Fire and Water Stream Video Recognition Software to Redirect a Water Stream onto a Fire

By Fintan Lyons

04/05/2023

Supervised by Bilal Kaddouh

Department of Mechanical Engineering

The University of Leeds

SCHOOL OF MECHANICAL ENGINEERING

UNIVERSITY OF LEEDS

MECH3890 – Individual Engineering Project

PROJECT TITLE: Fire and Water Stream Video Recognition Software to Redirect a Water Stream onto a Fire

PRESENTED BY	Fintan Lyons

SUPERVISED BY	Bilal Kaddouh

If the project is industrially linked, tick this box and provide details below ☐

COMPANY NAME AND ADDRESS:

STUDENT DECLARATION (from the "LU Declaration of Academic Integrity")

I am aware that the University defines plagiarism as presenting someone else's work, in whole or in part, as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance. I promise that in the attached submission I have not presented anyone else's work, in whole or in part, as my own and I have not colluded with others in the preparation of this work. Where I have taken advantage of the work of others, I have given full acknowledgement. I have not resubmitted my own work or part thereof without specific written permission to do so from the University staff concerned when any of this work has been or is being submitted for marks or credits even if in a different module or for a different qualification or completed prior to entry to the University. I have read and understood the University's published rules on plagiarism and also any more detailed rules specified at School or module level. I know that if I commit plagiarism I can be expelled from the University and that it is my responsibility to be aware of the University's regulations on plagiarism and their importance. I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes. I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and that I wish to have taken into account. I am aware of the University's policy on mitigation and the School's procedures for the submission of statements and evidence of mitigation. I am aware of the penalties imposed for the late submission of coursework.

Date: 04/05/2023

Signed: 

Table of Contents

List of Figures

List of Tables

List of Graphs

Abstract

In 2022, there were 184,437 fires in England, which caused millions of pounds in damages and harmed hundreds of lives (Office, 2023). Fires continue to prove incredibly dangerous to society. To improve our firefighting abilities, the use of firefighting drones is being explored. One aspect of firefighting drones yet to be researched is the development of code which can redirect a water stream onto a fire. In order to redirect a water stream onto a fire, the water stream and the fire must be detected from a video feed. This project develops basic fire detection code and novel water stream detection code from recorded videos. These codes are subsequently combined to produce code which can accurately move a water stream onto a fire. This report presents the developed code and, the approach taken to create it, followed by demonstrating the results from experimental testing. Finally, the project discusses the codes limitations and scopes for improvement.

.

Chapter 1: Introduction

1.1 Introduction

Fires can be incredibly destructive, posing significant risks to human life, property, and the environment. Each year, fires cause millions of pounds worth of damage and claim thousands of lives worldwide. In England alone, there was a total of 185,437 fires and 276 fire-related deaths in 2022 (Office, 2023). Firefighting has become a crucial service within our communities with firefighters often putting their lives at risk. From 1986 to 2013, 26 firefighters died in England whilst fighting fires, exemplifying the dangers of firefighting (Office, 2023).

Traditionally, firefighting teams are deployed with fire engines, equipped with water, hoses and other specialist gear. The optimisation of firefighting techniques, paired with improvements in fire education and legislation has resulted in a 39% decrease in the number of fires over the last 18 years in England, from 474,000 in 2004 to 184,437 in 2022 (Office, 2023).

To further improve humanities firefighting capabilities and to stop endangering human lives whilst fighting fires, the use of firefighting drones is being explored. Firefighting drones have several benefits over traditional firefighting techniques, they can access areas inaccessible to humans, provide a better vantage point and give extensive information about the nature of the fire - such as its source. Currently firefighting drones are used for search and rescue, aerial surveillance and fire detection (Akhloufi et al., 2021) (Bullock, 2020).

Several companies are exploring the use of firefighting drones. Latvian company Aerones developed a firefighting drone which can climb 300 m in 6 minutes (Peter, 2018). The London Fire Brigade uses drones to provide an aerial view of incidents, improving their response (London Fire Brigade, 2023). Chinese company EHang has developed a manned firefighting drone called the EHang 216F which can deliver 6 fire extinguisher projectiles and a high-pressure water jet using a laser aiming device (EHang, 2014). Despite the current advances, there are still significant challenges to fully utilise the potential of firefighting drones.

One important aspect limiting the widespread deployment of firefighting drones is the need for human operators. The deployment of large numbers of drones is limited by the number of trained drone operators. Thus, automating fire-fighting drones will enable deployment of a large numbers of efficient drones, which due to the lack of human operators, have lower operational costs.

This report aims to advance the development of firefighting unmanned aerial vehicles (UAV's) by automating the water nozzle aiming. Currently, the accuracy of firefighting UAVs is limited by the operator's ability to aim the water stream. Whilst fighting fires, the optimal location to aim at is the fires base. Humans cannot see the base of a fire; therefore they must estimate its location. An autonomous drone that uses a thermal camera can highlight and target a fires thermal hotspot (base), removing human error and improving the efficiency with which a firefighting drone extinguishes a fire.

This project aims to deliver software that uses python and the computer vision library OpenCV (Pypi, 2023) to analyse thermal, depth and Red Green Blue (RGB) video footage to identify the location of a fire and the position of a water stream. Then, a piece of code which combines the fire and water stream detection codes was written, to output a pitch and yaw angle to the water nozzle, to accurately reposition the water stream onto the fire. The report evaluates the effectiveness of the combined code, highlights its limitations and suggests future work for improvements.

1.2 Aim

The aim of this project is to develop fire and water stream video recognition software to accurately aim and shoot a water stream at a fire.

1.3 Objectives

1. Develop vision-based fire recognition software in python using OpenCV which utilises a RealSense D435i depth sensor and a MLX90640 thermal camera to locate and track fires.
2. Test the fire recognition software from Objective 1 on videos of fires to establish its effectiveness.
3. Develop vision-based water stream detection software in python using OpenCV which utilises a RealSense D435i depth sensor and a MLX90640 thermal camera to detect a water stream.
4. Undertake an experiment to obtain thermal, RGB and depth videos of a water stream, then test the water stream detection software on these videos.
5. Combine and develop software from Objectives 1 and 4 to accurately aim the water stream at the base of a fire.
6. Test software developed in Objective 5 by using it to control the shooting of water stream at a fire.

1.4 Report Layout

The report is split into 7 chapters:

- Chapter 1 introduces the project, detailing its purpose and motivations.
- Chapter 2 contains the first coding challenge of the project, fire detection. The chapter covers the method used to detect fire and evaluates its success through testing on pre-recorded videos.
- Chapter 3 details the development of water stream detection code, the success of the code and its limitations.
- Chapter 4 develops code to move a target point onto the location of the water stream using a proportional integral derivative (PID) controller.
- Chapter 5 details how the code must be modified to effectively aim the water stream at the base of a fire.
- Chapter 6 combines the code from all previous chapters to move a water stream onto a fire in real time and displays the final test's results.
- Chapter 7 concludes the project, highlights its limitations and suggests scopes for further work.

Chapter 2: Fire Detection

2.1 Introduction

Traditional fire detection methods, such as heat and smoke detectors, have been widely used successfully for decades. These devices are limited in their application and accuracy, as they can only cover limited enclosed areas, offer delayed detection, produce numerous false alarms and have limited sensing capabilities (AZoSensors, 2012). As technology has advanced, the implementation of alternative fire detection techniques such as video detection have been explored.

Many of the currently available VFD algorithms prioritize the detection of both smoke and flames, in contrast to earlier papers that focused solely on the detection of flames (Phillips et al., 2001). Smoke tends to spread faster than flames, enabling earlier detection of fires. It is an excellent indicator that a fire is occurring however it does not offer any meaningful location data, key information to extinguish a fire.

A perfect VFD algorithm is yet to be created due to the large variability of light, shapes, movement and colour of different detection environments. Present flame detection algorithms utilise flames unique characteristic such as their colour, shape, dynamic texture and flickering to identify them (Gaur et al., 2020). Cetin et al (2013) claimed the application of both RGB and infrared (IR) cameras improves the accuracy of VFD algorithms, thus this project uses both a RGB and thermal camera.

Many contemporary VFD algorithm's that only use a RGB camera harness deep learning algorithms such as convolutional neural networks, a type of neural network designed for image recognition that can automatically learn and identify features within an image (Saponara et al., 2020) (Frizzi et al., 2016). These techniques are beyond the scope of this project and were not explored.

This project does not attempt to advance the state of fire detection techniques, it seeks to develop an algorithm to reposition a water stream onto the base of a fire. To create successful aiming code, working VFD code is required. This chapter presents a VFD algorithm, that utilises a fires colour and heat properties to identify fires.

2.2 Methodology

Two key, widely recognised characteristics of fires are their heat and orange-based colour. The VFD algorithm developed uses these characteristics to identify fires, highlighting the location of thermal hotspots from the thermal camera and the location

of fire-coloured objects within the RGB camera. If an object was both orange and hot it was assumed to be a fire.

This project deals with the aiming of a water stream at a fire for a firefighting UAV, it assumes the drone has already found the broad location of a fire and is positioned such that the fire could be extinguished. Considering a fire is within the frame, the hottest orange coloured object is likely a fire.

To test the accuracy of the VFD algorithm, Dr Kaddouh provided 4 raw RGB and thermal videos from previously conducted firefighting UAV tests (Kaddouh, 2022). These videos were used to develop and test the VFD algorithm. The flowchart in Figure 2.1 shows how the VFD algorithm works:

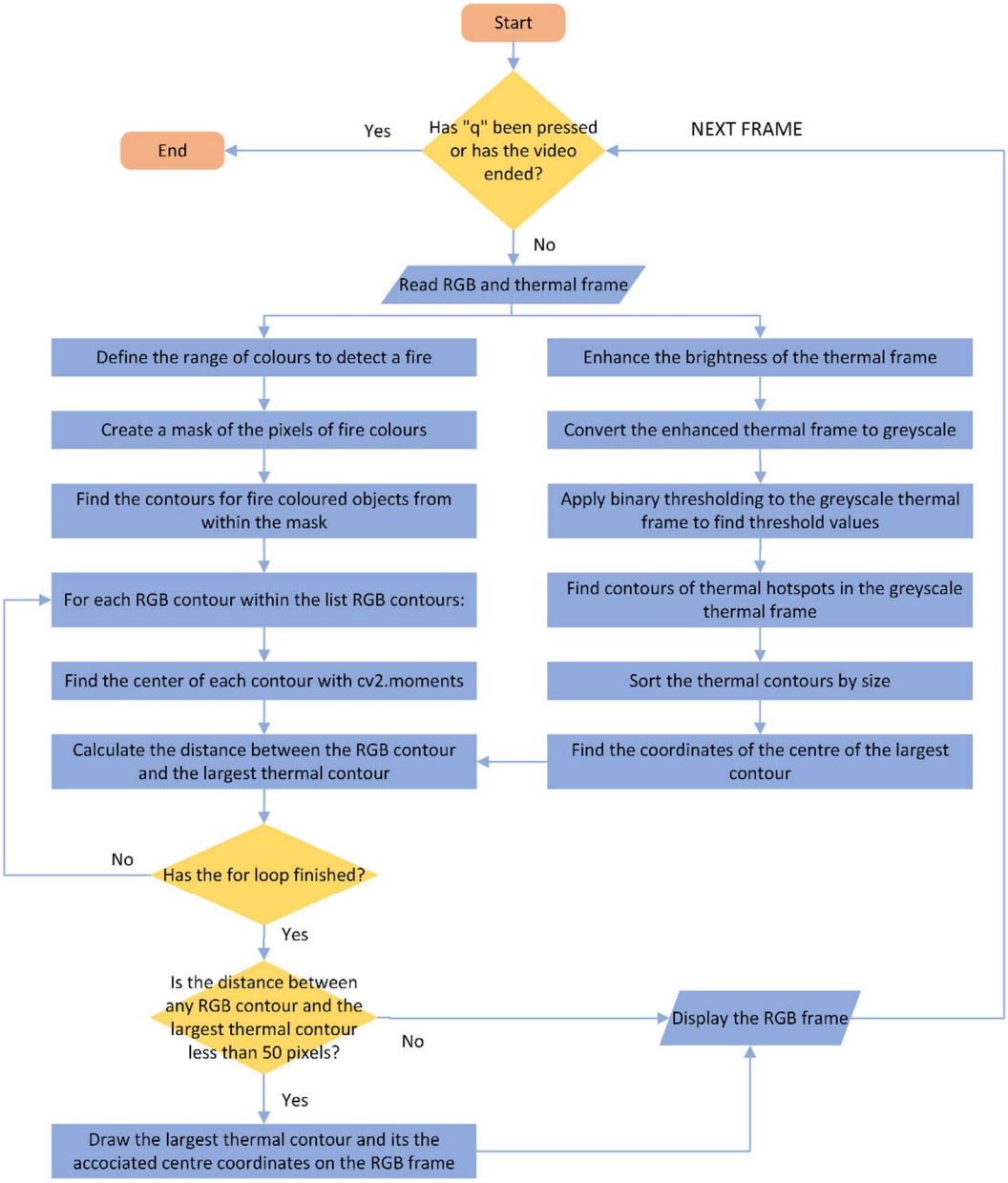


Figure 2.1: Flowchart of VFD algorithm

The thermal frame was converted from the RGB colour space into the LAB colour space to adjust the frames brightness. A brighter frame was used to create more contrast within the greyscale image and enable easier contour extraction. Figure 2.2 and Figure 2.3 show how an increase in the brightness allowed easier detection of thermal contours, Figure 2.3 contains a more visible thermal hotspot.

