding boxes are identified which are used to decide the direction in which the gimbals should move towards to centre the ROI. This process is done within the 'getContourPosition' function which creates a tally of information about all the bounding boxes in an image, this identifies which region is perceived as the most interesting. The code for this can be found in the appendices at section 13.4

All the contouring functionality is done within the 'contourProcessing' class. Once fully run, the algorithm identifies the regions of interest depending on the threshold map provided to it, this then outputs a contoured image showing the bounding boxes on the original image and also outputs the direction in the image the region of interest is in.

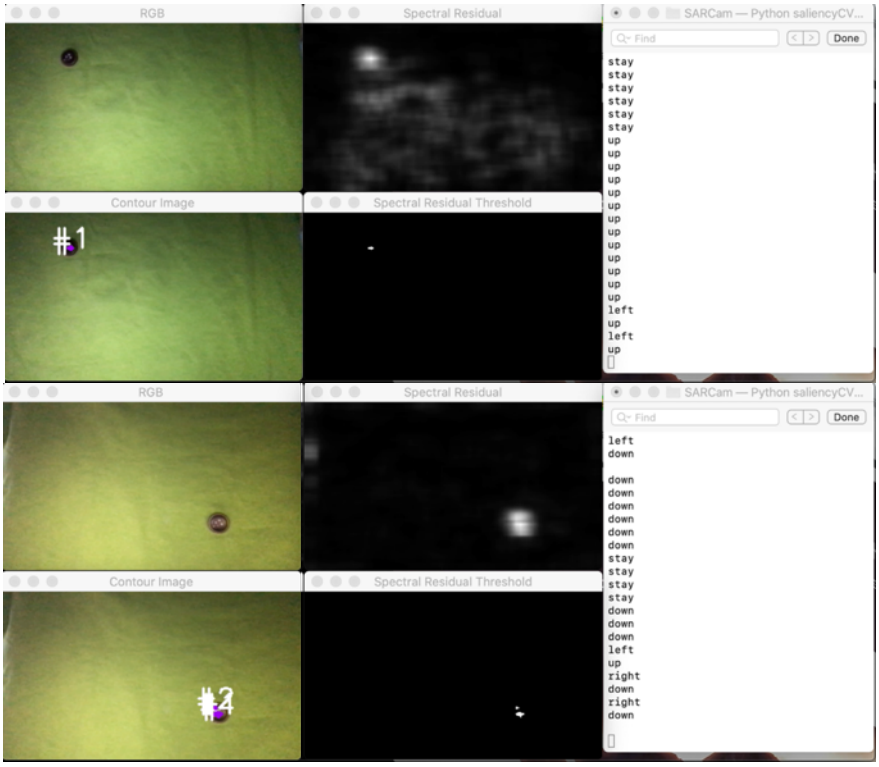


Figure 78 Samples of successful contouring as well as the identified ROI position

For this to be implemented into the final code, the script is simply imported, and the functions can be called in this way.

```
import saliencyCV2
threshperc = 50
# to get the Fine-Grained salience
saliencyMap, threshMap = saliencyCV2.findSaliencyFineGrained().
                                                    getSaliency(img,threshperc)
# to get the Spectral-Residual salience
saliencyMap, threshMap = saliencyCV2.findSaliencySpectralResidual().
                                                    getSaliency(img,threshperc)
# to get the outlined image and new and the bounding boxes
sortedImage, boundingBoxes = saliencyCV2.contourProcessing().
                                                    getContours(threshMap, img)
```

6.2 PAIRML Saliency Library

The PAIRML saliency library contains a set of different saliency techniques that use TensorFlow to identify directed objects of interest in an image. The saliency functions applied and tested included the SmoothGrad (SG), Vanilla Gradients (VG), SmoothGrad Guided Backpropagation (SGB), Vanilla Guided Backpropagation (VGB), SmoothGrad Integrated Gradients (SIG) and Vanilla Integrated Gradients (VIG) [37].

The details of these models are not fully understood as they are developed to utilise and optimise models of different types of convolutional as well as deep neural networks. They use standard network's architecture and apply slight modifications to them for improved results generating saliency maps of images based on the input model provided. The input model comes from the 'inception_v3' dataset which contains 1000 different types of objects but none are explicitly of a human, however, a close one was found to be a baseball player, model 981, and was forced as the prediction class for the algorithm so that it would be looking to identify a human shape [38].

The implementation of the code was done in a similar way as the example code on the saliency library documentation with some slight modifications to make it more applicable to the applications of the project [39]. These algorithms were not applied on the Raspberry Pi as this is a computationally expensive task, however, this was still experimented with as it may be useful for future potential applications. The full code for the algorithm can be found in appendices at section 13.5.

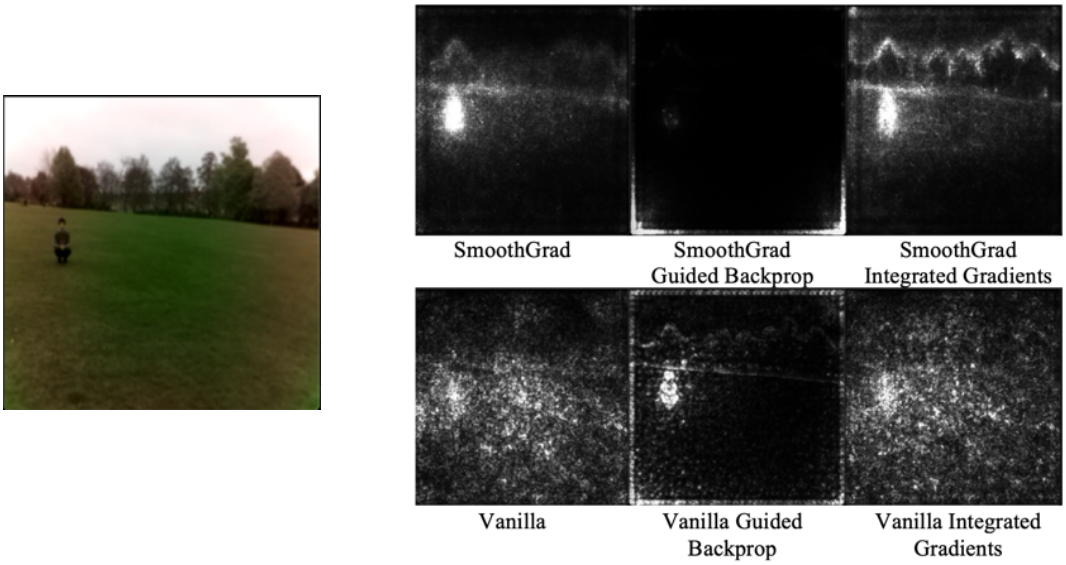


Figure 79 Showing a sample of the results generated from the PAIRML saliency models

6.3 Saliency Model Testing

In this section, the OpenCV and PAIRML saliency algorithms will be tested and analysed to identify which can have applications for the system. The data sets available to test will include some indoor testing for the OpenCV model and then applying them to the field test samples, furthermore, some sample from SARAA of real-life flights were also implemented to analyse how the algorithms would operate in those scenarios.

6.3.1 OpenCV Saliency Analysis

For the OpenCV saliency algorithms some indoor data was used as well as applying the field test.

Indoor Testing

The indoor testing was done by placing a button onto a green t-shirt to emulate a distinctive object in a green environment. This is an early test to ensure that the algorithms work for clear and obvious regions.

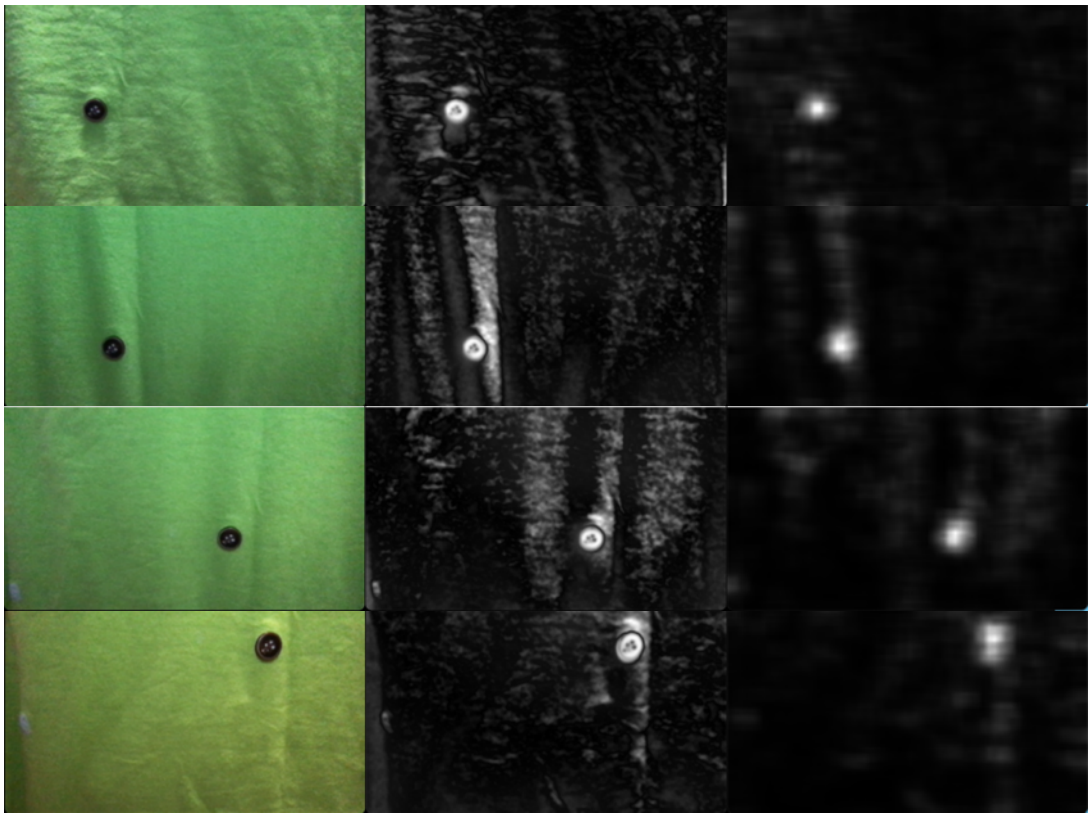


Figure 80 testing the OpenCV saliency model on simple images cases

It can be seen that the saliency model does a good job of identifying the button at different regions as it is moved along the camera frame. The Fine-Grained model does a very good job of identifying the button in the image but also generated noise at flatter regions which is not ideal. The Spectral-Residual saliency model generates a more uniform response from the input images and is less susceptible to changes in light intensity within the image. This makes the Spectral-Residual model more desirable in this idealised scenario.

With regards to the speed of the models, the Spectral-Residual model runs at a rate faster than the image capture ability of the Raspberry Pi taking about 0.02 seconds to run, however the Fine-Grained model ends up bottlenecking the system taking about 0.3 seconds to run causing a drop in the framerate of the system.

Outdoor Testing

The outdoor testing was done using the images captured from the field test. These show images at a range of distances with evaluation.

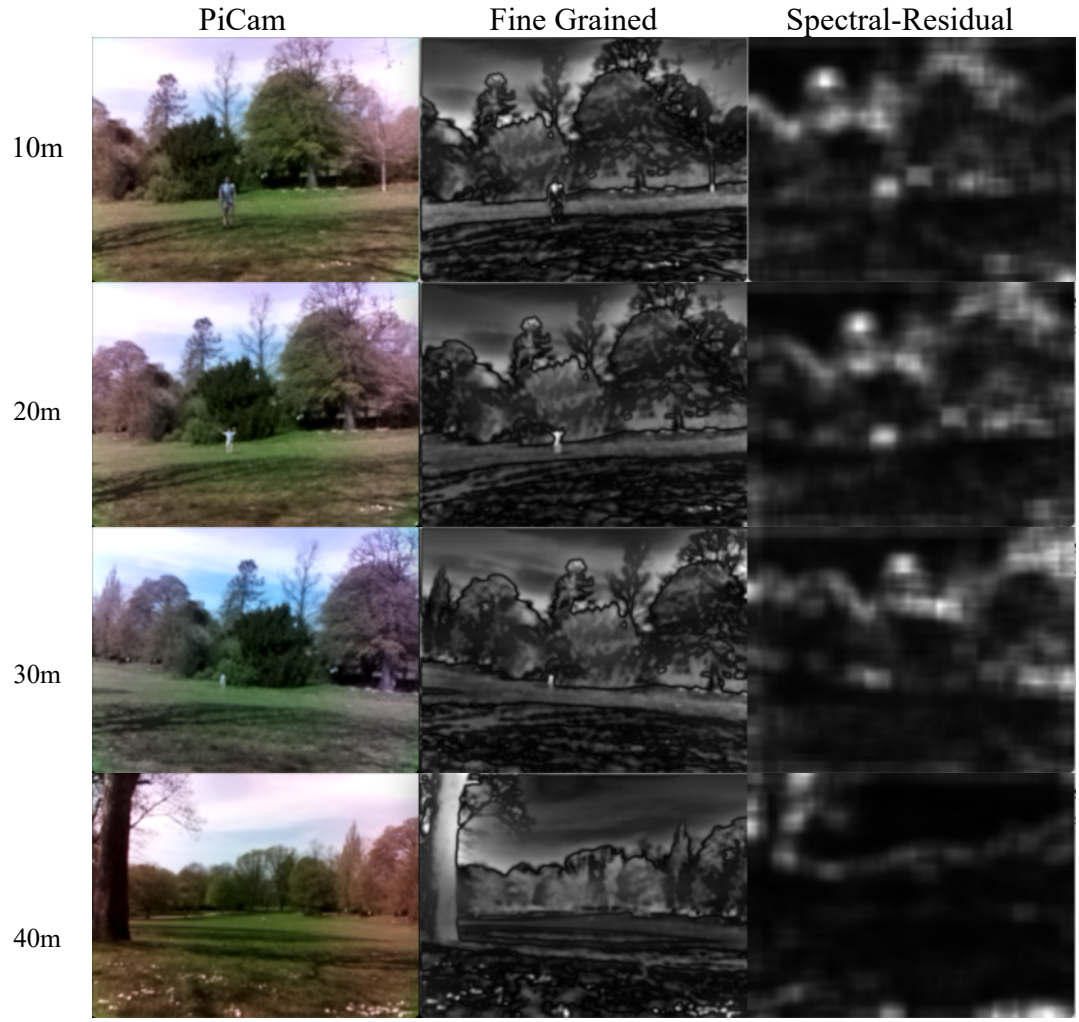


Figure 81 Fine-Grained and Spectral-Residual models applied to PiCam images of a person at different distances (m) from the field test on the sunny day

In the images in Figure 81, the Fine-Grained model does a really good job of identifying the person among the trees up to 30m and clearly picks out the position in which the person is standing. The Spectral-Residual model is also capable of identifying the region in which the person is standing up to this distance, however, it is very unclear by looking at the saliency map what the object being evaluated is. At 40m the person loses too much detail and the environmental change makes it difficult to compare the results.

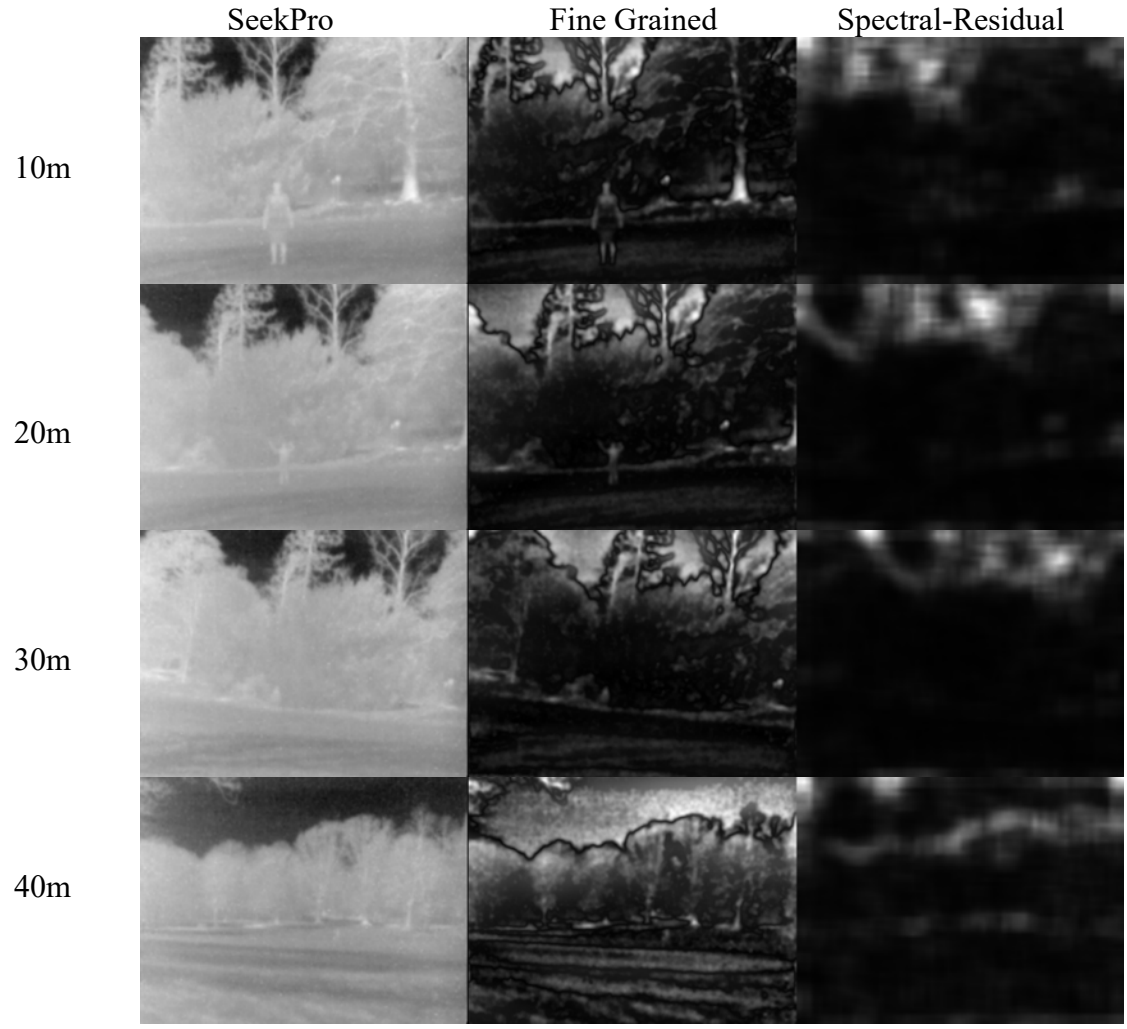


Figure 82 Fine-Grained and Spectral-Residual models applied to SeekPro images of a person at different distances (m) from the field test on the sunny day

In the images in Figure 82 the Fine-Grained model shows an outline of a person within the frame, these are clearest up to 20m but beyond that, the outline is difficult to see, although they are still visible, even up to 40m. The Spectral-Residual model does not identify the location of the person at all.