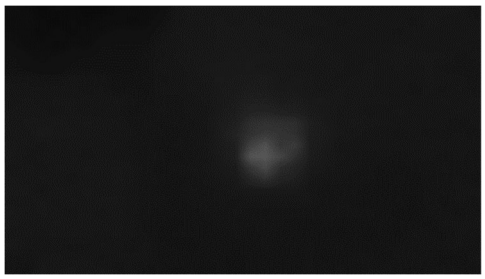


Figure 2.2: Raw thermal frame

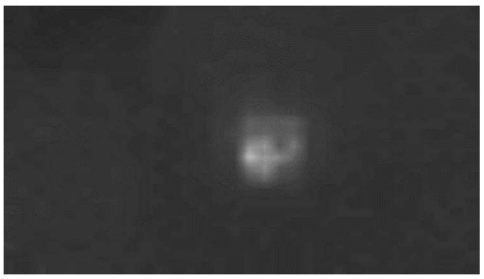


Figure 2.3: Processed thermal frame

The location of the fire was chosen to be at the centre of the largest thermal contour, rather than the centre of RGB contours, since the optimal location to shoot when extinguishing a fire is the fires hotspot (base), not the fires flames. Guidance for using a fire extinguisher as indicated by Edwards (2022) supports this assumption.

The location of the flames (RGB contours) and the base of the fire (thermal contour) did not align, therefore if the RGB contour was sufficiently close to the thermal contour, the thermal contour was identified as a fire. This distance value and other threshold values was chosen through systematic trial and error by observing the values that resulted in the best fire detection.

2.3 Results

Table 2.1: Success of VFD algorithm

Video	Total Frames	Fire Frames	Detected Fire Frames	False Positives	Accuracy (%)
1	359	0	2	2	0
2	643	320	235	2	72.8
3	933	322	260	3	79.8
4	1309	673	500	2	74.0

Table 2.1 shows the results of the VFD algorithm. The detected fire frames are the total number of frames the code detected fire. The fire frames are the total number of frames that actually contained a fire. False positives are the number of frames where the code detected a fire despite none being present. Accuracy is defined as:

$$Accuracy\ (\%) = \frac{Detected\ Fire\ Frames - False\ Positives}{Fire\ Frames} * 100 \qquad (1)$$

The total frames were calculated by adding a frame count to the code, after each frame, the frame count increased by one. The detected fire frames count worked in a similar way, but it only increased if the algorithm detected a fire. Human judgement was used to determine the true number of fire frames; it may be inaccurate.

Video 1 was disregarded as there was no fire within the video. The mean accuracy of videos 2, 3 & 4, was 75.5%. The thermal fire detection is shown in Figure 2.4 and the RGB fire detection is shown in Figure 2.5:

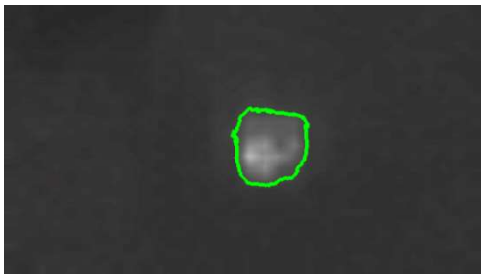


Figure 2.4: Thermal frame fire detection

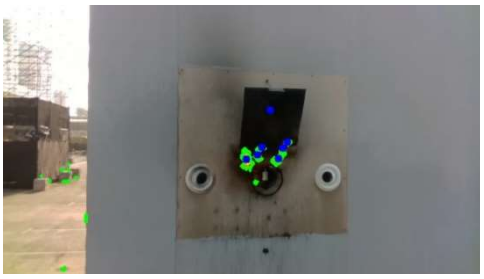


Figure 2.5: RGB frame fire detection

The final resultant fire detection is shown in Figure 2.6, the red dot represent the centre of the fire contour and the location of the fire.

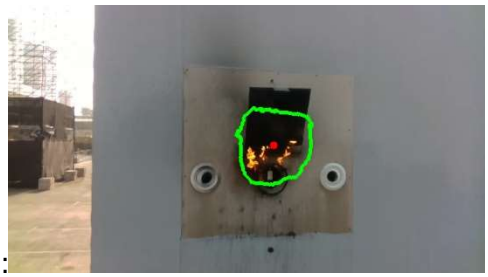


Figure 2.6: Fire detection

2.4 Discussion

Compared to contemporary VFD algorithms, the resultant success rate of 75.5% was relatively low. Çetin et al. (2013) evaluated 3 RGB detection methods across a wide range of different videos, he found an average success rate ranging from 78.4% to 87% accuracy. These methods utilised smoke detection and did not have access to a thermal camera so a direct comparison cannot be made. The 3 methods Cetin et al. evaluated would likely have a higher detection rate if paired with a thermal camera.

The primary explanation for the algorithm's failure to detect all incidents of fire is due to limitations of the thermal camera. When far from the fire, the thermal camera failed to highlight the thermal hotspots. Subsequently, a fire was not highlighted as the algorithm needs both a positive thermal and RGB detection to identify a fire. All actual fire frames were counted, regardless of distance, explaining the accuracy of 75.5%.

This project assumes the firefighting drone is positioned in a suitable location to extinguish the fire, for significant segments in the video footage provided, this was not true. When the drone is in position to extinguish the fire (close to the fire), the accuracy of the developed VFD algorithm substantially increases.

The RGB fire detection code could not be used exclusively as a fire detector because it only highlighted fire-coloured objects. Figure 2.5 depicts when the RGB fire detection code identified a collection of false positives, on the ground to the left of the building some non-fire, fire-coloured objects were incorrectly highlighted as fires.

The thermal fire detection could not be used exclusively as a fire detector because some hot objects are not fires. Occasionally a seemingly random thermal contour would appear within the test videos, causing some of the false positives in Table 2.1. For this reason, a verification system was implemented where a fire was only detected if both the thermal and RGB cameras both detected a fire.

The false positives detailed in Table 2.1 occurred when the coordinates of false positives RGB fire contours were within 50 pixels of the largest thermal contour. When the distance checker was reduced lower than 50 pixels, the number of false positives reduced, however the fire detection accuracy also decreased. A distance value of 50 pixels was found to be the optimal distance.

The VFD algorithm that was derived for this project has limited functionality, any hot fire-coloured object would be highlighted as a fire, if a fire-coloured object was close to a hot object, then the hot object would be highlighted as a fire. Fires that are not orange, such as a gas fire, will not be identified. To ensure accurate fire detection thermal camera must be relatively close to the fire.

The algorithm has undergone limited testing due to the projects time constraints; it may perform better in different conditions.

2.5 Conclusions

- The VFD algorithm produced resulted in a total accuracy of 75.5%.
- The code has limited functionality, the thermal camera must be close to the fire and a fire must already be within frame for the best results.
- The assumption that all hot orange objects are fires is not always correct and will sometimes result in false positives.

Chapter 3: Water Stream Detection

3.1 Introduction

To reposition a water stream onto a fire, the location of the water stream must be known. Chapter 3 shows the development of software to detect a water stream. Extensive research suggests very little work has been done in this area, only one research paper was found that used a water stream detection algorithm (Wu, 2016). Wu successfully detected a water stream using colour recognition (Wu, 2016), however Wu required very specific lighting, a still frame and the fire to be positioned on the ground.

This project concerns a firefighting drone, which will move and operate in a multitude of different conditions, utilising a colour recognition method similar to WU is unsuitable for the diverse range of operating settings.

Initially, experiments were conducted to collect footage from a thermal, RGB and depth camera to analyse and determine the best detection method. Several different water stream detection methods such as background subtraction, colour detection and moving object detection were tested on all the available cameras. Chapter 3 explains which water stream detection method was chosen and why, how the water stream detection code works, evaluates its effectiveness and highlights its limitations.

3.2 Methodology

The main premise behind the developed water stream detection algorithm is the utilisation of the OpenCV function cv2.absdiff, which displays the differences between two images as white and the similarities as black.

A still image taken just before the water stream starts is compared to a frame of the live water stream feed, this technique only required the use of the RGB camera.

The flowchart in Figure 3.1 shows how the water stream detection code works:

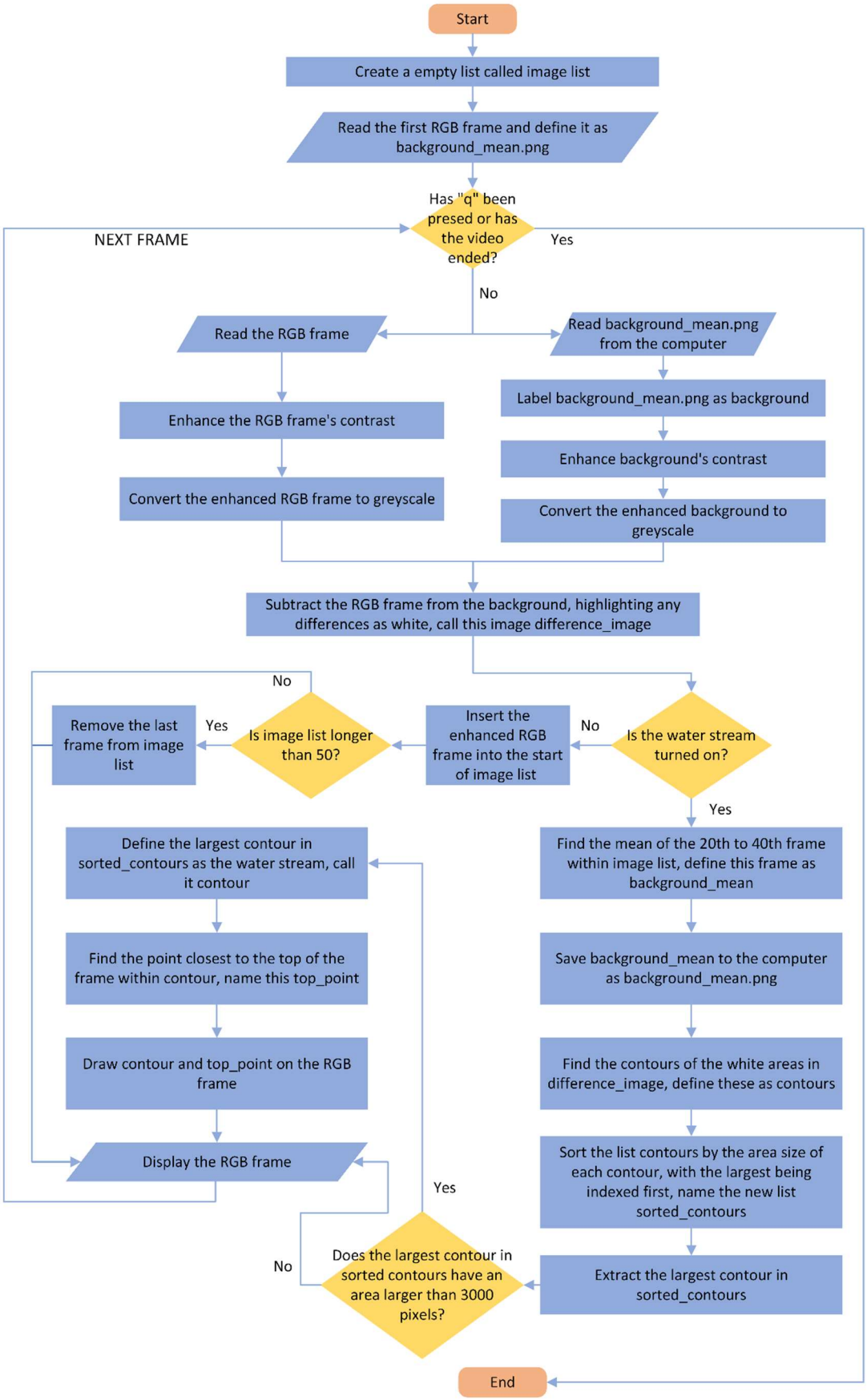


Figure 3.1: Flowchart of water stream detection algorithm

Locating the water stream with the thermal camera is not possible because difference in temperature between the water stream and the background is not enough to be detected by the MLX90640 thermal camera.