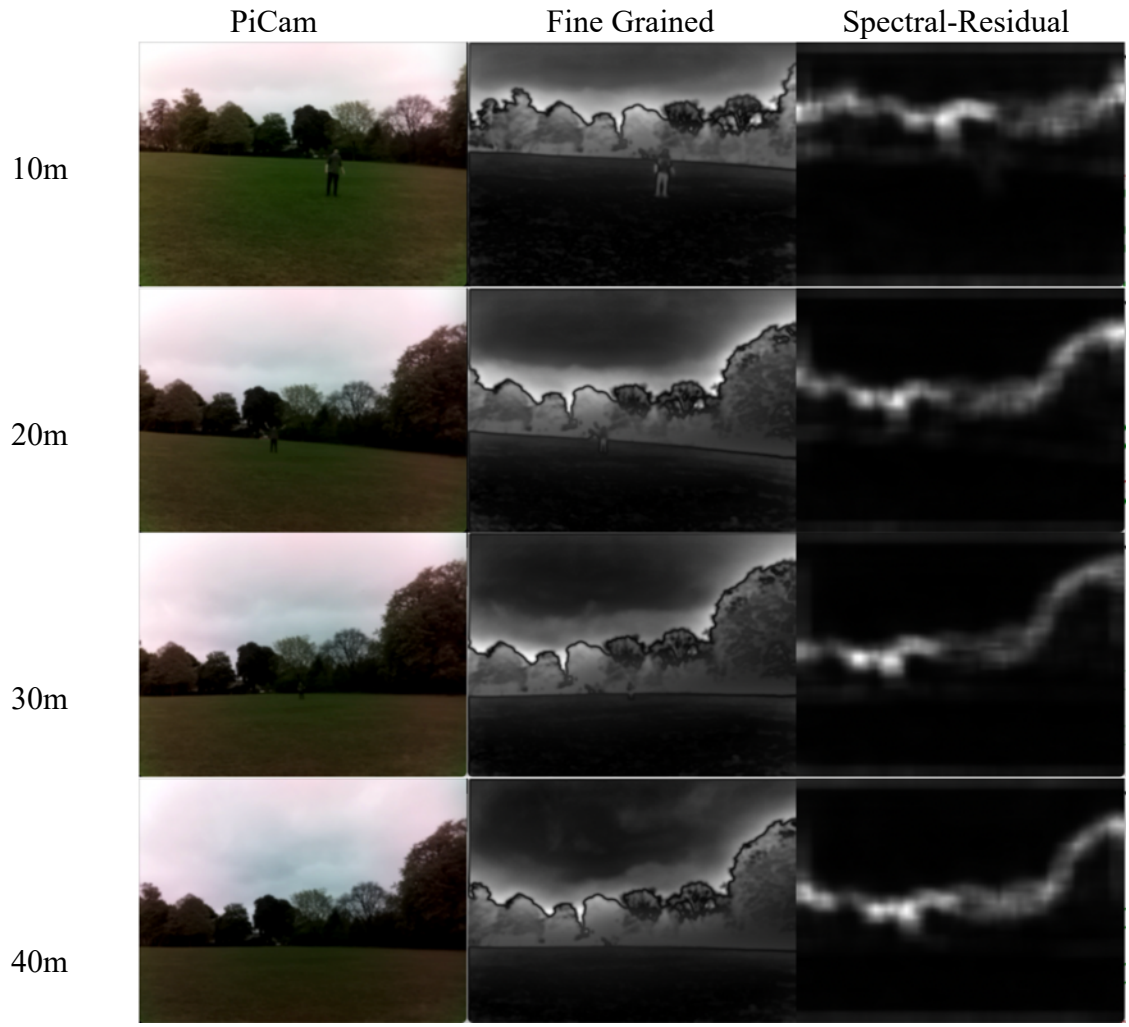


Figure 83 Fine-Grained and Spectral-Residual models applied to PiCam images of a person at different distances (m) from the field test on the cloudy day

In the images in Figure 83, the Fine-Grained model was able to identify an outline of the person at 10m, barely at 20 to 30m and not at all at 40m. The most salient parts of these images were along the treelines in these cases making the results of these images quite poor. In the Spectral-Residual model, the person was not identified at all throughout the process and in a similar way, the treeline was the most salient part of the image.

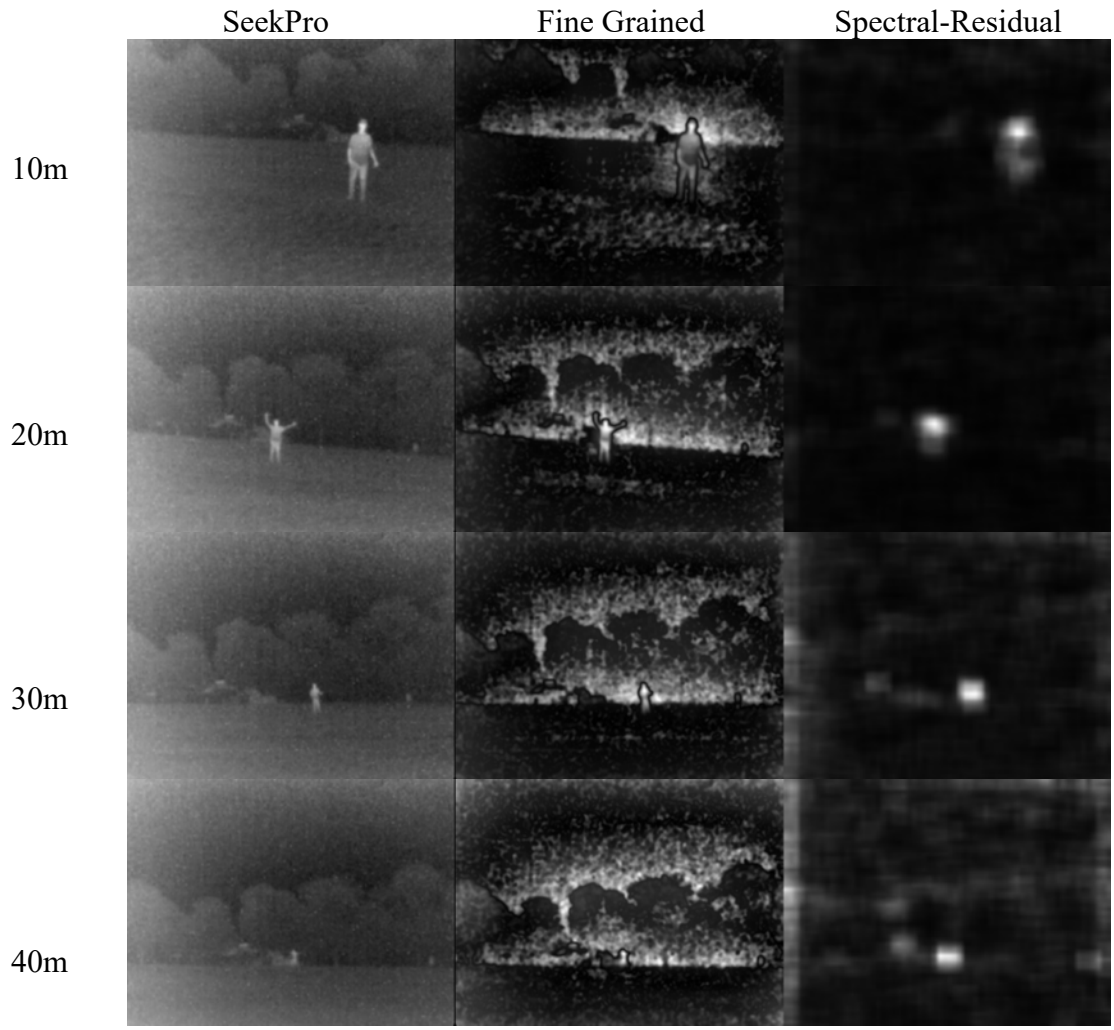


Figure 84 Fine-Grained and Spectral-Residual models applied to SeekPro images of a person at different distances (m) from the field test on the cloudy day

In Figure 84, the Fine-Grained model identifies the region in which the person is standing and can be identified at all the distances however there is lots of noise within the images produced. The Spectral-Residual model did a fantastic job in these images and very clearly located the person at all distances, it was noticed, however, that at 30m and 40m, some noise begins to generate throughout the image produced.

SARAA Testing

Using the Sample data that SARAA has provided, the algorithms were tested again on real life scenarios to evaluate their effectiveness, to remain analogous to the rest of the system, the sizes of these images were shrunk to 320x240 pixels.

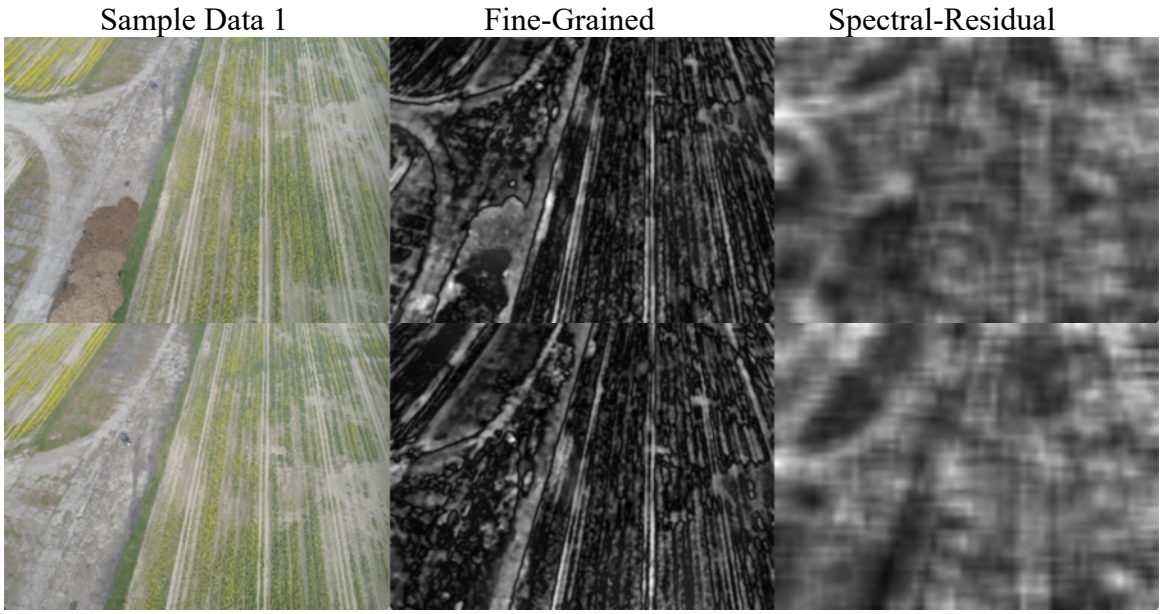


Figure 85 SARAA sample 1 - overhead image over a car and person in a field first is the full photo, second is a cropped sample applying Fine-Grained and Spectral-Residual models

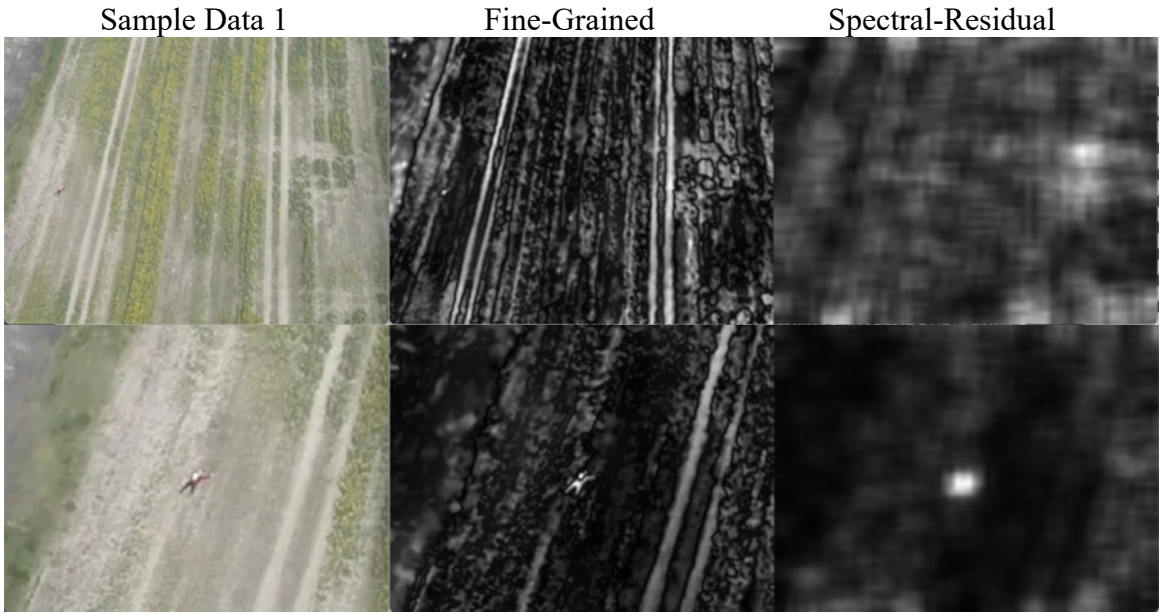


Figure 86 SARAA sample 1 – Further cropped image to capture only the person and re-applying Fine-Grained and Spectral-Residual models

In Figure 85 and Figure 86, the Fine-Grained model identifies a lot of detail in the images picking up the car from afar and the person from closer up but also with a lot of other features within the image. The Spectral-Residual model struggles to identify any of the objects of interest until much closer when the person lying down was a lot more visible.

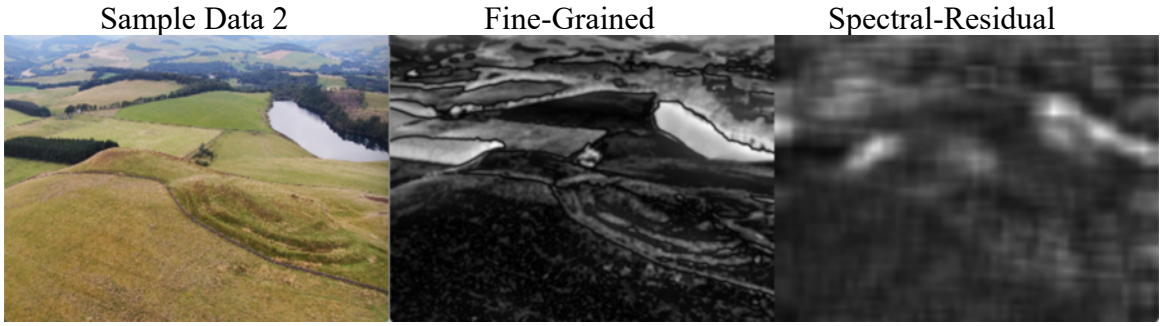


Figure 87 SARAA Sample 2 - People standing on-top of a hill with SG and VBG applying Fine-Grained and Spectral-Residual models

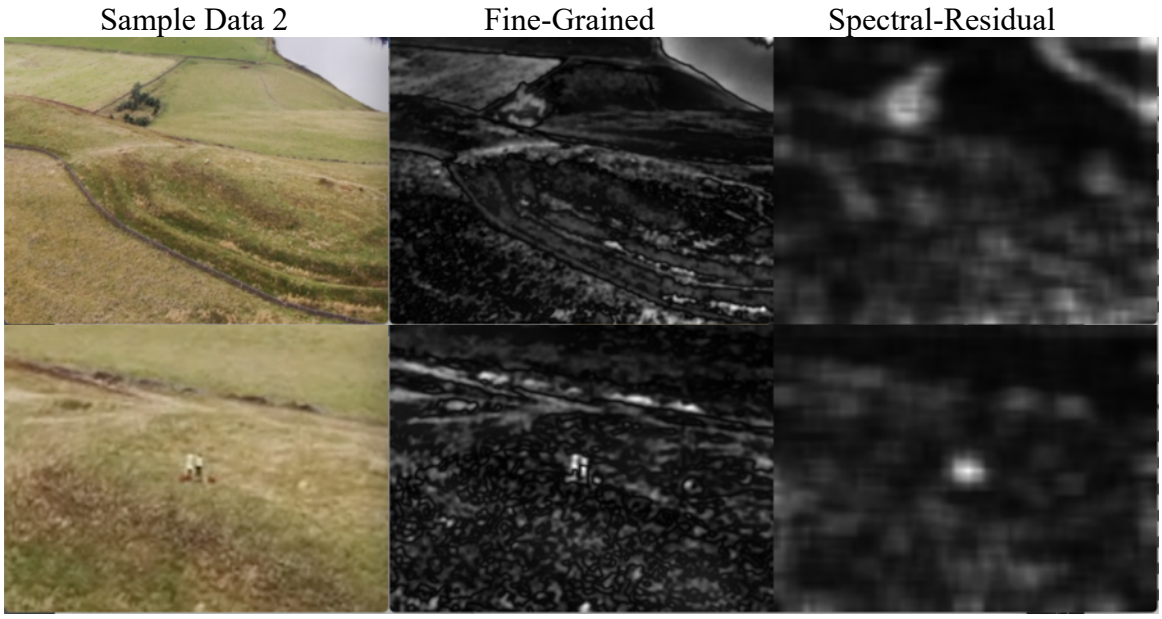


Figure 88 SARAA Sample 2 – Retesting the sample on cropped images applying Fine-Grained and Spectral-Residual models

In Figure 87 and Figure 88, both models struggled to identify the people standing on the hill until they were much more clearly visible.

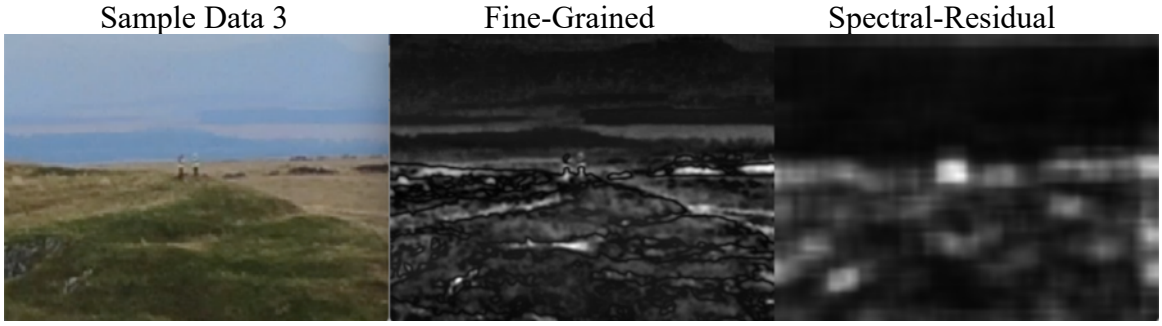


Figure 89 SARRA Sample 3 - People standing on top of a hill applying Fine-Grained and Spectral-Residual models

In Figure 89, the Fine-Grained model very barely identifies the people standing on top of the hill, but the Spectral-Residual identifies them quite clearly.

6.3.2 PAIRML Saliency Library Analysis

Much like the OpenCV saliency models, these saliency models were also tested to identify the most effective methods for application. Since these models are trained for a neural network, observations from an indoor test would not be very useful and was thus omitted. Outdoor tests and sample flight tests were still observed.

Initial Testing of the PAIRML Saliency library

An initial test was done to have a look at the outputs of the saliency models with some sample images from the outdoor test.

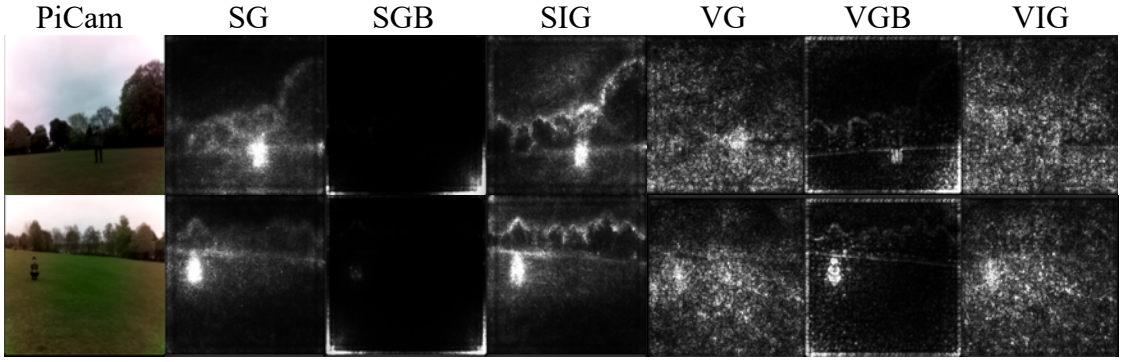


Figure 90 Applying a range of the saliency models to a couple image samples of a person outside.