The depth camera could not be used to locate the water stream because the location of the water stream on the depth camera does not align with the location of the water stream on the RGB camera. The water stream appears as black on the depth stream colour map with a depth of 0 m as a parabola. A comparison of a RGB and a depth frame can be seen in Figure 3.2, Figure 3.3 and Figure 3.4:

Figure 3.2: RGB frame 80

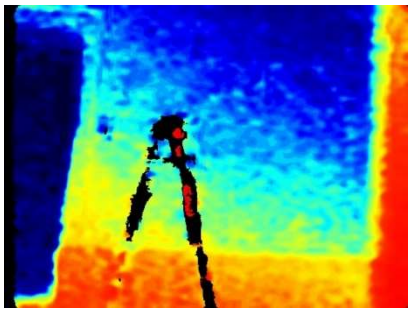


Figure 3.3: Depth frame 80

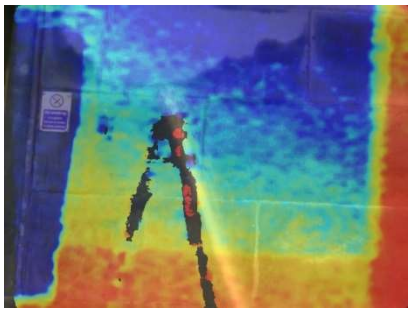


Figure 3.4: Comparison of RGB and depth water stream location

The water streams depiction as a black parabola is an error by the RealSense D435i depth sensor. The depth sensor works similarly to human eyes depth perception, using two cameras that are a known distance apart, by comparing their output, depth is calculated (IntelRealSense, 2019). The water stream appears directly in front of the two depth cameras; thus the two different cameras display vastly different images. The difference between the 2 camera frames is so vast that the depth sensor is unable to calculate the distance to the water stream, thus it is displayed as at 0 m as its distance is undefined. The same water stream is displayed in two separate locations, producing the parabola seen in Figure 3.3.

The RGB camera was used to detect the water stream as the using the thermal and the depth cameras was not possible. Colour detection is an inadequate technique as the water stream changes colour with changes in the background and lighting.

The function OpenCV function cv2.createBackgroundSubtractorMOG2() was explored, cv2.createBackgroundSubtractorMOG2() creates a static background from which it subtracts the current frame, leaving the moving objects (GeeksforGeeks, 2020). The result of testing on the water stream footage are shown in Figure 3.5:

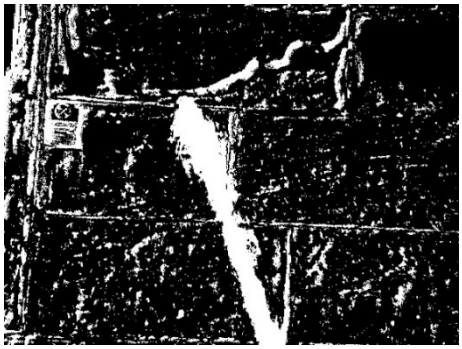


Figure 3.5: Water stream detection using cv2.createBackgroundSubtractorMOG2()

The water stream is detected in Figure 3.5, however there is also extensive noise caused by movements of the camera, shown as other white areas. cv2.BackgroundSubtractorMOG2 was not used in the water stream detection algorithm as there are insufficient input variables. Whilst the History (number of frames used to initialise the background) and VarThreshold (determines whether a frame belongs to the foreground or background) can be controlled, the configuration of the water nozzle (whether it is switched on or off), cannot be inputted.

The OpenCV function cv2.absdiff was chosen as the optimal technique to detect the water stream as it works very similarly to cv2.BackgroundSubtractorMOG2 but allows more input variables, such as the configuration of the water nozzle.

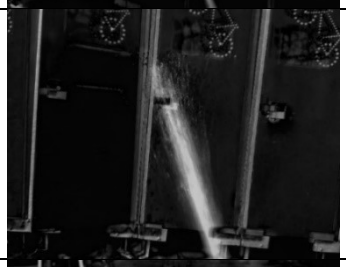
The variables, enhancement amount, contour area size and threshold values were chosen through systematic trial and error to provide the best water stream detection.

The length of the image list was limited to 50, to limit the storage required to run the water stream detection algorithm. The 20^{th} to 40^{th} frame within image list were averaged and set as the background image to blur the movement of the camera, reducing its associated error and enabling better detection of the water stream.

3.3 Results

A total of 8 different cases were analysed, the results are displayed in Table 3.1, the difference image displays the image which was used to find the water stream (labelled as difference_image in the flowchart). The RGB image displays what the water stream detection algorithm highlighted as the water stream.

Table 3.1: Water stream detection results

Case	RGB Image	Difference Image	Detected
1			Yes
2			Yes
3			No
4			No
5			Yes
6			Yes
7			Yes

8						Yes

The water stream detection software had a detection success rate of 75%, 6 out of the 8 cases highlighted the water stream correctly. The accuracy with which the code detected the water stream varied with each case.

3.4 Discussion

The 75% success rate does not portray the accuracy of the water stream detection algorithm. The accuracy of each cases water stream detection couldn't be quantified; whilst many cases detected the water stream very accurately, some cases included noise (false positives) as part of the detected water stream. False positives were included as part of the water stream due to movement of the camera relative to the background. The ideal conditions for this code require no movement of the camera or the background, thus the only difference between the background and RGB image would be the water stream. Unfortunately, movement occurred from both the camera and the background, due to the camera being carried and wind shaking trees.

In an attempt to mitigate any undesired movement, the largest contour within the difference image was considered the water stream. Any small areas of difference caused by the cameras moving were ignored. For the majority of cases this assumption allowed the correct identification of the water stream, however when camera movement caused areas of difference larger than the area of the water stream, the water stream was incorrectly identified. Slight movement of a bright object caused large white spots within the difference image, as anything touching them was considered to have moved and was highlighted as a difference. The result of cases 3 and 4 (Table 3.1) exemplify this, the sky is highlighted as a water stream.

Another reason cases 3 and 4 incorrectly highlighted the water stream is because the water stream was very difficult to see. The angle of the camera and the lighting conditions meant the water stream looked very similar to the background; thus it wasn't highlighted as a significant difference in the difference image.

The code correctly identified the water stream in cases 5, 6, and 8; however, it also identified additional background objects as part of the water stream. This occurred because small white areas appeared in the difference image due to camera

movement, and when these areas were adjacent to the water stream, they were highlighted as part of the water stream. The OpenCV function cv2.findContours counts a contour within the difference image as all continuous points along the boundary of an object of the same colour. Since all objects that change within the difference image are white, the water stream and any background changes adjacent to it are counted as the same contour. Consequently, some non-water stream objects were wrongly identified as part of the water stream.

The code does not work if an object (that isn't the water stream) moves into or within the frame after the background image has been established, as these objects are labelled as differences. If these differences have a larger area than the water stream, they will be incorrectly labelled as the water stream.

When the water stream is difficult to differentiate from the background such as at night and for low light conditions, the code will not work because the water stream cannot be seen with the RGB camera. It will not be highlighted as a difference; thus it will not be detected as the water stream.

3.5 Conclusions

- The water stream detection algorithm correctly identified a water stream in 75% of the test cases.
- The accuracy the water stream detection algorithm detected the water stream varied for the different cases. The highest accuracy was achieved when the camera was kept still, and the waters stream was very visible on the RGB camera. For further experiments, a conscious effort was made to ensure the camera remained as still as possible.
- The developed water stream detection software only works if the water stream can be seen by the RGB camera, in low light conditions the software does not work.
- If there is excess movement within the cameras frame or by the camera itself, such as if it was raining, the code will be unable to detect the water stream.
- Only a RGB camera is needed to detect a water stream

Chapter 4: Addition of a PID Controller

4.1 Introduction

A PID controller is a feedback control system that regulates a process or system by continuously adjusting the control signal based on the error between the desired setpoint and the actual process variable using three components: Proportional, Integral, and Derivative (Panda, 2012).

Having established the location of the water stream and a fire, a PID controller can be used to move the water stream onto the fire. Before conducting physical testing on a live system, testing was carried out on pre-recorded videos to verify the effectiveness of the PID controller in achieving the desired control outcomes. In recordings, it is impossible to move the location of the water stream, therefore code was written to move a target point onto the location of the water stream. Before conducting tests, the PID controller was calibrated, and successful code was established. Having conducted successful tests, the location of the water stream and the target point can be swapped, so that the water stream moves to the target point for real life tests.

4.2 Methodology

Rather than coding a PID controller from scratch, a simple PID controller package was imported from the Python Package Index (Pypi) (Pypi, 2021).

The simple PID controller from Pypi operates such that the PID controller is defined with the following line of code:

```
pid_x = PID(Kp=1.1, Ki=0, Kd=0.005, setpoint=target_x)
```

The setpoint is the target point which the PID controller will move the x coordinate towards. Kp, Ki and Kd define the Proportional, Integral and Derivative gains. The PID controller is applied using the code:

```
pid_output_x = pid_x(x)
x += pid_output_x
```

The value of x was updated and moved closer to target_x with each new frame. The PID controller code utilised two separate PID controllers, for the x and y directions.

The flowchart in Figure 4.1 depicts the addition of the PID controller code to the water stream detection code. The dotted box represents where additions were made to the water stream detection code. The flowchart is presented in this way to prevent the repetition of the water stream detection code flowchart.

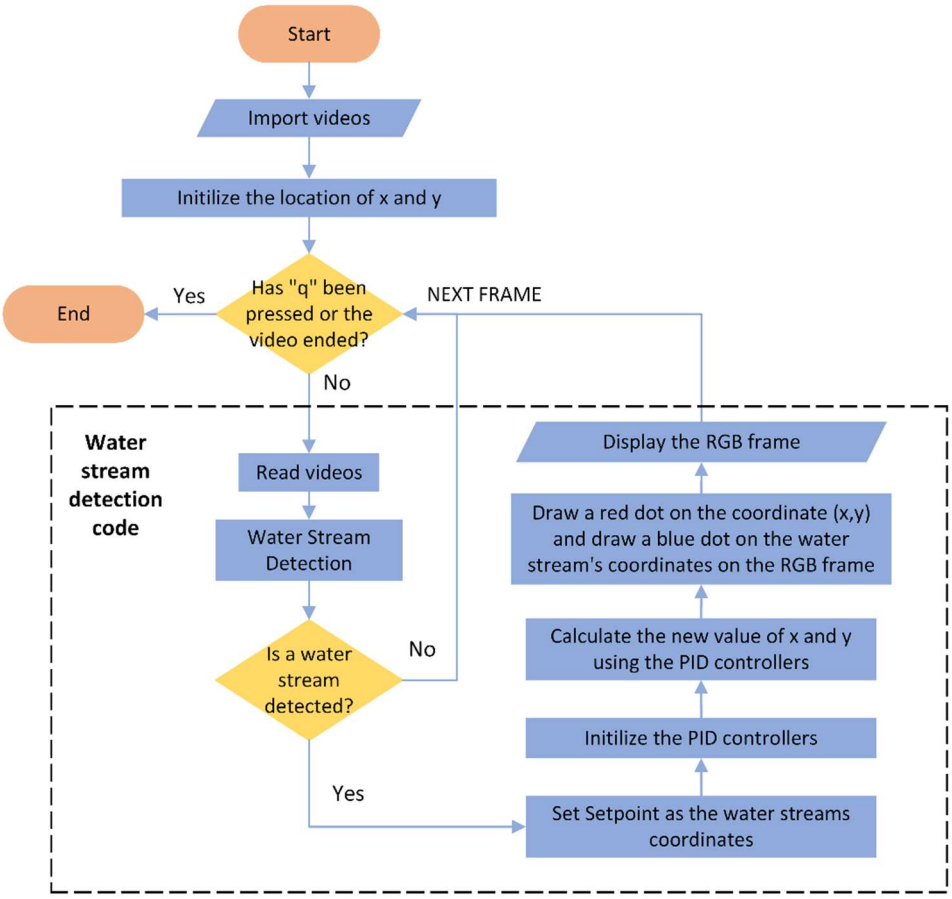


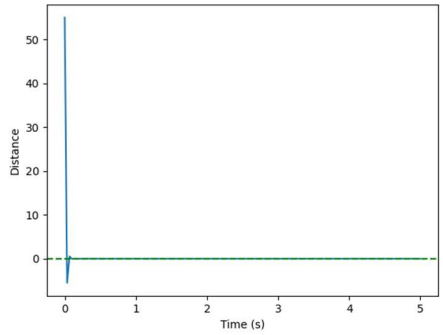
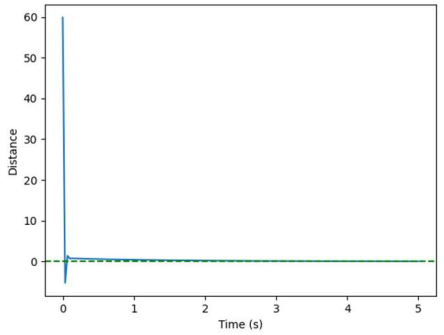
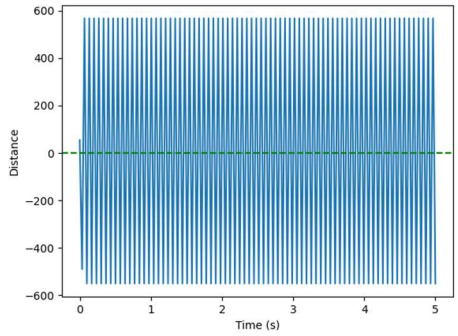
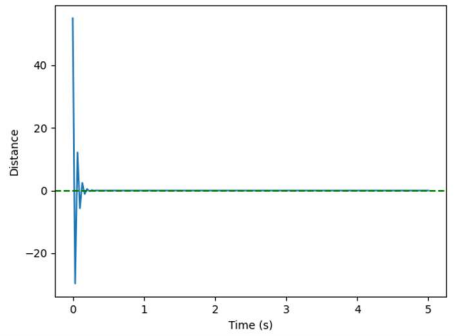
Figure 4.1: Flowchart of PID controller algorithm

4.3 Calibration

A PID controller has 3 variables, the proportional gain (Kp), the integral gain (Ki) and the derivative gain (Kd). Changing these variables resulted in different convergent times, oscillations and overshoots. Mashor (2018) found using a manual tuning method optimizes a PID controller better than using auto-tuning methods, therefore a manual tuning method was used in to calibrate the PID controller in this project.

Table 4.1 shows how the PID controller was calibrated. Initially the Ki and Kd were set to 0 and the Kp was increased until the system output oscillations of consistent amplitude and period. This occurred when Kp was 2. Then the Kp was reduced until the system converged, shown in case 1. Case 2 demonstrates the affect of a increase in the Ki, the steady state error reduced but the amount of overshoot increased. The increase in overshoot shown in case 2 did not justify the reduction in steady state error, therefore Ki was set to 0. Kd improves a systems response to changes in the setpoint. Initially Kd was set to 0.1, as seen in case 3. Case 3 resulted in the system no longer converging, so smaller Kd of 0.005 was tested as seen in case 4. Case 4 provided an optimal response; within 0.3 seconds the system settled.

Table 4.1: Calibration of the PID controller

Case 1	Case 2
	
Kp = 1.1, Ki = 0, Kd = 0	Kp = 1.1, Ki = 0.55, Kd = 0
Case 3	**Case 4**
	
Kp = 1.1, Ki = 0, Kd = 0.1	Kp = 1.1, Ki = 0, Kd = 0.005

4.5 Results

Table 4.2: PID controller results

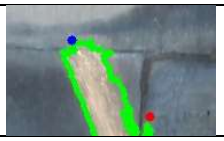
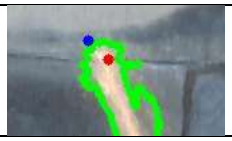
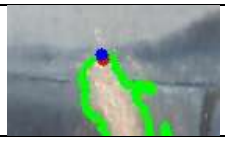
Frame	81	82	83	84
Photo				

Table 4.2 shows the results using the optimal PID controller setup from case 4 in the code. Within 4 frames the target point coincides with the water streams location.

4.6 Discussion

The code moved a target point onto the tip of the water stream. The PID controller was setup for the best response. For further experiments, the target point, and water stream coordinates will swap so that the water stream moves towards a target point.

4.7 Conclusion

- The optimal setup of the PID controller was Kp = 1.1, Ki = 0 and Kd = 0.005.
- PID controller code was successfully added to the water stream detection code to a target point onto the location of the water stream

Chapter 5: Aiming

5.1 Introduction

Chapters 2-4 established VFD, water stream detection and PID controller code. Chapter 5 seeks to calibrate these codes to work in real time.

Before this chapter, pixel coordinates of the water streams location are known, however these coordinates do not relate to the water stream's actual position. The location of the water stream within the camera frame needs to be calibrated to equal the actual location of the water stream.

5.2 Methodology

The water stream frame coordinates do not indicate where the water stream lands, rather they represent the peak of the water stream (shown as a star). Moving the water stream's pixel coordinates, to the fire's pixel coordinates would not work as the water stream would falls short of the fire. Figure 5.1 depicts the result of aligning the water stream and fire coordinates.

Figure 5.1: Water stream coordinates coinciding with the fire coordinates

The water stream coordinates need to be adjusted to represent where the water stream point of impact. Correcting the offset requires the angle of the water nozzle, the drone location (height of drone) and the target location (if the water is hitting a wall or the ground) to be known. Determining the correction required for every possible arrangement is not achievable within the timeframe of this project, therefore the project focused on one critical scenario.

Considering the safety implications associated with placing a fire on a wall, it was deemed more practical to develop code that solely addressed extinguishing fires located on the ground. To remove the effect of changing drone height, the drone was fixed at a height of 0.8 m. Both the water pressure and pitch angle can control the distance covered by the water stream, to reduce the number of changing variables the water pressure was kept constant.

The aiming system can be split into 2 sections, the yaw and pitch angle:

5.2.1 Pitch Angle

To determine the required input pitch angle, tests were conducted to find the relationship between the pitch angle and the distance to where the water impacted. The pitch of the water nozzle can operate from a minimum angle of -30° to a maximum of 70°. For every 10°, the input pitch angle was changed and the distance to where the water stream landed was recorded. An equation to link these variables was found.

The distance to the fire can be found from the depth camera. If this distance and the relationship between the pitch angle and distance to the water landing location are known, the pitch angle required to land the water stream on a fire can be found.

5.2.2 Yaw Angle

To relate the pixel coordinates to the input yaw angle, an experiment must be performed. The yaw of the water nozzle can operate from a minimum of -80° to maximum of 80°. If the water nozzle is set to 80°, the water nozzle will spray to the right of the frame, outside of the camera field of view. To calibrate the input yaw angle with the pixel water stream coordinates, the input yaw angles required to just enter each side of the camera frame are needed. Figure 5.2 illustrates the yaw angle calibration. Angles x and y equal the required input yaw angles to hit the left and right sides of the frame.

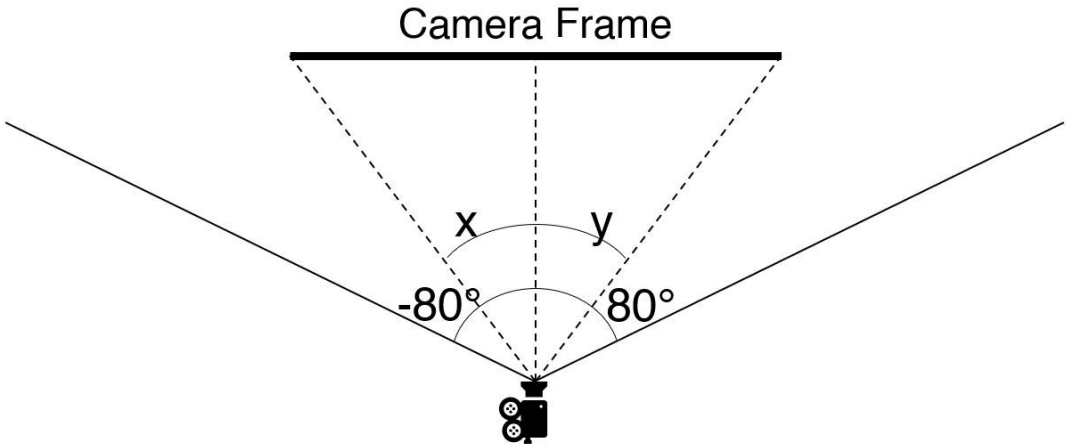


Figure 5.2: Angle required for water stream to enter frame

Once these two values are known, the water streams horizontal pixel coordinate can be converted into a yaw angle through Equation 2:

$$Input\ Yaw\ Angle = \frac{horizontal\ pixel\ coodinate * (|x| + |y|)}{640} + x \qquad (2)$$

Where 640 is the width of the frame in pixels.

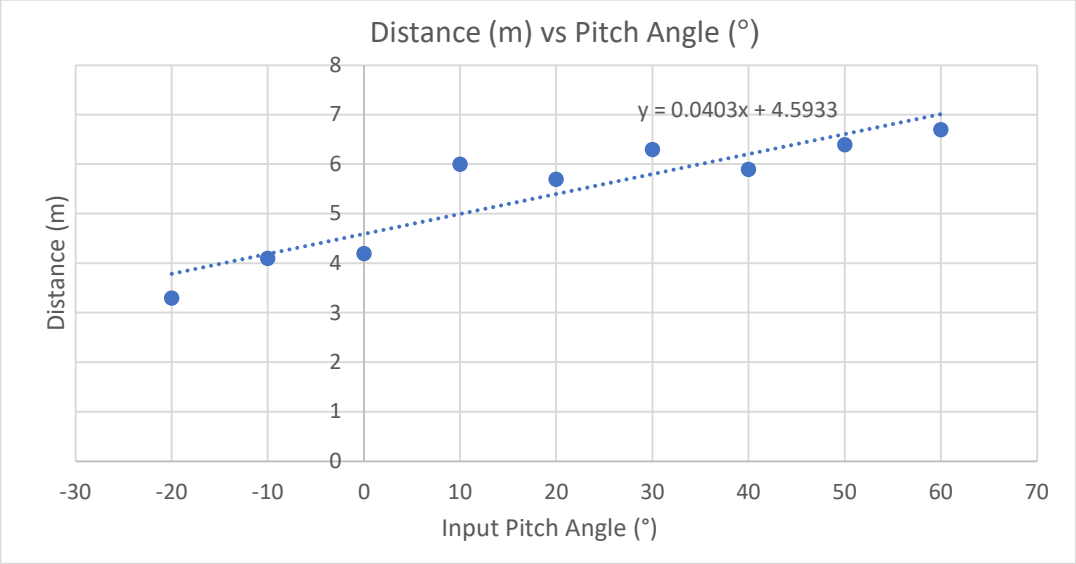
The input yaw angle required to just enter the water frame was hypothesised to change with changes in the input pitch angle, therefore the experiment conducted measured the yaw angle required for the water stream to just enter the frame for every 10° of the total range of input pitch angles.

5.3 Results

Experiment 1: Yaw angle limits

The yaw angle limits x and y were found to be -65° and 50°. The pitch angle did not affect the yaw angle limits.

Experiment 2: Input Pitch Angle vs distance to water stream landing location



Graph 5.1: Distance (m) to water stream against input pitch angle (°)

The relationship between the input pitch angle and the distance to where the water stream landed is shown in Graph 5.1. Equation 3 describes the line of best fit within Graph 5.1 between distance and the input pitch angle:

$$Distance = 0.403 * Input\ Pitch\ Angle + 4.5933 \quad (3)$$

The line of best fit had a coefficient of determination of 0.8314, suggesting there is a strong positive correlation between input pitch angle and distance.

5.4 Discussion

Equation 3 can be rearranged into Equation 4; so that the input pitch angle determines where the water stream lands:

$$Input\ Pitch\ Angle\ = \frac{Distance\ - 4.5933}{0.403} \quad (4)$$

Experiment 1 found x to be -65° and y to be 50°, inputting these values into Equation 2 creates Equation 5:

$$Input\ Yaw\ Angle = \frac{horizontal\ pixel\ coodinate * (115)}{640} - 65 \quad (5)$$

If the distance to a fire and the horizontal pixel coordinate of the water stream from the water stream detection code are known, the input yaw and pitch angles needed to hit the fire are known. The distance to the fire can be found using the coordinates of the detected fire from the VFD code and the depth camera.

Whilst recording results, it was observed that the water stream would land across a wide area approximately 1 m in length and 0.3 m in width. Judging the centre of this landing area was difficult, adding error to the results. Determining the exact angle at which the water stream entered the frame was not possible as small 1 degree changes in the input nozzle yaw angle resulted in no observable difference in water streams landing location.

Keeping the drone arm stationary throughout the experiment was difficult due to the weight of the water pouch, as the arm moved it changed the trajectory of the water stream, interfering with the results, causing error. As the water pouch ran out, the water pressure decreased. This added a new variable which affects the results, adding to the error.

The large landing area of the water stream counteracts the inaccuracy caused by the calibration errors. If the calibration of the water nozzle aiming is slightly incorrect the large landing area ensures the water stream would still land on the fire.