Due to the consistently noisy results generated by the VG and VIG, they were dismissed from further testing as the large amount of noise would make it them difficult to use if post processing was done. SGB, did not generate any salient regions apart from the edges of the model and thus it was also dismissed.

Further testing was done to evaluate the processing time for the different models.

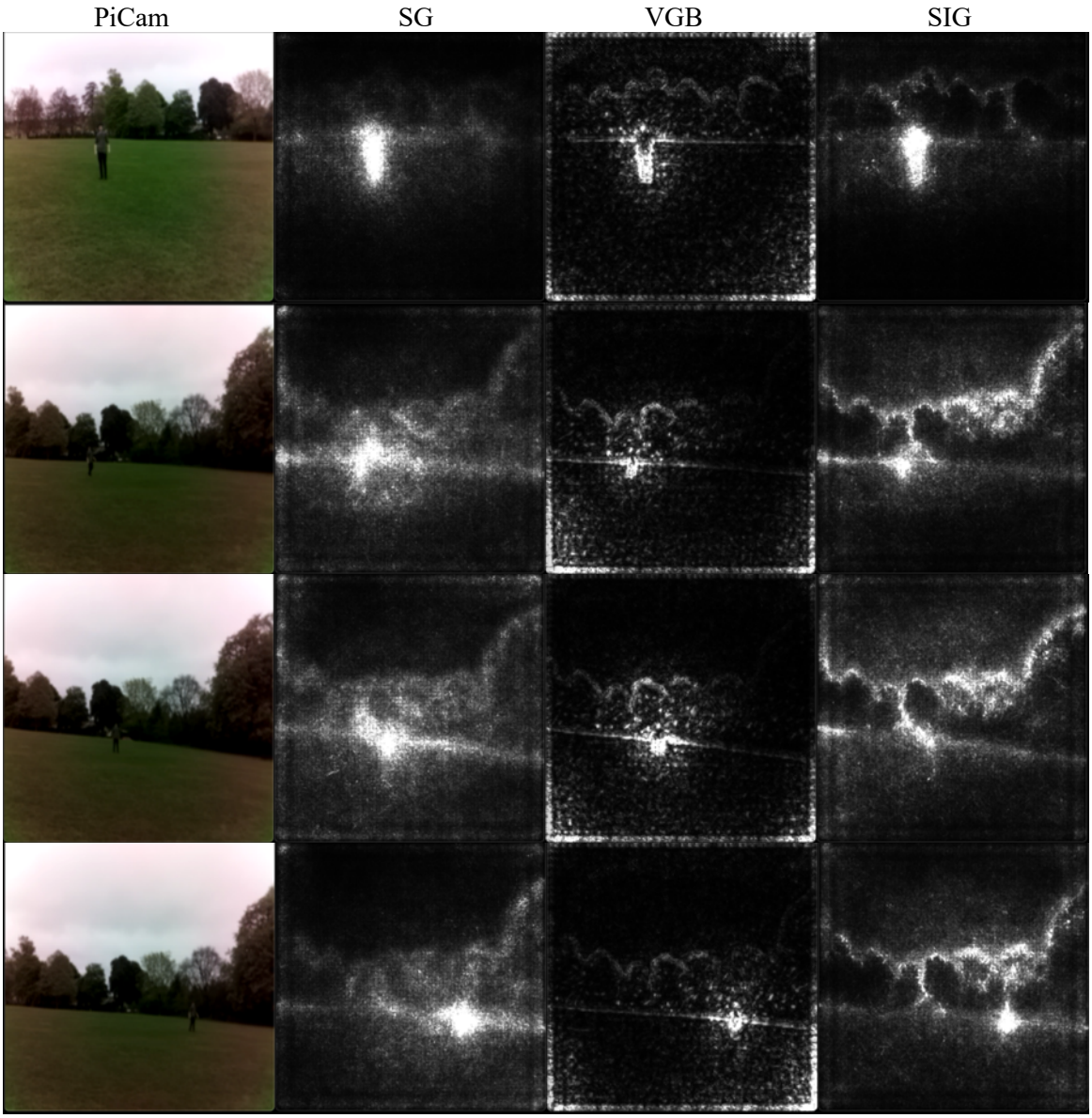


Figure 91 Further tests on the TF saliency algorithm

The SG saliency model consistently performed under 10 seconds of running time and the VGB consistently performed under 5 seconds, however, the SIG consistently took 200+ seconds to complete its computation.

SG and SIG generated a very similar style of results. The results generated from SIG were slightly more defined, however, this improved result is at a high expense of computational time, so SIG was dismissed from further testing.

Outdoor Testing

The same samples that were used in the outdoor testing of the OpenCV saliency models were reused to evaluate the remaining PAIRML saliency models.

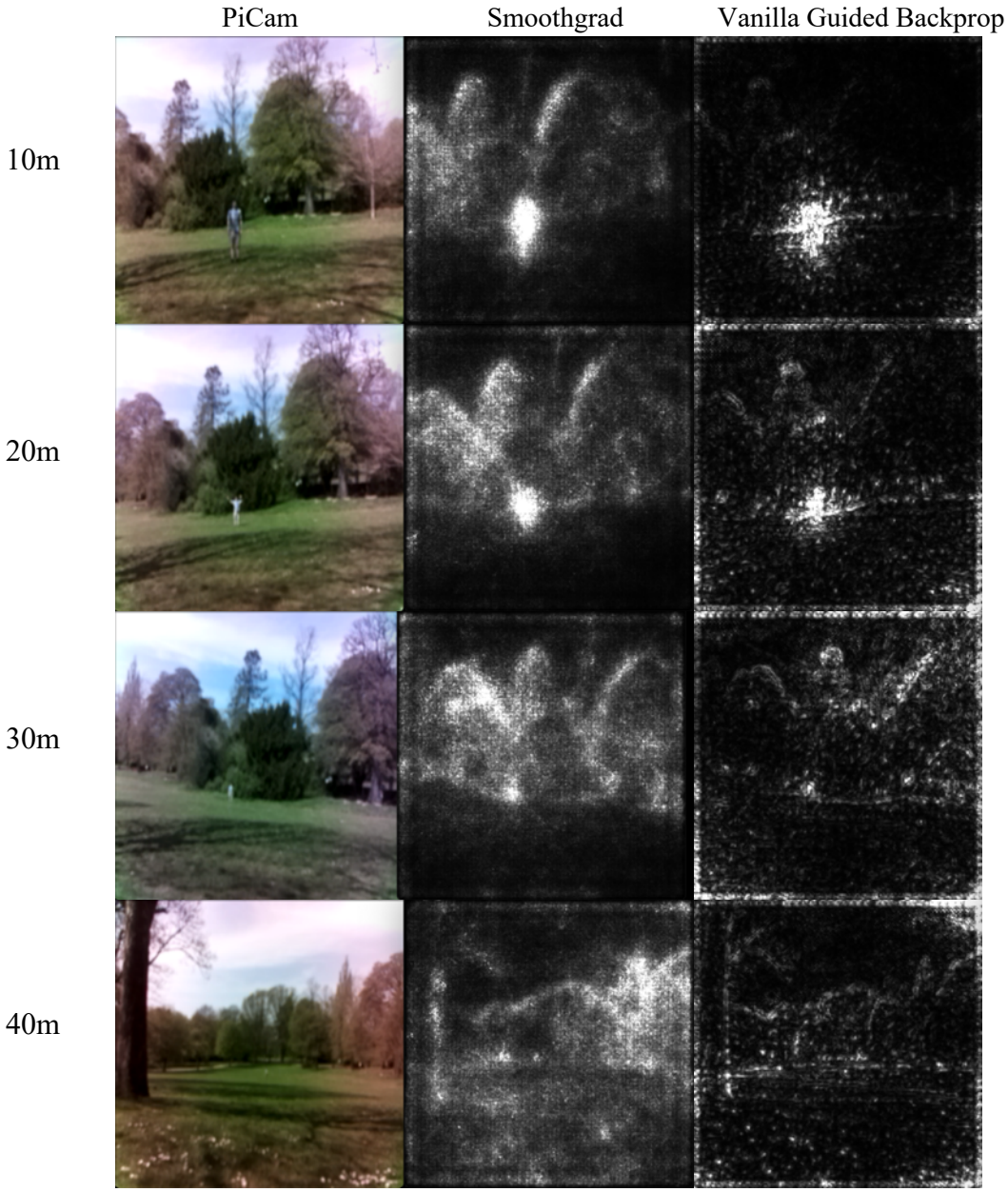


Figure 92 SG and VGB models applied to PiCam images of a person at different distances (m) from the field test on the sunny day

Both saliency models do a good job creating dense regions on the saliency map at the position of the body up to a 30m distance, beyond this the person is too undetailed to evaluate well. The SG model picks up a lot of other regions in the image as being quite salient as well which is not ideal. The VGB model does a better job of this and can identify the person in the image up to 30 m without picking up quite as many other regions as being points of interest.

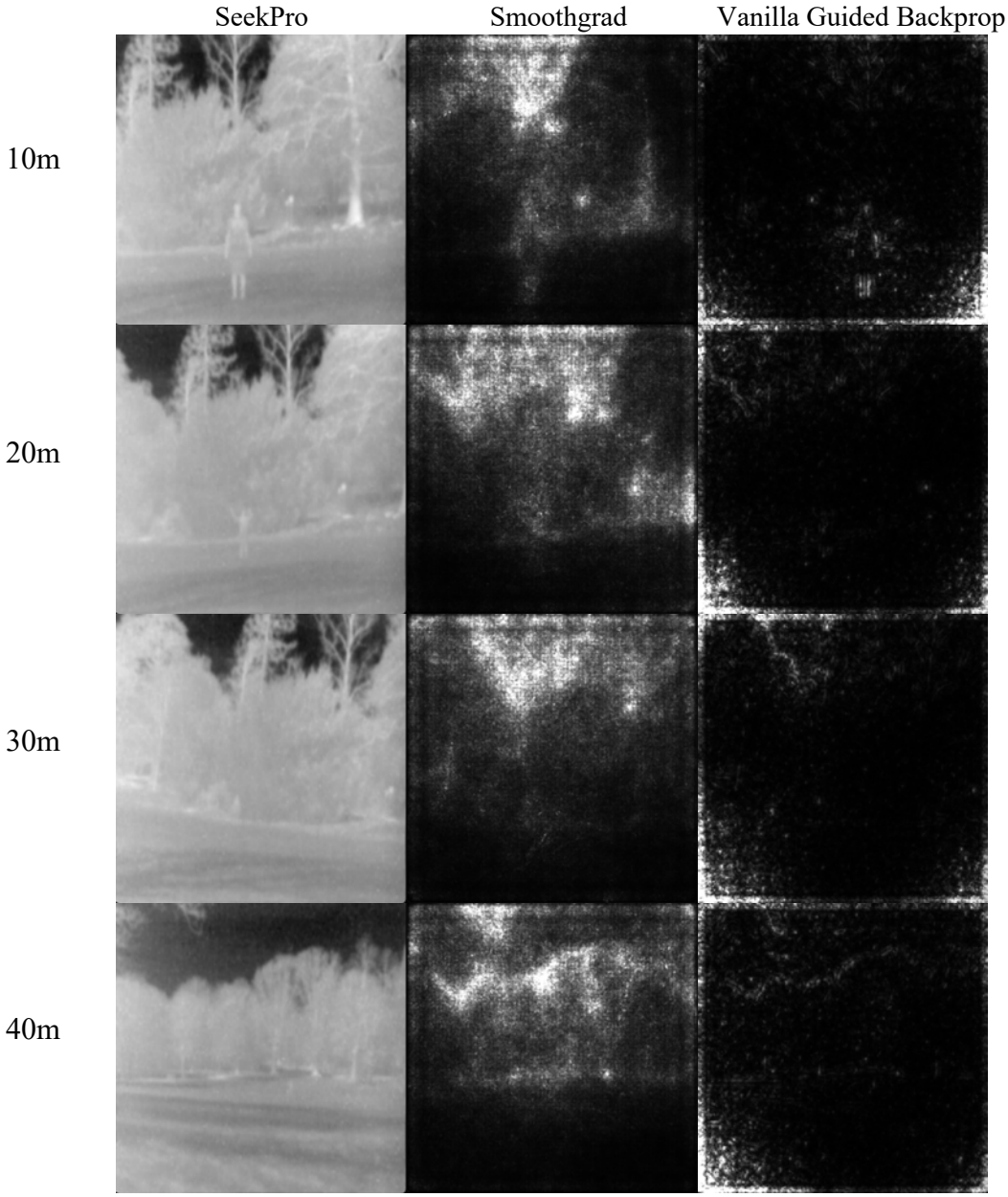


Figure 93 SG and VGB models applied to SeekPro images of a person at different distances (m) from the field test on the sunny day

The results in Figure 93 are very poor. Both models are only able to identify some parts of the person at 10m but beyond this, the person is not identified at all.

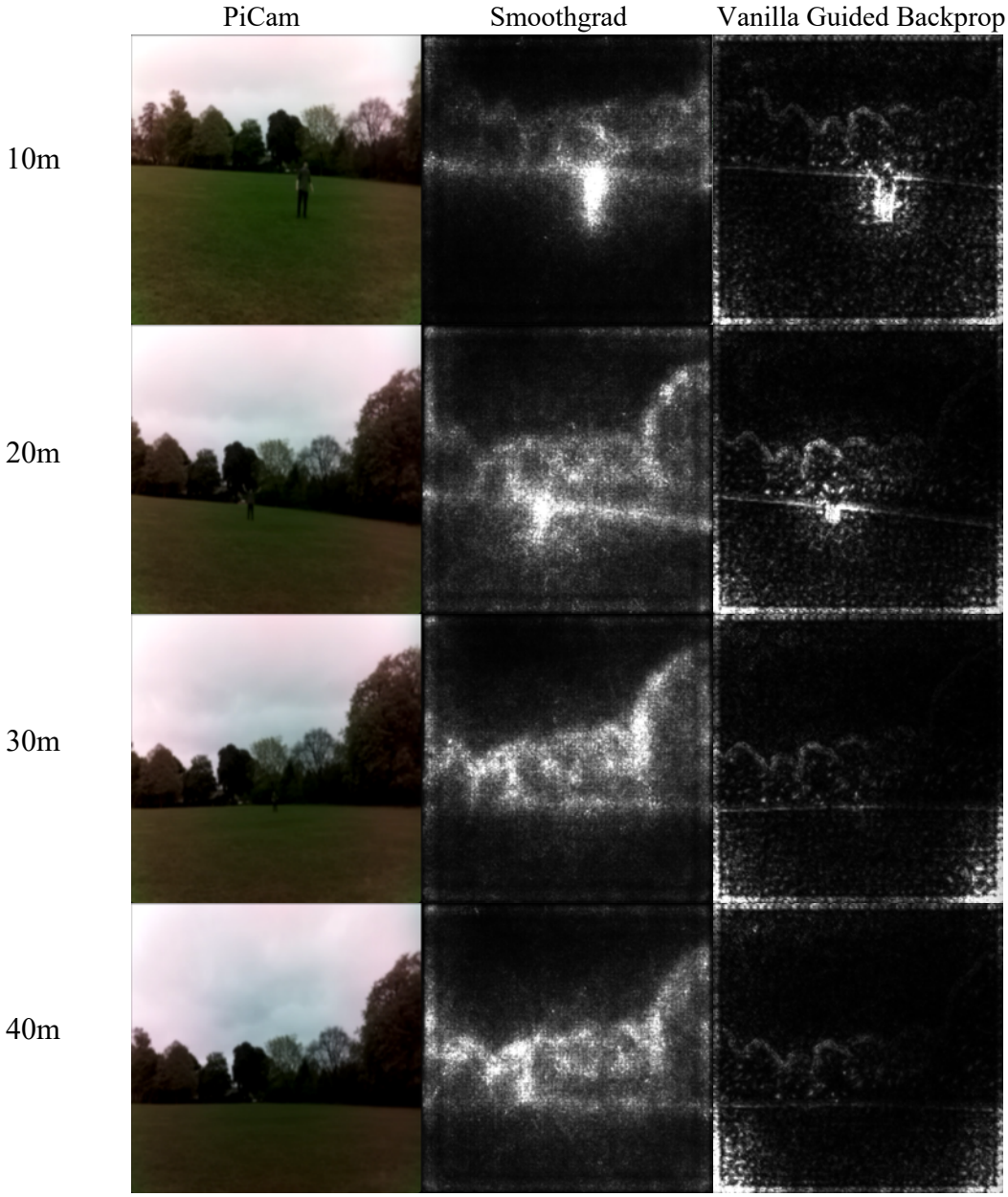


Figure 94 SG and VGB models applied to PiCam images of a person at different distances (m) from the field test on the cloudy day

In the test shown in Figure 94 both models are able to capture the person well at 10m. At 20m SG can identify the person having the region as the highest density spot, however VGB, again, does a better job of identifying the person without generating too much noise in the surrounding area as well. Both models struggle to identify the person at any distance beyond this.

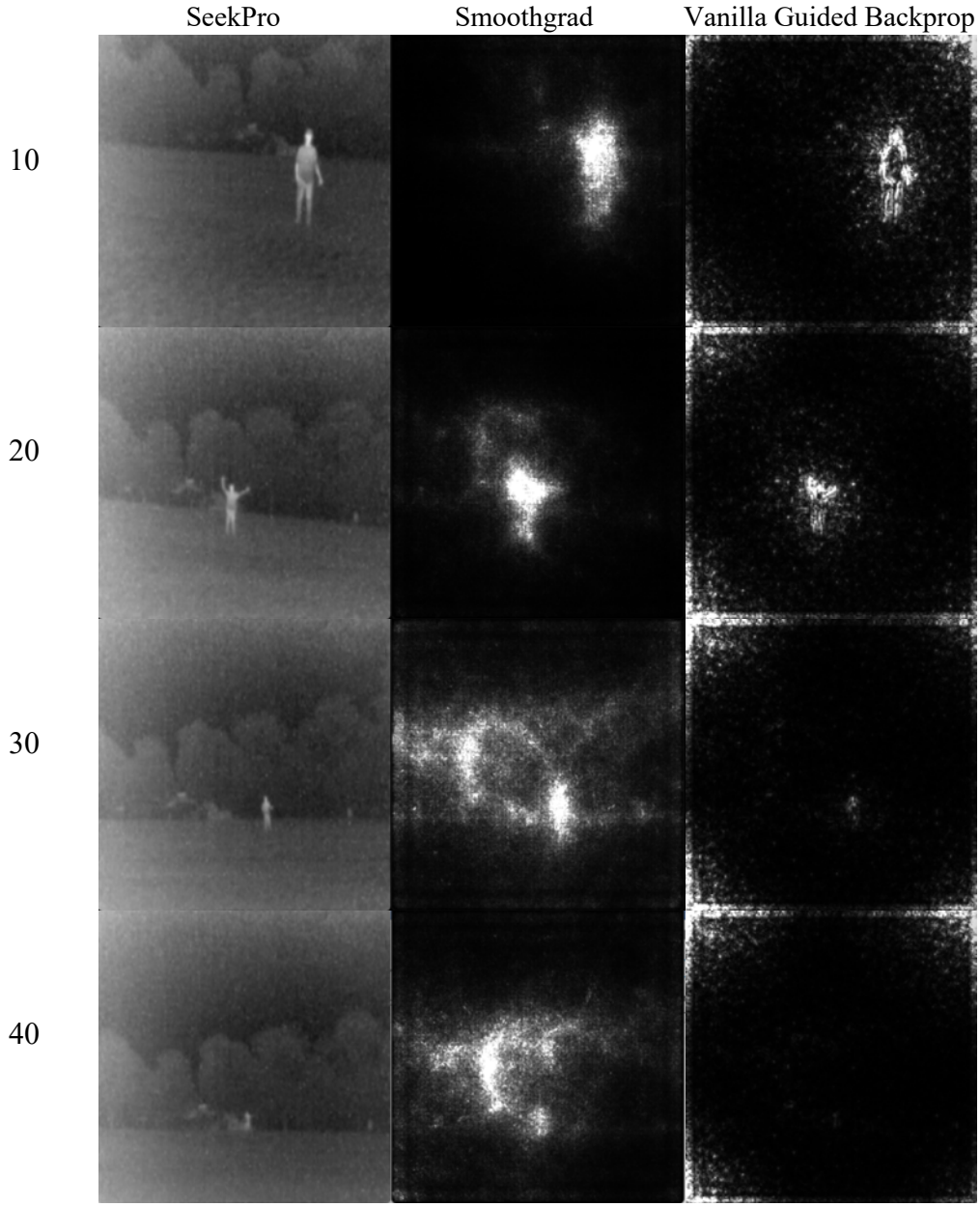


Figure 95 SG and VGB models applied to SeekPro images of a person at different distances (m) from the field test on the cloudy day

Finally, the test done in Figure 95 show that both models are capable of identifying the person very well at distances up to 20m. Beyond this, SG is still capable of identifying the person, however, other regions of the image begin to show up as salient regions as well. VGB barely identifies the person at 30m and not at all at 40m.