5.5 Conclusion

- Equation 4 and Equation 5 describe how to calculate the yaw and pitch output angles to aim the water nozzle at a fire. These equations were used in the final combined code to control the water nozzle.
- There was significant error in the results that determined Equation 4 and Equation 5 which make the equations less accurate. The large landing area of the water stream counteract these errors.

Chapter 6: Final test

6.1 Introduction

Chapter 6 combines all the previously developed code and presents how the code was altered to be used in real time.

Until Chapter 6, all previous code operated by analysing pre-recorded videos. Modifying the code to operate in real time presented significant challenges. The main issue was integrating the software and hardware. As this project is solely focused on developing the software aspect of the firefighting drone, assistance establishing cohesion with the hardware was provided by Dr Shival Dubey.

Having established code which ran in real time, an experiment was conducted to establish the effectiveness of the combined code. This chapter presents the combined real time code, shows the results of the experiment and discusses the codes success and failures.

6.2 Health and safety

The fire experiment was carried out in a private garden to remove the risk to the public and to limit disturbance caused by lighting a fire. The fire was lit on top of a barbecue to remove the danger of the fire spreading and to allow the fire to be extinguished and controlled by closing the lid. A fire extinguisher was on standby in case of the fire got out of control. All risks were considered and mitigated.

6.3 Methodology

The code utilises all previously developed code. Initially the VFD code is used to detect whether there is a fire and where the fire is. If a fire is detected, the water nozzle is toggled on. Once the water stream is turned on, the water stream detection code detects the coordinates of the water stream.

As presented in Chapter 5, the yaw angle of the water nozzle is controlled using a PID controller, the pitch angle of the water nozzle is controlled using the previously established relationship between pitch angle and distance. The distance to the fire is found using the depth camera.

The flowchart in Figure 6.1 shows how the developed code from earlier chapters was combined:

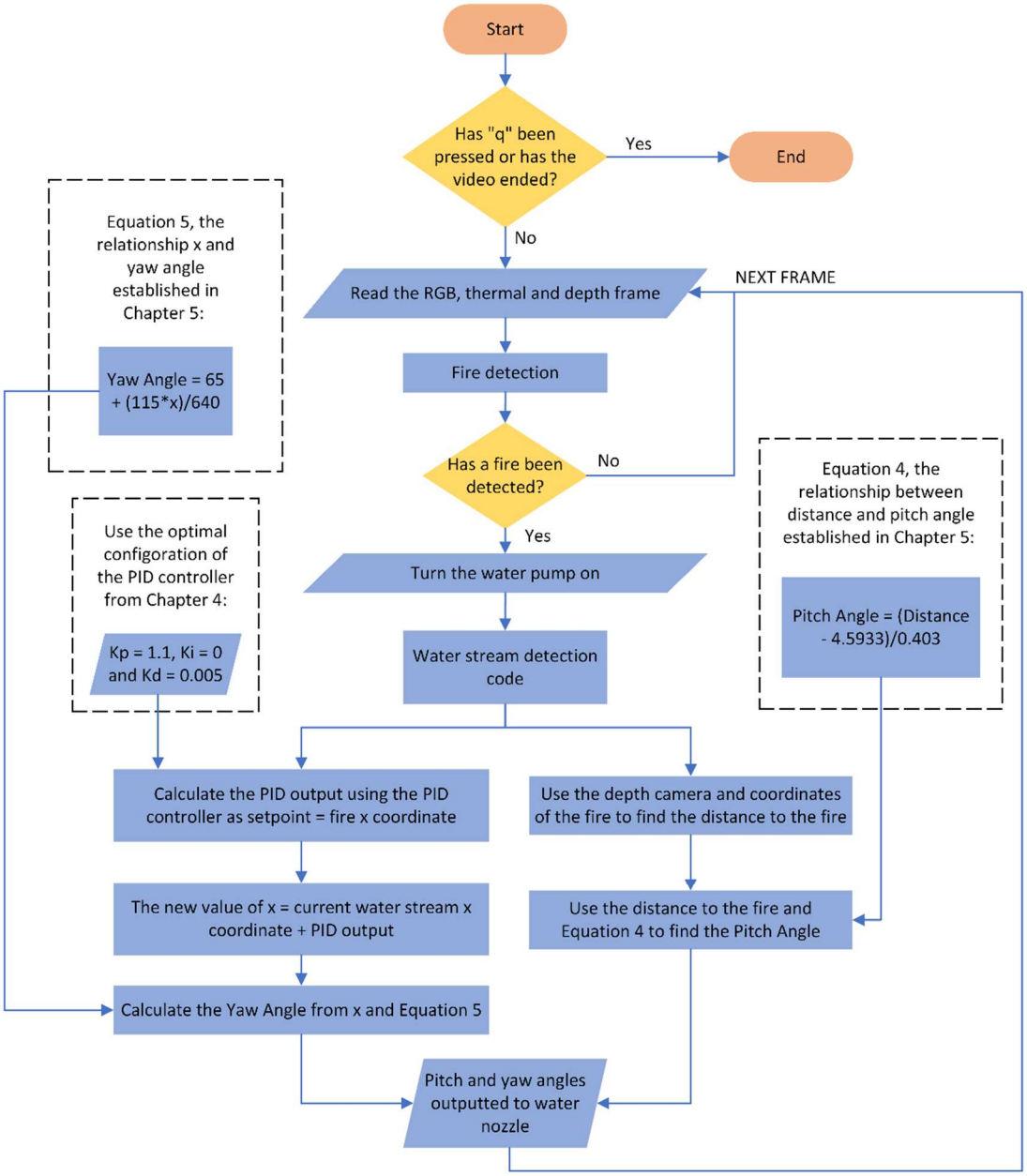


Figure 6.1: Flowchart of combined code

6.4 Results and Discussion

The software ran on a Raspberry Pi, taking various camera frames as inputs. After processing the video frames, the code outputted a yaw and pitch angle to the water nozzle, as well as the configuration of the water pump.

To observe the result of the combined code, a monitor was attached to the Raspberry Pi, to display the output of the RGB, thermal and result video feeds. Figure 6.2 shows the output of the RGB video feed, the fire and other fire-coloured objects were highlighted as part of the RGB fire detection code. Figure 6.3 shows the output of the thermal video feed, the fire was successfully highlighted as a thermal contour.

Figure 6.2: RGB fire detection

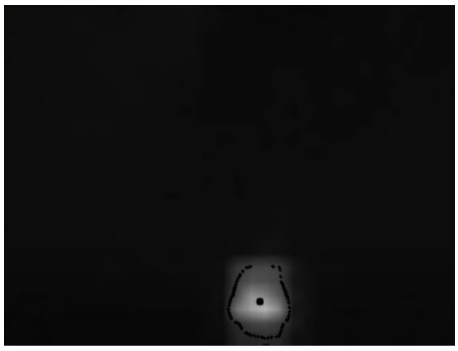


Figure 6.3: Thermal fire detection

The thermal and RGB frames did not align due to a misalignment of the thermal and RGB cameras. The location of the identified fire was located to the right of the actual fire, as seen in Figure 6.4. After a short delay the result frame showed a fire was detected, as shown in Figure 6.4.

Figure 6.4: Result frame, showing the final fire detection

Figure 6.5: Result frame after the water stream started

As a fire was detected in the result frame, the water stream was toggled on. As soon as the water pump was toggled on and water entered the frame, the fire was no longer detected within the result frame, as seen in Figure 6.5.

The VFD code was very temperamental, despite a RGB contour always being within 50 pixels of the thermal contour (often due to a false positive), a resultant fire was often not detected. It is hypothesised that this occurred due to processing limitations of the Raspberry Pi. The Depth Camera D435i's RGB camera operates at 30 fps (IntelRealSense, 2019). Within 1/30th of a second the Raspberry Pi must individually check the distance between every RGB contour and the thermal contour.

If the VFD code is not processed quick enough, it would be analysing a previous frame instead of the live frame. This would account for the short delay between the detection of individual RGB and thermal fires and the final fire detection. The code requires a few seconds to calculate the distance between each RGB fire contour and the thermal contour. The first few RGB contours in the list may be farther than 50

pixels away from the thermal contour, despite other RGB contours being within range. If this occurs, the water pump will turn on after a few seconds of delay when it eventually checks a RGB contour that is within range.

In an attempt to resolve this issue, an orange bag was placed behind the fire (at a safe distance) and the distance value was increased to 300 pixels so that most RGB contours (even those that aren't the fire) would trigger the detection of a fire. This successfully toggled the pump on quicker. The success of this alteration supports the hypothesis that there is limited processing power.

Once the water pump is toggled on, the water stream detection part of the code was triggered. An intensive part of the code that challenges the processing power of the Raspberry Pi, as a background frame must be subtracted from a live frame. Once the water pump was toggled on, the water nozzle did not move. Either the water stream was not being detected or Raspberry Pi was having processing issues.

In case the lack of movement was due to the water stream not being detected, the background was changed. The new background is shown in Figure 6.6, this change did not result in detection of the water stream.

Figure 6.6: Change in background

It cannot be conclusively determined whether the code unsuccessfully detected the water stream or whether the computer was unable to process all the information. Due to the success of earlier code and the change in background having no effect, it is hypothesised the failure was due to the Raspberry Pi's processing limitations.

6.5 Conclusion

- The combined code successfully ran in live conditions, the RGB, thermal and depth camera footage were successfully read.
- The combined code analysed the code and turned the water pump on.
- The combined code was unable to move the water stream onto the fire.

Chapter 7: Conclusion

7.1 Achievements

- VFD code was written and tested on pre-recorded videos. It achieved an accuracy of 75.5%.
- Video footage was obtained of the firefighting drone firing a water stream against a range of different backgrounds.
- Water stream detection code was successfully written and tested on the obtained video footage, achieving a success rate of 75%.
- PID controller code was written and combined with the water stream detection code to move a target point onto the detected water stream.
- The PID controller code was calibrated to find the optimal PID gain values, these were Kp = 1.1, Ki = 0 and Kd = 0.005.
- The input yaw and pitch angles were related to the water stream detection and fire detection codes by conducting calibration experiments.
- A final combined code which moves the water stream onto a fire was written.
- A final experiment was conducted to test the effectiveness of the final combined code.
- The final combined code successfully ran and detected the fire, resulting in the water pump initiating the spraying of the water stream.

7.2 Discussion

The project achieved all of its objectives. A successful VFD code was written and tested, achieving objectives 1 and 2. The VFD code achieved an accuracy of 75.5% on a set of 4 recorded videos. When the drone was close to the fire, the fire was successfully detected more often.

Water stream detection code was written and tested on recorded videos, achieving a success rate of 75%, with varying degrees of accuracy. Therefore objectives 3 and 4 were fulfilled.

The VFD code and the water stream detection code were combined with values from an aiming calibration test and a PID controller to produce a final set of code which could interact with a water nozzle in real time. The combined code successfully ran and was tested on a real fire, achieving objectives 5 and 6. In the final test, the combined code struggled with hardware processing limitations and was unsuccessful in moving the water stream onto the fire.

Initially code was written using recorded videos, transitioning from working with recorded videos to a live camera feed for Chapter 6 was challenging and caused substantial delays.

7.3 Conclusion

The project was mostly successful. All the objectives were achieved, three successful codes were written and tested. This project provides a good start for further work into water stream detection and aiming be built upon. The main failure of the project was the inability to move the water stream onto a fire in the final test.

7.4 Future Work

- Further work needs to be done to test the success of the combined code. The computer used in Chapter 6 needs to be upgraded to a computer with a better processor so that the frames can be analysed and processed in real time.
- Aligning the RGB camera and the thermal camera would improve the accuracy of the combined code.
- The VFD code could be improved by implementing more complex contemporary VFD techniques such as deep learning algorithms.
- Water stream detection can be explored through the use of a more sensitive thermal camera, which could detect the water stream. This would enable water stream detection to work in low light conditions and at night, overcome the limitations of the presented water stream detection code.
- The water stream could be coloured to test whether this improves the success rate of the water stream detection code.
- To further develop the water stream detection code, the threshold values could change with each change in background. A Gaussian function could be applied to the difference image to determine the optimum threshold values for every different background.

Reference list

Akhloufi, M.A., Couturier, A. and Castro, N.A. 2021. Unmanned Aerial Vehicles for Wildland Fires: Sensing, Perception, Cooperation and Assistance. *Drones*. **5**(1), p.15.

AZoSensors 2012. An Overview of Smoke Detectors. *AZoSensors.com*. [Online]. [Accessed 3 May 2023]. Available from: https://www.azosensors.com/article.aspx?ArticleID=19.

Bullock, C. 2020. Drones used in Australia for wildlife search and rescue. *AirMed&Rescue*. [Online]. [Accessed 3 April 2023]. Available from: https://www.airmedandrescue.com/latest/news/drones-used-australia-wildlife-search-and-rescue.

Çetin, A.E., Dimitropoulos, K., Gouverneur, B., Grammalidis, N., Günay, O., Habiboğlu, Y.H., Töreyin, B.U. and Verstockt, S. 2013. Video fire detection – Review. *Digital Signal Processing*. **23**(6), pp.1827–1843.

Edwards, R. 2022. Be Prepared: How to Properly Use a Fire Extinguisher. *SafeWise*. [Online]. [Accessed 9 April 2023]. Available from: https://www.safewise.com/blog/prepared-properly-use-fire-extinguisher/.

Ehang 2014. EHang | Smart City Management - EHang 216F. *Ehang.com*. [Online]. [Accessed 3 April 2023]. Available from: https://www.ehang.com/ehang216f/.

Frizzi et al 2016. S. Frizzi, R. Kaabi, M. Bouchouicha, J. -M. Ginoux, E. Moreau and F. Fnaiech, "Convolutional neural network for video fire and smoke detection," IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, Italy, 2016, pp. 877-882, doi: 10.1109/IECON.2016.7793196.

Gaur, A., Singh, A., Kumar, A., Kumar, A. and Kapoor, K. 2020. Video Flame and Smoke Based Fire Detection Algorithms: A Literature Review. *Fire Technology*. **56**(5), pp.1943–1980.

GeeksforGeeks 2020. Background subtraction OpenCV. *GeeksforGeeks*. [Online]. [Accessed 1 May 2023]. Available from: https://www.geeksforgeeks.org/background-subtraction-opencv/.

Gonzalez, R.C. and Woods, R.E. 2018. *Digital image processing*. New York, Ny: Pearson.

IntelRealSense 2019a. Beginner's guide to depth (Updated). *Intel® RealSense™ Depth and Tracking Cameras*. [Online]. [Accessed 19 April 2023]. Available from: https://www.intelrealsense.com/beginners-guide-to-depth/.

IntelRealSense 2019b. Depth Camera D435i – Intel® RealSense™ Depth and Tracking Cameras. *Intel® RealSense™ Depth and Tracking Cameras*. [Online]. Available from: https://www.intelrealsense.com/depth-camera-d435i/.

IntelRealSense 2021. librealsense/read_bag_example.py at master · IntelRealSense/librealsense. *GitHub*. [Online]. [Accessed 11 April 2023]. Available from: https://github.com/IntelRealSense/librealsense/blob/master/wrappers/python/examples/read_bag_example.py.

Kaddouh, B. 2022. Raw RGB and Thermal Firefighting UAV Footage. Available from: https://leeds365-my.sharepoint.com/personal/eenbk_leeds_ac_uk/_layouts/15/onedrive.aspx?ga=1&id=%2Fpersonal%2Feenbk%5Fleeds%5Fac%5Fuk%2FDocuments%2FVerification%20Node%2FFirefighting%20UAV%20Docs%2FC3%2Dvideos%2Fchallenge3%5Frehearsal%5Frecording%5Fand%5Fanalysis&view=0.

London Fire Brigade 2023. Drones. *London-fire.gov.uk*. [Online]. [Accessed 3 April 2023]. Available from: https://www.london-fire.gov.uk/about-us/services-and-facilities/vehicles-and-equipment/drones/.

Mashor, Y., Mahdi, M. and Hani, H. 2018. Performance of Manual and Auto-Tuning PID Controller for Unstable Plant- Nano Satellite Attitude Control System Neural network-based analysis of fine needle aspirated cells View project Design procedure for an industrial wastewater treatment plant implying aerated lagoons View project The 6 th International Conference on Cyber and IT Service Management (CITSM 2018) Performance of Manual and Auto-Tuning PID Controller for Unstable Plant -Nano Satellite Attitude Control System.

Ministry of Housing, Communities & Local Government 2014. Firefighter fatalities. *GOV.UK*. [Online]. [Accessed 24 April 2023]. Available from: https://www.gov.uk/government/publications/firefighter-fatalities.

Office, H. 2023. Fire and rescue incident statistics: England, year ending September 2022. *GOV.UK*. [Online]. [Accessed 24 April 2023]. Available from: https://www.gov.uk/government/statistics/fire-and-rescue-incident-statistics-england-year-ending-september-2022/fire-and-rescue-incident-statistics-england-year-ending-september-2022.

Panda, R.C. 2012. *Introduction to PID controllers : theory, tuning and application to frontier areas / monograph.* Rijeka, Croatia: Intech.

Peter 2018. Firefighting Drones Aim to Fly Higher, Save More Lives. *Robotics Business Review*. [Online]. [Accessed 14 March 2023]. Available from: https://www.roboticsbusinessreview.com/unmanned/firefighting-drones-aim-to-fly-higher-save-lives/#:~:text=Currently%20eight%20types%20of%20drones%20are%20considered%20useful,DJI%20Matrice%20210%208%20FlyByCopters%20Thermal%20Surveying%20X8.

Phillips, W., Shah, M., Da, N. and Lobo, V. 2001. *Flame recognition in video*.

Pypi 2023. opencv-python. *PyPI*. [Online]. [Accessed 3 May 2023]. Available from: https://pypi.org/project/opencv-python/.

Pypi 2021. simple-pid. *PyPI*. [Online]. [Accessed 3 April 2023]. Available from: https://pypi.org/project/simple-pid/.

Saponara, S., Elhanashi, A. and Gagliardi, A. 2020. Real-time video fire/smoke detection based on CNN in antifire surveillance systems. *Journal of Real-Time Image Processing*. **18**(3), pp.889–900.

Wu, H.B., Li, Z.J., Ye, J.H., Ma, S.C., Li, J.W. and Yang, X.N. 2016. Firefighting robot with video full-closed loop control. *International Journal of Safety and Security Engineering*. **6**(2), pp.254–269.

Appendix 1: Fire Detection Code

```
#importing libraries
import cv2
import numpy as np

#reading the video file
rgb_video =
cv2.VideoCapture("C:\\Users\\finta\\OneDrive\\Documents\\Year
3\\Python\\Fire Videos Bibal\\rgb_raw_T3.avi")
thermal_video =
cv2.VideoCapture("C:\\Users\\finta\\OneDrive\\Documents\\Year
3\\Python\\Fire Videos Bibal\\thermal_raw_T3.avi")

# Get the video dimensions
frame_width = int(rgb_video.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(rgb_video.get(cv2.CAP_PROP_FRAME_HEIGHT))
frame_fire_count = 0
frame_count = 0


# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
output_video = cv2.VideoWriter('output_video.mp4', fourcc, 30.0,
(frame_width, frame_height))


#looping through the video frames
while True:
    ret, rgb_frame = rgb_video.read()
    ret, thermal_frame = thermal_video.read()

    # Make a copy of the rgb frame to draw contours on
    result = rgb_frame.copy()

    # Define the range of colours to detect
    lower_fire = np.array([0, 100, 100])
    upper_fire = np.array([50, 255, 255])

    hsv = cv2.cvtColor(rgb_frame, cv2.COLOR_BGR2HSV)

    #convert the thermal frame to lab to increase contrast
    thermal_lab = cv2.cvtColor(thermal_frame, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(thermal_lab)
    l = cv2.add(l, 30)
    clahe = cv2.createCLAHE(clipLimit=3, tileGridSize=(32,32))
    cl = clahe.apply(l)
    limg = cv2.merge((cl,a,b))
    enhanced_thermal_frame = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
```

```
    # Convert the image to grayscale then apply a mask to the image
    gray = cv2.cvtColor(enhanced_thermal_frame, cv2.COLOR_BGR2GRAY)

    # Apply binary thresholding to find the fire
    _, thresh = cv2.threshold(gray, 70, 255, cv2.THRESH_BINARY)

    # Find contours of the thermal fire location
    thermo_contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

    # Find the largest contour and only display that
    sorted_contours = sorted(thermo_contours, key=cv2.contourArea,
reverse=True)

    # Draw the contours on the thermal frame
    if len(sorted_contours) > 0 and cv2.contourArea(sorted_contours[0])
> 100:
        cv2.drawContours(enhanced_thermal_frame, sorted_contours[0], -1,
(0, 255, 0), 3)
        M = cv2.moments(sorted_contours[0])
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        # cv2.circle(thermal_frame, (cx,cy), 5, (0, 0, 255), -1)

    # Create a mask for the fire
    mask = cv2.inRange(hsv, lower_fire, upper_fire)

    #find the contour of the rgb fire location
    rgb_contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

    # Draw the contours on the rgb frame
    cv2.drawContours(rgb_frame, rgb_contours, -1, (0, 255, 0), 3)

    # if rgb_contours are within a 50 pixel radius of the
thermal_contours, show the thermal_contours
    # Calculates the centre of each rgb_contour
    for rgb_contour in rgb_contours:
        M = cv2.moments(rgb_contour)
        if M["m00"] != 0:
            tx = int(M['m10']/M['m00'])
            ty = int(M['m01']/M['m00'])
        else :
            tx = 0
            ty = 0

        distance = np.sqrt((cx - tx)**2 + (cy - ty)**2)
```

```
        if distance <= 50 and len(sorted_contours) >0:
            cv2.drawContours(result, sorted_contours[0], -1, (0, 255,
0), 3)
            cv2.circle(result, (cx,cy), 5, (0, 0, 255), -1)
            print(cx, cy)
            cv2.circle(result, (tx,ty), 5, (255, 0, 0), -1)
            frame_fire_count += 1
            break


    output_video.write(result)

    cv2.putText(result, "Frames: " + str(frame_count), (50, 300),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
    cv2.putText(result, "Fire frames: " + str(frame_fire_count), (50,
350), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

    cv2.imshow('Enhanced Thermal', enhanced_thermal_frame)
    cv2.imshow('Result', result)
    cv2.imshow('Thermal',thermal_frame)
    cv2.imshow('Rgb',rgb_frame)

    frame_count += 1

    output_video.write(result)

    #exiting with pressing 'q'
    if cv2.waitKey(24) & 0xFF == ord('q'):
        break

#releasing and destroying windows
rgb_video.release()
output_video.release()
cv2.destroyAllWindows()
```