SARAA Testing

Using the Sample data that SARAA has provided, the algorithms were tested again on real life scenarios to evaluate their effectiveness.

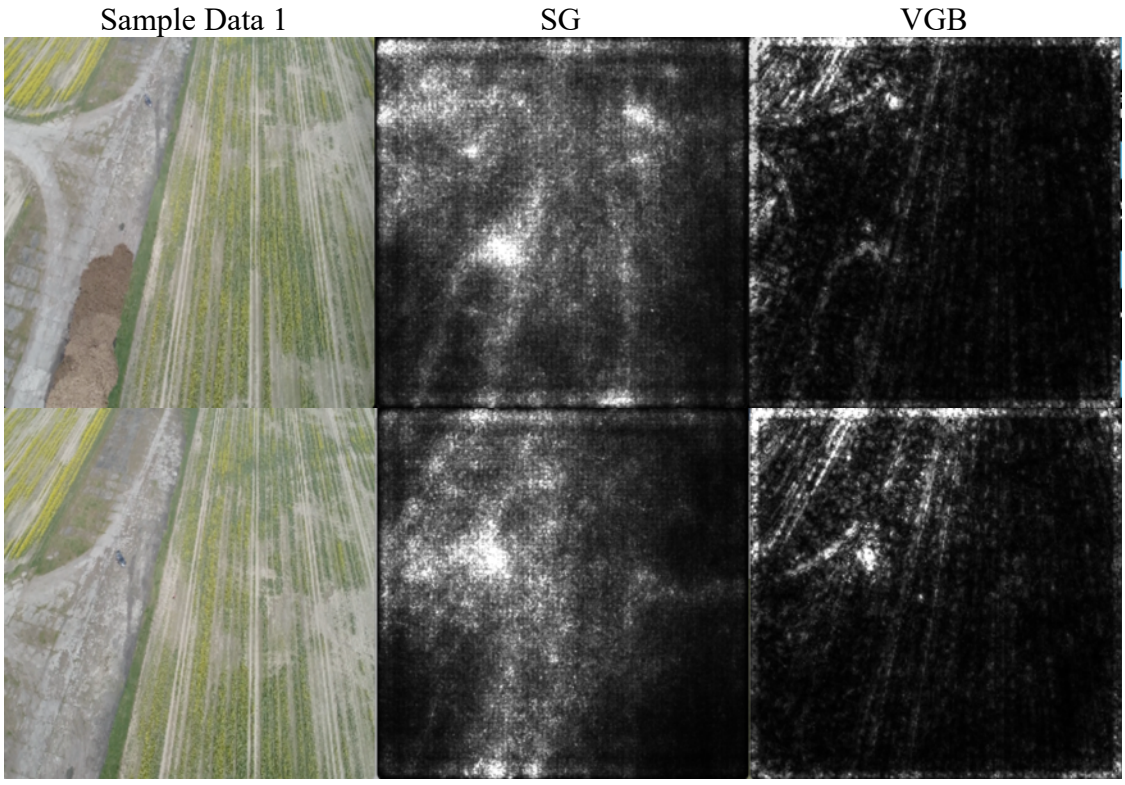


Figure 96 SARAA sample 1 - overhead image over a car and person in a field first is the full photo, second is a cropped sample applying SG and VGB

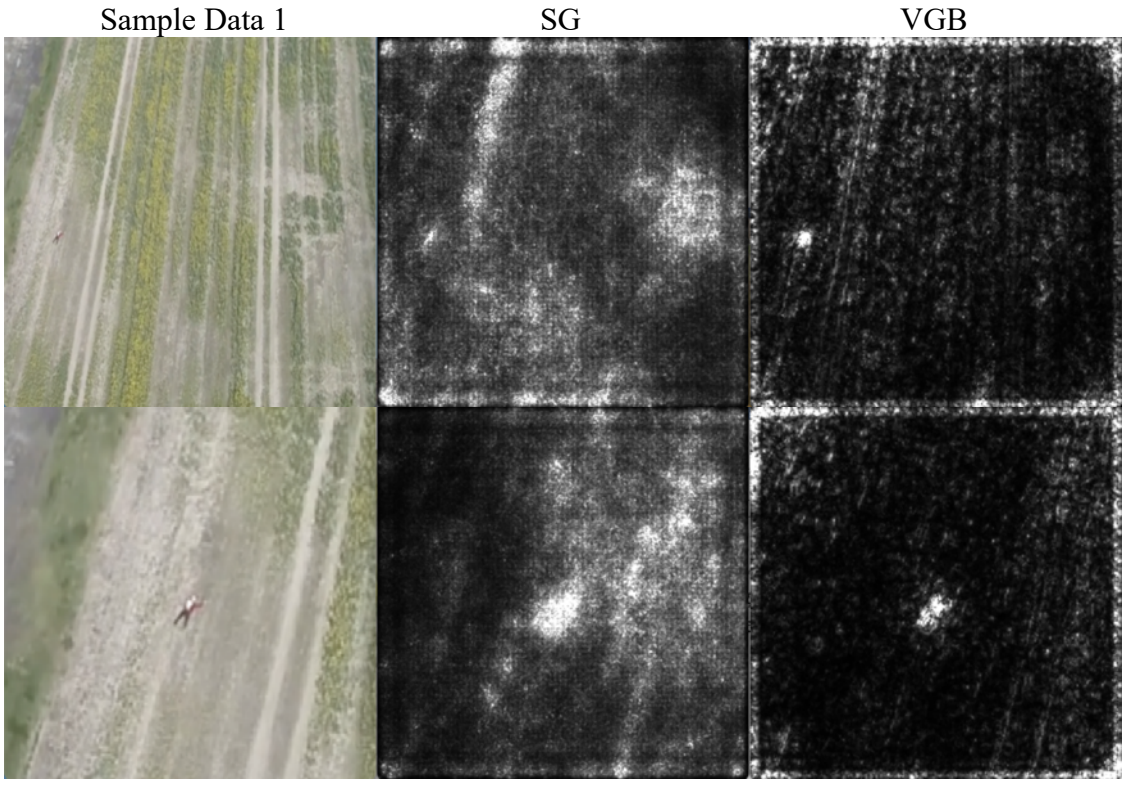


Figure 97 SARAA sample 1 – Further cropped image to capture only the person applying SG and VGB

It can be seen, in Figure 96 and Figure 97, that the SG algorithm does not do a very good job of identifying the person when there are lots of changing features in the image. However, the VGB does a surprisingly impressive job of sharply identifying a point of saliency for both the car and the person in these images.

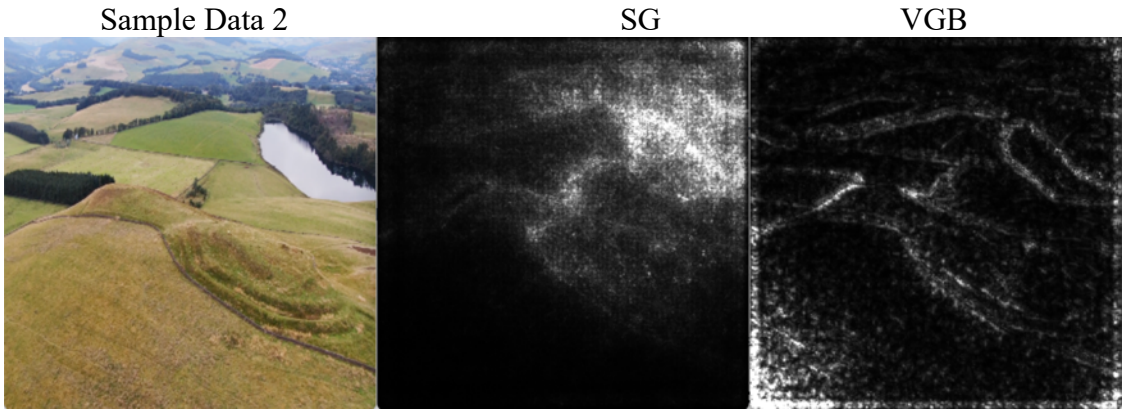


Figure 98 SARAA Sample 2 - People standing on-top of a hill applying SG and VGB

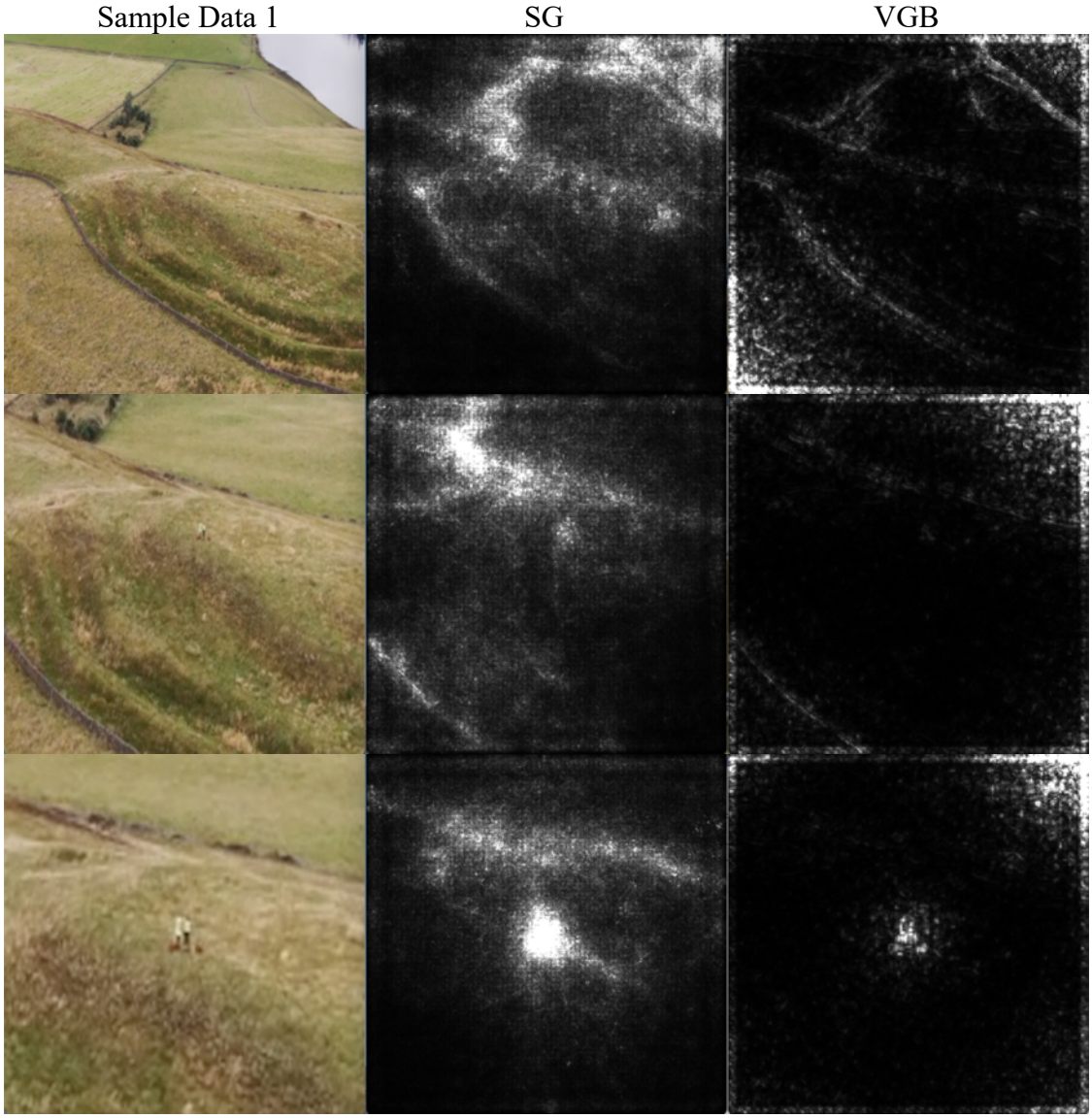


Figure 99 SARAA Sample 2 – Retesting the sample on cropped images applying SG and VGB

In Figure 98 and Figure 99, both algorithms found it difficult to locate and identify the people standing on the hill at full distance and only did a better job when the sample was zoomed into by quite a significant amount.

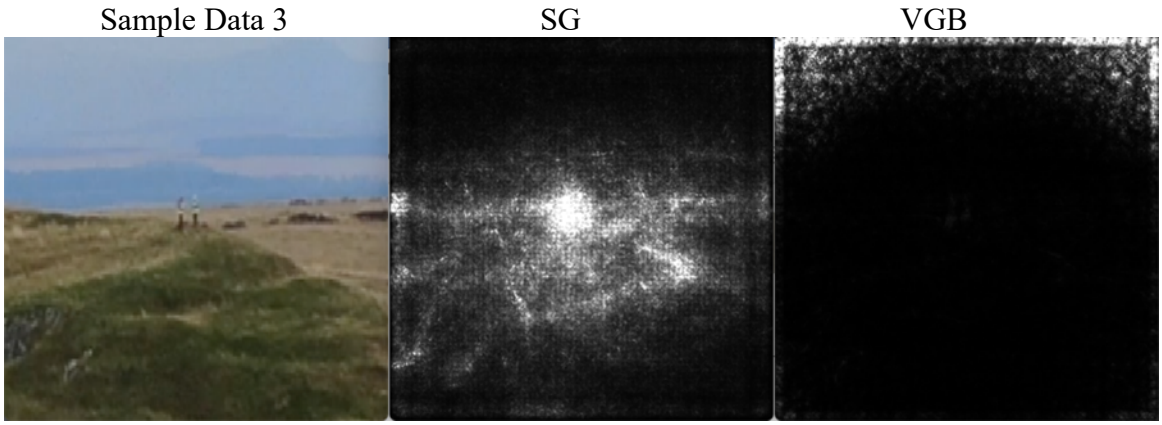


Figure 100 SARRA Sample 3 - People standing on top of a hill applying SG and VGB

The results of Figure 100 show the SG model very clearly identified to standing on-top of the hill, however the VGB model does not identify them at all.

6.4 Discussion – Region of Interest Modelling

By observing the performance of the visual images in the outdoor tests, it is evident that the lighting of regions has a big impact on what the saliency algorithms decide is interesting. On the hot sunny day, the algorithms did a better job of identifying the differences between the human and the environment compared to the darker cloudy day where the person appeared dark in the image and thus quite uninteresting. Another factor that needs to be taken into account is the clothing worn. On the hot day, the clothing worn was quite bright thus further improving the contrast, and on the cold day, the clothing worn was coloured grey and black thus causing the person to appear even darker. This will require more data before a conclusive explanation can be made regarding the effectiveness of the models based directly on lighting in an image.

Looking into particular models, starting with the Fine-Grained saliency model, this model generates lots of detail in the images it produces. It was found in the data providing good visual conditions, that this model generated quite a clear outline of the person as well as clearly identifying the outlines of shapes such as trees, this makes it difficult to extract only the people from the image without any additional processing. This is also the case in the sample images by SARAA where the images contain an extremely high amount of information but is still able to identify the person in the images, but this is alongside a lot of other detail. Maybe with some post processing the image produced could be cleared up to identify a much smaller number of salient regions.

The Spectral-Residual is quite different in that it produced extremely blurry saliency maps but in the right conditions, it successfully pinpointed regions of the image that did contain a person. It performed exceptionally well on the thermal images on the cold day, Figure 84, at all distances tested and did this at a quick speed as it is very efficient. This means that this model could be the right model to implement for the thermal camera's ROI system. The images from the sunny day, Figure 81, showed that the model could pick up the person in these conditions, however it did also identify other regions as interesting. Once again, if some post processing is done, it is possible this would also work quite well with the visual images produced on the sunny day.

Moving onto the SmoothGrad Saliency model. The benefits of this model are shown using visual images captured in the cold day, Figure 94. The model can identify the person as a highly interesting point up to 20m, the struggle at further distances is not unexpected as even visually it is very difficult to identify the person. This model works very well when the person is very visible in the image, however, as soon as the person becomes smaller within the image, more regions begin to be identified as interesting, this can be well observed in Figure 92. This is quite an unideal property of a saliency model as the nature of this project problem means the target will be quite small, especially as it is already quite a computationally expensive model so it would not be suitable to have to implement too much further post processing either.

Finally, Vanilla Guided Backpropagation. This model generated very promising results, performing quite well and consistently across the board, with a few exceptions. This model

was also capable of identifying the person up to 20m in the visual image on the cold day and up to 30m on the hot day, this was completed without the addition of a significant amount of noise. This model really showed its power in the first sample photo from SARAA, Figure 96 & Figure 97, where the car and the person was consistently identified in all of the images in this sample. However, the result from Figure 99 and especially Figure 100 indicate that the results are not always successful and can perform surprisingly poorly in some conditions. This model is believed to have a lot of potential for a ROI model but requires further investigation to understand where its limitations lie.

As a whole, the ROI model was intended to fully operate as a coarse searching object detector, however, only an analysis on a range of saliency models was reached at this point and further processing work was not implemented. The infrastructure of the contouring functionality is available as a placeholder, but it only operates well when there is a clear single region of interest that is visible within the image, and this is not realistically going to be the case in most scenarios. Further work will need to be done to the individual saliency models to conclusively identify their strengths and weaknesses in different environments, beyond this a range of post processing techniques need to be applied to the models individually to filter and classify the salient regions of the image to identify the final region of interest.

7 System Integration into SAR Drone

This section will look into how well this system will fit within a drone. This will mainly look at the physical restriction taking into account its physical size taking into account the batteries necessary to operate.

The restrictions of the system are bound by the system specifications. The ones governing the physical size of the system is SR4 found in Table 1. This states that 'The camera system should retain a sub-system payload limit of 1 kg'.

System Battery
Below are the components of the system with their weights and power consumptions identified.

Table 3 Weights and Power consumptions of the components within the system

Component	Weight (grams)	Max Voltage Input (V)	Max Current Input (A)	Max Power (W)
Cameras – PiCam, WebCam & SeekPro	75	-	-	-
Microcomputer - Raspberry Pi	50	5	3	15
Gimbal – STorM32-BGC v1.3	195	16.8	1.5 (per motor)	81

Before adding a battery to the system, it only weighs 330 grams, which is a very reasonable size for a camera system like this. The goal of the operating time for the system is set by SR8 which states 'The drone should be fully operational for 30 minutes. Due to the large variance in voltage input between the Raspberry Pi and the Gimbal system, there should be 2 batteries used to power each independently.

The Raspberry Pi would require a battery that could provide 7.5Whr or (1500mAh)
This could be achieved in a compact way using a 2S LiPo Battery connected to a 5V limited BEC to limit the voltage supply whilst generating enough current for the Raspberry Pi.