Appendix 2: Water Stream Detection Code

```
# enhances difference image
# works in all frames but the sky one
import pyrealsense2 as rs
# Import Numpy for easy array manipulation
import numpy as np
# Import OpenCV for easy image rendering
import cv2
# Import argparse for command-line options
import argparse
# Import os.path for file path manipulation
import os.path

#Experiment 1: D:\\D Downloads\\20230112_142006.bag
#Experiment 2.1: D:\\D Downloads\\20230210_153108.bag
#Experiment 2.2: D:\\D Downloads\\20230210_154422.bag
#Experiment 2.3: D:\\D Downloads\\20230210_154606.bag #DOESN'T WORK
#Experiment 2.4: D:\\D Downloads\\20230210_154853.bag #DOESN'T WORK
#Experiment 2.5: D:\\D Downloads\\20230210_155144.bag

# Create object for parsing command-line options
parser = argparse.ArgumentParser(description="Read recorded bag file and
display depth stream in jet colormap.\
                                Remember to change the stream fps and
format to match the recorded.")
# Add argument which takes path to a bag file as an input
parser.add_argument("-i", "--input", type=str, default="D:\\D
Downloads\\20230112_142006.bag", help="Path to the bag file")
# Parse the command line arguments to an object
args = parser.parse_args()
# Safety if no parameter have been given
if not args.input:
    print("No input paramater have been given.")
    print("For help type --help")
    exit()
# Check if the given file have bag extension
if os.path.splitext(args.input)[1] != ".bag":
    print("The given file is not of correct file format.")
    print("Only .bag files are accepted")
    exit()
try:
    # Create pipeline
    pipeline = rs.pipeline()

    # Create a config object
    config = rs.config()
```

```
    # Tell config that we will use a recorded device from file to be
used by the pipeline through playback.
    rs.config.enable_device_from_file(config, args.input)

    # Configure the pipeline to stream the depth stream
    # Change this parameters according to the recorded bag file
resolution
    config.enable_stream(rs.stream.depth, rs.format.z16, 15)
    config.enable_stream(rs.stream.color, rs.format.rgb8, 15)

    # Start streaming from file
    pipeline.start(config)

    # Create opencv window to render depth image
    cv2.namedWindow("Depth Stream", cv2.WINDOW_AUTOSIZE)
    # Create opencv window to render rgb image
    cv2.namedWindow("RGB Stream", cv2.WINDOW_AUTOSIZE)

    # Create colorizer object
    colorizer = rs.colorizer()

    # Make a list of images
    image_list = []
    depth_list = []
    zero_list = [0,0]
    water_stream = False

    fgbg = cv2.createBackgroundSubtractorMOG2(detectShadows=False)

    #define intel6_frame.jpg as the background image
    cv2.imwrite("intel6_frame.jpg",
np.asanyarray(pipeline.wait_for_frames().get_color_frame().get_data()))

    # Streaming loop
    while True:
        # Get frameset of depth
        frames = pipeline.wait_for_frames()

        # Get depth frame
        depth_frame = frames.get_depth_frame()

        # Get rgb frame
        rgb_frame = frames.get_color_frame()

        # Convert rgb_frame to numpy array to render image in opencv
        rgb_image = np.asanyarray(rgb_frame.get_data())

        # Colorize depth frame to jet colormap
        depth_color_frame = colorizer.colorize(depth_frame)
```

```
        # Convert depth_frame to numpy array to render image in opencv
        depth_color_image = np.asanyarray(depth_color_frame.get_data())

        #height and width of the depth image
        height, width = depth_color_image.shape[:2]

        #convert the rgb frame to lab
        rgb_lab = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(rgb_lab)
        l = cv2.add(l, -100)
        l = np.clip(l, 0, 255)
        clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
        cl = clahe.apply(l)
        limg = cv2.merge((cl,a,b))
        enhanced_rgb_image = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)

        #Read background image
        background = cv2.imread("intel6_frame.jpg")

        # enhance background image
        background_lab = cv2.cvtColor(background, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(background_lab)
        l = cv2.add(l, -100)
        l = np.clip(l, 0, 255)
        clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
        cl = clahe.apply(l)
        limg = cv2.merge((cl,a,b))
        enhanced_background_image = cv2.cvtColor(limg,
cv2.COLOR_LAB2BGR)
        background_gray = cv2.cvtColor(background, cv2.COLOR_BGR2GRAY)

        gray = cv2.cvtColor(enhanced_rgb_image, cv2.COLOR_BGR2GRAY)

        # adjust brightness of water stream
        gray = cv2.add(gray, 0)
        gray = np.clip(gray, 0, 255)

        # adjust brightness of background
        background_gray = cv2.add(background_gray, -30)
        bavkground_gray = np.clip(background_gray, 0, 255)

        # add blur
        difference_image = cv2.GaussianBlur(background_gray, (5,5), 0)

        # Compute the absolute difference between the current frame and
the background
        difference_image = cv2.absdiff(background_gray, gray)
```

```
        # Enhance the difference image
        difference_image_bgr = cv2.cvtColor(difference_image,
cv2.COLOR_GRAY2BGR)
        difference_image_lab = cv2.cvtColor(difference_image_bgr,
cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(difference_image_lab)
        l = cv2.add(l, -30)
        l = np.clip(l, 0, 255)
        clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
        cl = clahe.apply(l)
        limg = cv2.merge((cl,a,b))
        enhanced_difference_image = cv2.cvtColor(limg,
cv2.COLOR_LAB2BGR)
        enhanced_difference_image =
cv2.cvtColor(enhanced_difference_image, cv2.COLOR_BGR2GRAY)

        # count number of pixels with a depth value of 0.0
        array = depth_frame.get_data()
        no_zeros = height*width - np.count_nonzero(array)
        # Append zero counts to list
        zero_list.append(no_zeros)
        if zero_list[-1] > 4000+zero_list[-2]:
            water_stream = True
        if zero_list[-1]+6000 < zero_list[-2]:
            water_stream = False
        print(water_stream)

        # water stream enters frame at any point from the bottom``
        bottom_row = [depth_frame.get_distance(i, height-2) for i in
range(0+20, width-20)]

        # detects water stream
        if (0.0 in bottom_row) and water_stream == True:
            print('Detecting water stream')
            background_mean = np.mean(image_list[10:20], axis=0)
            cv2.imwrite("intel6_frame.jpg", background_mean)
            # find contours
            ret, thresh = cv2.threshold(enhanced_difference_image, 60,
255, cv2.THRESH_BINARY) #best value needs to be found. 60 does not
detect all of floor image
            contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
            # if the contour area is greater than 3000, draw a circle on
the top point and draw the contour of the water stream
            for contour in contours:
                if cv2.contourArea(contour) > 3000:
                        y_coordinates = [point[0][1] for point in
contour]
```

```
                        top_point =
contour[y_coordinates.index(min(y_coordinates))][0]
                        cv2.drawContours(enhanced_rgb_image, [contour],
-1, (0, 255, 0), 3)
                        cv2.circle(enhanced_rgb_image, tuple(top_point),
1, (0, 0, 255), 5)
                        print(top_point)
        else:
            image_list.insert(0, enhanced_rgb_image)
            if len(image_list) > 50:
                image_list.pop()

        # Render image in opencv window
        # cv2.imshow("Water", water_stream)
        cv2.imshow('Enhanced difference image',
enhanced_difference_image)
        cv2.imshow("Depth Stream", depth_color_image)
        cv2.imshow("enhanced rgb image",enhanced_rgb_image)
        key = cv2.waitKey(1)
        # if pressed escape exit program
        if key == 27:
            cv2.destroyAllWindows()
            break

finally:
    pass
```

Appendix 3: PID Controller Code

```
# Moves target point to top of contour
# Detect only the largest contour
# works in all frames but the sky one
import pyrealsense2 as rs
# Import Numpy for easy array manipulation
import numpy as np
# Import OpenCV for easy image rendering
import cv2
# Import argparse for command-line options
import argparse
# Import os.path for file path manipulation
import os.path
# Import PID controller
from simple_pid import PID

#Experiment 1: D:\\D Downloads\\20230112_142006.bag
#Experiment 2.1: D:\\D Downloads\\20230210_153108.bag
#Experiment 2.2: D:\\D Downloads\\20230210_154422.bag
#Experiment 2.3: D:\\D Downloads\\20230210_154606.bag #DOESN'T WORK
#Experiment 2.4: D:\\D Downloads\\20230210_154853.bag #DOESN'T WORK
#Experiment 2.5: D:\\D Downloads\\20230210_155144.bag

# Create object for parsing command-line options
parser = argparse.ArgumentParser(description="Read recorded bag file and
display depth stream in jet colormap.\
                                Remember to change the stream fps and
format to match the recorded.")
# Add argument which takes path to a bag file as an input
parser.add_argument("-i", "--input", type=str, default="D:\\D
Downloads\\20230112_142006.bag", help="Path to the bag file")
# Parse the command line arguments to an object
args = parser.parse_args()
# Safety if no parameter have been given
if not args.input:
    print("No input paramater have been given.")
    print("For help type --help")
    exit()
# Check if the given file have bag extension
if os.path.splitext(args.input)[1] != ".bag":
    print("The given file is not of correct file format.")
    print("Only .bag files are accepted")
    exit()
try:
    # Create pipeline
    pipeline = rs.pipeline()

    # Create a config object
```

```
    config = rs.config()

    # Tell config that we will use a recorded device from file to be
used by the pipeline through playback.
    rs.config.enable_device_from_file(config, args.input)

    # Configure the pipeline to stream the depth stream
    # Change this parameters according to the recorded bag file
resolution
    config.enable_stream(rs.stream.depth, rs.format.z16, 15)
    config.enable_stream(rs.stream.color, rs.format.rgb8, 15)

    # Start streaming from file
    pipeline.start(config)

    # Create opencv window to render depth image
    cv2.namedWindow("Depth Stream", cv2.WINDOW_AUTOSIZE)
    # Create opencv window to render rgb image
    cv2.namedWindow("RGB Stream", cv2.WINDOW_AUTOSIZE)

    # Create colorizer object
    colorizer = rs.colorizer()

    # Make a list of images
    image_list = []
    depth_list = []
    zero_list = [0,0]
    water_stream = False

    fgbg = cv2.createBackgroundSubtractorMOG2(detectShadows=False)

    #define intel6_frame.jpg as the background image
    cv2.imwrite("intel6_frame.jpg",
np.asanyarray(pipeline.wait_for_frames().get_color_frame().get_data()))

    # define x and y - arbituary values in this save - SHOULD BE CHANGED
TO POSITION OF WATER STREAM
    x = 600
    y = 470

    frame_count = 0

    # Streaming loop
    while True:
        # Get frameset of depth
        frames = pipeline.wait_for_frames()

        # Get depth frame
        depth_frame = frames.get_depth_frame()
```

```
        # Get rgb frame
        rgb_frame = frames.get_color_frame()

        # Convert rgb_frame to numpy array to render image in opencv
        rgb_image = np.asanyarray(rgb_frame.get_data())

        # Colorize depth frame to jet colormap
        depth_color_frame = colorizer.colorize(depth_frame)

        # Convert depth_frame to numpy array to render image in opencv
        depth_color_image = np.asanyarray(depth_color_frame.get_data())

        #height and width of the depth image
        height, width = depth_color_image.shape[:2]

        #convert the rgb frame to lab
        rgb_lab = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(rgb_lab)
        l = cv2.add(l, -100)
        l = np.clip(l, 0, 255)
        clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
        cl = clahe.apply(l)
        limg = cv2.merge((cl,a,b))
        enhanced_rgb_image = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)

        #Read background image
        background = cv2.imread("intel6_frame.jpg")

        # enhance background image
        background_lab = cv2.cvtColor(background, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(background_lab)
        l = cv2.add(l, -100)
        l = np.clip(l, 0, 255)
        clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
        cl = clahe.apply(l)
        limg = cv2.merge((cl,a,b))
        enhanced_background_image = cv2.cvtColor(limg,
cv2.COLOR_LAB2BGR)
        background_gray = cv2.cvtColor(background, cv2.COLOR_BGR2GRAY)

        gray = cv2.cvtColor(enhanced_rgb_image, cv2.COLOR_BGR2GRAY)

        # adjust brightness of water stream
        gray = cv2.add(gray, 0)
        gray = np.clip(gray, 0, 255)

        # adjust brightness of background
        background_gray = cv2.add(background_gray, -30)
```

```
        bavkground_gray = np.clip(background_gray, 0, 255)

        # add blur
        difference_image = cv2.GaussianBlur(background_gray, (5,5), 0)

        # Compute the absolute difference between the current frame and
the background
        difference_image = cv2.absdiff(background_gray, gray)

        # count number of pixels with a depth value of 0.0
        array = depth_frame.get_data()
        no_zeros = height*width - np.count_nonzero(array)
        # Append zero counts to list
        zero_list.append(no_zeros)
        if zero_list[-1] > 4000+zero_list[-2]:
            water_stream = True
        if zero_list[-1]+6000 < zero_list[-2]:
            water_stream = False
        print(water_stream)

        # water stream enters frame at any point from the bottom``
        bottom_row = [depth_frame.get_distance(i, height-2) for i in
range(0+20, width-20)]

        # detects water stream
        if (0.0 in bottom_row) and water_stream == True:
            print('Detecting water stream')
            background_mean = np.mean(image_list[20:40], axis=0)
            cv2.imwrite("intel6_frame.jpg", background_mean)
            # find contours
            ret, thresh = cv2.threshold(difference_image, 60, 255,
cv2.THRESH_BINARY) #best value needs to be found. 60 does not detect all
of floor image
            contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
            # if the contour area is greater than 3000, draw a circle on
the top point and draw the contour of the water stream
            sorted_contours = sorted(contours, key=cv2.contourArea,
reverse=True)
            if cv2.contourArea(sorted_contours[0]) > 3000:
                contour = sorted_contours[0]
                y_coordinates = [point[0][1] for point in contour]
                top_point =
contour[y_coordinates.index(min(y_coordinates))][0]
                cv2.drawContours(rgb_image, [contour], -1, (0, 255, 0),
3)

                # Target coordinates - NEED TO BE DEFINED BY FIRE
-  this save moves the dot to the stream, hence stream is target
```

```
                target_x = top_point[0]
                target_y = top_point[1]

                # Initialize the PID controllers for x and y directions
                pid_x = PID(Kp=1.1, Ki=0, Kd=0.005, setpoint=target_x)
                pid_y = PID(Kp=1.1, Ki=0, Kd=0.005, setpoint=target_y)

                # Calculate the PID output for x and y directions
                pid_output_x = pid_x(x)
                pid_output_y = pid_y(y)

                # Update the coordinates of the red dot - THIS NEEDS TO
UPDATE THE WATER NOZZLE POSITION
                x += pid_output_x
                y += pid_output_y

                # Ensure that the red dot remains within the dimensions
of the image
                x = max(0, min(x, 639))
                y = max(0, min(y, 479))

                # Draw the red dot for the top point of the water stream
                cv2.circle(rgb_image, (int(x),int(y)), 1, (0, 0, 255),
5)
                print(top_point)
                # Draw the blue dot for the target point
                cv2.circle(rgb_image, (target_x, target_y), 1, (255, 0,
0), 5)

                if x == target_x and y == target_y:
                    consecutive_frames += 1
                else:
                    consecutive_frames = 0
        else:
            image_list.insert(0, enhanced_rgb_image)
            if len(image_list) > 50:
                image_list.pop()

        fgmask = fgbg.apply(enhanced_rgb_image)
        # rgb_image = cv2.addWeighted( rgb_image, 0.5, fgmask, 0.5, 0)

        # Render image in opencv window
        cv2.imshow("RGB stream", fgmask)
        cv2.imshow('difference image', difference_image)
        cv2.imshow("Depth Stream", depth_color_image)
        cv2.imshow("rgb image",rgb_image)
        key = cv2.waitKey(1)
        # if pressed escape exit program
```

```
        frame_count += 1

        # if frame_count == 80:
        #     cv2.imwrite("frame_80_fgmask.jpg", fgmask)

        # if frame_count == 81:
        #     cv2.imwrite("frame_81.jpg", rgb_image)

        # if frame_count == 82:
        #     cv2.imwrite("frame_82.jpg", rgb_image)

        # if frame_count == 83:
        #     cv2.imwrite("frame_83.jpg", rgb_image)

        # if frame_count == 84:
        #     cv2.imwrite("frame_84.jpg", rgb_image)



        if key == 27:
            cv2.destroyAllWindows()
            break

finally:
    pass
```

Appendix 4: Combined Code

```
#importing libraries
import cv2 #imports
import pyrealsense2 as prs2 #import pyrealsense2
import numpy as np
import math
import time,board,busio
import adafruit_mlx90640
from gpiozero import AngularServo, Device
from gpiozero.pins.pigpio import PiGPIOFactory
Device.pin_factory = PiGPIOFactory()
import os     #importing os library so as to communicate with the system
os.system ("sudo pigpiod") #Launching GPIO library
time.sleep(1) # As i said it is too impatient and so if this delay is
removed you will get an error
import pigpio #importing GPIO library
from simple_pid import PID


Pipeline=prs2.pipeline() #the pipeline's purpose is to oversee the
dataflow from the depth camera/bag file
Configuration=prs2.config() #produce configuration to allow the program
to work with the intel realsense depth camera
Configuration.enable_stream(prs2.stream.color, 640, 480,
prs2.format.bgr8, 15)
Configuration.enable_stream(prs2.stream.depth, 640, 480,
prs2.format.z16, 15)
Object=prs2.align(prs2.stream.color) #ensures the RGB and depth data is
synchronised
Pro=Pipeline.start(Configuration) #begins the pipeline

pitch = AngularServo(17, min_angle=-90, max_angle=90)
yaw = AngularServo(18, min_angle = -90, max_angle=90)
ESC=27  #Connect the ESC in this GPIO pin
pi = pigpio.pi();

def set_water_nozzle(water_nozzle):
    if water_nozzle:
        pi.set_servo_pulsewidth(ESC, 2000)
    else:
        pi.set_servo_pulsewidth(ESC, 0)

#thermal camera setup
i2c = busio.I2C(board.SCL, board.SDA, frequency=800000) # setup I2C
mlx = adafruit_mlx90640.MLX90640(i2c) # begin MLX90640 with I2C comm
print("MLX addr detected on I2C", [hex(i) for i in mlx.serial_number])
mlx.refresh_rate = adafruit_mlx90640.RefreshRate.REFRESH_4_HZ # set
refresh rate
```

```
mlx_shape = (24,32)
frame =  [0] * 768
frames=Pipeline.wait_for_frames()
color_frame = frames.get_color_frame()
cv2.imwrite("background.jpeg", np.asanyarray(color_frame.get_data()))
#reading the video file
#rgb_video =
cv2.VideoCapture("C:\\Users\\finta\\OneDrive\\Documents\\Year
3\\Python\\Fire Videos Bibal\\rgb_raw_T4.avi")
#thermal_video =
cv2.VideoCapture("C:\\Users\\finta\\OneDrive\\Documents\\Year
3\\Python\\Fire Videos Bibal\\thermal_raw_T4.avi")
#depth_video

# Get the video dimensions
#frame_width = int(rgb_video.get(cv2.CAP_PROP_FRAME_WIDTH))
#frame_height = int(rgb_video.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Make a list of images
image_list = []
depth_list = []
zero_list = [0,0]
water_nozzle = False
set_water_nozzle(water_nozzle)

#Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
output_video_result = cv2.VideoWriter('result.mp4', fourcc, 30.0, (640,
480))
output_video_thermal = cv2.VideoWriter('thermal.mp4', fourcc, 30.0,
(640, 480))
output_video_rgb = cv2.VideoWriter('rgb.mp4', fourcc, 30.0, (640, 480))

#looping through the video frames
while True:
    frames=Pipeline.wait_for_frames()
    color_frame = frames.get_color_frame()
    depth_frame = frames.get_depth_frame()
    rgb_frame = np.asanyarray(color_frame.get_data())

    mlx.getFrame(frame)
    framesFromThermal=(np.reshape(frame,mlx_shape)) #obtain each frame
from thermal source
    framesFromThermal=cv2.resize(framesFromThermal,(640,480)) #resize
the frame to 800x400
    thermal_cam=np.uint8(framesFromThermal)

    #ret, rgb_frame = rgb_video.read()
    #ret, thermal_frame = thermal_video.read()
```

```
    # Make a copy of the rgb frame to draw contours on
    result = rgb_frame.copy()
    rgb_frame_copy = rgb_frame.copy()

    ##FIRE DETECTION##
    #THERMAL#
    # Apply binary thresholding to find the fire
    _, thresh_fire = cv2.threshold(thermal_cam, 70, 255,
cv2.THRESH_BINARY)

    # Find contours of the thermal fire location
    _, thermo_contours, _ = cv2.findContours(thresh_fire,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Find the largest contour and only display that
    sorted_contours = sorted(thermo_contours, key=cv2.contourArea,
reverse=True)
    #fire_contour = sorted_contours[0]

    # Draw the contours on the thermal frame
    if len(sorted_contours) > 0 and cv2.contourArea(sorted_contours[0])
> 100:
        cv2.drawContours(thermal_cam, sorted_contours[0], -1, (0, 255,
0), 3)
        M = cv2.moments(sorted_contours[0])
        # Thermal location of fire
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv2.circle(thermal_cam, (cx,cy), 5, (0, 0, 255), -1)
    else:
        cx = 640
        cy = 480
    #THERMAL END#

    #RGB#
    # Define the range of colours to detect
    lower_fire = np.array([150, 200, 200])
    upper_fire = np.array([255, 255, 255])

    hsv = cv2.cvtColor(rgb_frame, cv2.COLOR_BGR2HSV)

    # Create a mask for the fire
    mask = cv2.inRange(hsv, lower_fire, upper_fire)

    #find the contour of the rgb fire location
    _, rgb_contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
```

```
    # Draw the contours on the rgb frame
    cv2.drawContours(rgb_frame, rgb_contours, -1, (0, 255, 0), 3)
    #RGB END#

    #SEE IF RGB and THERMAL agree#
    # if rgb_contours are within a 50 pixel radius of the
thermal_contours, show the thermal_contours
    # Calculates the centre of each rgb_contour
    for rgb_contour in rgb_contours:
        M = cv2.moments(rgb_contour)
        if M["m00"] != 0:
            tx = int(M['m10']/M['m00'])
            ty = int(M['m01']/M['m00'])
        else:
            # May cause an error
            tx = 0
            ty = 0

        distance = np.sqrt((cx - tx)**2 + (cy - ty)**2)

        if distance <= 50 and len(sorted_contours) >0:
            water_nozzle = True # turn water nozzle is on with arduino
pin, may need to initiliase as false
            set_water_nozzle(water_nozzle)
            cv2.drawContours(result, sorted_contours[0], -1, (0, 255,
0), 3)
            cv2.circle(result, (cx,cy), 5, (0, 0, 255), -1)
            fx = cx
            fy = cy
            dist_fire = depth_frame.get_distance(fx, fy)
            print(dist_fire)
            cv2.circle(result, (tx,ty), 5, (255, 0, 0), -1)
        else:
            water_nozzle = False
            set_water_nozzle(water_nozzle)
            break
    #END OF SEE IF RGB and THERMAL agree#
    #OUTPUTS fx, fy which is the fire location#
    ##END OF FIRE DETECTION##

    ##WATER STREAM DETECTION##
     #convert the rgb frame to lab
    rgb_lab = cv2.cvtColor(rgb_frame_copy, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(rgb_lab)
    l = cv2.add(l, -100)
    l = np.clip(l, 0, 255)
    clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl,a,b))
```

```
    enhanced_rgb_image = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    gray = cv2.cvtColor(enhanced_rgb_image, cv2.COLOR_BGR2GRAY)

    #Read background image
    background = cv2.imread("background.jpeg")

    # enhance background image
    background_lab = cv2.cvtColor(background, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(background_lab)
    l = cv2.add(l, -100)
    l = np.clip(l, 0, 255)
    clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8,8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl,a,b))
    enhanced_background_image = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    background_gray = cv2.cvtColor(background, cv2.COLOR_BGR2GRAY)

    # adjust brightness of background
    background_gray = cv2.add(background_gray, -30)
    background_gray = np.clip(background_gray, 0, 255)

    # Compute the absolute difference between the current frame and the
background
    difference_image = cv2.absdiff(background_gray, gray)

    ##WATER STREAM PICUTRE OUTPUT, EXTRACT WHITE BITS##

    # detects water stream if water nozzle is on, if so runs
    if water_nozzle == True:
        print('Detecting water stream')
        background_mean = np.mean(image_list[20:40], axis=0)
        cv2.imwrite("background.jpeg", background_mean)
        # find contours
        _, thresh = cv2.threshold(difference_image, 60, 255,
cv2.THRESH_BINARY) #best value needs to be found. 60 does not detect all
of floor image
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
        # if the contour area is greater than 3000, draw a circle on the
top point and draw the contour of the water stream
        water_sorted_contours = sorted(contours, key=cv2.contourArea,
reverse=True)
        if cv2.contourArea(water_sorted_contours[0]) > 3000:
            contour = water_sorted_contours[0]
            y_coordinates = [point[0][1] for point in contour]
            top_point =
contour[y_coordinates.index(min(y_coordinates))][0]
            cv2.drawContours(result, [contour], -1, (0, 255, 0), 3)
```

```
            ### NEEDS TO BE CALIBRATED ### DISTANCE MATTERS, CONSIDER
DEPTH CAMERA
            # Pixel to angle ratio yaw
            pixel_to_angle_ratio = 115/640
            const = -65

            # Pixel to angle ratio pitch
            #pixel_to_angle_ratio_pitch = 480/100

            # Target coordinates - NEED TO BE DEFINED BY FIRE
            target_x = fx
            target_y = fy

            # Initialize the PID controllers for x and y directions
            pid_x = PID(Kp=1.1, Ki=0, Kd=0.005, setpoint=target_x)

            # define x and y
            x = top_point[0]

            # Calculate the PID output for x and y directions
            pid_output_x = pid_x(x)

            # Update the coordinates of the red dot - THIS NEEDS TO
UPDATE THE WATER NOZZLE POSITION
            x += pid_output_x

            # Ensure that the red dot remains within the dimensions of
the image
            x = max(0, min(x, 639))

            # Convert the pixel coordinates to angles
            yaw_output_angle = const+(x*pixel_to_angle_ratio)
            pitch_output_angle = (dist_fire - 4.5933)/0.0403

            # # Send the angle value to the Arduino board as a PWM
signal
            # WRITE TO ARDUINO
            yaw.angle = (yaw_output_angle)
            pitch.angle = (pitch_output_angle)

            # Draw the red dot and the blue dot on the image
            cv2.circle(result, (int(x),top_point[1])), 1, (0, 0, 255),
5)
            print(top_point)

            # Draw the blue dot for the target point
            cv2.circle(result, (target_x, target_y), 1, (255, 0, 0), 5)

            # if x == target_x and y == target_y:
```

```
            #     consecutive_frames += 1
            # else:
            #     consecutive_frames = 0
    else:
        image_list.insert(0, enhanced_rgb_image)
        if len(image_list) > 50:
            image_list.pop()


    output_video_result.write(result)
    output_video_thermal.write(thermal_cam)
    output_video_rgb.write(rgb_frame)


    # cv2.imshow('Enhanced Thermal', enhanced_thermal_frame)
    cv2.imshow('Result', result)
    cv2.imshow('Thermal',thermal_cam)
    cv2.imshow('Rgb',rgb_frame)

    #output_video.write(result)

    #exiting with pressing 'q'
    if cv2.waitKey(24) & 0xFF == ord('q'):
        break

#releasing and destroying windows
#rgb_video.release()
output_video_result.release()
output_video_thermal.release()
output_video_rgb.release()
cv2.destroyAllWindows()
```