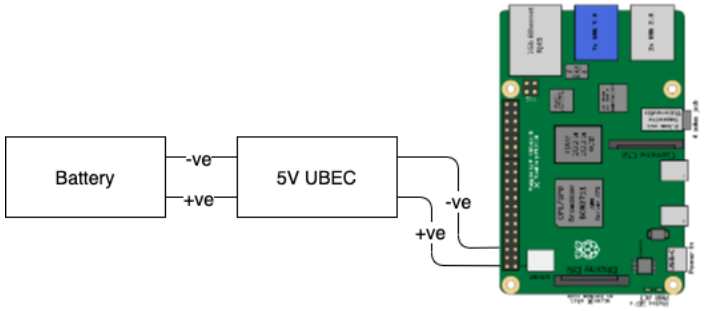


Figure 101 Infrastructure to power the Raspberry Pi from a drone

A Zippy-Compact-2100mAh [40] battery with a 2S3P cell configuration should provide more than enough sustained voltage and current to run the system with a battery weight of only 100g making this an ideal battery for this system allowing the raspberry pi to run for at least 45 minutes in a single run.

The gimbal will need a bigger battery that can provide 40.5Whr or (2200mAh) supplied using a 3S to 4S cell configuration LiPo Battery to achieve a constant 11.1-16.8V supply to the system. The system only needs to deliver up to 1.5A per motor and since there are 3 of them the battery only needs to provide up to 4.5A.

A custom battery can be configured using a set of 'Samsung 25R 18650 Battery' [41] placing 4 in series to create a 10,000 mAh battery that can provide up to 14.4V. At a maximum current draw of 4A, this should be able to operate for up to 2.5 hours on a single charge. This will have a total weight of 180g for the cells and potentially an additional 50 for the remaining wire and enclosure placing it at a total weight of 230g.

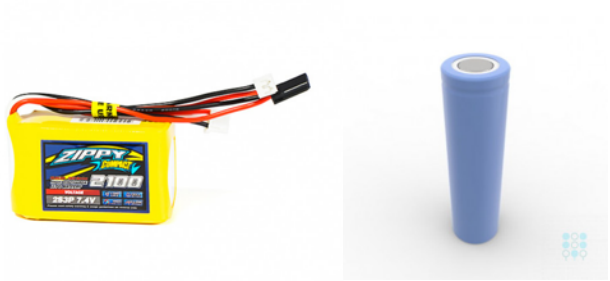


Figure 102 Zippy-Compact-2100mAh and Samsung 25R 18650 Battery respectively

System Size

As a whole the system comes out to weigh 660 grams leaving 340 grams for further development around the system. This would include a mounting system into the drone as well as a camera case to ensure the camera system is not too exposed beyond that is necessarywhen it is within the drone.

The height of the camera system goes up to 100mm at its tallest, depending on the gimbal orientation, with constant width and depth of 80mm.

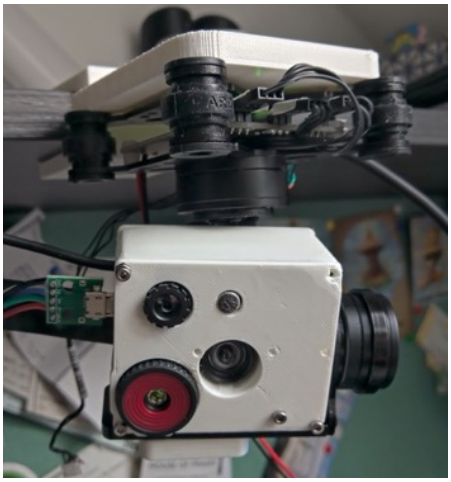


Figure 103 Camera system on the gimbal

System implementation

The camera system as a whole is still not prepared to be interfaced into the drone, this is because the gimbal still needs to be correctly implemented and the Region of Interest model still needs some work before the final system can function autonomously.

8 Future work

There are a few things that need completion before the camera system can be implemented as intended, furthermore other parts could be improved in further iterations of similar work.

8.1.1 More Field Testing

More field testing is arguably the most important future work to be implemented into the system. Images should be captured in many more environments and from a drone to capture the most similar to real life scenario.

8.1.2 Saliency Post Processing

More post processing needs to be applied to the saliency models that were applied. This should be done independently for all the different saliency models as they each have strengths and weaknesses so need to be treated independently. This can be done using processes such as using the Hough transform to remove streaks of long lines in an image for the fine-grained models or template matching to identify particular aspect ratios to match the size of humans in the Vanilla Guided Backpropagation model.

8.1.3 Using other object detection models

Other object detection models should be implemented as well, models like the YOLOv3 could have a powerful use in a project like this, however, this would require the development of a dataset trained from a lot of samples generated from the visual and thermal images. This will take a while to train, however, could be one of the better solutions to a problem like this if the microcomputer used can handle it. This could work in conjunction with the saliency models, where they would identify interesting spots very quickly, and the object detection algorithm would run several cropped sections of the original image to make a decision.

8.1.4 Gimbal Implementation

Although the gimbal was introduced, it was never fully implemented into the system, this came through complexity due to bad documentation. However, this is still an implementable feature that has been completed by others within the drone community and will be essential to get working before the system can be placed onto a drone to fully operate

8.1.5 Thermal camera active cooling system

As it was seen in the investigation of the SeekPro camera, it performs at its best when it is cold, it would be good to implement an active cooling system to keep the temperature quite low for the best performance.

8.1.6 Improvements in thermal calibration

The board used for the thermal calibration could be improved by using active heating and cooling pads on the board, such as peltier modules. This would ensure a consistent contrast is produced by the board allowing for improved success in the calibration model.

8.1.7 Perspective transformation improvements

The perspective transformation method could be improved by implementing a large board with uniformly spaced heat pads. By doing this the points will be a lot clearer for selection. Beyond this, pattern matching methods could potentially be used as well to identify the positions to automate the perspective transformation a more seamless process.

9 Conclusion

Image Capture
The outdoor tests performed with the camera system generated some very clear results showing the benefits and drawbacks of each camera within the system. It showed how the usage of visual and thermal imagery could complement each other where the visual camera could be the primary camera on hotter brighter days and the thermal camera could be used on the colder less bright days. It is recognised that the weather conditions are not one or the other making it difficult to know its performance under a different set of conditions.

Sensor Fusion
The implementation of image fusion into the system was successful at providing insight into the benefits of fusion in potential SAR environments. These are due to the same findings in the outdoor tests, but by putting the data together, it allows the information to be captured at the same time by the operator. As expected, the fusion of the images operates at its best in colder conditions. It shows off its best performance at distances beyond 20m from the target, this is when the fused images have lots of environmental detail from the visual image but also the necessary thermal hotspot to clearly identify the person. Unfortunately, the system does not perform well at all in hot weather because the thermal camera image is blown out due to the ground heating up and the fusion just takes away from the visual image in this scenario.

Region of Interest modelling
The region of interest modelling was all developed based on saliency models. The different saliency models that were implemented had a range of benefits and drawbacks depending on the situation provided to it. The Fine-Grained, Spectral-Residual and the Vanilla Guided Backpropagation were the overall favourites of the system. The OpenCV models operated quickly in the system and have the potential to be used individually for different scenarios. They will need some post processing before complete implementation can be done, and it is likely that the usage of a further object detector will need to be used to filter and verify the salient results generated. In contrast the PAIRML models, were very slow but the Vanilla Guided Backpropagation model in particular provided some surprisingly precise results in some scenarios. By applying post processing and passing the most salient regions into object detection algorithms, these systems may provide the solution to solve the intended goal of the ROI model.

System integration
From a physical design perspective, the system is relatively light, at 660g, and compact meaning it would fit quite well within the SAR drone developed without much struggle. It can operate with a minimum runtime of 45 minutes for the Raspberry Pi and 2.5 hours on the gimbal from a full charge making the system ideal in this regard. However, some work still needs to be done to allow the gimbal to operate with the system as well as some further development within the ROI model.

10 References

[1] D. Joshi, “Drone technology uses and applications for commercial, industrial and military drones in 2020 and the future,” Business Insider, 19 December 2019. [Online]. Available: https://www.businessinsider.com/drone-technology-uses-applications?r=US&IR=T.

[2] S. Whittaker, “Drones in Disaster Response,” Drone Below, 28 December 2017. [Online]. Available: https://dronebelow.com/2017/12/28/drones-disaster-response/.

[3] ERL, “ERL Emergency Local Tournament 2020,” ERL, 2020. [Online]. Available: https://sites.google.com/view/erlemergency2020.

[4] iMechE, “UAS CHALLENGE,” iMechE, 2020. [Online]. Available: https://www.imeche.org/events/challenges/uas-challenge/about-uas-challenge.

[5] UAV Challenge, “UAV Challenge,” UAV Challenge, 2020. [Online]. Available: https://uavchallenge.org/about/.

[6] P. e. al., “Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery,” Linkoping University, Linkoping, Sweden.

[7] L. et.al, “Cooperative Fire Detection using Unmanned Aerial Vehicles*,” University of Seville, Seville, 2005.

[8] N. e. al., “Fused Visible and Infrared Video for use in Wilderness Search and Rescue,” Brigham Young University, Utah, US, 2009.

[9] J. e. al., “You Only Look Once: Unified, Real-Time Object Detection,” University of Washington, Washington, USA, 2016.

[10] N. D. Elsa Sebastian, “A Survey on Various Saliency Detection Methods,” Viswajyothi College of Engineering and Technology, Kerala, India, 2017.

[11] A. Rosebrock, “OpenCV Saliency Detection,” pyimagesearch, 16 July 2018. [Online]. Available: https://www.pyimagesearch.com/2018/07/16/opencv-saliency-detection/.

[12] A. SHARMA, “WHAT ARE SALIENCY MAPS IN DEEP LEARNING?,” Analytics India Mag, 11 07 2018. [Online]. Available: https://analyticsindiamag.com/what-are-saliency-maps-in-deep-learning/.

[13] A. S. Sebastian Montabone, “Human detection using a mobile platform and novel features derived from a visual saliency mechanism,” Pontificia Universidad Catolica de Chile, Santiago, Chile, 2009.

[14] X. H. a. L. Zhang, “Saliency Detection: A Spectral Residual Approach,” Shanghai Jiao Tong University, Shanghai, 2007.

[15] D. e. al., “SmoothGrad: removing noise by adding noise”.

[16] K. e. al., “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” Visual Geometry Group, University of Oxford, Oxford, 2014.

[17] J. e. al., “STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET,” University of Freiburg, Freiburg, 2015.

[18] M. e. al., “Axiomatic Attribution for Deep Networks,” Cornell University , Cornell, 2017.

[19] Yida, “Rock Pi N10 vs Raspberry Pi 4 vs Jetson Nano – AI and Deep Learning Capabilities,” SeeedStudio, December 2019. [Online]. Available: https://www.seeedstudio.com/blog/2019/12/05/rk3399pro-vs-raspberry-pi-4-vs-jetson-nano-ai-and-deep-learning-capabilities/.

[20] UCTRONICS, “Arducam 8 MP Sony IMX219 camera module with CS lens 2718 for Raspberry Pi,” UCTRONICS, [Online]. Available: https://www.uctronics.com/index.php/arducam-8-mp-sony-imx219-camera-module-with-cs-lens-2717-for-raspberry-pi.html.

[21] ArduCam, “ArduCam Low Distortion M12 Lens Kit,” [Online]. Available: https://www.arducam.com/doc/m12_lens_kit_2.pdf.

[22] Logitech, “C270 Product Page,” Logitech, [Online]. Available: https://www.logitech.com/en-us/product/hd-webcam-c270#specification-tabular.

[23] Seek thermal, “Seek Compact Pro Datasheet,” Seek thermal, Santa Barbara.

[24] V. Couty, “Code to access Seek Compact Pro using python,” 24 September 2018. [Online]. Available: https://github.com/LaboratoireMecaniqueLille/Seek-thermal-Python/blob/master/seekpro.py.

[25] SciPy Documentation, “What is NumPy?,” SciPy Documentation, [Online]. Available: https://docs.scipy.org/doc/numpy/user/whatisnumpy.html.

[26] OpenCV, “About OpenCV,” OpenCV, 2020. [Online]. Available: https://opencv.org/about/.

[27] Pillow, “Pillow,” Pillow, 2020. [Online]. Available: https://pillow.readthedocs.io/en/stable/.

[28] “Saliency Python Library,” 26 Septempber 2019. [Online]. Available: https://pypi.org/project/saliency/.

[29] D. Jones, “PiCamera Documentation,” 2016. [Online]. Available: https://picamera.readthedocs.io/en/release-1.13/.

[30] W. L. Costa, “Pyusb 1.0.2,” 17 October 2017. [Online]. Available: https://pypi.org/project/pyusb/.

[31] OpenCV-Python Tutorials, “Camera Calibration,” OpenCV-Python Tutorials, 2013. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html.

[32] J. W. e. al., “A new calibration model of camera lens distortion,” Shanghai Jiao Tong University, Shanghai, 2007.

[33] OpenCV, “Basic concepts of the homography explained with code,” OpenCV, [Online]. Available: https://docs.opencv.org/master/d9/dab/tutorial_homography.html.

[34] OpenCV, “ColorMaps in OpenCV,” OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d3/d50/group__imgproc__colormap.html.

[35] OpenCV, “Saliency Class Reference,” OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d9/dcd/classcv_1_1saliency_1_1Saliency.html.

[36] OpenCV, “Contours : Getting Started,” OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

[37] “PAIRML Saliency GitHub,” [Online]. Available: https://github.com/PAIR-code/saliency.

[38] yrevar, “List of the models withing the Inception V3 dataset,” GitHub, [Online]. Available: https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a.

[39] vsubhashini, “Computing saliency masks with the PAIRML saliency library,” GitHub, [Online]. Available: https://github.com/pair-code/saliency/blob/master/Examples.ipynb.

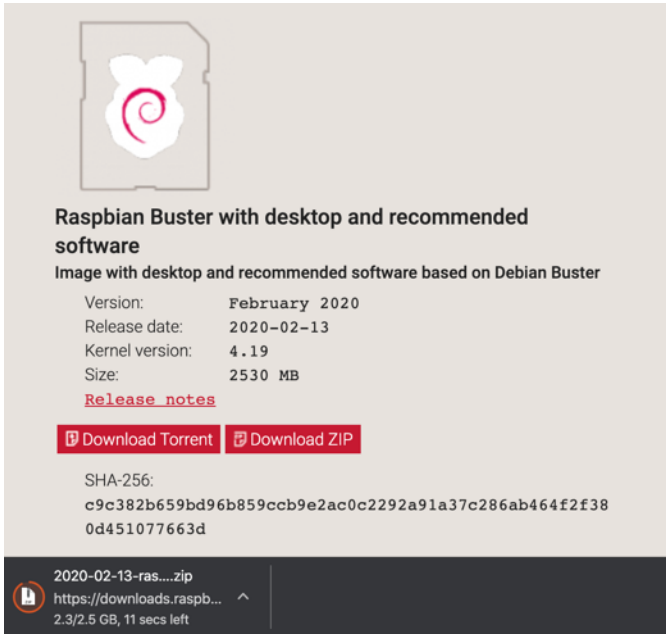
[40] Hobby King, “ZIPPY Compact 2100mAh 2S3P 7.4V Receiver Pack,” Hobby King, [Online]. Available: https://hobbyking.com/en_us/zippy-compact-2100mah-2s3p-7-4v-receiver-pack.html?queryID=&objectID=82493&indexName=hbk_live_magento_en_us_products&___store=en_us.

[41] voltaplex, “Samsung 25R 18650 Battery, 2500mAh, 20A, 3.6V, Grade A Lithium-ion (INR18650-25R),” voltaplex, [Online]. Available: https://voltaplex.com/samsung-25r-18650-battery-inr18650-25r.

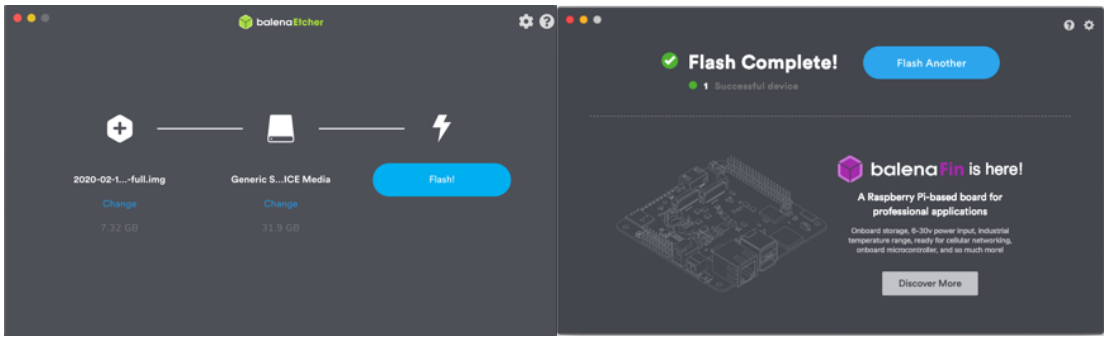
11 Appendices 1 – System Setup

11.1 Raspberry Pi OS Setup

Raspbian buster should be downloaded to an external computer from the official Rapsberry Pi webpage.



The downloaded file can then be flashed onto a 32GB SD card using balenaEtcher by selecting the zipped or unzipped (it works either way) as the image and the the SD card as the target to be installed into and then selecting ‘flash’.



Once this is complete the SD card needs to be inserted into the Rapsberry Pi and be turned on to ensure the flashing has been done correctly as well as setting up initial system settings.

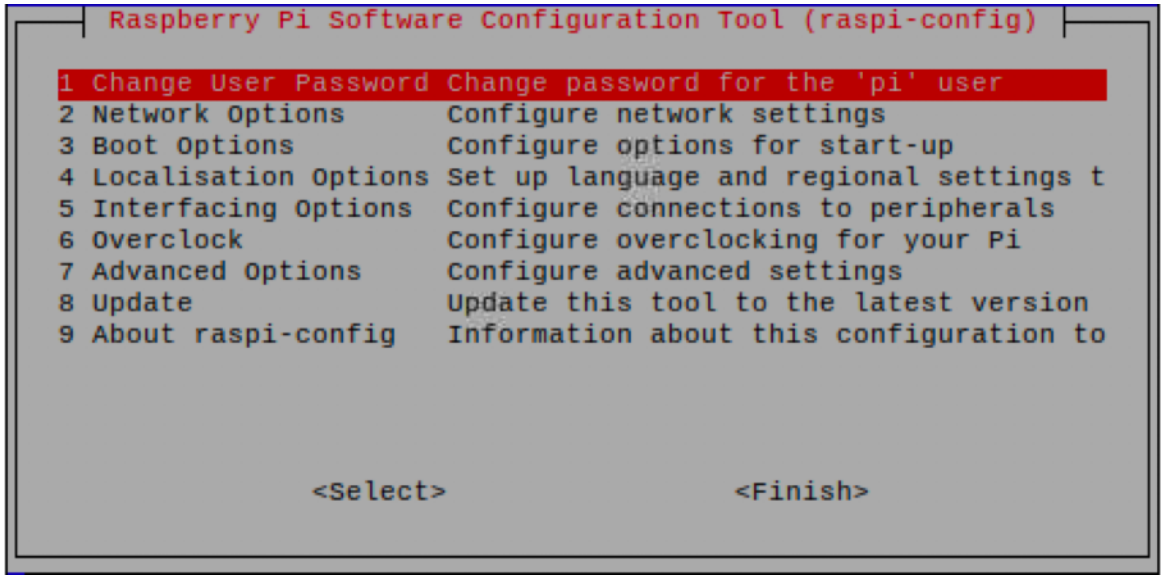
When the raspberry pi successfully boots up, an internet connection should be established, using wifi or ethernet. Ethernet was the connection of choice whilst working at the university due to the difficulty of connecting to the Eduroam servers via WIFI from a Rapsberry Pi.

In the terminal, the Raspberry Pi's firmware and libraries should be brought up to the latest version:

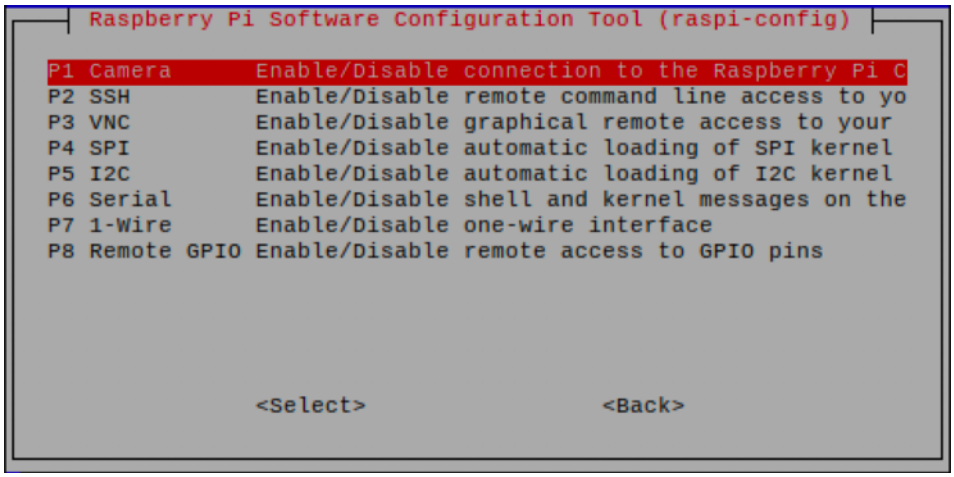
```
pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get upgrade
```

Once this is complete some setup in the configuration tool by typing:

```
pi@raspberrypi:~ $ sudo raspi-config
```



From here go into the 'Interfacing Options' menu and a list of interfaces should show up.



In this menu, enter the relevant interfaces and enable the Camera and SPI interfaces. The SSH and VNC interfaces can also be enables if the setup will be used from an external computer.

11.2 Visual camera setup

Both visuals cameras are read by the '*RGBCam.py'* script. The ArduCam module, described as the PiCam, should connect up to the Raspberry Pi using the CSI Camera interface, as showing in Figure 9, and the Logitech WebCam, this will be described as the WebCam, should be connected up to the Raspberry Pi via its USB connector.

The code begins by importing the necessary libraries needed to capture and apply some early minor processing to the images.

Code Excerpt 15 RGBCam - Library Imports and Initialisating Variables

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import cv2
import numpy as np

# initialising PiCam variables for resolution and framerate
RESOLUTION = 320, 240
FRAMERATE = 25
```

The resolution and framerate variables can be modfied to affect the visual performance of the camreas. In general the higher the resolution, the slower the capture, and the higher the framerate, the lower the image quality. Code Excerpt 16, shows how the frames from the PiCam are captured.

Code Excerpt 16 PiCam Class within the RGBCam script

```
class PiCam():
    def __init__(self):
        # initialize the camera and grab a reference to the raw camera capture
        self.camera = PiCamera()
        self.camera.resolution = RESOLUTION
        self.camera.framerate = FRAMERATE
        #Convert Data into readable array
        self.rawCapture = PiRGBArray(self.camera, RESOLUTION)
        #Camera Warmup Time
        time.sleep(0.1)

    def frameCapture(self):
        '''Capturing the RGB Image'''
        #Clearing the image stream for the next frame
        self.rawCapture.truncate(0)
        #Capturing a frame from the video stream to display
        for frame in self.camera.capture_continuous(self.rawCapture, format="bgr", use_video_port=True):
            img1 = frame.array
            return img1

    def stop(self):
        self.camera.close()
```

Images can be called from this class by calling *'PiCam().frameCapture()'* from anywhere in the script. This is later set as the WideImg variable in the final script which, will be covered later, in this way:

```
WideImg = PiCam().frameCapture()
```

Below is a sample of an indoor image captured by the Wide Camera

Figure 104 Sample image produced by the PiCam

The Raspberry Pi can interface with the WebCam using OpenCV 'VideoCapture' function independantly from the PiCam.

Code Excerpt 17 WebCam Class

```
class WebCam():
    def __init__(self):
        self.camera2 = cv2.VideoCapture(1)
        # allow the camera to warmup
        time.sleep(0.1)

    def frameCapture(self):
        "'Capturing the RGB Image (CAMERA 2)'"
        ret, img = self.camera2.read()
        (h,w) = img.shape[:2]
        imgNarrow = img[int(h/4):int(3*h/4), int(w/4):int(3*w/4)]
        imgNarrow = cv2.resize(imgNarrow, RESOLUTION)
        return img, imgNarrow

    def releaseCapture(self):
        # releases the camera
        self.camera2.release()
```

Code Excerpt 17 shows the initial code developed for the WebCam class to capture images using the webcam. Some data processing is done in the class to generate a narrower image better represent the narrow camera. There are 2 outputs being returned from the frameCapture function in this class as the image is intended to act as a placeholder for a narrow camera. *'WebCam().frameCapture()[0]*' should be called to capture the full size image and to *'WebCam().frameCapture()[1]'* to capture the narrow/cropped image.

Figure 105 Images produced by the WebCam left is WebCam().frameCapture()[0], and right is WebCam().frameCapture()[1]

11.3 Thermal Camera Setup

The Raspberry Pi interfaces with the thermal camera using the *'IRCam.py'* script. As previously mentioned, most of the code was developed to capture the image data was developed by Victro Couty however some modifications were made to improve the performance and interfacing ability with the rest of the system. The thermal camera will be referred to as the SeekPro for the remainder of this project.

The thermal camera connects to the Raspberry Pi using a USB interface, however since the thermal camera uses a Micro-USB B connector, a Female Micro-USB to Male USB-A adapter needs to be used to successfully connect this to the Raspberry Pi.

The script captures data in 16 bit range capturing thermal reading for every pixel with a value between 0 and 65,536. This raw data capture is grabbed using the *'get_image'* function in the SeekPro Class. The raw data 'IRdata' is then passed into the 'rescale' function allowing the data to convert the 16 bit range of values into an 8 bit scale, this allows it to be processed and displayed in a greyscale image. This process can be seen in Code Excerpt 18

Code Excerpt 18 Rescale function to convert thermal data into the greyscale range

```
def rescale(self, IRdata):
  """
  To adapt the range of values to the actual min and max and cast it into
  an 8 bits image
  """
  if IRdata is None:
     return np.array([0])
  mini = IRdata.min()
  maxi = IRdata.max()

  # Scaling the image to an 8 bit 'greyscale' range
  imgScale = ((np.clip(img-mini,0,maxi-mini)/(maxi-mini)*255.)).astype(np.uint8)
  return imgScale
```

The code can thus be run by and a thermal image can be captured in this way:

```
IRimg = IRCam().rescale(IRCam.get_image())
```

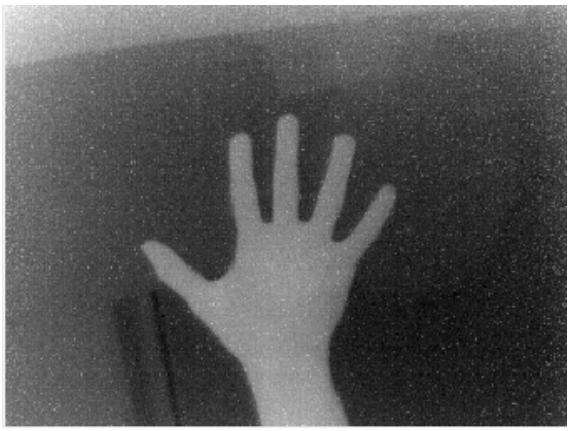


Figure 106 Thermal Image produced by the SeekPro

11.4 Python Library Setup

To install these libraries into python, first the 'package installed for python' (pip) needs to be installing into the Raspberry Pi. Pip allows users to quickly install packages that might otherwise take lots of time to do if the installation procedure was done from the source.

Setting up pip is quite easy and just requires an internet connection to be established to the Raspberry Pi. Pip for Python 3 can then easily be installed by typing in:

```
pi@raspberrypi:~ $ sudo apt-get python3-pip
```

Once the process is completed, the version should be checked to ensure it is installed correctly and for the correct version of Python.

```
pi@raspberrypi:~ $ pip -V
pip 20.0.2 from /usr/local/lib/python3.7/dist-packages/pip (python 3.7)
```

Now, the remaining libraries can simple be installed using the 'sudo pip install' function in this way:

```
pi@raspberrypi:~ $ sudo pip install numpy
```

Libraries: NumPy, Pillow, Saliency, PiCamera, PyUSB

The same process can be done for the other libraries by just replacing 'numpy' with the other library titles except for OpenCV.

For OpenCV core library as well as the contribution library should be installed onto the system. This can be done in this way:

```
pi@raspberrypi:~ $ sudo pip install opencv-python
pi@raspberrypi:~ $ sudo pip install opencv-contrib-python
```

12 Appendices 3 – Thermal camera problems

12.1 Clipping Problem

An observation that was made and persisted through testing was a clipping issue that occurred when the thermal images were exposed to cold temperatures. This can be seen in Figure 107below where in the left image there is no clipping, as a bottle is being introduced into the image, but when the coldest section of the bottle appears, a sort of error occurs and only the cold sections in the image appear and are whitened out.

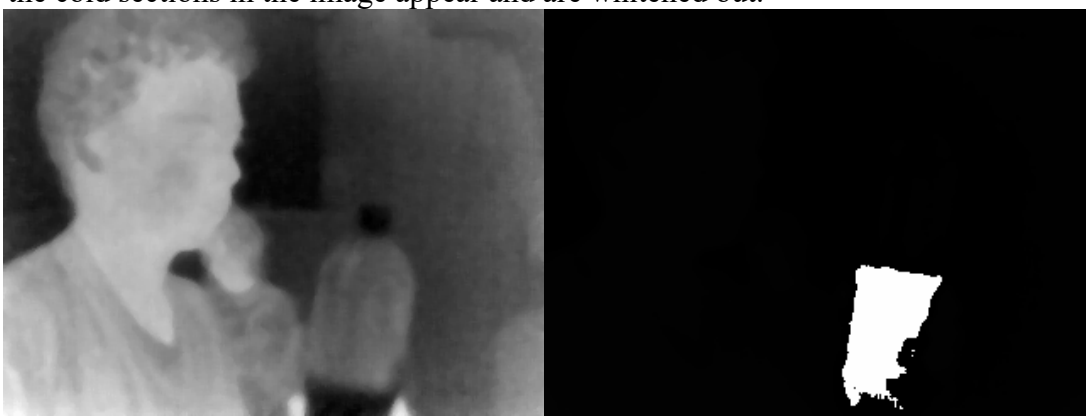


Figure 107 Bad thermal clipping when a cold bottle of water is introduced

This occurred more sporadically when the thermal camera was on for an extended period of time, the assumption was made that the thermal camera warmed up causing temperature differentials to be greater with colder objects and thus the issue became worse. This made this an important issue to solve as the thermal camera would be expected to operate stably for long periods of time. After analysing the data from the images, it was noticed that these cold sections were at extremely high values compared with the rest of the image which is bizarre as they are expected to have lower values since they were colder.

Variables – thermcameradata2

thermcameradata | thermcameradata2

240x320 uint16

	94	95	96	97	98	99	100	101	102	103	104	105	106	107
40	560	461	322	197	32	65535	23	65516	65526	34	65529	38	92	20
41	493	335	183	115	94	65528	65529	42	65533	65520	65522	65	62	7
42	393	181	97	14	58	8	10	40	23	36	16	51	24	1
43	262	176	12	67	135	65472	65456	8	65512	42	44	58	51	20
44	265	94	35	20	80	27	12	14	65511	23	67	72	89	65527
45	244	139	125	65532	31	3	15	65511	57	28	104	46	27	5
46	272	113	66	39	65518	65501	10	19	65535	3	16	33	44	8
47	289	129	62	3	13	65495	11	65533	65485	65518	65473	65501	64	65503
48	392	228	80	10	12	51	65525	31	13	65502	65463	65517	65487	65478
49	426	283	77	64	61	65480	65525	65489	65489	65458	65514	65486	65495	65455
50	473	476	253	71	15	30	65519	65453	65494	65500	65423	65468	65469	65512
51	489	418	287	133	65521	65492	65520	65519	65494	65465	65444	65471	65472	65423
52	533	345	240	48	28	65492	6	65417	65498	65520	65444	65417	65497	65431
53	604	349	252	23	65509	6	65510	65408	65470	65443	65494	65396	65519	65452
54	457	349	203	50	65462	65503	65466	65510	65447	21	65433	65430	65420	65413
55	401	280	83	22	65481	65487	65471	65450	65428	65465	136	65472	65483	65399
56	360	180	9	65488	65472	65417	65444	65402	65448	65492	65462	65434	65440	65395
57	208	94	65506	65476	65479	65435	65520	65407	65407	65442	65462	65457	65448	65458
58	92	65510	65484	65420	65476	65383	65480	65414	65462	65435	65506	65426	65476	65427
59	4	65521	65471	65457	65452	65419	65451	65420	65411	65421	65479	65378	65500	65432
60	65505	65495	65518	65498	65457	65469	65451	65409	65342	65457	65465	65378	65511	65431
61	65527	65518	65475	65496	65517	65446	65472	65422	65426	65440	65460	65423	65469	65420
62	65475	65496	65476	65472	65505	38	65431	65401	65446	65460	65378	65443	65473	65438
63	65475	65484	65471	65450	65459	65411	65443	65464	65455	65410	65449	65401	65425	65407
64	65492	65527	65468	15	65464	65449	65371	65441	65461	65442	65406	65401	65472	65402
65	65458	65474	65440	44	15	65401	65425	65398	65394	65431	171	65439	65435	65414
66	65470	65436	65415	65430	0	65464	65469	65396	65419	65461	65414	65461	65428	65445
67	65442	65415	65440	65450	65479	65449	65455	65437	65460	65471	65405	65414	65436	65425

Figure 108 Showing the raw data produced by the seek pro at the edges of a cold object

It was notices that the cold values were close to the value of 2^{16} (65536) and since the data is in a uint16 format, this indicated the values of the cold regions were below zero and looped around to the maximum values. Since the cold values in the images appeared as extremely high and the rescale function of the IRCam.py script determined the rest of the temperature values of the image based on the maximum and minimum value of the data captured, it was found that the there was an extremely large differential between the cold, now extremely high values, regions and the hot, now relatively low values, regions of the image which effectively created a bitmap showing only the cold regions within an image.

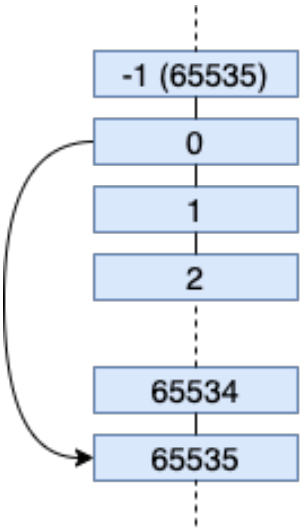


Figure 109 Showing how the data loops in 'uint' structures

This was fixed in the IRCam.py by shifting all the values of the image by adding half the 16-bit range to the input image data and then clipping any maxima values to 65535 and minima values to 1.

This modification of the code can be seen here:

Code Excerpt 19 Modified rescale function to improve results of cold objects

```
def rescale(self, img):
    """
    To adapt the range of values to the actual min and max and cast it into
    an 8 bits image
    """
    #shifting the range of the image to half the range of 16 bit numbers
    img = img+32768
    #Clipping the image to prevent looping the uint16 value below 0 or above
     65536
    imgClip = np.clip(img,1,65535)
    if img is None:
        return np.array([0])
    mini = imgClip.min()
    maxi = imgClip.max()
    # Scaling the image to an 8 bit 'greyscale' range
   imgScale = ((imgClip/(maxi-mini)*255.)).astype(np.uint8)
    return imgScale
```

12.2 Thermal Camera Temperature Performance

It was found that the thermal camera would lose its ability to identify thermal regions well with time. This was especially the case in the enclosure developed to hold the cameras in place.

A test was done to observe the performance of the camera over time.

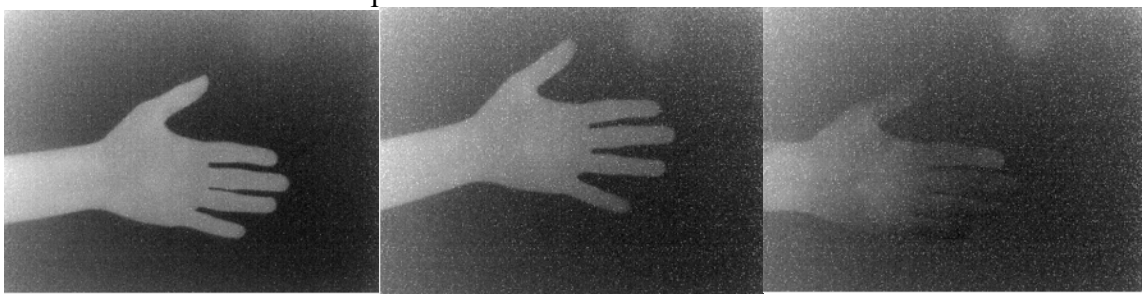


Figure 110 Thermal images captured at 0, 10 and 20 minutes with all cameras turned on

The temperature differential between the hand and the surrounds appearing to reduce and the noise within the image beginning to significantly increase. This is assumed to occur due to the thermal lens heating up with the camera module attached to it.
This test was done once again with the thermal camera turned on just by itself but still within the enclosure.

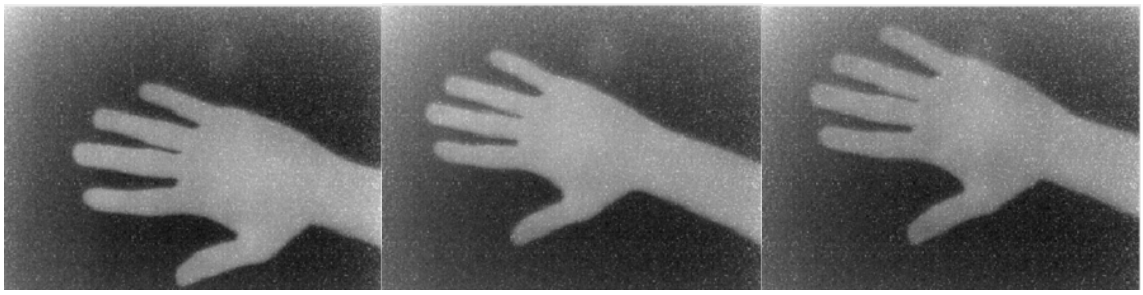


Figure 111 Thermal images captured at 0, 10 and 20 minutes with the thermal camera inside the enclosure

Following this, the test was recompleted with the thermal camera outside of the enclosure that was developed.

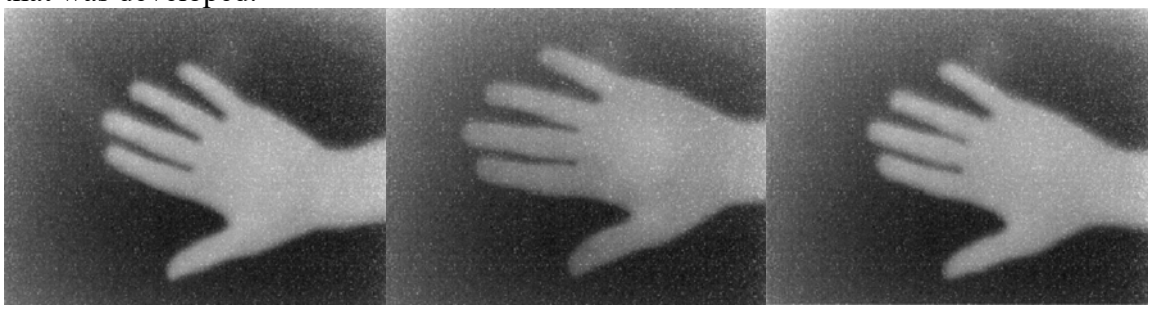


Figure 112 Thermal images captured at 0, 10 and 20 minutes with the thermal camera outside the enclosure

Although this performance is nowhere near as bad as when the camera was in the enclosure with the full system turned on, there is still a noticeable increase in temperature around the left edges of the thermal camera.
A final test was completed to observer the quality of the images produced by the SeekPro after being placed into the fridge to cool down the lens.

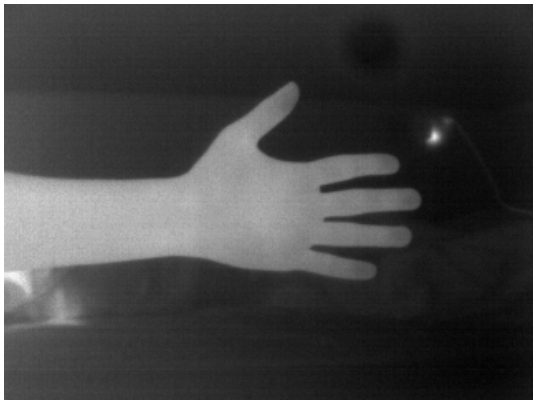


Figure 113 Image Capture after the thermal camera was placed into the fridge

There is a very surprising improvement in the thermal image quality. This identified that the thermal camera's performance would be directly affected by the surrounding temperatures and shows how an actively cooled system would improve the thermal results produced by the system.

13 Appendices 4 – SARCam Code

13.1 droneCam.py

```
##Giving OS Permissions to the script for the thermal camera to function
import os, sys, subprocess
if os.geteuid() != 0:
    subprocess.call(['sudo', 'python3'] + sys.argv)

#libraries for general image/data manipulation
import cv2
import numpy as np
# from imutils import contors
import imutils

#importing the saleincy library
from saliency import findSaliencyFineGrained, findSaliencySpectralResidual

#other general libraries
from time import time
import datetime
import concurrent.futures

#For visual cameras capture
import RGBCam
#For thermal camera capture
import IRCam

##Accessing the camera classes
PiCam = RGBCam.PiCam()
SeekPro = IRCam.SeekPro()
WebCam = RGBCam.WebCam(src=1).start()

##Getting the Saleincy functions from the saleincy script
findSaliencyFineGrained = findSaliencyFineGrained()
findSaliencySpectralResidual = findSaliencySpectralResidual()
contourProcessing = contourProcessing()
#setting the threshold for the saliency algorithm and initiating the saleincy counter
threshold = 65
saliencyCount = 1

##Calibration for the thermal camera
SeekProCalib    = np.loadtxt('/home/pi/SARCam/CameraCalibration/Images/SeekPro/cameraMatrix.txt', delimiter=',')
SeekProDist     = np.loadtxt('/home/pi/SARCam/CameraCalibration/Images/SeekPro/cameraDistortion.txt',
delimiter=',')
SeekProUnDisMat, roiSeekPro = cv2.getOptimalNewCameraMatrix(SeekProCalib,SeekProDist,(RESOLUTION),1,(RESOLUTION))

##Loading in the moving and fixed points of corresponding points between the thermal and visual image
movingPoints    = np.loadtxt('/home/pi/SARCam/movingPoints.txt', delimiter=',')
fixedPoints     = np.loadtxt('/home/pi/SARCam/fixedPoints.txt', delimiter=',')

##Generating the perspective transformation matrix using the moving and fixed points
tform, status = cv2.findHomography(movingPoints, fixedPoints)

purpMap = np.loadtxt('/home/pi/SARCam/purpColCustom3.txt', delimiter=',').astype(int)


class imageModification():
    '''class to apply image transfomrations and rotations'''
    def imageRotation(self, image, angle):
        ''' Rotates the image by a desired angle incase cameras are flipped'''
        (h, w) = image.shape[:2]
        centre = (w/2,h/2)
        RotMat = cv2.getRotationMatrix2D(centre, angle, 1)
        RotImg = cv2.warpAffine(image, RotMat,(w, h))
        return RotImg

    def imageCropping(self, img, roi):
        ''' function to crop the image to the centre of the thermal image  '''
        ## Cropping regions that are not in the image after, correction
        x,y,w,h = roi
        img = img[y:y+h, x:x+w]
        return img

    def customColourMap(self, img, LUT=purpMap):
        newImg = np.zeros((img.shape[0],img.shape[1],3)).astype(int)
        if img.shape[3] != 1:
            print(f'the image shape is incorrect for the custom colormap')
            return

        for y in range (0,img.shape[0]):
            for x in range (0, img.shape[1]):
                newImg[y][x] = LUT[img[y][x]]

        return newImg

    def imageFusion(self, background, foreground):
        if background.shape == foreground.shape:
            ## Adds the images together (has a transarancy factor)
```

```
            fusedImg = cv2.addWeighted(background,1,foreground,0.5,0)
            return fusedImg
        else :
            print('shape of background image and foreground are not the same')
            print(f'Background shape is :{background.shape}')
            print(f'Foreground shape is :{foreground.shape}')

    def saliencyGimbalCommand(self,img, threshold):
        #getting the saleincy maps from the openCV saliency Models
        (saliencyMap, threshMap) = findSaliencyFineGrained.getSaliency(img,threshold)
        (saliencyMap2, threshMap2) = findSaliencySpectralResidual.getSaliency(img,threshold)

        #getting the contours from the Spectral-Residual image
        (sortedImage, boundingBoxes) = contourProcessing.getContours(threshMap, saliencyMap2)
        gimbalMove = findSaliency.getContorPosition(sortedImage,boundingBoxes)
        return (saliencyMap, saliencyMap2, threshMap, threshMap2, sortedImage, boundingBoxes, gimbalMove)

    def skinDetector(self,img):
        '''Function to identify the skin of a person in an image'''
        #Skinmask HSV range
        lower = np.array([0, 48, 80], dtype = "uint8")
        upper = np.array([30, 255, 255], dtype = "uint8")

        # resize the frame, convert it to the HSV color space,
        # and determine the HSV pixel intensities that fall into
        # the speicifed upper and lower boundaries
        converted = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        skinMask = cv2.inRange(converted, lower, upper)

        # apply a series of erosions and dilations to the mask
        # using an elliptical kernel
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 9))
        skinMask = cv2.erode(skinMask, kernel, iterations = 2)
        skinMask = cv2.dilate(skinMask, kernel, iterations = 2)

        # blur the mask to help remove noise, then apply the
        # mask to the frame
        skinMask = cv2.GaussianBlur(skinMask, (7, 7), 0)
        skin = cv2.bitwise_and(frame, frame, mask = skinMask)

        return skin

    def thermHumanDetector(self, img):
        # apply a series of erosions and dilations to the mask
        # using an elliptical kernel
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 9))
        skinMask = cv2.erode(skinMask, kernel, iterations = 2)
        skinMask = cv2.dilate(skinMask, kernel, iterations = 2)

        # blur the mask to help remove noise, then apply the
        # mask to the frame
        skinMask = cv2.GaussianBlur(skinMask, (7, 7), 0)
        skin = cv2.bitwise_and(frame, frame, mask = skinMask)
        return skin


class CameraProcessing():

    def PiCamPost(self):
        '''post processing for the PiCam'''
        global WideImg
        ## Grabbing PiCam Image
        WideImg = PiCam.frameCapture()

        ## Image Processing region for the Wide Camera
        #PiCam image rotation of 180 degrees
        WideImg = imageModification().imageRotation(WideImg,180)
        return

    def WebCamPost(self):
        '''post processing for the WebCam'''
        global NarrowImg
        ##Grabbing WebCam Image
        NarrowImg = WebCam.read()

        #rescaling the image to default size
        NarrowImg = imutils.resize(NarrowImg, width=320)
        return

    def SeekProPost(self):
        '''post processing for the SeekPro before introduced'''
        global IRImg
        global IRWarp
        ##Grabbing SeekPro Image
        IRImg = SeekPro.rescale(SeekPro.get_image())

        ##Image processing of thermal image
        #Revoming image distortion
        IRWarp = cv2.undistort(IRImg, SeekProCalib, SeekProDist, None, SeekProUnDisMat)
        #Applying Perspective transformation matrix to overlay thermalimage onto visual
        IRWarp = cv2.warpPerspective(IRImg,tform,(RESOLUTION))
```

```
        return

if __name__ == '__main__':
    from time import time
    from time import strftime
    from time import sleep
    import os
    import concurrent.futures

    t0 = time()

    def saliencyImplement(img, threshold):
        '''implementing the OpenCV saliency function from the saleincyCV library'''
        try:
            saliencyMap, saliencyMap2, threshMap, threshMap2, sortedImage, boundingBoxes, gimbalMove =
imageModification().saliencyGimbalCommand(img, threshold)

        except:
            print('GimbalMovement failed')

    def thermalSmoothing(img):
        '''function to apply smoothing to the thermal image'''
        IRsmooth = cv2.medianBlur(img,3)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
        IRsmooth = cv2.erode(IRsmooth, kernel, iterations = 3)
        IRsmooth = cv2.dilate(IRsmooth, kernel, iterations = 3)
        IRsmooth = cv2.medianBlur(IRsmooth,5)
        IRsmooth = cv2.medianBlur(IRsmooth,5)
        IRsmooth = cv2.medianBlur(IRsmooth,3)
        return IRsmooth

    def flooding(img):
        '''function to find the edged of an thermak object and fill it with white'''
        edges = cv2.Canny(img,50,255)

        # Copy the thresholded image.
        im_floodfill = edges.copy()

        # Mask used to flood filling.
        # Notice the size needs to be 2 pixels than the image.
        h, w = edges.shape[:2]
        mask = np.zeros((h+2, w+2), np.uint8)

        # Floodfill from point (0, 0)
        cv2.floodFill(im_floodfill, mask, (0,0), 255)

        # Invert floodfilled image
        im_floodfill_inv = cv2.bitwise_not(im_floodfill)

        # Combine the two images to get the foreground.
        im_out = edges | im_floodfill_inv
        return im_out

    while True:
        #Showing the framrate of the system
        t = time()
        print("fps:",1/(t-t0))
        t0 = time()

        #Images being captured using threads to prevent bottlenecks within image capture stag
        with concurrent.futures.ThreadPoolExecutor() as executor:
            executor.submit(CameraProcessing().SeekProPost())
            executor.submit(CameraProcessing().PiCamPost())
            executor.submit(CameraProcessing().WebCamPost())
        IR1 = IRImg
        RGB1 = WideImg
        RGB2 = NarrowImg
        IRWarp = IRWarp



        IRColor = cv2.applyColorMap(IRWarp, cv2.COLORMAP_RAINBOW)
        IRFused = imageModification().imageFusion(RGB1,IRColor)

        ##Applying the colourmap
        IRColor = cv2.applyColorMap(IRWarp, pupMat)
        IRFusedCropped = imageModification().imageCropping(IRFused, (47,55,190,140))

        ##Implementing Saliency
        with concurrent.futures.ThreadPoolExecutor() as executor:
            executor.submit(saliencyImplement(RGB1, 55))

       ##Choosing the images to show
        # IRColor = imageModification().customColourMap(IR1,purpMap)
        cv2.imshow("piCam", RGB1)       #Showing the PiCam Image
        # cv2.imshow("webCam", RGB2)    #Showing the WebCam Image
        cv2.imshow("seekPro",IR1)       #Showing the SeekPro Image
        cv2.imshow("seekProWarp",IRWarp)#Showing the transformed SeekPro Image
        cv2.imshow("irColor", IRColor)  #Showing the SeekPro with colourmap
        # cv2.imshow("Fused", IRFused)  #Showing the fused PiCam + SeekPro image
```

```
        ##Exit Code
        # if the `q` key was pressed, break from the loop
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            print("[INFO] Exit key 'q' has been pressed")
            WebCam.stop()
            PiCam.stop()
            break
        cv2.waitKey(1)


    # CameraProcessing.ReleaseCameras()
    WebCam.stop()
    PiCam.stop()
    cv2.destroyAllWindows()

    print("[INFO] RGB Cameras Released √")
```

13.2 IRCam.py

```
# Original Author: Victor Couty
# Modified by: Omar Ali

### This module is where the thermal camera image can be captured

##Giving OS Permissions
import os
import sys
import subprocess

if os.geteuid() != 0:
    subprocess.call(['sudo', 'python3'] + sys.argv)

import usb.core
import usb.util
import numpy as np
import cv2

# Address enum
READ_CHIP_ID                    = 54 # 0x36
START_GET_IMAGE_TRANSFER        = 83 # 0x53

GET_OPERATION_MODE              = 61 # 0x3D
GET_IMAGE_PROCESSING_MODE       = 63 # 0x3F
GET_FIRMWARE_INFO               = 78 # 0x4E
GET_FACTORY_SETTINGS            = 88 # 0x58

SET_OPERATION_MODE              = 60 # 0x3C
SET_IMAGE_PROCESSING_MODE       = 62 # 0x3E
SET_FIRMWARE_INFO_FEATURES      = 85 # 0x55
SET_FACTORY_SETTINGS_FEATURES   = 86 # 0x56

WIDTH = 320
HEIGHT = 240
RAW_WIDTH = 342
RAW_HEIGHT = 260


class SeekPro():
  """
  Seekpro class:
    Can read images from the Seek Thermal pro camera
    Can apply a calibration from the integrated black body
    Can locate and remove dead pixels
  This class only works with the PRO version !
  """
  def __init__(self):
    self.dev = usb.core.find(idVendor=0x289d, idProduct=0x0011)
    if not self.dev:
      raise IOError('Device not found')
    self.dev.set_configuration()
    self.calib = None
    for i in range(5):
      # Sometimes, the first frame does not have id 4 as expected...
      # Let's retry a few times
      if i == 4:
        # If it does not work, let's forget about dead pixels!
        print("Could not get the dead pixels frame!")
        self.dead_pixels = []
        break
      self.init()
      status,ret = self.grab()
      if status == 4:
        self.dead_pixels = self.get_dead_pix_list(ret)
        break

  def get_dead_pix_list(self,data):
    """
    Get the dead pixels image and store all the coordinates
```

```
    of the pixels to be corrected
    """
    img = self.crop(np.frombuffer(data,dtype=np.uint16).reshape(
          RAW_HEIGHT,RAW_WIDTH))
    return list(zip(*np.where(img<100)))

  def correct_dead_pix(self,img):
    """For each dead pix, take the median of the surrounding pixels"""
    for i,j in self.dead_pixels:
      img[i,j] = np.median(img[max(0,i-1):i+2,max(0,j-1):j+2])
    return img

  def crop(self,raw_img):
    """Get the actual image from the raw image"""
    return raw_img[4:4+HEIGHT,1:1+WIDTH]

  def send_msg(self,bRequest,  data_or_wLength,
      wValue=0, wIndex=0,bmRequestType=0x41,timeout=None):
    """
    Wrapper to call ctrl_transfer with default args to enhance readability
    """
    assert (self.dev.ctrl_transfer(bmRequestType, bRequest, wValue, wIndex,
      data_or_wLength, timeout) == len(data_or_wLength))

  def receive_msg(self,bRequest, data, wValue=0, wIndex=0,bmRequestType=0xC1,
      timeout=None):
    """
    Wrapper to call ctrl_transfer with default args to enhance readability
    """
    return self.dev.ctrl_transfer(bmRequestType, bRequest, wValue, wIndex,
          data, timeout)

  def deinit(self):
    """
    Is it useful ?
    """
    for i in range(3):
        self.send_msg(0x3C, b'\x00\x00')

  def init(self):
    """
    Sends all the necessary data to init the camera
    """
    self.send_msg(SET_OPERATION_MODE, b'\x00\x00')
    #r = receive_msg(GET_FIRMWARE_INFO, 4)
    #print(r)
    #r = receive_msg(READ_CHIP_ID, 12)
    #print(r)
    self.send_msg(SET_FACTORY_SETTINGS_FEATURES, b'\x06\x00\x08\x00\x00\x00')
    #r = receive_msg(GET_FACTORY_SETTINGS, 12)
    #print(r)
    self.send_msg(SET_FIRMWARE_INFO_FEATURES,b'\x17\x00')
    #r = receive_msg(GET_FIRMWARE_INFO, 64)
    #print(r)
    self.send_msg(SET_FACTORY_SETTINGS_FEATURES, b"\x01\x00\x00\x06\x00\x00")
    #r = receive_msg(GET_FACTORY_SETTINGS,2)
    #print(r)
    for i in range(10):
      for j in range(0,256,32):
        self.send_msg(
            SET_FACTORY_SETTINGS_FEATURES,b"\x20\x00"+bytes([j,i])+b"\x00\x00")
        #r = receive_msg(GET_FACTORY_SETTINGS,64)
        #print(r)
    self.send_msg(SET_FIRMWARE_INFO_FEATURES,b"\x15\x00")
    #r = receive_msg(GET_FIRMWARE_INFO,64)
    #print(r)
    self.send_msg(SET_IMAGE_PROCESSING_MODE,b"\x08\x00")
    #r = receive_msg(GET_IMAGE_PROCESSING_MODE,2)
    #print(r)
    self.send_msg(SET_OPERATION_MODE,b"\x01\x00")
    #r = receive_msg(GET_OPERATION_MODE,2)
    #print(r)

  def grab(self):
    """
    Asks the device for an image and reads it
    """
    # Send read frame request
    self.send_msg(START_GET_IMAGE_TRANSFER, b'\x58\x5b\x01\x00')
    toread = 2*RAW_WIDTH*RAW_HEIGHT
    ret  = self.dev.read(0x81, 13680, 1000)
    remaining = toread-len(ret)
    # 512 instead of 0, to avoid crashes when there is an unexpected offset
    # It often happens on the first frame
    while remaining > 512:
      #print(remaining," remaining")
      ret += self.dev.read(0x81, 13680, 1000)
      remaining = toread-len(ret)
    status = ret[4]
    if len(ret) == RAW_HEIGHT*RAW_WIDTH*2:
      return status,np.frombuffer(ret,dtype=np.uint16).reshape(
            RAW_HEIGHT,RAW_WIDTH)
```

```
    else:
      return status,None

  def get_image(self):
    """
    Method to get an actual IR image
    """
    while True:
      status,img = self.grab()
      if status == 1: # Calibration frame
        self.calib = self.crop(img)-1600
      elif status == 3: # Normal frame
        if self.calib is not None:
          return self.correct_dead_pix(self.crop(img)-self.calib)


  def rescale(self, img):
    """
    To adapt the range of values to the actual min and max and cast it into
    an 8 bits image
    """
    #shifting the range of the image to half the range of 16 bit numbers
    img = img+32768

    #Clipping the image to prevent looping the uint16 value below 0 or above 65536
    imgClip = np.clip(img,1,65535)

    if img is None:
        return np.array([0])
    mini = imgClip.min()
    maxi = imgClip.max()

    # Scaling the image to an 8 bit 'greyscale' range
    imgScale = ((np.clip(img-mini,0,maxi-mini)/(maxi-mini)*255.)).astype(np.uint8)
    return imgScale

if __name__ == '__main__':

  from time import time
  from time import strftime
  from time import sleep
  # Setting thermal camera
  IRCam = SeekPro()
  cv2.namedWindow("Seek",cv2.WINDOW_NORMAL)
  t0 = time()

  while True:
      t = time()
      print("fps:",1/(t-t0))
      t0 = time()

      IRimg = IRCam.rescale(IRCam.get_image())
      cv2.imshow("Seek",IRimg)

      timestr = strftime("%d%m%Y-%H%M%S")

      if cv2.waitKey(1) & 0xFF == ord('q'):
            break
      cv2.waitKey(1)
```

13.3 RGBCam.py

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import cv2
import numpy as np
import imutils
from threading import Thread
import time

RESOLUTION = 320, 240
FRAMERATE = 25

import datetime

class PiCam():
    def __init__(self):
        # initialize the camera and grab a reference to the raw camera capture
        self.camera = PiCamera()
        self.camera.resolution = RESOLUTION
        self.camera.framerate = FRAMERATE
        #Convert Data into readable array
        self.rawCapture = PiRGBArray(self.camera, RESOLUTION)
        #Camera Warmup Time
        time.sleep(0.1)

    def frameCapture(self):
        '''Capturing the RGB Image'''
```

```
        #Clearing the image stream for the next frame (otherwise there will be a buffer error)
        self.rawCapture.truncate(0)
        #Capturing a frame from the video stream to display
        for frame in self.camera.capture_continuous(self.rawCapture, format="bgr", use_video_port=True):
            img1 = frame.array
            return img1

    def stop(self):
        self.camera.close()


class WebCam():
    def __init__(self, src=1):
        # initialize the video camera stream and read the first frame
        # from the stream
        self.stream = cv2.VideoCapture(src)
        (self.grabbed, self.frame) = self.stream.read()
        # initialize the variable used to indicate if the thread should
        # be stopped
        self.stopped = False
    def start(self):
        # start the thread to read frames from the video stream
        Thread(target=self.update, args=()).start()
        return self
    def update(self):
        # keep looping infinitely until the thread is stopped
        while True:
            # if the thread indicator variable is set, stop the thread
            if self.stopped:
                return
            # otherwise, read the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()
    def read(self):
        # return the frame most recently read
        return self.frame
    def stop(self):
        # indicate that the thread should be stopped
        self.stopped = True
```

13.4 SaleincyCV2.py

```
import cv2
import numpy as np
from imutils import contours
import imutils

class findSaliencyFineGrained():
      def __init__(self):
            ''' Initiating the saliency model to be use for this process'''
            self.saliency = cv2.saliency.StaticSaliencyFineGrained_create()
            # self.saliency = cv2.saliency.StaticSaliencyBinWangApr2014_create()

      def getSaliency(self,img,threshperc):
            ''' function to capture the saliency in a given image and output a threshold map based on
            a given threshold percentage desire'''
            #Grabbing the saliency and converting it into a grescale integer value
            saliencyMap = np.array(255* self.saliency.computeSaliency(img)[1],dtype="uint8")
            #Creating a threshold percentatge to greyscale conversion
            threshpercconv = np.int((threshperc/100)*255)
            #applying thresholding the to top 'threshperc' of the image
            threshMap = cv2.threshold(saliencyMap,threshpercconv,255,cv2.THRESH_BINARY)[1]
            return saliencyMap, threshMap

class findSaliencySpectralResidual():
      def __init__(self):
            ''' Initiating the saliency model to be use for this process'''
            self.saliency = cv2.saliency.StaticSaliencySpectralResidual_create()

      def getSaliency(self,img,threshperc):
            ''' funcion to capture the saliency in a given image and output a threshold map based on
            a given threshold percentage desire'''
            #Grabbing the saliency and converting it into a grescale integer value
            saliencyMap = np.array(255*self.saliency.computeSaliency(img)[1],dtype="uint8")
            #Creating a threshold percentatge to grescale conversion
            threshpercconv = np.int((threshperc/100)*255)
            #applying thresholding the to top 'threshperc' of the image
            threshMap = cv2.threshold(saliencyMap,threshpercconv,255,cv2.THRESH_BINARY)[1]
            return saliencyMap, threshMap


class contourProcessing():
      def getContours(self,threshMap, origImg):
            '''Function to find all the thresholded regions ie Contours in a saleint image
            and to identify the possible ROI'''

            # Finding the countors in the image
            cnts = cv2.findContours(threshMap, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
            cnts = imutils.grab_contours(cnts)
```

```
            #Sorting and numbering the Contours starting from the bottom of the image
            method = "bottom-to-top"
            (cnts, boundingBoxes) = contours.sort_contours(cnts, method=method)

            #Labelling the image with contour positions
            for (i, c) in enumerate(cnts):
                  sortedImage = contours.label_contour(origImg, c, i, color=(240, 0, 159))
            return sortedImage, boundingBoxes



      def getContourPosition(self, img, boundingBoxes, Hsignal=0, Vsignal=0):
            '''Generates a direction command for the gimabal to move depending on the concentration
            of points in the image '''

            #Left Right
            L, C, R = 0, 0, 0
            #Up Down
            T, M, B = 0, 0, 0

            #Initialising array
            try:
                  ROIPoints = np.zeros(shape=(len(boundingBoxes),2), dtype='uint16')
            except:
                  print('no ROI points')

            #Whatever image input is placed into here
            imgshape = img.shape

            #Creating a tally of positions where the most interesting features are
            for i in range (0,len(boundingBoxes)):
                  try:
                        #Saving the points of the image that
                        ROIPoints[i][0] = boundingBoxes[i][0]
                        if ROIPoints[i][0] <= int(imgshape[1]*1/3):
                              L = L + 1
                        elif ROIPoints[i][0] > int(imgshape[1]*1/3) and ROIPoints[i][0] < int(imgshape[1]*2/3):
                              C = C + 1
                        elif ROIPoints[i][0] >= int(imgshape[1]*2/3):
                              R = R + 1
                        else:
                              print('horizontal positional error')

                        ROIPoints[i][1] = boundingBoxes[i][1]
                        #Remember downwards positive axis
                        if ROIPoints[i][1] >= int(imgshape[0]*2/3):
                              B = B + 1
                        elif ROIPoints[i][1] > int(imgshape[0]*1/3) and ROIPoints[i][1] < int(imgshape[0]*2/3):
                              M = M + 1
                        elif ROIPoints[i][1] <= int(imgshape[0]*1/3):
                              T = T + 1
                        else:
                              print('vertical positional error')
                  except:
                        print('error in the ROI Location append loop')

            if L > C or L > R:
                  Hsignal = Hsignal-1
            elif R > C or R > L:
                  Hsignal = Hsignal+1
            else:
                  Hsignal = Hsignal
            if T > M or T > B:
                  Vsignal = Vsignal+1
            elif B > M or B > T:
                  Vsignal = Vsignal-1
            else:
                  Vsignal = Vsignal

            signal = Hsignal, Vsignal
            return signal

if __name__ == '__main__':

      from time import time
      from time import time
      t = 0
      t0 =0
      findSaliencyFineGrained = findSaliencyFineGrained()
      findSaliencySpectralResidual = findSaliencySpectralResidual()
      threshold = 100
      i = 0

      from RGBCam import PiCam
      PiCam = PiCam()

      signal = 0,0
      while True:
            t = time()
            # print("fps:",1/(t-t0))
```

```
            t0 = time()
            RGB = PiCam.frameCapture()

            if cv2.waitKeyEx(1) & 0xFF == ord('w'):
                  threshold = threshold + 5
            if cv2.waitKeyEx(1) & 0xFF == ord('s'):
                  threshold = threshold - 5

            #image saliency and thresholding
            (saliencyMap, threshMap) = findSaliencyFineGrained.getSaliency(RGB,threshold)
            (saliencyMap2, threshMap2) = findSaliencySpectralResidual.getSaliency(saliencyMap,threshold)

            #finding image contors
            contourImg = RGB.copy()
            try:
                  sortedImage, boundingBoxes = contourProcessing().getContours(threshMap2, contourImg)

            except:
                  # print('image contoring error')
                  print('')

            #Generating the direction command for the gimabal signal
            try:
                  signal = contourProcessing().getContourPosition(sortedImage, boundingBoxes, signal[0],signal[1])
            except:
                  print('no signal generated')

            #After 10 iterations identify the position of the region of interest
            if i == 10:
                  # signal = contourProcessing().getContourPosition(sortedImage, boundingBoxes,
signal[0],signal[1])
                  if signal[0] < -3:
                        print('left')
                  elif  signal[0] > 3:
                        print('right')
                  if signal[1] < -3:
                        print('down')
                  elif  signal[1] > 3:
                        print('up')
                  if signal[0] == 0 and signal[1] == 0:
                        print('stay')
                  #reset iterations and signal
                  i = 0
                  signal = 0,0
            i = i+1
            cv2.imshow('RGB',RGB)
            cv2.imshow('Contour Image',contourImg)
            cv2.imshow("Fine Grained",saliencyMap)
            cv2.imshow("Spectral-Residual",saliencyMap2)

            cv2.imshow("Fine Grained Threshold",threshMap)
            cv2.imshow("Spectral-Residual Threshold",threshMap2)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                  break
            cv2.waitKey(1)
```

13.5 SaliencyTF.py

```
# Boilerplate imports.
import tensorflow.compat.v1 as tf
import numpy as np
import PIL.Image
from matplotlib import pylab as P
import pickle
import os
import sys
import cv2
import tf_slim as slim
import time

old_cwd = os.getcwd()
sys.path.insert(1, 'models/research/slim/')
from nets import inception_v3

# From our PAIRML repository
from saliencydev import saliency

def LoadImage(file_path):
  im = cv2.imread(file_path)
  im = np.asarray(im)
  return im / 255 #127.5 - 1.0

ckpt_file = './inception_v3.ckpt'
##Adding a single logit tensor for which we want to compute the mask
graph = tf.Graph()

with graph.as_default():
  images = tf.placeholder(tf.float32, shape=(None, 299, 299, 3))#240,320,3))#
```

```
  with slim.arg_scope(inception_v3.inception_v3_arg_scope()):
    _, end_points = inception_v3.inception_v3(images, is_training=False, num_classes=1001)

    # Restore the checkpoint
    sess = tf.Session(graph=graph)
    saver = tf.train.Saver()
    saver.restore(sess, ckpt_file)

  # Construct the scalar neuron tensor.
  logits = graph.get_tensor_by_name('InceptionV3/Logits/SpatialSqueeze:0')
  neuron_selector = tf.placeholder(tf.int32)
  y = logits[0][neuron_selector]

  # # Construct tensor for predictions.
  prediction = tf.argmax(logits, 1)


## INPUT IMAGE SAMPLE HERE
im = LoadImage('/PATH/TO/IMAGE/OF/INTEREST')
im = cv2.resize(im, (299,299))
cv2.imshow('img',im)




## Make a prediction.
prediction_class = sess.run(prediction, feed_dict = {images: [im]})[0]

#Set to baseball player
prediction_class = 981
t0 = time.perf_counter()


#_____________________________________________________________________________________________________#
# ##Vanilla Gradient & SmoothGrad
# Construct the saliency object. This doesn't yet compute the saliency mask, it just sets up the necessary ops.

print('Vanilla Gradient & SmoothGrad START')
gradient_saliency = saliency.GradientSaliency(graph, sess, y, images)

# # Compute the vanilla mask and the smoothed mask.
vanilla_mask_3d = gradient_saliency.GetMask(im, feed_dict = {neuron_selector: prediction_class})
smoothgrad_mask_3d = gradient_saliency.GetSmoothedMask(im, feed_dict = {neuron_selector: prediction_class})

# # Call the visualization methods to convert the 3D tensors to 2D grayscale.
vanilla_mask_grayscale = saliency.VisualizeImageGrayscale(vanilla_mask_3d)
smoothgrad_mask_grayscale = saliency.VisualizeImageGrayscale(smoothgrad_mask_3d)

cv2.imshow('Vanilla Gradient',vanilla_mask_grayscale)
cv2.imshow('SmoothGrad',smoothgrad_mask_grayscale)

print('Vanilla Gradient & SmoothGrad END')
t1 = time.perf_counter()
#_____________________________________________________________________________________________________#
# ## Guided Backprop & SmoothGrad
# # Construct the saliency object. This doesn't yet compute the saliency mask, it just sets up the necessary ops.
# # NOTE: GuidedBackprop creates a copy of the given graph to override the gradient.
# # Don't construct too many of these!

print('Guided Backprop & SmoothGrad START')
guided_backprop = saliency.GuidedBackprop(graph, sess, y, images)

# # Compute the vanilla mask and the smoothed mask.
vanilla_guided_backprop_mask_3d = guided_backprop.GetMask(
  im, feed_dict = {neuron_selector: prediction_class})

smoothgrad_guided_backprop_mask_3d = guided_backprop.GetSmoothedMask(
  im, feed_dict = {neuron_selector: prediction_class})



# # Call the visualization methods to convert the 3D tensors to 2D grayscale.
vanilla_mask_grayscale = saliency.VisualizeImageGrayscale(vanilla_guided_backprop_mask_3d)
smoothgrad_mask_grayscale = saliency.VisualizeImageGrayscale(smoothgrad_guided_backprop_mask_3d)

cv2.imshow('Vanilla Guided Backprop',vanilla_mask_grayscale)
cv2.imshow('SmoothGrad Guided Backprop',smoothgrad_mask_grayscale)

print('Guided Backprop & SmoothGrad END')
t2 = time.perf_counter()
#_____________________________________________________________________________________________________#
# Integrated Gradients & SmoothGrad
print('Integrated Gradients & SmoothGrad START')

# Construct the saliency object. This doesn't yet compute the saliency mask, it just sets up the necessary ops.
integrated_gradients = saliency.IntegratedGradients(graph, sess, y, images)

# Baseline is a black image.
baseline = np.zeros(im.shape)
```

```
baseline.fill(-1)

# Compute the vanilla mask and the smoothed mask.
vanilla_integrated_gradients_mask_3d = integrated_gradients.GetMask(
  im, feed_dict = {neuron_selector: prediction_class}, x_steps=25, x_baseline=baseline)
# Smoothed mask for integrated gradients will take a while since we are doing nsamples * nsamples computations.
smoothgrad_integrated_gradients_mask_3d = integrated_gradients.GetSmoothedMask(
  im, feed_dict = {neuron_selector: prediction_class}, x_steps=25, x_baseline=baseline)

# Call the visualization methods to convert the 3D tensors to 2D grayscale.
vanilla_mask_grayscale = saliency.VisualizeImageGrayscale(vanilla_integrated_gradients_mask_3d)
smoothgrad_mask_grayscale = saliency.VisualizeImageGrayscale(smoothgrad_integrated_gradients_mask_3d)



cv2.imshow('Vanilla Integrated Gradients',vanilla_mask_grayscale)
cv2.imshow('Smoothgrad Integrated Gradients',smoothgrad_mask_grayscale)

print('Integrated Gradients & SmoothGrad END')
t3 = time.perf_counter()

#_____________________________________________________________________________________________________#
## Displaying the time taken to run
print(f'Time taken for smoothGrad is {t1-t0}')
print(f'Time taken for Vanilla Guided Backprop is {t2-t1}')
print(f'Time taken for Smoothgrad Integrated Gradients is {t3-t2}')

#_____________________________________________________________________________________________________#

def rescaler(old_value):
  ''''function to rescale the images produced to a 0 to 1 range'''
  old_min   = old_value.min()
  old_max   = old_value.max()
  new_min   = 0
  new_max   = 1
  new_value = (( (old_value - old_min)/(old_max - old_min) ) * (new_max - new_min) + new_min).astype(np.float)
  return new_value

##rescaling the image to gresycale range
smoothgrad_rescaled = rescaler(smoothgrad_mask_grayscale)
smoothgrad_thresh = cv2.threshold(smoothgrad_rescaled,150,255,cv2.THRESH_BINARY)[1]
vanilla_mask_rescaled = rescaler(vanilla_mask_grayscale)
vanilla_mask_thresh = cv2.threshold(vanilla_mask_rescaled,100,255,cv2.THRESH_BINARY)[1]



# ##Centre of mass of the object
import scipy.ndimage as ndi
smoothY, smoothX = ndi.center_of_mass(smoothgrad_thresh)
vanillaY, vanillaX = ndi.center_of_mass(vanilla_mask_thresh)

##Displaying the center of mass of the smoothgrad and vanillamask
print(smoothY,smoothX)
print(vanillaY,vanillaX)
cv2.waitKey(0)
```

13.6 CameraCalibration.py

```
#!/usr/bin/env python

"""
From https://opencv-python-
tutroals.readthedocs.org/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration
Script to generate the calibration matrices for the cameras
"""

import numpy as np
import cv2
import glob
import sys
import argparse

#-- SET THE PARAMETERS

##For the PiCam
# nRows = 5 # number of box intersections on the rows
# nCols = 8 # number of box intersections on the columns
# dimension = 50 # size of the boxes
# workingFolder = "./Images/PiCam" # path to image file
# imageType = 'png' #format of the image

##For the seekPro
nRows = 4 # number of box intersections on the rows
nCols = 5 # number of box intersections on the columns
dimension = 50 # size of the boxes
workingFolder = "./Images/SeekPro" # path to image file
imageType = 'png' #format of the image
```

```
# Termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, dimension, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((nRows*nCols,3), np.float32)
objp[:,:2] = np.mgrid[0:nCols,0:nRows].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
if len(sys.argv) < 6:
        print("\n Not enough inputs are provided. Using the default values.\n\n" \
              " type -h for help")
else:
    workingFolder   = sys.argv[1]
    imageType       = sys.argv[2]
    nRows           = int(sys.argv[3])
    nCols           = int(sys.argv[4])
    dimension       = float(sys.argv[5])

# Find the images files
filename    = workingFolder + "/*." + imageType
images      = glob.glob(filename)

print(filename)
print(images)

if len(sys.argv) < 6:
        print("\n Not enough inputs are provided. Using the default values.\n\n" \
              " type -h for help")
else:
    workingFolder   = sys.argv[1]
    imageType       = sys.argv[2]
    nRows           = int(sys.argv[3])
    nCols           = int(sys.argv[4])
    dimension       = float(sys.argv[5])

if len(images) < 9:
    print("Not enough images were found: at least 9 shall be provided!!!")
    sys.exit()

else:
    nPatternFound = 0
    imgNotGood = images[1]

    for fname in images:
        if 'calibresult' in fname: continue
        #-- Read the file and convert in greyscale
        img     = cv2.imread(fname)
        gray    = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

        print("Reading image ", fname)

        # Find the chess board corners
        ret, corners = cv2.findChessboardCorners(gray, (nCols,nRows),None)

        # If found, add object points, image points (after refining them)
        if ret == True:
            print("Pattern found! Press ESC to skip or ENTER to accept")
            #--- Sometimes, Harris cornes fails with crappy pictures, so
            corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)

            # Draw and display the corners
            cv2.drawChessboardCorners(img, (nCols,nRows), corners2,ret)
            cv2.imshow('img',img)
            # cv2.waitKey(0)
            k = cv2.waitKey(0) & 0xFF

            if k == 27: #-- ESC Button
                print("Image Skipped")
                imgNotGood = fname
                continue

            print("Image accepted")
            nPatternFound += 1
            objpoints.append(objp)
            imgpoints.append(corners2)

            # cv2.waitKey(0)
        else:
            imgNotGood = fname

cv2.destroyAllWindows()


if (nPatternFound > 1):
    print("Found %d good images" % (nPatternFound))
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)

    # Undistort an image
    img = cv2.imread(imgNotGood)
```

```
    h,  w = img.shape[:2]
    print("Image to undistort: ", imgNotGood)
    newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))

    # undistort
    mapx,mapy = cv2.initUndistortRectifyMap(mtx,dist,None,newcameramtx,(w,h),5)
    dst = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)

    # crop the image
    x,y,w,h = roi
    dst = dst[y:y+h, x:x+w]
    print("ROI: ", x, y, w, h)

    cv2.imwrite(workingFolder + "/calibresult.png",dst)
    print("Calibrated picture saved as calibresult.png")
    print("Calibration Matrix: ")
    print(mtx)
    print(dist)
    print("Disortion: ", dist)

    #--------- Save result
    filename = workingFolder + "/cameraMatrix.txt"
    np.savetxt(filename, mtx, delimiter=',')
    filename = workingFolder + "/cameraDistortion.txt"
    np.savetxt(filename, dist, delimiter=',')

    mean_error = 0
    for i in range(len(objpoints)):
        imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
        error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
        mean_error += error

    print("total error: ", mean_error/len(objpoints))

else:
    print("In order to calibrate you need at least 9 good pictures... try again")
```